Stochastic Process Semantics for Dynamical Grammars

Eric Mjolsness and Guy Yosiphon
Department of Computer Science
University of California, Irvine
emj@uci.edu
UCI ICS TR# 06-11

September 25, 2006

Abstract

We define a class of probabilistic models in terms of an operator algebra of stochastic processes, and a representation for this class in terms of stochastic parameterized grammars. A syntactic specification of a grammar is formally mapped to semantics given in terms of a ring of operators, so that composition of grammars corresponds to operator addition or multiplication. The operators are generators for the time-evolution of stochastic processes. The dynamical evolution occurs in continuous time but is related to a corresponding discrete-time dynamics. An expansion of the exponential of such time-evolution operators can be used to derive a variety of simulation algorithms. Within this modeling framework one can express data clustering models, logic programs, ordinary and stochastic differential equations, branching processes, graph grammars, and stochastic chemical reaction kinetics. The mathematical formulation connects these apparently distant fields to one another and to mathematical methods from quantum field theory and operator algebra. Such broad expressiveness makes the framework particularly suitable for applications in machine learning and multiscale scientific modeling.

1 Introduction

Probabilistic models of application domains are central to pattern recognition, machine learning, and scientific modeling in various fields. Consequently, unifying frameworks are likely to be fruitful for one or more of these fields. There are also more technical motivations for pursuing the unification of diverse model types. In multiscale modeling, models of the same system at different scales can have fundamentally different characteristics (e.g. deterministic vs. stochastic) and yet must be placed in a single modeling framework. In machine learning, automated search over a wide variety of model types may be of great advantage. General-purpose modeling languages can also provide software support for the creation of relevant mathematical models. In this paper we propose that Stochastic Parameterized Grammars (SPG's) and their generalization to Dynamical Grammars (DG's) can formally specify dynamical system models within such a unifying framework. These models may be variable-structure dynamical systems, in the sense that the dynamics governs the number of system objects and their relationships over time, in addition to governing the object state variables. To

create such a flexible modeling language, we will define mathematically both the syntax and the semantics of a formal modeling language based on grammar-like collections of rewrite rules.

The essential idea is that there is a "pool" of fully specified parameter-bearing terms such as $\{bacterium(x),\ macrophage(y),\ redbloodcell(z)\}$ where x, y and z might be position vectors. A grammar can include rules such as

$$\{bacterium(x), macrophage(y)\} \rightarrow macrophage(y) \quad \textbf{with} \quad \rho(\|x-y\|)$$

which specify the probability per unit time, ρ, that the macrophage ingests and destroys the bacterium as a function of the distance $\|x-y\|$ between their centers. Sets of such rules are a natural way to specify many processes. They may have more than one term on the left hand side, making them "context sensitive" rather than "context-free". We will define the semantics of grammars composed of such rules by mapping them to stochastic processes in both continuous time (Section 3.4) and discrete time (Section 3.6), and relating the two definitions (Section 3.8). A key feature of the semantic maps is that they are naturally defined in terms of an algebraic *ring* of time evolution operators: they map operator addition and multiplication into independent or strongly dependent compositions of stochastic processes, respectively.

The stochastic process semantics defined here is a mathematical, algebraic object. It is independent of any particular simulation algorithm, though we will discuss (Section 3.7.2) a powerful technique for generating simulation algorithms. The natural continuous-time semantics (Section 3.4) is related to the somewhat more involved discrete-time semantics (Section 3.6) by several propositions in (Section 3.8). We will demonstrate (Section 5.1) the interpretation of certain subclasses of SPG's as a logic programming language. Other applications that will be demonstrated are to data clustering (Section 4.1), chemical reaction kinetics (Section 4.2), graph grammars (Section 5.2), string grammars (Section 5.2.2), systems of ordinary differential equations (Section 5.3), and systems of stochastic differential equations (Section 5.3).

The present paper is a revised and expanded version of the summary presented in [1].

1.1 Related work

Other frameworks describing model classes that may overlap with those described here are numerous. They range from diagrammatic or textual notations for models, to full-scale programming languages, to mathematical objects that describe classes of dynamical systems. The features of posessed by some of these frameworks in common with SPG's and DG's include: time-varying model structure and number of variables; straightforward expression of probabilistic and stochastic models; straightforward expression of graph grammars; continuous and/or discrete time; straightforward expression of logic; type polymorphism; geometry through chain complexes; and operator algebra dynamics, among many other useful characteristics. We offer a tentative classification and comparison as follows.

Dynamical systems frameworks include ordinary differential equation systems, stochastic differential equations, and partial differential equations, all of which are continuous in time and state spaces and constant in their structure; spatial birth and death processes [2], multitype and spatial branching processes [3] and marked point processes [4] which have continuous time but discrete

state-transitions including changes in their number of variables; and cellular automata and branching processes [5] with discrete time and states. The continuous-time dynamical systems among these mathematical objects can be expressed using time evolution operators [6], as we will do for SPG's. However, they do not come with a formal language that straightforwardly embodies the expressive power of logic, nor do they come with the powerful model composition operations (such as parallel rule lists and recursive subgrammar calls) that such a formal language can provide. In addition the semantics of SPG's are not limited to branching processes, but can also express multiple-input/multiple-output events as well as topology-changing processes such as graph grammars.

In Artificial Intelligence (AI), there has been considerable work to extend the Bayes Network (BN) [7] framework in order to handle time-dependence and dynamic model structure. The Continuous Time Bayesian Networks of [8] are a form of BN that expresses a graphical decomposition of a Markov Chain rate table. These networks cannot model a dynamical system structure as can SPG's. The Dynamical Bayesian Network [9] is an extension of BNs for iterative time evolution processes. DBN's depict the probabilistic relations between variables within a time-slice and between adjacent time slices. However, DBN's can only represent a fixed size structures, meaning that objects cannot be created or annihilated. Another limitation of DBN's is that they can represent only discrete temporal processes. In a DBN, time is depicted in terms of fixed intervals and therefore systems which are composed of processes that evolve with different time granularities or systems that obtain evidence over irregular time intervals become intractable for simulation and inference.

The Relational Dynamic Bayesian Networks framework [10] extends DBN's to first order logic domains so as to deal with creation of objects over time and with dynamical structure. Another language for representing probabilistic dynamical systems is BLOG [11]. Both of these frameworks can represent only discrete-time processes. Also there is no composition-preserving mappping like $\Psi_{c/d}$ from syntax to an operator algebra semantics defined for BN's, DBN's, RDBN's, BLOG, or any of the following frameworks; nor is there any relationship derived between continuous-time and discrete-time semantics, as there is for SPG's. In their syntax, only SPG's and MGS (see below) support a polymorphic type system.

A different paradigm is represented by Probabilistic Constraint Nets (PCN) [12] which extends Constraint Networks to model uncertainty over time. PCNs generalize DBNs since they are not restricted to discrete time structure. But as with DBN's, PCN's cannot represent the dynamics and evolution over the number of objects in a system.

Arising in other parts of computer science, the L-systems [13] and MGS [14] frameworks were developed for modeling of biological processes. They both define transformation rules similar to SPG's. However the semantics of these frameworks are defined in terms of simulation algorithms rather than a parallel mathematical semantics as with SPG's. L-systems were extended to include differential equation dynamics as "differential L-systems" [15], but this extension was still deterministic. A similar integration of deterministic grammars and gene regulation networks was proposed as a framework for modeling biological development in [16]. MGS has an expressive capability missing in SPG's and DG's: the built-in ability to represent geometry by d-dimensional topological complexes. Graph grammars provide an avenue to future inclusion of this capability within SPG's.

Process calculi, which were introduced for the study of concurent computation, have also been adapted for applications in biology. Phillips and Cardelli [17] presented an abstract machine mechanism, to perform stochastic simulations from stochastic pi-calculus (SPC) models. The SPG frame-

work is more flexible in its semantics since it is not bound to any specified stochastic simulation technique as is the SPC abstract machine, which is based on the Gillespie stochastic simulation algorithm. Moreover, the SPC abstract machine has no extension for handling objects with continuously-changing parameters as do Dynamical Grammars.

One graphical formalism developed for the analysis of concurrent computation is the Petri Net (PN) [18] which depicts the structure of a distributed system as a directed bipartite graph. This framework models the creation and annihilation of objects by a set of "tokens" which are transmitted between "place" nodes by the firing of "transition" nodes. There are many generalizations of PNs such as Colored PNs [19] or Predicate/Transition Nets [20] that enhance PN with First Order Logic predicates. Since the execution of most forms of PNs is nondeterministic, the most relevant PN extensions are the stochastic or stochastic-colored PNs [21] that augment PNs with rates of execution. Nevertheless, the syntax of stochastic PNs does not contain features such as rule variables, firing rate functions, obliviousness in such firing rates, and type polymorphism as do SPGs, and PNs are defined only for discontinuous state transitions.

The composition-preserving mapping $\Psi_{c/d}$ from the syntax of a powerful modeling language to an operator algebra semantics, and the use of the Time-Ordered Product Expansion to derive a variety of simulation algorithms for such semantics as proposed here, appear to be novel.

1.2 Notation

In the rest of the paper we will use the following definition of a version of the Heaviside function Θ from Boolean values to integers as follows:

$$\Theta(P) \equiv \begin{cases} 1 & \text{if Predicate } P \text{ is true} \\ 0 & \text{otherwise} \end{cases} . \tag{1}$$

Also the Kronecker delta function $\delta_K(a,b)$ or δ_{ab} is

$$\delta_K(a,b) = \Theta(a=b) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

and $\delta(x,y) = \delta(x-y)$ is the Dirac delta (generalized) function appropriate to a particular measure μ on the measure space V. The two definitions coincide for a countable discrete measure space.

In addition to the standard set-builder notation $\{x|P(x)\}$ for defining the members of a set based on a predicate P, we will build ordered sets or lists in a similar way using square brackets. Thus $[x(i)||i \in \mathcal{I}]$, or more generally $[x(i)|P(x(i),i)||i \in \mathcal{I}]$ (read as "$x(i)$ such that $P(x(i),i)$ ordered by $\mathcal{I}$") imposes the image of a preexisting ordering of the index set $\mathcal{I}$ (such as the ordering of natural numbers if $\mathcal{I} \subseteq \mathbb{N}$) onto any elements $x(i)$ selected for inclusion by the predicate P, and thus denotes a set together with a total ordering. For example, the d-tuple $[x_k||k \in \{1,...,d\}]$ denotes the components of the vector $\boldsymbol{x}$. It may be abbreviated as $[x_k]$. Multiple ordering indices are defined recursively or lexically, so for example a "2D Array" $[M_{ij}||\, i,j \in \mathcal{I}]$ is an ordered set of ordered sets, $[[M_{ij}||i \in \mathcal{I}]\ ||j \in \mathcal{I}]$. It may be abbreviated as $[M_{ij}]$.

2 Syntax Definition

In this section we define the syntax of SPG's and of their extension to Dynamical Grammars. SPG semantics will be defined in Section 3, and examples in the rest of the paper.

2.1 Examples

Consider the rewrite rule

$$A_1(x_1), A_2(x_2), ..., A_n(x_n) \rightarrow B_1(y_1), B_2(y_2), ..., B_m(y_m) \quad \textbf{with} \quad \rho([x_i], [y_j]) \tag{2}$$

Here A_k and B_l are *terms* that denote elements τ_a of a set $\mathcal{T} = \{\tau_a | a \in \mathcal{A}\}$, indexed by elements $a \in \mathcal{A}$ of some totally ordered set $\mathcal{A}$. Members of $\mathcal{T}$ are distinct symbols called *types*, all different from one another. The reason Equation 2 is written using terms A_k and B_l rather than directly using the types τ_a is that different terms A and/or B may denote repeated appearances of the same type τ_a. The terms are each optionally followed by parenthesized expressions for parameters x_i or y_j, chosen from a base language $\mathcal{L}_P$ defined below. A term followed by such a parenthesized parameter (or parameter vector) x_i is called a *parameterized term*. We will frequently abbreviate this notation for terms, using "$\tau_a(x_i)$" to denote a parameterized term of type τ_a with a parameter whose value is given by x_i. The terms A_i on the left hand side (the *LHS*) can appear in any order, as can the terms B_j on the right hand side (the *RHS*) of the rule. The intended meaning of the "$A_1(x_1), ..., A_n(x_n) \rightarrow B_1(y_1), ..., B_m(y_m)$" part of this rule is that all the parameterized terms A are instantaneously converted or transformed into the parameterized terms B at some time t. At that moment in time, the rule is said to *fire*.

Also in Equation 2, ρ is a nonnegative function, assumed to be denoted by an expression in a base language $\mathcal{L}_R$ defined below, and also assumed to be an element of a vector space $\mathbb{F}$ of real-valued functions. Its arguments consist of all the parameters from all the parameterized terms on both sides of the rule, LHS and RHS, listed in a standardized order induced by the order on $\mathcal{A}$ (not the arbitrary order induced by the order of the A's and B's within the rule RHS and LHS.) Informally, ρ is interpreted as a nonnegative *probability rate*: the independent probability per unit time that any possible instantiation of the rule will fire if its left hand side precondition remains continuously satisfied for a small time interval. This interpretation will be formalized in the semantics.

As an example of a rule,

$$\begin{aligned} &\text{HydrogenAtom}(x), \text{HydrogenAtom}(y) \rightarrow \text{HydrogenMolecule}(z) \\ &\qquad \textbf{with} \quad f(||x-y||) \quad \exp(-\left(||x-z||^2 + ||y-z||^2\right)/2\sigma^2) \end{aligned} \tag{3}$$

might describe a chemical reaction complete with atomic position vectors x,y and z.

An example of a *stochastic parameterized grammar* built out of rules like Equation 2 or Equation 3 is the following:

grammar (discrete-time) *binaryclustergen* (nodeset(x) $\rightarrow$ {node(x_i)}) {

nodeset(x) $\rightarrow$ node(x) **with** q_0

$\text{nodeset}(x) \rightarrow \text{node}(x), \text{child}(x)$ **with** q_1

$\text{nodeset}(x) \rightarrow \text{node}(x), \text{child}(x), \text{child}(x)$ **with** q_2

$\text{child}(y) \rightarrow \text{nodeset}(x)$ **with** $\phi(x, y)$

}

Informally, the elements of this grammar are a header line and a body within brackets consisting of four rules on separate lines. The order of the rules does not affect the meaning of the grammar, in the semantics to be defined in Section 3. The header line begins with the keyword **grammar** (keywords are typeset in bold) followed optionally by a parenthesized directive (here, "(discrete-time)") that determines whether the continuous-time semantics or the discrete-time semantics is to be used, in case that is not obvious from the context. These two alternative semantic maps are defined and interrelated in Section 3. Then comes the name of the grammar (here, "*binaryclustergen*") followed optionally by a parenthesized rule, the "header rule" (here $\text{nodeset}(x) \rightarrow \{\text{node}(x_i)\}$) whose LHS specifies the inputs of the grammar (the parameterized term "nodeset(x)") which provides the initial condition for an otherwise empty collection or "pool" of parameterized terms (defined in Section 3.1), and whose RHS specifies the outputs of the grammar (a set of "node" terms indexed by i and bearing parameter x_i), which may or may not include the complete contents of the pool of parameterized terms (here it only includes "node" terms and not "nodeset" or "child" terms).

In the body of *binaryclustergen*, firing any of the first three rules allows a nodeset term to be replaced with one parent node and zero, one, or two child nodes, respectively; the rule and hence the number of children is chosen with probabilities proportional to q_n, normalized if necessary. Firing the last rule, on the other hand, replaces a "child" term with a "nodeset" term with a conditional distribution $\phi(x|y) \equiv \phi(x, y)/\int \phi(x, y)dx$ proportional to $\phi(x, y)$ (e.g. a Gaussian in $x - y$) on the child parameter vector x given the parent parameter vector y.

At any given discrete time there may be many possible instantiations of the LHS of each rule that could fire. If distributions q_n and $\phi(x, y)$ are each already normalized, instantiations of the first three rules have the same total probability of firing as instantiations of the last rule. Whether they are normalized or not, q_n is the relative probability for $\text{nodeset}(x)$ to be the LHS of each of the first three rules when they fire, and $\Phi(y) = \int \phi(x, y)dx$ is the relative probability for $\text{child}(y)$ to be the LHS of the last rule when it fires. Thus, the relative probabilities of firing the rules, and of firing them with different possible left hand sides, is determined by the same probability rate functions that determine the relative probabilities of the various possible right hand side rule outputs of a given rule (here, the parameter x in $\phi(x, y)$). Depending on $[q_n]$, this grammar may or may not terminate in a finite number of rule firings and a finite final collection of nodes. In this particular grammar, explicit graph links between children and parents are *not* created but that is also possible using the Object ID mechanism described below (Section 2.4.2).

2.2 Rule Syntax

Next we formalize the language $\mathcal{L}_P$ that constrains the parameterized terms in rules such as Equation 2, and the language $\mathcal{L}_R$ that constrains the rates. This will define the minimal allowed syntax for rules. Also we informally outline the intended semantics for a set of keywords including **with**, **subject to** and others.

We now define $\mathcal{L}_P$. Each parameterized term $A_i(x_i)$ or $B_j(y_j)$ is of type τ_a and its parameters x_i take values in an associated (ordered) Cartesian product set V_a of d_a factor spaces chosen (possibly with repetition) from a set of base spaces $\mathcal{D} = \{D_\beta | \beta \in \mathcal{B}\}$. Each D_β is a measure space with measure μ_β. Particular D_β may for example be isomorphic to the integers $\mathbb{Z}$ with counting measure, or the real numbers $\mathbb{R}$ with Lebesgue measure. The ordered choice of spaces D_β in $V_a = \prod_{k=1}^{d_a} D_{\beta=\gamma(ak)}$ constitutes the type signature $[\gamma_{ak} \in \mathcal{B} | 1 \leqslant k \leqslant d_a]$ of the type τ_a.

Polymorphic argument type signatures are supported by defining a derived type signature $[\sigma_{ab} \in \{0,1\} || a, b \in \mathcal{A}]$, based on elementary factor space compatibilities $[\tilde{\sigma}_{ak\beta} \equiv (D_\beta \subseteq D_{\gamma(ak)}) \in \{T, F\} || 1 \leqslant k \leqslant d_a, \beta \in \mathcal{B}]$. For example we can regard $\mathbb{Z}$ as a subset of $\mathbb{R}$. Then we can define the overall ability to cast type b as a subtype of type a using the 0/1-valued matrix σ_{ab}:

$$\delta_{ab} \leq \sigma_{ab} \leq \Theta(\exists \text{mapping} l(k,b) | \wedge_{1 \leq k \leq d_a} \tilde{\sigma}_{ak\gamma(bl(k,b))}) \tag{4}$$

where

$$\tilde{\sigma}_{ak\beta} \equiv \left(D_\beta \subseteq D_{\gamma(ak)}\right)$$

The simplest case is trivial polymorphism of types to themselves: $\sigma_{ab} = \delta_{ab}$, using the Kronecker delta notation for the identity matrix. Type polymorphism is intended to affect both rule-matching (Section 3.4 below) and grammar recursion (Section 2.4.1 below).

Correspondingly, parameter expressions x_i are tuples of length d_a, such that each component x_{ik} is either a constant in the space $D_{\beta=\gamma(ak)}$, or a variable $X_c (c \in \mathcal{C})$ that is restricted to taking values in that same space $D_{\beta(c)}$. The variables that appear in a rule this way may be repeated any number of times in parameter expressions x_i or y_j within a rule, providing only that all components x_{ik} take values in the same space $D_{\beta=\gamma(ak)}$. A *substitution* $\theta : c \mapsto D_{\beta(c)}$ of values for variables X_c assigns the same value to all appearances of each variable X_c within a rule. Hence each parameter expression x_i takes values in a fixed tuple space V_a under any substitution θ. This defines the language $\mathcal{L}_P$.

We now constrain the language $\mathcal{L}_R$. Each nonnegative function $\rho([x_i], [y_j])$ is a probability rate: the independent probability per unit time that any particular instantiation of the rule will fire, assuming its precondition remains continuously satisfied for a small interval of time. It is a function only of the parameter values denoted by $[x_i]$ and $[y_j]$, and not of time. Each ρ is denoted by an expression in a base language $\mathcal{L}_R$ that is closed under addition and multiplication and contains a countable field of constants, dense in $\mathbb{R}$, such as the rationals or the algebraic numbers. For example $\mathcal{L}_R$ could allow for standard operations such as algebraic expressions built form constants and variables by +, -, *, /, ^p for $p \in \mathbb{R}$, exp, log, and tuple concatenation and projection operations, where these are defined on the relevant spaces D_β . ρ is assumed to be a nonnegative-valued function in a Banach space $\mathcal{F}(V)$ of real-valued functions defined on the Cartesian product space V of all the value spaces $V_{a(i)}$ of the terms appearing in the rule, taken in a standardized order such as nondecreasing order of type index $\in \mathcal{A}$ on the left hand side followed by nondecreasing order of type index a on the right hand side of the rule.

Thus if $a(i)$ is the type of the i'th LHS term and $a'(j)$ is the type of the j'th RHS term, the argument ordering is induced by a slight generalization of the ordered set notation of Section 1.2 :

$$\rho_r\left([x_i | a(i) \in \mathcal{A} || i \in \mathcal{I}_L], [y_j | a'(j) \in \mathcal{A} || j \in \mathcal{I}_R]\right)$$

Terms x_i and $x_{i'}$ that share the same type $a(i)$=$a(i')$ are ordered arbitrarily among themselves, and the function ρ_r must be symmetric with respect to permutations of such terms. For example Equation 3 is symmetric with respect to interchange of x and y, but not x or y with z.

Provided $\mathcal{L}_R$ is expressive enough, it is possible to factor $\rho_r([x_i],[y_j])$ within $\mathcal{L}_R$ as a product ρ_r=$\rho_r^{\text{pure}}([x_i])\Pr_r([y_j]|[x_i])$ of a conditional distribution on output parameters given input parameters $\Pr_r([y_j]|[x_i])$ and a total probability rate $\rho_r^{\text{pure}}([x_i])$ as a function of input parameters only. This factoring may be a useful step for SPG implementations.

With these definitions we can use a more compact notation by eliminating the A's and B's, which denote types, in favor of the types themselves. (The expression $\tau_i(x_i)$ is a parameterized term, which can match to a parameter-bearing *object* or *term instance* in the pool of such objects or terms, defined in Section 3.1. The fine distinction between a parameterized term in a rule and a parameter-bearing term instance in the pool of terms is often omitted.) The problem is that a particular type τ_i may appear in a rule any finite number of times, and indeed a particular parameterized term $\tau_i(x_i)$ may appear any finite number of times. So we use multisets $\{...\tau_{a(i)}(x_i)...\}_*$ (in which the same object $\tau_{a(i)}(x_i)$ may appear as the value of a term for several different values of the index i) for both the LHS and RHS (Left Hand Side and Right Hand Side) of a rule. The asterisk subscript on the brackets means that the object is a multiset rather than a set. If $\mathcal{I}_L$ and $\mathcal{I}_R$ are index sets, we may use set-builder notation $\{\text{element}(i)|\,\text{Predicate}(i)\}$as a meta-language to write a general form for rules with fixed number of RHS and LHS elements:

$$\left\{\tau_{a(i)}(x_i)|i\in\mathcal{I}_L\right\}_* \rightarrow \left\{\tau_{a'(j)}(y_j)|j\in\mathcal{I}_R\right\}_* \quad \textbf{with} \;\; \rho_r([x_i],[y_j]) \tag{5}$$

Here the same object $\tau_{a(i)}(x_i)$ may appear as the value of several different values of the index i under the mappings $i\mapsto(a(i),x_i)$ and/or $i\mapsto(a'(i),y_i)$. Note that the multisets of terms in both RHS and LHS are intrinsically unordered, but the components of a vector parameter within a term are ordered. Either LHS or RHS can be the null multisets, denoted as "$\emptyset$". Finally we introduce the shorthand notation $\tau_i=\tau_{a(i)}$ and $\tau'_j=\tau_{a'(j)}$, and revert to the standard informal notation $\{\}$ for multisets; then we may informally write

$$\{\tau_i(x_i)\}\rightarrow\left\{\tau'_j(y_j)\right\} \qquad \textbf{with} \;\; \rho_r([x_i],[y_j]) \tag{6}$$

In addition to the **with** clause of a rule following the LHS→RHS rule header, several other alternative clauses can be used as follows. "**under** $E(x,y)$" is translated into "**with** $\exp(-E(x,y))/Z(x)$" where $Z(x)$ is the normalizing Boltzmann distribution partition function corresponding to $E(y)$, holding x constant. Equality constraints "**subject to** $f(x,y)$" is translated into "**with** $\delta(f(x,y))$" where δ is an appropriate Dirac or Kronecker delta function that enforces a constraint $f(x,y)=0$. (Inequality constraints such as "**subject to** $f(x,y)\lesseqgtr 0$" and Boolean combinations thereof, may be translated similarly using the $\Theta(P)$ function of Equation 1 above). A convenient synonym for **"subject to"** is **"where"**. Clauses of the form "**via** Γ" and "**substituting** Γ", which will be defined in Section 3.5, are used in order to call a grammar within a grammar as a subroutine or macro, respectively. A rule may have multiple clauses of the same or different keyword; each clause contributes a multiplicative factor to the overall firing rate ρ. In the absence of any clause ρ defaults to unity, for consistency with this multiplicative convention.

The set-builder metalanguage for describing the form of SPG rules is also convenient for specifiying multiple similar rules in a rule schema, all of which belong to a grammar. For example we

would like to admit a rule schema that could replace the first three rules in the *binaryclustergen* grammar:

$$\text{nodeset}(x) \rightarrow \text{node}(x), \{\text{child}(x) | 1 \leqslant i \leqslant n\} \textbf{ with } \; q(n) \; \textbf{ subject to } \; 0 \leqslant n \leqslant 2$$

This can be done by extending the rule language with a set-builder language $\mathcal{L}_S$ as follows: All sets are promoted to multisets; nested multisets (multisets whose elements are multisets) are allowed but flattened to a single level, which is the LHS or RHS multiset; unbound parameters in set-builder notation (such as n above) are treated in the semantics as if the rule were a rule schema expanded out into many copies of the rule with all possible combinations of values substituted in. Of course if the sublanguage $\mathcal{L}_S$ is taken to be trivial, no set-builder notation is allowed and all parameterized terms must be enumerated explicitly. The sublanguages $\mathcal{L}_P$, $\mathcal{L}_R$, and $\mathcal{L}_S$ are "parameters" of a full SPG language and characteristics of any implementation of it.

