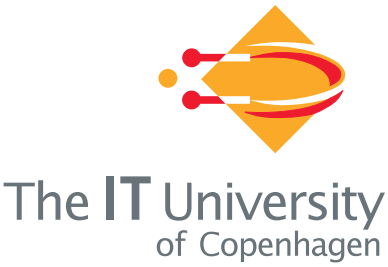


A Language for the Cell

Troels C. Damgaard
Vincent Danos
Jean Krivine

IT University Technical Report Series **TR-2008-116**

ISSN 1600–6100 **December 2008**

ISSN 1600–6100

ISBN 978877949908

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone: **+45 72 18 50 00**
Telefax: **+45 72 18 50 01**
Web **`www.itu.dk`**

A Language for the Cell

Troels C. Damgaard, Vincent Danos, and Jean Krivine

Abstract

We introduce a formal language, the $\mathcal{C}$-calculus, for modelling low-level interaction inside and among cells, the basic building blocks of all known life. We focus on two main actors of cells, proteins and membranes. Proteins are represented as clusters of *domains* sharing a common hidden name; domain-domain bonds are also represented via name-sharing. Compartments are formed by formal *membranes*. We treat also *channels* between membranes allowing transport of proteins, allowing us to capture an observable intermediate state in cell fusion or division, regulated by diffusion.

We illustrate the calculus by giving two example models. We exemplify the basic constituents of the calculus, by developing a model of simple cross-membrane signalling via a G-protein coupled receptor protein. We continue by developing a model illustrating part of the endocytic pathway—the formation of clathrin-coated cytoplasmic vesicles, through budding from the plasma membrane (the cell-wall).

1 Introduction

In this paper, we introduce a formal language, the $\mathcal{C}$-calculus, for modelling low-level interaction inside and among cells, the basic building blocks of all known life. We start by giving a brief overview of the biology of cells, and discuss the level of abstraction that we aim to capture in the calculus. In the $\mathcal{C}$-calculus, we focus on two main actors of cells, membranes and proteins.

Membranes In nature, biological membranes are vital for separating cells from their surroundings, for confining dangerous compounds to secure compartments, for serving as vehicles of transport, or, for promoting certain reactions by raising the local concentration of reactants. The membrane of a cell consists mostly of a double layer of proteins and lipids (fat-like molecules). Membranes may have a variety of molecules attached to them or embedded within them. Notably, such molecules may be *transmembrane* proteins, which serve to propagate signals across the membrane barrier; or, they may be various molecules, which serve as pumps for moving different smaller molecules in and out of the cell. The molecules attached to membranes also serve to differentiate the two sides of a membrane and to stabilize it. Organelles ("small organs") inside eu-

karyotic cells (cells with nuclei) are also wrapped in protective membranes, as are *vesicles* transporting material between cells.

Biological membranes are dynamic entities—under certain conditions, they may form *buds*, new small membrane-enclosed compartments that grow from a particular location on the membrane. Budding may serve to create vesicles, for instance. A membrane may also undergo *fission* and divide entirely in two, in the process also dividing the content in the enclosed compartment.[1] Both budding and cell division are typically complicated and regulated processes involving various kinds of machinery (depending on the type of membranes and division). For instance—in what is sometimes known as asexual reproduction of eukaryotic cells—when two eukaryotic cells divide, initially the chromosomes (the structures carrying DNA) in the cell nucleus are carefully separated into two nuclei. This process is called *mitosis*. In a following step called *cytokinesis*, the cytoplasm (the contents inside the cellular membrane) as well as the organelles and the cellular membrane are split up to form two new daughter cells.

Membranes may also *fuse* with other membranes. This occurs, for instance, when viruses infect cells, and it serves to create such structures as the long fibers of human skeletal muscles. When two cells fuse together, their membranes meet at one point and create a connection between the membranes. Eventually, the two membranes will form one single, continuous membrane that surrounds the contents of both cells.

Research has confirmed that both when cells fuse or divide, membranes will temporarily be in a partially fused state with a *neck* consisting of partially fused membrane material.[2] This neck acts as a bridge connecting two compartments and it typically allows some material to be exchanged by diffusion. Even further, it also happens that cells abort fusion (or fission). For instance, two cells may partially fuse, exchange some material by diffusion, and then part again.

In the $\mathcal{C}$-calculus, compartments are formed by formal membranes. To capture faithfully the dynamics of fusion and fission, we include in the $\mathcal{C}$-calculus also binary *channels* between membranes. Channels are, to our knowledge, a novel abstraction that allows us to capture an observable intermediate state in fusion and fission. In this intermediate state the connected compartments may exchange material regulated by diffusion. Formally, channels are formed by *gates* sharing a common name; gates can be seen as abstractions for the various molecular-structures that constitute the neck.

[1] As in computer science, biological terminology tends to be overloaded. In this paper, we shall use *fission* as a generic term to refer to any process by which membranes divide. In doing so, we stretch the terminology typically employed by biochemists a bit, but it is convenient to use the term to describe the reverse of fusion. We stress that our usage of fission should not be confused with so-called *binary fission*—the specific process by which prokaryotic cells (cells without a nuclei) divide.

[2] For instance, studies on cellular division in nematode worms show that an intermediate state may indeed be observed during cell fusion [Pod06]. Other studies have focused on different models for the intermediate structures in membrane fission [KK03]. Some studies also hypothesize that the shapes of the intermediate protein structures, which form the neck itself, may drive and serve to distinguish the process of fission and fusion [KC02].

Membranes-enclosed cells are large structures compared to the typical protein. While a eukaryotic cell may be around $10 - 100\mu$m, the typical diameter of a globular protein—like hemoglobin, for instance—is around 5nm. The difference in scale of mass is far larger. Thus, from the perspective of cells and membranes, proteins can be considered essentially atomic.

Proteins Proteins are linear chains of amino-acids. The string of amino-acids is known as the *backbone* of a protein. Gene-sequences in the DNA enclosed in all cells determines the amino-acid sequence for all the proteins that a cell can produce. Production (synthesis) of proteins involves *transcribing* the DNA to mRNA, various kinds of *post-transcriptional* modification, and *translating* the mRNA to proteins by way of small cellular factories called *ribosomes*. Finally, most proteins fold into three-dimensional structures called the *conformation* or the *tertiary* structure of the protein. The tertiary structure may in turn hide or reveal some parts of the protein.[3] The entire process of synthesis of proteins is complex, self-organized and, as nearly all biological phenomena, highly regulated by various molecular signals.

The peptide bonds forming the backbone make proteins relatively stable structures. Proteins may also form weaker bonds with other proteins to form multi-protein *complexes*. These bonds form on particular parts of the proteins referred to as *domains* or *sites*.

In the $\mathcal{C}$-calculus, our aim is to capture a domain-level description suitable for describing protein-protein interaction in a multi-compartment environment. To allow for transmembrane-proteins, in the $\mathcal{C}$-calculus proteins are represented as clusters of domains sharing a common name. This name is a formal representative of the backbone. Domain-domain bonds are also represented via name-sharing.

Formal foundation and sources of inspiration Formally, the calculus is founded on bigraphical reactive systems (due to Milner et al. [JM04, Mil06]) through the $\mathcal{B}^{\Sigma,\mathcal{R}}$-framework treated by Damgaard and Krivine [DK08] (see below). Thus, the $\mathcal{C}$-calculus admits a formally founded graphical language; and we inherit results for a tacit term syntax with well-understood structural congruence properties [Mil05, DB06], and a well-understood behavioral theory. Even further, due to recent results by Milner, Krivine, and Troina [KMT08], a stochastic extension of the nondeterministic calculus presented in this paper, is well-founded. Implementation of bigraphs has also been investigated [BDGM07], and a prototype implementation is available [BPL07].

The $\mathcal{B}^{\Sigma,\mathcal{R}}$-framework (presented in [DK08]) paves the way for our usage of bigraphical calculi for modelling biological interaction, in particular, by extending bigraphical reaction rules to allow testing of contextual negative side-conditions.

[3] Traditionally, biochemists refer to four distinct aspects of structure for proteins. *Primary* structure, the amino-acid sequence; *secondary* structure, describing regular local structures (such as so-called *alpha-helixes* or *beta-sheets*); *tertiary* structure, the 3-d structure; and, *quaternary* structure, the structure of *complexes* formed from several proteins.

It also introduces a self-contained and generic presentation of the operational semantics for a subset of bigraphical calculi, in a structural style more in line with how process calculi are typically presented. We shall make good use of this groundwork in this paper, eliding most discussion of bigraphical idiosyncracies, and focusing on the presentation and motivation of the calculus itself.

We are inspired by and combine features from several existing languages, in particular, the κ calculus [DL04], a rule-based language capturing domain-level protein-protein interaction. We base our domain-level model of proteins closely on our experience with capturing the κ-calculus in the $\mathcal{B}^{\Sigma,\mathcal{R}}$-framework (in [DK08]), only here generalized to a multi-compartment setting. A main aim of the work presented in this paper, has been to extend a κ-like language with dynamic compartments.

The model for membranes and their dynamics is influenced, in particular, by the brane-family of calculi [Car04b, DP04], which deals solely with membrane interaction; and the Bioambients [RPS$^+$04], one of the first calculi to investigate at the same time both molecular and membrane-level interaction.

We discuss in more detail the inspiration sources for the $\mathcal{C}$-calculus both in the main sections of the paper, and in Section 5.2.

An overview of the paper Having given an overview of the biology of cells and introduced the background that the $\mathcal{C}$-calculus builds on, in the next section we present the $\mathcal{C}$-calculus static model. In Sections 2.1 and 2.2, we introduce and motivate the model of proteins and membranes, and in Section 2.3 we define the grammar for $\mathcal{C}$-calculus *solutions* and define an associated structural congruence relation. In Section 2.4, we discuss how structural congruence classes of solutions correspond to bigraphs. This paves the way for a graph-based description of *well-formed* solutions, which we define and motivate in Section 2.5. We shall define well-formedness to ensure that well-formed solutions are those that we can interpret biologically.

In Section 3, we turn to the operational semantics of the calculus. The κ-calculus has been one of several advocates for what we may call *rule-based* modelling. Rules can be understood and manipulated separately, are easily visualized, and, easily express possible overlapping behaviors of certain configurations. On the other hand, we need to uphold certain well-formedness invariants to ensure that we can interpret model configurations sensibly as biological states. Hence, we advocate a middle-ground for the $\mathcal{C}$-calculus: *Modelling by rule refinement*. We shall give a set of rule-generators that each express a core biological action for membranes and proteins. The key idea is that a domain expert, say a bio-chemist, may pick and refine those core generators to give a domain-specific sub-calculus for the study of a particular biological application. He does this by giving a set of application conditions that express when an action may be applied.

We start by defining $\mathcal{C}$-calculus reactive systems and reaction rules in Section 3.1. In Section 3.2 we describe formally so-called *projective* descriptions of rules, inspired by the Projective Brane Calculus by Pradelier and Danos [DP04],

and, more recently, by the *patch* reactions treated by Cardelli for Bitonal Systems [Car08]. They let us describe conveniently sets of rules for reactions involving regions separated by one or two membrane-surfaces, while eliding the orientation of those separating membranes. In Sections 3.3 and 3.4, we introduce rules for protein-protein interaction and membrane (or channel) reconfiguration, respectively. In Section 3.5, we define refinements of rules and settle on a definition of those rules that we allow in the $\mathcal{C}$-calculus. Finally, in Section 3.6 we show that the allowable rules preserve well-formedness.

In Section 4, we illustrate the calculus with two examples. We exemplify the basic constituents of the calculus, by developing a model of simple cross-membrane signalling via a G-protein coupled receptor protein. We continue by developing a model illustrating part of the endocytic pathway—the formation of clathrin-coated cytoplasmic vesicles, through budding from the plasma membrane (the cell-wall).

In Section 5, we conclude and discuss some related and future work.

In Appendix 6, we include a series of definitions for solutions inherited directly from the $\mathcal{B}^{\Sigma,\mathcal{R}}$-framework for such properties as free names, normal forms, and substitution.

2 The $\mathcal{C}$-calculus—a Model of Proteins and Membranes

In Sections 2.1 and 2.2, we introduce and motivate the foundation of the $\mathcal{C}$-calculus—the model of proteins and membranes. In Section 2.3, we define formally the grammar for the $\mathcal{C}$-calculus and its associated structural congruence relation. In Section 2.4, we discuss how solutions up to structural congruence correspond to bigraphs, and, finally, in Section 2.5 we define and motivate well-formedness for the $\mathcal{C}$-calculus.

2.1 Proteins

In the $\mathcal{C}$-calculus, proteins are represented as groups of interaction *domains* sharing a common name, which we call the *backbone*. Domain-domain bonds, which let proteins form complexes, are also represented via name-sharing. While in the κ-calculus proteins are atomic entities, domains are the atoms of the $\mathcal{C}$-calculus. Domains are represented via atomic function symbols, and thus depicted as atomic nodes (i.e., they cannot contain other nodes).

Formally, we derive the set of domains from a protein-signature. Protein-names are drawn from a fixed set $\mathcal{P}$, which we require to be disjoint from all other kinds of names.

Definition 2.1 (protein-signature)**.** A protein-signature, $\Sigma^{\mathcal{P}}$, is a map from $\mathcal{P}$ to $\mathbb{N}^2$.

The protein-signature, $\Sigma^{\mathcal{P}}$, maps each protein-name p to a pair of integers (a, r), where a is the *arity* of the protein—it determines the number of domains

of p—and, r is the *receptor-arity* of p. We explain the receptor-arity below.

Definition 2.2 (domains)**.** Given a protein-signature $\Sigma^{\mathcal{P}}$, let $\mathcal{D} \subseteq \mathcal{P} \times \mathbb{N}$, the set of domains induced by $\Sigma^{\mathcal{P}}$, be given as

$$\mathcal{D} = \{(\mathsf{p}, i) \mid \mathsf{p} \in \mathcal{P},\ \Sigma^{\mathcal{P}}(\mathsf{p}) = (a, r),\ \text{and } 0 \leq i < a\}.$$

We choose a simple concrete language for expressing the domains of proteins. Suppose the protein-name pro has the signature $(4, 1)$. The arity 4 determines that there are four domains in a protein of type pro. We write its four domains as $\mathsf{pro0}, \mathsf{pro1}, \mathsf{pro2}, \mathsf{pro3}$. We interpret the receptor-arity for a protein-name in the following manner: If the receptor-arity r is larger than zero, pro is a transmembrane protein, and all well-formed instances of pro must take the shape of two subunits separated by a single membrane: one with domains 0 to $r-1$, and one with the domains numbered r to $n-1$.

Protein domains have one or two ports for connecting to named *links*. All domains of a protein share a common *backbone*-link connected to their first port, and pairs of bound domains may share a *complexation*-link on a second port. A domain has an associated complexation-state, which determines their current ability to form complexes. It may be *bound* to a link x; it may be in a *hidden* state due to the overall conformation of the protein it is part of; or it may be currently unbound, but *visible* and ready to form new complexation-links.[4]

We write $\mathsf{pro}i_a^x$ for the ith domain of a pro protein connected to the backbone a and bound to the complexation-link x; we write $\mathsf{pro}i_a$ for the domain in its visible, but unbound state; and, we write $\overline{\mathsf{pro}i}_a$ for the domain in its hidden state. Figure 1 shows how we illustrate these three kinds of domains. Figure 2 contains an illustration of an instance of the entire protein pro, which respects the protein-signature. Figure 2) contains an illustration of a well-formed instance of a closed protein pro. (We formally introduce the name-hiding operator (x) and the syntax for membranes in the following sections.) [5]

[4] We have some freedom in choosing exactly how to model as state. In particular, there is a subtle difference in electing to model unbound, *visible* domains (i) via a separate state as inspired by the κ-calculus, or (ii) via closed unary links. Model (i) allows us to use less names, while model (ii) allows us to write a reconfiguration rule, which matches both bound and unbound domains. In this paper, we have chosen model (i), thus building on the intuition from the κ-calculus.

[5] Our model and naming scheme for proteins and their domains is essentially a minimal extension to a multi-locality setting of the $\mathcal{B}^{\Sigma,\mathcal{R}}$-model of κ-calculus proteins described in [DK08]. Centrally, the atom representing the protein backbone has been substituted for a shared link, allowing us to conveniently model transmembrane proteins.

Though the protein-model and naming scheme is simplistic, it is sufficient for describing many interesting examples (as we shall see later in Section 4). However, one may easily extend the protein model with more complex kinds of state, as done for later versions of the κ calculus [DFF$^+$07, DFFK07], or with more complex naming schemes, for instance, for proteins or domains. For instance, in later versions of the κ-calculus (see *loc. cit.*), the capability to write such rules is added by introducing a small subtyping lattice for states and incorporating subtyping into the matching relation.

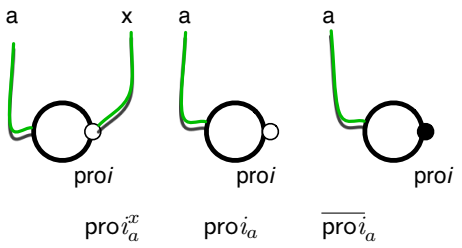
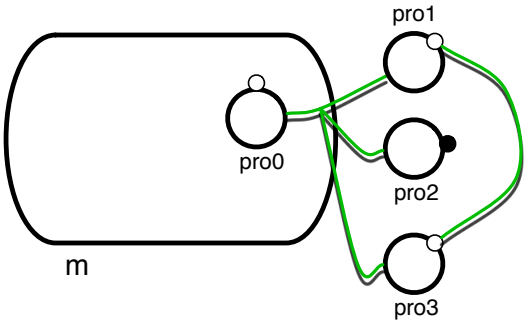


Figure 1: Domain states.



$(a, x)\, \mathsf{m}[\mathsf{pro0}_a] \mid \mathsf{pro1}_a^x \mid \overline{\mathsf{pro2}}_a \mid \mathsf{pro3}_a^x$

Figure 2: A well-formed instance of pro.

2.2 Adding Dynamic Compartments

To model membranes and their dynamics in the $\mathcal{C}$-calculus, we add two kinds of function symbols to the language. In keeping with the tradition set out by Cardelli et. al. for ambient and brane-calculi, we represent formal membranes by square brackets $[\cdots]$. The syntactic space between the brackets induces a region in which other entities, including other membranes, may float. As for proteins, it is convenient to allow different kinds of membranes to be distinguished by names, allowing us to distinguish, for instance, a vesicle from the plasma membrane encapsulating a cell. For example, we may write an expression to model that a plasma membrane contains an empty vesicle: $\mathsf{plasma}[\mathsf{ves}[]]$. We suppose that such membrane-names are drawn from another fixed pool of names, $\mathcal{M}$, disjoint from every other kind of names.

We add also *gates* to the language. They are the endpoints of binary *channels* between membranes. Gates, and the channels they form, are an abstraction of the molecular structure forming bonds between fusing or dividing membranes. We shall discuss the channel abstraction in more detail in a separate section below. Gates have one port to connect gates pairwise; for instance, the expression $(x)\, \mathsf{ves}[\triangle_x] \mid \mathsf{ves}[\triangle_x]$ denotes a channel formed between sibling vesicles.

While our gates, $\triangle_g$, are anonymous, one could certainly consider also al-

lowing gates to be named, as domains and membranes. This would allow us to distinguish different kinds of channels, potentially convenient for some models. In this paper, however, we shall focus on introducing the channel abstraction itself, and for that purpose, anonymous gates suffice.

2.2.1 On the Channel Abstraction

Compared to existing calculi treating membranes [Car04b, RPS+04, PQ05, LT06, Car08], channels allow us to capture an observable semi-fused intermediate state in cell fusion or fission. In this state, the membranes are still separate, but may exchange material regulated by diffusion. To capture partial fusion or fission as described in the introduction, we need to keep the material in each compartment separate.

In Figure 3, we illustrate the two most basic configurations of semi-fused membranes, and show how they are represented in the calculus with the help of channels. On the left, the surfaces of two co-located membranes are partially fused; on the bottom-left we show the $\mathcal{C}$-calculus representation of that situation. They may be cells in the process of fusing or dividing; or it may be a cell in the process of budding a small vesicle. On the right, we have a similar situation, only with one membrane inside the other. In both cases, the partial fusing of the membranes allows diffusion of material across the point where the membrane surfaces touch. On the left, the channel allows diffusion between the two compartments inside the membranes. On the right, the channel allows transport between the surroundings and the innermost compartment.

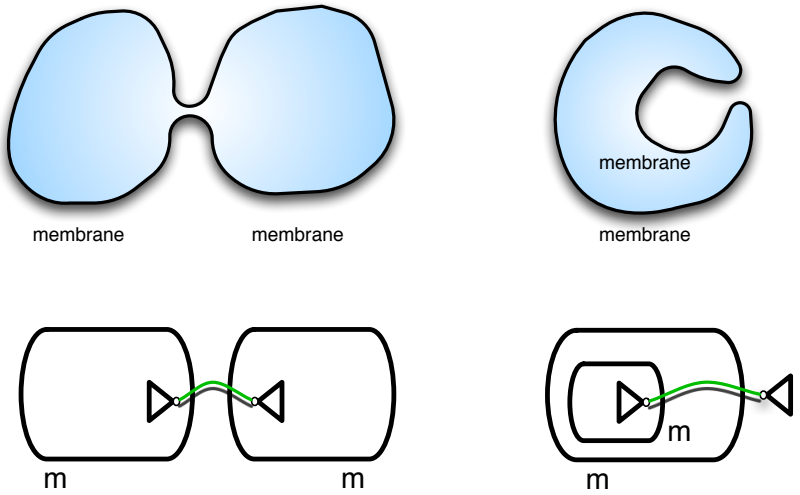


Figure 3: Fused membranes, and their representation via channels in the $\mathcal{C}$-calculus.

In the topmost part of Figure 4, we illustrate the three different stages in cell

fusion and fission that we wish to capture. The arrows indicate, that while the process of fusing two membranes is deterministic (resulting in a single membrane containing all the entities of both compartments), the division of the entities inside a single compartment is a nondeterministic process.

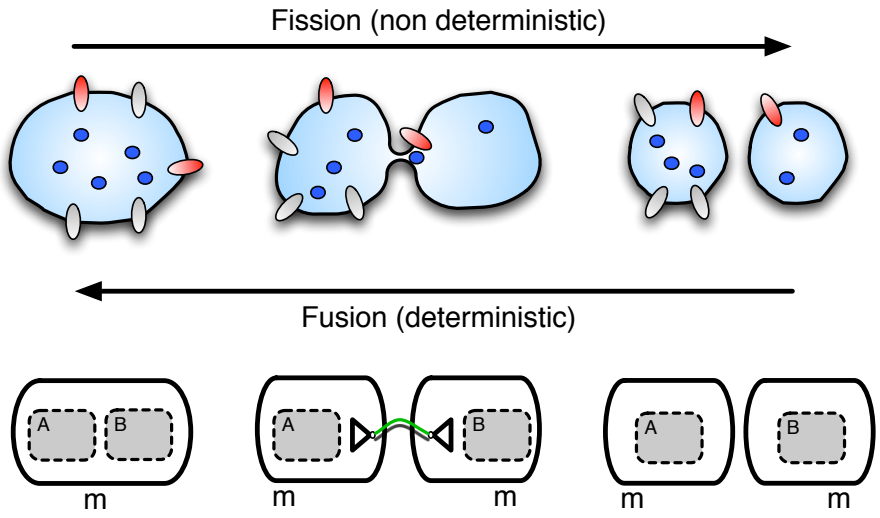


Figure 4: Representing an observable intermediate state—for cell fusion and division.

