Prospects for Declarative Mathematical Modeling of Complex Biological Systems

Eric Mjolsness ¹
May 1, 2018

Abstract

Declarative modeling uses symbolic expressions to represent models. With such expressions one can formalize high-level mathematical computations on models that would be difficult or impossible to perform directly on a lower-level simulation program, in a general-purpose programming language. Examples of such computations on models include model analysis, relatively general-purpose model-reduction maps, and the initial phases of model implementation, all of which should preserve or approximate the mathematical semantics of a complex biological model. Multiscale modeling benefits from both the expressive power of declarative modeling languages and the application of model reduction methods to link models across scale. Based on previous work, here we define declarative modeling of complex biological systems by defining the semantics of an increasingly powerful series of declarative modeling languages including reaction-like dynamics of parameterized and extended objects; we define semantics-preserving implementation and semantics-approximating model reduction transformations; and we outline a “meta-hierarchy” for organizing declarative models and the mathematical methods that can fruitfully manipulate them.

1 Introduction

We propose and discuss an informal definition of declarative modeling in general, and provide as examples a collection of specific mathematical constructions of processes and extended objects for use in declarative models of complex biological systems and their processing by computer. Such processing can be symbolic and/or numerical, including for example model reduction by coupled symbolic and machine learning methods. The resulting apparatus is intended for semi-automatic synthesis and analysis of biological models, a computational domain which must typically deal with substantial complexity in the subject modeled.

We will define declarative models in Section 2 (informally in general but formally in particular cases); generalize the approach to extended objects in Section 3 by way of discrete graphs; and discuss progress in model reduction (usually across scales) by machine learning in Section 4. In

¹ Department of Computer Science, University of California Irvine CA 92697. Email: emj@uci.edu.
Section 5 we will propose the elements of a larger-scale mathematically based “meta-hierarchy” for organizing many biological models and modeling methods, enabled by the declarative approach to modeling.

Much of this paper reviews previous work, extending it somewhat and setting it in a broader context. The aim of this paper is not to provide a balanced summary of work in the field; instead it is aimed mainly at outlining mathematically the possibilities of particular directions for future development.

1.1 Notation

We will use square brackets to build sets (tuples or lists) ordered by indices, so in a suitable context $f([x]) = f(\{x_j | j \in \{1, \ldots, n\}\}) = f(x_1, \ldots, x_n)$. Multisets are denoted $\{(\text{multiplicity})(\text{element}), \ldots\}$. Indices may have primes or subscripts and are usually deployed as follows: $r$ indexes rewrite rules, $i$ indexes individual domain objects, $a, \beta$ index domain object types, $p, q$ index arguments in one or more argument lists in a rule, $c$ indexes variables in a rule, $A$ indexes a list of defined measure spaces, and $j, k$ index internal components of a vector parameter. Object typing will be denoted by “:”, so for example $n : \mathbb{N}$ means that $n \in \mathbb{N}$ and the usual set of integer arithmetic operations pertain to $n$. Subtyping will be denoted by “::”.

2 Declarative modeling

A useful distinction in programming languages is that between declarative and imperative languages. Roughly, declarative languages aim to make it easy to specify the goals or criteria of a successful computation, leaving many of the procedural decisions about how exactly to pursue those goals up to other software. Imperative languages on the other hand support the process of telling the computer exactly what to do - which is eventually necessary anyway. For example the Fibonacci series is a sequence of integers defined by a recursion equation (declarative), but the recursion equation can be implemented in software expensively using simple recursion. Alternatively it can be implemented more efficiently using either conventional looping or tail recursion (Abelson et al. 1996). These are two imperative options, the latter closer to declarative style. A mixed declarative/imperative route would be to automatically recognize and solve the recurrence relation by computer algebra, and then generate imperative code from the analytic solution. In mathematical modeling languages as distinct from programming languages, the opportunities for the declarative style are much greater owing to the practical power of symbolic reasoning, such as exact or approximate algebraic calculations and proofs, either before resorting to numerical simulation or mixed together with it.

An example of a formalizable language for modeling biology is a collection of partial differential equations in which the variables represent local concentrations of molecular species and the spatial differential operators are all diffusion operators $\nabla_x : (C_\alpha(x) \nabla_x)$. Such a deterministic reaction/diffusion model can be represented in a computer by one or more abstract syntax trees (ASTs) including nodes for variable names, arithmetic operations, spatial differential operators such as the Laplacian, the first-order time derivative operator, equality constraints including boundary conditions, and possibly function definitions. Such an AST can be used to denote a reaction-diffusion model as a data object that can be manipulated by computer algebra. It can also under
some conditions be transformed symbolically, for example by separation of time scales replacing a subset of ordinary differential equations (without spatial derivatives) by function definitions of algebraic rate laws to be invoked in the differential equations for the remaining variables (e.g. Ermentrout (2004)).

We generalize from this example. A formal language should have a “semantics” map $\Psi$, giving a mathematical meaning to some defined set of valid expressions in the language. For a modeling language, each valid model declaration $M$ should correspond to an instance $\Psi(M)$ in some space $S$ of “dynamical systems” interpreted broadly, so that such systems may be stochastic and/or infinite-dimensional. If some of the semantically meaningful model expressions are composed of meaningful sub-expressions, and their semantics can be combined in a corresponding way, then the semantics is “compositional”; composition commutes with the semantic map. Likewise for valid transformations of model declarations, one would like the semantics before and after transformation to yield either the same mathematical meaning (equivalent dynamics), or two meanings that are related somehow eg. by approximate equality under some conditions on parameters that may be partially known by proof and/or numerical experiment.

So in the context of modeling languages in general and biological modeling languages in particular, we think the key advantages of the declarative language style are captured by the following informal (but perhaps formalizable) definition:

A declarative modeling language $L$ is a formal language together with (a) compositional semantics $\Psi : L \rightarrow S$ that maps all syntactically valid models $M$ in $L$ into some space $S$ of dynamical systems, and (b) conditionally valid or conditionally approximately valid families of transformations on model-denoting AST expressions in the modeling language $L$. These AST transformations can be expressed formally in some computable meta-language, though the meta-language need not itself be a declarative modeling language.

Under this informal definition the utility of a declarative modeling language would depend on its expressive power, addressed in Sections 2.1 and 5 below, and on the range, value, and reliability of the model-to-model transformations that can be constructed for it, to be discussed in Sections 4 (implementation), 5 (model reduction), and 5 (wider prospects). Multiscale modeling benefits from both the expressive power (e.g. representing cellular and molecular processes in the same model) and model reduction (finding key coarse-scale variables and dynamics to approximate fine-scale ones) elements of this agenda.

Although it is plausible that “one can’t proceed from the informal to the formal by formal means” (Perlis 1982), so that the task of formalizing a complex biological system to create models must begin informally, we will nevertheless try to be systematic about this task by building the semantic map $\Psi$ up out of “processes” and “objects” of increasing generality and structure. These processes and objects are represented by “expressions” in a language, and mapped to mathematical objects that together define a model. So the map $\Psi$ will be concerned with expressions, processes, and objects.

### 2.1 Semantic Maps

We show how to define a series of simple quantitative modeling languages of increasing expressive power, accompanied by semantic maps $\Psi(M)$ to operator algebras. To do this we need some idea concerning how to represent elementary biological processes with syntactic expressions that can include numerical quantities. One such idea begins with chemical reaction notation.
2.1.1 Pure Reaction Rules

In biology, generalized versions of rewrite rules naturally specify biochemical processes in a declarative modeling language, as well as model transformations in a metalanguage. The most straightforward example is chemical reaction notation. As in (Mjolsness 2010) we could use chemical addition notation:

\[
\left( \sum_{\alpha=1}^{A_{\text{max}}} m_{\alpha}(r) A_{\alpha} \right) \xrightarrow{k_{(r)}} \left( \sum_{\beta=1}^{A_{\text{max}}} n_{\beta}(r) A_{\beta} \right),
\]

where \( m_{\alpha}(r) \) and \( n_{\beta}(r) \) are nonnegative integer-valued stoichiometries for molecular species \( A_{\alpha} \) indexed by \( \alpha \) in reaction \( r \) with nonnegative reaction rate \( k_{(r)} \), and we may omit summands with \( m_{\alpha}(r) = 0 \) or \( n_{\beta}(r) = 0 \). But really the left and right hand sides of the reaction arrow are not sums but multisets with nonnegative integer multiplicities of all possible reactants, defaulting to zero if a reactant is not mentioned:

\[
\{ m_{\alpha}(r) A_{\alpha} \}_{*} \xrightarrow{k_{(r)}} \{ n_{\beta}(r) A_{\beta} \}_{*}.
\]

Either syntax expresses the transformation of one multiset of symbols into another, with a numerical or symbolic quantitative reaction rate \( k_{(r)} \). The syntax is easily encoded in an abstract syntax tree (AST). An example AST transformation might be a meta-rule that reverses an arrow (Yosiphon 2009) and changes the name of the reaction rate to (for example) \( k_{(r')} \), for a new reaction number \( r' \), allowing for the possibility of detailed balance to be satisfied in a collection of reactions. Many other reaction arrow types (e.g. substrate-enzyme-product) can then be defined by computable transformation to combinations of these elementary mass action reactions (e.g. (Shapiro et al. 2003; Yang et al. 2005; Shapiro et al. 2015) for examples in a declarative modeling context), using either commercial (Wolfram Research 2017) or open-source (Joyner et al. 2012) computer algebra system software.

How can we define a compositional semantics for this reaction notation? Fortunately the operator algebra formalism of quantum field theory can be adapted to model the case of ordinary (non-quantum) probabilities governed by the law of mass action in a Master Equation (Doi 1976a,b; Peliti 1985; Mattis and Glasser 1998; Mjolsness 2010) (and Morrison and Kinney (2016) for the equilibrium case). The corresponding semantics \( \Psi(M) \) is the creation/annihilation operator monomial

\[
\hat{W}_r = k_{(r)} \left\{ \prod_{\beta \in \text{rhs}(r)} (\hat{a}_\beta)^{n_{\beta}(r)} \right\} \left\{ \prod_{\alpha \in \text{lhs}(r)} (a_\alpha)^{m_{\alpha}(r)} \right\}
\]

which acts on probability distributions over states given by vectors of nonnegative integers \( n_\alpha \) (distinguished from \( n_{\beta}(r) \) by its lack of a superscript), one for each molecular species. The mass action law is encoded in the annihilation operators \( a_\alpha \). Annihilation operator subscript \( \alpha \) indexes the species present in the multiset on the left hand side (LHS) of Equation (2); the corresponding operator is raised to the power of its multiplicity or ingoing stoichiometry \( m_{\alpha}(r) \) in reaction \( r \), destroying that many particles of species \( \alpha \) if they exist (and contributing zero probability if they
don’t). Likewise creation operator \( \hat{a}_\beta \) has subscript \( \beta \) indexing the species present in the multiset on the right hand side (RHS) of Equation (2), and the operator is raised to the power of its outgoing stoichiometry \( n^{(r)}_\beta \) which indicates how many particles of species \( \beta \) are to be created.

If the number of particles of a given species is any nonnegative integer, as we assume for molecular species in solution, then annihilation and creation operators have infinite dimensional representations and satisfy the commutator of the Heisenberg algebra \([a_\alpha, \hat{a}_\beta] = a_\alpha \hat{a}_\beta - \hat{a}_\beta a_\alpha = \delta_{\alpha\beta} I_\alpha\), where \( I \) is the identity operator and \( \delta \) is the Kronecker delta. This commutator can be used to enforce a “normal form” on polynomials in which annihilation operators precede creation operators in each monomial, as in Equation (3). We can define the diagonal number operator \( N_\alpha = \hat{a}_\alpha a_\alpha \).

Another case is that in which the number of “particles” of any given species must be in \( \{0, 1\} \), as for example if the species are individual binding sites occupied by a particular molecular species; then the operators are \( 2 \times 2 \) matrices \( a = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \) and \( \hat{a} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \) obeying \([a_\alpha, \hat{a}_\beta] = a_\alpha \hat{a}_\beta - \hat{a}_\beta a_\alpha = \delta_{\alpha\beta} (I_\alpha - 2\hat{a}_\alpha a_\alpha)\). In this case we can define the diagonal number operator \( N_\alpha = \hat{a}_\alpha a_\alpha = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \), the zero-checking operator \( Z_\alpha = I_\alpha - N_\alpha = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \), and the “erasure” projection operator \( E_\alpha = (Z_\alpha + a_\alpha) = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \) which takes either state to the zero-particle state. In this case also \( a^2 = \hat{a}^2 = 0 \).

The semantics given by Equation (3) is compositional or structure-respecting because:

(a) The operators for multiple rules indexed by \( r \) in a ruleset map to operator sums:

\[
W = \sum_r W_r , \quad \text{where} \quad W_r \equiv \hat{W}_r - \text{diag}(1 \cdot \hat{W}_r) \quad \text{and} \quad \hat{p} = W \cdot \hat{p} .
\]

In Equation (4) the first statement is ruleset compositionality, the second ensures conservation of probability, and the third is the resulting Chemical Master Equation (CME) stochastic dynamical system for the evolving state probability \( p(n) \). Under Equation (3) one may calculate that \( \text{diag}(1 \cdot \hat{W}_r) \) equals the diagonal operator \( \prod_{i \in \text{lhs}(r)} (N_\alpha)^{m_\alpha^{(r)}} \). One may regard this linearity as a linear mapping of vector spaces (hence as a morphism or category arrow): The source vector space is spanned by basis vectors that correspond to ordered pairs of multisets of species symbols, weighted by scalar reaction rates \( k_r \), and the target vector space is a vastly larger space of possible probability-conserving operators. Also,

(b) The multisets on the left hand side and right hand side of a rule each map to an operator product in normal form (incuding powers for repeated multiset elements) in Equation (3). Each product consists of commuting operators so their order is not important.

There are various possible formalizations of the “structure respecting” nature of mapping \( \Psi \) from rulesets to operator algebra, in addition to the vector space mapping just mentioned. One could consider just a vector space orthant with nonnegative reaction rate coefficients. One could take a page from Homotopy Type Theory (discussed in Section 3) and upgrade \( \Psi \) to a functor between categories, by making each vector space a category with morphisms such as piecewise linear paths from one vector to another, among other options. We do not yet have a uniformly preferred candidate for formalizing the idea of “respecting structure” in mathematical modeling languages.

The integer-valued index \( r \) we used to name the reactions is part of a meta-language for the present theorizing, and not part of the modeling language. One slightly confusing point is that
these unordered collections of chemical reactions are expressions in the language, but they also
have a form reminiscent of a grammar for another language - albeit a language of multisets repres-
enting the system state, rather than of strings or trees, and a language that may be entirely devoid
of terminal symbols (e.g. inert products such as waste). The CME as semantics was suggested by
(Mjolsness and Yosiphon 2006) and by (Cardelli 2008), though it can be regarded as implicit in
the Doi-Peliti formalism. There is also a projection from continuous-time (CME) semantics to
discrete-time probabilistic semantics.