2.3 Grammar Syntax

A Stochastic Parameterized Grammar (SPG) Γ consists of (minimally) a collection of such rules with common type set $\mathcal{T}$, common base space set $\mathcal{D}$, type signature specification σ, term language $\mathcal{L}_P$, probability rate language $\mathcal{L}_R$, and set-builder language $\mathcal{L}_S$.

As in the example of Section 2.1, the header line of a grammar begins with the keyword **grammar** followed optionally by a parenthesized directive ("(continuous-time)" or "(discrete-time)") that determines whether continuous-time semantics, discrete-time semantics, or some other semantics is to be used. Then comes the name of the grammar followed optionally by a parenthesized rule, the *header rule* whose LHS specifies the inputs of the grammar which provides the initial condition for an otherwise empty pool of parameterized terms, and whose RHS specifies the returnable outputs of the grammar. Thus the optional global "rule header" (LHS→RHS) determines which terms will be considered input and output, as will be described in more detail in Section 3.5 and Section 4 below. Following the rule header comes the grammar body consisting of "{" followed by an unordered list of rules, each on one or more separate lines, followed by a terminating "}". The unordered list is to be interpreted mathematically as a multiset of rules, so that the union of rulesets preserves the total multiplicities of any rules they have in common. This interpretation is required for compositionality of semantics, though rarely would the same rule appear twice in a human-generated grammar, and such multiplicities could just be absorbed into the probability rate functions. After defining the semantics of such grammars, it may be possible to invent "semantically equivalent" classes of SPG's (as defined in Section 5.2.1) which have the same semantics but a different type system, or have richer sublanguages $\mathcal{L}_P$ (perhaps including some of the operations allowed in $\mathcal{L}_R$), $\mathcal{L}_R$, and $\mathcal{L}_S$, or other variants.

2.4 Advanced Grammar Syntax

This section is not essential for all SPG applications.

2.4.1 Syntax of recursion

A rule of the form

$$\left\{\tau_{a(i)}(x_i)|i \in \mathcal{I}_L\right\} \rightarrow \left\{\tau_{a'(j)}(y_j)|j \in \mathcal{I}_R\right\} \quad \textbf{via} \quad \tilde{\Gamma} \tag{7}$$

occurring within another calling grammar Γ or even within the called grammar $\tilde{\Gamma}$, is how one continuous-time grammar can "call" another one, even recursively, in the manner of a subroutine. Because term multisets are unordered, there is no positional notation to specify the mapping of terms and types between calling grammar and called grammar. So any parameterized terms in Γ's rule header that are to be associated with different type names (different elements of $\mathcal{T}$) in the two grammars, should be mapped explicitly.

Consider calling an SPG named $\tilde{\Gamma}$ with header

$$\textbf{grammar}\ \tilde{\Gamma}\ \left(\left\{\tau_{b(i)}(x_i)|i \in \tilde{\mathcal{I}}_L\right\} \rightarrow \left\{\tau_{b'(j)}(y_j)|j \in \tilde{\mathcal{I}}_R\right\}\right)$$

The syntax for mapping of parameterized terms in a **via** call to this grammar is a set of 1:1 term mapping rules:

$$\begin{aligned}&\left\{\tau_{a(i)}\left(x_i\right)|i \in \mathcal{I}_L\right\} \rightarrow \left\{\tau_{a'(j)}\left(y_j\right)|j \in \mathcal{I}_R\right\} \\ &\qquad \textbf{via}\ \ \tilde{\Gamma}\left(\left\{\tau_{a(i)}\left(x_i\right) \rightarrow \tau_{b(i)}\left(\varphi_i\left(x_i\right)\right)|i \in \mathcal{I}_L\right\} \rightarrow \left\{\tau_{b'(j)}\left(y_j\right) \rightarrow \tau_{a'(j)}\left(\tilde{\varphi}_j\left(y_j\right)\right)|j \in \tilde{\mathcal{I}}_R\right\}\right)\end{aligned} \tag{8}$$

or less formally

$$\{\tau_i(x_i)\} \rightarrow \left\{{\tau'}_j(y_j)\right\} \quad \textbf{via}\ \ \tilde{\Gamma}(\{\tau_i(x_i) \rightarrow \sigma_i(\varphi_i(x_i))\} \rightarrow \left\{{\sigma'}_j(y_j) \rightarrow {\tau'}_j(\tilde{\varphi}_j(y_j))\right\}) \quad .$$

Each set of one-to-one rules (LHS and RHS arguments to Γ) can themselves provide a one-to-one or many-to-one type substitution tables in the form of 0/1-valued matrices ${M^{(1)}}_{ba}$ and $\tilde{M}^{(1)}_{ab}$. In addition argument conversion functions $\{\varphi_i|i \in \mathcal{I}_L\}, \{\tilde{\varphi}_j|j \in \tilde{\mathcal{I}}_R\}$ must be provided if necessary by expressions in $\mathcal{L}_R$ to map the correct spaces: $\varphi_i : V_{a(i)} \mapsto V_{b(i)}$ and $\tilde{\varphi}_i : V_{b'(j)} \mapsto V_{a'(j)}$. These could just be identity mappings, permutations, or subpermutations (that drop some components) of the respective argument lists if compatible with the value spaces, for example in accordance with Equation 4 for polymorphic types.

The essential idea is that a transformation from $\{\tau_i\}$ to $\{{\tau'}_j\}$ can be established by straightforwardly mapping τ_i to σ_i, calling another grammar to transform from $\sigma_i(x_i)$ to ${\sigma'}_j(y_j)$, and then straightforwardly argument conversion mapping ${\sigma'}_j$ to ${\tau'}_j$. The straightforward argument conversion mappings φ_i and $\tilde{\varphi}_j$ are associated with tables of allowed type translations. These tables can be summarized as sparse 0/1-valued matrices $M^{(1)}$ and $\tilde{M}^{(1)}$ for which $\mathbf{1} \cdot M \leqslant 1$ and $\mathbf{1} \cdot \tilde{M} \leqslant 1$. These type translations could also be one-to-many, but that would require an extra step in the semantics of stochastically choosing the translation of each type in the semantics of calling a grammar, which we do not define in Section 3.

To the syntax-specified type conversions we may also add any other conversions allowed automatically by type polymorphism matrix $\sigma_{ab} \geq \delta_{ab}$ defined in Section 2.2, using matrix multiplication

to state the constraints for sufficiency of automatic conversion pathways:

$$M^{(2)}{}_{ba} \leq \sum_{cd} \sigma_{bd} M^{(1)}{}_{dc} \sigma_{ca} \qquad \text{and} \qquad \sum_{b} M^{(2)}{}_{ba} \leqslant 1$$
$$\tilde{M}^{(2)}_{ab} \leq \sum_{cd} \sigma_{ac} \tilde{M}^{(1)}_{cd} \sigma_{db} \qquad \text{and} \qquad \sum_{a} \tilde{M}^{(2)}_{ab} \leqslant 1$$

Any single or multiple inheritance type-conversion algorithm that satisfies these inequalities is allowed; it is always possible to satisfy them since $\sigma_{ab} \geq \delta_{ab}$ and one could enforce exact type matches $M^{(2)} = M^{(1)}$ and $\tilde{M}^{(2)} = \tilde{M}^{(1)}$ on the calling syntax. In this way we encode subgrammar argument polymorphism using the M and $\tilde{M}$ matrices.

Any types not occuring in the translation tables are not to be shared between calling and called grammars $\tilde{\Gamma}$ and $\tilde{\Gamma}$. Thus a final step of processing M is to have it push all remaining types onto an inaccessible stack, and for $\tilde{M}$ to pop the stack. Conceptually, represent a larger pool vector space by the vector concantenation ($\tilde{\Gamma}$ pool, Γ and all other calling grammar pools). Then the resulting type translation table is

$$M = M^{(3)} = \begin{pmatrix} M^{(2)} & 0 \\ I - \mathrm{diag}(\mathbf{1} \cdot M^{(2)}) & 0 \end{pmatrix}; \tilde{M} = \tilde{M}^{(3)} = \begin{pmatrix} \tilde{M}^{(2)} & I \\ 0 & 0 \end{pmatrix};$$

thus,

$$\tilde{M}M = \begin{pmatrix} \tilde{M}^{(2)} M^{(2)} + I - \mathrm{diag}(\mathbf{1} \cdot M^{(2)}) & 0 \\ 0 & 0 \end{pmatrix} \leqslant \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} \quad \text{, elementwise.}$$

In this way we also encode nonoverlapping subgrammar contexts using the M and $\tilde{M}$ matrices.

As a syntactic convenience, it frequently happens that types names are to be mapped into themselves. In those cases the inner arrow of the one-to-one mappings can be omitted, as for $i \in \mathcal{I}_{L1}$ and $j \in \mathcal{I}_{R1}$ below:

$$\text{... } \textbf{via } \tilde{\Gamma}(\{\{\tau_{a(i)}(x_i) | i \in \mathcal{I}_{L1}\}, \{\tau_{a(i)}(x_i) \to \tau_{b(i)}(x_i) | i \in \mathcal{I}_{L2}\}\} \to \{\{\tau_{b'(j)}(y_j) | j \in \mathcal{I}_{R1}\}, \{\tau_{b'(j)}(y_j) \to \tau_{a'(j)}(y_j) | j \in \mathcal{I}_{R2}\}\}) \quad (9)$$

(again using the convention of "flattening" nested sets in RHS and LHS) or even skipped entirely (as in Equation 7 above). If all terms are syntactically mapped into themselves, $M^{(1)} = I$ but the polymorphism ($M^{(2)}$) and pool separation ($M^{(3)}$) steps are still in operation.

As in Section 2.2, a single rule could have both **via** and **with** or other clauses, in which case their firing rates are multiplied. For possibly nonconverging SPG's the **via** call can add an extra time argument, so that the syntax of Equation 7 can be "...**via** $\tilde{\Gamma}(t)$".

Another form of reference between grammars is more analogous to the "macros" than subroutines in a programming language: it is similar to inserting the called grammar rules into the calling grammar for possible one-at-a-time rule firings. In this case the syntax uses the clause "**substituting** $\tilde{\Gamma}$", possibly sped up or slowed down by an optional "**with** $\rho(x)$" clause. The syntax of "**substituting** $\tilde{\Gamma}$" analogous to Equation 7 is

$$\{\tau_{a(i)}(x_i) | i \in \mathcal{I}_L\} \to \{\tau_{a'(j)}(y_j) | j \in \mathcal{I}_R\} \quad \textbf{substituting } \tilde{\Gamma} \qquad (10)$$

and likewise for Equation 8 and Equation 9. But unlike a macro, the RHS→LHS syntax constrains the substituting grammar $\tilde{\Gamma}$'s inputs and outputs, again using the M and $\tilde{M}$ type translation tables in matrix form. The semantics of rules using **via** and **substituting** will be defined in Section 3.5, assuming that type translation table matrices M and $\tilde{M}$ have been defined as above.

2.4.2 Syntax of object identifiers (OIDs)

Consider the grammar syntax

grammar (discrete-time) *binaryclustertreegen* (nodeset(x; Null) → {node(x_i)}) {

nodeset(x; G) → P := node(x; G), {child(x; P)|$1 \leqslant i \leqslant n$}

with $q(n)$ **subject to** $0 \leqslant n \leqslant 2$

child(y; P) → nodeset(x; P) **with** $\phi(x|y)$

}

This is a variant of *binaryclustergen* that generates terms with Object Identifiers which act as pointers from child nodes to parent nodes. The novel syntactic form required for parameterized terms using OIDs is

$$P := \tau_{a(i)}(x_i; LP)$$

where P is an OID and LP is an OID or a vector of OID's. In Section 5.2 will show how the semantics of SPG's without OIDs can be extended to the semantics of SPG's with OIDs, by mapping the syntax of the latter to the syntax of the former.

2.4.3 Set-builder quantifiers

A practically useful SPG syntax is the NotExists ($\nexists_i$) quantifier. It prevents a LHS match if the pool contains any parameterized terms of the specified type. As an example, the following rule will be matched only to nodes that have no outgoing edges (thus, only to leaves in a tree structure):

$$\{\text{node}(i,x), \{\nexists_j \text{edge}(i,j)\}\} \rightarrow \{\text{leaf}(i,x)\} \quad \textbf{with} \quad f(x)$$

The semantics of this quantifier and an example are exhibited in Section 5.1.

It can also be useful to allow an unspecified number of objects on the LHS by means of a universal For-All ($\forall_i$) quantifier, which matches to the set of *all* parameterized terms in the pool that fit the pattern. For example the following rule should remove simultaneously all the edges outgoing from node i, with a probability rate function depending on the node's location:

$$\{\text{node}(i,x), \{\forall_j \text{edge}(i,j)\}\} \rightarrow \text{node}(i,x) \quad \textbf{with} \quad f(x)$$

The universal quantifier raises theoretical difficulties for operator semantics that we will pose (Section 5.1.1) but not solve in this paper.

2.5 Dynamical Grammars and Implementation

2.5.1 Dynamical Grammar syntax

Dynamical Grammars are defined as a generalization of Stochastic Parameterized Grammars that include rules with **solving** clauses:

$$\{\tau_i(x_i)|i \in \mathcal{I}_L\} \rightarrow \{\tau_i(x_i)|i \in \mathcal{I}_L\} \quad \textbf{solving} \left\{ \frac{dx_i}{dt} = F_i([x_j|j \in \mathcal{I}_L])|i \in \mathcal{I}_L \right\} \tag{11}$$

where the square brackets in the argument to F_i refer to ordered rather than unordered function arguments, and each F_i is an expression in $\mathcal{L}_R$ denoting a function in $\mathcal{F}(V)$. Such clauses may be used in order to incorporate Ordinary Differential Equations (ODEs, as in differential L-systems [11]), Partial Differential Equations (PDEs), and Stochastic Differential Equations (SDEs), within a Dynamical Grammar. The ODE case is illustrated above. Their semantics will be defined in terms of the semantics of grammars containing **with** clauses, along with extra time-evolution operators, in Section 5.3. In this way, Dynamical Grammars can specify members of a very general class of dynamical systems.

2.5.2 Implementation

We have created a preliminary implementation of an interpreter for most of this syntax in the form of a *Mathematica* notebook called "Plenum", which draws samples according to the continuous-time semantics of Section 3 below. The current implementation includes **with**, **under**, **subject to**, **solving**, $\nexists$, and $\forall$, and a limited form of **via**, but not **substituting** or type polymorphism.

3 Semantic Maps

We provide a semantics function $\Psi_c(\Gamma)$ as an algebraic construction that results in a dynamical system in the form of a stochastic process, if it exists, or a special "undefined" element if the stochastic process doesn't exist (Sections 3.1 and 3.4 below). The stochastic process is defined by a very high-dimensional differential equation (the master equation) for the evolution of a probability distribution in continuous time. On the other hand we will also provide a semantics function $\Psi_d(\Gamma)$ that results in a discrete-time stochastic process for the same grammar, in the form of an operator that evolves the probability distribution forward by one discrete rule-firing event (Sections 3.1 and 3.6 below). In each case the stochastic process specifies the time evolution of a probability distribution over the contents of a "pool" of grounded parameterized terms $\tau_a(x_a)$ each of which can be present in the pool with any allowed multiplicity from zero (not present) to $n_a^{\max}$. We will relate these two alternative "meanings" of an SPG, $\Psi_c(\Gamma)$ in continuous time and $\Psi_d(\Gamma)$ in discrete time, in Section 3.8.

Both semantic maps are given in terms of operator algebra. Starting with the grammar we construct a linear mapping from a probability distribution over states at one time to a function proportional to the probability distribution over states at a later time. The mapping is constructed

by algebraic operations (operator addition and multiplication, and scalar-operator mulitplication) from more elementary linear mappings. To do so we need to define the states.

3.1 System states and master equation

A *pool state* or state of the pool of term instances is defined as a nonnegative-integer-valued function $n : \mathcal{V} \to \mathbb{Z}^* = \{0, 1, 2, ...\}$. It is the "number of copies" $n_a(x_a) \in \{0, 1, 2, ...\}$ of each parameterized term $\tau_a(x_a)$ that is grounded (has no variable symbols X_c), for any combination $(a, x_a) \in \mathcal{V} = \coprod_{a \in \mathcal{A}} a \otimes V_a$ of a type index $a \in \mathcal{A}$ and a parameter value $x_a \in V_a$. We may denote this pool state by $\{n_a(x)\}$, as a shorthand for such functions. The pool itself may be identified with the set $\{(a, x_a, n_a(x_a))|n_a(x_a)) \neq 0\}$, i.e. those term instances that are present in a pool state. Each type τ_a may be assigned a maximum possible value $n_a^{(\max)}$ for all $n_a(x_a)$, commonly ∞ (no constraint on copy numbers) or 1 (so $n_a(x_a) \in \{0, 1\}$ which means each term-value combination is simply present or absent). The *system state* or state of the full system at time t is defined as a probability distribution on all possible values of this (already large) pool state: $\Pr(\{n_a(x_a)|(a, x_a) \in \mathcal{V}\}; t) \equiv \Pr(\{n_a(x_a)\}; t)$. The probability distribution concentrated on one particular pool state $\{n_a(x_a)\}$ is called a *pure state* of the dynamical system and denoted $|\{n_a(x_a)\}\rangle$. A probability distributions that is not a pure state is called a *mixed state* of the system.

For continuous-time we define the *semantics* $\Psi_c(\Gamma)$ of our grammar as the unique solution, if one exists, of the following differential equation:

$$\begin{gathered}\frac{d}{dt}\Pr(\{n_a(x)\}\,;t) = \sum_{\{m_a(x)\}} H_{\{n\}\{m\}} \Pr(\{m_a(x)\}\,;t), \quad i.e. \text{ in matrix notation} \\ \frac{d}{dt}\Pr(t) = H \cdot \Pr(t)\end{gathered} \tag{12}$$

starting from any initial condition $\Pr(0)$. This is called the *master equation* [25]; see also [6]. It has the formal solution

$$\Pr(t) = \exp(tH) \cdot \Pr(0). \tag{13}$$

There may not be a unique solution of the master equation for all times $t > 0$, then we define the *definition limit* for Γ and $\Pr(0)$ as the the least upper bound $T \in [0, +\infty]$ of times $T' \in [0, +\infty)$ for which there is a unique solution for all times $t \in [0, T']$ starting from initial condition $\Pr(0)$. T exists and is at least zero since the initial condition itself is the unique solution on $[0, 0] = \{0\}$. Furthermore for every nonnegative integer k, there is a unique solution on $[0, (1 - 1/k) \times T]$. For each initial condition $\Pr(0)$, we define $\Psi_c(\Gamma)$ to be (a) the common unique solution on $t \in [0, T) = \bigcup_{k=1}^{\infty}[0, (1 - 1/k) \times T]$; concatenated with (b) a special "not really defined" symbol (such as "$\perp$") thereafter for $t \in [T, +\infty]$. We do not attempt to maintain "partial definedness" for mixed states that include nonzero weights for pure states for which the master equation has a unique solution at time t as well as pure states for which the master equation has no unique solution at time t, so this is a fairly conservative definition of $\Psi_c(\Gamma)$. The operator H will be defined in Section 3.4, completing the definition of $\Psi_c(\Gamma)$.

For *discrete-time semantics* $\Psi_d(\Gamma)$ there is some probability update map U which acts on probability vectors, in a manner disignated by "$\circ$", to evolve them forward by one rule-firing time step. Then after k discrete time steps or rule-firings the probability is:

$$\Pr(k) = U \circ ...U \circ \Pr(0) \equiv U^k \circ \Pr(0) \tag{14}$$

which, taken over all $k \geqslant 0$ and $\Pr(\{n_a(x)\}; 0)$, defines $\Psi_d(\Gamma)$. The operator U will be defined in Section 3.6, completing the definition of $\Psi_d(\Gamma)$. Both H and U will be determined by an operator $\hat{H}$ computed from the SPG syntax.

In both continuous-time and discrete cases the long-time evolution of the system may converge to a limiting distribution, e.g. $\Psi_c^*(\Gamma) \cdot \Pr(0) = \lim_{t\to\infty} \Pr(\{n_a(x)\}; t)$, which is a key feature of the semantics. But we do not define the semantics $\Psi_{c/d}(\Gamma)$ as being only this long-time limit even if it exists. Thus semantics-preserving transformations of grammars are fixedpoint-preserving transformations of grammars but the converse may not be true.

Fortunately, even though the mathematical objects just defined are large, they are completely determined by the *generators* H and $\hat{H}$ which in turn are simply composed from elementary operators acting on the space of such probability distributions. Indeed they are elements, or limits of elements, of the operator polynomial ring $\mathbb{R}[\{B_\alpha\}]$ defined over a set of basis operators $\{B_\alpha\}$ in terms of operator addition, scalar multiplication, and noncommutative operator multiplication. These basis operators $\{B_\alpha\}$ provide elementary manipulations of the copy numbers $n_a(x)$. The operator algebra is meaningful: operator addition corresponds to composition of parallel processes, nonnegative scalar multiplication corresponds to speeding up or slowing down a process (as is done in the product of scalar rate functions from different clauses in a single rule), and operator multiplication corresponds to the obligatory co-occurrence of the constituent events that define a process, in immediate succession. Commutation relations between operators describe the exact extent to which the order of event occurrence matters.

3.2 Probabilistic Fock spaces

As a foundation, the function space that probability distributions $\Pr(\{n_a(x_a)\})$ occupy may be formalized as follows. (This subsection is not essential to understanding the applications presented later.)

From Section 2.2 we know that each value space V_a is a measure space, with a σ-algebra σ_a of "events" on which probability is to be defined. A probability distribution on a measure space X (such as V_a in Section 2.2) is just a nonnegative measure P on the σ-algebra for which $P(X) = 1$. We now construct a probabilistic version of a many-particle "symmetric Fock space" following [26]. For any nonnegative integer n_a we can can define the set of states that have a total a total of n_a "copies" of grounded parameterized term $\tau_a(x_a)$, as the permutation-symmetrization of the Cartesian product of n_a copies of V_a:

$$f_a(n_a) = \left(\bigotimes_{m=1}^{n_a} V_a \right) / \mathcal{S}(n_a) \quad .$$

Here $\mathcal{S}(n)$ is the symmetric group on n items. The division sign produces equivalence classes of Cartesian-product members that differ only by a permutation of n_a items. The idea here is that instantiated terms don't have individual identities aside from the values of their parameters; two terms of the same type and value are equivalent. A new σ-algebra is induced on the space $f_a(n_a)$ by the Cartesian product operation and another new σ-algebra is induced by the symmetrization operation. Next, any finite nonnegative number n_a of terms are allowed in a disjoint union of

measure spaces $f_a(n_a)$, and the construction is repeated in a cross product over for all term types a:

$$f_a = \bigoplus_{n_a=0}^{\infty} f_a(n_a) \text{ and } f = \bigotimes_a f_a$$

Now f is a measure space (since it has an induced σ-algebra) and thus defines a *probabilistic Fock space* $\mathcal{F}$ as the set of probability distributions defined on f. In this way we arrive at a symmetric Fock space for probability distributions. It is comparable to the usual construction in quantum mechanics which produces a Hilbert space of probability amplitude functions for many-particle systems, except that probability distrubutions do not require the Hilbert space framework as quantum amplitudes do. This probabilistic version of a Fock space is suitable for defining probability distibutions over the sets of copy numbers that label pure states $|\{n_a(x_a)\}\rangle$.

3.3 Operator algebra

The simplest basis operators $\{B_\alpha\}$ are elementary creation operators $\{\hat{a}_a(x)|a \in \mathcal{A} \wedge x \in V_a\}$ and annihilation operators $\{a_a(x)|a \in \mathcal{A} \wedge x \in V_a\}$ that increase or decrease each copy number $n_a(x)$ in a particular way (reviewed in [27]):

$$\hat{a}_a(x)|\{n_b(y)\}\rangle = |\{n_b(y) + \delta_K(a,b)\delta(x,y)\}\rangle \tag{15}$$

$$a_a(x)|\{n_b(y)\}\rangle = n_a(x)|\{n_b(y) - \delta_K(a,b)\delta(x,y)\}\rangle \tag{16}$$

where $\delta_K(a,b)$ is the Kronecker delta function (defined in Section 1.2). These two operator types then generate $N_a(x) = \hat{a}_a(x)a_a(x)$

$$N_a(x)|\{n_b(y)\}\rangle = \hat{a}_a(x)a_a(x)|\{n_b(y)\}\rangle = n_a(x)|\{n_b(y)\}\rangle\,,$$

and they satisfy

$$[a_a(x), \hat{a}_b(y)] \equiv \textit{commutator of } a \text{ and } \hat{a} \equiv a_a(x)\hat{a}_b(y) - \hat{a}_b(y)a_a(x)$$
$$= 0 \quad \text{if} \quad a \neq b \text{ or } x \neq y.$$

We can write these operators $\hat{a}, a$ as finite or infinite dimensional matrices depending on the maximum copy number $n_a^{(\max)}$ for type τ_a. If $n_a^{(\max)}=1$ (for a fermionic term), and we if omit the type and value subscripts which are all assumed equal and discrete below, then

$$\hat{a} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, a = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

$$\{a, \hat{a}\} \equiv \textit{anticommutator of } a \text{ and } \hat{a} \equiv a\hat{a} + \hat{a}a$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I;\ \hat{a}a = N \equiv \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

These 2×2 matrices can be interpreted as follows. They operate on the two-dimensional vector space of probabilities $(p(0), p(1))$ that the number of objects present is $n = 0$ or $n = 1$. They do not in general conserve total probability, so this is the positive orthant of a two-dimensional space.

The operator a moves all the probability $p(n=1)$ to the $n=0$ state, i.e. destroys an object, and it simply eliminates the original probability $p(n=0)$ from the system:

$$a \begin{pmatrix} q \\ p \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} q \\ p \end{pmatrix} = \begin{pmatrix} p \\ 0 \end{pmatrix}$$

Similarly $\hat{a}$ creates an object but doesn't conserve probability. Probability conservation will be restored using more complex operators built out of these fundamental ones. Here and below, the matrix rows and columns are indexed by number n of indistinguishable objects (number of copies of an object of a given type a and parameter x) immediately before and after an operator is applied.