In nature, various kinds of cell division may be identified. For instance, due to their different structure, the division processes are fundamentally different between eukaryotic cells and prokaryotic cells. Common to them all are that they are quite complicated and highly regulated processes—in particular, for determining the division of contained entities. By adding an intermediate state, we allow users of the $\mathcal{C}$-calculus to model such regulatory behavior during the process of division, in more detail. In particular, it allows rules for diffusion to regulate the final division of entities in the newly formed compartments.

At the bottom of Figure 4, we illustrate the three different stages as they are captured in the $\mathcal{C}$-calculus. We focus on membranes and gates, and use variables to parameterize the model over the contents of the membranes. We shall return to these stages, when we discuss the dynamics of the $\mathcal{C}$-calculus in Section 3.

As a final remark, we may also note that while the main motivation for channels is to represent an intermediate step in division regulated by (nondeterministic) diffusion, one might also envision using channels to model more stable kinds of transport channels, such as so-called *tunneling nano-tubes* (TNTs) [RSM$^+$04, GBG08]. TNTs are long and stable binary channels between co-located membranes that were recently discovered and bear a striking resemblance to our

binary gated channels.[6]

2.3 Signature and Syntax

In the previous section, we have informally introduced proteins, membranes, and gates and their syntax. In this section, we formally define the signature and grammar of the $\mathcal{C}$-calculus.

2.3.1 Signature

The signature of the $\mathcal{C}$-calculus defines the classes of function symbols in $\mathcal{C}$-calculus, and their type and number of ports for linkage.

Definition 2.3 ($\mathcal{C}$-calculus signature)**.** Given a protein-signature $\Sigma^{\mathcal{P}}$ and a set of membrane-names $\mathcal{M}$, the signature for the $\mathcal{C}$-calculus is

$$\begin{array}{rcrcl} \Sigma & = & \mathcal{D} \times \{\mathrm{v}, \mathrm{h}\} & : & \mathrm{atomic}(1) \\ & \uplus & \mathcal{D} \times \{\mathrm{b}\} & : & \mathrm{atomic}(2) \\ & \uplus & \mathcal{M} & : & \mathrm{active}(0) \\ & \uplus & \{\triangle\} & : & \mathrm{atomic}(1) \end{array}$$

where $B : \mathit{kind}(n)$ is short for stating that the elements of B are assigned the type *kind* and n ports.

The signature formally records what we have stated informally above. Function symbols from the set of domains, $\mathcal{D}$ (as defined in Definition 2.2), are atomic and exists in three states, visible (v), hidden (h), or bound (b). They all have one port (for linking to a protein backbone) and in their bound state an extra port (linked to a complexation-link). The function symbols used for membranes have no ports, but are active—i.e., they may nest other solutions inside them—and gates are atomic and have a single port (linked to another gate).

2.3.2 Grammar

We call the basic computing units in the $\mathcal{C}$-calculus *solutions*, and we shall define their grammar below. We use a small set of well-known operators from process calculi equipped with pure names, in particular, we take an operator for hiding names, (x), and parallel product, $|$. We shall also quotient the grammar over a structural congruence relation ensuring, for instance, that bound names are alpha-convertible and that parallel product is associative and commutative as usual. In formally defining solutions, we presuppose an unbounded supply of pure names (disjoint from all other kinds of names); we use lowercase italic letters $a, b, c, \ldots$ for pure names.

For defining parametric reaction rules in the following section, we shall also need solutions with variables (sometimes called holes), and groups—ordered sequences of solutions that may share names. Variables in the $\mathcal{C}$-calculus are numbered, i.e., they are drawn from a countable set of variables $\mathbb{V} = \{\Box_0, \Box_1, \ldots\}$.

[6] We discuss TNTs in more detail under future work (see Section 5.1.)

$x, y, z, \ldots$	pure names
$A, B, C, \ldots$	variables
d, e	generic domains
m, n	generic membranes

Figure 5: Notational conventions—names, variables, domains, and membranes

Formally, we require for any solution, that all variables in a solution are distinct (in the following, we assume all solutions and groups to respect this requirement). We let metavariables $A, B, C, \ldots, X, Y, Z, \ldots$ range over variables. In practice, we shall not be concerned with the actual numbers used for variables, only their order (i.e., relatively to the other variables) in a solution. We record this in the structural congruence relation below.

Formally, the $\mathcal{C}$-calculus is a $\mathcal{B}^{\Sigma,\mathcal{R}}$-calculus [DK08]. This means that the following grammar, definitions for structural congruence, substitution, etc. for the $\mathcal{C}$-calculus is derived directly from the $\mathcal{C}$-calculus signature, and the generic grammar given for $\mathcal{B}^{\Sigma,\mathcal{R}}$-processes. In this paper, we shall elide the full formal treatment of certain definitions derived directly from the $\mathcal{B}^{\Sigma,\mathcal{R}}$-framework, such as variables and their ordering, substitution, definitions of free and bound names, and certain normal forms. In this section, we limit ourselves to informal introductions. For a formal treatment, we refer the reader to the paper introducing $\mathcal{B}^{\Sigma,\mathcal{R}}$-calculi [DK08]; for easy reference, we have included in Appendix 6 the central definitions inherited from $\mathcal{B}^{\Sigma,\mathcal{R}}$.

In the remainder of the paper we shall overload the following names to be metavariables for domain and membrane-names: d and e are generic domain-names, m and n are generic membrane-names. Note, that this also means that later on, when we give rules using these names, formally these rules should be read as schemas for rules using concrete names. We summarize these metavariable conventions in Figure 5.

In Definition 2.4, we define formally solutions, and in Definition 2.5, we define solution groups. A solution is a special case of a group—namely, a group with exactly one solution. We use this in definitions and properties, and overload the symbols we use for operations such as structural congruence and substitution to work for both solutions and groups.

Definition 2.4 (solutions)**.** Solutions are collections of protein-domains, and channel-gates nested inside compartments induced by membranes. A solution may either by empty; a combination of two colocated solutions; a solution with a hidden name x; a domain $\mathsf{d} \in \mathcal{D}$ in one of three possible states linked to backbone b and (when bound) to complexation-link x; a gate $\triangle$ linked to a channel-link g; a membrane $\mathsf{m} \in \mathcal{M}$; or, a variable X.

We define solutions inductively as follows

S, T	::=	0	the empty solution
	|	$\mathsf{S} \mid \mathsf{T}$	parallel product
	|	$(x)\,\mathsf{S}$	new name
	|	D	a domain
	|	$\triangle_g$	a channel-gate
	|	$\mathsf{m}[\mathsf{S}]$	a membrane
	|	X	variable

where

D	::=	d_b^x	a bound domain
	|	d_b	a visible and unbound domain
	|	$\overline{\mathsf{d}}_b$	a hidden domain

Definition 2.5 (groups)**.** Groups are defined inductively as

G, F	::=	S	a single solution
	|	$(x)\,\mathsf{G}$	new name
	|	$\mathsf{G} \,\|\|\, \mathsf{F}$	wide parallel product
	|	ϵ	the empty group

Call solutions or groups without variables *ground*. Call nonground solutions *presolutions*, and nonground groups *pregroups*—in general, we call nonground solutions or groups, *contexts*. We write $\mathcal{S}^{\Sigma}$ for the solutions over the signature Σ.

We use parentheses for grouping as usual. We let $\mid$ bind tighter than $||$, which in turn binds tighter than the operator (x). When $\tilde{x} = \{x_1, \cdots, x_n\}$ we write $(x_1 \cdots x_n)\,\mathsf{S}$ or $(\tilde{x})\,\mathsf{S}$ to mean $(x_1) \cdots (x_n)\,\mathsf{S}$. Finally, as usual, we shall typically elide the 0 in empty membranes, for instance, writing $\mathsf{ves}[]$ instead of $\mathsf{ves}[0]$.

As usual, the new name operator $(x)\,\mathsf{S}$ is a binder on pure names, that is, instances of the pure name x in S are α-convertible. We define inductively the set of free, $\mathrm{fn}(G)$, or bound (pure) names, $\mathrm{bn}(G)$, of an expression as usual (cf. Definition 6.3 in Appendix 6). (Recall that names for proteins, and membranes are distinct and different from the pure names). We call a solution or group *closed* if all its pure names are bound; and *open* if it is not closed. Generally, we shall refer to pure names as just names, qualifying protein-names and membrane-names. We say that two solutions S and T are *connected* if they share free names, that is, if $\mathrm{fn}(\mathsf{S}) \cap \mathrm{fn}(\mathsf{T}) \neq \emptyset$.

Structural congruence We quotient solutions and groups according to a structural congruence relation, $\equiv$, enforcing prominently that names in the scope of a binder are α-convertible and that (x) floats freely in a term, as long as we do not capture free instances of the name x. We also allow order-preserving reordering of variables, for instance, that $\Box_0 \mid \Box_1 \equiv \Box_0 \mid \Box_2$. This formalizes the notion that we are only concerned with ordering of variables internal to an expression (cf. Definition 6.1 in Appendix 6). Furthermore, we make parallel

product associative, allow reordering, and stipulate that we may introduce (or delete) empty solutions. For groups, we make wide parallel product associative and make ϵ the neutral element.

Definition 2.6 (structural congruence)**.** Structural congruence, $\equiv$, on solutions and groups is the least congruence relation containing α-equivalence (i.e., bijective renaming of bound names), order-preserving renumbering of variables, and s.t.:

- parallel product, $|$, is associative and commutative with 0 as neutral element;
- wide parallel product, $||$, is associative with ϵ as neutral element;

and including the following *scope extrusion* laws

$$
\begin{array}{rcll}
\mathsf{m}[(x)\,\mathsf{S}] & \equiv & (x)\,\mathsf{m}[\mathsf{S}] & \text{(extrusion - mem)} \\
\mathsf{S} \mid (x)\,\mathsf{T} & \equiv & (x)\,\mathsf{S} \mid \mathsf{T} \quad \text{if } x \notin \mathrm{fn}(\mathsf{S}) & \text{(extrusion - par)} \\
((x)\,\mathsf{S}) \mid\mid \mathsf{T} & \equiv & (x)\,\mathsf{S} \mid\mid \mathsf{T} \quad \text{if } x \notin \mathrm{fn}(\mathsf{T}) & \text{(extrusion - wide par left)} \\
\mathsf{S} \mid\mid (x)\,\mathsf{T} & \equiv & (x)\,\mathsf{S} \mid\mid \mathsf{T} \quad \text{if } x \notin \mathrm{fn}(\mathsf{S}) & \text{(extrusion - wide par right)} \\
(x)(y)\,\mathsf{S} & \equiv & (y)(x)\,\mathsf{S} & \text{(reordering)} \\
(x)\,\mathsf{S} & \equiv & \mathsf{S} \quad \text{if } x \notin \mathrm{fn}(\mathsf{S}) & \text{(elision)}
\end{array}
$$

As usual, it is easy to check that free names are invariant under structural congruence, that is, for solutions $\mathsf{S} \equiv \mathsf{T}$, $\mathrm{fn}(\mathsf{S}) = \mathrm{fn}(\mathsf{T})$.

Substitution Variables in a solution S may be substituted for a group of solutions G. In the $\mathcal{C}$-calculus, we shall find use for both total substitution $\mathsf{S} \cdot \mathsf{G}$, and $\mathsf{S} \lhd \mathsf{G}$. The treatment of (variables and) substitution is transferred directly from the $\mathcal{B}^{\Sigma,\mathcal{R}}$-framework, so we introduce substitution only informally here (for easy reference, the definitions are included in Appendix 6, cf. Definitions 6.7 and 6.9).

We may, for instance, substitute three variables in a solution S by a group G of three solutions. Essentially, the resultant solution is computed by exchanging the ith ordered variable in S by the ith solution in G. If S has exactly three variables, the substitution is *total*, and we write $\mathsf{S} \cdot \mathsf{G}$ for the result. If S has more than three variables, the substitution is *partial*, and we write $\mathsf{S} \lhd \mathsf{G}$ for the result; remaining variables in S are reordered to have higher numbers than any variable in G. We distinguish the two kinds of substitution because total substitutions are associative while partial substitutions are not.

Notably, in parametric reaction rules, variables play the role of placeholders serving only to carry parameters across a reaction. For this usage, neither the numbers nor the ordering of variables matters (as we shall close the reaction relation under substitution for arbitrarily ordered groups of solutions). We require only that the same variables be used on both sides of a reaction rule. Therefore in reaction rules, we overload the usage of the metavariables $A, B, C, \ldots$ to allow them to be used instead of concrete variables (thus eliding the numbers of each variable).

2.4 Solutions as Graphs

For programming in the $\mathcal{C}$-calculus, and for giving and checking many properties, it is convenient to have a concise and formal term-base language as we have defined in the previous section. For visualization and for expressing certain properties and conditions it is convenient also to consider solutions as certain kinds of graphs.

Up until now, we have used a number of illustrations to illustrate solutions. Since the $\mathcal{C}$-calculus is formally founded on bigraphs, through the $\mathcal{B}^{\Sigma,\mathcal{R}}$-framework, these illustrations are actually based on a fully formal graphical language for solutions and groups, both ground and nonground. Formally, structural equivalence classes of solutions and groups correspond to certain bigraphs over a corresponding bigraph signature; hence, we may directly use the graphical language for bigraphs to describe $\mathcal{C}$-calculus-solutions. This is formally stated in [DK08, Proposition 5.4]; we shall not go into detail with that here.

For expressing and checking well-formedness conditions, it is however convenient to be able to describe and reason about solutions from a graph-based understanding of solutions. Hence, below we describe informally how solutions correspond to certain *bi*graphs—the combination of a tree and a hyper-graph sharing nodes.

First of all, every function symbol in the signature corresponds to the kind (or *control*) for a node in a bigraph. Eliding names, we may view a solution as a tree—sometimes called the *place* graph—with unordered children (parallel product is commutative) induced by the containment relation for membranes, whose leaves are domains, gates, variables, or empty solutions. In other words, the tree is simply the abstract syntax tree of the expression for a solution, only with unordered children to model directly the properties of the parallel product. Groups of solutions correspond to place graphs that are forests, instead.

Name-sharing between domains and gates creates a separate graph—sometimes called the *link* graph—of (hyper-)links across the tree. In closed solutions every link is an unnamed edge, while in open solutions, the links that correspond to free names are named.

When we depict bigraphs, instead of drawing the two graphs of bigraphs separately, we draw them in one picture, indicating the place graph with containment—just as we have done in the illustrations of $\mathcal{C}$-calculus solutions up until now (e.g., in Figure 2).

Finally, it may be convenient to point out, that graphically, we may think of substitution as inserting subtrees in the holes made by variables. Reactions, which we shall consider later, induce changes in the graph by creating or deleting nodes, by creating or severing links, or by moving entire subtrees (by way of parameters in reaction rules).

2.5 Well-formedness

The grammar for the $\mathcal{C}$-calculus allows us to write expressions for many solutions, which may be hard to interpret biologically. We wish to restrict to those

solutions, which respect the motivations that we have given for each type of construct in the language.

To capture such solutions we give a set of well-formedness criteria. It is convenient to state the requirements on solutions as properties of the (bi)graphs that $\mathcal{C}$-calculus solutions correspond to. The key theorem in the following section, which presents the operational semantics of the $\mathcal{C}$-calculus, states that well-formedness is preserved by the reaction relation induced by the class of reaction-rules that we allow. In verifying this theorem, we shall also make use of the bigraphical underpinning of the $\mathcal{C}$-calculus. We motivate each of the criteria in the paragraphs following the definition.

Definition 2.7 (well-formedness)**.** We say that a solution is well-formed, if

- (*link sorting*) links are well-sorted, i.e., for any two function symbols u, v, if the ith port of u and the jth port of v is part of the same link, then u and v are either two gates, or two domains and $j = i$;
- (*binary channels*) all gates are linked pairwise;
- (*binary complex*) complexation-links are binary;
- (*local complex*) complexation-links are mono-located, i.e., if two domains d and e are linked by their complexation-ports, then they are siblings;
- (*fixed backbone*) for any protein pro with signature (n, r), every domain $\mathsf{pro}i$ is linked by its backbone port to exactly $n - 1$ domains $\mathsf{pro}0$, ..., $\mathsf{pro}i\text{-}1$, $\mathsf{pro}i\text{+}1$, ..., and $\mathsf{pro}n\text{-}1$.
  Further, if $r = 0$, all those domains of are siblings; else if $r > 0$, the domains $\mathsf{pro}0$, ..., $\mathsf{pro}r\text{-}1$ are siblings, and the domains $\mathsf{pro}r$, ..., $\mathsf{pro}n\text{-}1$ are siblings, and these two subunits of the protein are separated by a single membrane.
- (*bitonality*) all connected gates are separated by either 0 or 2 membranes.

Each kind of function symbol represents different types of entities. We require links to be *sorted*, such that links only connect entities from the same class. This ensures us that we only have links that we can interpret as protein backbones, domain-domain complexation links, and channels.

We require that channels and complex-formation links be *binary*. We have already discussed the motivation behind the channel-abstraction. Domain-domain complexation links represent a variety of weak forces attracting certain parts of proteins to each other. While at some level of abstraction, one could conceiveably consider domains interacting with more than one other domain at a time, we shall disregard that possibility in this exposition and require that complexation-links be binary.

We require also for complexation links that they be *local*, that is, that they do not cross membrane borders. The weakness of the forces working to establish complexes also enforce a high degree of locality. Only the backbone of a protein

may cross membranes, the biological correspondent being that the amino-acid structure of a transmembrane protein actually penetrates the membrane.

We require that the backbone link of a protein is *fixed*, such that all instances of proteins respect their protein-signature (cf. Definition 2.1). This simply means, that for any protein pro with signature $(n, 0)$, all instances of pro takes the form

$$\mathsf{pro0} \mid \ldots \mid \mathsf{pro}n\text{-}1.$$

For any protein pro with signature (n, r) for $r > 0$, the following two kinds of configurations are well-formed

$$\mathsf{pro0} \mid \ldots \mid \mathsf{pro}r\text{-}1 \mid \mathsf{m}[\mathsf{pro}r \mid \ldots \mid \mathsf{pro}n\text{-}1 \mid \mathsf{S}]$$
$$\mathsf{m}[\mathsf{pro0} \mid \ldots \mid \mathsf{pro}r\text{-}1 \mid \mathsf{S}] \mid \mathsf{pro}r \mid \ldots \mid \mathsf{pro}n\text{-}1,$$

for some membrane-name $\mathsf{m} \in \mathcal{M}$ and arbitrary well-formed solutions S.

Thus, we may at all times identify the protein to which a domain belongs by its backbone.

Finally, we require that channels respect *bitonality*. This requirement deserves a bit more discussion.

2.5.1 On Bitonality

As discussed earlier, membranes consist of a lipid bilayer, two layers of lipids 'back-to-back'. Such a layer works as an *amphipathic* layer, that is, a layer with both hydrophilic and hydrophobic parts. This layer induces a strong barrier for fluids on the two sides of the membrane. In fact, these forces are also a main stabilizing factor of a membrane.

A direct consequence is that, in a calculus seeking to capture membrane interaction, we should try to enforce that entities may not cross membrane barriers, and that the reconfigurations of membranes only happen in such a way that the orientation of membranes is respected. As discussed by Cardelli for brane calculi [Car04b], we may abstractly capture this by considering the compartments induced by membranes to be coloured alternately black and white according to their nesting depth; and, essentially, then requiring of our operational semantics that it keeps black entities and white entities separate.

We should remark also that, while bitonality serves as good design abstraction, many important reactions in nature do, however, *not* respect bitonality. For instance, the plasma membrane of cells, is penetrated by many structures, some of which work as channels for the direct intake of smaller molecules.

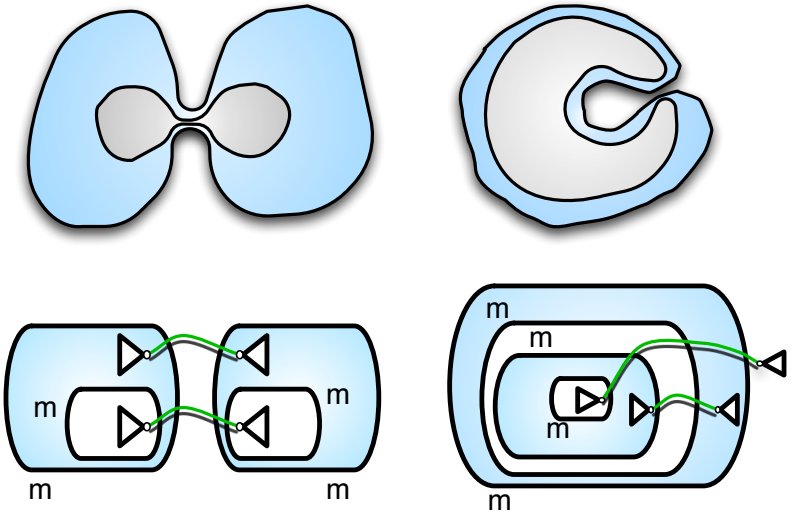
In the $\mathcal{C}$-calculus, any kind of transport between compartments involves channels, and we may enforce a correspondent to the bitonality constraint by requiring that channels form and exist between compartments separated by two membrane surfaces. This allows the basic configurations of membranes and channels that we have illustrated in Figure 3, but disallows a variety of more exotic configurations.

The basic configurations that we wish to disallow are those with channels of odd length, such as

$$(g) \triangle_g \mid \mathsf{m}[\triangle_g].$$

Although, as discussed, in nature limited transport across membranes *do* occur, we wish to keep channels as an abstraction for the neck of partially fused membranes.

In Figure 6, we illustrate two other prominent configurations that are not well-formed. We give the corresponding $\mathcal{C}$-calculus solutions below each illustration. In both cases, it is the h channel that is the perpetrator; it has been stretched, so to speak. On the left, two fusing membranes have stretched across a channel between another pair of partially fused membranes. On the right, we have a similar situation, only with the direction of the fusing membranes inverted.



$(g,h)\,\mathsf{m}[\triangle_g \mid \mathsf{m}[\triangle_h]\!] \mid \mathsf{m}[\triangle_g \mid \mathsf{m}[\triangle_h]\!]$ $\qquad$ $(g,h)\,\triangle_h \mid \mathsf{m}[\triangle_g \mid \mathsf{m}[\mathsf{m}[\triangle_g \mid \mathsf{m}[\triangle_h]\!]]\!]$

Figure 6: Examples of disallowed membrane configurations, and their representation via channels in the $\mathcal{C}$-calculus.

We disallow these configurations in our definition of bitonality, not because they would allow colours to mix (only channels of odd length would do that), but because we would struggle to interpret them biologically. Though biological membranes are quite deformable, their plasticity still makes configurations such as those depicted in Figure 6 very improbable. The examples are prominent, however; we need to take care in the treatment of reaction rules that allow creating or transporting membranes, to disallow reactions that lead to these configurations from well-formed ones. Consider, for instance, this (well-formed) configuration

$$(g,h)\,\mathsf{m}[\triangle_g \mid \mathsf{m}[\triangle_h] \mid \mathsf{m}[\triangle_h]\!] \mid \mathsf{m}[\triangle_g].$$

If we allowed one of the membranes with a gate $\triangle_h$ to be transported across the g channel, we get the leftmost configuration in Figure 6. We may write a

similar wellformed pre-configuration for the rightmost configuration. We shall discuss how to avoid such mishaps in Section 3.