From this operator algebra semantics and the time ordered product expansion one can de-
rive valid exact stochastic simulation algorithms including the Gillespie algorithm and various
generalizations, as detailed in (Mjolsness 2013). Such algorithms can also be accelerated exactly
(Mjolsness et al. 2009), and accelerated further by working hierarchically at multiple scales and/or
using parallel computing (Orendorff and Mjolsness 2012). The same theory can be used to derive
algorithms for the inference of reaction rate parameters from sufficient data (Wang et al. 2010) (cf.
Golightly and Wilkinson (2011)), although sufficient data may be hard to obtain.

Thus, unordered collections of pure chemical reactions provide a simple example of a model-
ing language with a compositional semantics. But for most biological modeling we need much
more expressive power than this.

2.1.2 Parameterized Reaction Rules

The first modeling language escalation beyond pure chemical reaction notation is to particle-like
objects or “agents” that bear numerical and/or discrete parameters which affect their reaction
reaction rates. For example, the size of a cell may affect its chances of undergoing cell division.
This kind of multiset rewrite rule can be generalized to (Mjolsness 2010)

\[ \left\{ \tau_{a(p)}[x_p] | p \in L_r \right\} \rightarrow \left\{ \tau_{b(q)}[y_q] | q \in R_r \right\} \quad \text{with} \quad \rho_r([x_p], [y_q]). \] (5)

The parameters \( x_p, y_q \) (indexed by positions \( p, q \) in their respective argument lists, and which may
themselves be vectors \( x_p \)) introduce a new aspect of the language, analogous to the difference
between predicate calculus and first order logic: Each parameter can appear as a constant or as
a variable, and the same variable \( X_c \) can be repeated in several components of several parameter
lists in a single rule. Thus it is impossible in general to say whether two parameters in a rule are
equal or not, and thus whether two terms \( \tau_{a(p)}[x_p] \) in a rule are the same or not, just from looking
at the rule - that fact may depend on the values of the variables, known only at simulation time.
The \( p, q \) subindex notation is as in (Mjolsness 2013). Here we have also switched from molecule-
like term names \( A_a \) to more generic logic-like term names \( \tau_{a(p)} \). The reaction rate \( \rho_r([x_p], [y_q]) \)
now depends on parameters on one or both sides of the rewrite rule which can be factored (auto-
matically in a declarative environment, as in (Yosiphon 2009)) into a rate depending on the LHS
parameters only and a conditional distribution of RHS parameters given LHS parameters.

For example a stem cell of volume \( V \) might divide asymmetrically yielding a stem cell and a
transit-amplifying cell:

\[
\text{stemcell}[x, V, \ldots] \rightarrow \text{TACell}[x + \Delta x, V/2, \ldots], \text{stemcell}[x - \Delta x, V/2, \ldots]
\]

\[ \text{with} \quad \hat{\rho}(V) \mathcal{N}(\Delta x; V^{1/d}), \]
where \( \dot{\rho}(V) \) is a probability per unit time or “propensity” for cell division depending on cell volume, and \( \mathcal{N}(\Delta x; cV^{1/d}) \) is a Gaussian or normal probability density function with diagonal covariance proportional to a lengthscale set by cell volume. This kind of example (Yosiphon 2009, Mironova et al. 2012, Mjolsness 2013) can be implemented in a computer algebra system (Yosiphon 2009, Shapiro et al. 2013), and shows that with parameters, reaction-like rewrite rules can represent (for example) both cellular and molecular processes in the same model, and of course their interactions, which is a key expressiveness capability for multiscale modeling.

Even for multilevel modeling of a single-scale reaction network, dynamic parameters allow for the possibility of aggregate objects that keep track of many individual particles including their number. Such aggregate objects would have their own rules as discussed in (Yosiphon 2009 Section 6.4), possibly obtained from fine-scale reaction rules by meta-rule transformation.

To specify the compositional semantics for parameterized rewrite rules it is necessary to specify the probability space in which distributions are defined. In (Mjolsness 2010) a fairly general space \( S \) of dynamical systems available as targets for the semantics map \( \Psi(M) \in S \) is specified in terms of master equations (\( \dot{p} = W \cdot p \)) governing the evolution of (classical, non-quantum) probability distributions \( p \) in Fock spaces constructed out of elementary measure spaces for the parameters.

In order to define the semantics of variables we will need to integrate over their possible values. Each variable \( X_c \) will have a type required by its position(s) in the argument list \([x_p]\) of one or more terms \( \tau_{A(c)}(p) \), and a corresponding measure space \( D_{A(c)} \) and measure \( \mu_{A(c)} \). Here \( A \) indexes some list of available measure spaces including discrete measure on \( \mathbb{N} \) and Lebesgue measure on \( \mathbb{R}^d \).

Let \( V_{\alpha} = \bigotimes_p D_{A(\alpha,p)} \) be the resulting measure space for parameter lists of terms of type \( \alpha \). To summarize briefly the “symmetric Fock space” construction outlined in (Mjolsness and Yosiphon 2006; Mjolsness 2010), for each nonnegative integer \( n_{\alpha} \) we define a measure space of states that have a total of \( n_{\alpha} \) “copies” of parameterized term \( \tau_{A(\alpha)}(x_{\alpha}) \):

\[
f_{\alpha}(n_{\alpha}) = \left( \bigotimes_{m=1}^{n_{\alpha}} V_{\alpha} \right) / S(n_{\alpha})
\]

where \( S(n) \) is the permutation group on \( n \) items - in this case the elements of the \( n_{\alpha} \) measure space factors. Next, any number \( n_{\alpha} : \mathbb{N} \) of terms is accommodated in a disjoint union of measure spaces \( f_{\alpha}(n_{\alpha}) \), resulting in a measure space for all terms of type \( \alpha \), and a cross product is taken over all species \( \alpha \):

\[
f_{\alpha} = \bigoplus_{n_{\alpha}=0}^{\infty} f_{\alpha}(n_{\alpha}) \quad \text{and} \quad f = \bigotimes_{\alpha} f_{\alpha}.
\]

This is the measurable system state space. In it, objects of the same type \( \alpha \) are indistinguishable except by their parameters \( x_p : V_{\alpha} \).

The operator semantics now becomes an integral over all the variables \( X_c \):

\[
\hat{W}_r = \int \ldots \int_{D_{c}[c]} \left( \prod_k d\mu_{A(c)}(X_c) \right) \rho_r \left( \left[ x_p([X_c]) \right], \left[ y_q([X_c]) \right] \right) 
\times \left\{ \prod_{q \in \text{rhs}(r)} a_{\rho(q)}(y_q([X_c])) \right\} \left\{ \prod_{p \in \text{lhs}(r)} a_{A(p)}(x_p([X_c])) \right\}.
\]
This expression is again in normal form, with annihilation operators preceding (to the right of) creation operators. When other operator expressions need to be converted to normal form one uses the same kinds of commutation relations as before except that the $\delta_{\alpha\beta}$ Kronecker delta functions are now augmented by Dirac and product delta functions as needed to cancel out the corresponding measure space integrals:

$$ [a_\alpha(x), a_\beta(y)] = \begin{cases} 
\delta_{\alpha\beta} \delta_{\mu A} (x - y) I_\alpha & \text{for } n \in \mathbb{N} \\
\delta_{\alpha\beta} \delta_{\mu A} (x - y) (I_\alpha - 2N_\alpha(x)) & \text{for } n \in \{0, 1\} 
\end{cases} $$

(9)

The integrals over measure spaces act a bit like quantifiers in first order logic, binding their respective variables. We speculate that the product of two such operator expressions could be computed in part by using the logical unification algorithms of computational symbolic logic, since the problem of finding the most general unifier arises naturally when integrating over several sets of variables during restoration of canonical form (cf. the proof of Proposition 1 in Section 3.3 below) using the commutation relations of Equation (9) and their delta functions.

Equations (5) and (8) comprise the syntax and semantics of the basic ruleset portion of Stochastic Parameterized Grammars (SPG) language of (Mjolsness and Yosiphon 2006; Yosiphon 2009). A simulation algorithm is derived in (Mjolsness 2013). Other features such as submodels, object type polymorphism and graph grammars are also included.

This notation is fundamentally more powerful than pure chemical reactions, because now reaction/process rates $\rho_r([x_p^p], [y_q^q])$ can be functions of all the parameters involved in a rule, and a rule firing event can change those parameters. It becomes possible to express sorceror’s apprentice models which purport to accomplish an infinite amount of computing in a finite simulated time, though this situation can also be avoided with extra constraints on the rate functions in the language.

A related fully declarative modeling language rides under the banner of “L-systems”, named after Lindenmeyer and championed by Prusinkiewicz (e.g. (Prusinkiewicz and Lindenmeyer 1990)). L-systems and their generalizations have been effective in the modeling of a great variety of developmental phenomena, particularly in plant development, because of their declarative expressive power. But the usual semantics of L-systems falls outside the family of languages considered in this section and in Section 3.3.2 below, for a theoretically interesting reason. The applicable rules of L-systems are usually defined to fire in parallel, and synchronously in discretized time so that one tick of a global clock may see many discrete state updates performed. This semantics seems to be incompatible with the addition of local, continuous-time operators defined in Equation (4), because “atomic” uninterruptible combinations of events are represented in operator algebra by multiplication rather than by addition of operators (and even then they are serialized), and on the other hand continuous-time parallel processes are represented by addition rather than multiplication of operators, in the master equation that describes the operation of processes in continuous time. Despite this difference, one could seek structure-respecting mappings between operator algebra and L-system semantics at least at the level of individual rule-firings.

Another relevant point of comparison for the operator algebra semantics of parameterized reaction network languages is the BioNetGen modeling language (Blinov et al. 2004). This language has been applied to many problems in signal transduction pathway modeling with discretely parameterized terms representing multistate molecular complexes. It also represents labelled graph
structures that arise in such molecular complexes, placing it also in the class of graph rewrite rule
dynamics languages discussed in detail in Section 3.3.

These (parameterized rewrite rule) models would seem to be non-spatial, but particle move-
ment through space can already be encoded using discrete or continuous parameters that denote
spatial location. Such motion would however have to occur in discrete steps. The solution to that
limitation is another language escalation.

2.1.3 Differential Equation Rules

Another form for parameterized rewrite rules licenses locally attached ordinary differential equa-
tions (ODEs) for continuous parameters, as in (Prusinkiewicz et al. 1993; Mjolsness 2013; Mjol-
sness et al. 1991), as part of the language \( L \):

\[
\begin{align*}
\{ \tau_{\alpha(p)}[x_p]|p \in L_r = R_r \} \rightarrow & \{ \tau_{\alpha(p)}[x_p + dx_p]|q \in R_r = L_r \} \text{ with } \left\{ \begin{array}{l}
\frac{dx_p}{dt} = v_p([x_k])dt \\
\end{array} \right\}, \text{ i.e.} \tag{10}
\end{align*}
\]

where the second form uses component rather than vector notation for all the ODEs. The first
form is more readily generalizable to stochastic differential equations (SDEs) \( dx_p = v_p([x_k])dt + \eta_p \sqrt{dt}, \eta_p \sim \pi_{\eta([x_k])} \) where \( \pi_{\eta} \) is a probability density function conditional on \([x_k]|k\). The ODE
semantics is given by the corresponding differential operators:

\[
\hat{W}_{\text{drift}} = - \int d\{x\} \int d\{y\} \hat{a}([y])a([x]) \left( \sum_j \nabla_{y_j} v_j([y]) \prod_k \delta(y_k - x_k) \right) \tag{11}
\]

as shown for example in (Mjolsness 2013). The SDE case is discussed in (Mjolsness and Yosiphon
2006), section 5.3.

Equations (5), (10) and (8), (11) comprise the syntax and semantics respectively of the ba-
sic ruleset portion of Dynamical Grammars (DG) language of (Mjolsness and Yosiphon 2006;
Yosiphon 2009), by addition of differential equations to SPGs. A simulation algorithm is de-
rived in (Mjolsness 2013). The generalization to operator algebra semantics for Partial Differential
Equations (PDEs) and Stochastic Partial Differential Equations (SPDEs), as a limit of spatially dis-
cretized ODE and SDE systems, is outlined (although not rigorously) in (Mjolsness 2010).

These differential equation bearing rules can be used to describe processes of growth and
movement of individual particle-like objects, as in “agent-based” modeling. For example, a cell
may grow according to a differential equation and divide with a probability rate that depends on
its size, as in the plant root growth model of (Mironova et al. 2012). In general rewrite rules can
be used to describe individual processes within a model, if they are augmented (as above) with
a symbolically expressed quantitative component such as a probability distribution or a differ-
ential equation. Each such process has a semantic map \( \Psi \) to a ring of operators, and processes
operating in parallel on a common pool of objects compose by operator addition (Equation (11)).
Declarative computer languages based on this and other chemical reaction arrows, transformed
to ordinary differential equation deterministic concentration models, include (Shapiro et al. 2015,
2003; Mjolsness 2013; Yosiphon 2009) among others.
Given a collection of rules with differential equations attached, is there a way to get rid of the rules and just keep the differential equations \( \frac{dx_j}{dt} = v_j([x_k]) \)? That would only be the case if the left hand side and right hand side of the rules all take the same form as in Equation (10), so that only the parameters of modeled objects are dynamic, not the number or structural relationships of such objects. There is for example the deterministic approximation of stochastic chemical kinetics in which that assumption is valid: chemical reactions are assumed to be well-mixed in a fixed compartment such as the cytoplasm of a cell, and the real-valued concentration of each molecular species becomes a parameter of the enclosing compartment. The reaction-like rules of the chemical reaction network are all dropped in favor of mass action differential equations attached to the compartment, yielding a multicompartamental modeling language that translates to fixed differential equations (similar to (Shapiro et al. 2003) among many others). But since cells actually divide and change their interconnections, it is still valuable to add in cell division and reconnection rules and translate instead to dynamically changing sets of differential equations whose structure gets discretely altered every so often (Shapiro et al. 2013), as permitted by Equations (5) and (8).

Complex biological objects often have \textit{substructure} whose dynamics is not easily captured by a fixed list of parameters and a rate function or differential equation that depends on those parameters. Extended objects such as molecular complexes, cytoskeletal networks, membranes, and tissues comprising many cells linked by extra-cellular matrix are all cases in point. We now wish to extend the syntax and semantics of the foregoing class of declarative languages to handle extended objects systematically, by creating a compositional language and semantics for biological \textit{objects} as well as processes, and then extending the semantics for processes accordingly. In the case of discrete substructure this can be done with labelled graph structures (discussed in Section 3.2 below). In some cases the sub-objects (such as lipid molecules in a membrane, or long polymers in cytoskeleton) are so numerous that an approximate spatial continuum model is justified and simpler than a large spatially discrete model. Spatial continuum models with geometric objects can be built out of manifolds and their embeddings, along with biophysical fields represented as functions defined on these geometries, in various ways we now discuss.