Likewise if $n_a^{(\max)}{=}\infty$ (for a bosonic term),

$$\hat{a} = \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots \\ 1 & 0 & 0 & 0 & \\ 0 & 1 & 0 & 0 & \\ 0 & 0 & 1 & 0 & \\ \vdots & & & \ddots & \ddots \end{pmatrix} = \delta_{n,m+1} \quad \text{and} \quad a = \begin{pmatrix} 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 2 & 0 & \\ 0 & 0 & 0 & 3 & \\ 0 & 0 & 0 & 0 & \ddots \\ \vdots & & & & \ddots \end{pmatrix} = m\delta_{n+1,m},$$

and

$$[a, \hat{a}] \equiv (a\hat{a} - \hat{a}a) = I = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots \\ 0 & 1 & 0 & 0 & \\ 0 & 0 & 1 & 0 & \\ 0 & 0 & 0 & 1 & \\ \vdots & & & & \ddots \end{pmatrix}; \hat{a}a = N_a \equiv \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots \\ 0 & 1 & 0 & 0 & \\ 0 & 0 & 2 & 0 & \\ 0 & 0 & 0 & 3 & \\ \vdots & & & & \ddots \end{pmatrix}.$$

By truncating these matrices to finite size $n^{(\max)} < \infty$ we may compute that for some polynomial $Q(N|n^{(\max)})$ of degree $n^{(\max)}$-1 in N with rational coefficients,

$$[a, \hat{a}] = I + NQ(N|n^{(\max)}).$$

Eg. if $n^{(\max)}{=}1$ then $Q = -2$; if $n^{(\max)}{=}\infty$ then $Q = 0$. If the parameters x are continuous e.g. real-valued, then the general *commutator* relation becomes

$$[a(x), \hat{a}(y)] = \delta(x-y)[I + NQ(N|n^{(\max)})] \tag{17}$$

where δ is again the Dirac delta (generalized) function appropriate to the (product) measure μ on the relevant value space V.

For any measure space of parameter values x, and for any $n^{(\max)}$, the set of all operators of the form $a(x)$ and $\hat{a}(x)$ generate an *algebra* over the real numbers by scalar multiplicaton and operator-operator addition and multiplication. This algebra is associative. The *commutators* listed above (in Equation 17) suffice to derive *commutator* expressions for all pairs of operators in this algebra; thus they provide a specification of the *Lie algebra* associated with the operator algebra, in which all operator triples satisfy the Jacobi identity $[A,[B,C]] + [B,[C,A]] + [C,[A,B]] = 0$. The importance of the *commutator* for dynamical system applications is that it characterizes the nature of the noncommutation and hence interdependence (or interference) of different kinds of

time-evolution operators that occur in the same dynamical system, as will be illustrated in Section 3.7.1 . (*Anticommutators* play an analogous role in "Lie superalgebras" that arise in supersymmetric particle theories and elsewhere in physics, and are potentially important since we frequently use the special case $n^{(\max)} = 1$.) Thus, *commutators* and commutation relations are fundamental to the operator formulation of dynamical systems.

3.4 Continuous-time semantics

For a grammar rule number "r" of the form of (Equation 5) we define the operator that first (instantaneously) destroys all parameterized terms on the LHS and then (immediately and instantaneously) creates all parameterized terms on the RHS. This happens independently of time or other terms in the pool. Assuming that the parameter expressions x, y contain no variables X_c, the effect of this event is:

$$\hat{O}_r = \rho_r([x_i], [y_j]) \left[\prod_{i \in \text{rhs}(r)} \hat{a}_{a(i)}(x_i) \right] \left[\prod_{j \in \text{lhs}(r)} a_{b(j)}(y_j) \right] \tag{18}$$

The operators within each of the two products above commute, so their order within each product is arbitrary. If there are variables $\{X_c\}$, we must sum or integrate over all their possible values in $\bigotimes_c D_{\beta(c)}$:

$$\hat{O}_r = \int_{D_{\beta(1)}} ... \int_{D_{\beta(c)}} ... \left(\prod_c d\mu_{\beta(c)}(X_c) \right) \rho_r([x_i(\{X_c\})], [y_j(\{X_c\})]) \times \left[\prod_{i \in \text{rhs}(r)} \hat{a}_{a(i)}(x_i(\{X_c\})) \right] \left[\prod_{j \in \text{lhs}(r)} a_{b(j)}(y_j(\{X_c\})) \right] \tag{19}$$

Thus, syntactic variable-binding has the semantics of multiple integration. This is the same result one would get if each rule with variables were replaced with a (finite, countable, or uncountably infinite) set of rules with all possible values substituted in for all the variables, with firing rates weighted by the relevant measure, and running in parallel.

For the same reason, nontrivial polymorphism of rule input types (Section 2.2) may be supported using summation over all possible subtypes d of each rule input type $b(j)$, for which $\sigma_{b(j)d} = 1$, as follows:

$$\hat{O}_r = \int_{D_{\beta(1)}} ... \int_{D_{\beta(c)}} ... \left(\prod_c d\mu_{\beta(c)}(X_c) \right) \rho_r([x_i(\{X_c\})], [y_j(\{X_c\})]) \times \left\{ \prod_{i \in \text{rhs}(r)} \hat{a}_{a(i)}(x_i(\{X_c\})) \right\} \left\{ \prod_{j \in \text{lhs}(r)} \left[\sum_d \sigma_{b(j)d} a_d(y_j(\{X_c\})) \right] \right\}. \tag{20}$$

Again summation gives the same semantics as rule replication would. If σ is the identity matrix, this equation reduces to Equation 20.

Constructed this way, the semantics for each rule is *oblivious* in that every possible rule firing has a probability per unit time which does not depend on the number of other possible rule firings of the same or different rules. The increasing integer entries of the annihilation operators ensure this property. This is also true of chemical reaction networks but not, for example, of multi-token Petri net transitions. For example a rule that requires as input exactly two copies of a given parameterless term τ_a, finds its probability rate function multiplied by $n_a(n_a-1) \equiv (n_a)_2$ (from two powers of the annihilation operator) which is proportional to $\binom{n_a}{2}$, the number of ways those two inputs can be chosen from the pool. Thus the probability per possible instantiated rule firing is independent of n_a, and likewise for other term numbers n_b.

Likewise for a rule with k identical inputs, the annihilation operator monomial $(\hat{a}_a)^k$ gives a factor of $n_a!/(n_a-k)! \equiv (n_a)_k$ to the total firing rate, which is proportional to the number of ways of choosing k unordered inputs from the pool $\binom{n_a}{k}$. The proportionality factor of $k!$, like $\rho_r([x_i],[y_j])$, is an intrinsic property of the rule r and independent of the pool size n_a; thus we define ρ_r in Equation 18 so that it already contains this factor. This definition is a matter of convention, but our choice of convention also has the advantage of reproducing the chemical Law of Mass Action for large n_a, with ρ_r and not $\rho_r/k!$ as the reaction rate for reaction r, thus agreeing with chemical usage (see Section 4.2) in this important limit.

A *monotonic* rule has all of its LHS terms appear also on the RHS, so that nothing is destroyed, in which case

$$\hat{O}_r = \rho_r([x_i],[y_j]) \left[\prod_{i \in \mathrm{rhs}(r) \setminus \mathrm{lhs}(r)} \hat{a}_{a(i)}(x_i) \right] \left[\prod_{j \in \mathrm{lhs}(r)} N_{b(j)}(y_j) \right] . \tag{21}$$

Unfortunately the foregoing expressions for $\hat{O}_r$ don't conserve probability because probability inflow to new states (described by $\hat{O}_r$) must be balanced by outflow from current state (diagonal matrix elements). The following operator does conserve probability:

$$O_r = \hat{O}_r - \mathrm{diag}(\mathbf{1}^T \cdot \hat{O}_r) \equiv \hat{O}_r - D_r$$

For Equation 18, assuming discrete parameters y, we may simply calculate D_r in terms of $(N)_n \equiv N(N-I)(N-2I)...(N-(n-1)I) = N!/(N-nI)!$:

$$D_r = \rho_r([x_i],[y_j]) \quad \left[\prod_{b \in \mathcal{A}, y \in V_b} (N_b(y))_{|\{j | j \in \mathrm{lhs}(r) \wedge (b(j)=b) \wedge (y_j=y)\}|} \right] \tag{22}$$

Note that lower bounds on nonzero D_r elements are determined by ρ.

For the entire grammar the time evolution operator is simply a sum of the generators for each rule:

$$H = \sum_r O_r = \sum_r \hat{O}_r - \sum_r D_r \equiv \hat{H} - D \ . \tag{23}$$

This superposition implements the basic principle that every possible rule firing is an exponential process, all happening in parallel until a firing occurs. Note that (Equation 18, Equation 19) and $\hat{H} = \sum_r \hat{O}_r$ are encompassed by the polynomial ring $\mathbb{R}[\{B_\alpha\}]$ where the basis operators include all creation and annihilation operators. Ring addition (as in Equation 23 or Equation 20) corresponds

to independently firing processes, such as arise from different rules in the same grammar. Ring operator multiplication (as in Equation 18) corresponds to obligatory event co-occurrence.

Equation 23 completes the definition of $\Psi_{(c)}$ from Section 3.1.

Thus, SPG's have an "operational semantics" (the solution of the master equation for time evolution operators) that is also "compositional", since the syntactic union or concatenation of rules (or multisets of rules) in a SPG corresponds to composition of semantics by operator addition in Equation 23. Operator multiplication is used to construct the time evolution operator for each rule. The compositionality of the semantics may be related to its asynchronicity: the operator semantics doesn't in general impose a unique preferred execution order on rule-firing events.

3.4.1 Relation to fixed points

The Ergodic Theorem gives conditions under which a stochastic process will converge to a limiting distribution. It is tempting in that case to take the semantics to be the limiting distribution rather than the much larger object that is the family of approaches to equilibrium depending on the initial distribution. However, it would be less general than to keep the full semantics and apply an application-dependent projection operation afterwards.

3.4.2 Relation to quantum mechanics

If $t \to it = \sqrt{-1}t$ and

$$\hat{a} = \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots \\ 1 & 0 & 0 & 0 & \\ 0 & \sqrt{2} & 0 & 0 & \\ 0 & 0 & \sqrt{3} & 0 & \\ \vdots & & & \ddots & \ddots \end{pmatrix} = \sqrt{n}\delta_{n,m+1} \quad \text{and} \quad a = \begin{pmatrix} 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & \sqrt{2} & 0 & \\ 0 & 0 & 0 & \sqrt{3} & \\ 0 & 0 & 0 & 0 & \ddots \\ \vdots & & & & \ddots \end{pmatrix} = \sqrt{m}\delta_{n+1,m},$$

then we recover the particle number basis for time-evolution operators in quantum mechanical dynamics. The same commutation algebra $[a, \hat{a}] = I$ holds, for $n_{\max} = \infty$. In this way SPG syntax can be given another continuous-time semantics as a quantum mechanical system. However, the construction of Fock spaces is slightly different, requiring Hilbert spaces [26], and the standard interpretation of the state vector in terms of probability is different and loses quantum phase information, so the mapping to discrete-time algorithms (undertaken below for the standard stochastic process semantics) becomes more problematic. We may call an SPG with continuous-time quantum semantics a "quantum grammar" and specify quantum semantics with a semantics directive (Section 2.3) by using header syntax of the form "**grammar** (quantum) ..." . Currently no implementation of this SPG semantics exists.

3.5 Semantics of recursion among grammars

This section is not essential for all SPG applications.

Suppose SPG Γ calls SPG $\tilde{\Gamma}$ using a **via** clause. In Section 2.4.1 we have defined the type translation matrices M and $\tilde{M}$. If the limiting distribution $\Psi_c^*(\tilde{\Gamma})$ exists for all initial states $\Pr(0)$, it defines a new operator $B^*(\tilde{\Gamma}) = \lim_{t\to\infty} \exp tH(\tilde{\Gamma})$. It is possible to project this operator onto a subspace for which $n_a(x) = 0$ for all but a few term types τ_a, using subspace projection operators $P(\{\tau_{a(i)}|i \in \mathcal{I}_L\})$ and $\tilde{P}(\{\tau_{a'(j)}|j \in \tilde{\mathcal{I}}_R\})$ on types $\{\tau_{a(i)}\}$ that are induced by the (much smaller) 1:1 or many:1 type substitution tables M_{ba} and $\tilde{M}_{ab}$ specified using the syntactic forms of Section 2.4.1:

$$\begin{aligned} O_r &= \left(\tilde{P}(\left\{\tau_{a'(j)}|j \in \mathcal{I}_R\right\})\right) B^*(\tilde{\Gamma}) \left(P(\left\{\tau_{a(i)}|i \in \mathcal{I}_L\right\})\right) \\ B^*(\Gamma) &= \lim_{t\to\infty} \exp tH(\Gamma) \end{aligned} \tag{24}$$

where P is obtained from M by

$$P(\left\{\tau_{a(i)}|i \in \mathcal{I}_L\right\}) = \sum_{i\in\mathcal{I}_L} \int_{V_{a(i)}} \sum_b M_{ba} S\left((b, \varphi_i(x_i)), (a(i), x_i)\right) dx_i$$

using the shift operator

$$S(B, A)_{n_B n_A} = \delta_{n_B n_A} = \begin{pmatrix} 0 & 0 \\ I & 0 \end{pmatrix}$$

where $B = (b, y \in V_b)$ and $A = (a, x \in V_a)$. Likewise for $\tilde{M}$ and $\tilde{P}$ using the inverse shift operator S^T.

This time evolution operator O_r can be used to define the semantics of a rule of the form

$$\left\{\tau_{a(i)}(x_i)|i \in \mathcal{I}_L\right\} \to \left\{\tau_{a'(j)}(y_j)|j \in \mathcal{I}_R\right\} \quad \textbf{via} \ \tilde{\Gamma} \tag{25}$$

within the calling grammar Γ or even if Γ=$\tilde{\Gamma}$, for direct recursion. This is how one continuous-time grammar can "call" another one, even recursively, in the manner of a subroutine. Argument substitution in the calling mechanism is defined by the projection operator P above. A single rule could have both **with** and **via** clauses, in which case the two firing rates are multiplied. For nonconverging SPG's, one can project to the probability distribution on states after a definite elapsed time t using the operator $B(\tilde{\Gamma}|t) = \exp tH(\tilde{\Gamma})$ in place of $B^*(\tilde{\Gamma})$ in Equation 24. In this case the syntax of Equation 25 is "...**via** $\tilde{\Gamma}(t)$".

Another form of reference between grammars is more analogous to the "macros" than subroutines in a programming language. In this case the syntax uses the clause "**substituting** Γ", possibly sped up or slowed down by an optional "**with** $\rho(x)$" clause. The semantics of "**substituting** $\tilde{\Gamma}$" is

$$O_r = \rho(x) \left(\tilde{P}(\left\{\tau_{a'(j)}|j \in \tilde{\mathcal{I}}_R\right\})\right) H(\tilde{\Gamma}) \left(P(\left\{\tau_{a(i)}|i \in \mathcal{I}_L\right\})\right) \tag{26}$$

where as usual ρ defaults to 1. All interactions between super-grammar and sub-grammar, whether mediated by "**via**" or "**substituting**" clauses, are restricted by the subgrammar's rule header.

3.6 Discrete-time SPG semantics

The operator $\hat{H}$ describes the flow of probability per unit of continuous time, over an infinitesimal continuous-time interval, into new states that result from a single rule-firing of any type. We seek a related semantics for the same SPG in discrete time.

If we start in the state $p_0 = |\{m_a(x_a)\}\rangle$ (a pure state of the pool of parameterized terms) and condition the probability distribution at later times $t > 0$ on a single rule having fired, thereby setting aside the probability weight for all other possibilities, the resulting distribution p_1 on pool states should be proportional to $\hat{H} \cdot p_0$ with a proportionality constant that ensures normalization. So if the n'th component of p_1 is:

$$[p_1]_n \equiv \Pr_{\text{discrete}}\left(|\{n_a(x_a)\}\rangle \middle| k = 1 | \{m_a(x_a)\}\rangle) \right)$$

then we could define the discrete-time dynamics using

$$p_1 = \left(\hat{H} \cdot p_0\right) / \left(\mathbf{1} \cdot \hat{H} \cdot p_0\right) \qquad \text{if} \qquad \mathbf{1} \cdot \hat{H} \cdot p_0 \neq 0. \tag{27}$$

Since p_0 is a pure state, the l'th component of p_0 is $[p_0]_l = \delta(l, m)$ and the n'th component of this expression is equal to

$$\frac{\sum_l \hat{H}_{n,l} * [p_0]_l}{\sum_{n'} \sum_l \hat{H}_{n',l} * [p_0]_l} = \frac{\hat{H}_{n,m}}{\sum_{n'} \hat{H}_{n',m}}$$

$$= \sum_l \left(\hat{H}_{n,l} / \left(\sum_{n'} \hat{H}_{n',l} \right) \right) ([p_0]_l) = \left[\hat{H} \cdot \text{diag}(\mathbf{1} \cdot \hat{H})^{-1} \cdot p_0 \right]_n .$$

The problem with this definition is the possibility of dividing by zero when the pure (pool) state is also a terminal state (for which $\hat{H} \cdot p_0 = 0$), so that no probability flows out of it.

3.6.1 Principal discrete-time semantics

To avoid division by zero in the definition of $\Psi_d(\Gamma)$, we first define

$$\tilde{H}(\omega) = \hat{H} + \omega \, \text{diag}(\Theta(\mathbf{1} \cdot \hat{H} = 0)) \quad \text{and} \quad \tilde{D}(\omega) = \text{diag}(\mathbf{1} \cdot \tilde{H}(\omega)) \tag{28}$$

for some fixed, real-valued $\omega \geqslant 0$ (here Θ is applied elementwise), and

$$\left[D'(D)\right]_{nm} = \begin{cases} 1/D_{nn} & \text{if} \quad n = m \quad \text{and} \quad D_{nn} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{29}$$

By this definition of the "prime" operation, for diagonal matrices $D'' = D$. Note that the continuous-time dynamics is unaffected by the value of ω:

$$H = \hat{H} - D = \tilde{H} - \tilde{D} \tag{30}$$

Now we can define the first rule-firing update

$$p_1 = \tilde{H} \cdot \tilde{D}' \cdot p_0, \quad \text{where} \quad \tilde{D}(\tilde{H}) = \text{diag}(\mathbf{1} \cdot \tilde{H}) \tag{31}$$

which is the same as Equation 27 for nonterminal pure states, but for terminal states either results in $p_1 = \mathbf{0}$ (if ω=0) or $p_1 = p_0$(for $\omega > 0$); A terminal state either has its probability vector vanish

or stay fixed upon further update, depending on the value of the fixed parameter ω. This update is now in a form that can be iterated as a linear map, even if it is applied to a potentially mixed state such as p_1, and hence can be iterated and interpreted as a stochastic algorithm:

$$p_k = \tilde{H} \cdot \tilde{D}' \cdot p_{k-1} = \left(\tilde{H} \cdot \tilde{D}'\right)^k \cdot p_0 \quad \text{for} \quad k \in \mathbb{N}. \tag{32}$$

This expression directly specifies a discrete-time execution algorithm. It represents a Markov chain if $\omega > 0$. To see that, sort pool states into nonterminal states followed by terminal states and write $\tilde{H}$ and $\tilde{D}$ in (nonterminal, terminal) block form:

$$\tilde{H} = \begin{pmatrix} \tilde{H}_{11} & 0 \\ \tilde{H}_{21} & \omega I \end{pmatrix}; \tilde{D} = \begin{pmatrix} \mathrm{diag}(\mathbf{1} \cdot \tilde{H}_1) & 0 \\ 0 & \omega I \end{pmatrix}; \tag{33}$$

and

$$\tilde{D}' = \begin{pmatrix} \mathrm{diag}\left(\mathbf{1} \cdot \tilde{H}_1\right)^{-1} & 0 \\ 0 & I/\omega \end{pmatrix}; \tilde{H} \cdot \tilde{D}' = \begin{pmatrix} \tilde{H}_{11} \cdot \mathrm{diag}(\mathbf{1} \cdot \tilde{H}_1)^{-1} & 0 \\ \tilde{H}_{21} \cdot \mathrm{diag}(\mathbf{1} \cdot \tilde{H}_1)^{-1} & I \end{pmatrix} \tag{34}$$

Then $\mathbf{1} \cdot \tilde{H} \cdot \tilde{D}' = \mathbf{1}$ and Equation 32 represents a Markov chain for $\omega > 0$. In the $\omega > 0$ case, further updates beyond the terminal state simply leave the probability unchanged ($[\tilde{H} \cdot \tilde{D}']_{22} = I$); it is as if there were an extra rule that fires to no effect. These updates may be called *pseudo-events*. Equation 30 and Equation 33-34 show that the actual value of $\omega > 0$ is irrelevant to continuous-time and discrete-time dynamics respectively, so we are free to pick some arbitrary nonnegative value scaled by some property of $\hat{H}$ (since ω has units of frequency or inverse time). But we are also free to retain the adjustability of ω, which is the strategy adopted here.

If ω=0 the update formula of Equation 32 is still interpretable as a stochastic algorithm, which halts upon reaching a terminal state, even though it is not a Markov chain:

$$\tilde{H} = \hat{H} = \begin{pmatrix} \hat{H}_{11} & 0 \\ \hat{H}_{21} & 0 \end{pmatrix}; \tilde{D} = \begin{pmatrix} \mathrm{diag}(\mathbf{1} \cdot \hat{H}_1) & 0 \\ 0 & 0 \end{pmatrix}; \tag{35}$$

and

$$\tilde{D}' = \begin{pmatrix} \mathrm{diag}(\mathbf{1} \cdot \hat{H}_1)^{-1} & 0 \\ 0 & 0 \end{pmatrix}; \tilde{H} \cdot \tilde{D}' = \begin{pmatrix} \hat{H}_{11} \cdot \mathrm{diag}(\mathbf{1} \cdot \hat{H}_1)^{-1} & 0 \\ \hat{H}_{21} \cdot \mathrm{diag}(\mathbf{1} \cdot \hat{H}_1)^{-1} & 0 \end{pmatrix} = \hat{H}D \tag{36}$$

The distribution p_k as a function of step number (k) on possible execution traces is defined as the discrete-time semantics $\Psi_d(\Gamma)$, even if total "probability" decreases with iterations due to terminal states. A disadvantage of the semantics in the ω=0 case is that in subsequent iterations the probability state vector p_k carries no information about *which* terminal state the system ended up in.

Termination probabilities $p_k(\triangle)$ (a new scalar, not a vector like p_k, at each step k) may be included in the ω=0 case so that total probability is conserved, by the following equivalent formulation which is an affine map but not a Markov chain:

$$\begin{pmatrix} p_{k+1} \\ p_{k+1}(\triangle) \\ 1 \end{pmatrix} = \begin{pmatrix} \hat{H} \cdot D' & 0 & 0 \\ -\mathbf{1} \cdot \hat{H} \cdot D' & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_k \\ p_k(\triangle) \\ 1 \end{pmatrix}$$
$$\left\{ \begin{array}{c} p_k = \hat{H} \cdot D' \cdot p_{k-1} = \left(\hat{H} \cdot D'\right)^k \cdot p_0 \\ p_k(\triangle) = 1 - \hat{H} \cdot D' \cdot p_{k-1} = 1 - \left(\hat{H} \cdot D'\right)^k \cdot p_0 \end{array} \right\} \tag{37}$$

We may consider the formal symbol "$\triangle$" to be a new *halted* pool state (the discrete-time successor to all terminal states including itself) which is treated specially by the discrete-time semantics and not needed by the continuous-time semantics. But this interpretation is not essential to the discrete-time semantics of Equation 32.

Of course if there are no terminal pool states, then these variations are equivalent: $\Psi_d(\Gamma, \omega > 0) = \Psi_d(\Gamma, \omega = 0)$. These variations may be called the *principal discrete-time Markovian semantics* (Equation 33-34) and the *principal discrete-time halting semantics* (Equations 35-36 or Equation 37) respectively, and the *principal discrete-time semantics* (Equation 32) collectively. They both provide linear or affine maps U that may be substituted into Equation 14, to complete the definition of Ψ_d.

3.6.2 Alternative discrete-time semantics

We may also define the *alternative discrete-time semantics* ${\Psi'}_d(\Gamma)$ for k rule firings as follows. For a second rule firing, we assume there are no terminal states and we iterate the form of Equation 27 (now applied to both pure and mixed states) rather than Equation 31:

$$p_2 = \frac{\hat{H} \cdot p_1}{\left(\mathbf{1} \cdot \hat{H} \cdot p_1\right)} = \frac{\hat{H} \cdot \left[\left(\hat{H} \cdot p_0\right) / \left(\mathbf{1} \cdot \hat{H} \cdot p_0\right)\right]}{\left(\mathbf{1} \cdot \hat{H} \cdot \left[\left(\hat{H} \cdot p_0\right) / \left(\mathbf{1} \cdot \hat{H} \cdot p_0\right)\right]\right)} =$$
$$= \frac{\left(\hat{H} \cdot \left(\hat{H} \cdot p_0\right)\right) / \left(\mathbf{1} \cdot \hat{H} \cdot p_0\right)}{\left(\mathbf{1} \cdot \hat{H} \cdot \left(\hat{H} \cdot p_0\right)\right) / \left(\mathbf{1} \cdot \hat{H} \cdot p_0\right)} = \left(\hat{H}^2 \cdot p_0\right) / \left(\mathbf{1} \cdot \hat{H}^2 \cdot p_0\right)$$

Iterating, the state of the discrete-time grammar after k rule firing steps is given by the normalized version of $\hat{H}^k \cdot p_0$:

$$p_k = \left(\hat{H}^k \cdot p_0\right) / \left(\mathbf{1} \cdot \hat{H}^k \cdot p_0\right) \tag{38}$$

where $\hat{H} = \sum_r \hat{O}_r$ as before. This expression depends on a normalization constant $c_k = 1/(\mathbf{1} \cdot \hat{H}^k \cdot p_0)$. The normalizing division is analogous to the normalizing subtraction in the exponent of the continuous-time semantics. For unbounded operators of infinite dimension this normalization can be state-dependent and hence dependent on n, so there is no constant k such that for all k $c_k = \alpha^k$. (An example with $c_k \neq \alpha^k$ will be given in Section 4.1.1). This is an important distinction between the alternative discrete semantics and the $\omega > 0$ Markov Chain models, for which there exist such a constant $\alpha = c_1$.

The two semantic maps $\Psi_d(\Gamma)$ (Equation 32) and $\Psi'_d(\Gamma)$ (Equation 38) agree in the special case in which $k = 1$ and p_0 is a pure nonterminal state. Unlike Equation 32, Equation 38 is nonlinear (and non-affine) in p_0 and hence more difficult to implement as an iterative sampling algorithm.