As a special case we allow degenerate channels such as

$$\mathsf{U} = (g)\,\mathsf{m}[\triangle_g \mid \triangle_g].$$

As shall be apparent when we introduce the operational semantics, they may be formed when colocated membranes form several channels between themselves, before fusing. For instance, we shall allow the following chain of reactions to form the solution U:

$$\mathsf{m}[] \mid \mathsf{m}[] \to (g)\,\mathsf{m}[\triangle_g] \mid \mathsf{m}[\triangle_g] \to (g,h)\,\mathsf{m}[\triangle_g \mid \triangle_h] \mid \mathsf{m}[\triangle_g \mid \triangle_h] \to (g)\,\mathsf{m}[\triangle_g \mid \triangle_g].$$

We may illustrate this chain of reactions as in Figure 7.

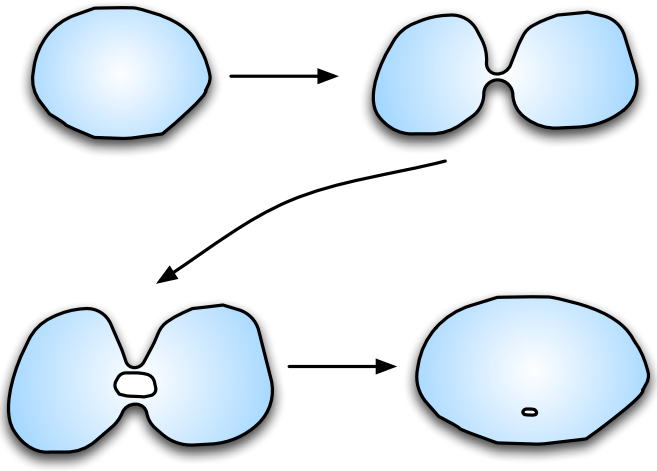


Figure 7: Multi-touch may form degenerate channels.

The first reaction is a basic instance of a so-called *touch*; so is the second reaction. The third reaction is a *fusion* around the h channel, which leaves the g channel as a residual of the first touch.

One could contemplate giving side-conditions on touch-rules to disallow the second reaction; it turns out that this is not sufficient for preserving well-formedness across reaction, however. Multi-channels may be formed by fusion as consequence of a sequence of earlier touch events, each of which should be locally allowable, in the sense that no multi-channels are created. A minimal situation is illustrated in Figure 8. In general, testing for multi-channels is highly non-local requiring a traversal of the entire link graph of channels. And disallowing the final fusion in Figure 8 would be counter-intuitive with regard to biological consistency. Indeed, though multi-touches and the shape of membranes in the second step of Figure 8 are highly unlikely to happen in nature, they are not impossible.

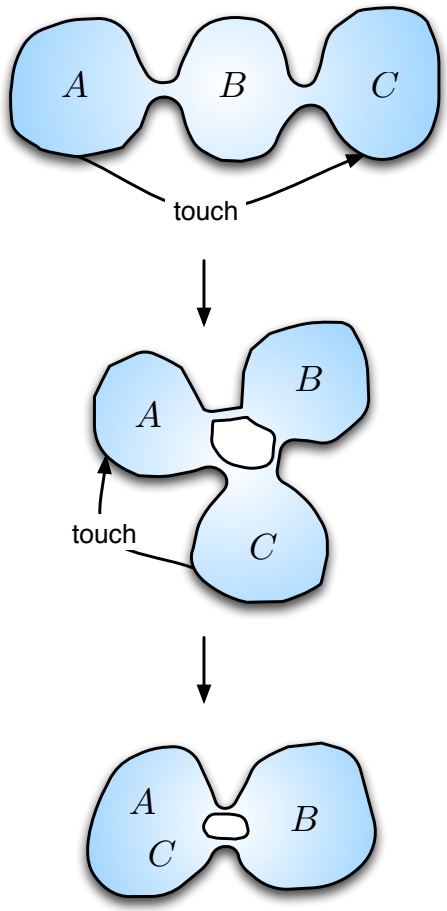


Figure 8: A multi-touch as a consequence of earlier touches.

For the reasons discussed above, we choose to allow multi-touches. This keeps our reaction rules simpler, and it allows us to model membranes still separate, but in different stages of fusing. In turn, the design choice has the side-effect that degenerate channels can be formed. Such degenerate channels are, however, harmless; no diffusion or reconfiguration may occur along them.[7]

3 Dynamics

In this section, we introduce the operational semantics of the $\mathcal{C}$-calculus.

[7] We may even consider extending the structural congruence with a rule to "garbage collect" them, i.e.: $(g) \triangle_g \mid \triangle_g \equiv 0$. However, that would remove the 1-1 correspondence between solutions and bigraphs, in turn preventing us from using bigraphical reasoning in the proofs. Hence, we leave this idea for future work.

A user develops a model in the $\mathcal{C}$-calculus by selecting a set a proteins and membranes to work with (by giving a protein-signature, and a set of membrane-names), and a set of reaction rules. We cannot allow any collection of reaction rules; such a loose policy would not allow us to maintain well-formedness. This would also lay upon the shoulders of modellers the task of inventing their own abstractions of biological events; this in turn would complicate, for instance, composition of different models, something which we would like to be an easy task. Instead, we characterize in this section a fixed set of canonical biological *actions*, encapsulated in classes of certain reaction rules. We allow modellers to refine these actions by giving contextual preconditions for rules. In all, we characterize a set of core rules, which provide modellers with a toolbox of generic actions that can be incorporated into a model by specializing them to a particular signature. Given this toolbox of actions, a $\mathcal{C}$-calculus reactive system can essentially be designed by domain experts (rather than computer scientists).

In summary, modelling in $\mathcal{C}$-calculus is done by giving a signature and choosing and refining a set of core rules over this signature. Thus, as set out in the Introduction, the slogan for modelling in the $\mathcal{C}$-calculus is: *Modelling by rule refinement*.

First, in Section 3.1 we define $\mathcal{C}$-calculus reactive system and explain reaction rules. In Section 3.2, we treat so-called *projective* descriptions of rules. They let us describe conveniently sets of rules for reactions involving regions separated by membrane-surfaces, while eliding the *orientation* of those separating membranes. We continue to introduce reaction rules; they divide into two categories. In Section 3.3, we discuss rules that involves protein-protein interaction, that is, forming or breaking complexation-links, changing the state of domains, or creating or deleting proteins. These kinds of rules work essentially in the κ-like fragment of the calculus, and, as we have treated an encoding of the κ-calculus before [DK08], we shall go into less detail with these kinds of rules. In Section 3.4, we discuss rules involving membrane (or channel) reconfiguration; the rules for transport of material along channels need special care—we shall discuss and motivate them at length. In Section 3.5, we formally define the allowable rules, and, in Section 3.6 we verify that any allowable rule preserves well-formedness.

3.1 $\mathcal{C}$-calculus Reactive Systems

In the $\mathcal{C}$-calculus, we work with reaction rules of the following format:$(\mathsf{L} \to \mathsf{R}, \varphi)$, where L and R are expressions for solutions and φ is a side-condition. The expressions may have a number of variables, which serve to carry parameters unchanged across from the left-hand side to the right-hand side. The side-condition φ is a (possible empty) predicate that tests parameters. We shall use side-conditions for expressing contextual conditions on the surroundings of L and R for the rules handling transport.

The $\mathcal{C}$-calculus reaction rules are fairly simple (as compared to the generality allowed for bigraphical rules, say). We need only consider rules, where free names are preserved, that is, $\mathrm{fn}(\mathsf{L}) = \mathrm{fn}(\mathsf{R})$, and where variables occur

exactly once on the left-hand side and on the right-hand side (i.e., all rules are linear). These are exactly the kind of rules, we have treated in the $\mathcal{B}^{\Sigma,\mathcal{R}}$-framework [DK08], and we may instantiate the theory for reaction and reactive systems from that work. We outline how reaction works below. (As verified in *loc. cit.*, it also follows that $\mathcal{C}$-calculus reaction is bigraphical reaction modulo side-conditions, which ensures us that we may reason (bi)graphically in proofs.)

Loosely, any ground solution S that is a substitution-instance of the left-hand side of a rule may be rewritten with that rule. Given a rule $(\mathsf{L} \to \mathsf{R}, \varphi)$, if for some (ground) group G, we have $\mathsf{S} \equiv \mathsf{L} \cdot \mathsf{G}$, we may perform a reaction $\mathsf{S} \to \mathsf{S}' \equiv \mathsf{R} \cdot \mathsf{G}$, if $\varphi(\mathsf{G})$ is satisfied. We say that S *matches* L with the *parameter* G, since G consists of the group of solutions that we will substitute for the variables in R. Reactions may occur in any solution context. We make this explicit in Definition 3.1.

We contextualize reactions according to standard tradition for process calculi. We close reactions under syntactic constructions, structural congruence, and also under (bijective) renaming of free names. Definition 3.1 records a set of rules that together gives a simple characterization of the reactive system over solutions. The definition is a straight instantiation of the generic operational semantics given for $\mathcal{B}^{\Sigma,\mathcal{R}}$ (in [DK08, Definition 3.16]). Free names are preserved across reaction, that is, if $\mathsf{S} \to \mathsf{S}'$, $\mathrm{fn}(\mathsf{S}) = \mathrm{fn}(\mathsf{S}')$.

As outlined in the introduction to this section, we do not wish, however, to allow all kinds of reaction rules. Therefore, in the Definition 3.1, we have restricted to a set of (as yet unspecified) *allowable* reaction rules. Informally, we wish to allow rules that encapsulates a single biological action, such as breaking a domain-domain bond or fusing two partially fused membranes. The following sections shall be concerned with introducing core rules that incorporate biological actions (Sections 3.3 and 3.4).

Definition 3.1 (reactive system)**.** Given a set of *allowable* reaction rules $\mathcal{R}$ and a signature Σ, $\mathcal{T}^{\mathcal{R}}_{\Sigma}$ the reactive system associated with $\mathcal{R}$, is given by the reaction relation $\to$, the least binary relation over $\mathcal{S}^{\Sigma}$, s.t.

$$\textsc{rule}\frac{(\mathsf{L} \to \mathsf{R}, \varphi) \in \mathcal{R} \quad \exists G \text{ s.t. } \mathsf{S} = \mathsf{L} \cdot G \text{ and } \mathsf{S}' = \mathsf{R} \cdot G \quad \varphi(G) \text{ satisfied}}{\mathsf{S} \to \mathsf{S}'}$$

$$\textsc{par}\frac{\mathsf{S} \to \mathsf{S}'}{\mathsf{S} \mid \mathsf{T} \to \mathsf{S}' \mid \mathsf{T}} \qquad \textsc{membrane}\frac{\mathsf{S} \to \mathsf{S}'}{\mathsf{m}[\mathsf{S}] \to \mathsf{m}[\mathsf{S}']}$$

$$\textsc{close}\frac{\mathsf{S} \to \mathsf{S}'}{(x)\mathsf{S} \to (x)\mathsf{S}'} \qquad \textsc{struct}\frac{\mathsf{T} \equiv \mathsf{S} \quad \mathsf{S} \to \mathsf{S}' \quad \mathsf{S}' \equiv \mathsf{T}'}{\mathsf{T} \to \mathsf{T}'}$$

$$\textsc{subst}\frac{\mathsf{S} \to \mathsf{S}' \quad \exists \tilde{x}.\alpha : \mathrm{fn}(\mathsf{S}) \leftrightarrow \tilde{x}}{\alpha(\mathsf{S}) \to \alpha(\mathsf{S}')}$$

where α is a bijection between the free names of S and fresh names $\tilde{x}$, and $\alpha(\mathsf{S})$ is the solution S with free names substituted by names $\tilde{x}$.

However, we also wish to allow a modeller to test parts of the surroundings, to give application conditions for reactions. We allow a modeller to refine core

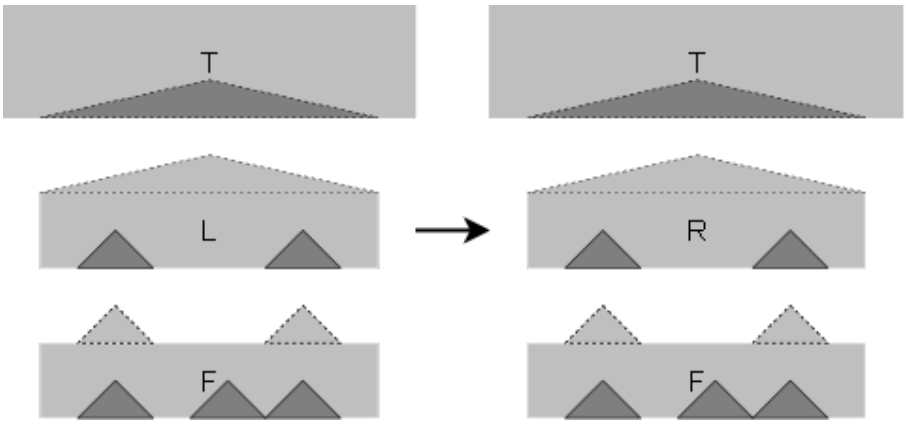
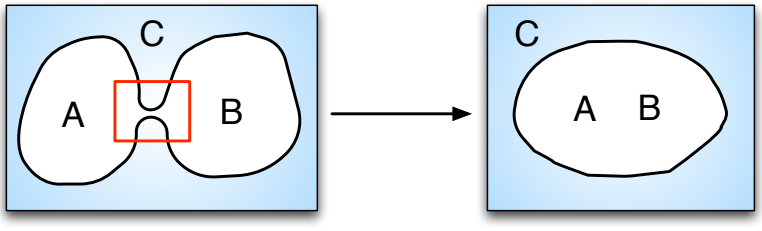


Figure 9: Illustrating refinement of a rule.

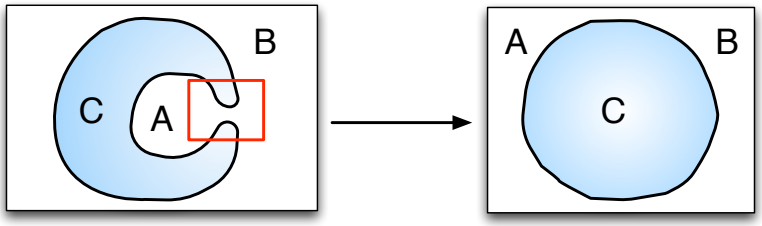
rules by adding extra context to rules. It turns out that we may capture refinement succinctly with substitution. We define formally refinement and allowable rules in Section 3.5 (allowing us to explain and motivate refinement with a concrete example using a core rule introduced in the previous sections). Figure 9 illustrates schematically how refinement works. Suppose we have the core rule $\mathsf{L} \rightarrow \mathsf{R}$, where L and R have two variables each (depicted as dark triangles inside L and R). The user may refine that rule by substituting both sides L and R into an external context, T (with one variable), and by substituting a common group of solutions into the variables of L and R, yielding the rule $\mathsf{T} \cdot \mathsf{L} \cdot \mathsf{F} \rightarrow \mathsf{T} \cdot \mathsf{R} \cdot \mathsf{F}$. For full generality, we still need to treat rules with side-conditions, however; we return to refinement in detail in Section 3.5.

3.2 Projective Descriptions

The model of membranes in the $\mathcal{C}$-calculus is influenced by Cardellis brane calculi [Car04b]. We build, however, also on the insights reported by Pradalier and Danos on the so-called *projective* brane calculi [DP04], and, more recently, *patch* reactions by Cardelly [Car08]. Their insights were to note, that for membrane calculi all reactions that are described for membranes occur irrespective of membrane orientation. For instance, the reaction



is allowed, and so is



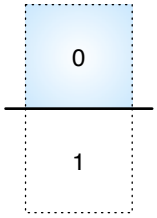
If we focus on the local patches of membranes that are merging (indicated by the rectangular lenses), we may observe that from a *local* perspective, the preconditions for the reactions look highly similar. Only the *orientation* of the curvature of the membranes is different—a property, which is hardly detectable in nature.[8]

Indeed, some descriptions of biological reactions abstract away from the orientation of the membrane, describing instead only the conditions local to a patch of a membrane wall. It does, of course, matter whether a signal propagates inwards or outwards across the cellular membrane; the central observation is, that this is *not* due to the orientation of the cellular membrane.

We wish to allow such descriptions for $\mathcal{C}$-calculus reaction rules; adopting the terminology introduced by Pradalier and Danos, we call the descriptions *projective*. To that end, we start by defining lateral and horizontal *projection* of binary groups of solutions.

3.2.1 Projective Contexts and Projective Groups

In essence, projective descriptions, involve two regions separated by a membrane wall, as depicted below:



We may interpret such descriptions as a schema for two reaction rules, one for each orientation of the separating membrane. In Figure 10, we illustrate the two ways one may interpret a description of two regions separated by a single membrane wall. As depicted, it simply boils down to determining which of the regions numbered 0 or 1 contain the other. We may colour the regions in

[8] Although, as reported in [HR05], curvature *does* occasionally matter and may serve to distinguish and sort vesicles by size.

accordance with the colouring scheme discussed for the bitonality constraint in the previous section, and call the descriptions *heterogenous*, as they involve differently coloured regions.

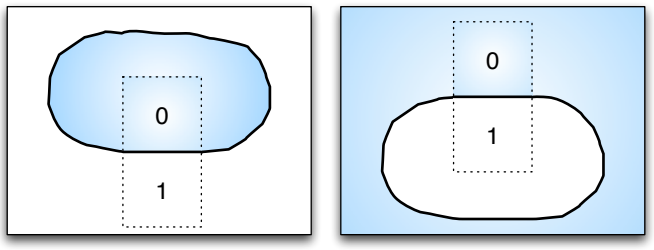
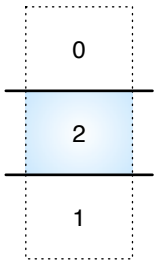


Figure 10: Heterogenous projective contexts illustrated.

Similarly projective descriptions of reactions involving regions separated by two membranes, involve three regions as depicted below:



We call such descriptions *homogenous* as the regions 0 and 1 have equal colour. In Figure 11, we illustrate the three ways one may orient the membranes, such that the regions 0 and 1 are separated (three, as the first is symmetric). Consequentially, we may read homogenous projective descriptions as a schema for three rules.

We may generate projections by substitution into sets of *projective contexts*, defined to match each of the projections in Figures 10 and 11. Formally, we let the two sets of projective contexts be parameterized over one or two names (for heterogenous and homogenous contexts, respectively) to allow projection across any kind of membrane(s).

Definition 3.2 (Projective contexts)**.** For all $\mathsf{m}, \mathsf{n} \in \mathcal{M}$, the *heterogenous* projective contexts $\eta_{\downarrow}^{\mathsf{m}}$ and $\eta_{\uparrow}^{\mathsf{m}}$, and the *homogenous* projective contexts $\omega_{\downarrow}^{\mathsf{m},\mathsf{n}}$, $\omega_{\uparrow}^{\mathsf{m},\mathsf{n}}$, and $\omega_{\leftrightarrow}^{\mathsf{m},\mathsf{n}}$ are

$$\begin{array}{lll} \omega_{\downarrow}^{\mathsf{m},\mathsf{n}} & = \mathsf{m}[\mathsf{n}[\Box_1] \mid \Box_2] \mid \Box_0 & \qquad \eta_{\downarrow}^{\mathsf{m}} \quad = \mathsf{m}[\Box_1] \mid \Box_0 \\ \omega_{\uparrow}^{\mathsf{m},\mathsf{n}} & = \mathsf{m}[\mathsf{n}[\Box_0] \mid \Box_2] \mid \Box_1 & \qquad \eta_{\uparrow}^{\mathsf{m}} \quad = \mathsf{m}[\Box_0] \mid \Box_1 \\ \omega_{\leftrightarrow}^{\mathsf{m},\mathsf{n}} & = \mathsf{m}[\Box_0] \mid \mathsf{n}[\Box_1]. & \end{array}$$

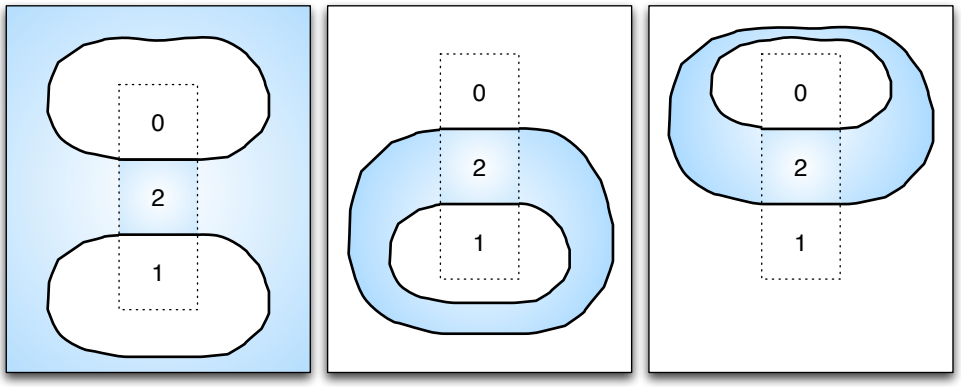


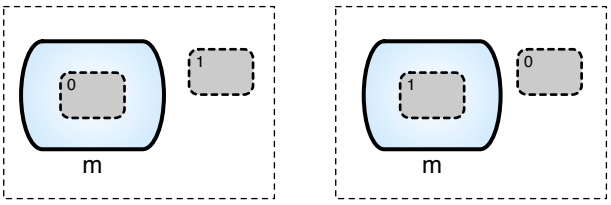
Figure 11: Homogenous projective contexts illustrated.

The sets of generic homogenous and heteregonous projective contexts are

$$\begin{aligned}\omega_\delta = &\quad \{\mathsf{S} \mid \exists \mathsf{m}, \mathsf{n} \in \mathcal{M}.\mathsf{S} = \omega_\delta^{\mathsf{m},\mathsf{n}}\}, \text{for } \delta \in \{\uparrow, \downarrow\} \\ \eta_\delta = &\quad \{\mathsf{S} \mid \exists \mathsf{m}, \mathsf{n} \in \mathcal{M}.\mathsf{S} = \eta_\delta^{\mathsf{m}}\}, \text{for } \delta \in \{\uparrow, \downarrow, \leftrightarrow\}.\end{aligned}$$

We shall typically use the sets of generic projective contexts, because we want to refer to the set of projective contexts generated by all valid membrane-names; for instance, $\omega_\downarrow = \{\mathsf{S} \mid \exists \mathsf{m}, \mathsf{n} \in \mathcal{M}.\mathsf{S} = \omega_\downarrow^{\mathsf{m},\mathsf{n}}\}$. In the following we shall allow ourselves a convenient sloppyness: to use directly in expressions the unqualified projective contexts, such as in the expression $\omega_\downarrow \triangleleft \mathsf{G}$. Formally, this means that we are talking about a set of solutions $\{\mathsf{S} \triangleleft \mathsf{G} \mid \exists \mathsf{m}, \mathsf{n} \in \mathcal{M}.\mathsf{S} = \omega_\downarrow^{\mathsf{m},\mathsf{n}}\}$. This shall be particularly convenient when writing schemas for rules.[9]

We depict the contexts defined in Definition 3.2 in Figures 12 and 13 (colouring the compartments for effect).