3 Extended objects

Can we declaratively model non-pointlike, extended biological objects such as polymer networks, membranes, or entire tissues in biological development? To achieve constructive generality in treating such extended objects we introduce, in Section 3.2, ideas based on discrete graphs and their possible continuum limits. These ideas include graded graphs, abstract cell complexes, stratified graphs, and combinations of these ideas. Dynamics by rewrite rules, beginning with graph rewrite rules, are developed in Section 3.3. But first we discuss the more general nonconstructive types of extended objects that we may wish to approximate constructively.

3.1 Nonconstructive extended objects

Classical mathematics is not constrained by the requirement of being computationally constructive, though it can be. This lack of constraint makes easier to establish useful mappings and equivalences (compared for example to Intuitionist mathematics), so it is easier to reuse what is already known in proving new theorems. We would like to retain these advantages in a theoretical phase of work before mapping to constructive computer simulations.
Classical mathematical categories such as topological spaces, measure spaces, manifolds, CW cell complexes, and stratified spaces (the latter two composed of manifolds of heterogeneous dimension) provide models of extended continuum objects such as biological cell membranes, cytoskeleton, and tissues made out of many adjacent cells. Objects in such categories may (depending on the category) have rich and useful collections of \( k \)-ary operators (unary, binary, countable associative, etc. object-valued operators, not to be confused with the probability-shifting dynamical creation/annihilation operators of Section 2.1) such as category sums and products, and function arrows for Cartesian Closed Categories such as compactly generated topological spaces (Steenrod 1967; Booth and Tillotson 1980). Such \( \otimes_C, \otimes_C, \rightarrow_C \) operators for category \( C \) can be targeted by compositional semantics from context-free grammar rules that generate expressions including these operators in an AST in a modeling language; these are essentially “type constructor” type inference rules in standard programming language semantics (Pierce 2002).

Further object-generating operators may require mathematical objects in heterogeneous but related categories, such as (a) defining a quotient topological space from an equivalence relation (whose morphisms are coarsenings) on points of the original space, or (b) defining new submanifolds by level sets of continuous functions using the regular value theorem (related to the implicit function theorem), or as the image of a continuously differentiable embedding ([Hirsch 1976] Chapter 1, Theorems 3.1 and 3.2.). Such level set functions could be biophysical fields such as concentration of morphogen for tissue domain boundary, or cortical microtubules in the preprophase band for plant cell division, or they could be purely mathematical phase fields that rapidly interpolate between discrete values for different compartments. Also objects in algebraic categories such as categories of groups, rings, fields, and vector spaces may bring their “internal” algebraic operations into the purview of a semantic map that is further constrained to represent these operations. Thus \( \Psi \) should be a highly constrained structure-respecting function between the object collections of the corresponding categories of model object expressions and mathematical objects.

With extended objects we encounter the possibility that the type of one object is itself another typed object. For example a point \( x : \tilde{S} \) may be constrained to lie on the surface of a sphere \( \tilde{S} : \text{Manifold}(d = 2) :: \text{Manifold} \) which is itself of type 2-dimensional manifold or more generally a Manifold. Indeed this possibility may be taken to define an “extended” object like \( \tilde{S} \): It is, or it at least informs, the type of its sub-objects. But for consistency one might like to map domain objects to “mathematical objects” that find their formal packaging in category theory by the same definition, whether they are constituent objects like point \( x \) or extended objects like 2D manifold \( \tilde{S} \). There are at least two natural ways to do this: (1) Define a mathematical object \( x \) as a point in a “space” \( X \) which is a category-object in some (preferably topological) category, and also define a category of categories containing \( \text{Top} \), such as the small monoidal categories (though this choice might be too inclusive), to legitimize space \( X \) as also being mathematical object by this definition; or (2) Define a mathematical object \( x \) directly as an object in a category, and define a category whose category-objects are the points of a topological space such as \( \tilde{S} \) (itself a category-object in \( \text{Top} \) or \( \text{Manifold} \)). The latter option is taken in Homotopy Type Theory (HTT 2013), with continuous paths in a space as morphisms between its points. An enriched category results. In our case this would lead to homotopy on manifolds, CW complexes, and stratified spaces as well as graph path homotopy considerations.

By these means we could nonconstructively define the mathematical semantics map \( \Psi \) of an extended object model in terms of functors (or at least structure-representing functions) to classical mathematical categories. However the mathematical objects in these spaces are not generally
defined in a computable way, so we can’t get all the way to computable implementations by this route.

Our strategy to achieve computability will be to define another fundamental kind of map in addition to semantics maps $\Psi$: namely, Implementation maps $I$ from restricted versions of potentially nonconstructive but standard mathematical categories, to labelled graphs. The restricted version could for example be something like piecewise linear stratified spaces (Weinberger 1994), or low-order polynomial splines to better preserve the differential topology of manifolds and stratified spaces ((Hirsch 1976) Chapter 3; (Weinberger 1994) Part II). By abuse of terminology such implementation maps may alternatively run from language (with classical mathematics semantics) to language (with graph semantics), provided the corresponding diagram commutes. In other words, implementation should commute with the semantic map, both respecting composition. This constraint applies to $I$ both as it acts on extended objects, and on the process models (e.g. differential equations or Markov chains) under which they evolve. Diagram 5 in Section 3.3.3 will provide examples.

Finite extended objects can be modeled by labelled graphs defined in the next section. These are eminently computable. Infinite graphs comprise a borderline case. One can use sequences of finite, computable labelled graphs to approximate continua. In this way we can have access to continuum extended objects as declarative modeling “semantics” for very fine-grained biological objects, while “implementing” these infinite mathematical objects in terms of explicitly finite and computable ones. Of course one’s computational resources may or may not be adequate to making the approximation needed.

A major goal for declarative modeling of continuum objects is declarative modeling of partial differential equations (PDEs), before they have been spatially discretized into e.g. ordinary differential equation systems. Software that already supports some PDE models declaratively, though not in conjunction with all the other process types of Section 2 or 3.3, includes the weak form specification language of Fenics (Logg et al. 2012) for finite element method (FEM) numerical solution, and at least one commercial computer algebra system (Wolfram Research 2017). Of course the choice of numerical solution methods for PDEs is a vast area of research; here we just observe that methods for which the mapping from continuum to discrete descriptions (e.g. meshing) is dynamic and opportunistic, such as Discontinuous Galerkin methods generalizing FEMs, or Lagrangian methods including particle methods for fluid flow, would place extra demands on the flexibility of the graph formalism for extended objects introduced in the next section.

### 3.2 Constructive extended objects via graphs

Discrete graphs, especially when augmented with labels, are mathematical objects that can represent computable objects and expressions at a high level of abstraction.

#### 3.2.1 Standard definitions

An (undirected/directed) graph is a collection $V$ of nodes and a collection $E$ of edges each of which is an (unordered/ordered) pair of vertices. A graph homomorphism is a map from vertices (also called nodes) of source graph to vertices of target graph that takes edges (or links) to edges but not to non-edges. Graphs (undirected or directed) and their homomorphisms form a category of graphs and graph homomorphisms. The category of undirected graphs can be modeled within the
category of directed graphs by a functor representing each undirected edge by a pair of oppositely
directed edges between the same two vertices.

Either category of graphs and graph homomorphisms contains resources with which to for-
malize labelled graphs (specifically vertex-labelled graphs), bipartite graphs, and graph colorings all
as graph homomorphisms (to a fully connected and self-connected graph \( \hat{K}_\Lambda \) of \( L = |\Lambda| \) distinct
labels \( \Lambda = \{\lambda_{i=1\ldots L}\} \); a 2-node graph without self-connections; and more generally a clique of
\( L \) distinct nodes fully interconnected but without self-connections, respectively). Edge-labelled
graphs (as opposed to the default vertex-labelled graphs) can be had by way of a functor from
graphs to bipartite graphs which maps vertexes to red vertices and edges to blue vertices; the
resulting bipartite graph can be further labelled as needed.

As usual, a tree is an undirected graph without cycles and a Directed Acyclic Graph (DAG)
is a directed graph without directed cycles. A directed rooted tree is a DAG whose undirected
counterpart is a tree, and whose (directed) edges are consistently directed away from (or consis-
tently towards) one root node. Then Abstract Syntax Trees (previously discussed as the substrate
for declarative model transformations) can be represented as node- and edge-labelled directed
rooted trees with symbolic labels for which the finite number of children of any node are totally
ordered (e.g. by extra numerical edge labels), so there is a unique natural “depth-first” mapping
of ASTs to parenthesized strings in a language. Alternatively it is also possible to totally order the
nodes in a tree or DAG in “breadth-first” ways that respect the directed-edge partial order, as in
task scheduling.

The automorphism group of a labelled graph is in general reduced in size from that of the
corresponding unlabelled graph, making it easier to detect particular subgraphs. Some binary
operators of type \( (\text{graph}, \text{graph}) \rightarrow \text{graph} \), namely categorical disjoint sum \( \boxplus_{\text{Graph}} \)
and the graph cross product \( \times_{\text{Graph}} \), can be defined as universal diagrams in the Graph category. The graph box product \( \square \), essential for meshing, can also be formulated
as a tensor product universal diagram but not strictly within the category ((Knauer 2011) Theorem
4.3.5). Graph functions by binary operator \( \rightarrow_{\text{Graph}} \) that produces a graph are also possible but
require an altered definition of the category (Brown et al. 2008).

3.2.2 Non-standard definitions

A special case of a labelled graph is a numbered graph with integer labels \( \lambda'_i = i \), \( |\Lambda'| \geq |V| \), and
the labelled graph homomorphism to \( \hat{K}_\Lambda \) is one-to-one (but not necessarily onto), so each vertex
receives a unique number in \( \Lambda' = \{1\ldots k \geq |V|\} \). Then one way to express a labelled graph
\( G \rightarrow_{\text{Graph}} \hat{K}_\Lambda \) is as the composition \( G \rightarrow_{\text{Graph}} \hat{K}_{\Lambda'} \rightarrow_{\text{Graph}} \hat{K}_\Lambda \) of a graph numbering \( G \rightarrow_{\text{Graph}} \hat{K}_{\Lambda'} \)
followed by a relabelling determined by a mapping of sets \( \Lambda' \rightarrow_{\text{Set}} \Lambda \cup \{\emptyset\} \), with \( \emptyset \) the value
taken on ununused number labels \( i \). The possibility of unused numbers (\( |\Lambda'| \geq |V| \)) will be
needed when consistently labelling two different graphs.
We consider graph homomorphisms to five particular integer-labelled graphs:

\[ \mathbb{N} \equiv (\mathbb{N}, \text{Successor}) = \text{nonnegative integers \{0, 1, \ldots\} as vertices,} \]

with (possibly directed) edges from each integer \( i \) to its immediate successor \( i + 1 \) and to itself;

\[ \hat{J}_D = \begin{cases} (\mathbb{N}_D \equiv \{0, \ldots D \geq 0\}, \geq) & \text{directed graphs;} \\
\mathbb{N}_D = \hat{K}_{\{0, \ldots D\}} & \text{undirected graphs} \end{cases} \]

\[ \hat{N}_D^{\text{op}} = \begin{cases} \text{integers \{0, \ldots D\} with (i, j) edge iff } i = j \pm 1, \quad & \text{directed graphs;} \\
\text{integers \{0, \ldots D\} with (i, j) edge iff } |i - j| \leq 1, \quad & \text{undirected graphs} \end{cases} \]

\[ C_D = \hat{N} \square \hat{J}_D \]

\[ \hat{C}_D = \hat{N} \square \hat{N}_D^{\text{op}} \]

(In this notation the hats \( \hat{\cdot} \) refer to the addition of self-edges.) Graph homomorphisms to these graphs will provide the core of the next few definitions. Following ([Ehrig et al. 2006] Chapter 2), such homomorphisms themselves form useful graph-related categories. These definitions can all be used in either the undirected or directed graph contexts. The two contexts are related by the standard functor in which an undirected edge corresponds to two oppositely directed edges, defining a mapping from undirected graphs to directed graphs.

A graded graph is a homomorphism from a graph \( G \) to \( \mathbb{N} \). Additional assumptions may allow an undirected graded graph to “approach” a continuum topology such as a manifold or other metric space, in the limit of large level number. The metric can be derived via a limit of a diffusion process on graphs. For example, \( D \)-dimensional rectangular meshes approach \( \mathbb{R}^D \) this way (e.g. Mjolsness and Cunha (2012)). A conflicting definition for the term “graded graph”, disallowing connections within a level, was given in (Fomin 1994) but we need both \( \Delta l = 0 \) and \( \Delta l = +1 \) edges. In the category of undirected graphs, \( \Delta l = \pm 1 \) are indistinguishable except by consulting the vertex labels \( l \). Of course, \( (\mathbb{N}, \text{id}_{\mathbb{N}}) \) is itself a graded graph.

A stratified graph of dimension \( D \) is a homomorphism from a graph \( G \) to \( \hat{J}_D \). In the undirected graph case, such a homomorphism is equivalent to a graph whose vertices are labelled by dimension number, since it imposes no constraints on edges. The reason for reversing the edges in the directed graph case, compared to the graded graph definition, is that the level-changing edges then correspond to standard boundary relationships from higher to lower dimensional strata.

Abstract cell complex: A special case of a stratified graph of dimension \( D \) is a graph homomorphism from \( G \) to \( \hat{N}_D^{\text{op}} \), the directed graph of integers \{0, \ldots D\} with self-connections and immediate predecessor connections \( i \rightarrow i - 1 \) within the set. Since \( \hat{N}_D^{\text{op}} \) maps to \( \hat{J}_D \) by graph homomorphism, this is (equivalent to) a special case of stratified graphs.

A graded stratified graph of dimension \( D \) is a graph homomorphism from \( G \) to \( C_D = \hat{N} \square \hat{J}_D \). Two nodes are connected iff either their level numbers are equal (and, in the directed case, the source node dimension is \( \geq \) the target node dimension), or if their level numbers differ by one and their dimensions are equal. Because \( G \) projects to both \( \hat{N} \) and \( \hat{J}_D \), a graded stratified graph maps straightforwardly (by functors) to both a graded graph and to a stratified graph. However, it enforces an additional consistency constraint on level number and dimension: they cannot both
change along one edge (and, in the directed case, the source node dimension must be \( \geq \) the target node dimension).

A graded abstract cell complex of dimension \( D \) is likewise a graph homomorphism from \( G \) to \( \tilde{C}_D = \mathbb{N} \otimes \mathbb{N}_D^{op} \).