3.7 Simulation/execution algorithms

To convert the operator for a stochastic process into an algorithm for drawing samples from that process, one must transform the starting probability (which in simulations is usually a delta function corresponding to a pure state) by the exponential operator $\exp(tH)$, and produce samples from the resulting distribution. There are several standard approaches to computing and/or sampling this operator exponential. One that appears to be underexploited in the design of algorithms is the time-ordered product expansion of Section 3.7.2.

The goal of obtaining a stochastic algorithm can be served by deriving Markov Chains or more generally (as in the previous section) one or more affine maps on probability vectors that, when iterated by matrix multiplication in specified combinations, converge to the target distribution. This in turn can be done by equating or approximating the exponential $\exp(tH)$ with a polynomial in various operators with nonnegative coefficients that can be interpreted or transformed into probabilities. Here we discuss several ways to expand out the exponential function on operators into useful operator polynomials from which stochastic algorithms may in turn be derived. Further details are provided in the Appendices.

3.7.1 Euler, Trotter, and BCH formulae

The operator exponential $\exp(tH)$ in Equation 13 has a the Taylor series expansion, which in turn implies Euler's formula:

$$\exp(tH) = \lim_{n\to\infty}\left[I + \frac{t}{k}H\right]^k = \lim_{n\to\infty}\left[I + \frac{t}{k}\sum_r O_r\right]^k$$

This formula may be used to sample from the distribution $p(t)$ as follows: sample from the initial distribution $p(0)$=p_0 (which is often a delta function), and at each infinitesimal time step t/n, execute one of the rules according to their infinitesimal probabilities (after scaling by t/n) or, if none are selected, do nothing. So for large finite k this operator polynomial, derived from Euler's formula for the exponential, can be interpreted as an inefficient algorithm corresponding to the Forward Euler method of solving ordinary differential equations. Both have first-order error as a function of $1/k$.

More efficient, higher-order methods are possible using the Trotter product formula as follows:

$$\exp[t(H_0 + H_1)] = \lim_{n\to\infty}\left[I + \frac{t}{k}(H_0 + H_1)\right]^k$$

$$= \lim_{n\to\infty}\left[\left(I + \frac{t}{k}H_0\right)\left(I + \frac{t}{k}H_1\right)\right]^k = \lim_{k\to\infty}\left[e^{(t/k)H_0}e^{(t/k)H_1}\right]^k.$$

Now, the sampling algorithm allows alternation of two different processes H_0 and H_1. It is an analog of "operator splitting" [28], [29] in numerical integration, in which versions of the Baker-Campbell-

Hausdorff (BCH) formula [30]

$$\exp(tH_0)\exp(tH_1) = \exp\left(tH_0 + tH_1 + \frac{t^2}{2}[H_0, H_1] + \frac{t^3}{12}[H_0,[H_0,H_1]] - \frac{t^3}{12}[H_1,[H_0,H_1]] + O(t^4)\right) \tag{39}$$

can be used to derive higher-order simulation algorithms such as that used for example in [31]. Of course, if $[H_0, H_1] = 0$ then the BCH formula becomes trivial because in that special case exponentiation distributes over addition: $\exp(tH_0 + tH_1) = \exp(tH_0)\exp(tH_1)$.

More advanced methods such as the Gillespie stochastic simulation algorithm (suitably generalized to handle parameterized types using the factorization $\rho_r([x_i],[y_j])=\rho_r^{\text{pure}}([x_i])\Pr_r([y_j]|[x_i])$) can be derived from the Time-Ordered Product expansion.

3.7.2 Time-Ordered Product Expansion (TOPE)

A valuable tool for studying such stochastic processes in physics is the Time-Ordered Product Expansion [32, 33]. We use the following form (derived in Appendix 1):

$$\begin{aligned}&\exp(tH)\cdot p_0 = \exp(t(H_0+H_1))\cdot p_0 \\ &= \sum_{k=0}^{\infty}\left[\int_0^t dt_1 \int_{t_1}^t dt_2 \cdots \int_{t_{k-1}}^t dt_k \exp((t-t_k)H_0)H_1\exp((t_k-t_{k-1})H_0)\cdots H_1\exp(t_1H_0)\right]\cdot p_0\end{aligned} \tag{40}$$

where H_0 is a solvable or easily computable part of H, so the exponentials $\exp(tH_0)$ can be computed or sampled more easily than $\exp(tH)$. See for an elementary probabilistic derivation of this form. This expression can be used to generate Feynman diagram expansions, in which k denotes the number of interaction vertices in a graph representing a multi-object history [27]. If we apply (Equation 40) with

$$H_1 = \hat{H} \quad \text{and} \quad H_0 = -D$$

we derive the well-known Gillespie algorithm for simulating chemical reaction networks [34], which can now be applied to SPG's. This derviation of a widely-used stochastic algorithm is explained in more detail at the end of Appendix I. With SPG's we need to consider also the possibility of terminal states, in which case one may alternatively use

$$H_1 = \tilde{H}(\omega) \quad \text{and} \quad H_0 = -\tilde{D}(\omega).$$

However many other decompositions of H are possible, one of which is used in Section 5.3 below. Because the operators H can be decomposed in many ways, there are many valid simulation algorithms for each stochastic process. The particular formulation of the Time-Ordered Product Expansion used in (Equation 40) has the advantage of being recursively self-applicable.

Thus, (Equation 40) entails a systematic approach to the creation of novel simulation algorithms.

3.8 Relation between continuous- and discrete-time semantic maps

In the SPG semantics approach we start with continuous-time stochastic dynamics and specialize to discrete-time (this section) and/or deterministic dynamics. The following three propositions relate the resulting continuous-time and discrete-time semantics.

Proposition 1. Given the stochastic parameterized grammar (SPG) rule syntax of Equation 5

(a) There is a semantic function Ψ_c mapping from any continuous-time stochastic parameterized grammar Γ via a time evolution operator $H(\hat{H}(\Gamma))$ to a joint probability density function on the parameter values and birth/death times of grammar terms, conditioned on the total elapsed time, t. For any initial probability distribution p_0, there is a maximal $T_{\rm def}(p_0) \in [0, +\infty]$ such that for all times on $t \in [0, T_{\rm def})$, $\Psi_c(\Gamma)(t) \cdot p_0$ is a probability density and is the unique solution of the master equation on that interval.

(b) There is a semantic function Ψ_d mapping any discrete-time stochastic parameterized grammar Γ via a time evolution operator $\tilde{H}(\Gamma, \omega)$ to a joint probability density function on the parameter values and birth/death times of grammar terms, conditioned on the total discrete time defined as number of rule firings, k. It depends on whether a global real-valued parameter $\omega \geqslant 0$ is $=0$ or >0; for $\omega > 0$ it is a Markov chain.

Proof of Proposition 1:

(a): Section 3.1 and Section 3.4. The definition of $T_{\rm def}(p_0)$ in Section 3.1 as the least upper bound of times $T' \geqslant 0$ for which there is a unique solution of the master equation for all times $t \in [0, T']$ starting from initial condition p_0, implies that there also exists such a solution on $[0, T_{\rm def}) = \cup_{0<T'<T_{\rm def}}[0, T']$. Suppose this $T_{\rm def}$ were not maximal as claimed in (a). Then there would exist some $T^* > T_{\rm def}$ for which the master equation has a unique solution on $[0, T^*)$, an interval which properly contains $[0, T_{\rm def})$. Then there would exist a unique solution on the subinterval $[0, T' = (T^* + T_{\rm def})/2]$. $T_{\rm def}$ being an upper bound of such T'values, we have $(T^* + T_{\rm def})/2 \leqslant T_{\rm def}$ which implies $T^* \leqslant T_{\rm def}$, in contradiction to $T^* > T_{\rm def}$.

(b): Section 3.1 and Section 3.6.

We have a defined probability $\Pr_{\rm continuous}(\{n_a(x)\}|t) \equiv \Pr_{\rm continuous}(\cdot|t)$ for $t \in [0, T_{\rm def})$; formally it is $\exp(tH) \cdot p_0$ which can be calculated by the TOPE of Section 3.7.2. We would like to compute the probability $\Pr_{\rm continuous}(\cdot|k)$ given k rule-firings, and compare it with $\Pr_{\rm discrete}(\cdot|k)$. Both of these k-step probabilities need further defininition which we now provide. To remove the dependence on t for large times in $\Pr_{\rm continuous}(\cdot|t)$, we need a prior distribution $\Pr_{\rm continuous}(t)$ which is uninformative except at large times for which $\exp(-D\Delta t) \cong 0$ and any bias introduced by the prior on t is unimportant.

Definition 1. Define the *continuous-time k-event probability* as the density $\Psi_c(\Gamma)(t)$ multiplied by the uniform distribution of times t from 0 to some $T < T_{\rm def}(p_0)$, integrated over t from 0 to T and then conditioned on k firings the last of which occured at exactly time t; it is denoted

$\Pr_{\text{continuous}}(\cdot|k, \tau_k = t - t_k = 0, T, p_0)$ where t_k is the time of the k'th rule firing. In equations:

$$\Pr(t) = \begin{cases} 1/T & \text{if } \ t \in [0,T] \\ 0 & \text{otherwise} \end{cases} = \Theta(0 \leqslant t \leqslant T)/T$$

$$\begin{aligned} \Pr_{\text{continuous}}(\cdot, \tau_0, ..., \tau_k, k|T, p_0) \quad & \equiv \int_0^\infty dt \Pr_{\text{continuous}}(\cdot, \tau_0, ..., \tau_k, k|t, p_0) \Pr(t) \\ & = \int_0^T dt \Pr_{\text{continuous}}(\cdot, \tau_0, ..., \tau_k, k|t, p_0) \Pr(t) \end{aligned}$$

(integration over time t, where $\tau_j = t_{j+1} - t_j$=j'th time difference between rule firings). Also

$$\Pr_{\text{continuous}}(\cdot, \tau_k = 0, k|T, p_0) = \int_0^\infty d\tau_0 \cdots \int_0^\infty d\tau_{k-1} \Pr_{\text{continuous}}(\cdot, \tau_0, ..., \tau_{k-1}, \tau_k = 0, k|T, p_0)$$

(last firing happens exactly at time $t_k = t$, and others are unconstrainted) and

$$\Pr_{\text{continuous}}(\cdot|k, \tau_k = 0, T, p_0) = \frac{\Pr_{\text{continuous}}(\cdot, \tau_k = 0, k|T, p_0)}{\Pr_{\text{continuous}}(\tau_k = 0, k|T, p_0)}$$

(conditioned on k=number of rule-firings).

Definition 2. Define the *discrete-time k-event probability* as discrete-time density $\Psi_d(\Gamma)$, conditioned on k and on the absence of a halt by step k (i.e. not reaching a terminal state in the case $\omega = 0$ by step k-1), denoted $\Pr_{\text{discrete}}(\cdot|\text{not halted}, k \text{ events}, p_0)$.

$$\Pr_{\text{discrete}}(\cdot|\text{not halted}, k \text{ events}, p_0) = \frac{\Pr_{\text{discrete}}(\cdot, \text{not halted}|k \text{ events}, p_0)}{\Pr_{\text{discrete}}(\text{not halted}|k \text{ events})}$$

Definition 3. Define the continuous-time *k-liveness* probability as the same probability time integral as in Definition 1, but joint rather than conditional in k, summed over all pool states. It is denoted $\Pr_{\text{continuous}}(\tau_k = 0, k|T, p_0)$:

$$\Pr_{\text{continuous}}(\tau_k = 0, k|T, p_0) = \mathbf{1} \cdot \Pr_{\text{continuous}}(\cdot, \tau_k = 0, k|T, p_0).$$

Definition 4. Define the discrete-time *k-liveness probability* as the discrete-time probability $\Psi_d(\Gamma)$ of not halting after k events, denoted $\Pr_{\text{discrete}}(\text{not halted}|k \text{ events})$:

$$\Pr_{\text{discrete}}(\text{not halted}|k \text{ events}) = \mathbf{1} \cdot \Pr_{\text{discrete}}(\cdot, \text{not halted}|k \text{ events}, p_0).$$

Proposition 2. Suppose that SPG Γ involves only discrete-valued parameters, and that the nonzero elements of $\text{diag}(\tilde{H})$ are bounded below by $\delta > 0$ (so that either $\omega = \delta$ or $\omega \geqslant 0$). If $0 < T < T_{\text{def}}(p_0)$ and $0 < \epsilon \ll 1$ and

$$T\delta \gg k|\log(\epsilon/k)| \qquad \left(i.e. \ \ \epsilon \gg ke^{-T\delta/k}\right),$$

then

(a) Either both the continuous-time k-liveness probability $\Pr_{\text{continuous}}(\tau_k = 0, k|T, p_0)$ is zero and the discrete-time k-liveness probability $\Pr_{\text{discrete}}(\text{not halted}|k \text{ events})$ is zero, or they are both

nonzero and $\Pr_{\text{continuous}}(\cdot|k, \tau_k = 0, T, p_0)$ approximates $\Pr_{\text{discrete}}(\cdot|\text{not halted}, k \text{ events}, p_0)$ with relative error $O(\epsilon)$:

$$\Pr_{\text{continuous}}(\cdot|k, \tau_k = 0, T, p_0) \cong \Pr_{\text{discrete}}(\cdot|\text{not halted}, k \text{ events}, p_0).$$

In particular if $T_{\text{def}}(p_0)=+\infty$ and the liveness probabilities are nonzero, then the $T \to \infty$ limit of the continuous-time k-event probability vector is equal to the discrete-time k-event probability vector:

$$\lim_{T\to\infty}\Pr_{\text{continuous}}(\cdot|k, \tau_k = 0, T, p_0) = \Pr_{\text{discrete}}(\cdot|\text{not halted}, k \text{ events}, p_0);$$

Also

(b) If Γ has no terminal pool states and involves only discrete-valued parameters, and if $\omega=0$, then there exists another grammar $\Gamma'(\Gamma)$ derived from Γ such that without using any prior distribution on t, and under the same assumptions on T, δ and k as in (a) above, with a relative approximation error of $O(\epsilon)$

$$\Pr_{\text{continuous}\Gamma'}(\cdot, k|t = T, p_0) = p^*_k(t = T) \cong \Pr_{\text{discrete}\Gamma}(\cdot|\text{not halted}, k \text{ events}, p_0).$$

Proof of Proposition 2:

(a) Equation 40 (the TOPE) is the starting point; it is established in Appendix 1. Then from Appendix 2, under the stated conditions,

Case I: $\Pr_{\text{continuous}}(\tau_k = 0, k|T, p_0) = 0$: Then $\tau_k = t - t_k$ and

$$\Pr_{\text{continuous}}(\tau_k = 0, k|T, p_0) = 0 = \Pr_{\text{discrete}}(\text{not halted}|k \text{ events})$$

Case II: $\Pr_{\text{continuous}}(\tau_k = 0, k|T, p_0) > 0$:

If $\omega \neq 0$, with a relative approximation error of $O(\epsilon)$,

$$\Pr_{\text{continuous}}(\cdot|k, \tau_k = 0, T, p_0) \cong \Pr_{\text{discrete}}(\cdot|k, p_0)$$

If $\omega = 0$ with a relative approximation error of $O(\epsilon)$,

$$\Pr_{\text{continuous}}(\cdot|k, \tau_k = 0, T, p_0) \cong \Pr_{\text{discrete}}(\cdot|\text{not halted}, k \text{ events}, p_0).$$

This is also true of the case $\omega \neq 0$ since in that case there is zero probability of halting and $\Pr_{\text{discrete}}(\cdot|k, p_0) = \Pr_{\text{discrete}}(\cdot|\text{not halted}, k \text{ events}, p_0)$.

(b) Appendix 3.

Notes :

1. In the limit $T \to 0$, the inequalities postulated in Proposition 2 cannot all be satisfied and no approximation follows in (a) or (b).
2. Equation 22 of Section 3.4 allows us to evaluate δ for some SPG's.

Proposition 3. For any SPG Γ if there are no terminal states and if $T_{\text{def}}(p_0)>0$, then the short-time limit of the density $\Psi_c(\Gamma)$ conditioned on t as $t \to 0$ and conditioned on k is equal to the alternative discrete-time semantics $\Psi'_d(\Gamma)$ after k steps.

Proof of Proposition 3: Appendix 4.

Corollary. The diagram of Figure 1 commutes:

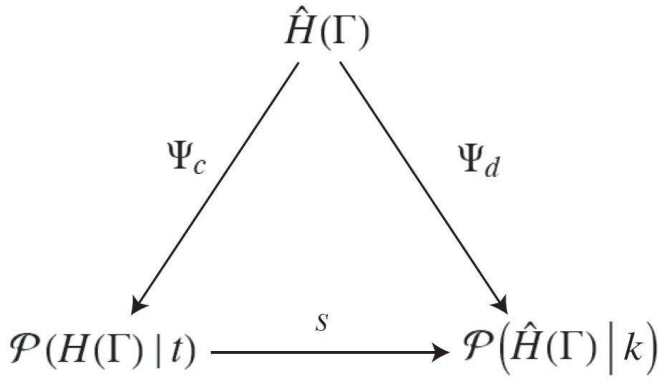


Figure 1: Commutative diagram for continuous-time and discrete-time semantics. Here k = number of rule firings, t= continuous time, S = a map from continuous-time to discrete-time probabilities associated with large integration times T (Proposition 2a, 2b) or small evaluation times t (Proposition 3) in the continous-time semantics.

3.9 Discussion: Morphisms of SPG's

Given a new kind of mathematical object (here, SPG's or DG's) it is generally productive in mathematics to consider the transformations or "morphisms" of such objects (mappings from one object to another or to itself) that preserve key properties. Examples include transformational geometry (groups acting on lines and points), homomorphisms acting on groups, and functors acting on categories. In the case of SPG's, two possibilities for the preserved property are salient. First, an SPG syntactic transformation $\Gamma \to \Gamma'$ could preserve the semantics $\Psi(\Gamma) = \Psi(\Gamma')$ either fully or just in fixed point form: $\Psi^*(\Gamma) = \Psi^*(\Gamma')$. Still more relaxed would be a criterion that preserved some projection P_O of the (possibly fixed point) semantics onto an observed subspace O: $P_O\Psi^{(*)}(\Gamma) = P_O\Psi^{(*)}(\Gamma')$, for example that defined by a grammars's header rule. Preserving the full semantics, without projection, would be required of a simulation algorithm.

As a second possibility, an inference algorithm could reverse input and output (as defined by header rule) but preserve the joint probability distribution $\Pr(\text{in}, \text{out})$ on input and output random variables, by using Bayes' rule,

$$\Pr_\Gamma(\text{out}, \text{internal}|\text{in})\Pr(\text{in}) = \Pr(\text{in}, \text{internal}, \text{out}) = \Pr_{\text{Inference}}(\text{in}, \text{internal}|\text{out})\Pr(\text{out})$$

where $(\text{in}, \text{internal}, \text{out})$ are collections of parameterized terms that are inputs to, internal to, and outputs from the grammar Γ respectively. Thus, the header rule for an inverse or inference grammar is a global arrow-reversal of the header rule of the generative grammar.

One approach to creating inference algorithms is to reverse the arrows locally on the individual grammar rules, changing the probability rate functions as well, so as to implement one step of an iterative algorithm whose fixed point is the inference distribution $\Pr_{\text{Inference}}$. In this way a rule r such as

$$\text{input}(x) \to \text{output}(y) \quad \textbf{with} \quad \rho_{\text{forward}} \quad \Pr_r(y|x)$$

is replaced by a reversed rule r' such as

$$\text{output}(y) \to \text{input}(x) \quad \textbf{with} \quad \rho_{\text{reverse}} \quad \Pr_{r'}(x|y),$$

possibly together with new rules for iteration. An example of such an algorithm will be shown in Section 4.1.3. A similar approach, taken in [35], is known as "arc reversal" in Bayesian Networks (*BN*s). By a series of arc transformation a *BN* is transformed to an equivalent network in which the "evidence" appears in the root nodes and therefore is available for inference tasks. Such arc transformations may be prohibitively expensive for large *BN*'s, especially in Dynamic Bayesian Networks (*DBNs*) which are used for modeling stochastic temporal processes. Nevertheless, it was shown by [36] that even localized arc reversals in each time slice of a *DBN* can boost the performance of an inference algorithm based on Monte Carlo methods. In its application to SPG's, this approach would necessarily generalize from finite-dimensional to arbitrary-infinite-dimensional Monte Carlo Markov Chain-like sampling algorithms.

4 Examples

4.1 Cluster trees

Here is a simple cluster-generating grammar that generalizes *binaryclustergen* by allowing any number of elements per cluster:

grammar (discrete-time) *clustergen* $(\text{nodeset}(x) \to \{\text{node}(x_i)\})$ {

$\text{nodeset}(x) \to \text{node}(x), \{\text{child}(x) | 1 \leqslant i \leqslant n\}$ **with** $q(n)$ **subject to** $n \geqslant 0$.

$\text{child}(y) \to \text{nodeset}(x)$ **with** $\phi(x|y)$

}

Such generative data cluster models have considerable utility for problem formulation in pattern recognition, image analysis, and machine learning. Since there is only one term on each LHS, the grammar is "context free". Its behavior is shown in Figure 2. We take it to define the probabilistic model of a *context free feature tree*, $\mathcal{T}(q, \phi)$.

4.1.1 Context-free grammars

Here is the alternative discrete-time semantics $\Psi'_d(\Gamma)$ of $\mathcal{T}(q, \phi)$ (omitting for simplicity the node labels x, and just keeping the tree structure):

$$\hat{H} = \sum_{k=0}^{\infty} q(k)\hat{a}^k a = g(\hat{a})a \qquad H = g(\hat{a})a - N \tag{41}$$

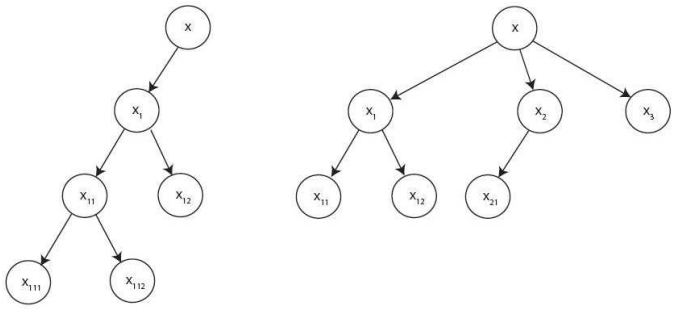


Figure 2: Two feature trees generated by the *clustergen* stochastic parameterized grammar. (a) Pr $= q(1)q(2)^2q(0)^3 \times\phi(x_1|x)\phi(x_{11}|x_1)\phi(x_{12}|x_1) \times\phi(x_{111}|x_{11})\phi(x_{112}|x_{11})$. (b) Pr $= q(3)q(2)q(1)q(0)^4 \times\phi(x_1|x)\phi(x_2|x)\phi(x_3|x) \times\phi(x_{11}|x_1)\phi(x_{12}|x_1)\phi(x_{21}|x_2)$.

$$\begin{aligned}\hat{H}^2 &= g(\hat{a})^2a^2 + g(\hat{a})g'(\hat{a})a \\ \hat{H}^3 &= g(\hat{a})^3a^3 + 3(g(\hat{a}))^2g'(\hat{a})a^2 + g(\hat{a})\left(g'(\hat{a})\right)^2a + (g(\hat{a}))^2g''(\hat{a})a; \ldots\end{aligned} \tag{42}$$

where

$$g(z) = \sum_{n=0}^{\infty} z^n q(n) \quad \text{and} \quad g(1) = \sum_{n=0}^{\infty} q(n) = 1$$

In this model, every power of $\hat{H}$, and the continuous-time evolution $\exp tH$, can be formally expressed and computed using efficient power series operations (composition and reversion) on generating functions. With generating functions $f(x)$, operators $(a, \hat{a})$ are represented by $(\partial_x, x\times)$ respectively. Then $\hat{H} \mapsto [g(x)\partial_x]$, and $H \mapsto [(g(x) - x)\partial_x]$. Defining

$$J(x; x_0) = \int_{x_0}^{x} \frac{du}{g(u) - u} \quad \text{and} \quad K(x; x_0) = \int_{x_0}^{x} \frac{du}{g(u)} \tag{43}$$

Then, considering $J(x; x_0)$ to be a function of just its first argument x,

$$\frac{d}{dJ} = \frac{dx}{dJ}\frac{d}{dx} = \frac{1}{dJ/dx}\frac{d}{dx} \leftrightarrow H$$

and

$$e^{tH}f(x) \mapsto e^{t(d/dJ)}f(J^{-1}(J(x))) = f(J^{-1}(t + J(x))) \tag{44}$$

by Taylor's theorem in the form $e^{\alpha\partial_x}f(x) = f(x+\alpha)$. Thus we need only calculate $J^{-1}(t+J(x))$ using power series reversion and composition. [1] (Section III.3 eq. (7)) provides a different derivation. A similar calculation holds for discrete-time semantics (Equation 42) using K, so that

$$\begin{aligned}e^{s\hat{H}}f(x) \mapsto f(K^{-1}(s+K(x))) &= f(x+sg(x)+\frac{s^2}{2}g(x)g'(x)+\frac{s^3}{3!}\left(g(x)\left(g'(x)\right)^2 + (g(x))^2g''(x)\right)+\ldots) \\ &= f(x) + sg(x)\partial_x f(x) + \ldots \leftrightarrow (I + sg(\hat{a})a + \ldots)\, f(x),\end{aligned} \tag{45}$$

from which we can recalculate (Equation 42). In either case the grammar is tractable because *clustergen* is a context-free grammar: there is only one term on the left hand side of each rule. Thus,

both the continuous-time and discrete-time semantics can be at least formally solved in this special case.

Example. $\hat{H} = \rho(1 + c^2\hat{a}^2)a$ where $\rho = 1/(1 + c^2)$, represents a birth-death process with one birth or death event per discrete time step. Each possible birth or death event is oblivious of the all others in its continuous-time firing rate. Then the alternative discrete-time semantics has $K(x) = \arctan(cx)/(\rho c)$, so $e^{s\hat{H}}|1\rangle \mapsto (\tan(s\rho c) + cx)/[c(1 - cx\tan(s\rho c))]$. Therefore $\mathbf{1} \cdot e^{s\hat{H}}|1\rangle = (\tan(s\rho c) + c)/[c(1 - c \ \ \tan(s\rho c))]$, which has singularities at finite s and therefore is not equal to $e^{\alpha s}$for any α. So in this case, $\mathbf{1} \cdot \hat{H}^n|1\rangle \neq \alpha^n$.

In continuous time for $c = 1$, this model also has a simple solution: $J^{-1}(t + J(x)) = \frac{t+2x-tx}{t+2-tx} = \frac{t}{2+t} + \sum_{n=1}^{\infty} \frac{4t^{n-1}}{(2+t)^{n+1}} x^n$.