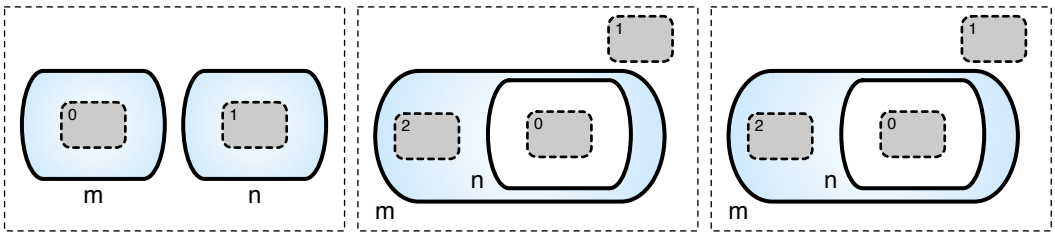


$\eta_\uparrow^{\mathsf{m}} = \mathsf{m}[\Box_0] \mid \Box_1$ $\quad$ $\eta_\downarrow^{\mathsf{m},} = \mathsf{m}[\Box_1] \mid \Box_0$

Figure 12: Heterogenous projective contexts.

Now we may define heterogenous and homogenous projection of binary groups by way of substitution.

[9] In the following definitions, we shall consistently use partial substitution $\mathsf{G} \triangleleft \mathsf{S}$, though total substitution $\mathsf{G} \cdot \mathsf{S}$ would be sufficient in some cases (specifically, for heterogenous contexts). (Recall, that $\triangleleft$ denotes partial substitution—see Section 2.3.2 and Def. 6.9).



$\omega^{\mathsf{m},\mathsf{n}}_{\leftrightarrow} = \mathsf{m}[\Box_0]|\mathsf{n}[\Box_1]$ $\omega^{\mathsf{m},\mathsf{n}}_{\uparrow} = \mathsf{m}[\mathsf{n}[\Box_0]|\Box_2]|\Box_1$ $\omega^{\mathsf{m},\mathsf{n}}_{\downarrow} = \mathsf{m}[\mathsf{n}[\Box_1]|\Box_2]|\Box_0$

Figure 13: Homogenous projective contexts.

Definition 3.3. Given a binary group $\mathsf{G} = \mathsf{S} \mid \mathsf{T}$, its heterogenous projections is the sets of solutions generated by

$$\eta_{\downarrow} \lhd \mathsf{G}, \text{ and } \eta_{\uparrow} \lhd \mathsf{G},$$

and its homogenous projections is the sets of solutions generated by

$$\omega_{\downarrow} \lhd \mathsf{G}, \omega_{\uparrow} \lhd \mathsf{G}, \text{ and } \omega_{\leftrightarrow} \lhd \mathsf{G}.$$

Letting δ range over the directions $\{\uparrow, \downarrow, \leftrightarrow\}$ (up, down, and lateral), we may summarize Definition 3.3, by stating that the heterogenous projections are those characterized by

$$\eta_{\delta} \lhd \mathsf{G},$$

while the homogenous projections can be characterized as

$$\omega_{\delta} \lhd \mathsf{G}.$$

3.2.2 Projective Rules

We utilize our characterization of projective groups, to define projective descriptions of reaction rules.

We define projective *generators*, projective descriptions that constitute schemas for rules, which we *project* to form concrete rules.

Let $\rho \in \{\omega, \eta\}$, and recall that $\delta \in \{\uparrow, \downarrow, \leftrightarrow\}$. We write projective generators as

$$\mathsf{rule}^{\rho} : \mathsf{L} \rightarrow_{\rho} \mathsf{R},$$

using the subscripted ρ as reminder that these are rule-schemas that await a projective direction before application.

From generators, we generate *projected* rules. For homogenous generators we have

$$\mathsf{rule}^{\omega}_{\delta} : \omega_{\delta} \lhd \mathsf{L} \rightarrow \omega_{\delta} \lhd \mathsf{R},$$

and for heterogenous generators we have

$$\mathsf{rule}^{\eta}_{\delta} : \eta_{\delta} \lhd \mathsf{L} \rightarrow \eta_{\delta} \lhd \mathsf{R}.$$

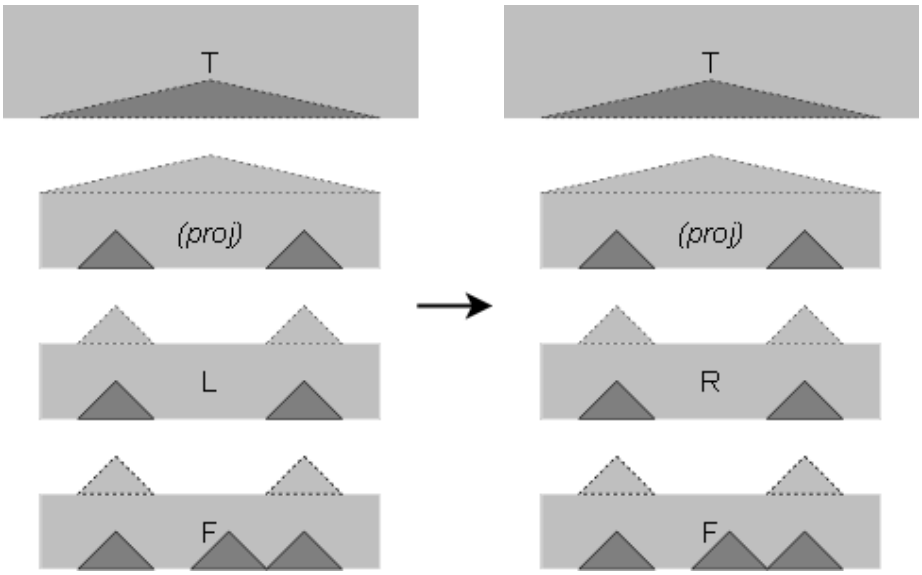


Figure 14: Illustrating refinement of projective generators.

For some generators, we shall declare once and for all, that they are either heterogenously or homogenously projective, and elide the projection, ρ, from the name of the rule. We depict generators in correspondence with the illustrations for projective groups above, separating groups with one or two horizontal lines denoting the projected membranes.

Expanding on the illustration of refinement in Figure 9, Figure 14 illustrates schematically how generators are projected to form concrete rules, before giving application conditions for them. In the illustration, L and R are groups containing two solutions, and we suppose that $\mathsf{L} \rightarrow_\rho \mathsf{R}$ is a heterogenous generator. By inserting the group into the heterogenous projective contexts, we may obtain (projected) rules. These rules we may refine as usual.

3.3 Protein-protein Interaction

Our domain-level model of proteins is closely related to that of the κ-calculus, only generalized to a distributed setting by using backbone links instead of nodes to connect domains of the same protein. In the $\mathcal{C}$-calculus, we shall allow essentially the same kind of reactions as in the κ-calculus, that is, rules creating or destroying protein complexation-links and/or changing domain state. We also take rules that create or delete proteins, whose domains are all unbound (i.e., visible or hidden).

For the κ-calculus it has been discussed in detail, how one may restrict to rules with a suitable level of atomicity (see [DL04]). This level of atomicity may be enforced by restricting to so-called *monotone* and *anti-monotone* re-

action rules. As we have essentially imported a generalized version of the core κ-calculus inside the $\mathcal{C}$-calculus, we may adopt such (anti-)monotonicity requirements more or less directly. We elide such treatment here, as the restrictions are not central for the main theorem—the preservation of well-formedness—that we wish to establish; and because we wish to focus on the extension of a κ-like calculus with dynamic compartments.

We take instead a simpler set of rules that allow to us capture core κ-reactions. Definition 3.4 defines base domain-level rules; a set of rules, which each incorporate a single core biological action.

Definition 3.4 (base domain-level rules)**.** For arbitrary domains d and e, we allow the following rules:

$$\begin{array}{lrcl}
\textsc{bind}: & \mathsf{d}_a \mid \mathsf{e}_b & \rightarrow & (x)\,(\mathsf{d}^x_a \mid \mathsf{e}^x_b) \\
\textsc{break}: & \mathsf{d}_a \mid \mathsf{e}_b & \leftarrow & (x)\,(\mathsf{d}^x_a \mid \mathsf{e}^x_b) \\
\textsc{show}: & \overline{\mathsf{d}}_a & \rightarrow & \mathsf{d}_a \\
\textsc{hide}: & \overline{\mathsf{d}}_a & \leftarrow & \mathsf{d}_a .
\end{array}$$

We allow also rules for *synthesis*, protein-creation, and for *degradation*, protein-deletion. These kinds of rules come in two variants depending on the signature of the protein involved. For proteins p with $\Sigma^{\mathcal{P}}(\mathsf{p}) = (a, 0)$, they take the forms:

$$\begin{array}{lrcl}
\textsc{synth}: & 0 & \rightarrow & (b)\,\left(\mathsf{p}\dot{0}_b \mid \ldots \mid \mathsf{p}\dot{a}\text{-}1_b\right) \\
\textsc{degrade}: & 0 & \leftarrow & (b)\,\left(\mathsf{p}\dot{0}_b \mid \ldots \mid \mathsf{p}\dot{a}\text{-}1_b\right),
\end{array}$$

where we use the shorthand $\mathsf{p}\dot{i}$ to range over visible or hidden domains $\mathsf{p}i$, i.e., $\mathsf{p}\dot{i} \in \{\mathsf{p}i, \overline{\mathsf{p}i}\}$. (In other words, we allow SYNTH and DEGRADE rules to create or delete domains that are unbound—whether visible or hidden.)

For receptor-proteins p, with $\Sigma^{\mathcal{P}}(\mathsf{p}) = (a, r)$ for $r > 0$, synthesis and degradation are described by the following projective generators:

$$\begin{array}{lrcl}
\textsc{synth}: & 0 \mid\mid 0 & \rightarrow_\eta & (b)\,\left(\mathsf{p}\dot{0}_b \mid \ldots \mid \mathsf{p}\dot{r}\text{-}1_b \mid\mid \mathsf{p}\dot{r}_b \mid \ldots \mid \mathsf{p}\dot{a}\text{-}1_b\right) \\
\textsc{degrade}: & 0 \mid\mid 0 & \leftarrow_\eta & (b)\,\left(\mathsf{p}\dot{0}_b \mid \ldots \mid \mathsf{p}\dot{r}\text{-}1_b \mid\mid \mathsf{p}\dot{r}_b \mid \ldots \mid \mathsf{p}\dot{a}\text{-}1_b\right)
\end{array}$$

The rules are pairwise inverses. We can summarize the actions in the reaction rules as follows:

- BIND establishes binary complexation links between visible and co-located domains;
- BREAK breaks closed complexation links between (bound) domains;
- HIDE/SHOW changes the domain-state between visible and hidden;
- SYNTH allows creation of the domains and the backbone that constitutes a protein; all domains should be unbound (i.e., visible or hidden); and,
- DEGRADE allows deletion of the domains and the backbone that constitutes a protein; all domains should be unbound (i.e., visible or hidden).

For SYNTH (and DEGRADE), we require further that the protein is created (or deleted) in a configuration, which respects its protein signature (cf. Definition 2.1).

Together the actions incorporated into the base rules allow us to capture reaction for proteins comparable to those in the κ-calculus, extended to a multi-compartment setting, but with considerably tighter atomicity-requirements on reactions. In the κ-calculus, one may, for instance, break several complexation-links in unison or synthesize proteins and bind them to other proteins in one go. It is not particularly hard to relax the atomicity requirements we have given on domain-level rules in the $\mathcal{C}$-calculus; however, in this paper, we shall make do with a few extra rules combining the actions captured in the base domain-level rules.

Definition 3.5 (domain-level rules)**.** Domain-level rules are those described by BIND, BREAK, HIDE, SHOW, SYNTH, and, DEGRADE above, and the following rules (for arbitrary domains d, e and f):

$$\begin{array}{lrcl}
\text{BIND+HIDE}: & \mathsf{d}_a \mid \mathsf{e}_b \mid \mathsf{f}_c & \to & (x)\left(\mathsf{d}^x_a \mid \mathsf{e}^x_b \mid \bar{\mathsf{f}}_c\right) \\
\text{BREAK+SHOW}: & \mathsf{d}_a \mid \mathsf{e}_b \mid \mathsf{f}_c & \leftarrow & (x)\left(\mathsf{d}^x_a \mid \mathsf{e}^x_b \mid \bar{\mathsf{f}}_c\right) \\
\text{BIND+SHOW}: & \mathsf{d}_a \mid \mathsf{e}_b \mid \bar{\mathsf{f}}_c & \to & (x)\left(\mathsf{d}^x_a \mid \mathsf{e}^x_b \mid \mathsf{f}_c\right) \\
\text{BREAK+HIDE}: & \mathsf{d}_a \mid \mathsf{e}_b \mid \bar{\mathsf{f}}_c & \leftarrow & (x)\left(\mathsf{d}^x_a \mid \mathsf{e}^x_b \mid \mathsf{f}_c\right).
\end{array}$$

Together the extra kinds of rules, BIND+SHOW, BREAK+HIDE, BIND+HIDE, and BREAK+SHOW, allows us to model that creation or deletion of a complexation-link may invoke a change of conformation on a protein.

One can also combine other of the base rules, obtaining rules such as SYNTH+BIND to synthesize a protein *and* bind one of its domains to an existing domain, or BIND+BIND to allow two complexation-links to be created in one go. For the concrete examples we shall discuss in Section 4, the set of rules captured in Definition 3.5 is sufficient, however.

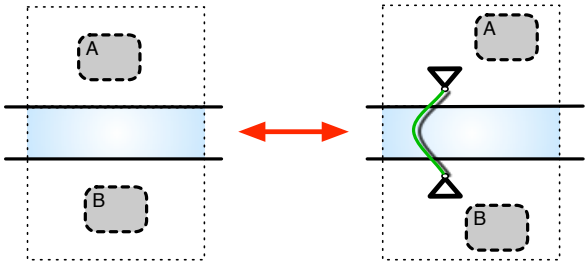
3.4 Membrane Reconfiguration

Membrane-based actions model structural reorganization of membranes such as fusion or fission. As we discussed in Section 2.5, as studied by pioneering works [DP04, Car08], oriented membrane interactions can be factored into a small set of projective interactions, which do not take into account the global orientation of the involved membranes. We capture this by giving generators for projective rules as defined in Section 3.2.

3.4.1 Touch and Part

We start by describing generators incorporating the dual actions of establishing and breaking a channel between colocated membranes. In terms of the bitonality constraint, these involve regions separated by two membrane surfaces; hence the generators are homogenous, that is, we generate rules by composing with the homogenous contexts (cf. Definition 3.2).

The TOUCH and PART generators are each others inverses; they are depicted in Figure 15. The PART generator allows us to model the first step of a fusion process. It creates a channel between two colocated membranes, bringing them into a partially fused state.



TOUCH ($\rightarrow_\omega$) and PART ($\leftarrow_\omega$): $A \,||\, B \;\leftrightarrow_\omega\; (x)(\triangle_x \mid A \,||\, \triangle_x \mid B)$

Figure 15: The homogenous generators TOUCH and PART.

The PART generator is the counterpart of TOUCH; it allows one membrane to detach from another by severing a channel that links the two.

3.4.2 Bud and Merge

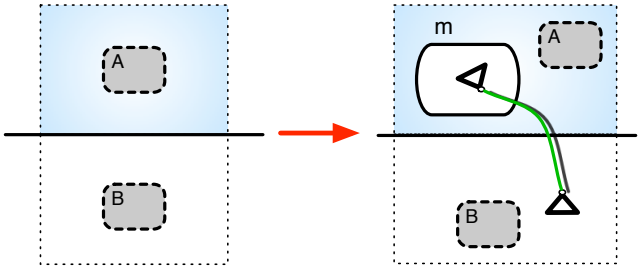
In this section, we treat generators incorporating the actions of creating and destroying a membrane. These generators are heterogenous; they involve a single membrane, which bends and *buds* a new small compartment; or, which *merges* entirely with another compartment that it is partially fused with.

The BUD generator, on the left in Figure 16, describes how a membrane may start to divide to create a new compartment.

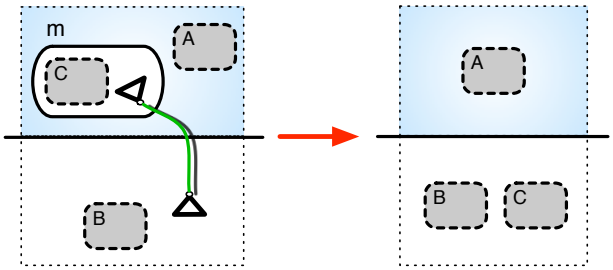
The MERGE generator, on the right in Figure 16, describes how a membrane may finish its fusion with another membrane. A membrane merges into another by releasing its content, C, into the second membrane. The MERGE generator clearly generates left inverses of BUD (take C to be empty).

Note, that the MERGE generator applies even if the involved membranes have channels—to other membranes in the context, or to the projected membrane containing B. (The latter case, is the one we have discussed already in Section 2.5.) This neatly captures that after merging two membranes, the resulting membrane inherits the partial fusings of both membranes.

The BUD generator creates an empty compartment that is still connected to the original membrane via a channel. This channel will allow us to model exchange of materials between the two compartments (by diffusion, which we shall treat below). For simple models where empty vesicles are budded from the cellular membrane, this may be sufficient. However, in most natural cases, a bud is formed around something, for instance, a complex bound to the cell wall. (For a concrete example, see the model of the formation of clathrin-coated vesicles in Section 4.2.) To allow this, we need to allow the (deterministic) transfer of a parameter into the newly formed membrane. To fullfill that need,



BUD: $A \,||\, B \;\rightarrow_\eta\; (x)(A \mid \mathsf{m}[\triangle_x] \,||\, B \mid \triangle_x)$



MERGE: $(x)(A \mid \mathsf{m}[\triangle_x \mid C] \,||\, \triangle_x \mid B) \rightarrow_\eta A \,||\, C \mid B$

Figure 16: The heterogenous generators BUD and MERGE.

in the following section we define the PINCH generator, a generalization of the BUD generator.

3.4.3 Transport Rules

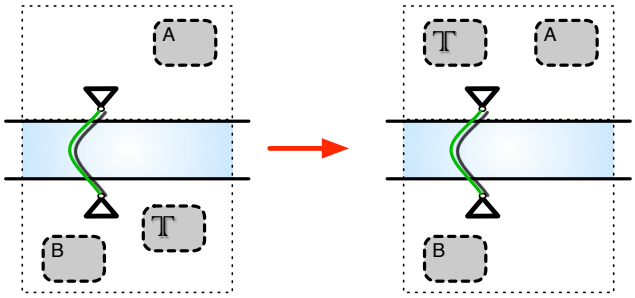
In this section, we shall discuss how to encapsulate *transport*. We define two generators, one that encapsulates the action of diffusion, and another, the PINCH generator, a generalization of BUD introduced above. These generators involve transport of complexes across channels, and we need to take care to uphold well-formedness. (The MERGE generator already allows transport, of course, but in a manner non-problematic with regard to well-formedness.)

We start by developing a model of transport for the generator encapsulating diffusion. We discuss and motivate in some detail the choices that we make; also to show that the generators that we give are not cast in iron, but may be slanted towards different scenarios depending on the biological model one has in mind. We round off by applying our model of transport to generalize BUD to PINCH.

The action we need to encapsulate is fairly simple: "Should any complex find itself beside a (non-degenerate) channel, it may travel across it." From this specification it is immediate that the generator is homogenous (since well-formed channels go between compartments separated by two membrane-surfaces), and

needs to be parametric in the cargo, that is, the complex (or complexes) that it allows to diffuse. Furthermore, in nature diffusion is typically highly regulated, so we need to allow a modeller to specify that cargo-parameter.

Our first attempt is the following rule:



$$(x)\,(A \,|\, \triangle_x \,||\, \triangle_x \,|\, \mathbb{T} \,|\, B) \rightarrow_\omega (x)\,(A \,|\, \mathbb{T} \,|\, \triangle_x \,||\, \triangle_x \,|\, B), \text{ for any closed complexes } \mathbb{T}$$

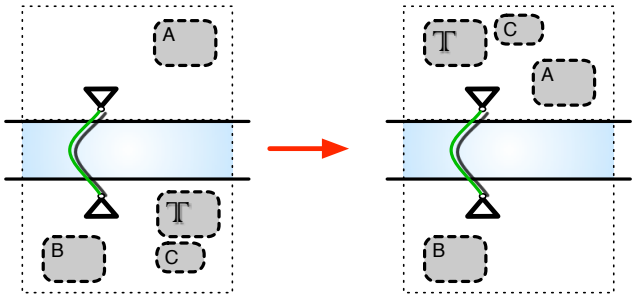
Here $\mathbb{T}$ denotes the cargo-parameter given by the modeller. We certainly preserve well-formedness if we restrict $\mathbb{T}$ to be a closed product of zero or more domains; that is just another way to require that $\mathbb{T}$ contain zero or more fully specified complexes. In other words, formally we require that $\mathbb{T}$ be on the form:

$$\mathbb{T} \equiv (\tilde{y})(\mathsf{d}_{b_1}^{x_1} \mid \cdots \mid \mathsf{d}_{b_k}^{x_k}),$$

where all for i, $x_i \in \tilde{y}$ and $b_i \in \tilde{y}$.

The generator above is computationally complete, in the sense that it allows transport of arbitrary complexes. So we might leave it at that. It is, however, also somewhat impractical to use; there is a huge number of possible configurations of complexes (biologically speaking, *species*) in which a particular protein may be a part. Furthermore, in many cases a central regulation mechanism of diffusion consists of certain proteins or complexes that act as "chaperones"; they essentially allow anything that they bind to to pass. In such situations it would be unfeasible to require a modeller to enumerate all the possible complexes that a chaperone was known to bind to, as well as being a somewhat unfaithful capture of the biological knowledge. In essence, to help practical modelling, we want to be able to give a rule to say that: "any complex(es) C bound to a chaperone (given in $\mathbb{T}$) may diffuse", without having to specify any more than $\mathbb{T}$.

Consequentially, we need to allow $\mathbb{T}$ to be open, to allow it to drag one or more complexes along with it. Our next attempt is then the following rule:



$$\frac{\mathrm{fn}(\mathbb{T} \mid C) = \emptyset}{(x)\,(A \mid \triangle_x \mid\mid \triangle_x \mid \mathbb{T} \mid C \mid B) \rightarrow_\omega (x)\,(A \mid \mathbb{T} \mid C \mid \triangle_x \mid\mid \triangle_x \mid B)}$$

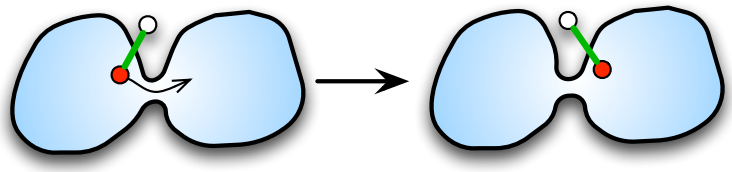
$\mathbb{T}$ still denotes the user-given parameter, while C contains the complexes that are dragged along with $\mathbb{T}$. Again, we require that $\mathbb{T}$ be on the form

$$\mathbb{T} \equiv (\tilde{y})(\mathsf{d}_{b_1}^{x_1} \mid \cdots \mid \mathsf{d}_{b_k}^{x_k}),$$

but this time stipulating no conditions on the names (thus allowing $\mathbb{T}$ to be open). Instead, we require that the product of $\mathbb{T}$ and C be closed. We state this formally as a contextual side-condition, as it needs to be checked for the unknown parameter C before a reaction may occur.[10]

This version of the diffusion-rule allows us to give diffusion-rules that only mention chaperones, and not the cargo that they drag along. However, the generator may also allow any solution *not* connected to $\mathbb{T}$ to travel along in C, thus undermining the possibility of modelling highly regulated chaperoning.