Our aim with these definitions is to discretely and computably model continuum stratified spaces (including cell complexes) in the limit of sufficiently large level numbers. Stratified spaces are topological spaces decomposable into manifolds in more general ways than CW cell complexes are. In addition, we would like each manifold to be a differentiable manifold with a metric related to a Laplacian, since that will be the case in modeling spatially extended biological or physical objects.

Thus we arrive at the constructive slice categories \( S(H) : G \to H \) with morphisms \( h : G \to H \) that makes a commutative diagram \( \varphi_G = h \circ \varphi_{G'} \):

\[
\begin{array}{ccc}
G & \xrightarrow{h} & G' \\
\downarrow{\varphi_G} & & \downarrow{\varphi_{G'}} \\
H & \xleftarrow{\chi_G} & H
\end{array}
\]

(Diagram 1)

where \( H = \hat{\mathbb{N}}, \hat{J}_D, \hat{\mathbb{N}}_D^{op}, \hat{C}_D, \text{ or } \hat{C}_D \). Here it is essential that each possible target graph \( H \) includes all self-connections. In particular a homomorphism of graded graphs is a graph homomorphism with \( H = \hat{\mathbb{N}} \), so graded graphs together with level-preserving graph homomorphisms form a slice category.

For a discrete approximation to a stratified space by a stratified graph, we can identify the “graph strata”, and their interconnections by boundary relationships, as connected components of constant dimension. So, by eliminating links between nodes of different dimension, and then finding the connected components that remain, we identify the strata in a stratified graph (or in a stratified labelled graph). In this commutative diagram:

\[
\begin{array}{ccc}
G & \xrightarrow{h} & G_S \\
\downarrow{\chi_G} & & \downarrow{\chi_{G_S}} \\
\hat{J}_D & \xleftarrow{\chi_{G_S}} & \hat{J}_D
\end{array}
\]

(Diagram 2)

each inverse image \( (\chi^{-1}_{G_S})(d) \) must be a fully disconnected graph, with each vertex then corresponding (by \( h^{-1} \)) to a \( d \)-dimensional stratum in \( G \). The directed graph associated to \( G_S \) (directed by dimension number in the case of undirected graphs) is a DAG, due to disconnection within each dimension. The graph homomorphism \( h \) becomes a homomorphism of stratified graphs.

**Observation 1** The resulting \( G_S \) in Diagram 2 is the graph of strata, a minimal structure for modeling complex geometries. *It is therefore a natural graph on which to specify rewriting rules for major structure-changing processes* such as biological cell division, mitochondrial fusing, neurite or cytoskeletal branching and other essential processes of biological development.
The geometry of cytoskeleton, in particular, is better captured by stratified graphs and stratified spaces than by cell complexes, because 1D and 0D cytoskeleton is often embedded directly into 3D cytosol rather than into 2D membrane structures.

**Observation 2** In a graded stratified graph a useful special case occurs if $G_S$ restricted by level number stabilizes beyond a constant number of levels, so the description in terms of strata has a continuum limit $G_S^\infty$. In that case, the limiting $G_S$ is also the natural locus for verification of compatible boundary conditions between PDEs of different dimensionality governing the evolution of biophysical fields defined on continuum-limit strata.

The edges remaining in $G_S$, connecting strata of unequal dimensionality, model in-contact relations such as “boundary” and/or “inside”. Further constraints are needed to disentangle these alternatives. Similar ideas to $G_S^\infty$ may be involved in the “persistent homology” approach to unsupervised learning of data structure (Bendich et al. 2007).

As observed above, the inverse image of a node in $G_S$ identifies the corresponding stratum in $G$. It is natural to use the graph Laplacian on each such stratum to define: (a) geometric distances, using the Green’s function of the Laplacian operator; (b) regularizers for regression of functions from sparsely provided data (cf. Poggio and Girosi (1990)); (c) a Sobolev space $H^2$ of functions in the infinite-graph limit, for biophysical fields; (d) the definition of *functional integrals* in the infinite-graph limit, using a kernel $k_{m,\lambda}(x,y) = (m-\lambda\nabla^2)\delta_{\text{Dirac}}(x-y)$ to form a statistical mechanics partition function such as the Gaussian functional integral (e.g. Mandl and Shaw (2010)):

$$
Z[m,\lambda,J] = \int D[f] \exp\left\{ -\frac{1}{2} \int dxdyf(x)k_{m,\lambda}(x,y)f(y) + \int dxJ(x)f(x) \right\} 
$$

and/or (e) Graph Convolutional Networks, a generalization of deep convolutional neural networks for machine learning from rectangular grids to general graphs (Hammond et al. 2011, Kipf and Welling 2016). But in order to recover the expected continuum properties from the graded graph of meshes associated with a stratum, it may be necessary to reweight the edges of the graph (and hence its Laplacian) according to the local scale of its embedding into for example the three dimensions of flat biological space. Laplacians and their heat kernels on Riemannian manifolds are sufficient to recover the local geometric structure by way of triangulated coordinate systems (Jones et al. 2008). Graph heat kernels converge to the manifold ones in cases where a sequence of graphs “approximate” the manifold (Coifman and Lafon 2006).

Another useful special case of a stratified graph occurs if $G_S$ obeys the constraint that only adjacent dimensions connect. In that case one has the discrete graph structure which we have defined as an abstract cell complex (ACC), although there are several related claimants for that phrase and we make no claim for the superiority of this one. This condition results from using the Hasse diagram for the dimension-labelled strata, with other possible boundary relationships among cells being obtained by multiple steps along $\Delta d = -1$ ACC edges. In most geometric applications one may in addition observe that (b) two strata of dimensionality labels $d-1$ and $d+1$ are always mutually adjacent to either zero or two strata of dimension $d$, though in common with the “abstract complexes” of MGS (Giavitto and Michel 2001) we do not make this part of the definition. ACCs have been used for declarative developmental modeling at least of plant development (Lane 2015) and neural tube development (Spicher 2007), the latter using MGS.
PDEs: If some of the strata in $G_S$ are host to partial differential equations (respecting possibly dynamical boundary conditions at adjacent lower and higher dimensional strata) then $G_S$ will be too minimal for solution algorithms like finite elements or finite volumes, and those strata may have to be meshed. In that case the strata of $G_S$ may need to be subdivided sufficiently finely into patches that can each host a local coordinate system, compatible with its neighboring patches of different dimension and (under one possible strategy) separated by extra artificial boundary strata patches from its neighbors of the same dimension. Compatible finer meshes (computed eg. by programs such as Tetgen (Si and Gartner 2005; Mjolsness and Cunha 2012) could be aligned with the local coordinate systems. As a simpler example, a cell complex graph whose cells embed into 3D as polyhedra can be subdivided into cuboids whose main diagonals each stretch from a vertex of the starting polyhedron to its centroid; but this decomposition may or may not have good numerical properties such as condition number of a meshed mechanical stiffness matrix. A problem for developmental biology, as for fluid simulation, is to maintain mesh quality while the system simulated undergoes large deformations. Meshing for PDEs is a vast field of applied mathematical research to which we cannot do justice; the point here is that stratified graded graphs (and perhaps other graph slice categories) provide a way to formalize many of the problems and capabilities that have to be represented explicitly for declarative modeling to be applied.

Abstract cell complex graphs are also the key structure for Finite Element Methods (Hughes 2000) and for Discrete Exterior Calculus (DEC) discretizations of PDEs (Desbrun et al. 2005, Arnold et al. 2010). DEC allows for the separate discrete representation of $k$-forms and the full exploitation of the generalized Stokes’ theorem, including symplectic PDE integrators and the Helmholtz/Hodge decomposition of function spaces for possible PDE solutions. DEC can be combined with subdivision (de Goes et al. 2016) in the manner of a graded stratified graph. On the other hand, analysis on more general stratified spaces is also possible including differential operators for corners with singularities allowed on the approach to lower dimensional strata (Schulze and Tarkhanov 2003).

By these various means one would like to generalize from ODE-bearing rules to PDE-bearing rules which would be of two basic types: (1) PDEs for the evolution of biophysical fields such as diffusion within a manifold (possibly including a hyperbolic term for finite speed information propagation, as in the Telegrapher’s Equation derivable for stochastic diffusion (Kac 1974) on the way to the Heat Equation), and (2) PDEs for the evolution of the embeddings of strata into higher strata, such as cytoskeleton mechanics (1D into 2D or 3D) or the biomechanics of 2D membranes embedded into ordinary three dimensional Euclidean space. This could be accomplished by way of level sets for example.

3.3 Dynamics on Graphs

There are at least two ways to mathematically define the semantics of graph rewrite rules for use in biological modeling of extended objects. The operator algebra approach, extending the form of rewrite rule semantics given in Section[2] has been used as the theory behind some molecular complex modeling and tissue-level developmental modeling methods. It takes advantage of the ring of operators to blend with continuous-time process models by scalar multiplication of rates and operator addition of parallel processes, thereby also gaining compatibility with quantitative methods of statistical mechanics and field theory in physics, and with machine learning by con-
tinuous optimization. A second approach, the graph homomorphism pushout diagram approach championed in (Ehrig et al. 2006), has been used in molecular complex modeling (Danos et al. 2012). It takes advantage of the category theory formulation of graphs, discussed in Section 3. We will first briefly discuss pushout semantics, since it leverages graph categories but we have little to add to this approach, and then we will discuss operator algebra semantics for graph grammars in continuous time.

### 3.3.1 Pushout semantics

A pure (unlabelled) graph grammar rule $G \rightarrow_K G'$ can replace graph $G$ with graph $G'$, holding common subgraph $K$ constant, anywhere that $G$ (and its subgraph $K$) occurs as a subgraph inside of some “host” graph $C$ within the current pool of one or more graphs (indeed $C$ can be taken to be the entire pool of graphs in the current state since that’s just a big disconnected graph). The result will be an altered version $C'$ of the host or pool graph. Using graph homomorphism arrows, the double pushout diagram for the firing of a graph grammar rule relates all these graphs as follows:

$$
egin{array}{ccc}
G & \xrightarrow{K} & G' \\
\downarrow & \downarrow & \downarrow \\
C & \xrightarrow{D} & C'
\end{array}
$$

(Diagram 3)

The diagram is to be universal at $C$ and $C'$ (so any rival occupant for either position is essentially just a homomorphic image of the universal occupant), hence is double pushout. If such a $C'$ exists, then it is a candidate outcome for the effect of a rule firing of the given rule on the given pool.

What is nice about this diagram is that, as in Section 3.2 above, its objects and arrows can be reinterpreted in other graph-related categories including slice categories such as labelled graphs and other typed attributed graphs. We also imagine that it should be possible to implement a slice graph category rewrite rule under the double pushout semantics in terms of ordinary (double pushout) labelled graph rewrite rule firings, by use of suitable labels. A possible sticking point is verifying implementation of the universalities of the pushout construction in the slice category. In this paper we will only study the analogous question for operator algebra semantics (Section 3.3.3 below).

There is also a closely related single pushout diagram version of the semantics, and a collection of “independence” conditions for two successive rule firings to have an order-independent result (Ehrig et al. 2006).

### 3.3.2 Graph rewrite rule operators

By generating unique integer “Object ID” parameters for each new object created in a parameterized grammar rule, it is possible to implement graph grammar rules by parameterized grammar rules alone, just using repeated OID values to represent graph links. This route was detailed in (Mjolsness and Yosiphon 2006) and implemented declaratively in (Yosiphon 2009). But it is also possible to define graph grammar rewrite rule semantics directly as a continuous-time dynamical system, using operator algebra as in the previous semantics definitions of Section 2.
Suppose we have two labelled graphs \( G_1 : G_1^{\text{pure}} \rightarrow_{\text{Graph}} \hat{K}_{\Lambda_1} \) and \( G_2 : G_2^{\text{pure}} \rightarrow_{\text{Graph}} \hat{K}_{\Lambda_2} \). We decompose them each into a numbered graph \( G_i^{\text{num}} : G_i^{\text{pure}} \rightarrow_{\text{Graph}} \hat{K}_{\{1, \ldots, k_i \vert V_i \}} \) and a relabelling \( \hat{K}_{\{1, \ldots, k_i \vert V_i \}} \rightarrow_{\text{Graph}} \hat{K}_{\Lambda_i} \) determined by a mapping of sets \( \{1, \ldots, k_i \} \rightarrow_{\text{Set}} \Lambda_i \) determined in turn by an ordered listing of labels \( \lambda_i \), possibly augmented by the nullset symbol \( \emptyset \). The whole decomposition can be denoted \( G_i^{\text{num}} \langle \langle \lambda_i \rangle \rangle \). We are interested in graph rewrite rules \( G_i^{\text{num}} \langle \langle \lambda_i \rangle \rangle \rightarrow G'_i^{\text{num}} \langle \langle \lambda' \rangle \rangle \) that respect a single consistent numbering of vertices of the two numbered graphs before their relabellings. In that case vertices in \( G_1 \) and \( G_2 \) that share a vertex number are regarded as the same vertex \( v \), before and after rewriting (similar to the shared graph \( K \) in double pushout), so that any graph edges contacting \( v \) and not mentioned in the rewrite rule are preserved.

Here (Equation (14) below) is an example pertaining to refinement of triangular meshes in 2D. This is one of four rewrite rules that suffice to implement a standard mesh refinement scheme. Three of those rules including this one deal with partially refined triangle edges, an intermediate state produced by the previous refinement of nearby triangles. It can also be interpreted as an (unlabelled) graded graph rewrite rule since it preserves the graded graph constraints on level numbers \( l \), if they are satisfied initially. (The other rewrite rules are similar but deal with the cases of zero, two, or three partially refined triangle sides.) The labelled graph rewrite rule is:

\[
\begin{pmatrix}
\begin{array}{c}
1 \\
4
\end{array} \\
\begin{array}{c}
2 \\
3
\end{array}
\end{pmatrix} 
\langle \langle l_1, l_2, l_3, l_4 \rangle \rangle 
\]

\[
\rightarrow 
\begin{pmatrix}
\begin{array}{c}
1 \\
4
\end{array} \\
\begin{array}{c}
2 \\
5 \\
3
\end{array}
\end{pmatrix} 
\langle \langle l_1, l_2, l_3, l_4, \max(l_1, l_2, l_3, l_4), \\
\max(l_1, l_2, l_3, l_4) \rangle \rangle
\]

Note that in this example there is a shared numbering of nodes of the two graphs, and node numbers 1-4 occur in both graphs. This is equivalent to identifying the shared subgraph \( K \) in the double pushout semantics of Diagram 3. Any extra edges that contact nodes 1-4 in a subgraph of the pool graph, e.g. parts of other nearby triangles, will remain after this rule fires. Of the other three grammar rules, two are analogs of (14) for two or three hanging side nodes like node 4 above; only one rule increases the maximal level number:

\[
\begin{pmatrix}
\begin{array}{c}
1 \\
2 \\
3
\end{array}
\end{pmatrix} 
\langle \langle l_1, l_2, l_3 \rangle \rangle 
\]

\[
\rightarrow 
\begin{pmatrix}
\begin{array}{c}
1 \\
4 \\
6
\end{array} \\
\begin{array}{c}
2 \\
5 \\
3
\end{array}
\end{pmatrix} 
\langle \langle l_1, l_2, l_3, \max(l_1, l_2, l_3) + 1, \\
\max(l_1, l_2, l_3) + 1, \max(l_1, l_2, l_3) + 1 \rangle \rangle
\]

In the resulting four-rule grammar, the order in which triangles are refined does not interfere with the potential refinement of other triangles. If for example a global maximum-level constraint \( l \leq L \) were imposed via the firing rate of each rule, then the refinement process would converge by many alternative paths to the same fully refined terminal triangular mesh.
A more complicated rewrite rule can maintain not only $l =$ level number but also $d =$ dimension in a cell complex representation, during this kind of triangle refinement.