Example. If q has a normalized power-law distribution:

$$q(n) = \begin{cases} p_0 & n = 0 \\ (1 - p_0)\, n^{-\alpha}/\zeta(\alpha) & n > 0 \end{cases}$$

then $g(z) = p_0 + (1 - p_0)\mathrm{Li}_\alpha(z)/\zeta(\alpha)$ (where $\mathrm{Li}_\alpha(z)$ is the polylogarithm function) and the continuous-time and alternative discrete-time dynamics are given formally by the integrals

$$J(x; x_0) = \int_{x_0}^{x} \frac{dz}{p_0 - z + (1 - p_0)\,\mathrm{Li}_\alpha(z)/\zeta(\alpha)} \quad \text{and} \quad K(x; x_0) = \int_{x_0}^{x} \frac{dz}{p_0 + (1 - p_0)\,\mathrm{Li}_\alpha(z)/\zeta(\alpha)}$$

But these integrals, unlike those of the previous example, do not appear in integral tables and may not have analytic solutions in terms of commonly studied special functions.

4.1.2 Clustering algorithms

The following grammar is *equivalent* to clustergen, in its conditional distribution $\Pr(\{\mathrm{node}(x_I)|1 \leqslant I \leqslant N\}|N)$, of interest because it is a resource-bounded version of clustergen and its discrete-time semantics is essentially the resource-bounded feature tree model $\mathcal{T}(N, q, \phi)$. It constitutes a valid *grammar transformation* of clustergen:

grammar (discrete-time) *rseqclustergen* $(\mathrm{nodeset}(x, N) \to \{\mathrm{node}(x_i)\})$ {

$\mathrm{nodeset}(x, N) \to \mathrm{node}(x), \mathrm{children}(x, n, N - 1)|1 \leqslant i \leqslant n\}$ **with** $r(n|N)$

$\mathrm{children}(x, n, N) \to \mathrm{child}(x, N'), \mathrm{children}(x, n - 1, N - N')$ **with** $R(N'|n, N)$

$\mathrm{children}(x, 0, N) \to \varnothing$

$\mathrm{child}(y, N) \to \mathrm{nodeset}(x, N)$ **with** $\phi(x|y)$

}

The resource-bounded cluster generator, *rseqclustergen,* is important since using *clustergen* to sample bounded size feature cluster trees for many different distributions (different $q(n)$ functions in the unbounded feature tree model $\mathcal{T}(q, \phi)$) can be very inefficient or even intractable, as was found in simulations. For example, when $q(0)$=0 the probability that a clustergen grammar simulation

will converge to a cluster tree of finite size vanishes. The functions R and r can be computed with reasonable efficiency by reversion of series using generating functions [37], though clearly analytic solutions yield yet more efficient algorithms when available.

A currently popular approach to clustering models is by way of Dirichlet processes. The stick-breaking construction of a Dirichlet process can be expressed with this discrete-time grammar (following [38]):

grammar (discrete-time) *DP* (start(N) $\rightarrow$ {cluster(i, θ_k, π_k)$|1 \leqslant k < \infty$}) {

start(N) $\rightarrow$ cluster$'(0, 0, 0, 1, 0)$

cluster$'(k, \theta_k, \beta_k, \Xi_k, \pi_k) \rightarrow$ cluster(k, θ_k, π_k), cluster$'(k+1, \theta_{k+1}, \beta_{k+1}, \Xi_{k+1}, \pi_{k+1})$

with β_{k+1} Beta$(\cdot|1, \alpha) = (\Gamma(1+\alpha)/\Gamma(\alpha))(\beta_{k+1})^{\alpha-1}$

with $G_0(\theta_k)$

where $\pi_{k+1} = \beta_{k+1}\Xi_k$

where $\Xi_{k+1} = (1-\beta_{k+1})\Xi_k$

}

Then the Chinese Restaurant process for cluster generation is:

grammar (discrete-time) *CRP* (start(N) $\rightarrow$ {sample(x)$|1 \leqslant k \leqslant N$}) {

start(N) $\rightarrow$ samples(N), {cluster(k, θ_k, π_k)$|1 \leqslant k < \infty$} **via** DP

samples(N), C = {cluster(i, θ_k, π_k)$|1 \leqslant k < \infty$} $\rightarrow$ samples($N-1$), C, sample$'(\theta_k)$

with π_k

subject to $N > 0$

sample$'(\hat{\theta}) \rightarrow$ sample(x) **with** p$(\cdot|\hat{\theta})$

}

The *clustergen* grammars can be specialized and limited so as to function in a very similar manner to DP and CRP above, with a Binomial-Beta substituted for the Beta distribution [37]. However, *clustergen* determines a more general family of distributions. For example one can control the histogram of cluster sizes.

Thus,

Proposition 4. There is a serial context-free grammar Γ_{tree} whose asymptotic probability distribution is that of the context-free feature tree $\mathcal{T}(q, \phi)$, and another context-free grammar $\Gamma_{\text{rl}-\text{tree}}$ whose asymptotic probability distribution is that of the resource-limited context-free feature tree $\mathcal{T}(N, q, \phi)$.

4.1.3 "Arrow reversal" inference algorithm for hierarchical clustering

This section presents an inference scheme for an SPG based on Monte Carlo techniques. Here we do not attempt to strictly 'reverse' the production rules but rather transform them to association rules that weigh the matching between precondition elements and the outcome elements. We assume that the number of latent (hidden) objects of each type is known in advance, and therefore the search is only over a subspace of the pool configurations. Expanding the search to unknown number of latent objects is a question for future work.

The inference scheme will be demonstrated by using a data clustering example. The *data-generator* grammar produces a hierarchical tree structure of clusters where the bottom level nodes are the data items. The cluster distributions are Gaussians and generate a limited number of objects according to the cluster size control factor $(S - s)$ in the rate function, which serves to ensure that each cluster eventually gets exactly S children. Note that the cluster size limit, S, can be varied according to the cluster level.

grammar *data-generator* $(\text{start}(\hat{x}) \rightarrow \{\text{cluster}(x, \sigma, l, s), \text{item}(\tilde{x})\})$ {

$\text{start}(\hat{x}) \rightarrow \text{cluster}(\hat{x}, \hat{\sigma}, 0, 0)$

$\text{cluster}(x, \sigma, l, s) \rightarrow \{\text{cluster}(x, \sigma, l, s+1), \text{cluster}[x', \sigma/V, l+1, 0]\}$

with $(S - s) * N(x'; x, \sigma)$

subject to $l < L$

$\text{cluster}(x, \sigma, L, s) \rightarrow \{\text{cluster}(x, \sigma, L, s+1), \text{item}(x')\}$

with $(S - s) * N(x'; x, \sigma)$

}

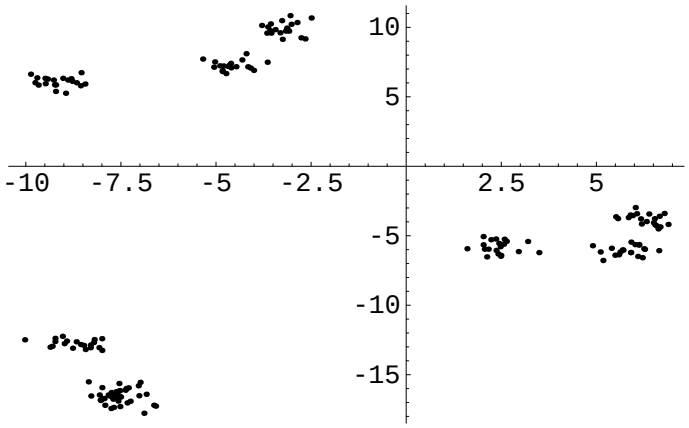


Figure 3: Output of *data-generator* grammar execution for two level cluster tree (L = 2)

The inference algorithm for *data-generator* is given in a form of another stochastic grammar, *cluster-Infer*. The input for this grammar is an initialized cluster tree and the observed data items as

the lowest level clusters. The main part of the inference grammar is the *match-subcluster* subgrammar that uses the same forward rule of *data-generator* in order to match the subclusters to the clusters in the level above and then regenerate them. Once a cluster in a given level has been matched to offspring subclusters and a parent supercluster, then the conditional probability of its location given their locations is readily available and Gaussian. There are two choices, either to jump to the most likely position which is the mean (*compute-new-mean1*) or to make a stochastic move in the conditional distribution (*compute-new-mean2)* resembling a Gibbs sampling [39] algorithm. *data-generator* iterates between matching and parameter re-estimation steps over the cluster tree for every inner level until convergence.

Related algorithms for data clustering such as K-means [40] and the EM algorithm [41] work similarly by iterating between the two stages of cluster-item matching and cluster locations recalculation. Although both algorithms are defined for a single level clustering tree, they can easily be generalized to a multilevel hierarchical clustering tree. The main differences between the inference grammar and these algorithms is that they are deterministic in both steps and it is unclear how to incorporate in them the cluster size control factor $(S - s)$.

For brevity, we define the following "macros" which are used in the grammar:

clusters$[l] := \{\text{cluster}[x, \sigma, l, s, \chi]\}$ // the set of cluster elements in level l

clustersB$[l] := \{\text{clusterB}[x, \sigma, l, s, \chi]\}$ // the set of clusterB elements in level l

The usage of these set-defining macros is allowed in the following inference grammars since the cluster tree has a finite height and finite size at each level; these parameters are given as input and remain fixed during the simulation. Therefore, a grammar rule that uses clusters$[l]$ and clustersB$[l]$ set macros can be handled as if they were rule schemas which are expanded into copies of the rule for every level in the cluster tree.

The following grammars also illustrate the use of special types to partially serialize an intrinsically parallel computing framework. We use both Mathematica style comments "(*...*)" and C++ style comments "//...".

grammar *cluster-Infer* $(\{\text{start}, \text{cluster}(x, \sigma, l, s), \text{item}(\hat{x})\} \rightarrow \{\text{cluster}(\tilde{x}, \sigma, l, s), \text{item}(\hat{x})\})$ {

$\text{start} \rightarrow \text{Step}_1[L - 1]$

$\text{StepFinish}[s, l] \rightarrow \text{Step}[s + 1, l]$

(match children to the clusters *)*

$\{\text{Step}[1, l], \text{clusters}[l], \text{clusters}[l + 1]\} \rightarrow \{\text{StepFinish}[1, l], \text{clusters}[l], \text{clustersB}[l + 1]\}$

via *match-subcluster*

$\{\text{Step}[2, l], \text{clustersB}[l + 1]\} \rightarrow \{\text{StepFinish}[3, l], \text{clusters}[l + 1]\}$ **via** *unblock*

(match the clusters to their parents *)*

$\{\text{Step}[3, l], \text{clusters}[l - 1], \text{clusters}[l]\} \rightarrow \{\text{StepFinish}[4, l], \text{clusters}[l - 1], \text{clustersB}[l]\}$

via *match-subcluster*

$\{\text{Step}[4, l], \text{clustersB}[l]\} \rightarrow \{\text{StepFinish}[4, l], \text{Step}_1[-\text{Mod}[l - 1, L - 1]], \text{clusters}[l]\}$

via *compute-new-mean*

(return to the start *)*

$\text{Step}[5, l] \rightarrow \text{Step}[1, -\text{Mod}[l-1, L-1]]$// return to the start

}

grammar *match-subcluster* $(\{\text{Step}[s, l], \text{clusters}[l], \text{clusters}[l{+}1]\} \rightarrow \{\text{StepFinish}[s, l], \text{clusters}[l], \text{clustersB}[l{+}1]\})\{$

$\{\text{cluster}[x, \sigma, l, s, \chi], \text{cluster}[x', \sigma', l+1, s', \chi']\} \rightarrow$

$\{\text{cluster}[x, \sigma, l, s+1, \chi + x'], \mathit{clusterB}[x', \sigma', l+1, s'+1, \chi' + x]\}$

with $(S - s) * N(x'; x, \sigma)$

(alternative without S *)*

// **with** $N(x'; x, \sigma)$

$\text{Step}[s, l] \rightarrow \text{StepFinish}[s, l]$

}

grammar *unblock* $(\{\text{Step}[s, l], \text{clustersB}[l]\} \rightarrow \{\text{StepFinish}[s, l], \text{clusters}[l]\})\{$

$\text{clusterB}[x, \sigma, l, s, \chi] \rightarrow \text{cluster}[x, \sigma, l, s, \chi]$

$\text{Step}[s, l] \rightarrow \text{StepFinish}[s, l]$

}

grammar *compute-new-mean* $(\{\text{Step}[s, l], \text{clustersB}[l]\} \rightarrow \{\text{StepFinish}[s, l], \text{clusters}[l]\})\{$

*(*first version (most probable move) *)*

$\mathit{clusterB}[x, \sigma, l, s, \chi] \rightarrow \text{cluster}[\frac{\chi}{s}, \sigma, l, 0, 0]$ **with** 1

*(*second version (stochastic move) *)*

// $\mathit{clusterB}[x, \sigma, l, s, \chi] \rightarrow \text{cluster}[\tilde{x}, \sigma, l, 0, 0]$ **with** $\tilde{x} \sim N(\frac{\chi}{s}, \sigma)$

$\text{Step}[s, l] \rightarrow \text{StepFinish}[s, l]$

}

We conducted experiments to verify the accuracy of the inference grammar by generating data sets using *data-generator* and comparing the partitions of the real clusters to the infered ones. Following the methodology presented in [42], the two partitions were matched by finding the optimal pairing using their confusion matrix and the Linear Assignment (LA) measure was calculated to quantify the similiarity between the partitions. In additon, we have measured another comparison index between the partitions, the Normalized Mutual Information (NMI) , which is also described in detail in [42]. Three configurations of the inference grammar were compared and the results are shown in Figures 4 and 5 . The default grammar configuration with the first version (most probable move) of *compute-new-mean* and using the cluster size control factor, which is clearly superior, is

represented by the straight line. The dotted line represents the results when the cluster size control factor is not used and the dashed line represents the results when the second version (stochastic move) of *compute-new-mean* is used instead.

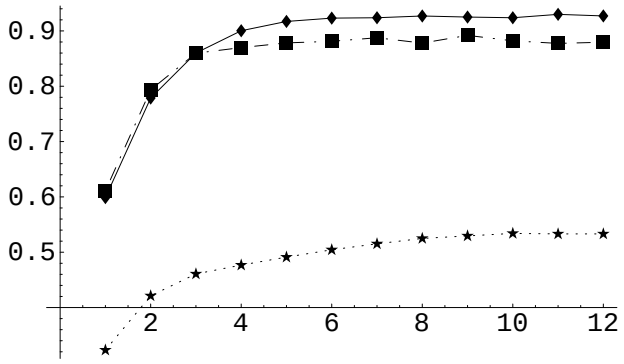


Figure 4: Linear Assignment (LA): Horizontal axis is the number of iterations of the inference grammar where an iteration is an update over all the cluster tree, while the vertical axis is the LA distance. The measurments are averaged over 50 experiments, in which there were 3 clusters of 3 subclusters partitioning 200 data points total.

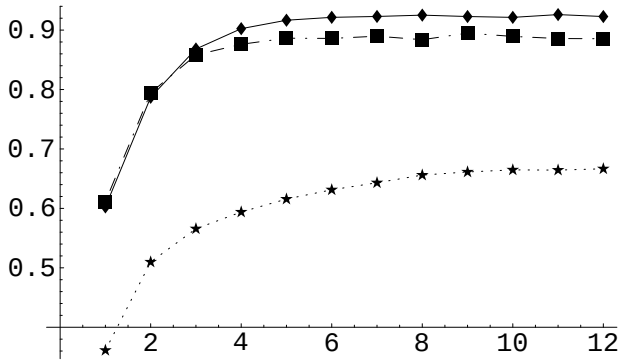


Figure 5: Normalized Mutual Information (NMI): same as figure above for the NMI index.

4.2 Biochemical reaction networks

Given the chemical reaction network syntax

$$\left\{m_a^{(r)} A_a | 1 \leqslant a \leqslant A_{\max}\right\} \xrightarrow{k_{(r)}} \left\{n_b^{(r)} A_b | 1 \leqslant a \leqslant A_{\max}\right\}, \tag{46}$$

we can eliminate the non-SPG syntax of integer-valued "stoichiometric" multiplicities $m_a^{(r)}$ and $n_b^{(r)}$ on the chemical inputs and outputs, by defining an index mapping

$$a(i) = \sum_{c=1}^{A_{\max}} c\Theta\left(\sum_{d=1}^{c-1} m_d^{(r)} < i \leqslant \sum_{d=1}^{c} m_d^{(r)}\right) = \begin{cases} 1 & \text{if } 0 < i \leqslant m_1^{(r)} \\ 2 & \text{if } m_1^{(r)} < i \leqslant m_1^{(r)} + m_2^{(r)} \\ \cdots & \cdots \\ a & \text{if } \sum_{c=1}^{a-1} m_c^{(r)} < i \leqslant \sum_{c=1}^{a} m_c^{(r)} \\ \cdots & \cdots \end{cases}$$

and likewise for $b(j)$ as a function of $\{n_b^{(r)}\}$. Then (Equation 46) can be translated to the following equivalent grammar syntax for the multisets of parameterless terms

$$\left\{\tau_{a(i)} | 0 < i \leqslant \sum_{c=1}^{A_{\max}} m_c^{(r)}\right\}_* \rightarrow \left\{\tau_{a'(j)} | 0 < j \leqslant \sum_{c=1}^{A_{\max}} n_c^{(r)}\right\}_* \qquad \textbf{with} \quad k_{(r)}$$

whose semantics is the time-evolution generator

$$\hat{O}_r = k_{(r)} \left[\prod_{i \in \text{rhs}(r)} \hat{a}_{a(i)}\right] \left[\prod_{j \in \text{lhs}(r)} a_{b(j)}\right] \quad . \tag{47}$$

This generator is equivalent to the stochastic process model of mass-action kinetics for the chemical reaction network (Equation 46).

In this application, the Time Ordered Product Expansion with $H_1 = \hat{H}$ and $H_0 = -D$ represents a sampling algorithm that is essentially the same as the Gillespie Stochastic Simulation Algorithm.

4.3 Multicellular systems and biological development

Two examples will establish the applicability of the Dynamical Grammars modeling framework to variable-structure systems arising in biology at the multicellular scale.

4.3.1 *Arabidopsis*

The growing tip (shoot apical meristem) of the vascular plant *Arabidopsis thaliana* provides an illustrative example of a variable-structure dynamical system at three different scales: the molecular, cellular, and organ levels. At the molecular level, the primary molecules are auxin (a plant growth hormone) and PIN1 (a membrane-bound auxin efflux carrier or "pump"). In a simple model [43], protein PIN1 in the membrane of cell i bounding cell j acts as a catalyst in removing auxin molecules from cell i (modeled with annihilation) and simultaneously inserting them into cell j (creation). Reciprocally, auxin acts on PIN1, directing its incorporation into the nearest membrane compartments of neighboring cells and (optionally) locally enhancing its synthesis. The reactions in this positive feedback loop can be simplified as:

$$\left\{ \overset{\text{PIN1}[i,j]}{\text{auxin}[i] \Longrightarrow \text{auxin}[j]}, \overset{\text{auxin}[i]}{\emptyset \Longrightarrow \text{PIN1}[i]}, \overset{\text{auxin}[j]}{\text{PIN1}[i] \Longrightarrow \text{PIN1}[i,j]}, \text{PIN1}[i,j] \rightarrow \emptyset, \text{PIN1}[i,j] \rightarrow \text{PIN1}[i] \right\},$$

where for example

$$\overset{\text{PIN1}[i,j]}{\text{auxin}[i] \Longrightarrow \text{auxin}[j]} =$$
$$\{\text{auxin}[i], \text{PIN1}[i,j] \rightarrow \text{Complex1}, \text{Complex1} \rightarrow \text{auxin}[i], \text{PIN1}[i,j], \text{Complex1} \rightarrow \text{auxin}[j], \text{PIN1}[i,j]\}$$

At the cell level, cells have internal state including the above reactions and also cell mass and position. When mass exceeds a threshold, cell divide. Mass influences the resting length of elastic springs connecting neighboring cells which determine their positions. Positions determine which cells are neighbors, therefore which regulatory subnetworks are connected. There is variable structure both in the objects (cells) and their relationships (communicating neighbors; lineage trees of cell ancestry).

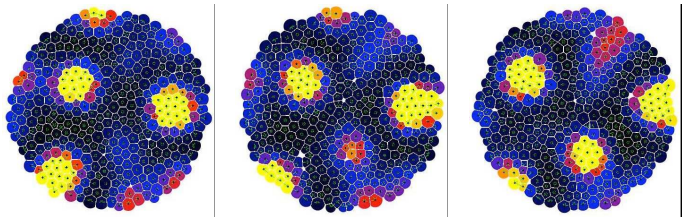


Figure 6: SAM, top view, three time slices. Color = auxin concentration. Emergent peaks correspond to floral meristem primordia. [Images courtesy of Henrik Jönsson.]

Figure 6 shows the dynamical pattern of auxin (yellow/blue scale) evolving over time, from a very similar model as detailed in [43]. Emergent phenomena are the auxin peaks that form off center and move out radially to make room for new peaks. The peaks are hypothesized to determine the position of the primordia for new floral meristems in the phyllotactic pattern of flowers, leaves and branches for the above-ground part of the plant. The variable-structure objects at this scale are the primordia.

The variable structure dynamics illustrated by this model is not analytically tractable but is expressible at each of three levels (the molecular, cellular, and organ levels) using the dynamical grammars formalism. The ability of a cell to divide and interact with its neighbors gives rise, at a coarser spatial and temporal scale, to the ability of a shoot apical meristem to branch and create the floral meristems. This example indicates the potential relevance of Dynamical Grammar framework for multiscale modeling in biology.

4.3.2 *Anabaena*

As a second example of application to multicellular biological modeling, a model [44] of filamentous cyanobacteria *Anabaena catenula* has been fully reimplemented within our "Plenum" prototype im-

plementation of a Dynamical Grammar modeling language, with added stochasticity in the criterion for cell division. The model requires both discrete-time and continuous-duration rules. We used the Time Ordered Product expansion (Section 3.7.2) to derive a simulation algorithm for this case. A typical state of the system is shown in Figure 7(b).

In the *Anabaena* model, cells are attached to two neighboring cells forming a row structure which can be modeled by a simple graph (actually a one-dimensional linked list). We will use the notation of a graph grammar that is formally defined by reduction in Section 5.2. The $w := C(l, c; (w_1, w_2))$ notation assigns an object reference to the variable w, the Object Identifier (OID). C is the cell object which has four parameters: l is the cell length , c is the compound concentration, and (w_1, w_2) are the OID references to the left and right neighbors. A cell with OID can be of two types: $V(w)$, *vegetative,* or $H(w)$, *heterocyst.* When an object is used by a rule but left unchanged, we abbreviate and mention only the object identifier *(w)* in the RHS, as can be seen in the first rule.

Defining the soft threshold function (for small ϵ):

$$\sigma(x) = \frac{\kappa}{1 + e^{-x/\epsilon}} \simeq \begin{cases} \kappa & \text{if } \ x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

we write the grammar

grammar Anabaena {

*(*continuous change in vegetative cell's concentration and length,*

depending on neighbors' parameters)*

$\{w_1 := C(l_1, c_1; (w_0, w)), w := C(l, c; (w_1, w_2)), w_2 := C(l_2, c_2; (w, w_3)), V(w)\}$

$\rightarrow \{w_1, C(l, c; (w_1, w_2)), w_2, V(w)\}$

solving $\{\frac{dc}{dt} = \eta(c_1 + c_2 - 2c) - \mu c, \frac{dl}{dt} = \lambda l\}$

*(*split a vegetative cell into two vegetative cells ,*

if its length is longer than Λ. Connect the new cells to neighbors)*

$\{w := C(l, c; (w_1, w_2)), V(w)\} \rightarrow$

$\{w_{11} := C(kl, c; (w_1, w_{12})), w_{12} := C((1-k)l, c; (w_{11}, w_2)), V(w_{11}), V(w_{12})\}$

with $\sigma(l - \Lambda)$

*(*change a vegetative cell into heterocyst if the concentration*

level drops below certain level)*

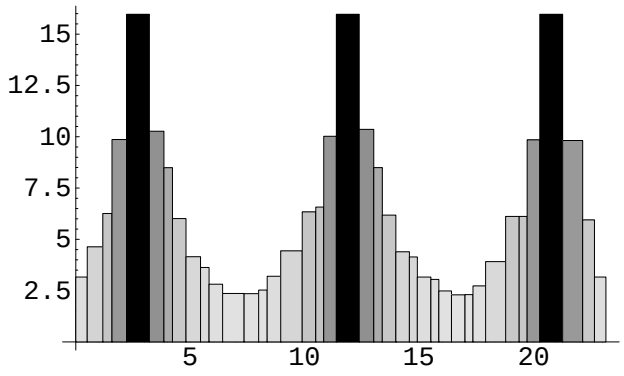
$\{w := C(l, c; (w_1, w_2)), V(w)\} \rightarrow \{w := C(l, c; (w_1, w_2)), H(w)\}$

with $\sigma(\psi - c)$

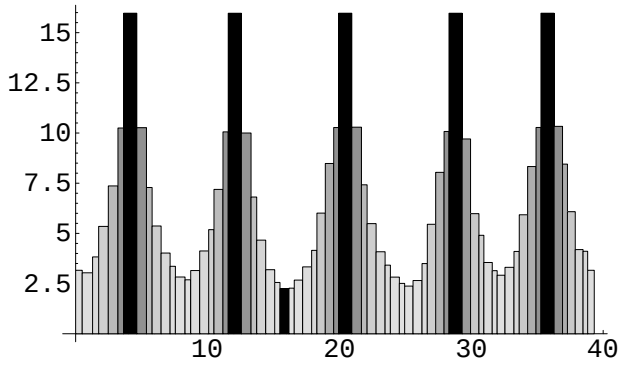
*(*continuous change in heterocyst cell concentration and length*)*

$\{w := C(l, c; (w_1, w_2)), H(w)\} \rightarrow \{w := C(l, c; (w_1, w_2)), H(w)\}$

solving $\{\frac{dl}{dt} = \psi * (\Lambda - l), \frac{dc}{dt} = \varphi * (K - c)\}$



(a)



(b)

Figure 7: Simulation snapshots from the *Anabaena* model: The light and dark gray bars represent vegetative cells, and the black bars represent heterocyst cells. A cell's signal concentration c is signified by both the bar height and the gray level, while the bar width represents the cell's length. System's state after $k =$ (a) 40 and (b) 80 iterations. The simulation was initiated with only 3 vegetative cells. Note the new heterocyst at the location near 17. The distance between the heterocyst cells, both in length and in number of separating cells, remains relatively constant despite cell divisions and continuous growth.