The generator above is also too fine-grained to allow a phenomenon known as *receptor-sliding*.When a vesicle is forming on a cell membrane, receptors tied to the membrane wall may on some occasions *slide* from the cytoplasm to the vesicle. The figure below illustrates the pivot-motion that the receptor-protein follows:



Our generator, as stated above, locks receptors to their membranes; to allow receptors to travel, we need to allow an open complex to travel along a channel. In conclusion, our second version is both too coarse, allowing any context to be dragged along, and in a sense also too fine-grained, disallowing receptor-sliding.

Taking a lesson from these failings, let us try to specify more precisely what it is that we would like to drag along with $\mathbb{T}$: We want to capture the *local*,

[10] A central contribution of the $\mathcal{B}^{\Sigma,\mathcal{R}}$-framework was to work out how such contextual side-conditions may be adopted for bigraphically based calculi (see [DK08]).

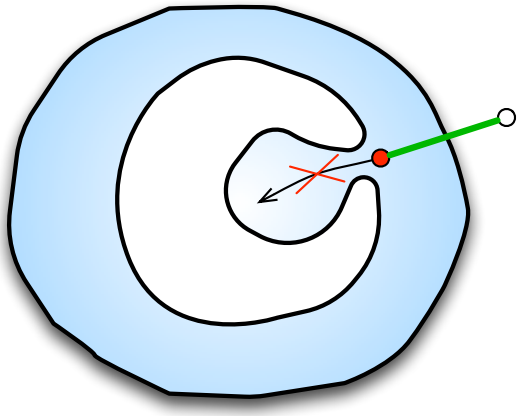
minimal context of the parameter $\mathbb{T}$. By local, we mean colocated with domains in $\mathbb{T}$; by minimal, we mean to restrict to domains connected to domains in $\mathbb{T}$. In other words, we want to capture the *local connected component* of domains in $\mathbb{T}$. Let us first try to formulate a non-constructive side-condition to capture this specification:

$$\frac{B \equiv B^+ \mid B^- \qquad B^+ \text{ minimal s.t. } \mathrm{fn}(B^-) \cap \mathrm{fn}(\mathbb{T} \mid B^+) = \emptyset \qquad \delta = \downarrow : \mathrm{fn}(\mathbb{T} \mid B^+) = \emptyset}{(x)\,(A \mid \triangle_x \mid\mid \triangle_x \mid B^- \mid B^+ \mid \mathbb{T}) \;\rightarrow_\omega\; (x)\,(A \mid \triangle_x \mid \mathbb{T} \mid B^+ \mid\mid \triangle_x \mid B^-)},$$

allowing the same form for $\mathbb{T}$ as in our previous attempt.

First of all, note that instead of allowing any split of surroundings of $\mathbb{T}$ on the left-hand side (as in the previous rule), we specify in the side-condition that the parameters B^- and B^+ should be a certain split of the surroundings of $\mathbb{T}$. Here, we have simply stated informally, that B^+ must be "minimal", such that there are no links between $\mathbb{T}$ plus its dragged cargo B^+, and B^-, the part of the context left behind.[11]

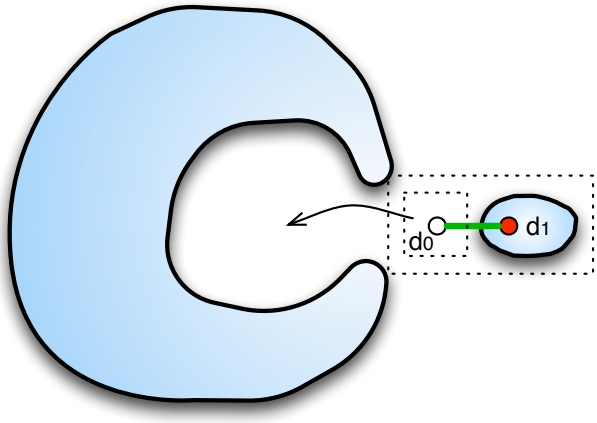
Note also, that we have kept the side-condition requiring closure of the entire cargo $\mathbb{T} \mid B^+$, for the downwards projection of the generator. This breaks from the intuition that membrane reconfiguration is projective, which we described in Section 3.2. Indeed, it turns out that in the $\mathcal{C}$-calculus, because we model cross-membrane links, such as receptor-backbones and channels, the intuition underpinning projectivity does not hold for a certain kind of limit case. And, it is for that reason that we keep the side-condition for the downwards projection. The limit case, which the side-condition prevents, essentially breaks down to a simple geometrical property, and can be illustrated as below:



We wish to transfer the domain d_0 (indicated by the dotted rectangle) down the channel. Trying to compute the minimal surroundings of d_0, we find that it is tied to a membrane in the context that encapsulates also the membranes with

[11] Note, that non-aliasing matching as introduced and discussed in [DK08] is necessary to infer the lack of any immediate links from the lack of name-sharing expressed by the side-condition.

the channel. Without breaking links or violating well-formedness we cannot perform any transfer; the problem is precisely that the tied membrane *surrounds* the partially fused membranes. If the tied membrane had had the inverse projection, a transfer would in principle be possible, if we allow membranes to be diffused:



In this case, we may compute a closed cargo to be transfer along with d_0 (indicated by the outer dotted rectangle).

The conclusion is, that membrane curvature *does* matter when our transfer is directed downwards. We cannot drag along enclosing membranes; thus the side-condition restricts to closed cargoes for the downwards projection.

The fact remains that the side-condition is still non-constructive and informally stated, however. More critically, our analysis also highlights another flaw: the minimal context of $\mathbb{T}$ may contain a membrane, as illustrated above.

In other words, as the generator is stated above, we also admit the dragging of membranes in B^+ bound to receptors in $\mathbb{T}$. Though this preserves well-formedness as we have defined it in Definition 2.7, it is hardly biologically feasible. Though on some occasions tiny vesicles may travel along certain special channels (such as in tunneling nano-tubes [RSM$^+$04]), our channels are intended mainly as an abstraction allowing transport of protein complexes, not membranes. The difference in scale between proteins and membranes is such that, broadly speaking, the "pulling power" of forces affecting proteins cannot drag along a membrane. In a situation such as the one illustrated above, the most likely scenario would probably be that the tied membrane prevented any transfer from happening at all; while for certain applications, a more likely scenario might be that any tied complexation-links be broken. In this paper, we shall make the first choice, as it equips diffusion with a transactional guarantee.

Consequentially, in giving a concrete definition of "minimality", we would like to capture also that membranes are too large to travel in channels. However, as the discussion above illustrates, it is probably beneficial not to hardwire exactly the collection of the cargo. Hence, we choose to isolate and implement the final version of the side-condition as a small function that computes the

surroundings. For certain applications this function may then be tweaked. We may, for instance, specify in this function that under certain conditions the transport is not possible; thus allowing us to prevent membrane-dragging.

We cannot get around, however, that computing the connected component of a node in a graph requires iteration. In our case, we want to find a particular subset of the connected component of the user-parameter $\mathbb{T}$ (following backbone-links and complexation-links); namely the part of it that is colocated with $\mathbb{T}$. Formally, we can define the function that computes the local connected component by taking the least fixed point of a function that computes the immediate (local) neighbours; as usual, we may then compute this least fixed point by iterating the latter function.

We start by defining collect(T, S), the function that collects those top-level domains in S (i.e., not nested inside membranes), which are connected with domains in T.

Definition 3.6 (collecting immediate local neighbours)**.** The function collect(T, S) is defined for any solution T and any open solution

$$\mathsf{S} \equiv (\mathsf{S}_0 | \ldots | \mathsf{S}_{n-1}),$$

where each S_i is either a domain, a gate or a membrane (empty or with another solution inside).

The function is defined as follows (iterating over S from right to left):

$$\begin{array}{rcl}
\text{collect}(\mathsf{T}, 0) & = & (\mathsf{T}, 0) \\
\text{collect}(\mathsf{T}, \mathsf{S} \mid \mathsf{d}_b^x) & = & \text{collect}(\mathsf{T}, \mathsf{S}) \mid (0, \mathsf{d}_b^x), \text{ if } \{x, b\} \cap \text{fn}(\mathsf{T}) = \emptyset \\
\text{collect}(\mathsf{T}, \mathsf{S} \mid \underline{\mathsf{d}_b}) & = & \text{collect}(\mathsf{T}, \mathsf{S}) \mid (0, \underline{\mathsf{d}_b}), \text{ if } \{b\} \notin \text{fn}(\mathsf{T}) \\
\text{collect}(\mathsf{T}, \mathsf{S} \mid \overline{\mathsf{d}_b}) & = & \text{collect}(\mathsf{T}, \mathsf{S}) \mid (0, \overline{\mathsf{d}_b}), \text{ if } \{b\} \notin \text{fn}(\mathsf{T}) \\
\text{collect}(\mathsf{T}, \mathsf{S} \mid \triangle_g) & = & \text{collect}(\mathsf{T}, \mathsf{S}) \mid (0, \triangle_g) \\
\text{collect}(\mathsf{T}, \mathsf{S} \mid \mathsf{m}[\mathsf{S}']) & = & \text{collect}(\mathsf{T}, \mathsf{S}) \mid (0, \mathsf{m}[\mathsf{S}']),
\end{array}$$

where $\mid$ is lifted pointwise to tuples of solutions.

Formally, the normal form for solutions ensures us that collect(T, S) is defined for all open solutions S (cf. Definition 6.4 in Appendix 6). The function collect(T, S) returns another tuple of solutions $(\mathsf{T}', \mathsf{S}')$, where those domains in S connected to T (i.e., sharing names) have been removed from S and added to T, to form T'. This can be summed up in the following little lemma.

Lemma 3.7. *If* collect$(\mathsf{T}, \mathsf{S}) = (\mathsf{T}', \mathsf{S}')$ *then there exists some solution* S''*, s.t.* $\mathsf{T}' \equiv \mathsf{T} \mid \mathsf{S}''$ *and* $\mathsf{S} \equiv \mathsf{S}' \mid \mathsf{S}''$*.*

In other words, we always have collect$(\mathsf{T}, \mathsf{S} \mid \mathsf{S}') \equiv (\mathsf{T} \mid \mathsf{S}, \mathsf{S}')$, for some S. It is also easy to see that collect(T, S) has least fixed points; either collect(T, S) removes part of S, making further progress towards the base case for collect$(\mathsf{T}, 0)$; or nothing is removed from S, and we are done.

Lemma 3.8. *The function* collect *has least fixed points.*

The least fixed point can be computed by iterating collect*; call the least fixed point* collect**.*

And finally, we can define lcc, the function that computes the *draggable* local connected component. We define $\mathrm{lcc}(\mathsf{T}, \mathsf{S})$ as a partial function, that uses $\mathrm{collect}^*$ to compute the local connected component $(\mathsf{T}^*, \mathsf{S}^*)$, and then tests whether the resulting solutions are still connected, that is, still share names. If so, this is because domains are tied across a membrane, and we let lcc be undefined in this case.

Definition 3.9 (draggable local connected component)**.** For

$$\mathsf{T} \equiv (\tilde{x})\, \mathsf{T}',$$

where T' is a parallel product of domains, and

$$\mathsf{S} \equiv (\tilde{y})\, \mathsf{S}',$$

where S' is any open solution, s.t. $\tilde{x} \cap \tilde{y} = \tilde{x} \cap \mathrm{fn}(\mathsf{S}) = \tilde{y} \cap \mathrm{fn}(\mathsf{T}) = \emptyset$,

$$\mathrm{lcc}(\mathsf{T}, \mathsf{S}) = \begin{cases} ((\tilde{y})\, \mathsf{T}^*, (\tilde{y})\, \mathsf{S}^*) & \text{if } \mathrm{collect}^*(\mathsf{T}, \mathsf{S}') = (\mathsf{T}^*, \mathsf{S}^*) \text{ and } \mathrm{fn}(T^*) \cap \mathrm{fn}(S^*) = \emptyset \\ \bot & \text{else} \end{cases}$$

The restrictions on the bound and free names of T and S are purely technical, as we can always α-convert bound names to avoid any clashes.

We state in the following proposition the properties of the solutions computed by lcc. The proposition is straight-forwardly verified from the definitions and lemmas above.

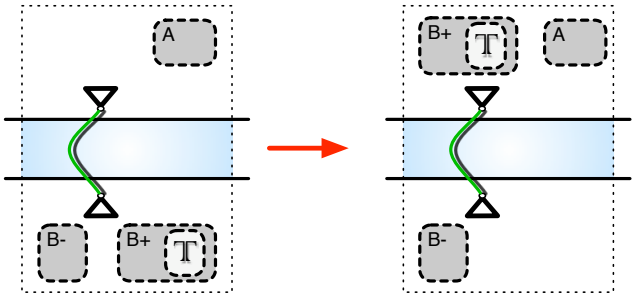
Proposition 3.10. *Either* $\mathrm{lcc}(\mathsf{T}, B^+ \mid B^-)$ *is undefined or* $\mathrm{lcc}(\mathsf{T}, B^+ \mid B^-) \equiv (\mathsf{T} \mid B^+, B^-)$*, and* B^+ *is solution that contains only domains, such that every domain in* B^+ *can trace a path of (backbone or complexation) links to a domain in* T*, and such that no domain in* B^- *can trace such a path.*

In other words, the result of invoking lcc on the user-parameter and its context, $\mathrm{lcc}(\mathbb{T}, B^+ \mid B^-)$ is either undefined, indicating that a tied membrane prevents transfer, or a result $(\mathbb{T} \mid B^+, B^-)$, such that B^+ is a parallel product of domains; and, all domains in B^+ are in the connected component of $\mathbb{T}$, while none of the domains in B^- are.

Having captured and isolated the computation of the solution that $\mathbb{T}$ drags along in lcc, we may now state the final generator for DIFF. Figure 17 contains the definition and an illustration that slightly overloads our graphical language to illustrate the computation of the draggable local connected component. We take Proposition 3.10 as verifying that we have implemented faithfully the last non-constructive specification of the generator.

We may similarly enhance the pinch-rule to allow also the transfer of an initial parameter, as discussed in the previous section. The resulting generator is depicted and defined in Figure 18.

As a final remark in this section, a note on expressivity: Not unlike the encoding of κ into micro-κ [DL04], we may implement the lcc-function using vanilla reaction rules encoding the traversal performed in lcc. It essentially



DIFF($\mathbb{T}$):

$$\frac{\mathrm{lcc}(\mathbb{T}, B^+ \mid B^-) \equiv (\mathbb{T} \mid B^+, B^-) \qquad \delta = \downarrow\colon \mathrm{fn}(\mathbb{T} \mid B^+) = \emptyset}{(x)\,(A \mid \triangle_x \mid\mid \triangle_x \mid B^- \mid B^+ \mid \mathbb{T}) \;\rightarrow_\omega\; (x)\,(A \mid \triangle_x \mid \mathbb{T} \mid B^+ \mid\mid \triangle_x \mid B^-)}$$

Figure 17: The homogenous generator DIFF.

requires the addition of a single control for temporarily gathering the connected connect; and a number of extra rules for recognizing preconditions corresponding to the left-hand sides of the lcc-function. We find, however, that the higher abstraction-level gained by taking the lcc-function is sufficiently important to justify our choice. Also, as the function is inductively given, it lends itself directly to implementation.

Table 1 summarizes all the generators introduced in this section for membrane reconfiguration.

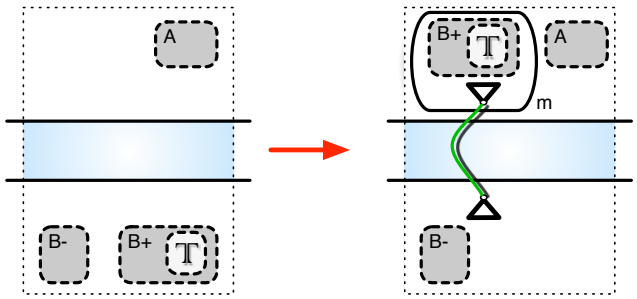
3.5 Refinement and Allowable Rules

In the previous sections we have introduced a series of generators all encapsulating a core biological action. And in Section 3.1, and Figures 9 and 14, we have introduced refinement formally. We now turning to defining refinement, formally.

The idea is, that we want to allow in $\mathcal{C}$-calculus any rule that is a suitable refinement of (a projection of) one of the generators for membrane actions or domain-level rules given in the previous section. All (projected) generators are certainly valid rules by themselves. However, we want to allow a biochemist to specify that, for instance, a vesicle may only TOUCH the plasma membrane, if a protein pro attached to its surface is bound to a receptor rec on the plasma membrane. Let is work out this example in detail.

From the informal specification above, we want a rule that might look like this:

$$\begin{array}{rcrl} r_0 & = & \mathsf{pro0}^x_a \mid \mathsf{ves}[& \mathsf{pro1}_a \mid A] \mid \mathsf{rec0}^x_b \mid \mathsf{plasma}[\ \mathsf{rec1}_b \mid B] \rightarrow \\ & & (y)\,\mathsf{pro0}^x_a \mid \mathsf{ves}[\triangle_y \mid & \mathsf{pro1}_a \mid A] \mid \mathsf{rec0}^x_b \mid \mathsf{plasma}[\triangle_y \mid \mathsf{rec1}_b \mid B], \end{array}$$



$$\textsc{pinch}(\mathbb{T}):\ \frac{\mathrm{lcc}(\mathbb{T}, B^+ \mid B^-) \equiv (\mathbb{T} \mid B^+, B^-) \qquad \delta = \downarrow : \mathrm{fn}(\mathbb{T} \mid B^+) = \emptyset}{A \,||\, B^- \mid B^+ \mid \mathbb{T} \ \rightarrow_\eta \ (x)(A \mid \mathsf{m}[\mathbb{T} \mid B^+ \mid G_x] \,||\, B^- \mid \triangle_x)}$$

Figure 18: The heterogenous generator for parametric PINCH.

where, for effect, we have aligned counterparts on the left- and right-hand sides. Let is spell out precisely in what sense r_0 is a refinement of the lateral projection of TOUCH.

First, the lateral projection of TOUCH, selecting the appropriate kinds of membranes, yields the following rule:

$$\mathsf{ves}[A] \mid \mathsf{plasma}[B] \rightarrow (y)\, \mathsf{ves}[\triangle_y \mid A] \mid \mathsf{plasma}[\triangle_y \mid B].$$

Next, in refining a generated rule, we want to allow refinement of the parameters; in practice, by substituting for any variable X, a solution S (that might in turn contain variables). We call this *inner* refinement of X. Refining the parameters A and B in the lateral projection of TOUCH, we get:

$$\mathsf{ves}[\mathsf{pro1}_a \mid A] \mid \mathsf{plasma}[\mathsf{rec1}_b \mid B] \rightarrow (y)\, \mathsf{ves}[\triangle_y \mid \mathsf{pro1}_a \mid A] \mid \mathsf{plasma}[\triangle_y \mid \mathsf{rec1}_b \mid B]$$

And finally, we want to allow *outer* refinement of a reaction rule by specifying parts of surrounding context, where the reaction occurs. In practice, for a rule $\mathsf{L} \rightarrow \mathsf{R}$, we want to allow any rule $\mathsf{L} \mid \mathsf{S} \rightarrow \mathsf{R} \mid \mathsf{S}$. By contextualizing the rule above with $\mathsf{pro0}_a^x \mid \mathsf{rec0}_b^x$, we get r_0.

We may sum up refinement conveniently with the help of substitution. First however, for full generality, we also need to consider rules with side-conditions.

Consider the following (somewhat artificial) rule and side-condition:[12]

$$r_1 = A \rightarrow \mathsf{p} \mid A, \ \varphi = \text{"}A \text{ does not contain a } \mathsf{p}.\text{"}$$

If we perform inner refinement on A to get

$$r_1' = \mathsf{p} \mid A \rightarrow \mathsf{p} \mid \mathsf{p} \mid A, \ \varphi' = ?,$$

[12]In writing side-conditions for reaction rules, it is convenient to overload the usage of variable-names in the rule, such as A, to refer to the solution that instantiates the variable.

Homogenous generators

$$\textsc{part} \qquad (x)(\triangle_x \mid A \mid\mid \triangle_x \mid B) \quad \to_\omega \quad A \mid\mid B$$

$$\textsc{touch} \qquad A \mid\mid B \quad \to_\omega \quad (x)(\triangle_x \mid A \mid\mid \triangle_x \mid B)$$

$$\textsc{diff}(\mathbb{T}) \qquad \frac{\mathrm{lcc}(\mathbb{T}, B^+ \mid B^-) \equiv (\mathbb{T} \mid B^+, B^-) \qquad \delta =\downarrow\colon \mathrm{fn}(\mathbb{T} \mid B^+) = \emptyset}{(x)\,(A \mid \triangle_x \mid\mid \triangle_x \mid B^+ \mid B^- \mid \mathbb{T}) \;\to_\omega\; (x)\,(A \mid \triangle_x \mid \mathbb{T} \mid B^+ \mid\mid \triangle_x \mid B^-)}$$

Heterogenous generators

$$\textsc{merge} \qquad (x)(A \mid \mathsf{m}[\triangle_x \mid C] \mid\mid \triangle_x \mid B) \quad \to_\eta \quad A \mid\mid C \mid B$$

$$\textsc{pinch}(\mathbb{T}) \qquad \frac{\mathrm{lcc}(\mathbb{T}, B^+ \mid B^-) \equiv (\mathbb{T} \mid B^+, B^-) \qquad \delta =\downarrow\colon \mathrm{fn}(\mathbb{T} \mid B^+) = \emptyset}{A \mid\mid B^+ \mid B^- \mid \mathbb{T} \;\to_\eta\; (x)(A \mid \mathsf{m}[\mathbb{T} \mid B^+ \mid G_x] \mid\mid B^- \mid \triangle_x)}$$

for any $\mathbb{T} \equiv (\tilde{y})(\mathsf{d}_{b_1}^{x_1} \mid \cdots \mid \mathsf{d}_{b_k}^{x_k})$ and for lcc as given in Def. 3.9.

Table 1: Summary of the generators for membrane reconfiguration.

what should the side-condition φ' say? We elect that A should be refined in the side-condition, as well. This yields: $\varphi' =$ "$(\mathsf{p} \mid A)$ does not contain a p." This in turn yields a valid rule (that, however, never applies, since the side-condition is equal to falsity). In other words, when substituting a parameter X for T, we also need to substitute that parameter in φ. We finally settle on the following definition for refinement:

Definition 3.11 (refinement)**.** For any rule $(\mathsf{L} \to \mathsf{R}, \varphi)$, any solution context S with one hole, and any group of solutions G, s.t., the number of solutions in G is equal to the number of variables in L and R, a refinement of this rule is $(\mathsf{S} \triangleleft \mathsf{L} \triangleleft \mathsf{G} \to \mathsf{S} \triangleleft \mathsf{R} \triangleleft \mathsf{G}, \varphi')$. The side-condition φ' is true for a group of parameters F if $\varphi(\mathsf{G} \triangleleft \mathsf{F})$ holds.

We call the contexts S and G the outer and inner refinement, respectively. Together, we call them the *application conditions* used to refine the rule $\mathsf{L} \to \mathsf{R}$.

In our case, for the generators in Table 1, the only parameters mentioned in side-conditions are the B-parameters in PINCH and DIFF. In practice, for those rules, to ensure that refined rules not be unappliable, do not refine B^- to include a domain that is connected to $\mathbb{T}$, and do not refine B^+ to include a domain that is *not* connected to $\mathbb{T}$.