To implement a 2D version of the the polyhedron $\rightarrow$ cuboid mesh refinement scheme mentioned in Section 3.2.2, one could alternatively start with four triangle $\rightarrow$ quadrilateral refinement rules; for example instead of rule (14) one would have:

$$
\begin{align*}
&\begin{pmatrix}
1 & 4 \\
2 & 3 \\
\end{pmatrix} \\
&\rightarrow \\
&\begin{pmatrix}
1 & 4 & 6 \\
2 & 5 & 3 \\
\end{pmatrix}
\end{align*}
$$

$$
\langle\langle l_1, l_2, l_3, l_4 \rangle\rangle
$$

However, one is then committed to designing a larger grammar that can refine both triangles and quadrilaterals with varying numbers of hanging side nodes like node number 4 above, possibly requiring six more quadrilateral refinement rules since there are two ways to place two such side nodes. Abstraction via sub-grammars may then become important. Further notational innovations could also reduce the number of rules needed. This completes a discussion of various particular examples of labelled graph rewrite rules.

In general suppose $G$ and $G'$ are numbered graphs sharing a common numbering of their vertices, with index-ordered adjacency matrices $[g_{pq}|p,q]$ and $[g'_{p'q'}|p',q']$ whose elements take values in $\{0,1\}$. Then an unambiguous graph rewrite rule can be expressed as:

$$
G\langle\langle \lambda \rangle\rangle \rightarrow G'\langle\langle \lambda' \rangle\rangle.
$$

(17)

where the double angle brackets denote label substitutions: $\sigma(1) \mapsto \lambda_1, \sigma(2) \mapsto \lambda_2, \ldots$ and $\sigma'(1) \mapsto \lambda'_1, \sigma'(2) \mapsto \lambda'_2, \ldots$ where $\sigma : \mathbb{N} \rightarrow \mathbb{N}$ and $\sigma' : \mathbb{N} \rightarrow \mathbb{N}$ are strictly monotonically increasing mappings of initial segments of integers into the shared index space.

We give graph grammar rule semantics for the case of directed graphs in terms of binary state vectors for node and edge existence; creation and annihilation operators all have dimension $2 \times 2$. Assuming there is some label in the label space, and provided the global state is initialized to have zero probability of active edges for unused nodes, then the off-diagonal portion of the operator algebra semantics is:

$$
\hat{W}_r \propto \rho_r(\lambda, \lambda') \sum_{\{i_1, \ldots, i_k\}} \prod_{p \in \text{lhs}(r) \setminus \text{rhs}(r)} \prod_i E_{i_p} i E_{i_p} \left[ \prod_{p' \in \text{rhs}(r)} \left( \hat{a}_{i_{p'}q'} \right) \right] \left[ \prod_{p' \in \text{rhs}(r)} \left( \hat{a}_{i_{p'}\lambda'_q} \right) \right] \left[ \prod_{p \in \text{lhs}(r)} a_{i_p \lambda_p} \right] \left[ \prod_{p \in \text{lhs}(r) \setminus \text{rhs}(r)} Z_{i_p \lambda_p} \right].
$$

(18)
For an unlabelled graph, add a single formal label, so $\Lambda = \{\varnothing\}$. The leading (rightmost) operators $Z_{i_p} \lambda_p$ in Equation (18) find unused nodes to build new graph out of, if needed. Their indices $i_p$ should be each averaged rather than summed over, since it doesn’t matter which unused node is brought into service; hence the proportionality rather than equality in (18). If only a finite range of index values $i_p \in \{0, \ldots I_{\text{overflow}} - 1\}$ are possible and they are all used up, then only rewrite rules that don’t create new nodes will be able to fire; one is naturally interested in the infinite storage space limit $I_{\text{overflow}} \to +\infty$ for which this problem doesn’t occur. The trailing (leftmost) “garbage collection” erasure operators $\prod E_i E_{i_p}$ erase any edges $(i_p, i)$ or $(i, i_p)$ dangling from nodes $i_p$ that have just been deleted, making those nodes suitable for reuse in future rule firings. These operators maintain the invariance of the statement that inactive nodes have all their edges inactive as well, which we assumed is true of the initial condition.

Equations (17), (18) provide the built-in syntax and semantics for the basic ruleset portion of a proposed Dynamical Graph Grammar (DGG) generalization of Stochastic Parameterized Grammars (5) and (8) or Dynamical Grammars (5), (10) and (8), (11).

As is the case for Equation (3), Equation (18) can be taken as a normal form for rewrite rule dynamics, but now applied to graphs. We will show next that (somewhat akin to the “Concurrency theorems” of the double pushout approach, but different in detail) the product of two such forms can be rewritten as a (possibly large) integer-weighted sum of expressions having the same form, or a form of equivalent meaning with extra factors of $Z$ and $E$ that don’t affect the active node set; however, some of the weights may be negative.

As a hasty preview of the proof below, this simplification of the product of two graph rewrite rule forms happens because the $Z$ and $E$ factors can be grouped into members of the compound rule’s rhs \ lhs and lhs \ rhs, which commute past everything else, and the remaining factors which are expressed in terms of $a$ and $\hat{a}$ (including negative coefficients from $Z$) and join in the interior graph normal form factors using commutators. The extra factors are indexed by a particular set of indices computed below. In more detail, we have:

**Proposition 1** The product of two operators taking the form of Equation (18) can be rewritten as an integer-weighted sum of expressions taking the same form, except possibly with extra factors of $Z$ and $E$ that don’t affect the active node set.
Proof: The product of two expressions of the form of Equation (18) initially takes the general form

\[
\hat{W}_{r_2} \hat{W}_{r_1} \propto (\rho_{r_1}(\lambda_1, \lambda'_1) \rho_{r_2}(\lambda_2, \lambda'_2)) \\
\times \sum_{\{i_1, \ldots, i_{k_1}\}} \sum_{\{j_1, \ldots, j_{k_2}\}} \left[ \prod_{p \in \text{lhs}(r_2) \setminus \text{rhs}(r_2)} E_{i_p} \right] \left[ \prod_{p \in \text{rhs}(r_2)} E_{i_p} \right] \\
\times \left[ \prod_{p', q' \in \text{rhs}(r_2)} (\hat{a}_{i_p', i_q'})^{g'_{2,p' q'}} \right] \left[ \prod_{p' \in \text{rhs}(r_2)} \hat{a}_{i_p', \lambda'_{2,p'}} \right] \\
\times \left[ \prod_{p, q \in \text{lhs}(r_2)} (a_{i_p q})^{g_{2,p q}} \right] \left[ \prod_{p \in \text{lhs}(r_2)} a_{i_p, \lambda_{2,p}} \right] \\
\times \left[ \prod_{p' \in \text{rhs}(r_1)} \left( \hat{a}_{i_p', i_q'} \right)^{\delta'_{1,p' q'}} \right] \left[ \prod_{p' \in \text{rhs}(r_1)} \hat{a}_{i_p', \lambda'_{1,p'}} \right] \\
\times \left[ \prod_{p, q \in \text{lhs}(r_1)} (a_{i_p q})^{g_{1,p q}} \right] \left[ \prod_{p \in \text{lhs}(r_1)} a_{i_p, \lambda_{1,p}} \right] \left[ \prod_{p \in \text{rhs}(r_1) \setminus \text{lhs}(r_1)} Z_{j_p, \lambda_{1,p}} \right] \\
\times \left[ \prod_{p, q \in \text{lhs}(r_1) \setminus \text{lhs}(r_1)} E_{i_p} \right] \left[ \prod_{p \in \text{lhs}(r_1)} E_{i_p} \right]
\] (19)

This expression should equal a sum of expressions that take the same form as Equation (18), although possibly with negative signs and irrelevant extra factors of \(E\) and \(Z\) as stated, for various choices of graph rule matrices \(g_{1,2v}\) and \(g'_{1,2v}\) \((v\) indexing the summands that result). The leading factors of \(\rho\) multiply properly to give a new leading \(\rho\) for each summand, possibly to be multiplied by integers arising from commutation relations. The \(\sum \sum\) will become a new \(\{i_1, \ldots, i_{k_1}\} \{j_1, \ldots, j_{k_2}\}\) after some nonnegative number of index collisions involving commutators proportional to \(\delta_{i_p, j_{p_2}}\) and \(\delta_{i_{p_1}, j_{p_2}}\) for various \(p\) and \(q\) subindices reduce the number of indices summed over by demanding index equality.

The proof work occurs in two steps: (1) commuting factors of \(E\) in line 5 to join those in line 2 and factors of \(Z\) in line 5 to join those in line 7, and (2) commuting factors of \(\hat{a}\) from line 6 past (to the left of) factors of \(a\) in line 4 to restore normal form, more specifically for (2a) the edges and (2b) the node labels. We will discuss these two steps in reverse order, since Step 2 is simpler.

Recall that all \(a, \hat{a}\) commutators are either zero, when operator types or indices don’t match, or they are diagonal and a linear combination of the identity and a normal form \(N = \hat{a}a\) matrix, multiplied by a delta function that eliminates one or more indices from the sum.

Step 2. Since \(a_{U}^2 = \hat{a}_{U}^2 = 0\) for all indices \(U\), and since \(E\) and \(Z\) are multilinear in the basis \(\{I, a_{U}, \hat{a}_{U}\}\) all indices \(U\), as is the interior of Equation (18) in between its \(E\) and \(Z\) factors, the interior of the resulting fully commuted forms will also all be multilinear in the basis \(\{I, a_{U}, \hat{a}_{U}\}\) all indices \(U\). From that fact one can pick out appropriate matrices \(g_{1,2}\) and \(g'_{1,2}\) on the edge labels. Their entries will all be either 0 or 1 since higher powers \(a_{U}^{l \geq 2} = 0 = \hat{a}_{U}^{l \geq 2}\) contribute nothing to the full operator. Likewise with labels.
Step 1. We now translate the calculation of Step (1) to set notation as follows. For each summand in \( \sum_{i} \sum_{j} \ldots \), the variables \([i_p|p]\) and \([j_q|q]\) take values in some sufficiently large range of integers \(\{0, \ldots, \text{overflow} - 1\}\). On the other hand each subindex \(p, p', \ldots\) (there are several different dummy bound indices with each of these names in Equation (19)) takes values in \(\{1, \ldots, k_1 \leq k_1L + k_1R\}\), and likewise for \(q, q', \ldots\) in \(\{1, \ldots, k_2 \leq k_2L + k_2R\}\). Given fixed mappings \([i_p|p]\) and \([j_q|q]\), these indices now effectively run over the following sets of pool nodes:

\[
\begin{align*}
\text{lhs}(r_1) & \mapsto L_1 & \text{rhs}(r_1) & \mapsto R_1, \\
\text{lhs}(r_2) & \mapsto L_2 & \text{rhs}(r_2) & \mapsto R_2.
\end{align*}
\]

(20)

Fortunately it is only the singly-\((p, q)\)-indexed \(Z_{i_p^*}\) and \(E_{j_q^*}\) operator factors that need to be commuted past other factors to get to their target locations to the rightmost (respectively leftmost) factor collections in the compound rule analog of Equation (18). It should be clear that some of the \(E\) factors commute leftwards unobstructed, due to the delta functions in the commutation relations that give zero commutator whenever subscripts don’t match. This happens for \(E\) terms indexed by members of the set \(L_1 \setminus R_1\) (arising from line 5 in Equation (19)), provided they are not in any of the index sets for graph edges that occur on lines 3 and 4: namely, \(L_2\) or \(R_2\). This set then is \((L_1 \setminus R_1) \setminus L_2 \setminus R_2\). These unobstructed \(E\) factors will always join those already on line 2 of Equation (19) indexed by \((L_2 \setminus R_2)\), to form the union set \(E_{1,2}\) defined in Equation (21) below. The remaining obstructed terms can be fully absorbed into the normal form terms that obstructed them, feeding into Step 2.

In the same manner we calculate the set \(Z_{1,2}\) of factors that always end up in the rightmost collection of \(Z\) factors in line 7 of Equation (19) after commutation; \(Z_{1,2}\) is given in Equation (21) below. It turns out to be related to \(E_{1,2}\) by the time-reversal duality \(L_1 \leftrightarrow R_2, L_2 \leftrightarrow R_1\), which also pertains to the rest of this calculation.

On the other hand, by Equation (18) what we need in these positions in the compound rule is \(L_{1,2} \setminus R_{1,2}\) for the \(E\) factors and \(R_{1,2} \setminus L_{1,2}\) for the \(Z\) factors, where \(L_{1,2}\) corresponds to the LHS of the compound rule \(r_{\text{effective}} = \text{“1,2”}\) (our made-up index to denote the new compound rule). So what are \(L_{1,2}\) and \(R_{1,2}\)? The left hand (input) side set of the compound rule is the left hand side \(L_1\) of the first-acting rule, namely rule 1, together with any inputs to rule 2 that are not supplied by outputs of rule 1 \((L_2 \setminus R_1)\) but instead must be input to the compound rule. Thus \(L_{1,2} = L_1 \cup (L_2 \setminus R_1)\). By similar argument, or by time-reversal duality, \(R_{1,2} = R_2 \cup (R_1 \setminus L_2)\).