}

As defined by the grammar rules, vegetative cells' signal concentration c is continuously changing according to the ODE $dc/dt = \eta(c_1+c_2-2c)-\mu c$ which combines diffusion and decay. The vegetative cell elongates according to an exponential rate λ until it reaches a threshold length somewhere near the threshold value Λ, and then it divides into two vegetative cells. A transformation into a heterocyst cell occurs when the vegetative cell reaches a signal concentration somewhere near the threshold value ψ. A heterocyst cell's length and concentration level converge exponentialy to the limiting values Λ and K as described by the ODEs in the last rule. Note also the use of terms C, V and H to implement a simple type hierarchy, with C as base type for V and H.

4.3.3 Population biology

To cement the case that SPG's can be used to model dynamics at a wide variety of scales, we observe that common population biology models, both deterministic and stochastic, can be expressed very concisely using biochemical reaction notation as in Section 4.2. For example in the epidemiological susceptible-infected-susceptible (SIS) model [45] there are two types of individuals, S=susceptible (but uninfected) and I=infected. The processes or reactions are infection and recovery:

$$S, I \to I, I \quad \textbf{with} \quad \rho_i \quad \text{//infection}$$
$$I \to S \quad \textbf{with} \quad \rho_r \quad \text{//recovery}$$

The pool state is $(n_I, n_S = N - n_I)$ and the time evolution operator is $\hat{H} = \rho_i \hat{a}_I^2 a_S a_I + \rho_i \hat{a}_S a_I$. Using the conservation law $n_S = n_{\text{total}} - n_I$, we may reduce dimensionality by projecting out n_S from the pool state space, which is accomplished by mapping

$$a_S \mapsto N_S = \text{diag}(n_S) = \text{diag}(n_{\text{total}}) - \text{diag}(n_I) = n_{\text{total}} I_I - N_I,$$

where I_I is the identity matrix in the n_I space. Likewise $\hat{a}_S \mapsto I$ and $n_I^{(\max)} \mapsto n_{\text{total}}$. Then $\hat{H} \mapsto \check{H} = \rho_i \hat{a}_I N_I (I - N_I/n_{\text{total}}) + \rho_i a_I$, which still has all nonnegative entries for $n_I \leqslant N$. From the commutation relations we calculate $\check{H} = \rho_i (1 - 1/n_{\text{total}}) \hat{a}_I^2 a_I - (\rho_i / n_{\text{total}}) \hat{a}_I^3 {a_I}^2 + \rho_i a_I$. This could be formally interpreted in terms of birth and death rules (first and third terms of $\check{H}$) for a single type as in Section 4.1.1, together with an unusual negative-probability rate rule "$I, I \to I, I, I$ **with** $-\rho_i/n_{\text{total}}$", in some as-yet undefined generalization of Equation 18. But such an interpretation of $\check{H}$ is not essential.

5 Reductions

A number of other frameworks and formalisms can be reduced to SPG's as just defined. We give a sampling here.

5.1 Logic programs and cardinality operators

Consider a logic program (e.g. in pure Prolog) consisting of Horn clauses of positive literals

$$p_1 \wedge ... \wedge p_n \Rightarrow q, n \geqslant 0.$$

Axioms have $n = 0$. We can *translate* each such clause into a monotonic SPG rule

$$p_1, ..., p_n \to q, p_1, ..., p_n \tag{48}$$

where each different literal p_i or q denotes an unparameterized type τ_a with $n_a \in \{0, ...n_a^{\max}\} = \{0, 1\}$. Since there is no **with** clause, the rule firing rates default to $\rho = 1$. The corresponding time-evolution operator is monotonic as in Equation 21:

$$\hat{H} = \sum_r \hat{O}_r = \sum_r \left[\prod_{i \in \text{rhs}(r) \setminus \text{lhs}(r)} \hat{a}_{a(i)} \right] \left[\prod_{j \in \text{lhs}(r)} N_{b(j)} \right] \tag{49}$$

Here $N_{b(j)} = \hat{a}_{b(j)} a_{b(j)}$ removes and then replaces all the p_i's. As before, the operators within each product commute.

The semantics of the logic program is its least model or minimal interpretation. It can be computed (Knaster-Tarski theorem) by starting with no literals in the "pool" and repeatedly drawing all their consequences according to the logic program. This is equivalent to converging to a fixed point $\Psi^*(\Gamma) \cdot |\mathbf{0}\rangle$ of the grammar consisting of rules in the form of (Equation 48).

Of course this convergence may not happen in a finite time. Just as for SPG's like clustergen (Section 4.1), the fixed point $\Psi^*(\Gamma) \cdot |\mathbf{0}\rangle$ may not be reachable with a finite number of rule-firings if for example every equilibrium distribution has support on infinitely many terms (and each rule only adds finitely many terms to a finite initial condition such as $|\mathbf{0}\rangle$), or if there is no equilibrium distribution.

More general clauses include negative literals $\neg r$ on the LHS:

$$p_1 \wedge ... \wedge p_n \wedge \neg r_1 \wedge ... \wedge \neg r_m \Rightarrow q, \quad n, m \geqslant 0 \tag{50}$$

or even more general cardinality constraint atoms $0 \leqslant l \leqslant |S| = \sum_{i \in A} \Theta(p_i) \leqslant u \leqslant \infty$ [46], where $S = \{i | p_i\}$. For negative literals, l=u=0, the more general SPG concept is the NotExists ($\nexists_i$) quantifier of Section 2.4.3. It prevents a match if the pool contains any parameterized terms of the specified type. This constraint cannot be expressed in operator algebra for bosonic terms ($n_{\max} = \infty$) using polynomials in creation and annihilation operators alone, but it can be expressed by enlarging the basis operator set $\{B_\alpha\}$ to include

$$Z_i = \begin{pmatrix} 1 & 0 & 0 & \cdots \\ 0 & 0 & 0 & \\ 0 & 0 & 0 & \\ \vdots & & & \ddots \end{pmatrix} \quad \text{for} \quad n_{\max} = \infty; \quad Z_i = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = I_i - N_i \quad \text{for} \quad n_{\max} = 1$$

for each parameterized term indexed by i (e.g by $i = r_1$ through r_m above). The semantics of nonexistence expressions can then be built out of (potentially large) products of "zero" operators Z_i for different i's, for example a factor of $\prod_{k=1}^{m} Z_{r_k}$ for the $\neg r_k$ negative literals in Equation 50.

Atoms with function symbols may be admitted using parameterized terms $\tau_a(x)$.

5.1.1 Set cardinality operators and universal quantifier

Other cardinality constraints of the form $0 \leqslant l \leqslant |S| \leqslant u \leqslant \infty$ can be expressed in operator algebra by again enlarging the basis operator set $\{B_\alpha\}$ beyond the basic creation and annihilation operators. For example the cardinality of a set S of positive literals $\{p_i | i \in A\}$ is computed by the diagonal operator

$$N_S = \log_2 \bigotimes_{i \in A} (I_i + \Theta(p_i) N_i)$$

(where Θ is defined by Equation 1). Both I_i and N_i are diagonal matrices in each subspace i. Here we are using $n_a^{\max} = 1$, $\log_2(1 \text{ or } 2) = 0$ or 1 and $\log \prod = \sum \log$ to implement the cardinality

operator N_S as follows. Each factor of the cross product is diagonal, so the entire cross-product is also diagonal, so (a) its elements are products of the factor elements, each equal to 1 or 2, hence cross-produce elements are equal to powers of 2, and (b) the logarithm is applied elementwise (for diagonal operators). Taking this elementwise logarithm gives us zero or one power(s) of 2 for each element of the cross product diagonal. This just counts the number of factors of 2 rather than 1 in the cross product over i, which is the number of i's for which the predicate $p_i = (i \in S)$ is true, which is the cardinality of S. So N_S is a diagonal operator whose elements are the cardinalities of S in different pure states. Thus for any pure state P, we have the vector equation $N_S \cdot P = (|S| \text{ in state } P)P$ and therefore the scalar equation $\mathbf{1} \cdot N_S \cdot P = (|S| \text{ in state } P)$. This result is a straightforward generalization of the ordinary cardinality operator N_i for a single type τ_i, for which $\mathbf{1} \cdot N_i \cdot P = (|\text{terms } \tau_i| \text{ in state } P)$. If P is a mixed state we get averages $\langle \cdot \rangle_P$ over the distribution P instead:

$$\mathbf{1} \cdot N_i \cdot P = \langle |\text{terms } \tau_i| \rangle_P \qquad \text{and} \qquad \mathbf{1} \cdot N_S \cdot P = \langle |S| \rangle_P.$$

A further elementwise thresholding function can be applied element-by-element to the nonzero diagonal terms of such an operator:

$$\theta_{lu}(n) \equiv \Theta(l \leqslant n \leqslant u)$$

from which we finally deduce the desired operator expression $\theta_{lu}(N_S)$, which for any pure or mixed state P satisfies

$$\mathbf{1} \cdot \theta_{lu}(N_S) \cdot P = \left\langle \Theta(l \leqslant |S| = \sum_{i \in A} \Theta(p_i) \leqslant u) \right\rangle_P \quad .$$

Neither of the operator→operator functions $\log_2(\cdot)$ or $\theta_{lu}(\cdot)$ have polynomial expansions, so neither N_S nor $\theta_{lu}(N_S)$ are in the operator polynomial ring generated by creation and annihilation operators alone. Possibly, sufficient approximations are.

The universal quantifier of Section 2.4.3 may be viewed as an extreme version of a cardinality operator: it should find *all* elements of a set specified in the LHS of a rule, and (temporarily) annihilate them. It appears to be yet more difficult to translate into operator algebra, since a naive rendition using Equation 18 would involve a product of annihilation operators each raised to the power of the current number of objects of a given type: ${a_i}^{N_i}$ to remove all "N_i" occurrences of a parameterized term τ_i, prior to restoring some or all of them with suitable creation operators. In the case of a null RHS, this expression may be given meaning as follows: $a|n\rangle = n|n-1\rangle$, so $a^k|n\rangle = (n)_k|n-k\rangle$, so $a^n|n\rangle = n!|0\rangle$ and

$$\hat{a}_i^{N_i} \to \begin{pmatrix} 0 & 1! & 2! & 3! & \dots \\ 0 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 0 & \end{pmatrix} = |n_i = 0\rangle \left(\sum_{n=0}^{n_{\max}} n_i! \langle n_i = n| \right)$$

However, nonnull RHS's will require further work.

5.2 Semantics of object identifiers (OIDs) and graph grammars

Graph grammars are composed of local rewrite rules for graphs (see for example [47]). We now express a class of graph grammars in terms of SPG's. A precursor example for graph grammars is

the *Anabaena* model in which the graph nodes represent cells and the edges are connections between adjacent cells. Pointers between objects exist in the *Anabaena* grammar from Section 4.3.2, where cell objects are connected to neighboring cell objects in a row. For example, the second rule in the *Anabaena* model which divides a vegetative cell in two is:

$$\{w := C(l, c; (w_1, w_2)), V(w)\} \rightarrow$$
$$\{w_{11} := C(kl, c; (w_1, w_{12})), w_{12} := C((1-k)\, l, c; (w_{11}, w_2)), V(w_{11}), V(w_{12})\}$$

This rule can be converted automatically into the following form, which does not use special syntax for Object Identifiers:

$$\{C(w, l, c, w_1, w_2), V(w), \text{OIDGen}(\text{NextOID})\} \rightarrow$$
$$\{C(\text{NextOID}, kl, c, w_1, \text{NextOID} + 1), C(\text{NextOID} + 1,$$
$$(1-k)\, l, c, \text{NextOID}, w_2)\,, V(w_{11}), V(w_{12}), \text{OIDGen}(\text{NextOID} + 2)\}$$

Thus the $w := C(...)$ notation associates a unique object reference to the variable w, the Object Identifier.

The following syntax introduces Object Identifier (OID) labels L_i for each parameterized term, and allows labelled terms to point to one another through a graph of such labels. The graph is related to two subgraphs of neighborhood indices $N(i, \sigma)$ and $N'(j, \sigma)$ specific to the input and output sides of a rule. Like types or variables, the label symbols appearing in a rule are chosen from an alphabet $\{L_\lambda | \lambda \in \Lambda\}$. Unlike types but like variables X_c, the label symbols $L_{\lambda(i)}$ actually denote unique values in some discrete domain such as the nonnegative integers, thus serving as unique addresses or object identifiers.

A graph grammar rule is of the form, for some nonnegative-integer-valued functions $N(i, \sigma)$, $N'(j, \sigma)$, and one-to-one nonnegative-integer-valued functions $\lambda(i)$ and $\lambda'(i)$ for which $(\lambda(i) = \lambda(j)) \Rightarrow (i = j)$, $(\lambda'(i) = \lambda'(j)) \Rightarrow (i = j)$:

$$\left\{L_{\lambda(i)} := \tau_i(x_{a(i)}; \left[L_{N(i,\sigma)} || \sigma \in 1..\sigma^{\max}_{a(i)}\right]) | i \in \mathcal{I}\right\}$$
$$\rightarrow \left\{L_{\lambda(i)} | i \in \mathcal{I}_1 \subseteq \mathcal{I}\right\} \cup \left\{L_{\lambda'(j)} := \tau_j(x'_{a'(j)}; \left[L_{N'(j,\sigma)} || \sigma \in 1..\sigma^{\max}_{a'(j)}\right]) | j \in \mathcal{J}\right\}$$
$$\textbf{with} \;\; \rho_r(\left\{x'_{a'(j)}\right\} \mid \left\{x_{a(i)}\right\})$$

(compare to Equation 5). Note that the fanout of the graph is limited by $\sigma^{\text{cur}}_i \leqslant \sigma^{\max}_{a(i)}$. Let

$$\mathcal{I} = \mathcal{I}_1 \cup \mathcal{I}_2 \;\; \text{and} \;\; \mathcal{I}_1 \cap \mathcal{I}_2 = \varnothing$$
$$\mathcal{J}_1 = \left\{j \in \mathcal{J} \wedge \left(\exists i \in \mathcal{I}_2 | \lambda(i) = \lambda'(j)\right)\right\}$$
$$\mathcal{J}_2 = \left\{j \in \mathcal{J} \wedge \left(\nexists i \in \mathcal{I}_2 | \lambda(i) = \lambda'(j)\right)\right\}$$
$$\mathcal{I}_3 = \left\{i \in \mathcal{I}_2 \wedge \left(\nexists j \in \mathcal{J}_1 | \lambda(i) = \lambda'(j) \subseteq \mathcal{I}_2\right)\right\}$$

This syntax may be translated to an ordinary non-graph grammar rule. The input objects indexed by $\mathcal{I}$ are divided into those which don't change at all ($\mathcal{I}_1$) and those which change ($\mathcal{I}_2$) including those which disappear ($\mathcal{I}_3$). The output objects are divided into the unchanged inputs

($\mathcal{I}_1$), the modified input objects ($\mathcal{J}_1$), the novel output objects ($\mathcal{J}_2$), and a record of the destroyed input objects ($\mathcal{I}_3$). The translation is as follows (where NextOID is a variable, and OIDGen and Null are types reserved for the translation):

$$\begin{gathered}
\left\{\tau_{a(i)}(L_{\lambda(i)}, x_{a(i)}, \left[L_{N(i,\sigma)}||\sigma \in 1..\sigma_i^{\text{cur}}\right])|i \in \mathcal{I}\right\}, \text{OIDGen(NextOID)} \\
\rightarrow \left\{\tau_{a(i)}(L_{\lambda(i)}, x_{a(i)}, \left[L_{N(i,\sigma)}||\sigma \in 1..\sigma_i^{\text{cur}}\right])|i \in \mathcal{I}_1\right\} \cup \\
\left\{\tau_{a'(j)}(L_{\lambda'(j)}, x'_{a'(j)}, \left[L_{N'(j,\sigma)}||\sigma \in 1..\sigma_j^{\text{cur}}\right])[|j \in \mathcal{J}_1] \wedge [i \in \mathcal{I}_2] \wedge \left(\lambda(i) = \lambda'(j)\right)\right\} \cup \\
\left\{\tau_{a'(j)}(L_{\lambda'(j)}, x'_{a'(j)}, \left[L_{N'(j,\sigma)}||\sigma \in 1..\sigma_j^{\text{cur}}\right])|j \in \mathcal{J}_2\right\} \cup \left\{\text{Null}(L_{\lambda(i)})|i \in \mathcal{I}_3\right\} \\
\cup \left\{\text{OIDGen(NextOID} + |\mathcal{J}_2|)\right\} \\
\textbf{with} \;\; \rho_r(\left\{x'_{a'(j)}\right\} | \left\{x_{a(i)}\right\}) \prod_{j \in \mathcal{J}_2} \delta_K(L_{\lambda'(j)}, \text{NextOID} + |\{j' \in \mathcal{J}_2 | j' < j\}|)
\end{gathered}$$

which already has a defined semantics $\Psi_{c/d}$. Note that all set membership tests can be done at translation time because they do not use information that is only available dynamically during the grammar evolution. Optionally we may also add a rule schema (one rule per type, τ_a) to eliminate any dangling pointers:

$$\begin{gathered}
\tau_a(L_{\lambda(1)}, x, \left[L_{N(1,\sigma)}||\sigma \in 1..\sigma_1^{\text{cur}}\right]), \text{Null}(L_{\lambda(2)}) \\
\rightarrow \tau_a(L_{\lambda(1)}, x, \left[L_{N(1,\sigma)}|N(1,\sigma) \neq \lambda(2)||\sigma \in 1..\sigma_1^{\text{cur}}\right]), \text{Null}(L_{\lambda(2)}) \\
\textbf{with} \;\; \rho_{\text{cleanup}} \sum_{\sigma \in 1..\sigma^{\max}} \delta_K(L_{N(1,\sigma)}, L_{\lambda(2)})
\end{gathered}$$

As a practical matter, the assignment of unique OID's to new RHS terms could be handled in a less global (hence less serialized) and more distributed way than in the foregoing translation to SPG's. For any rule firing, each newly created RHS term could inherit a uniquely altered OID from one of the LHS terms arbitrarily designated as its parent. For example parent terms could keep track of the number of such "OIDchildren" they have had, and each new child could be labelled with childOID=ζ(parent OID, #OIDchildren). If ζ is a hash function then there would be a small chance of OID conflict, as a tradeoff for having all OIDs fit inside a constant-sized address space. If ζ represents the concatenation of integer strings, there can be no conflicts but the OID size can grow indefinitely. Such integer strings were eliminated without OID conflict by using a change of variables, in the related context of constrained energy functions and Boltzmann distributions arising from a class of context-free SPG's [48].

As an example of a graph grammar, discrete link "color" labels can be used to specify a flexible meta-graph-grammar in terms of a matrix $G^{ab}_{i_n j_n} \in \{0,1\}$ of allowed color transitions from parent to child links as a function of predetermined child node labels (i_k, j_k) [49]. In the following grammar, boldface indices $\boldsymbol{i}, \boldsymbol{j}$ refer to sequences $(i_1, i_2, ...i_{k-1})$ and $(i_j, j_2, ...j_{k-1})$ which can index nodes in a binary tree, and the operation $(\boldsymbol{i}, i_k)$ yields the concatenated sequence $(i_1, i_2, ...i_k)$ that can index child nodes. Also the variable $A_{(\boldsymbol{i}, i_k)}$ is a 0/1-valued "aliveness" variable, that indicates which rooted subtree of the infinite binary tree has resulted from a node-generating grammar such as *binaryclustergen* (Section 2.1). If the subindices i_l are allowed to range over a larger set than {0,1} then the notation generalizes to trees with some larger fixed fanout than binary. With these notations, we can link up nodes randomly and recursively as follows:

grammar (discrete-time) *graph-recursion* (start $\to \{\text{node}(\boldsymbol{i}), \text{link}(a, \boldsymbol{i}, \boldsymbol{j}\})$ {

start $\to \text{node}((0)), \text{link}(1, (0), (0))$

$N := \text{node}(\boldsymbol{i}) \to N, \{\text{node}((\boldsymbol{i}, i_k))|A_{(\boldsymbol{i},i_k)} = 1 \wedge i_k < i_{\max}\}$

under $E = \mu \sum_{i_k} A_{(\boldsymbol{i},i_k)}$

$\text{link}(a, \boldsymbol{i}, \boldsymbol{j}), N := \text{node}((\boldsymbol{i}, i_k)), M := \text{node}((\boldsymbol{i}, i_k)) \to$

$\{\text{link}(b, (\boldsymbol{i}, i_k)(\boldsymbol{j}, j_k))|G^{ab}_{i_k j_k} = 1\}, N, M$

}

5.2.1 Transformations of SPG's

Given graph grammar capabilities as above, with discrete-time semantics, it is possible to implement a graph grammar that performs a syntactic transformation on an SPG. We have done this in the Plenum implementation of SPG's (Section 2), implementing a very simple syntactic transformation that swaps the left and right hand sides of each grammar rule, effectively reversing the arrows, without modifying the rate function. This is not generally a correct solution to the problem (Section 5.2.1) of creating an inverse inference algorithm for a given generative SPG, but it illustrates the potential for using (meta-) SPG's to implement such SPG morphisms if they can be found. One advantage of this approach is that a meta-SPG may also be able to check its own conditions of applicability before acting, if they are known mathematically. Other selectively applicable grammar transformations may be implementable by meta-grammars, such as transformations from continuous-time or discrete-time SPG's to efficient discrete-time SPG simulation algorithms. In this way, SPG's and DG's in principle can become dynamical systems themselves, in addition to their usual role in specifying dynamical systems via their semantic maps.

5.2.2 String rewrite rule grammars

Strings may be encoded as one-dimensional graphs using either a singly or doubly linked list data structure as was shown for the *Anabaena* example above. String rewrite rules of the form

$$\left[\tau_{a(i)}(x_i)|i \in \mathcal{I}_L\right] \to \left[\tau_{a'(j)}(y_j)|j \in \mathcal{I}_R\right] \quad \textbf{with} \quad \rho_r([x_i|i \in \mathcal{I}_L], [y_j|j \in \mathcal{I}_R]) \tag{51}$$

(note ordering of arguments, in contrast to Equation 5) can be emulated as graph rewrite rules, whose semantics are defined above. This form is capable of handling many L-system grammars [13]. As usual, if ρ_r is not supplied it defaults to 1. The *Anabaena* example with continuous growth rules replaced by discrete cell state change rules thus provides an elementary example of a string rewrite grammar.

5.3 Stochastic and ordinary differential equations

There are SPG rule forms corresponding to stochastic differential equations governing diffusion and transport. Given the SDE or equivalent Langevin equation (which specializes to a system of ordinary

differential equations when $\eta(t) = 0$):

$$dx_i = v_i([x_k])dt + \sigma([x_k])dW \quad \text{or} \tag{52}$$

$$\frac{dx_i}{dt} = v_i([x_k]) + \eta_i(t) \tag{53}$$

under some conditions on the noise term $\eta(t)$ the dynamics can be expressed [33] as a Fokker-Planck equation for the probability distribution $P(\{x\}, t)$:

$$\frac{\partial P(x,t)}{\partial t} = -\sum_i \frac{\partial}{\partial x_i} v_i(x) P(x,t) + \sum_{ij} \frac{\partial^2}{\partial x_i \partial x_j} D_{ij}(x) P(x,t) \tag{54}$$

Let $P(y, t|x, 0)$ be the solution of this equation given initial condition $P(y, 0) = \delta(y - x) = \prod_k \delta(y_k - x_k)$ (with Dirac delta function appropriate to the particular measure μ used for each component). Then at $t = 0$,

$$\frac{\partial P(y,0|x,0)}{\partial t} \equiv \rho([y_i] \,|\, [x_i]) = -\sum_i \frac{\partial}{\partial y_i} v_i(x) \delta(y - x) + \sum_{ij} \frac{\partial^2}{\partial y_i \partial y_j} D_{ij}(x) \delta(y - x)$$

Thus the probability rate $\rho([y_i]|[x_i])$ is given by a differential operator acting on a Dirac delta function. It can be decomposed into drift and diffusion:

$$\rho_{\text{drift}}([y_i] \,|\, [x_i]) = -\sum_i \frac{\partial}{\partial y_i} v_i(x) \prod_i \delta(y_i - x_i) \tag{55}$$

$$\rho_{\text{diffusion}}([y_i] \,|\, [x_i]) = \sum_{ij} \frac{\partial^2}{\partial y_i \partial y_j} D_{ij}(x) \prod_i \delta(y_i - x_i) \tag{56}$$

from which by (Equation 20) we construct the evolution generator operators $O_{\text{FP}} = O_{\text{drift}} + O_{\text{diffusion}}$, where

$$O_{\text{drift}} = -\int dx \int dy \hat{a}(y) a(x) \left(\sum_i \nabla_{y_i} v_i(y) \prod_k \delta(y_k - x_k) \right) \tag{57}$$

$$O_{\text{diffusion}} = \int dx \int dy \hat{a}(y) a(x) \left(\sum_{ij} \nabla_{y_i} \nabla_{y_j} D_{ij}(y) \prod_k \delta(y_k - x_k) \right) \tag{58}$$

The second order derivative terms give diffusion dynamics and also regularize and promote continuity of probability in parameter space both along and transverse to any local drift direction. So, these two time-evolution operators may be identified with the corresponding differential operators $-\sum_i \frac{\partial}{\partial x_i} v_i(\{x\})$ and $\sum_{ij} \frac{\partial^2}{\partial x_i \partial x_j} D_{ij}(\{x\})$in the Fokker-Planck partial differential equation (Equation 54), respectively.