Our choice to refine also side-conditions ensures us that application conditions only restrict the applicability of rules. In particular, we easily verify the following convenient little lemma.

Lemma 3.12 (application conditions restrict rules)**.** *Suppose $r' = (\mathsf{L}' \to \mathsf{R}', \varphi')$ is a refinement of $r = (\mathsf{L}, \mathsf{R}, \varphi)$ as defined in Definition 3.11. Then, if $\mathsf{T} \to \mathsf{T}'$ by r', then also $\mathsf{T} \to \mathsf{T}'$ by r.*

Suppose, we had *not* refined side-conditions also; then this lemma would not hold. (To see this, consider the reaction $\mathsf{p} \to \mathsf{p} \mid \mathsf{p}$ against the example rule r_1 and against the refined rule r_1' *without* refining also the application condition.)

Having treated the technicalities concerning refinement, we can sum up the allowable rules.

Definition 3.13 (allowable rules)**.** The allowable rules in the $\mathcal{C}$-calculus are refinements of

- domain-level rules, as defined in Definition 3.5; and,
- projections of the generators in Table 1.

In Section 4, we give several concrete examples of rules.

3.6 Preservation of Well-formedness

In this section, we establish the promised property that any reaction due to an allowable rule preserves well-formedness.

The main theorem is the following. We devote the rest of this section to prove this theorem.

Theorem 3.14 ($\mathcal{C}$-calculus reactive systems preserve well-formedness)**.** *If* S *is well-formed, and* $\mathsf{S} \to \mathsf{T}$*, then* T *is well-formed.*

Proof. Follows from Lemmas 3.12, 3.15, 3.16, and 3.17. □

All well-formedness conditions stipulate conditions on links—on the presence, arity, or sorting of links or on the locality of the ports they connect. That fact, and the nature of allowable $\mathcal{C}$-calculus rules, makes graph-based reasoning on solutions most convenient. As explained in Section 2.4, solutions correspond to certain bigraphs. Further, as recorded in [DK08, Theorem 4.5], reaction for any $\mathcal{B}^{\Sigma,\mathcal{R}}$-calculus corresponds 1-1 to bigraphical reaction (under so-called non-aliasing contexts). Hence, due to the founding of $\mathcal{C}$-calculus on bigraphs, we can directly apply (bi)graphical reasoning to verify the following propositions.

We need to check the change(s) induced by a reaction rule against each of the well-formedness conditions.

In outline, we proceed as follows: Most allowable rules, specifically domain-level rules and those generated from TOUCH, PART, and BUD create or break links (plus some domains or membranes) contained entirely inside the left-hand side or right-hand side of a rule. Consequentially, it is easily checked that they cannot break well-formedness. We start by stating this in Lemmas 3.15 and 3.16. Those rules that allow bunched transportation of subsolutions, contained in parameters of the rules, need a bit more care. We check that rules generated from MERGE are innocuous; and, for the rules generated from DIFF and PINCH we check that our development in Section 3.4.3 has been sound. We handle these three transport rules in Lemma 3.17. Together the three lemmas imply that all reactions due to allowable rules preserve well-formedness, as stated in Theorem 3.14.

A simple, but convenient corollary of Lemma 3.12 is that any reactive system $\mathcal{T}'$ over refined rules will be a sub-system of a reactive system $\mathcal{T}$ over the corresponding unrefined rules. It follows that, if we prove preservation of well-formedness for reactive systems over core rules without refinements, i.e., domain-level rules and the projections of the generators in Table 1 without any application conditions, then, in particular, any reactive systems using refined rules also preserves well-formedness. Hence in the following lemmas, we shall simply check unrefined rules.

Lemma 3.15 (domain-level rules preserve well-formedness)**.** *If* S *is well-formed and* $\mathsf{S} \to \mathsf{S}'$ *by a domain-level rule, then* S' *is well-formed.*

Proof. Below we sketch the proof in some detail. The proof is essentially a straight-forward case-analysis of a number of cases corresponding to the kinds of links in the bigraphs that correspond to well-formed solutions. We thus rely on bigraphical reasoning. In this paper, we have not introduced bigraphical theory in full; however, we refer the reader to the informal summary given in Section 2.4.

From Definition 3.5 we know that all domain-level rules $\mathsf{L} \to \mathsf{R}$ incorporate one change (for the base rules) or two changes. Those changes translate directly to changes which relate the (bi)graphs S to S'.

Let us consider first, what occurs under SYNTH and DEGRADE.

- SYNTH All domains affected by this reaction have just been created in S and are unbound. The only created link is the protein backbone, and it is explicitly required to respect the well-formedness condition (*fixed backbone*), together with the created domains it links.

- DEGRADE No complexation-links are created or deleted by this reaction. A backbone link is deleted, and it is explicitly required that every domain in a protein is deleted together with it (ensuring us that (*fixed backbone*) is upheld).

Now, we analyze each of the base rules, which change existing or newly created domains or links, for their effect on a domain in S and consider well-formedness.

- BIND Suppose d in S is affected by this reaction. Then it is visible and will be bound to a co-located d' in S'. In effect, S' will have an added link compared to S. We check this link against the well-formedness conditions; the three interesting conditions are (*binary complex*), (*link sorting*), and (*local complex*). They are all upheld, as it is explicitly required that the link be between two complexation-ports of co-located domains. The remaining conditions are voidly upheld.

- BREAK Suppose d in S is affected by this reaction. Then it is bound, and will be visible in S' (as well as another d' in S). In effect, S' will have lost a link compared to S. Checking against the well-formedness conditions, it easily seen that no conditions are violated. In particular, note that

since we explicitly require the entire link to be matched and deleted in BREAK, no faulty unary links can be created (which would violate (*binary complex*)).

- HIDE/SHOW These reactions change no links, making them non-problematic with regard to well-formedness.

Finally, from the check of HIDE/SHOW, it easily seen that the combinations of BIND and BREAK with one of HIDE and SHOW (i.e., BIND+SHOW, BIND+HIDE, BREAK+SHOW, and, BREAK+HIDE) also preserve well-formedness.

In conclusion, no domain-level reaction rule may induce a change in S, which breaks well-formedness in S'. □

We turn to checking the generators described in Section 3.4 and the changes they induce. We start by considering projections of the rules that do not involve transport.

Lemma 3.16 (non-transport generators preserve well-formedness)**.** *If* S *is well-formed and* $\mathsf{S} \to \mathsf{S}'$ *by a rule generated from* TOUCH*,* PART*, or* BUD*, then* S' *is well-formed.*

Proof. We sketch the proof.

Any reaction rule generated from TOUCH and PART induces exactly the following changes on S: The addition or deletion of two gates as well as the creation or deletion of a link between them. By definition, the homogenous projections will ensure us that these changes occur in compartments separated by exactly two membranes, in turn ensuring us that any such channel respects (*bitonality*). It is immediate from the definition of the generators that the conditions (*link sorting*) and (*binary channels*) are upheld.

Any reaction rule generated from BUD adds exactly one membrane as well as two connected gates (as for TOUCH above). The heterogenous projection in combination with the created membrane ensures that the gates respect (*bitonality*). (Again, it is immediate that the conditions (*link sorting*) and (*binary channels*) are upheld.) □

It remains for us to consider those rules, which move subtrees—captured in parameters in the rules—in the solution. To illustrate the reasoning in the following proof, we shall sketch the parts of the tree corresponding to the solution, where the reaction occurs.

We are reasoning about rules, which stem from generators projected to form two or three rules. To concisely illustrate and reason about all projections in one go, we may illustrate the subtrees corresponding to generators in the manner depicted for heterogenous projection in Figure 19 and for homogenous projection in Figure 20. Below, we refer to such illustrations as *projective*. At the top of the figures, projective groups are illustrate as in Section 3.2.1. When we consider solutions as trees whose parenthood-relation is given by membrane-containment, this corresponds to stating that one or two parenthood relations

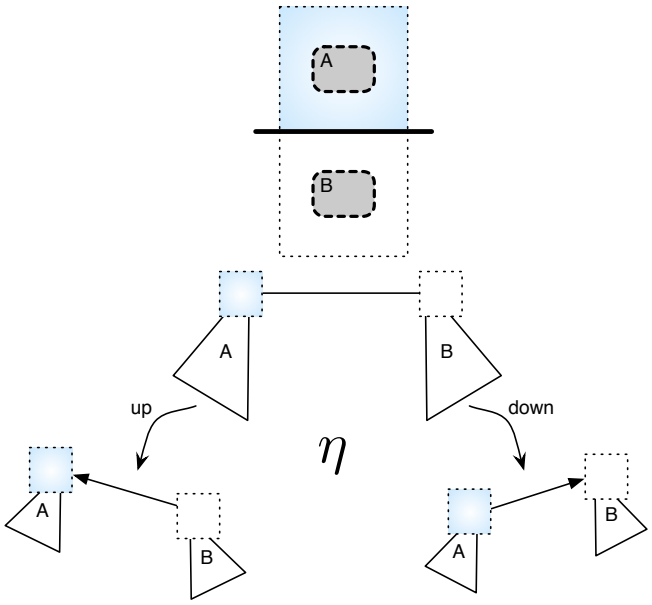


Figure 19: Depicting heterogeneous generators projectively, using unoriented parenthood-edges.

(for heterogenous and homogenous, respectively) are unspecified. We can conveniently illustrate that by leaving one or two parenthood-edges in the tree corresponding to a solution be unoriented. We depict that in the middle of Figures 19 and 20. At the bottom, we illustrate how the projections work to produce the two or three trees corresponding to the projective illustrations.

Lemma 3.17 (transport generators preserve well-formedness)**.** *If* S *is well-formed and* $\mathsf{S} \to \mathsf{S}'$ *by a rule generated from* MERGE*,* DIFF*, or* PINCH*, then* S' *is well-formed.*

Proof. Below we sketch and illustrate the proof in some detail. As for the proofs above, the proof is a case-analysis of a number of cases for the kinds of links in bigraphs corresponding to well-formed solution. We thus rely on bigraphical reasoning; we refer the reader to the intuition given in Section 2.4.

We consider the generators MERGE and PINCH, and discuss the links that we need to check with the help of projective illustrations as explained above. We elide a detailed sketch of the analysis for the case for DIFF, as the verification is similar to that for PINCH.

MERGE: A membrane and two connected channels are deleted; and the contents of the membrane (captured in the parameter C) is transfered to the other end of the channel. The deletion of the connected channels correspond to the action captured in PART, and we need not consider them further.

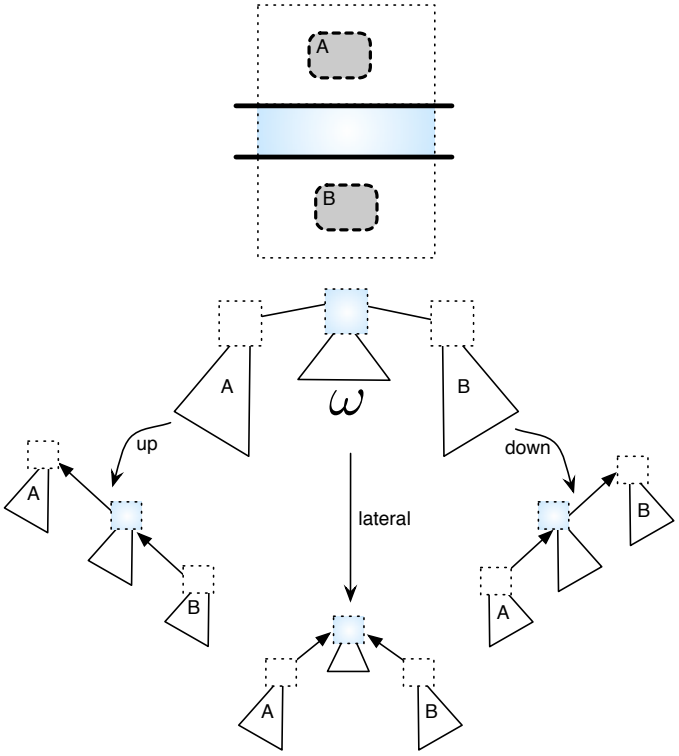
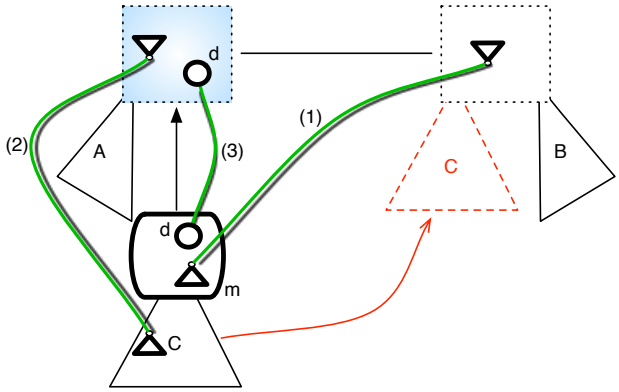


Figure 20: Depicting homogenous generators projectively, using unoriented parenthood-edges.

The projective illustration below illustrates the generator, and highlights representatives of links, which we need to consider.



On the bottom-left is the subtree corresponding to C. The arrow pointing to the right and the dotted subtree on the right indicates the transport that C will undergo as a consequence of MERGE. C is inside a membrane (to illustrate, we indicate this with a membrane M at the top of subtree illustrating C), which is inside a compartment, which contains another subtree (A). This compartment either has a parent or child (depending on the projection) containing the subtree B.

Those links that may prove problematic are those spanning regions, and it is clear that we need only consider those connected to nodes inside C. On the illustration, we have drawn representatives of each kind of link, that we need to consider.

Channels, whose gates lie within C, may come in two versions:

- Well-formed channels that are top-level in C may be connected to the region that C is transported into. Such an example is illustrated and tagged with (1) above. As a consequence of the transport such channels will be degenerate (i.e., their length will be 0), but well-formed in S'.

- Well-formed channels that are inside a top-level membrane in C may only be connected to the parent of C. We have illustrated and tagged such a link with (2) above. As can be seen from the illustration, after the transport of C, the channel will still span two membranes.

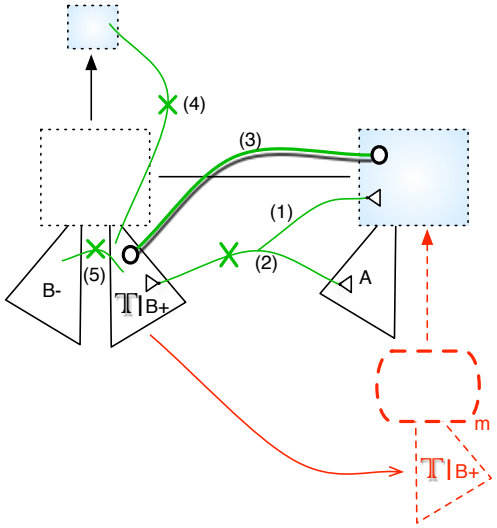
Every other well-formed channel is either contained entirely within or outside C.

The link tagged with (3) illustrates a protein backbone that crosses the membrane that is destroyed as a consequence of MERGE. Checking, we see that this backbone also preserves its well-formed after the transfer.

PINCH: A membrane and two connected channels are created, and the cargo $\mathbb{T} \mid B^+$ is transfered to the compartment inside the new membrane. It is simple

to check that the creation of the membrane itself and the connected channels respect well-formedness. We concern ourselves with the links, whose endpoints change location.

We illustrate the generator projectively, and again we highlight representatives of links that we need to consider more carefully.



On the middle-left, we illustrate with a dotted region the compartment containing the user-parameter $\mathbb{T}$ and the parameter, B^+, it is dragging along. The parameter that remains, B^-, is a sibling. The compartment, that the new membrane is created in, is a parent or a child (depending on the projection) of the compartment that $\mathbb{T} \mid B^+$ stems from. We illustrate the transfer with an arrow, and on the bottom-right, we draw the newly created membrane and the transferred $\mathbb{T} \mid B^+$ with dotted lines.

The parent-compartment of the new membrane (on the top-right) has a subtree A. Finally, in case the projection is downwards (i.e., in case the undirected parent-edge is oriented from right to left), then we need to consider also links to the parent-region of the compartment containing $\mathbb{T}$, B^+, and B^- (illustrated on the top-left).

As for MERGE, we need concern ourselves mainly with links spanning compartments. However, we should remark that this is only because we have already verified (in Proposition 3.10) that the split of B into B^+ and B^- by lcc, ensures that there are no links, channels or backbones, between $\mathbb{T} \mid B^+$ and B^- (we illustrate such a link above with a crossed link and tag it with (5)).

Also, the side-condition $\mathrm{fn}(\mathbb{T} \mid B^+) = \emptyset$ given for the downwards projection, ensures that there can be no links to a parent region (we illustrate such a link with a crossed link tagged with (4)).

Having treated the links that the side-conditions prevent, we turn to channels captured partially in $\mathbb{T} \mid B^+$. However, lcc ensures ensures that *no* gates may be

captured in $\mathbb{T} \mid B^+$. Hence, the two possibilities for channels (illustrated with crossed lines and tagged with (1) and (2) above), are prevented. It remains again to consider backbones that span the projected membrane, such as the link tagged (3) above. After the PINCH they will instead span the newly created membrane—having followed the pivot-motion illustrated and discussed in Section 3.4.3—and will still be well-formed.

DIFF: The analysis is similar to that for PINCH. □

4 Examples

Now for some examples.

4.1 G-protein Coupled Receptor

We start by developing a model of cross-membrane signal transduction (i.e., propagation of signals across say a cell wall in a signalling pathway) via *G-protein coupled receptor proteins* (GCPRs). A GCPR is a kind of receptor belonging to a large family of certain transmembrane proteins—proteins integral to the cell wall that have protrusions on both sides. The family of GCPRs is large and diverse, but they all employ a similar technique to implement a mechanism that allows extracellular signals to cross the plasma membrane. In the following description and model of GCPRs we shall abstract away from many details to develop a model illustrating some of the central properties shared by GCPRs; in any case, the answers to many questions pertaining to GCPRs are not known.

The extracellular part of a GCPR, illustrated in Figure 21, constitutes a receptor site that is shaped to be able to bind certain *ligands* (signalling molecules). Its inner part is bound to a partner G-protein (a guanine nucleotide-binding protein). The G-protein in its inactive state is actually a little complex of three different protein subunits bound to each other: G_α, G_β, and G_γ.[13] The β and γ subunits are bound tightly together (together they are called the $G_{\beta\gamma}$-complex), while the α complex is relased as part of the activation of the GCPR. Particular to GCPRs are that they do not pass any substance through the membrane, but rely on structural changes to the folding shape of the receptor. The folding shape of a protein is called its *conformation*, hence such structural changes are known as conformational changes.

Loosely, the GCPR propagates a signal in the following manner: A ligand is bound to the receptor site activating the GCPR by changing the conformation of the receptor in such a way that the G-protein is activated, causing the G_α subunit to bind to a molecule of GTP (guanosine triphosphate). This in turn causes it to release itself from the rest of the G-protein, as illustrated in Figure 22. The remaining $G_{\beta\gamma}$-complex also detaches from the receptor. The release of the G_α subunit has exposed sites on the $G_{\beta\gamma}$-complex that may interact with other molecules that serve as effectors for further propagation of

[13] For simplicity, we restrict to treating here heterotrimeric G proteins, also called "large" G-proteins.

the signal. A receptor in an activated state may bind to other G-proteins and activate them. Depending on the stability of the ligand-receptor complex, the ligand is released at some point. At this point an inactive G-protein may again bind to it to return the entire complex to the initial inactive state.

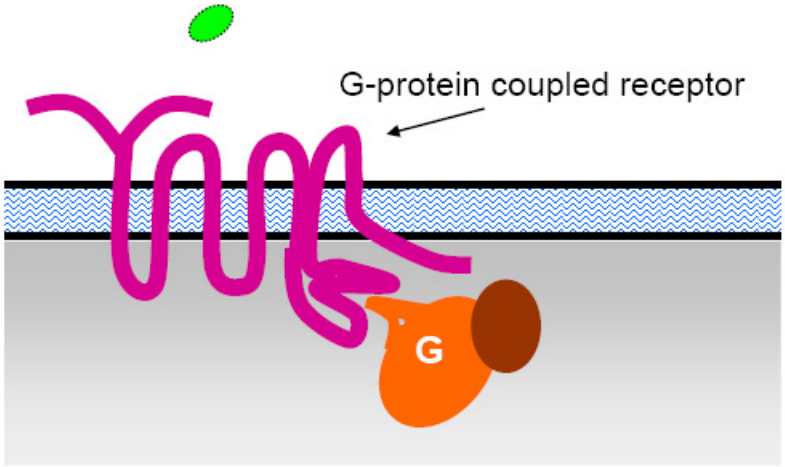


Figure 21: A G-protein receptor before binding to a ligand (*illustration from* [Coh07]).

A $\mathcal{C}$-calculus model We work with the following protein signature:

$$\{\mathsf{lig} : (1,0), \mathsf{re} : (2,1), \mathsf{gbc} : (3,0), \mathsf{ga} : (2,0), \mathsf{gtp} : (1,0), \mathsf{eff} : (1,0)\},$$

and a single membrane type plasma to model plasma membranes.

In Figure 23, we give seven rules that encapsulates a model of the core reactions involved in the firing of a G-protein coupled receptor. We explain the rules one-by-one, below.

- Rule 0: This rule allows a ligand lig to bind to the extra-cellular domain of the receptor, re0.
- Rule 1: The rule models the conformational change of the receptor induced by the ligand binding in rule 0, by activating the G_α subunit, specifically the domain ga1, able to bind to the GTP molecule, gtp. We make rule 1 η-projective, i.e., we intend that it may be applied if the two solutions in the groups in the left and right hand side are separated by exactly one membrane. In this manner, we capture conveniently and consisely the fact that these transmembrane receptors transfer signals across the membrane, independent of the orientation of the membranes.

  We should note, that though this serves as a nice illustration of projectivity, in nature the particular family of transmembrane receptors we have

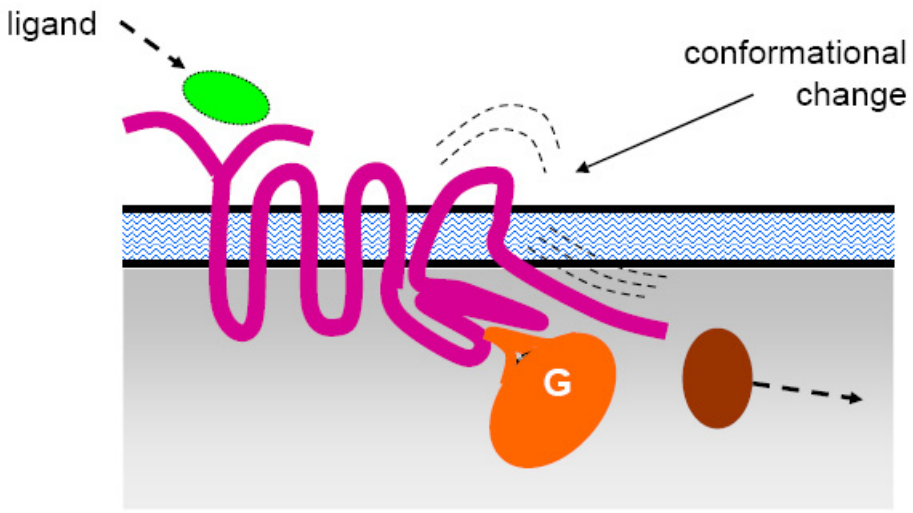


Figure 22: A G-protein receptor firing due to a conformational change, after binding to a ligand (*illustration from* [Coh07]).

chosen, GCPRs, work only in the plasma membrane, working to propagate extra-cellular signals inwards.Note also, that we disassociate the ligand-binding event with the conformational change.