So we have these compound rule index set definitions:

\[
\begin{align*}
E_{1,2} &= (L_2 \setminus R_2) \cup ((L_1 \setminus R_1) \setminus L_2 \setminus R_2), \\
Z_{1,2} &= (R_1 \setminus L_1) \cup ((R_2 \setminus L_2) \setminus R_1 \setminus L_1), \\
L_{1,2} &= L_1 \cup (L_2 \setminus R_1), \\
R_{1,2} &= R_2 \cup (R_1 \setminus L_2), \\
\Delta &= (L_2 \setminus R_2) \cap (R_1 \setminus L_1) \equiv \overline{R_2} \cap L_2 \cap R_1 \cap \overline{L_1}.
\end{align*}
\]

(21)

Together they imply by Boolean logic the following (dual) disjoint set unions:

\[
E_{1,2} \equiv L_{1,2} \setminus R_{1,2} \cup \Delta \quad \text{and} \quad Z_{1,2} \equiv R_{1,2} \setminus L_{1,2} \cup \Delta, \quad \text{(22)}
\]

where \(\cup\) denotes disjoint set union. The Boolean logic proving Equation (22) is carried out in full, in the 16-row truth table below (Table I).
Thus, the $E_{1;2}$ and $Z_{1;2}$ sets have all the entries they need to assume the form of Equation (18) for the compound rule - plus a few more, namely those indexed by $\Delta$. What is the meaning of $\Delta$? From (Table I), it is easy to verify that another expression for $\Delta$ is $\Delta = L_2 \cap R_1 \cap L_{1;2} \cap R_{1;2}$. These are the indices of vertices that are neither ingoing to nor outgoing from the composite rule, yet are both outgoing from rule 1 and ingoing to rule 2. They correspond exactly to the “churning” (allocation and immediate deallocation) of extra unused node and edge memory by rule 1 followed by rule 2, that is not needed in the compound rule. (These operators also maintain the invariance of the statement that inactive nodes have all their edges inactive as well.) So the extra factors of $E$ and $Z$ just correspond to an equivalence class of operators that differ only by “churning” extra unused memory.

Thus, after full use of commutation relations, Equation (19) becomes a (weighted, signed) sum of operators each equivalent to a compound rule’s operator in the form of Equation (18). Furthermore the equivalence relation is given by extra factors of $E$ and $Z$ arising from set $\Delta$, as stated in the proposition. QED.

**Corollary 1** The commutator $[\hat{W}_{r_1}, \hat{W}_{r_2}]$ of two operators taking the form of Equation (18) can also be rewritten as an integer-weighted sum of expressions taking the same form, except possibly with extra factors of $Z$ and $E$ that don’t affect the active node set. Compared to the product $\hat{W}_{r_1} \hat{W}_{r_2}$, however, many summands may cancel.

**Observation 3** In this sense (Proposition 1 and Corollary 1), there is a higher-level algebra of graph rewrite rule operators that can be “implemented” by (mapped compositionally by ring homomorphism to) a sufficiently large indexed collection of binary state variables with their own lower-level state-changing operator algebra.

**Observation 4** If instead of the creation and annihilation operators for Boolean state variables we were to use creation and annihilation operators for $N$-valued numbers of identical particles in the definition (18) of $\hat{W}_r$, then in Proposition 1 all the signs from commuting the inner parts of the rule operators would come out $\geq 0$, which is simpler. But (1) the handling of memory allocation and deallocation might have to be revised, and (2) graph grammar rules could have unintended semantics in terms of multigraphs: graphs with nonnegative integer edge weights.

**Conjecture 1** Since the operator algebra and pushout diagram semantics are alternative ways of defining the “same” operation, they must be related. So we conjecture there is a map $\Psi_2$ making Diagram 4 commute:

![Diagram 4](Image)

The conjectured mapping $\Psi_2$ can be defined as transferring probability to the subspace of states of the Fock space corresponding to a $C'$ post-firing pool graph, from the subspace of states of the Fock space corresponding to a $C$ pre-firing pool graph.

**Semantics alternatives.** The semantics of Equation (18) permit several rule nodes (indexed by $p$) to map to one pool node (indexed by $i_p$) if that node is self-connected, as does the pushout rule firing semantics described in Section 3.3.1 (since graph homomorphisms can be many-to-one).
Table 1: Truth table proving Equation (22). False entries are shown as blank for ease of visualization, and every entry is either true or false. Note that the last (16th) line is entirely blank i.e. false. The entries augmenting $E_{1,2}$ and $Z_{1,2}$ by $\Delta$, when $L_{1,2} \setminus R_{1,2}$ and $R_{1,2} \setminus L_{1,2}$ are both false, are boxed.
To prohibit many-to-one node matching in general one could constrain the summed pool indices to all be unequal. To prohibit many-to-one node matching in particular cases one can (a) add differentiating node labels and use the semantics of (18), or (b) alter and generalize the semantics to include factors of $\prod_{LHS}(Z_{ip\ell_q})^{\delta_{pq}}$ where $\delta$ is a 0/1-valued matrix representing a second type of graph edge that identifies prohibited connections on the LHS, and likewise $\prod_{RHS}(Z_{ip'\ell_q'})^{\delta'_{pq'}}$ for the RHS. The normal form could put these new $Z$ product factors just to the right of (acting just before) the corresponding factors for $g$ and $g'$ in Equation (18). If corresponding entries of $g$ and $g'$ both take the value 1, that inconsistency has no effect since their product has a factor $aZ = a(I - \hat{a}a) = a - (\hat{a}\hat{a})a = a - (I - \hat{a}a)a = a - a + \hat{a}a^2 = 0$.

### 3.3.3 Slice Rewrite Rule Operators

The slice categories of Diagram 1, with $H = \hat{N}$, $\hat{D}$, $\hat{N}_D^{\text{op}}$, $C_D$, or $\hat{C}_D$ for graded graph, stratified graph, abstract cell complex, graded stratified graph, and graded abstract cell complex respectively, are all variants of the category of labelled graphs $\varphi : G \rightarrow H$ whose labels $\kappa$ are nodes of $H$, with extra constraints added on the integer-valued labels. We can encode these constraints in each case with a predicate $P_H(\varphi)$, and enforce them with a real-valued “gating” indicator function $\Theta(P_H(\varphi))$ which takes the value 1 if the predicate is satisfied and zero otherwise. If an ordering on the nodes of $G$ is established, as we have assumed, then these objects become $P_H(\kappa)$ and an indicator function $\Theta(P_H(\kappa))$. In the foregoing graph rewrite rule semantics, such an ordering is established by the arbitrary indexing scheme of $p$ and $q$. So we may generalize the graph rewrite rule to cover these slice categories as well by mapping $\rho_{\text{slice}}r(\kappa, \kappa')$ to a corresponding $\rho_{\text{graph}}r((\kappa, \lambda), (\kappa', \lambda'))$:

$$\rho_{\text{graph}}r((\kappa, \lambda), (\kappa', \lambda')) = \Theta(P_H(\kappa)) \times \Theta(P_H(\kappa')) \times \rho_{\text{slice}}r(\lambda, \lambda').$$  \hspace{1cm} (23)

The first indicator function in Equation (23) could be omitted if the initial condition gives nonzero probability only to valid $H$-slice graphs and all rules in the grammar are valid $H$-rewrite rules, maintaining the validity conditions $P_H$ using the second indicator function in in Equation (23). The remaining operator products in Equation (18) can remain the same, yielding the operator algebra semantics of these (and potentially other) slice category rewrite rules, complete with provision for extra domain-model specific labels $\lambda$.

In this way we could implement special graph grammar syntax for slice graph rewrite rules. For example in the case of undirected graded graphs, we could let directed edges represent $\Delta l = +1$ edges and undirected edges represent $\Delta l = 0$. The triangle refinement example would then become:

$$\left( \begin{array}{ccc} 1 & \rightarrow & 4 \\ 2 & & 3 \end{array} \right) \quad \left( \begin{array}{ccc} 1 & \rightarrow & 4 \\ 2 & & 5 \end{array} \right) \quad \left( \begin{array}{ccc} 6 & \rightarrow & 3 \end{array} \right)$$ \hspace{1cm} (24)

Directed graded graphs could be represented too, with just one bit of edge label information to record whether there is a change of level number along a directed edge or not. In both cases the integer-valued level number edge labels are removed, to be restored automatically by an implementation map $I$ from slice graph grammar rule syntax to ordinary graph grammar rule syntax.
(This implementation map could even be expressed in the form of a meta-grammar rule, mapping rules like (24) to a slight variant of rules like (14) with an AST for the labels.) Similarly, for stratified graphs one could label edges with $\Delta d$ and allow the interpretation process to restore $d$. For abstract cell complexes one instead needs only to record one extra bit of edge information regarding $d$: $\Delta d \in \{0, -1\}$. As in the case of rule (14), with these extra edge labels rule (24) could be made more elaborate by retaining all relevant strata and their connections at smaller level numbers rather than just the graded graph “frontier” comprising the deepest substrata in each stratum.

**Observation 5** In all these slice graph category cases the implementation mapping on rewrite rules should match the implementation mapping on their semantics as proposed above, so that slice implementation commutes with semantics:

$$\Psi$$

**3.3.4 Efficient implementations**

We just saw that slice graph grammar rules can be implemented (efficiently) in terms of ordinary labelled graph grammar rules.

The efficient implementation of graph grammars rules themselves can also be considered. We have mentioned that they can be implemented in terms of parameterized grammars with parameters devoted to recording integer-valued ObjectIDs. That implies that worst-case performance for parameterized grammars can be as bad as finding small unlabelled subgraphs in small unlabelled graphs, though finding subgraphs in practice is a lot easier than finding them in worst case, and labels help substantially. So one option is just to deploy algorithms that match small symbolic expressions, or use computer algebra systems that have done the same. But another option is available specifically for declarative modeling: to find the rules with the most commonly occurring rule firings in a model, and to use meta-grammars or a meta-language (discussed in Section 2) to transform those rules into submodels comprising rules taking only special forms that can be compiled into special-case efficient simulation code. Examples of special-form rules amenable to special-case simulation code include parameterless rules, terms with parameter sets that take only a few values, rules that consist only of differential equations, and many other possibilities. Then, use strategies like operator splitting to simulate quickly most of the time, slowing down only for occasional higher-cost operations like cell division in a tissue model.

**3.4 Meta-Hierarchy via Graphs**

If we seek models in discrete mathematics for the idea of a “hierarchy” such as a hierarchy of biological systems and subsystems, or a hierarchy of modeling methods, the simplest possibility is a tree: a directed graph whose undirected counterpart has no cycles. This graph could be labelled with the names of the subsystems, methods, or other concepts in the hierarchy. Such a restrictive definition could be appropriate for a compositional hierarchy, or for a strict classification aimed
at reconstructing clades, but not in general for a hierarchy of specializations or subsets in which a node may have several parent nodes. For that case a less restrictive possibility is to model a hierarchy as a labelled DAG (directed acyclic graph), which has no cycles as a directed graph. Thus a labelled DAG is a natural model for the idea of a hierarchy that is more general than a tree.

However, as the foregoing examples show, a hierarchy may be composed of items related in several different ways (composition, specialization, mutually exclusive specialization, and so on.) This fact suggests a further generalization. If the edge labels in a labelled DAG take values in a further DAG of possible relationships, themselves related by specialization, the resulting compound structure can be called a meta-hierarchy since it encodes a hierarchy of interrelated hierarchies. This kind of structure has precedent in eg. the more general “typed attributed graphs” of (Ehrig et al. 2006).

In Section 5 we will consider a meta-hierarchy whose vertices index into (are labelled by) formal languages for modeling aspects of biology.

4 Model Reduction

Model reduction can be a reasonable strategy to deal with biological complexity. Instead of picking out the most important variables and processes to include in a model a priori, one can include some reasonable representation of many variables and processes (the model still won’t be complete) with reasonable parameter values (easier said than done) in a fine-scale model, and then computationally seek a smaller, coarser scale model that behaves in approximately the same way, on some some set of “observables” or “quantities of interest”, in some relevant region of parameter space.

In addition to eliminating conditionally unnecessary state variables for simplicity’s sake, model reduction has the potential to: (a) enable scaling up to very large models through increased computational efficiency in simulation; (b) mathematically connect predictive models across scales of description for causal authenticity and greater accuracy at each scale; (c) enable the study of the great diversity of possible emergent phenomena, as a function of the parameters, structures, and initial conditions of fine-scale models. Repeated model reduction can result in a hierarchical stack of inter-related models, with which to systematically maximize these advantages.

How can we use machine learning to perform the computational search for reduced models? Given enough data from a pure (parameterless) stochastic chemical reaction network, and the correct structure of the network, it is possible to learn the reaction rates (Wang et al. 2010; Golightly and Wilkinson 2011). In Section 4.1 below we summarize how to learn not just reaction rates but an effective reduced-state space model in the form of a time-varying Boltzmann distribution in at least some examples by following very general principles, for the parameterless case (Johnson et al. 2015) and for the case in which parameters include spatial positions (Ernst et al. 2018).

The general theme of using machine learning to create computationally efficient reduced models is rapidly advancing. In computational chemistry for example (Smith et al. 2017) develop a neural network for learning from, interpolating, and much more efficiently applying the energy and therefore force information computed in Density Functional Theory fine-scale calculations. Likewise, other work (e.g. Burkardt et al. (2006)) addresses difficult problems in fluid flow.

As for the general semantics ($\Psi$) and implementation ($I$) families of structure-respecting mappings, we will denote model reduction mappings by a family symbol $R$. 

28
4.1 Learning Boltzmann Distribution Dynamics

In the parameterless rewrite rule (e.g. a chemical reaction as in Equation (2)) case we learn a coarse-scale approximation \( \tilde{p} \) of \( p \) as a time-varying version of a Boltzmann distribution or Markov Random Field (MRF):

\[
\tilde{p}(s, t; [\mu_\alpha]) = \frac{1}{Z(\mu(t))} \exp \left[ -\sum_\alpha \mu_\alpha(t)V_\alpha(s_i \in C_\alpha) \right].
\] (25)

Here each potential function \( V_\alpha \) is a function of a subset or clique \( C_\alpha \) of the components of \( s \), creating a bipartite “factor graph” of variable-nodes indexed by \( i \) and factor-nodes indexed by \( \alpha \) (Lauritzen 1995; Frey 2003). If the factor graph is not connected then its connected components all factorize into independent probability distributions whose product is \( \tilde{p} \).

To get the dynamics of \( \mu \) one can minimize the KL divergence between \( \tilde{p} \) and \( p \), where \( \tilde{p} \) evolves under a differential equation

\[
\frac{d\mu_\alpha}{dt} = F_\alpha(\mu) \] (26)

whose right hand sides \([F_\alpha] \) can be taken to be a learned combination of a large number of hand-designed basis functions (Johnson et al. 2015; Johnson 2012), optimizing a KL divergence between distributions \( p \) and \( \tilde{p} \). This is the “Graph-Constrained Correlation Dynamics” (GCCD) model reduction method. It was used to achieve a substantial reduction in number of variables for modeling a molecular complex in synapses.