As a check one can use the relations

$$|z\rangle = \hat{a}(z\ )|0\rangle\,, \quad \langle w| = \langle 0|a(w)$$
$$[a(x), \hat{a}(y)] = \delta(y - x)[1 + N(x)Q(N(\ x), n^{\max})]$$
$$\langle w|z\rangle = \delta(w - z)$$

to calculate operator matrix elements $\langle w|\exp(tO_{\mathrm{FP}})|z\rangle$. For example,

$$\begin{aligned}
\langle w\,|O_{\mathrm{drift}}|\,z\rangle &= -\left\langle w\left|\int dx\int dy\left(\sum_i \nabla_{y_i} v_i(y)\delta(y-x)\right)\hat{a}(y)a(x)\hat{a}(z)\right|0\right\rangle \\
&= -\left\langle w\left|\int dx\int dy\left(\sum_i \nabla_{y_i} v_i(y)\delta(y-x)\right)\hat{a}(y)\delta(z-x)[1+N(x)]\right|0\right\rangle \\
&= -\int dx\int dy\left(\sum_i \nabla_{y_i} v_i(y)\delta(y-x)\right)\delta(z-x)\,\langle w|y\rangle \\
&= -\int dy\left(\sum_i \nabla_{y_i} v_i(y)\delta(y-z)\right)\delta(w-y) \\
&= +\int dy\delta(y-z)\left(\sum_i v_i(y)\nabla_{y_i}\delta(w-y)\right) \\
&= \sum_i v_i(z)\nabla_{z_i}\delta(w-z)
\end{aligned}$$

Computing higher powers yields

$$\begin{aligned}
\langle w\,|\exp(tO_{\mathrm{drift}})|\,z\rangle &= \exp(t\sum_i v_i(z)\nabla_{z_i})\delta(w-z) \\
&= \delta(w-\left(\{z(0)=z\}+\int_0^t v_i(z(t))dt\right))
\end{aligned}$$

which is a formal solution of the drift-only differential equation $(dx_i)/dt = v_i(x_k)$.

Diffusion/drift rules can be combined with chemical reaction rules to describe reaction-diffusion systems [27, 50]. The foregoing approach can be generalized to encompass partial differential equations (PDE's) and stochastic partial differential equations (SPDE's) [37]; these extensions are taken to be part of the definition of allowed rules in a DG. With suitable PDE's, one can then express models of dynamical manifolds (as in General Relativity) and dynamical manifold embeddings using explicit or level set representations.

The foregoing operator expressions all correspond to natural extended-time processes given by the evolution of continuous differential equations (DE's). The operator semantics of the differential equations is given in terms of derivatives of delta functions in the manner of (Equation 52), (Equation 53), (Equation 55), (Equation 56). The "**solving**" keyword may be used to introduce such ODE/SDE rule clauses in the SPG syntax. This syntax could be eliminated in favor of a "**with**" clause by using derivatives of delta functions in the rate expression $\rho_{\mathrm{DE}}(\{y_i\}|\{x_i\})$, provided that such generalized functions are in the Banach space $\mathcal{F}(V)$ as a limit of functions.

Thus, ODE's and SDE's have operator expressions for their semantics and can therefore be added to the allowed syntax of Dynamical Grammars. These kinds of dynamics can now be freely combined with reaction networks and other discrete-time event processes whose dynamics is also defined by operator algebra generators. Indeed if a grammar includes both DE rules and non-DE rules, a conventional DE solver can be used to compute $\exp((t_{n+1}-t_n)O_{\mathrm{FP}})$ in the time-ordered product

expansion (Equation 40) for $\exp(tH)$ as a hybrid simulation algorithm for discontinuous (jump) stochastic processes combined with stochastic differential equations. The analogous combination for grammars with deterministic dynamics semantics appears in [44] which exhibits simulation algorithms, in [15] which introduces the "**solve**" keyword for L-systems, and in [16] which specifies a dynamical grammar modeling framework for developmental biology.

5.4 Discussion: Relevance to artificial intelligence and computational science

The relevance of the modeling language defined here to *artificial intelligence* includes the following points. First, pattern recognition and machine learning both benefit foundationally from better, more descriptively adequate probabilistic domain models. As an example, Section 4.1 exhibits hierarchical clustering data models expressed very simply in terms of SPG's and relates them to recent work. Graphical models are probabilistic domain models with a fixed structure of variables and their relationships, by contrast with the inherently flexible variable sets and dependency structures resulting from the execution of stochastic parameterized grammars. Thus SPG's, unlike graphical models, are Variable-Structure Systems (defined in [37]), and consequently they can support compositional description of complex situations such as multiple object tracking in the presence of cell division in biological imagery [51]. Second, the reduction of many divergent styles of modelling and computation to a common SPG syntax and operator algebra semantics enables new possibilities for deep semantic-level integration of hybrid forms of modeling and computing. For example one could combine logic programming with probability distribution models and quantitative areas such as stochastic processes, or one could combine discrete-event stochastic and differential equation models as discussed in Section 5.3, in possibly new ways.

As a third point of AI relevance, from SPG probabilistic domain models it is possible to systematically derive *algorithms* for simulation (as in Section 3.7.2) and inference either by hand or automatically. Of course, inference algorithms are not as well worked out yet for SPG's as for graphical models. SPG's have the advantage that simulation or inference algorithms could be expressed again in the form of SPG's, a possibility demonstrated by the clustering inference grammar in Section 4.1.3 and also in part by the encoding of logic programs as SPG's. Since both model and algorithm are expressed as SPG's, it is possible to use SPG transformations that preserve relevant quantities (Section 5.2.1) as a technique for deriving such novel algorithms or generating them automatically. For example we have taken this approach to rederive by hand the Gillespie simulation algorithm for chemical kinetics. This derivation is different from the one in Section 3.7.2. Because SPG's encompass graph grammars it is even possible in principle to express families of valid SPG transformations as meta-SPG's. All of these points apply *a fortiori* to Dynamical Grammars as well.

The relevance of the modeling language defined here to *computational science* includes the following points. First, as argued previously, multiscale models must encompass and unify heterogeneous model types such as discrete/continuous or stochastic/deterministic dynamical models; this unification is provided by SPG's and DG's. In this way, the fine scale dynamics (such as reaction events) in a two-scale model can be approximated by a coarse-scale model (for example, mass-action kinetic ODEs for concentrations). Such a coarse-scale approximation of the fine scale dynamics can then be integrated, by simple addition of time-evolution generators, with the dynamics of intrinsically coarse-scale processes, such as cell division and cell-type changing events in a developmental biology model. Second, a representationally adequate computerized modeling language can be of great as-

sistance in constructing mathematical models in science, as demonstrated for biological regulatory network models by Cellerator [52] and other cell modeling languages. DG's extend this promise to more complex, spatiotemporally dynamic, variable-structure system models such as occur in biological development. Third, machine learning techniques could in principle be applied to find simplified approximate or reduced models of emergent phenomena within complex domain models. Indeed, one indicator of intelligent behavior is the ability to construct operationally adequate internal models of the environment. In that case the forgoing AI arguments apply to computational science applications of machine learning as well.

Both for artificial intelligence and computational science, future work will be required to determine whether the prospects outlined above are both realizable and compelling. The present work is intended to provide a mathematical foundation for achieving that goal.

6 Conclusions and future directions

We have established a syntax and semantics for a probabilistic modeling language based on independent processes leading to events linked by a shared set of objects. The semantics is based on a polynomial ring of time-evolution operators. The syntax is in the form of a set of rewrite rules. Variable-binding occurs by integration of the rule firing rate function over parameter value spaces. Stochastic Parameterized Grammars and the more general Dynamical Grammars expressed in this language can compactly encode disparate models: generative cluster data models, biochemical networks, logic programs, graph grammars, string rewrite grammars, and stochastic differential equations among other others. The time-ordered product expansion connects this framework to powerful methods from quantum field theory and operator algebra.

One future direction for Dynamical Grammar applications is in dynamic spatial modeling for biological development ([37, 14, 13, 53, 16]). To this end it will be interesting to explore the relationship between graph grammars for spatial structures and their continuum limits including PDE's, both encoded as DG's. For multicellular structures it may be useful to consider simultaneously continuum limits at both the subcellular scale and the multicellular tissue level. At the latter scale, developmental systems can act as dynamic information-processing manifolds embedded dynamically in $\mathbb{R}^{d=3}$.

Also in the future, it may be useful to formalize non-textual, labelled graph representations for the syntax of SPG's and Dynamical Grammars. Using graph grammars such a representation could allow the semantics functions $\Psi_{c/d}$ to be applied iteratively. To create such a graph representation, one could use diagrammatic representations such as Markov Random Fields or Bayes Networks for the language $\mathcal{L}_R$ which specifies the firing rate functions $\rho_r([y_j],[x_i])$ which are also members of function spaces $\mathcal{F}(V)$, provided that such diagrams are augmented with a nonnegative scalar multiplier to represent unnormalized firing rates. In this connection Dependency Diagrams [37] generalize many other such representations. For the actual grammar itself, there exists a bipartite graph $\{G_{ra}, G_{ar}\}$ of types τ_a (indexed by a) and rules (indexed by r), in which type node a links to rule r ($G_{ra} = 1$) iff some term of type τ_a occurs in the LHS multiset of rule r, and rule r links to type node a ($G_{ar} = 1$) iff rule r contains some term of type τ_a in its RHS multiset. This bipartite graph is similar to the set of *arcs* between *places* (our types) and *transitions* (our rules) in a Petri Net, and indeed there are generalizations such as Colored Petri Nets [19] in which CPN *tokens* (our

grounded term instances or objects) contain *values* (our vector of parameter values). However our operator semantics appears to be nonstandard in detail by comparison with the existing Petri Net literature, and the SPG syntax contains features not found in Petri Nets such as rule variables, parameter vectors, type signatures, polymorphic type signatures, oblivious semantics for all possible (Section 3.4) rule firings and firing rate functions.

Acknowledgements. Useful discussions with Pierre Baldi, Ashish Bhan, Michael Duff, Sergei Nikolaev, Przemyslaw Prusinkiewicz, Alex Sadovsky, Bruce Shapiro, Padhraic Smyth, Michael Turmon, and Max Welling are gratefully acknowledged. The detailed criticisms of an anonymous reviewer were very helpful and caused us to reformulate the discrete-time SPG semantics and to introduce Proposition 2. The work was supported in part by the National Science Foundation's Frontiers in Biological Research (FIBR) program, award number EF-0330786, by a Biomedical Information Science and Technology Initiative (BISTI) grant (number R33 GM069013) from the National Institute of General Medical Sciences, and by the Center for Cell Mimetic Space Exploration (CMISE), a NASA University Research, Engineering and Technology Institute (URETI), under award number #NCC 2-1364.

7 Appendices

7.1 Time-ordered operator expansion

We rederive the Time-Ordered Product expansion (Equation 2.14 of [27] Equation 4.29 of [33]) by elementary probabilistic means as follows. For arbitrary operators H_0 and H_1we wish to calculate

$$\exp(tH)\cdot p_0 = \exp(t\,(H_1+H_0))\cdot p_0$$

To this end we introduce an extra variable z, which can ultimately be set to 1, in order to create a generating function that keeps track of the number of times operator H_1 is applied in polynomial expansions of the exponential:

$$\begin{aligned}
S(z) &= \exp(t\,(H_1 z+H_0))\cdot p_0 = \sum_{n=0}^{\infty} s_k z^k \\
&= \sum_{k=0}^{\infty}\frac{z^k}{k!}\Big[\partial_z{}^k \exp(t\,(H_1 z+H_0))\ \Big]_{z=0}\cdot p_0 \\
&= \sum_{k=0}^{\infty}\frac{z^k}{n!}\Big[\partial_z{}^k \sum_{l=0}^{\infty}\frac{(t\,(H_1 z+H_0))^l}{l!}\ \Big]_{z=0}\cdot p_0 \\
&= \sum_{k=0}^{\infty}\frac{z^k}{k!}\Big[\sum_{l=k}^{\infty}\frac{1}{l!}\sum_{\{0\leqslant i_p\leqslant l-k\}\wedge\sum_{p=0}^{k} i_p=l-k} k!(tH_0)^{i_k}t\hat{H}(tH_0)^{i_{k-1}}\cdots t\hat{H}(tH_0)^{i_0}\ \Big]\cdot p_0 \\
&= \sum_{k=0}^{\infty} z^k t^k\Big[\sum_{l=0}^{\infty}\frac{1}{(l+k)!}\sum_{\{0\leqslant i_p\leqslant l\}\wedge\sum_{p=0}^{k} i_p=l}(tH_0)^{i_k}H_1(tH_0)^{i_{k-1}}\cdots H_1(tH_0)^{i_0}\Big]\cdot p_0 \\
&= \sum_{k=0}^{\infty} z^k t^k\Big[\sum_{l=0}^{\infty}\sum_{\{0\leqslant i_p\leqslant l\}\wedge\sum_{p=0}^{k} i_p=l}\frac{\prod_{p=0}^{k}(i_p)!}{\left(\sum_{p=0}^{k} i_p+k\right)!}\frac{(tH_0)^{i_k}}{(i_k)!}H_1\frac{(tH_0)^{i_{k-1}}}{(i_{k-1})!}\cdots H_1\frac{(tH_0)^{i_0}}{(i_0)!}\Big]\cdot p_0 \\
&= \sum_{k=0}^{\infty} z^l t^l\Big[\sum_{\{0\leqslant i_p\leqslant\infty\}}\frac{\prod_{p=0}^{k}(i_p)!}{\left(\sum_{p=0}^{k}(i_p+1)-1\right)!}\frac{(tH_0)^{i_k}}{(i_k)!}H_1\frac{(tH_0)^{i_{k-1}}}{(i_{k-1})!}\cdots H_1\frac{(tH_0)^{i_0}}{(i_0)!}\Big]\cdot p_0 \\
&= \sum_{k=0}^{\infty} z^k t^k\Big[\sum_{\{0\leqslant i_p\leqslant\infty\}}\frac{\prod_{p=0}^{k}\Gamma(i_p+1)}{\Gamma\left(\sum_{p=0}^{k}(i_p+1)\right)}\frac{(tH_0)^{i_k}}{(i_k)!}H_1\frac{(tH_0)^{i_{k-1}}}{(i_{k-1})!}\cdots H_1\frac{(tH_0)^{i_0}}{(i_0)!}\Big]\cdot p_0
\end{aligned}$$

Now we use the Multinomial-Dirichlet normalization integral

$$\frac{\prod_{p=0}^{n}\Gamma(i_p+1)}{\Gamma\left(\sum_{p=0}^{n}(i_p+1)\right)}=\int_0^1 d\theta_0\cdots\int_0^1 d\theta_k\delta\left(\sum_{p=1}^{k}\theta_p-1\right)\prod_{p=0}^{k}(\theta_p)^{i_p}.$$

Accordingly,

$$\begin{aligned}S(z)=\sum_{k=0}^{\infty}z^k\ \ t^k\Big[\sum_{\{0\leqslant i_p\leqslant\infty\}}\int_0^1 d\theta_0\cdots\int_0^1 d\theta_k\delta\Big(\sum_{p=1}^{k}\theta_p-1\Big)\Big(\prod_{p=0}^{k}(\theta_p)^{i_p}\Big)\\ \frac{(tH_0)^{i_k}}{(i_k)!}H_1\frac{(tH_0)^{i_{k-1}}}{(i_{k-1})!}\cdots H_1\frac{(tH_0)^{i_0}}{(i_0)!}\Big]\cdot p_0\\ =\sum_{k=0}^{\infty}z^k\ \ t^k\Big[\sum_{\{0\leqslant i_p\leqslant\infty\}}\int_0^1 d\theta_0\cdots\int_0^1 d\theta_k\delta\Big(\sum_{p=1}^{k}\theta_p-1\Big)\\ \frac{(\theta_k tH_0)^{i_k}}{(i_k)!}H_1\frac{(\theta_{k-1}tH_0)^{i_{k-1}}}{(i_{k-1})!}\cdots H_1\frac{(\theta_0 tH_0)^{i_0}}{(i_0)!}\Big]\cdot p_0\\ =\sum_{k=0}^{\infty}z^k\ \ t^k\Big[\int_0^1 d\theta_0\cdots\int_0^1 d\theta_k\delta\Big(\sum_{p=1}^{k}\theta_p-1\Big)\\ \sum_{\{0\leqslant i_0\leqslant\infty\}}\frac{(\theta_k tH_0)^{i_0}}{(i_k)!}H_1\sum_{\{0\leqslant i_1\leqslant\infty\}}\frac{(\theta_{k-1}tH_0)^{i_1}}{(i_{k-1})!}\cdots H_1\sum_{\{0\leqslant i_k\leqslant\infty\}}\frac{(\theta_0 tH_0)^{i_k}}{(i_0)!}\Big]\cdot p_0\\ =\sum_{k=0}^{\infty}z^k\ \ t^k\Big[\int_0^1 d\theta_0\cdots\int_0^1 d\theta_k\delta\Big(\sum_{p=1}^{k}\theta_p-1\Big)\exp(\theta_k tH_0)H_1\exp(\theta_{k-1}tH_0)\cdots H_1\exp(\theta_0 tH_0)\Big]\cdot p_0\end{aligned}$$

Thus

$$S(z)=\sum_{k=0}^{\infty}z^k\left[\int_0^t d\tau_0\cdots\int_0^t d\tau_k\delta\Big(\sum_{p=1}^{k}\tau_p-t\Big)\exp(\tau_k H_0)H_1\exp(\tau_{k-1}H_0)\cdots H_1\exp(\tau_0 H_0)\right]\cdot p_0 \quad (59)$$

In summary (since p_0 was never used in the above calculations),

$$\begin{aligned}&\exp(t\,(H_1+H_0))\\ &\qquad=\sum_{k=0}^{\infty}\Big[\int_0^t d\tau_0\cdots\int_0^t d\tau_k\delta\Big(\sum_{p=1}^{k}\tau_p-t\Big)\exp(\tau_k H_0)H_1\exp(\tau_{k-1}H_0)\cdots H_1\exp(\tau_0 H_0)\Big]\quad .\end{aligned}$$

Alternatively, define $t_1=\tau_0$, $t_2=t_1+\tau_1$, ... $t_{n+1}=t_n+\tau_n=t$. Then the evolution of the state vector is given by

$$\begin{aligned}&\exp(t\,(H_1-H_0))\cdot p_0=\\ &\sum_{k=0}^{\infty}\ \Big[\int_0^t dt_1\int_{t_1}^t dt_2\cdots\int_{t_{k-1}}^t dt_k\exp((t-t_k)\,H_0)H_1\exp((t_k-t_{k-1})\,H_0)\cdots H_1\exp(t_1H_0)\Big]\cdot p_0\end{aligned}$$

In the special case $H_1 = \hat{H}, H_0 = -D$ this simplifies to:

$$\exp(t\left(\hat{H} - D\right)) \cdot p_0 =$$
$$\sum_{k=0}^{\infty} \left[\int_0^t dt_1 \int_{t_1}^t dt_2 \cdots \int_{t_{k-1}}^t dt_n \exp(-(t-t_k)\,D)\hat{H}\exp(-(t_k-t_{k-1})\,D)\cdots\hat{H}\exp(-t_1 D)\right] \cdot p_0 \tag{60}$$

Since D is diagonal, the terms $\exp(-\tau D)$ are analytically calculable and easy to simulate with large jumps in time. Between these easy terms are interposed single powers of $\hat{H}$ representing the occurrence of discrete-time grammar events that must be simulated.

These last two expression for $\exp(t(\hat{H}-D))$ have a significant interpretation in the case of reaction kinetics: they correspond to the Gillespie algorithm for stochastic simulation. The exponential distribution of waiting times until the next reaction is given by $\exp(-\tau D)$, which depends on the state of the system but doesn't change it, and the reaction events are modeled by the interdigitated powers of $\hat{H}$.

The same derivation can be accomplished for *any* decomposition of H into a solvable part H_0 (above, $-D$, but it need not be diagonal) plus a more difficult term H_1(here, $\hat{H}$):

$$\exp(t\,(H_0 + H_1)) \cdot p_0 =$$
$$\sum_{k=0}^{\infty} \left[\int_0^t dt_1 \int_{t_1}^t dt_2 \cdots \int_{t_{k-1}}^t dt_k \exp((t-t_k)\,H_0)H_1\exp((t_k-t_{k-1})\,H_0)\cdots H_1\exp(t_1 H_0)\right] \cdot p_0 \tag{61}$$

This is one formulation of the time-ordered product expansion.

This perturbative approach is equivalent to the use of perturbative methods including Feynman diagram calculations in quantum field theory, except for an occasional factor of $i{=}\sqrt{-1}$ which would turn our probabilities into the complex-valued probability factors of quantum mechanics, as discussed in Section 3.4.2.

7.2 Relation of discrete-time and continuous-time grammars

The continuous and discrete-time grammar executions are related as follows. After continuous time t, the *joint* probability density on the states of the original system and on the number k of discrete events (inlcuding rule firings and post-termination pseudo-events with rate ω) is

$$p_k = \Pr_{\text{continuous}}\left(|\,\{n_a(x_a)\}\rangle \middle| k, |\,\{m_a(x_a)\}\rangle\right)$$

and has the generating function $S(z)$ (from Equation 59) with $H_1 = \tilde{H}$and $H_0 = \tilde{D}$ ($=\hat{H}$ and D respectively if $\omega{=}0$):

$$S(z) = \exp\left(t\left(z\tilde{H} - \tilde{D}\right)\right) \cdot p_0$$

Expanding in z, successive powers of z will accompany successive powers of $\tilde{H} = \hat{H} + \omega \operatorname{diag}(\Theta(\mathbf{1} \cdot \hat{H} = 0))$ which can be interpreted as rule firings (applications of the operator $\hat{H}$) or, if $\omega \neq 0$, as post-termination pseudo-events that don't change anything. We can expand this generating function as in Appendix 1 above:

$$S(z) =$$

$$= \sum_{k=0}^{\infty} z^k \quad \left[\int_0^t d\tau_0 \cdots \int_0^t d\tau_k \delta\left(\sum_{p=1}^{k} \tau_p - t \right) \exp(-\tau_k \tilde{D}) \tilde{H} \quad \exp(-\tau_{k-1} \tilde{D}) \cdots \tilde{H} \exp(-\tau_0 \tilde{D}) \right] \cdot p_0$$

Equating terms in the generating function for the joint distribution over k,

$$\Pr_{\text{continuous}}(\cdot, k | t, p_0) = \int_0^t d\tau_0 \cdots \int_0^t d\tau_k \delta\left(\sum_{p=1}^{k} \tau_p - t \right) \exp(-\tau_k \tilde{D}) \tilde{H} \quad \exp(-\tau_{k-1} \tilde{D}) \cdots \tilde{H} \exp(-\tau_0 \tilde{D}) \cdot p_0 \tag{62}$$

which can be disagreggated into a joint distribution on inter-event delay times $\tau_0 \cdots \tau_k$:

$$\Pr_{\text{continuous}}(\cdot, \tau_0 \cdots \tau_k, k | t, p_0) = \delta\left(\sum_{p=1}^{k} \tau_p - t \right) \exp(-\tau_k \tilde{D}) \tilde{H} \quad \exp(-\tau_{k-1} \tilde{D}) \cdots \tilde{H} \exp(-\tau_0 \tilde{D}) \cdot p_0$$

By construction all entries of the diagonal matrix D are nonnegative. If all entries of D are also bounded below by $\delta > 0$ (and in particular there are no terminal states with zero exit probability), and $\epsilon \ll 1$ is a desired relative agreement in probabilities, then we may introduce a probability distribution $\Pr(t)$ on time that is uniform for $t \in [0, T]$ where $T \gg k |\log(\epsilon / k)| / \delta$:

$$\Pr(t) = \begin{cases} 1/T & \text{if} \quad t \in [0, T] \\ 0 & \text{otherwise} \end{cases} = \Theta(0 \leqslant t \leqslant T)/T$$

Then

$$\Pr_{\text{continuous}}(\cdot, \tau_0, ..., \tau_k, k | T, p_0) \quad = \int_0^\infty dt \Pr_{\text{continuous}}(\cdot, \tau_0, ..., \tau_k, k | t, p_0) \Pr(t)$$

$$= \frac{\Theta\left(\sum_{p=1}^{k} \tau_p < T \right)}{T} \exp(-\tau_k D) \tilde{H} \quad \exp(-\tau_{k-1} \tilde{D}) \cdots \tilde{H} \exp(-\tau_0 \tilde{D}) \cdot p_0$$

$$\Pr_{\text{continuous}}(\cdot, \tau_k = 0, k | T, p_0) = \int_0^\infty d\tau_0 \cdots \int_0^\infty d\tau_{k-1} \Pr_{\text{continuous}}(\cdot, \tau_0, ..., \tau_{k-1}, \tau_k = 0, k | T, p_0)$$

$$= \int_0^\infty d\tau_0 \cdots \int_0^\infty d\tau_{k-1} \frac{\Theta\left(\sum_{p=1}^{k} \tau_p < T \right)}{T} \exp(-(\tau_k = 0) \tilde{D}) \tilde{H} \quad \exp(-\tau_{k-1} \tilde{D}) \cdots \tilde{H} \exp(-\tau_0 \tilde{D}) \cdot p_0$$

$$= \int_0^T d\tau_0 \cdots \int_0^T d\tau_{k-1} \frac{\Theta\left(\sum_{p=1}^{k} \tau_p < T \right)}{T} \tilde{H} \quad \exp(-\tau_{k-1} \tilde{D}) \cdots \tilde{H} \exp(-\tau_0 \tilde{D}) \cdot p_0$$

Since all entries of D are also bounded below by $\delta > 0$, then for any ϵ we can find a $T > 0$ so large that

$$\frac{\int_{T/k}^{T} d\tau \quad \exp(-\tau\delta)}{\int_{0}^{T} d\tau \quad \exp(-\tau\delta)} < \frac{\int_{T/k}^{\infty} d\tau \quad \exp(-\tau\delta)}{\int_{0}^{T/k} d\tau \quad \exp(-\tau\delta) + \int_{T/k}^{T} d\tau \quad \exp(-\tau\delta)}$$

which,

$$< \frac{\int_{T/k}^{\infty} d\tau \quad \exp(-\tau\delta)}{\int_{0}^{T/k} d\tau \quad \exp(-\tau\delta)} = \frac{\exp(-T\delta/k)}{1 - \exp(-T\delta/k)} \ll \epsilon/k \tag{63}$$

In particular, $T \gg k|\log(\epsilon/k)|/\delta$ suffices.

$$\Pr_{\text{continuous}}(\cdot, \tau_k = 0, k|T, p_0)$$

$$\cong \left[\int_0^{T/k} d\tau_0 \cdots \int_0^{T/k} d\tau_{k-1} \frac{\Theta\left(\sum_{p=1}^{k} \tau_p < T\right)}{T} \tilde{H} \quad \exp(-\tau_{k-1}\tilde{D}) \cdots \tilde{H} \exp(-\tau_0 \tilde{D}) \cdot p_0 \right] \times (1 + O(\epsilon))$$

$$= \left[\frac{1}{T} \int_0^{T/k} d\tau_0 \cdots \int_0^{T/k} d\tau_{k-1} \tilde{H} \quad \exp(-\tau_{k-1}\tilde{D}) \cdots \tilde{H} \exp(-\tau_0 \tilde{D}) \cdot p_0 \right] \times (1 + O(\epsilon))$$

$$= \left[\frac{1}{T} \tilde{H} \left[\int_0^{T/k} d\tau_{k-1} \exp(-\tau_{k-1}\tilde{D}) \right] \cdots \tilde{H} \left[\int_0^{T/k} d\tau_0 \exp(-\tau_0 \tilde{D}) \right] \cdot p_0 \right] \times (1 + O(\epsilon))$$

$$\cong \left[\frac{1}{T} \left(\tilde{H} \quad \tilde{D}^{-1} \right)^k \cdot p_0 \right] \times (1 + O(\epsilon))$$

where we have used

$$\int_0^{T/k} d\tau \exp(-\tau D) = \left[1 - \exp(-T\tilde{D}/k)\right] \tilde{D}^{-1}$$

$$\cong \begin{cases} T/k & \text{for} \quad \tilde{D}_{ss} = 0 \\ \left(1/\tilde{D}_{ss}\right) \times (1 + O(\epsilon/k)) & \text{for} \quad \tilde{D}_{ss} \neq 0 \end{cases} = \tilde{D}'[T/k] \times (1 + O(\epsilon/k)) \quad .$$

Sorting into nonterminal and terminal pool states ($\mathbf{1} \cdot \hat{H} \cdot p > 0$ or $\mathbf{1} \cdot \hat{H} \cdot p = 0$) and using block matrix notation as in Equation 34,

$$\mu(\omega, T/k) \equiv \begin{cases} T/k & \text{if} \quad \omega = 0 \\ \omega & \text{otherwise} \end{cases}$$

$$\tilde{H} \left[\int_0^{T/k} d\tau_{k-1} \exp(-\tau_{k-1}\tilde{D}) \right] \cong \begin{pmatrix} \tilde{H}_{11} & 0 \\ \tilde{H}_{21} & \omega I \end{pmatrix} \begin{pmatrix} \text{diag}\left(\mathbf{1} \cdot \tilde{H}_1\right)^{-1} & 0 \\ 0 & \mu(\omega, T/k) I \end{pmatrix} \times (1 + O(\epsilon/k))$$

which,

$$= \begin{pmatrix} \tilde{H}_{11} \cdot \text{diag}(\mathbf{1} \cdot \tilde{H}_1)^{-1} & 0 \\ \tilde{H}_{21} \cdot \text{diag}(\mathbf{1} \cdot \tilde{H}_1)^{-1} & I\Theta(\omega > 0) \end{pmatrix} \times (1 + O(\epsilon/k)) = \tilde{H} \cdot \tilde{D}' \times (1 + O(\epsilon/k)) \tag{64}$$

(last step uses Equation 34 and Equation 36) Now we conditionalize on k and τ_k.