- Rule 2: In this rule the domain ga1 of the G_α subunit binds to the gtp.
- Rule 3: This rule allows the G_α subunit to break the assocation to the G-protein leaving the $G_{\beta\gamma}$-complex bound to the receptor. The dissassociation reveals (and activates) a part of the $G_{\beta\gamma}$-complex hidden by the G_α subunit. We model that by the activation of the domain gbc1. Note that rules 2 and 3 are not causally dependent. The ga1 may bind to gtp before or after the ga0 domain has broken the bond to gbc2 domain. In other words, we model that the G_α subunit may bind to the GTP molecule independently of the dissassocation from the $G_{\beta\gamma}$-complex.
- Rule 4: After the activation of gbc1, the $G_{\beta\gamma}$-complex may release itself from the receptor to seek out an effector to propagate the signal further. Rule 4 captures this breakage of the complexation-link between the internal receptor-domain, re1, and the domain of $G_{\beta\gamma}$, gbc0.
- Rule 5: And in rule 5 the $G_{\beta\gamma}$-complex, represented by the activated gbc1-domain, binds to some effector-domain—here abstractly represented as an eff-molecule. Note that for variation, here we have made the formation of the bond between the effector and the $G_{\beta\gamma}$-complex dependent of the breakage of the bond between the receptor and the $G_{\beta\gamma}$-complex.

$$
\begin{array}{ll}
\mathrm{r}_0 & \mathsf{lig}_l \mid \mathsf{re0}_r \;\to\; (x)\,\mathsf{lig}_l^x \mid \mathsf{re0}_r^x \\
\mathrm{r}_1 & (\mathsf{lig}_l^x \mid \mathsf{re0}_r^x) \mid\mid \left(\mathsf{re1}_r^y \mid \mathsf{gbc0}_b^y \mid \mathsf{gbc2}_b^z \mid \mathsf{ga0}_a^z \mid \overline{\mathsf{ga1}}_a\right) \to_\eta \\
 & (\mathsf{lig}_l^x \mid \mathsf{re0}_r^x) \mid\mid (\mathsf{re1}_r^y \mid \mathsf{gbc0}_b^y \mid \mathsf{gbc2}_b^z \mid \mathsf{ga0}_a^z \mid \mathsf{ga1}_a) \\
\mathrm{r}_2 & \mathsf{ga1}_a \mid \mathsf{gtp}_g \;\to\; (u)\,\mathsf{ga1}_a^u \mid \mathsf{gtp}_g^u \\
\mathrm{r}_3 & (z)\,\overline{\mathsf{gbc1}}_b \mid \mathsf{gbc2}_b^z \mid \mathsf{ga0}_a^z \mid \mathsf{ga1}_a^u \mid \mathsf{gtp}_g^u \;\to \\
 & \mathsf{gbc1}_b \mid \mathsf{gbc2}_b \mid \mathsf{ga0}_a \mid \mathsf{ga1}_a^u \mid \mathsf{gtp}_g^u \\
\mathrm{r}_4 & (x)\,\mathsf{re1}_r^x \mid \mathsf{gbc0}_b^x \mid \mathsf{gbc1}_b \;\to\; \mathsf{re1}_r \mid \mathsf{gbc0}_b \mid \mathsf{gbc1}_b \\
\mathrm{r}_5 & \mathsf{gbc1}_b \mid \mathsf{eff}_e \;\to\; (y)\,\mathsf{gbc1}_b^y \mid \mathsf{eff}_e^y \\
\mathrm{r}_6 & \mathsf{re1}_r \mid \mathsf{gbc0}_b \mid \overline{\mathsf{gbc1}}_b \mid \mathsf{gbc2}_b^z \mid \mathsf{ga0}_a^z \mid \overline{\mathsf{ga1}}_a \;\to \\
 & (x)\,\mathsf{re1}_r^x \mid \mathsf{gbc0}_b^x \mid \overline{\mathsf{gbc1}}_b \mid \mathsf{gbc2}_b^z \mid \mathsf{ga0}_a^z \mid \overline{\mathsf{ga1}}_a
\end{array}
$$

Figure 23: Rules for modelling a G-protein coupled receptor

- Rule 6: Finally, rule 6 captures that a receptor with no G-protein may bind to another (inactive) G-protein.

4.2 Clathrin-dependent Endocytosis

For an example involving both formation and fusion of compartments, we turn to one of the celebrities in the world of research organisms, the illustrious nematode worm *Caenorhabditis elegans* (or *C. elegans* among friends). It exists in spades in common dirt, and is a multicellular eukaryote, with a small enough amount of cells (959 in adult hermaphrodites, 1031 in adult males) to have been extensively researched. This has made it possible to document in great detail the developemental lineage of every cell in the lifespan of a single organism. A key property for the tractability of such a study has also been that unlike mammals, for instance, in *C. elegans* the fate of cells are largely invariant between individual worms. The online compendium WormBook [WB] documents all aspects of research into these wondrous creatures.

We shall focus on endocytosis—the process by which cells internalize various extracellular material—as studied for *C. elegans*. The best studied endocytic pathway for both worms and mammals is the receptor-mediated endocytosis by formation of clathrin-coated vesicles. We base the following high-level presentation on the excellent review by Grant and Sato [GS06, Section 1] (available in the WormBook). The schematic illustration of clathrin-dependent endocytosis in Figure 4.2 we also borrow from *loc. cit.*

We may describe the process of clathrin-dependent endocytosis as follows: On the cell wall receptor proteins await, ready to bind to ligands, cargo molecules that the cell wishes to internalize. The purpose of the uptake of such cargo molecules may be diverse, and may be part of signalling pathways, but may also, for instance, involve the uptake of molecules that the cell needs—such

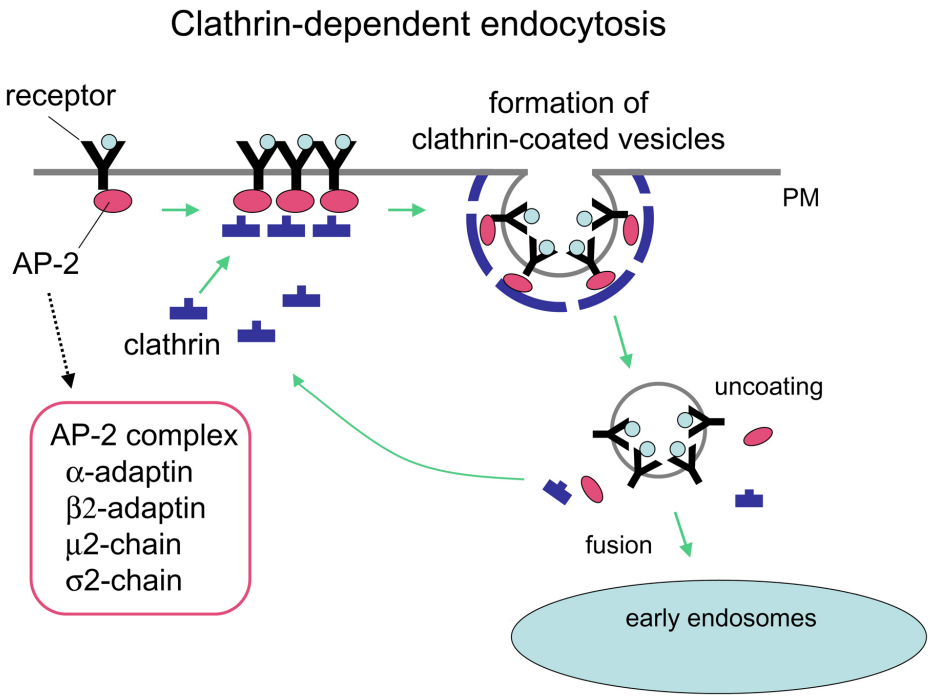


Figure 24: Mechanism of clathrin-dependent endocytosis. Clathrin and cargo molecules are assembled into clathrin-coated pits on the plasma membrane together with an adaptor complex called AP-2 that links clathrin with transmembrane receptors, concluding in the formation of mature clathrin-coated vesicles (CCVs). CCVs are then actively uncoated and transported to early/sorting endosomes (*illustration and caption from* [GS06]).

pathways inevitably lead the digested particles to degradation (i.e., destruction) in the lysosomes (with the suitable nickname, the "suicide sacs"). Over time, it so happens that clusters of receptors bound to ligands tend to form on the plasma membrane. The intra-cellular part of the receptors bind to adaptor complexes called AP2, who in turn bind to so-called clathrin lattices. The macro-shape of clusters of clathrin is thought to provide the down-force required to curve the plasma membrane into small inward buds. These inward buds are mechanically constricted and snipped off to form clathrin-coated vesicles floating freely in the cytoplasm. These coated vesicles are subsequently uncoated actively (with the help of chaperone protein hsc-70 and its compatriot the co-chaperone auxillin). With the aid of another GTPase, Rab5, these uncoated vesicles now fuse with each other and with so-called early endosomes, which serve a sorting purpose. These endosomes have a slightly reduced pH inside their membranes, which tends to cause the receptors to let go of their ligands. From the endosomes most receptors (some still with ligands bound) now form

new vesicles and eventually recycle to the cell wall; while most ligands continue their transport—possibly through further vesicle transport towards their endgoal, say, lysosomes.

At this point we shall leave our protagonists, and turn to developing a $\mathcal{C}$-calculus model. We underline again that, though detailed, the description above is quite high-level. Most steps in the pathway involve intricate biochemical steps; to name one, for instance, the snipping of buds happens through the intervention of a GTPase dynamin—it works essentially by forming a coil around the invaginated bud and then gradually, through a process of GTP hydrolysis, mechanically constricts until the bud is closed off to form a coated vesicle. In our model, we shall aim at hitting the same level of abstraction as in the description above, focusing mostly on illustrating how $\mathcal{C}$-calculus allows us to model induced vesicle formation, and later fusion of such vesicles with each other and with endosomes.

A $\mathcal{C}$-calculus model We make certain simplifications to make the model manageable: We model the 4-subunit complex AP2 as a single protein with only the two domains necessary for the reactions in this example. Also, we do not model exactly how the chaperone and co-chaperone, hsc-70 and auxillin, or the GTPase Rab5 work; we only require that they be present for certain reaction to take place. (For the same reason, we shall elide the mention of any complexation-ports on these three latter proteins.)

We arrive at the following protein signature:

$$\{\mathsf{lig} : (1,0), \mathsf{re} : (2,1), \mathsf{ap} : (2,0), \mathsf{cla} : (1,0), \mathsf{hsc\text{-}70} : (0,0), \mathsf{aux} : (0,0), \mathsf{rab} : (0,0)\}.$$

We take three membrane types plasma, ves, and end; the latter two types model endosomes and vesicles.

In Figure 25, we give a set of rules to model clathrin-dependent endocytosis as described above. Again, we explain the rules one-by-one.

- Rules 0, 1, and 2: These rules allow a ligand to bind to the extra-cellular receptor domain, re0, (rule 0); allow the receptor-binding domain, ap0, of the adaptin AP2 to bind to the intra-cellular receptor domain re1, (rule 1) inducing a conformational change that allows clathrin, cla, to bind to the adaptin domain ap1 (in rule 2).

- Rule 3: Rule 3 is a refinement of PINCH. (Since there is no need to repeat the generic side-condition for PINCH or DIFF, we just write by the rule $\varphi_{\text{PINCH}}(\mathbb{T} = \ldots)$ to remind that the rule has a side-condition.)

  We take the downwards projection of to model the clathrin-coating and forming of a vesicle inside a cell. The rule allows creations of buds of vesicle-type, ves[...]. We have refined the vanilla PINCH-rule to require at least one receptor coated with clathrin to be present for the bud to form, by setting $\mathbb{T} = \mathsf{re0}^x_r$, requiring the extra-cellular part of the receptor to be bound (to a ligand), and giving application conditions inside the plasma

r_0 $\quad \mathsf{lig}_l \mid \mathsf{re0}_r \to (x)\, \mathsf{lig}_l^x \mid \mathsf{re0}_r^x$

r_1 $\quad \mathsf{re1}_r \mid \mathsf{ap0}_a \mid \overline{\mathsf{ap1}}_a \to (x)\, \mathsf{re1}_r^x \mid \mathsf{ap0}_a^x \mid \mathsf{ap1}_a$

r_2 $\quad \mathsf{ap1}_a \mid \mathsf{cla}_c \to (x)\, \mathsf{ap1}_a^x \mid \mathsf{cla}_c^x$

r_3 $\quad B^- \mid B^+ \mid \mathsf{re0}_r^x \mid \mathsf{plasma}[\mathsf{re1}_r^y \mid \mathsf{ap0}_a^y \mid \mathsf{ap1}_a^z \mid \mathsf{cla}_c^z] \to$
$(g)\, \triangle_g \mid B^- \mid \mathsf{plasma}[\mathsf{ves}[\triangle_g \mid B^+ \mid \mathsf{re0}_r^x] \mid$
$\mathsf{re1}_r^y \mid \mathsf{ap0}_a^y \mid \mathsf{ap1}_a^z \mid \mathsf{cla}_c^z],\ \varphi_{\mathrm{PINCH}}(\mathbb{T} = \mathsf{re0}_r^x)$

r_4 $\quad (g)\ (\, B^- \mid B^+ \mid \mathsf{lig}_l^x \mid \triangle_g) \mid\mid (\triangle_g \mid A) \to_\omega$
$(g)\ (\, B^- \mid \triangle_g) \mid\mid (\triangle_g \mid B^+ \mid \mathsf{lig}_l^x \mid A)\,, \qquad \varphi_{\mathrm{DIFF}}(\mathbb{T} = \mathsf{lig}_l^x)$

r_5 $\quad (g)\, \triangle_g \mid \mathsf{plasma}[A \mid \mathsf{re1}_r^y \mid \mathsf{ap0}_a^y \mid \mathsf{ap1}_a^z \mid \mathsf{cla}_c^z \mid \mathsf{re1}_s^v \mid$
$\mathsf{ap0}_b^v \mid \mathsf{ap1}_b^w \mid \mathsf{cla}_d^w \mid \mathsf{ves}[\triangle_g \mid \mathsf{re0}_r^x \mid \mathsf{re0}_s^u \mid B]\,] \to$
$\mathsf{plasma}[A \mid \mathsf{re1}_r^y \mid \mathsf{ap0}_a^y \mid \mathsf{ap1}_a^z \mid \mathsf{cla}_c^z \mid \mathsf{re1}_s^v \mid$
$\mathsf{ap0}_b^v \mid \mathsf{ap1}_b^w \mid \mathsf{cla}_d^w \mid \mathsf{ves}[\mathsf{re0}_r^x \mid \mathsf{re0}_s^u \mid B]\,]$

$\mathrm{r}_{6\mathrm{a}}$ $\quad (y, z)\, \mathsf{hsc\text{-}70}_h \mid \mathsf{aux}_b \mid \mathsf{re1}_r^y \mid \mathsf{ap0}_a^y \mid \mathsf{ap1}_a^z \mid \mathsf{cla}_c^z \to$
$(z)\, \mathsf{hsc\text{-}70}_h \mid \mathsf{aux}_b \mid \mathsf{re1}_r \mid \mathsf{ap0}_a \mid \mathsf{ap1}_a^z \mid \mathsf{cla}_c^z$

$\mathrm{r}_{6\mathrm{b}}$ $\quad (z)\, \mathsf{ap0}_a \mid \mathsf{ap1}_a^z \mid \mathsf{cla}_c^z \to \mathsf{ap0}_a \mid \mathsf{ap1}_a \mid \mathsf{cla}_c$

$\mathrm{r}_{7\mathrm{a}}$ $\quad \mathsf{rab}_r \mid \mathsf{ves}[\, A\,] \mid \mathsf{ves}[\, B\,] \to (g)\, \mathsf{rab}_r \mid \mathsf{ves}[\, \triangle_g \mid A\,] \mid \mathsf{ves}[\, \triangle_g \mid B\,]$

$\mathrm{r}_{7\mathrm{b}}$ $\quad \mathsf{rab}_r \mid \mathsf{ves}[\, A\,] \mid \mathsf{end}[\, B\,] \to (g)\, \mathsf{rab}_r \mid \mathsf{ves}[\, \triangle_g \mid A\,] \mid \mathsf{end}[\, \triangle_g \mid B\,]$

$\mathrm{r}_{8\mathrm{a}}$ $\quad (g)\, \mathsf{ves}[\, \triangle_g \mid A\,] \mid \mathsf{ves}[\, \triangle_g \mid B\,] \to \mathsf{ves}[\, A \mid B\,]$

$\mathrm{r}_{8\mathrm{b}}$ $\quad (g)\, \mathsf{ves}[\, \triangle_g \mid A\,] \mid \mathsf{end}[\, \triangle_g \mid B\,] \to \mathsf{end}[\, A \mid B\,]$

r_9 $\quad (x)\, \mathsf{end}[\, \mathsf{lig}_l^x \mid \mathsf{re0}_r^x\,] \to \mathsf{end}[\, \mathsf{lig}_l \mid \mathsf{re0}_r\,]$

Figure 25: Rules for modelling clathrin-dependent endocytosis

membrane to require the intra-cellular part to clathrin-coated. (For simplicity, we suppose that the downforce of a single clathrin-coating is enough to create a small bud.) The extra-cellular receptor-domain is pulled into the bud—dragging, in the surroundings, B^+, the bound ligand—and a channel is created.

- Rule 4: We refine the vanilla DIFF-generator to enable free diffusion of ligands in and out of the pinched bud. For variation, in this rule we just take every (homogenous) projection of the DIFF (indicated by keeping the ω-subscript on the reaction-arrow)—for every kind of membranes in the signature. This allows the rule to also allow diffusion among fused uncoated vesicles and endosomes later in the process—which we comment on below. Note, that should the ligand be bound to a receptor, then the receptor will be dragged into the bud (or out of the bud) as well as part of the surroundings, B^+. This may result in further or lessened clathrin-coating of a bud.

- Rule 5: Having pinched a bud via rule 3, rule 5 allows the bud to break off and part from the plasma membrane. Rule 5 is a straight refinement of the downwards projection of the PART-generator. For variation, we suppose, that the downforce of two clathrin-coatings is necessary to break off a bud. This necessitates the diffusion of more bound ligands into a formed bud—via rule 4.

- Rules 6a and 6b: These rules model, on a high level of abstraction, the uncoating of vesicles (as discussed above) in two steps. We simple require the presence of the two representatives of the chaperone hsc-70, hsc-70, and the co-chaperone auxillin, aux, for the uncoating reaction to occur.

- Rules 7a, 7b, 8a, and 8b: These rules allow the two-step procedure of TOUCH and MERGE, that results on fusion of uncoated vesicles with each other (rules 7a and 8a) and with endosomes-compartments, end[...] (rules 7b and 8b). For both fusions to occur, we require the GTPase Rab5, represented by rab, to be present for initiating the process. To illustrate the leverage we have in choosing refinements, for these rules, we take only the lateral projections of the TOUCH- and MERGE-rules; thus the reaction-arrows have no subscripts.

- Rule 9: Finally, rule 9 models that the cargo-ligand may be released when it has been enveloped by an endosome.

As a final remark, we may note, that one might also consider adding a rule to allow smaller clathrin-coated buds formed on the plasma membrane to merge, thus forming larger buds. This is probably a biologically viable alternative explanation to the formation of larger clusters of clathrin-coated receptors, resulting in larger buds.

5 Conclusion

We have presented and motivated in detail a novel calculus, the $\mathcal{C}$-calculus, for modelling low-level interaction inside and among cells, the basic building blocks of all known life. The focus is on two main actors of cells, proteins and membranes.

The calculus takes inspiration from several calculi, in particular, the κ-calculus and the brane-calculi. The extension of a κ-like calculus with dynamic compartments is a novel contribution, in itself. The $\mathcal{C}$-calculus also introduces a novel abstraction, *channels*, for modelling partially fused membranes, and uses these to give a novel treatment of transport of complexes through diffusion and parametric pinching of buds. Furthermore, as opposed to most process calculi the $\mathcal{C}$-calculus allows domain experts considerable freedom in instantiating a domain-specific sub-calculus for the study of a particular biological application.

A user develops a model in the $\mathcal{C}$-calculus by selecting a set a proteins and membranes to work with and selects reaction rules from a toolbox of core rules, encapsulating basic biological actions. Subsequently, the core rules may be refined by giving contextual application conditions for rules.

In summary, the slogan for modelling in the $\mathcal{C}$-calculus is: *Modelling by rule refinement.*

5.1 Future Work

We should start by underlining that the reason for including names for domains and membranes was to be able to express directly the concrete examples in Section 4. The central parts of the calculus—the core actions, captured in the projective generators, and the capture of diffusion and fission (via PINCH)—could have been discussed without names and state for domains, and does not rely on the specific model chosen. In this first version of the calculus, the concrete model for representing names and state for domains (and membranes) has been chosen mainly because it allows readers familiar with the κ-calculus as presented in [DL04] to build on the intuition from there. In future work, we expect to revise the model for names and state, to allow the expression of rules that mention domains eliding their name or state, and, more generally, to allow a more uniform treatment of names and state for all the entities in the calculus.

The rules for transport in the $\mathcal{C}$-calculus allow only transport of proteins and complexes. However, by generalizing the PINCH generator, we may model cell-division in eukaryotic cells. As sketched in the introduction, this involves, in particular, the division of membrane-enclosed organelles into two daughter-cells. We may model this by allowing membranes and gates to travel across channels. Due to the isolation of the computation of the connected component in the lcc function, it suffices with a simple refinement of the PINCH-rule and of the lcc-function.

The generic tool for working with bigraphs, the BPL tool [BPL07], allows experimentation with the fragment of the $\mathcal{C}$-calculus without transport (i.e., the rules PINCH and DIFF). In experimenting with different semantics for the

calculus, this has very been useful. However, as for the κ-calculus [DFFK07], a dedicated implementation can be optimized directly for the $\mathcal{C}$-calculus. For instance, knowing that well-formedness is preserved across reaction allows an implementation to optimize for this, utilizing, in particular, that all links except backbones are binary, and that backbone links are restricted to one or two specific shapes.

A good way to mature the language is to investigate larger models. In particular, the level of abstraction in the $\mathcal{C}$-calculus leads us to believe that we may capture smoothly the steps from receptor transcription to placement (this is a part of the so-called exo-cytic pathway). This involves, in particular, the steps of transcription for exo-particles and transmembrane receptor proteins, which in turn involves recognition of partial proteins in the cytosol during transcription by ribosomes. As domains, not full proteins, are atoms in the $\mathcal{C}$-calculus, we should be able to capture these reactions by extending the allowable domain-level reactions. Recognition of a partial protein induces a lodging of the partial protein in the endoplasmic reticulum surrunding the nucleus—this is essentially a merging between two kinds of membrane-enclosed compartments. After final transcription, a vesicle is formed to encapsulate the exo-part of the protein, and the vesicle is transported through the (3-layer) Golgi apparatus sorting mechanism, and finally merged with the plasma membrane.