The goal of training for model reduction may be summarized as minimal degradation over time of the approximation on a set of observables:

\[
\begin{align*}
\tilde{p}(t) & \quad \Delta t \quad \tilde{p}(t + \Delta t) \\
R & \quad \simeq O(t) \quad \approx O(t + \Delta t) \quad (Diagram 6)
\end{align*}
\]

Here \( R \) is a restriction operator mapping fine-scale to coarse-scale states, and \( \Delta t \) indicates the passage of time under model dynamics. This diagram can be stacked horizontally, for teacher-forcing model training, or vertically, for application across more than two scales.

This model reduction method can be extended to the case of continuous spatial parameters (Equations (5) and (10)) as follows: Instead of a discrete state vector \( s \) for all the numbers of all the possible (usually molecular) species, we have a representation comprising the total number \( n : \mathbb{N} \) of objects (e.g. molecules) present, indexed by \( i \in \{1, \ldots n\} \), together with an \( n \)-dimensional vector \( \alpha \) of discrete species types \( \alpha_i \) and an \( n \)-dimensional vector \( x \) of continuous spatial parameter vectors \( x_i \) such as \( d = 3 \) dimensions of space (though orientation could contribute \( \binom{d}{2} = 3 \) more components). Then a pure state vector (probability 1 concentrated on one state) can be denoted by the “ket” basis vector \( |n, \alpha, x, t\rangle \), analogous to a single vector \( n \) in GCCD above, and the full state of the system is a probability mixture of the basis states. Similar to GCCD, construct a
coarse-scale mixed state based on an energy function and a Boltzmann distribution that sums over 1-particle contributions \(v_1(\alpha_{i_1}, x_{i_1}, t)\) to the energy, summed over one particle index \(i_1\), plus 2-particle contributions \(v_2((\alpha_{i_1}, \alpha_{i_2}), (x_{i_1}, x_{i_2}), t)\) that obey permutation invariance, summed over two indices \(i_1 < i_2\), and so on up to \(k\)-particle contributions of order \(k = K\):

\[
[[v_k|k \in \{1, \ldots K\}], t] = \sum_{n=0}^{\infty} \sum_{\alpha} \int dx \tilde{p}(n, \alpha, x, t) \mid n, \alpha, x, t\rangle,
\]

with mixture probabilities \(\tilde{p}(n, \alpha, x, t)\):

\[
\tilde{p} = \langle n, \alpha, x, t | [v_k|k], t \rangle = \frac{1}{Z[[v_k|k]]} \exp \left[ - \sum_{k=1}^{K} \sum_{\langle i \rangle^n_k} v_k(a_{\langle i \rangle^n_k}, x_{\langle i \rangle^n_k}, t) \right]
\]

where \(\langle i \rangle^n_k = \{i_1 < i_2 < \cdots < i_k : i \in [1, n]\}\) denotes ordered subsets of \(k\) indexes each in \(\{1, \ldots, n\}\). Here the partition functional \(Z[[v_k|k]]\) has two nested square brackets, the outer brackets indicating that \(Z[\ldots]\) depends on its arguments as a functional depends on functions rather than as a function depends on numbers, and the inner brackets indicating that the \(v_k\) functions indexed by \(k\) should all be included in the argument list.

The goal as in Diagram 6 is to find \(\tilde{p}\) that approximates the solution to the master equation \(\dot{p}(n, \alpha, x, t) = W \cdot p(n, \alpha, x, t)\), where \(W\) sums over all processes that affect the state such as chemical reactions and diffusion (studied in Ernst et al. (2018)) but eventually also active transport, crowding effects and so on.

To define the time evolution of the reduced model \(\tilde{p}(t)\), introduce a set of functionals \([F_k|k \in \{1 \ldots K\}]\) (ODEs, PDEs, or even other forms for \(F\) could be tried) to create a governing dynamical system for the interaction functions \([v_k|k]\):

\[
\frac{d}{dt} v_k(a_{\langle i \rangle^n_k}, x_{\langle i \rangle^n_k}, t) = F_k [[v_k'(\alpha, x, t)||k']].
\]  

(28)

The criterion for choosing the functions \(F\) is to minimize an action that integrates a dissimilarity measure (KL-divergence) between \(p\) and \(\tilde{p}\):

\[
S = \int_{0}^{\infty} dt \; D_{KL}(p||\tilde{p}), \quad \text{where} \quad D_{KL}(p||\tilde{p}) = \sum_{n=0}^{\infty} \sum_{\alpha} \int dx \; p(n, \alpha, x, t) \ln \frac{p(n, \alpha, x, t)}{\tilde{p}(n, \alpha, x, t)}.
\]

(29)

Assuming this objective has been minimized, (Ernst et al. 2018) use the chain rule of calculus to show:

**Proposition 2** Given a reaction network and a fixed collection of \(K\) interaction functions \([v_k]_{k=1}^{K}\), the linearity of the CME in reaction operators \(\dot{p} = \sum_r W^{(r)} p\) extends to the functionals \(F_k = \sum_r F^{(r)}_k\).

In this very limited technical sense, there is a vector space homomorphism (which is a particular kind of structure-respecting map \(R\)) from the spatial operator algebra semantics to the Dynamic Boltzmann reduced model semantics.

### 4.2 Speculation: Expression Dynamics for Model Search

One more significant connection between modeling languages and graph grammars is the possible use of graph grammars for structural model learning. Reaction and rewrite rule model classes
share with artificial neural networks, Markov Random Fields, and many other machine-learnable model classes the property that model architecture is determined by a weighted graph (e.g. the bipartite graph of reactions and reactants) whose structure can be sculpted by setting nonzero weights (e.g. reaction rates) to zero or vice versa, with some expectation of continuity in the neighborhood of zero weight in an optimization formulation of training. This operation would correspond to the deletion or insertion of weak rules. However, other “structural” moves in graph space are bolder and may have higher potential optimization gain. For example if a rule is duplicated while maintaining the sum of the rates, then the model behavior and objective function are unchanged but at least one of the daughter rules may be freed up to drift and adopt new functions. Analogous “duplicate and drift” mechanisms are believed to enable neofunctionalization resulting in selective advantage in genetic evolution (Moore and Purugganan 2003; Thompson et al. 2016). Such an operation is easy to encode with a graph grammar meta-rule.

Duplication with arrow-reversal of one daughter rule, on the other hand, would require for continuity that the unreversed rule retain nearly its full strength and the reversed one enter the ruleset at very small strength. But as discussed in Section 2.1 such a reversed rule could subsequently evolve in strength to establish detailed balance. Other meta-rules could implement the mutation, crossover and three-parent rule-generation operations of genetic algorithms and differential evolution. Semantic word embeddings as used in current machine learning methods for natural language processing may provide a way to learn the inter-substitutability of individual symbols in rewrite rules based on similarity of vectors evolved under previous successful substitutions. This kind of mechanism also promotes the evolution of evolvability. So in addition to the generic metarules we have listed, modeling-language specific or domain-specific rules could be optimized. All of these mechanisms for evolutionary structural optimization of models are local in the model AST and so can be encoded and studied using graph grammars.

5 A Meta-Hierarchy for Declarative Modeling

We have developed the ideas of declarative modeling including various formal modeling languages, together with structure-respecting maps (some of them category morphisms or functors) between formal languages and related mathematical objects for semantics, implementation, and model reduction. We have also defined a “meta-hierarchy” as a DAG whose edges are labelled by a DAG of types of relationships, the relationship types forming a specialization hierarchy.

We propose here a particular extensible meta-hierarchy (as defined in Section 3.4) of formal languages aimed at declarative modeling (as defined in Section 2) of complex scientific domains such as developmental biology. We specify the top levels of the meta-hierarchy but leave lower levels free to evolve with usage. Every node in the meta-hierarchy is either a symbolic placeholder (typical for top level nodes) or represents a formal language or sub-language to be used in a way that satisfies the definition of declarative modeling. The top level nodes in the meta-hierarchy are symbolic placeholders for a classification of deeper-level nodes along two independent labellings: (a) “ontolexical”, in which the labels are “process”, “object”, and “expression” as used in Section 2 and (b) knowledge domain, in which the labels are “science”, “mathematics”, and “computing”. In addition there is a “declarative dynamical model informatics” or simply “models” knowledge domain node, aimed at mediating between the other three. Of course many deep sub-classifications are possible especially for the knowledge domains, beginning with specializa-
tion links to physics, chemistry, and biology, together with reduction links among their further specializations. The resulting meta-hierarchy is named “Tchicoma” after a volcanic formation in the southern Rocky Mountains.

As discussed in Section 3.4, the edges of the meta-hierarchy are labelled by relationship types (e.g. composition vs. specialization, the latter specialized e.g. into mutually exclusive and/or exhaustive vs. overlapping specializations; also proven vs. machine-verifiably proven vs. unproven relationships; and so on) that themselves stand in a specialization hierarchy. Such relationship links could be used in the construction of maps $\Psi$, $\mathcal{I}$, and $\mathcal{R}$ (for semantics, implementation and model reduction respectively) in declarative modeling, e.g. to retrieve similar known maps from previous work. Specialization links can be used to insert conditions that enable theorems and algorithms to work, and to evolve those conditions as knowledge accumulates. Automatic curation of these link types would also provide an opportunity to keep usage-based statistics on the edges of each type at each node in the meta-hierarchy from prior successful model-building activities, for human visualization, for automatic heuristic search, and for targeting the invention of new nodes in the meta-hierarchy to regions with high previous application. New nodes could be specializations or generalizations of single nodes or of several nodes jointly, resulting in a cumulative reculpting of the metahierarchy and its relationship type DAG under pressure of maximal utility.

The languages at the nodes of this meta-hierarchy can be specified by formal grammars, or they can be generated by unary and binary operators defined for all objects in some mathematical category $\mathcal{C}$, including but not limited to category-level binary operators such as universal sum, product, or function arrow that can be defined purely in category-theoretic terms. Such a generated language has the advantage that there is a built-in mathematical semantics taking values in the category $\mathcal{C}$ objects denoted by the operator expression trees. In full generality the detection of semantic equivalence between expressions in such languages is intractable, though it can often be specialized to a solvable problem.

The near-top nodes of the Tchicoma meta-hierarchy comprise a Cartesian product of three ontolexical nodes and four knowledge domain nodes. With the resulting twelve-element cross product classification one can identify the following potential kinds of structure-respecting inter-language mappings, differentiated by top-level source and target positions in the Tchicoma meta-hierarchy:

1. Mappings discussed in this paper:
   
   (a) $\Psi$: Semantics: Mapping from model object and/or process expressions in a modeling language $L$, to mathematical objects. This mapping is an essential part of declarative modeling as defined in Section 2, and was detailed in Sections 2 and 3.
   
   (b) $\mathcal{I}$: Implementation: Mapping from mathematical objects that are the semantics of model objects and/or model processes, to computational objects and processes respectively; or mapping from model expressions to computational expressions that denote these objects and processes; as introduced in Section 3 and illustrated in Diagram 5. Computer Science seeks common target objects for many efficient implementation maps; current examples include Trilinos and PetSc for large-scale scientific computing.
   
   (c) $\mathcal{R}$: Model reduction from model expressions to model expressions with approximately the same semantics under projection to a set of observables; as exemplified in Section 4 and Diagram 6.
(d) \( \mathcal{X} \): Transformation: *From* expressions *to* expressions, preserving or approximating semantics (or projections thereof); an essential part of declarative modeling as defined in Section 2. For example such transformations could include model reduction \( \mathcal{R} \), computational implementation \( \mathcal{I} \), and/or model translation \( \mathcal{T} \) (below).

2. Mappings not detailed in this paper:

(a) \( \mathcal{C} \): Model creation: *From* domain (object, process) expressions in a domain language *to* (object, process) expressions respectively in a modeling language. Domain languages may incorporate ontology formalizations such as the Web Ontology Language (“OWL”) or more general Description Logics; examples include the BioPAX biological pathway description language and ontology (Demir et al. 2010), and the Gene Ontology (GO) three-part hierarchy of biological objects and processes that has an object hierarchy and two process hierarchies at two scales, with “Is-A” specialization links within each hierarchy.

(b) \( \mathcal{T} \): Translation: *From* expressions *to* expressions, preserving or approximating semantics (or projections thereof to observables). E.g. translation could formalize the process of translating between the inputs to different modeling software systems. By defining such translations first at the level of classical (and/or intuitionistic) mathematics rather than software, it is much easier to establish equivalences where they exist. Then one can explore whether and how deeply implementing or updating such translations is worth the effort.

(c) \( \mathcal{A} \): Model analysis (phase diagrams, bifurcation diagrams, and the like): *From* models *to* mathematical analysis products such as reduced-parameter spaces, discontinuity or phase-change strata therein, and long-time asymptotics of observables. Model analysis enables further reduction of submodels, for example by algebraic solution of fast subnetworks.

For most of these kinds of maps other than \( \Psi \), several maps of the same kind could have the same source and target nodes; in that case subscripting the mapping symbols by *mapping subkinds* could become necessary - although one would prefer to elaborate the meta-hierarchy nodes instead, if possible.

The curation of this meta-hierarchy may provide a fruitful application area for automatic theorem verification software, since (a) many of these map types require the assertion of mathematical equivalences and approximations that could be proven, possibly with computer help, in an automatically verifiable form; (b) the applicability of a particular map to a particular modelling problem could be subject to logical inference on applicability conditions; and (c) the retrieval or synthesis of valid map compositions that achieve some formalizable goal could be achieved by forward- and/or reverse-chaining style theorem-proving algorithm. Some of these applications of computer logic could be systematized using type theory and the Curry-Howard-Lambek correspondence which in some circumstances at least maps logical inference to type theory and to category theory [Lambek and Scott 1986]. For “computing” nodes in the meta-hierarchy, and implementation maps that target them, predictive declarative models of computational resource use as a function of problem statement could also be collected and trained on past data.
6 Conclusion

We aim to formalize aspects of mathematical biological modeling so that they become amenable both to computer assistance and to cooperative human development of complex biological models. A conceptual framework, based on declarative modeling, in support of that goal is presented. The elements of the conceptual framework include an informal definition of declarative biological modeling with formalized examples; physics-derived semantics for reaction/rewrite rule process expressions including parameters, variables, graph structure, and differential equations in terms of operator algebras built from creation and annihilation operators; a constructive labelled-graph approach to the semantics of extended objects including graphs labelled by approximation level number and/or stratum dimension and identity, which can approximate and computably implement classical nonconstructive geometries up to stratified spaces; labelled, graded, and stratified graph rewrite rule operator semantics for these constructive extended objects; a potentially generic model reduction framework based on dynamically evolving Boltzmann distributions; and a proposed overarching meta-hierarchy of modeling sub-languages within which the structure-respecting maps required for declarative biological modeling could be defined, curated, and evolved through experience for maximal utility.

This framework could support improved multiscale modeling through more expressive modeling languages and widely applicable model reduction. The framework may provide opportunities for mathematical biologists to contribute to systematic mappings for complex biological model creation, definition, reduction, implementation and analysis in ways that could be greatly amplified by automation and artificially intelligent computational improvement.

Acknowledgements

The author is indebted to many discussants who are not responsible for errors herein. The author wishes to acknowledge the hospitality of the Sainsbury Laboratory Cambridge University, and funding from the Leverhulme Trust, NIH grant R01HD073179, and USAF/DARPA FA8750-14-C-0011.

References

(Abelson et al. 1996) Harold Abelson and Gerald Jay Sussman Structure and Interpretation of Computer Programs - Second Edition MIT Press. Section 1.2.1
(Arnold et al. 2010) Douglas N. Arnold, Richard S. Falk, and Ragnar Winther, “Finite Element Exterior Calculus: From Hodge Theory to Numerical Stability”. Bulletin of the American Mathematical Society. Volume 47, Number 2, Pages 281Ð354, April 2010.
(Bendich et al. 2007) Inferring Local Homology from Sampled Stratified Spaces Paul Bendich, David Cohen-Steiner, Herbert Edelsbrunner, John Harer, Dmitriy Morozov. Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science, pages 536-546, 2007.
(Blinov et al. 2004) Michael L. Blinov, James R. Faeder, Byron Goldstein and William S. Hlavacek? BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. Bioinformatics, Vol. 20 no. 17, pages 3289D3291, 2004.
(Booth and Tillotson 1980) P. Booth and J. Tillotson, Monoidal closed, Cartesian closed and convenient categories of topological spaces. Pacific J. Math. Volume 88, Number 1 (1980), 35-53.

(Brown et al. 2008) R. Brown, I. Morris, J. Shrimpton and C.D. Wensley Graphs of morphisms of graphs the electronic journal of combinatorics 15 (2008), #A1.

(Burkardt et al. 2006) John Burkardt, Max Gunzburger, Hyung-Chun Lee POD and CVT-based reduced-order modeling of Navier-Stokes flows Computer Methods in Applied Mechanics and Engineering Volume 196, Issues 1D3, Pages 337-355, 1 December 2006.

(Cardelli 2008) Luca Cardelli On process rate semantics Theoretical Computer Science 391 (2008) 190–215.

(Coifman and Lafon 2006) Coifman RR, Lafon S Diffusion maps. Appl Comput Harm Anal 21:5D30, 2006.

(Danos et al. 2012) Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, Jonathan Hayman, Jean Krivine, Christopher D. Thompson-Walsh, Glynn Winskel, “Graphs, Rewriting and Pathway Reconstruction for Rule-Based Models”. FSTTCS 2012: 276-288.

(de Goes et al. 2016) Fernando de Goes, Mathieu Desbrun, Mark Meyer, and Tony DeRose, “Subdivision Exterior Calculus for Geometry Processing”. ACM Trans. Graph., 35(4), Art. 133, 2016.

(Demir et al. 2010) Demir, Emek et al. “BioPAX - A Community Standard for Pathway Data Sharing.” Nature biotechnology 28.9 : 935–942, 2010.

(Desbrun et al. 2005) Mathieu Desbrun, Anil N. Hirani, Melvin Leok, Jerrold E. Marsden, “Discrete Exterior Calculus”. arXiv:math/0508341v2, May 2002.

(Doi 1976a) M. Doi, Journal of Physics A: Mathematical and General 9, 1465 (1976).

(Doi 1976b) M. Doi, Journal of Physics A: Mathematical and General 9, 1479 (1976).

(Ermentrout 2004) B. Ermentrout, Simplifying and Reducing Complex Models. In Computational Modeling of Genetic and Biochemical Networks, Bower and Bolouri, eds. MIT Press, 2004.

(Ernst et al. 2018) Oliver K. Ernst, Tom Bartol, Terrence Sejnowski, and Eric Mjolsness. “Learning Dynamic Boltzmann Distributions as Reduced Models of Spatial Chemical Kinetics”, ArXiv 1803.01063, March 2018.

(Fomin 1994) Serbey Fomin, Duality of Graded Graphs. Journal of Algebraic Combinatorics 3, 357-404, 1994.

(Frey 2003) Brendan Frey, “Extending Factor Graphs so as to Unify Directed and Undirected Graphical Models” Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI2003), 2003. Also arXiv:1212.2486.

(Giavitto and Michel 2001) Jean-Louis Giavitto and Olivier Michel, “MGS: A Programming Language for the Transformations of Topological Collections”. LaMI Technical Report 61-2001, Université d’Evry Val d’Essone May 2001.

(Golightly and Wilkinson 2011) Andrew Golightly, Darren J. Wilkinson Bayesian parameter inference for stochastic biochemical network models using particle Markov chain Monte Carlo Interface Focus 2011 1 807-820; DOI: 10.1098/rsfs.2011.0047. 25 October 2011.

(Hammond et al. 2011) David K. Hammond, Pierre Vanderheyst, and Rémi Gribonval, “Wavelets on graphs via spectral graph theory”. Applied and Computational Harmonic Analysis, Elsevier, 2011, 30 (2), pp.129-150.

(Hirsch 1976) Morris Hirsch, Differential Topology, Graduate Texts in Mathematics 33, Springer-Verlag New York, 1976.
(HTT 2013) Homotopy Type Theory: Univalent Foundations of Mathematics 2013
(Hughes 2000) Thomeas J. R. Hughes, Finite Element Methods. Dover 2000.
(Johnson 2012) Gary Todd Johnson, “Dependency Diagrams and Graph- Constrained Correlation Dynamics: New Systems for Probabilistic Graphical Modeling”, PhD thesis, Computer Science Department, University of California, Irvine, March 2012.
(Johnson et al. 2015) Todd Johnson, Thomas Bartol, Terrence Sejnowski, and Eric Mjolsness. “Model Reduction for Stochastic CaMKII Reaction Kinetics in Synapses by Graph-Constrained Correlation Dynamics”, Physical Biology 12:4, July 2015
(Jones et al. 2008) Peter W. Jones, Mauro Maggioni, and Raanan Schul “Manifold parametrizations by eigenfunctions of the Laplacian and heat kernels” Proc. Nat. Acad. Science USA, vol. 105 no. 6 1803-1808, 2008
(Joyner et al. 2012) David Joyner, Ondřej Čertšk, Aaron Meurer, Brian E. Granger, “Open source computer algebra systems: SymPy”. ACM Communications in Computer Algebra. 45 (3/4): 225Ð234, 2012.
(Kac 1974) Mark Kac, “A Stochastic Model Related to the Telegraphers Equation”. Rocky Mountain Journal of Mathematics, Volume 4, Number 3, Summer 1974.
(Kipf and Welling 2016) Thomas N. Kipf, Max Welling, “Semi-Supervised Classification with Graph Convolutional Networks”. arXiv:1609.02907.
(Knauer 2011) Ulrich Knauer Algebraic Graph Theory: Morphisms, Monoids and Matrices De-Gruyter, Sept. 2011 978-3-11-025509-6 Section 4.3, Theorem 4.3.5.
(Lambek and Scott 1986) J. Lambek and P. J. Scott, “Introduction to Higher Order Categorical Logic”, Cambridge University Press 1986.
(Lane 2015) Brendan Lane, “Cell Complexes: The Structure of Space and the Mathematics of Modularity”, PhD thesis, University of Calgary, September 2015.
(Lauritzen 1995) Steffen L. Lauritzen, Graphical Models, Oxford Science Publications 1995.
(Logg et al. 2012) Anders Logg and Kent-Andre Mardal and Garth N. Wells and others, Automated Solution of Differential Equations by the Finite Element Method. Springer, 2012.
(Lovasz 2012) Laszlo Lovasz, Large networks and graph limits American Mathematical Society Colloquium Publications Volume 60. 2012.
(Mandl and Shaw 2010) Mandl and Shaw Quantum Field Theory, 2nd Edition. J. Wiley. Section 13.2, requiring a Wick rotation \( t \rightarrow \pm it \) to relate quantum and statistical field theories.
(Mattis and Glasser 1998) D. C. Mattis and M. L. Glasser, Rev. Mod. Phys. 70, 979 (1998).
(Mironova et al. 2012) Mironova VV, Omelyanchuk NA, Novoselova ES, Doroshkov AV, Kazantsev FV, Kochetov AV, Kolchanov NA, Mjolsness E., Likhoshvai VA. “Combined in silico/in vivo analysis of mechanisms providing for root apical meristem self-organization and maintenance.” Annals of Botany 110:2 pp 349-360 (DOI: 10.1093/aob/mcs069), July 2012.
(Mjolsness et al. 1991) Eric Mjolsness, David H. Sharp, and John Reinitz, “A Connectionist Model of Development”. Journal of Theoretical Biology, vol 152 no 4, pp 429-454, 1991.
(Mjolsness and Yosiphon 2006) Eric Mjolsness and Guy Yosiphon, “Stochastic Process Semantics for Dynamical Grammars”, Annals of Mathematics and Artificial Intelligence, 47(3-4) August 2006.
(Mjolsness et al. 2009) E. Mjolsness, D. Orendorff, P. Chatelain, P. Koumoutsakos, “An Exact Accelerated Stochastic Simulation Algorithm”. Journal of Chemical Physics 130 144110, 2009.
(Mjolsness 2010) “Towards Measurable Types for Dynamical Process Modeling Languages”, Eric Mjolsness. Proceedings of the 26th Conference on Mathematical Foundations of Programming Semantics (MFPS 2010). Electronic Notes in Theoretical Computer Science (ENTCS), vol. 265, pp. 123-144, 6 Sept. 2010, Elsevier.

(Mjolsness and Cunha 2012) Eric Mjolsness and Alexandre Cunha, “Topological object types for morphodynamic modeling languages”. PMA 2012: IEEE Fourth International Symposium on Plant Growth Modeling, Visualization and Applications. Shanghai China, October 2012. IEEE Press.

(Mjolsness 2013) Eric Mjolsness, “Time-ordered product expansions for computational stochastic systems biology”. Physical Biology, v 10, 035009, June 2013.

(Moore and Purugganan 2003) Richard C. Moore and Michael D. Purugganan, “The early stages of duplicate gene evolution”. Proceedings of the National Academy of the United States of America, 100 (26) 15682-15687; December 23, 2003. https://doi.org/10.1073/pnas.2535513100

(Peliti 1985) Peliti, L., J. Phys. France 46, 1469 (1985).

(Perlis 1982) Alan Perlis, epigram number 102, in “Epigrams in Programming”, ACM SIGPLAN September, 1982. Also: https://cpsc.yale.edu/epigrams-programming , accessed March 2018.

(Pierce 2002) Benjamin C. Pierce, Types and Programming Languages. MIT Press 2002.

(Prusinkiewicz and Lindenmeyer 1990) Przemyslaw Prusinkiewicz and Aristed Lindenmeyer Algorithmic Beauty of Plants Springer-Verlag 1990.

(Poggio and Girosi 1990) T. Poggio; F. Girosi “Regularization Algorithms for Learning that are Equivalent to Multilayer Networks” Science, New Series, Vol. 247, No. 4945. , pp. 978-982, Feb. 23, 1990.

(Prusinkiewicz et al 1993) “Animation of Plant Development”, Przemyslaw Prusinkiewicz, Mark S. Hammel, and Eric Mjolsness, SIGGRAPH ’93 Conference Proceedings, ACM 1993.

(Schulze and Tarkhanov 2003) B.-W. Schulze N. Tarkhanov “Differential analysis on stratified spaces” In: Hyperbolic Differential Operators and Related Problems (Eds. Ancona and J. Vaillant), Lecture Notes in Pure and Appl. Math., vol. 233, Marcel Dekker, New York, pp. 157-178, 2003.

(Shapiro et al. 2015) Bruce E. Shapiro and Eric Mjolsness, “Pycellerator: An arrow-based reaction-like modelling language for biological simulations”. Bioinformatics, Oct 26. 2015. DOI: 10.1093/bioinformatics/btv596.

(Shapiro et al. 2003) Bruce E. Shapiro, Andre Levchenko, Elliot M. Meyerowitz, Barbara J. Wold and Eric D. Mjolsness. “Cellerator: Extending a computer algebra system to include biochemical arrows for signal transduction simulations.” Bioinformatics, 19(5):677-678, 2003.

(Shapiro et al. 2013) Bruce E. Shapiro, Elliot Meyerowitz, Eric Mjolsness, “Using Cellzilla for Plant Growth Simulations at the Cellular Level”. Frontiers in Plant Biophysics and Modeling, 4:00408, 2013.
(Si and Gartner 2005) Si H. and Gartner K., Meshing Piecewise Linear Complexes by Constrained Delaunay Tetrahedralizations, Proceeding of the 14th International Meshing Roundtable, September 2005.

(Smith et al. 2017) J. S. Smith O. Isayev and A. E. Roitberg “ANI-1: an extensible neural network potential with DFT accuracy at force field computational cost” Chemical Science Chem. Sci., 8, 3192-3203, 2017.

(Spicher 2007) Antoine Spicher and Olivier Michel, “Declarative modeling of a neurulation-like process”. Bio Systems, vol 87 2-3, pp. 281-8, 2007.

(Steenrod 1967) N. E. Steenrod, “A convenient category of topological spaces”. Michigan Mathematical Journal, Volume 14, Issue 2, 133-152, 1967.

(Thompson et al. 2016) Ammon Thompson, Harold H. Zakon, and Mark Kirkpatrick “Compensatory Drift and the Evolutionary Dynamics of Dosage-Sensitive Duplicate Genes” Genetics, vol. 202 no. 2 765-774; February 1, 2016. https://doi.org/10.1534/genetics.115.178137

(Wang et al. 2010) Yuanfeng Wang, Scott Christley, Eric Mjolsness, and Xiaohui Xie, “Parameter inference for discretely observed stochastic kinetic models using stochastic gradient descent”. BMC Systems Biology 4:99, 2010.

(Weinberger 1994) S. Weinberger, The Topological Classification of Stratified Spaces, University of Chicago Press, 1994.

(Wolfram Research 2017) Author: Wolfram Research, Inc. Title: Mathematica Edition: Version 11 Publisher: Wolfram Research, Inc. Place of publication: Champaign, Illinois Date of publication: 2018

(Yang et al. 2005) Chin-Rang Yang, Bruce E. Shapiro, Eric D. Mjolsness and G. Wesley Hatfield “An enzyme mechanism language for the mathematical modeling of metabolic pathways” Bioinformatics, vol. 21 no. 6 , pages 774-0780, 2005. doi:10.1093/bioinformatics/bti068

(Yosiphon 2009) Guy Yosiphon, “Stochastic Parameterized Grammars: Formalization, Inference, and Modeling Applications”, PhD thesis, UC Irvine Computer Science Department, June 2009.