Our binary gated channels also bear a striking resemblance to a recent discovery [RSM$^+$04, GBG08], so-called *tunneling nanotubes* (TNTs), long and thin stable membrane-enclosed nanotubes (diameter of of 50 to 200nm, several cell diameters in length) tunneling through the divide between two cells. TNTs were originally reported to allow transport only of vesicles (i.e., essentially diffusion of vesicles), but seem later to have been found to permit direct intercellular transfer of organelles, cytoplasmic molecules and membrane components [GBG08]. Many things are as yet unknown about these entities and the mechanisms and conditions they work under. They seem to be correspondents of membrane channels observed in plants, so-called *plasmodesmata* (PD), membrane channels providing both membrane and cytoplasmic connectivity between cells. The paper by Gurke et al. [GBG08], provides a recent overview and summary of the current knowledge about TNTs, and contains an overview of three proposals on the nano-structure and mechanics of TNTs. It would be valuable to investigate whether, in the $\mathcal{C}$-calculus, we may model easily each of these three different proposals.

We have also made a preliminary investigation of the analysis of *causality* for pure bigraphs, which we expect to be instantiable for the $\mathcal{C}$-calculus. Essentially, we may give a presentation for describing causal relations in terms of modification-testing dependencies, in which we may describe classical notions such as precedence, weak permutation, casuality, and concurrency.

Finally, due to recent results by Milner, Krivine, and Troina [KMT08], a stochastic extension of the nondeterministic calculus presented in this paper, is well-founded.

5.2 Related Work

Several languages have been proposed in order to model biomolecular systems. We have already discussed in detail, how the $\mathcal{C}$-calculus is inspired by several previous efforts, in particular, by the κ-calculus [DL04]. The Bio-κ-calculus by Laneve and Tarissan extends the κ-calculus with basic brane-inspired membranes [LT06]. However, splitting of membranes is a non-atomic procedure, which requires a considerable amount of (computationally expensive) encoding. In comparison, in the $\mathcal{C}$-calculus we have taken the atomic *pinch* primitive for membrane division, and isolated any associated splitting of proteins in a side-condition.

The idea to use formal calculi for mobile and distributed systems was launched by Regev, Shapiro, and Silverman. They use the π-calculus to represent and simulate [RSS01] biomolecular processes underlying protein signalling networks. They continue that work in the stochastic version of the π-calculus [PRSS01] (as treated by Priami in [Pri95]) taking into account both the time and probability of biochemical reactions. Proteins are encoded as parallel products of a series of domains, encoded via channels. As in the $\mathcal{C}$-calculus, name-sharing encodes both protein-backbones and complexation. As is standard in the π-calculus locality (i.e., membrane-encapsulated compartments) is encoded via the sharing of names, and mobility is encoded via name-passing. Conformational change is also encoded via mobility. The basic combinators of π-calculus suffice to encode sequential events, independent events (via parallel), or, mutually exclusive events (via sum). The line of work on using languages inspired by the stochastic π-calculus is continued in the school of work on β-binders [PQ05] lead by Priami and co-authors. In particular, β-binders add syntactic constructs for modelling membranes, but do not offer any solution to the problem of fission.

In Bio-ambients [RPS$^+$04], Regev et al. extend their earlier work by developing a modified version of the ambient calculus developed by Cardelli and Gordon [CG98, CG00]. The ambient calculus is extended to include parent-to-child communication as well as intra-ambient communication. The calculus is also equipped with a variety of dual capabilities to allow actions such as *entry*, *exit*, and *merging* of ambients. Named ambients model membrane-enclosed compartments; the ambient abstraction is also used to model proteins, allowing merge to model certain kinds of complex-formation and membrane fusion. The calculus encodes cross-membrane proteins by parent-child communication—by allowing proteins to communicate outside the membrane they are enclosed in. This has the side-effect of requiring the user to *not* model cross-membrane proteins via ambients, but instead as naked processes. Also, protein complex breakage is complicated by using merge to model complex-formation.

Cardelli continues the investigation of membrane interactions in the family of brane calculi [Car04a]. Here, the focus is aimed mainly at finding good abstractions for membranes and their behavior. Molecular structures are limited to atomic structures attached to membrane-surfaces. Cardelli identifies a set of basic membrane-actions that is motivated by the hydrophilic and hydrophobic properties of membranes, and coin the concept "bitonality" to describe their

constraints. In capturing a small set of membrane-actions in the $\mathcal{C}$-calculus, we are inspired by the brane calculi, and, in particular, by the *projective* variant of the brane calculi, by Danos and Pradellier [DP04]. Projective descriptions (which we explain and define in detail in the paper) let us describe conveniently sets of rules for reactions involving regions separated by membrane-surfaces, while eliding the *orientation* of those separating membranes.

In the recent paper on Bitonal Membrane Systems [Car08], Cardelli continues the line of work on membrane calculi, and investigates further the notion of projectivity and orientation to unify membrane-reactions that from a local perspective (a so-called local *patch* of the membranes) involve the same pattern of reconfiguration.

Finally, also related is the school of work on P-systems [Pau02] initiated by Paun. P-systems are essentially rewriting systems with locations at a high level abstraction. The aim of P-systems is more angled towards developing a general computational model *inspired* by biology, than towards giving a language for describing and investigating biology, however.

Acknowledgements The authors are grateful for discussions with and suggestions from Lars Birkedal, Søren Debois, Luca Cardelli, and Robin Milner. Some of this work was developed while the first author was visiting Catuscia Palamidessi's group at Laboratory for Informatics at École Polytechnique, Paris, and the third author was a research fellow in the same group.

6 Definitions Inherited from the $\mathcal{B}^{\Sigma,\mathcal{R}}$-framework

We include below, for easy reference, a few central definitions from the $\mathcal{B}^{\Sigma,\mathcal{R}}$-framework, instantiated to the signature used for the $\mathcal{C}$-calculus. For more extensive explanation and motivation, we refer the reader to [DK08].

Ordering of variables We are not interested in the particular numbers used for variables in a given term, only in their relative ordering inside that term. For instance, we do not wish to distinguish the solution $\Box_{45} \mid \Box_{56}$ from $\Box_0 \mid \Box_1$. All variables in terms are distinct, so we may order the variables in any term uniquely according to their numbers. It is therefore clear what we mean, when we refer to the ith variable in a given term.

Formally, we define order-preserving renumbering, and include it in the structural congruence relation. Write $\mathrm{nv}(C)$ for the set of numbers of variables in C.

Definition 6.1 (order-preserving renumbering)**.** Given a group G and an order-preserving and injective map $r : \mathrm{nv}(C) \to \mathbb{N}$, let $[\mathsf{G}]^r$ be the group G with all variables renumbered according to r.

Order-preserving renumberings are partial maps on natural numbers. We define also the *stratifying* renumbering, the renumbering that maps every vari-

able to the number given by its relative order. To that end, we order the renumberings themselves, pointwise.

Definition 6.2 (stratifying renumbering)**.** The stratifying renumbering of variables in a group G is the least order-preserving renumbering defined on $\mathrm{nv}(\mathsf{G})$.

We write $[\mathsf{G}]$ for the group G renumbered according to this renumbering, and call $[\mathsf{G}]$ stratified.

Free and bound names The new name operator $(x)\,\mathsf{S}$ is a binder for pure names—instances of the name x in S are α-convertible. We define inductively the set of free or bound names of a term as usual.

Definition 6.3 (free and bound names)**.** For solutions S the free names $\mathrm{fn}(\mathsf{S})$ and the bound names $\mathrm{bn}(\mathsf{S})$ are defined inductively as:

$$\begin{array}{rclcrcl}
\mathrm{fn}(\mathsf{m}[\mathsf{S}]) &=& \mathrm{fn}(\mathsf{m}[\mathsf{S}]) & & \mathrm{bn}(\mathsf{m}[\mathsf{S}]) &=& \mathrm{bn}(\mathsf{S}) \\
\mathrm{fn}(\mathsf{d}_b^x) &=& \{x, b\} & & \mathrm{bn}(\mathsf{d}_b^x) &=& \emptyset \\
\mathrm{fn}(\mathsf{d}_b) &=& \{b\} & & \mathrm{bn}(\mathsf{d}_b) &=& \emptyset \\
\mathrm{fn}(\overline{\mathsf{d}}_b) &=& \{b\} & & \mathrm{bn}(\overline{\mathsf{d}}_b) &=& \emptyset \\
\mathrm{fn}(\triangle_g) &=& \{g\} & & \mathrm{bn}(\triangle_g) &=& \emptyset \\
\mathrm{fn}(\mathsf{S} \mid \mathsf{T}) &=& \mathrm{fn}(\mathsf{S}) \cup \mathrm{fn}(\mathsf{T}) & & \mathrm{bn}(\mathsf{S} \mid \mathsf{T}) &=& \mathrm{bn}(\mathsf{S}) \cup \mathrm{bn}(\mathsf{T}) \\
\mathrm{fn}((x)\,\mathsf{S}) &=& \mathrm{fn}(\mathsf{S}) \setminus x & & \mathrm{bn}((x)\,\mathsf{S}) &=& x \cup \mathrm{bn}(\mathsf{S}) \\
\mathrm{fn}(0) &=& \emptyset & & \mathrm{bn}(0) &=& \emptyset \\
\mathrm{fn}(A) &=& \emptyset & & \mathrm{bn}(A) &=& \emptyset
\end{array}$$

For groups we extend the definition above to include also:

$$\begin{array}{rclcrcl}
\mathrm{fn}(\mathsf{G} \mid\mid \mathsf{F}) &=& \mathrm{fn}(\mathsf{G}) \cup \mathrm{fn}(\mathsf{F}) & & \mathrm{bn}(\mathsf{G} \mid\mid \mathsf{F}) &=& \mathrm{bn}(\mathsf{G}) \cup \mathrm{bn}(\mathsf{F}) \\
\mathrm{fn}((x)\,\mathsf{G}) &=& \mathrm{fn}(\mathsf{G}) \setminus x & & \mathrm{bn}((x)\,\mathsf{G}) &=& x \cup \mathrm{bn}(\mathsf{G}) \\
\mathrm{fn}(\epsilon) &=& \emptyset & & \mathrm{bn}(\epsilon) &=& \emptyset
\end{array}$$

Normal form Using structural congruence laws we may push binders to the top (performing α-conversion as needed), remove superfluous binders via elision, and remove empty solutions or groups to bring every solution and group to a normal form, resembling the standard form for CCS [Mil80].

Proposition 6.4 (normal forms)**.** *Every solution* S *is structurally congruent to a normal form*

$$\mathsf{S} \equiv (\tilde{x})(\mathsf{S}_0 \mid \cdots \mid \mathsf{S}_{n-1})$$

where each $\mathsf{S}_0, \cdots, \mathsf{S}_{n-1}$ *is a domain, a gate, or a membrane (empty or with another solution inside) containing no binders; and where* $\tilde{x} \subseteq \mathrm{fn}(\mathsf{S}_0) \cup \cdots \cup \mathrm{fn}(\mathsf{S}_{n-1})$*. (If* $n = 0$*, then* $\mathsf{S}_0 \mid \cdots \mid \mathsf{S}_{n-1} \stackrel{\text{def}}{=} 0$*, and if* $\tilde{x} = \emptyset$ *then the binder* $(\tilde{x})$ *is not there.)*

Every group G *is structurally congruent to a normal form*

$$\mathsf{G} \equiv (\tilde{x})(\mathsf{S}_0 \mid\mid \cdots \mid\mid \mathsf{S}_{n-1})$$

where each $\mathsf{S}_0, \cdots, \mathsf{S}_{n-1}$ *is non-empty, contain no binders, and is otherwise on (solution) normal form; and where* $\tilde{x} \subseteq \mathrm{fn}(\mathsf{S}_0) \cup \cdots \cup \mathrm{fn}(\mathsf{S}_{n-1})$*. (If* $n = 0$*, then* $\mathsf{S}_0 \,||\, \cdots \,||\, \mathsf{S}_{n-1} \stackrel{\text{def}}{=} \epsilon$*, and if* $\tilde{x} = \emptyset$ *then the binder* $(\tilde{x})$ *is not there.)*

The forms are unique up to α*-equivalence, and reordering of binders and parallel solutions (i.e., up to the commutative law for* $|$*).*

We write $\mathsf{G} \equiv_{\mathbf{N}} \mathsf{G}'$, if $\mathsf{G} \equiv \mathsf{G}'$ and G' is on normal form.

Definition 6.5 (width of group)**.** For

$$\mathsf{G} \equiv_{\mathbf{N}} (\tilde{x})(\mathsf{S}_0 \,||\, \cdots \,||\, \mathsf{S}_{n-1})$$

let $\mathrm{width}(\mathsf{G}) = n$.

Substitution Nonground solutions (and groups) have variables for which we may substitute other solutions.

We start by defining basic substitution of the variables in a group G by a solution S. (We define substitution only for groups, G. The definition for solutions follows as a special case.) Substitution is capture-avoiding, as usual.

Definition 6.6 (raw substitution)**.** Let φ be a bijective map from variables to solutions $\mathsf{S}_0, \mathsf{S}_1, \ldots, \mathsf{S}_{n-1}$.

The substitution $\mathsf{G}\varphi$ of variables in G by the solution in φ is defined when $|\,\mathrm{var}(\mathsf{G})| = n$, for all $i \in n$, $\mathrm{bn}(\mathsf{G}) \cap \mathrm{fn}(\mathsf{S}_i) = \emptyset$. In that case, we define $\mathsf{G}\varphi$ inductively over the structure of G,

$$\begin{array}{ll}
\mathsf{m}[\mathsf{S}]\,\varphi & = \mathsf{m}[\mathsf{S}\varphi] \\
\mathsf{d}_b^x\,\varphi & = \mathsf{d}_b^x \\
\mathsf{d}_b\,\varphi & = \mathsf{d}_b \\
\overline{\mathsf{d}}_b\,\varphi & = \overline{\mathsf{d}}_b \\
\triangle_g\,\varphi & = \triangle_g \\
((x)\,\mathsf{S})\,\varphi & = (x)\,(\mathsf{S}\varphi) \\
(\mathsf{S} \mid \mathsf{T})\,\varphi & = (\mathsf{S}\varphi \mid \mathsf{T}\varphi) \\
0\,\varphi & = 0 \\
X\varphi & = \begin{cases} \mathsf{S}_i & \text{if } \varphi(X) = \mathsf{S}_i \\ X & \text{else} \end{cases} \\
 & \\
((x)\,\mathsf{G})\,\varphi & = (x)\,(\mathsf{G}\varphi) \\
(\mathsf{G} \,||\, \mathsf{F})\,\varphi & = (\mathsf{G}\varphi \,||\, \mathsf{F}\varphi) \\
\epsilon\,\varphi & = \epsilon.
\end{array}$$

As each variable is a leaf in G it is easy to see that the substituted term respects the grammar (i.e., in Definition 2.4 and in Definition 2.5). In general, however, raw substitution is not well-behaved; it may lead to terms violating the requirement that numbered variables occur only once in expressions. We shall only use raw substitution as a means to define two versions of substitution, where this issue is resolved.

We start by defining total substitution, $\mathsf{G}\cdot\mathsf{F}$, the substitution of all variables in a group G with the solutions in a group F.

We define total substitution by pushing the binders of F to the top of the created term; hence, we also require that both bound and free names of G be distinct from bound names in F. This is simply a technical requirement, as we can always α-convert bound names of F to avoid any clashes.

Definition 6.7 (total substitution)**.** Substitution $\mathsf{G}\cdot\mathsf{F}$ of variables in G by solutions in F is defined when $|\operatorname{var}(\mathsf{G})| = \operatorname{width}(\mathsf{F})$ and $\operatorname{bn}(\mathsf{G})\cap\operatorname{fn}(\mathsf{F}) = \operatorname{bn}(\mathsf{G})\cap\operatorname{bn}(\mathsf{F}) = \operatorname{fn}(\mathsf{G})\cap\operatorname{bn}(\mathsf{F}) = \emptyset$. In this case, for

$$\mathsf{F} \equiv_{\mathbf{N}} (\tilde{x})(\mathsf{S}_0 \mid\mid \cdots \mid\mid \mathsf{S}_{n-1})$$

let

$$\mathsf{G}\cdot\mathsf{F} = (\tilde{x})[\mathsf{G}]\{\Box_0 \mapsto \mathsf{S}_0, \cdots, \Box_{n-1} \mapsto \mathsf{S}_{n-1}\},$$

Note, that we use stratifying renumbering to renumber the variables in G before substituting, to ensure that the variables in G are numbered severally from 0 to $n-1$. Substitution is associative.

Proposition 6.8 (total substitution is associative)**.** $(\mathsf{G}\cdot\mathsf{F})\cdot\mathsf{H} = \mathsf{G}\cdot(\mathsf{F}\cdot\mathsf{H})$.

We generalize our definition of substitution to partial substitution—where only some of the variables of the context G are substituted. Definition 6.9 generalizes Definition 6.7 to the cases, where F has fewer solutions than G has variables.

Definition 6.9 (partial substitution)**.** Partial substitution $\mathsf{G}\lhd\mathsf{F}$ of variables in G by solutions in F is defined when $|\operatorname{var}(\mathsf{G})| \geq \operatorname{width}(\mathsf{F})$ and (as for total substitution) when $\operatorname{bn}(\mathsf{G})\cap\operatorname{fn}(\mathsf{F}) = \operatorname{bn}(\mathsf{G})\cap\operatorname{bn}(\mathsf{F}) = \operatorname{bn}(\mathsf{G})\cap\operatorname{fn}(\mathsf{F}) = \emptyset$. In this case, let

$$\mathsf{G}\lhd\mathsf{F} = \mathsf{G}\cdot([\mathsf{F}] \mid\mid \Box_k \mid\mid \cdots \mid\mid \Box_{k+n}),$$

for $|\operatorname{var}(\mathsf{G})| - \operatorname{width}(\mathsf{F}) = n$ and $|\operatorname{var}(\mathsf{F})| = k$.

It is immediate from the definitions, that in the case where $|\operatorname{var}(\mathsf{G})| = \operatorname{width}(\mathsf{F})$, $\mathsf{G}\cdot\mathsf{F} \equiv \mathsf{G}\lhd\mathsf{F}$. In the case, where G has more variables, than F has solutions, F is extended with appropriately numbered variables.

Partial substitution is not in general associative; by convention we take it to be left-associative.

References

[BDGM07] Lars Birkedal, Troels Christoffer Damgaard, Arne J. Glenstrup, and Robin Milner. Matching of bigraphs. *Electronic Notes in Theoretical Computer Science*, 175(4):3–19, 2007.

[BPL07] The BPL Group. BPLweb—the BPL tool web demo, 2007. IT University of Copenhagen, Denmark. Prototype.

[Car04a] Luca Cardelli. Brane calculi. In Vincent Danos and Vincent Schächter, editors, *CMSB*, volume 3082 of *Lecture Notes in Computer Science*, pages 257–278. Springer, 2004.

[Car04b] Luca Cardelli. Brane calculi - interactions of biological membranes. In *Computational Methods in Systems Biology*, pages 257–278. Springer, 2004.

[Car08] Luca Cardelli. Bitonal membrane systems: Interactions of biological membranes. *Theor. Comput. Sci.*, 404(1-2):5–18, 2008.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In Maurice Nivat, editor, *FoSSaCS*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 1998.

[CG00] Luca Cardelli and Andrew Gordon. Mobile ambients. *Theoretical Computer Science*, 24:177–213, 2000.

[Coh07] William Cohen. *A Computer Scientist's Guide to Cell Biology.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[DB06] Troels C. Damgaard and Lars Birkedal. Axiomatizing binding bigraphs. *Nordic Journal of Computing*, 13(1-2):58–77, 2006.

[DFF+07] Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, and Jean Krivine. Rule-based modelling of cellular signalling. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR*, volume 4703 of *LNCS*, pages 17–41, 2007. Tutorial paper.

[DFFK07] Vincent Danos, Jérôme Feret, Walter Fontana, and Jean Krivine. Scalable modelling of biological pathways. In Z. Shao, editor, *Proceedings of APLAS 2007*, volume 4807, pages 139–157, 2007.

[DK08] Troels C. Damgaard and Jean Krivine. A generic language for biological systems based on bigraphs. Technical Report TR-2008-115, IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300 Copenhagen V, December 2008.

[DL04] Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theor. Comput. Sci.*, 325(1):69–110, 2004.

[DP04] Vincent Danos and Sylvain Pradalier. Projective brane calculus. In *Proceedings of CMSB'04*, pages 134–148, 2004.

[GBG08] Steffen Gurke, João F.V. Barroso, and Hans-Hermann Gerdes. The art of cellular communication: tunneling nanotubes bridge the divide. *Histochem Cell Biol.*, 129(5):539–550, May 2008.

[GS06] B. D. Grant and M. Sato. Intracellular trafficking. *Wormbook*, January 2006. Available online at `http://www.wormbook.org`.

[HR05] Reinhart Heinrich and Tom Rapoport. Generation of nonindentical compartments in vesicular transport systems. *The Journal of Cell Biology*, 168(2):271–280, January 2005.

[JM04] Ole H. Jensen and Robin Milner. Bigraphs and mobile processes (revised). Technical Report UCAM-CL-TR-580, University of Cambridge, February 2004.

[KC02] Michael M. Kozlov and Leonid Chernomordik. The protein coat in membrane fusion: Lessons from fission. *Traffic*, 3(4):256–267, 2002.

[KK03] Yonathan Kozlovsky and Michael M. Kozlov. Membrane fisson: Model for intermediate structures. *Biophysical Journal*, 85(1):85–96, July 2003.

[KMT08] Jean Krivine, Robin Milner, and Angelo Troina. Stochastic bigraphs. In *Porc. of MFPS'08, 24th Conference on the Mathematical Foundations of Programming Semantics*, volume to appear of *ENTCS*. Elsevier, 2008.

[LT06] Cosimo Laneve and Fabien Tarissan. A simple calculus for proteins and cells. In *Proceedings of the Workshop MeCBIC'06.*, 2006.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[Mil05] Robin Milner. Axioms for bigraphical structure. *Mathematical Structures in Computer Science*, 15(6):1005–1032, 2005.

[Mil06] Robin Milner. Pure bigraphs: structure and dynamics. *Information and Computation*, 204(1):60–122, 2006.

[Pau02] Gh. Paun. *Membrane Computing. An Introduction*. Springer, 2002. ISBN 3-540-43601-4.

[Pod06] Benjamin Podbilewicz. Cell fusion. *WormBook*, pages 1–32, January 2006. Available online at `http://www.wormbook.org`.

[PQ05] Corrado Priami and Paola Quaglia. Beta binders for biological interactions. In *Computational Methods in Systems Biology*, volume 3082, pages 20–33, 2005.

[Pri95] Corrado Priami. Stochastic π-calculus. *The Computer Journal*, 38(6):578–589, 1995.

[PRSS01] Corrado Priami, Aviv Regev, William Silverman, and Ehud Shapiro. Application of stochastic name-passing calculus to representation and simulation of molecular processes. *Infomation Processing Letters*, 80(1):25–31, 2001.

[RPS+04] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, and Ehud Shapiro. Bioambients: An abstraction for biological compartments. *Theoretical Computer Science*, 325:141–167, 2004.

[RSM+04] Amin Rustom, Rainer Saffrich, Ivanka Markovic, Paul Walther, and Hans-Hermann Gerdes. Nanotubular highways for intercellular organelle transport. *Science*, 303:1007–1010, 2004.

[RSS01] Aviv Regev, William Silverman, and Ehud Shapiro. Representation and simulation of biochemical processes using the π-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. World Scientific Press, 2001.

[WB] Online WormBook. `http://www.wormbook.org/`.