Case I: $\Pr_{\text{continuous}}(\tau_k = 0, k|p_0) = 0$: Then

$$0 = \mathbf{1} \cdot \left(\tilde{H} \quad \tilde{D}'\right)^k \cdot p_0 \Rightarrow \omega = 0 \quad \Rightarrow \left(\tilde{H} = \hat{H} \text{and} \tilde{D} = D\right); \text{thus}$$
$$0 = \mathbf{1} \cdot \left(\hat{H} \quad D'\right)^k \cdot p_0 = 1 - p_k(\triangle) = \Pr_{\text{discrete}}(\text{not halted}|k \text{ events})$$

(by Equation 37). So

$$\Pr_{\text{continuous}}(\tau_k = 0, k|T, p_0) = 0 = \Pr_{\text{discrete}}(\text{not halted}|k \text{ events})$$

Case II: $\Pr_{\text{continuous}}(\tau_k = 0, k|T, p_0) > 0$: We can conditionalize on k and τ_k by dividing by the sum over the remaining parameters, namely the pool states $|\{n_a(x_a)\}\rangle$:

$$\Pr_{\text{continuous}}(\cdot|k, \tau_k = 0, T, p_0) =$$
$$\frac{\Pr_{\text{continuous}}(\cdot, \tau_k = 0, k|T, p_0)}{\Pr_{\text{continuous}}(\tau_k = 0, k|T, p_0)} = \frac{\Pr_{\text{continuous}}(\cdot, \tau_k = 0, k|T, p_0)}{\mathbf{1} \cdot \Pr_{\text{continuous}}(\cdot, \tau_k = 0, k|T, p_0)}$$
$$\cong \left[\frac{\left(\tilde{H} \quad \tilde{D}'\right)^k \cdot p_0}{\mathbf{1} \cdot \left(\tilde{H} \quad \tilde{D}'\right)^k \cdot p_0}\right] \times (1 + O(\epsilon))$$

using Equation 64.

If $\omega \neq 0$ (and therefore in the present context $\omega \geqslant \delta$) then $\mathbf{1} \cdot (\tilde{H} \quad \tilde{D}')^k \cdot p_0 = 1$ by the definition of $\tilde{D}$ ($\tilde{D} = \text{diag}(\mathbf{1} \cdot \tilde{H})$), and

$$\Pr_{\text{continuous}}(\cdot|k, \tau_k = 0, T, p_0) = \left(\tilde{H} \quad \tilde{D}'\right)^k \cdot p_0 \times (1 + O(\epsilon)).$$

so we conclude that, with a relative approximation error of $O(\epsilon)$,

$$\Pr_{\text{continuous}}(\cdot|k, \tau_k = 0, T, p_0) \cong \Pr_{\text{discrete}}(\cdot|k, p_0)$$

If $\omega = 0$ we can conditionalize the discrete distribution on not halting using Equation 37:

$$\Pr_{\text{discrete}}(\cdot|\text{not halted}, k \text{ events}, p_0) = \frac{\Pr_{\text{discrete}}(\cdot, \text{not halted}|k \text{ events}, p_0)}{\Pr_{\text{discrete}}(\text{not halted}|k \text{ events})} = \frac{\left(\tilde{H} \quad \tilde{D}'\right)^k \cdot p_0}{\mathbf{1} \cdot \left(\tilde{H} \quad \tilde{D}'\right)^k \cdot p_0}$$

so we conclude that, with a relative approximation error of $O(\epsilon)$,

$$\Pr_{\text{continuous}}(\cdot|k, \tau_k = 0, T, p_0) \cong \Pr_{\text{discrete}}(\cdot|\text{not halted}, k \text{ events}, p_0).$$

This is also true of the case $\omega \neq 0$ since in that case there is zero probability of halting and $\Pr_{\text{discrete}}(\cdot|k, p_0) = \Pr_{\text{discrete}}(\cdot|\text{not halted}, k \text{ events}, p_0)$.

7.3 Shadow grammars

A related calculation establishes another approximation of discrete by continuous-time SPG semantics, in which the instantaneous examination of the state resulting from the k'th rule firing is done not by Bayes' rule inference, but by adding "flash-freezing" events as part of the continuous-time dynamics.

Suppose there is one or more terminal states (with zero probability of exit), and we want to know the probability that the continuous-time system falls into a terminal state after k rule firings and then stays there forever. This assumption can be met as follows:

Augment the original grammar Γ with a new one-parameter global "live" object added to every RHS and LHS, whose Boolean "liveness" parameter must be True in the LHS for any rule firing to occur (and remains True in the RHS). This doubles the state space by adding a "shadow" terminal state to every original state of the pool. Also for every rule r, add a "shadow" rule $r'(r)$ with the same LHS as r, whose RHS switches the liveness parameter to False and leaves all the other input terms unchanged. The shadow rule r'should have the same probability rate function as does r, times a small constant factor $\kappa > 0$.

Thus each rule r in Γ

$$\left\{\tau_{a(i)}(x_i)|i \in \mathcal{I}_L\right\} \to \left\{\tau_{a'(j)}(y_j)|j \in \mathcal{I}_R\right\} \quad \textbf{with} \quad \rho_r([x_i], [y_j])$$

becomes two rules in $\Gamma'(\Gamma)$, the modified rule r and its shadow rule $r'(r)$:

$$\begin{aligned}
\text{live(True)}, \left\{\tau_{a(i)}(x_i)|i \in \mathcal{I}_L\right\} &\to \text{live(True)}, \left\{\tau_{a'(j)}(y_j)|j \in \mathcal{I}_R\right\} \quad \textbf{with} \quad \rho_r([x_i], [y_j]) \\
\text{live(True)}, \left\{\tau_{a(i)}(x_i)|i \in \mathcal{I}_L\right\} &\to \text{live(False)}, \left\{\tau_{a'(j)}(y_j)|j \in \mathcal{I}_R\right\} \quad \textbf{with } \kappa\rho_r([x_i], [y_j])
\end{aligned}$$

and no rule has live(False) in its LHS.

If we order the states so that first all the live states appear, then all the nonlive shadow states, then using 2x2 block matrix notation the full $\tilde{H}^*$and $\tilde{D}^*$ can be written in terms of the original $\tilde{H}$ and $\tilde{D}$ (using the matrix "prime" operation of Equation 29):

$$\tilde{H}^* = \begin{pmatrix} \tilde{H} & 0 \\ \kappa\tilde{H} & \omega I \end{pmatrix} \quad ; \quad \tilde{D}^* = \begin{pmatrix} (1+\kappa)\,\tilde{D} & 0 \\ 0 & \omega I \end{pmatrix} \quad ; \quad p_0^* = \begin{pmatrix} p_0 \\ 0 \end{pmatrix}$$

$$\text{also} \tilde{D}^{*\prime} = \begin{pmatrix} \tilde{D}'/(1+\kappa) & 0 \\ 0 & (\Theta(\omega > 0)/\omega)\, I \end{pmatrix} \Rightarrow \tilde{H}^*\tilde{D}^{*\prime} = \begin{pmatrix} \tilde{H}\tilde{D}'/(1+\kappa) & 0 \\ \kappa\tilde{H}\tilde{D}'/(1+\kappa) & \Theta(\omega > 0) I \end{pmatrix}$$

Assume all other entries of $\tilde{D}$ (besides those associated with terminal states, whether they are shadow states or not) are bounded below by $\delta > 0$ (and in particular there are no terminal states with zero exit probability), and $\epsilon \ll 1$ is a desired relative agreement in probabilities. Also define T so that $T \gg k|\log(\epsilon/k)|/\delta$. Then

$$p_k^*(t) = \Pr_{\text{continuous}}(\cdot, k|t, p_0) = \frac{\Pr_{\text{continuous}}(\cdot|k, t, p_0)}{\mathbf{1} \cdot \Pr_{\text{continuous}}(\cdot|k, t, p_0)}$$

where

$$\Pr_{\text{continuous}}(\cdot|k, t=T, p_0) =$$
$$\left[\int_0^T d\tau_0 \cdots \int_0^T d\tau_k \delta\left(\sum_{p=1}^{k} \tau_p - t\right) \quad \exp(-\tau_k \tilde{D}^*)\tilde{H}^* \quad \exp(-\tau_{k-1}\tilde{D}^*)\cdots\tilde{H}^*\exp(-\tau_0\tilde{D}^*)\right] \cdot p_0^*$$

and we want to compute the long-time limit $\lim_{T\to\infty} p_k(T)$ for arbitrary but finite k. In that way the relative error parameter ϵ will become as small as desired, and zero in the limit.

From Equation 62 and Equation 63,

$$\Pr_{\text{continuous}}(\cdot|k, t=T, p_0) \cong$$
$$\left[\int_0^{T/k} d\tau_0 \cdots \int_0^{T/k} d\tau_{k-1} \int_0^{t} d\tau_k \delta\left(\sum_{p=1}^{k-1} \tau_p + \tau_k - t\right) \exp(-\tau_k \tilde{D}^*)\tilde{H}^* \quad \exp(-\tau_{k-1}\tilde{D}^*)\cdots\tilde{H}^*\exp(-\tau_0\tilde{D}^*)\right]$$
$$\cdot p_0^* \quad \times (1 + O(\epsilon))$$

None of the factors $\exp(-\tau_0 D)$ ... $\exp(-\tau_{k-1} D)$ can contribute anything from the D_{ss}=0 terminal states, because the next matrix multiplication by $\hat{H}$ (to the left in the matrix product) would zero out such contributions. So the only contributions from the terminal states come from the innermost integral:

$$\int_0^t d\tau_k \delta\left(\sum_{p=1}^{k-1} \tau_p + \tau_k - t\right) \exp(-\tau_k \tilde{D}^*) = \exp\left(-\left(t - \sum_{p=1}^{k-1} \tau_p\right)\tilde{D}^*\right)$$
$$= \begin{cases} 1 & \text{for} \left(\tilde{D}^*\right)_{ss} = 0 \\ \leqslant \exp(-T\delta/k) = O(\epsilon) & \text{for} \left(\tilde{D}^*\right)_{ss} \neq 0 \end{cases} = \theta(\tilde{D}^*) + O(\epsilon)$$

where define the matrix function $\theta(\tilde{D})$ elementwise on matrix $\tilde{D}$:

$$\theta(\left(\tilde{D}^*\right)_{ss'}) = \begin{cases} 1 & \text{if} s = s' \text{and} \left(\tilde{D}^*\right)_{ss} = 0 \\ 0 & \text{otherwise} \end{cases}$$

Here, at large times t, any nonzero diagonal element becomes O(ϵ) while zero diagonal elements receive a factor of 1. We assumed the terminal state is reachable in any number of rule firings k, in which case only the terminal states will receive O(1) weight

$$\Pr_{\text{continuous}\Gamma'}(\cdot|k, t=T, p_0) \cong \left(\theta(\tilde{D}^*) + O(\epsilon)\right) \times$$
$$\left[\int_0^{T/k} d\tau_0 \cdots \int_0^{T/k} d\tau_{k-1} \tilde{H}^* \exp(-\tau_{k-1}\tilde{D}^*) \quad \cdots \tilde{H}^*\exp(-\tau_0\tilde{D}^*)\right] \cdot p_0^* \times (1 + O(\epsilon))$$
$$\cong \left(\theta(\tilde{D}^*) + O(\epsilon)\right) \times \left[\left(\tilde{H}^* \quad \tilde{D}^{*\prime}\right)^k \cdot p_0^*\right] \times (1 + O(\epsilon))$$

(using the same argument as for Equation 64), where the matrix "prime" operation is defined in Equation 29. Conditionalizing,

$$p_k^*(t=T) = \Pr_{\text{continuous}}(\cdot, k | t=T, p_0) = \frac{\left(\theta(\tilde{D}^*) + O(\epsilon)\right) \times \left[\begin{pmatrix} \tilde{H}^* & \tilde{D}^{*\prime} \end{pmatrix}^k \cdot p_0^*\right] \times (1 + O(\epsilon))}{\mathbf{1} \cdot \left(\theta(\tilde{D}^*) + O(\epsilon)\right) \times \left[\begin{pmatrix} \tilde{H}^* & \tilde{D}^{*\prime} \end{pmatrix}^k \cdot p_0^*\right] \times (1 + O(\epsilon))}$$

Assuming $\omega = 0$, and no original terminal states:

$$\theta(\tilde{D}^*) = \begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix} \quad ; \quad \tilde{H}^* \tilde{D}^{*\prime} = \begin{pmatrix} \tilde{H}\tilde{D}'/(1+\kappa) & 0 \\ \kappa \tilde{H}\tilde{D}'/(1+\kappa) & 0 \end{pmatrix};$$

$$\left(\tilde{H}^* \tilde{D}^{*\prime}\right)^k = \begin{pmatrix} \left(\tilde{H}\tilde{D}'/(1+\kappa)\right)^k & 0 \\ \kappa(\tilde{H}\tilde{D}'/(1+\kappa))^k & 0 \end{pmatrix} = \begin{pmatrix} I & 0 \\ \kappa I & 0 \end{pmatrix} \otimes \left(\tilde{H}\tilde{D}'/(1+\kappa)\right)^k$$

$$\begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ \kappa I & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ \kappa I & 0 \end{pmatrix}$$

$$p_k^*(t=T) \cong \frac{\left(\theta(\tilde{D}^*) + O(\epsilon)\right) \times \left[\begin{pmatrix} \tilde{H}^* & \tilde{D}^{*\prime} \end{pmatrix}^k \cdot p_0^*\right]}{\mathbf{1} \cdot \left(\theta(\tilde{D}^*) + O(\epsilon)\right) \times \left[\begin{pmatrix} \tilde{H}^* & \tilde{D}^{*\prime} \end{pmatrix}^k \cdot p_0^*\right]}$$

Now we calculate

$$\mathbf{1} \cdot \left[\begin{pmatrix} 0 & 0 \\ \kappa & 0 \end{pmatrix} \otimes \left(\tilde{H}\tilde{D}'/(1+\kappa)\right)^k\right] \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes p_0 = \left\{(\kappa, 0) \otimes \left[\mathbf{1} \cdot \left(\tilde{H}\tilde{D}'/(1+\kappa)\right)^k\right]\right\} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes p_0 =$$

$$= \mathbf{1} \cdot \frac{\kappa(\tilde{H}\tilde{D}')^k}{(1+\kappa)^k} \cdot p_0$$

$$p_k^*(t=T) \cong \frac{\left[\begin{pmatrix} 0 & 0 \\ \kappa & 0 \end{pmatrix} \otimes \frac{(\tilde{H}\tilde{D}')^k}{(1+\kappa)^k}\right] \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes p_0}{\mathbf{1} \cdot \frac{\kappa(\tilde{H}\tilde{D}')^k}{(1+\kappa)^k} \cdot p_0} = \frac{\begin{pmatrix} 0 \\ \kappa \end{pmatrix} \otimes \left[\frac{(\tilde{H}\tilde{D}')^k}{(1+\kappa)^k} \cdot p_0\right]}{\kappa \mathbf{1} \cdot \frac{(\tilde{H}\tilde{D}')^k}{(1+\kappa)^k} \cdot p_0} =$$

$$= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \frac{\left(\tilde{H}\tilde{D}'\right)^k \cdot p_0}{\mathbf{1} \cdot \left(\tilde{H}\tilde{D}'\right)^k \cdot p_0}$$

As in the previous proof, we may conclude that, with a relative approximation error of $O(\epsilon)$,

$$\Pr_{\text{continuous}\Gamma'}(\cdot, k | t=T, p_0) = p_k^*(t=T) \cong \Pr_{\text{discrete}\Gamma}(\cdot | \text{not halted}, k \text{ events}, p_0).$$

7.4 Relation of alternative discrete-time and continuous-time grammars

The continuous and discrete-time grammar executions are related as follows. After continuous time t, the joint probability density on the states of the original system and on the number of discrete

rule firings, k, has the generating function

$$S(z) = \sum_{k=0}^{\infty} s_k z^k = \exp\left(t\left(\hat{H}z - D\right)\right) \cdot p_0$$

so that

$$s_k = \mathrm{Coef}_k\left(\exp\left(t\left(\hat{H}z - D\right)\right), z\right) \cdot p_0.$$

An alternative approach to the semantics of the discrete-time grammar is to take the short-time limit of the continuous-time grammar's conditional distribution given that n rule firings occurred:

$$\lim_{t\to 0}\left[s_k/1 \cdot s_k\right] = \lim_{t\to 0}\left[\mathrm{Coef}_k\left(\exp\left(t\left(\hat{H}z - D\right)\right), z\right) \cdot p_0 / 1 \cdot \mathrm{Coef}_n\left(\exp\left(t\left(\hat{H}z - D\right)\right), z\right) \cdot p_0\right].$$

This result follows by a short calculation from the following general expression for S:

$$\begin{aligned}
S(z) &= \sum_{k=0}^{\infty} s_k z^k = \exp\left(t\left(\hat{H}z - D\right)\right) \cdot p_0 \\
&= \sum_{k=0}^{\infty} \frac{z^k}{k!}\left[\partial_z{}^k \exp\left(t \quad \left(\hat{H}z - D\right)\right) \quad\right]_{z=0} \cdot p_0 \\
&= \sum_{k=0}^{\infty} \frac{z^k}{k!}\left[\partial_z{}^k \sum_{l=0}^{\infty} \frac{\left(t\left(\hat{H}z - D\right)\right)^l}{l!} \quad\right]_{z=0} \cdot p_0 \\
&= \sum_{k=0}^{\infty} \frac{z^k}{k!}\left[\sum_{l=k}^{\infty} \frac{1}{l!} \sum_{\{0\leqslant i_p \leqslant l-k\}\wedge \sum_{p=0}^{k} i_p = l-k} k!(-tD)^{i_k} t\hat{H}(-tD)^{i_{k-1}} \cdots t\hat{H}(-tD)^{i_0} \quad\right] \cdot p_0 \\
&= \sum_{k=0}^{\infty} z^k \left[\sum_{l=0}^{\infty} \frac{1}{(l+k)!} \sum_{\{0\leqslant i_p \leqslant l\}\wedge \sum_{p=0}^{k} i_p = l} t^k (-tD)^{i_k} \hat{H}(-tD)^{i_{k-1}} \cdots \hat{H}(-tD)^{i_0}\right] \cdot p_0
\end{aligned}$$

From this expression we can take the small-time limit, picking out only the $i_p = 0$ terms:

$$\lim_{t\to 0} S(z) = \sum_{k=0}^{\infty} z^k \left[\frac{1}{(l+k)!}|_{(l=0)} t^k \hat{H}^k\right] \cdot p_0 = \sum_{k=0}^{\infty} \frac{z^k t^k}{k!} \hat{H}^k \cdot p_0$$

Thus

$$\lim_{t\to 0}\left[s_k/1 \cdot s_k\right] = \hat{H}^k \cdot p_0 / \left(1 \cdot \hat{H}^k \cdot p_0\right)$$

References

[1] Mjolsness, E.: *Stochastic Process Semantics for Dynamical Grammar Syntax: An Overview.* Paper presented at the Ninth International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida, 4-6 January 2006

[2] Preston, C. J.: *Spatial birth-and-death processes.* Bull. Int. Statist. Inst. , **46(2)**, 371–391 (1977)

[3] Lange, K.: *Applied Probability.* Section 9.6. Springer-Verlag, New York Berlin Heidelberg (2004)

[4] Snyder, D. L., & Miller, M. I. : *Random Point Processes in Time and Space.* Wiley, New York (1991)

[5] Athreyea, K. B., & Ney, P. E.: *Branching Processes*: Dover (1972)

[6] Engel, K., & Nagel, R.: *One-Parameter Semigroups for Linear Evolution Equations.* Springer Graduate Texts in Mathematics 194, New York (2000)

[7] Pearl, J.: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, San Francisco (1988)

[8] Nodelman, U., Shelton, C., & Koller , D.: *Continuous Time Bayesian Networks.* In: Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI), 378–387. Morgan Kaufmann Publishers, San Francisco (2002)

[9] Dean, T., & Kanazawa, K.: *A model for reasoning about persistence and causation.* Comp. Intelligence, **5**, 142–150. (1989)

[10] Sanghai, S., Domingos, P., & Weld, D.: *Relational dynamic Bayesian networks.* Journal of Artificial Intelligence Research. **24**, 759-797. (2005)

[11] Milch, B., Marthi, B., Russell, S., & Sontag, D.: *BLOG: Probabilistic Models with Unknown Objects.* In: Proceedings of the 19th International Joint Conference on Artificial Intelligence. 1352-1359. (2005)

[12] St-Aubin, R., Friedman, J., & Mackworth, A. K.: *A Formal Mathematical Framework for Modeling Probabilistic Hybrid Systems.* Paper presented at the Ninth International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida, 4-6 January 2006

[13] Prusinkiewicz, P., & Lindenmeyer, A.: *The Algorithmic Beauty of Plants.* Springer-Verlag, New York Berlin Heidelberg (1990)

[14] Giavitto, J., & Michel, O.: *MGS: a Programming Language for the Transformations of Topological Collections.* LaMI - Universite d' Evry Val d'Essonne. Technical Report 61-2001. Cited in May 2001

[15] Prusinkiewicz, P., Hammel, M. S., & Mjolsness, E.: *Animation of Plant Development.* Computer Graphics, 351-360 (1993)

[16] Mjolsness, E., Sharp, D. H., & Reinitz, J.: *A Connectionist Model of Development.* Journal of Theoretical Biology, **152(4)**, 429–454. (1991)

[17] Phillips, A., & Cardelli, L., *A Correct Abstract Machine for the Stochastic Pi-Calculus.* http://lucacardelli.name/Bibliography.htm. Cited Sep 2006 (2006)

[18] Petri, C. A.: *Kommunikation mit Automaten.* Dissertation, University of Bonn (1962)

[19] Jensen, K.: *Coloured Petri Nets I, II, III.* Springer-Verlag, New York Berlin Heidelberg (1997)

[20] Genrich, H.: *Predicate/Transition nets.* Advances in Petri nets : APN , **1**, 208–247. (1986)

[21] Haas, P. J.: *Stochastic Petri Nets.* Springer-Verlag, New York Berlin Heidelberg (2006)

[22] Liggett, T. M.: *Interacting Particle Systems.* Springer-Verlag, New York Berlin Heidelberg (1985)

[23] Smith, C., Prusinkiewicz, P., & Samavati, F.: *Local specification of surface subdivision algorithms.* In: J. Pfaltz, M. Nagl & B. Bohlen (Ed.), *Applications of Graph Transformations with Industrial Relevance (AGTIVE 2003): Lecture Notes in Computer Science, vol. 3062* , pp. 313–327. Springer-Verlag, New York Berlin Heidelberg (2003)

[24] Richardson, M., & Domingos, P.: *Markov Logic Networks.* Machine Learning, **62** , 107–136. (2006)

[25] van Kampen, N. G.: *Stochastic Processes in Physics and Chemistry.* North-Holland (1981)

[26] Reed, M., & Simon, B.: *Methods of Modern Mathematical Physics: Functional Analysis I.* Academic Press, New York (1972)

[27] Mattis, D. C., & Glasser, M. L.: *The uses of quantum field theory in diffusion-limited reactions.* Reviews of Modern Physics, **70**, 979–1001. (1998)

[28] McLachlan , R. I., & Quispel, G. R.: *Splitting methods.* Acta Numerica, **11**, 341–434. (2002)

[29] Hatano, N., & Suzuki, M.: *Finding Exponential Product Formulas of Higher Orders.* http://arxiv.org/pdf/math-ph/0506007 Cited in 2 Jun 2005 (2005)

[30] Hall, B. C.: In: *Lie Groups, Lie Algebras, and Representations: An Elementary Introduction* , pp. 63–76. Springer-Verlag, New York Berlin Heidelberg (2003)

[31] Müser, M. H.: *The path-integral Monte Carlo approach of rigid linear molecules in three dimensions.* Molecular Simulation, **17**(131) (1996)

[32] Dyson, F.: Phys. Rev., **75**, 486. (1949)

[33] Risken, H.: *The Fokker-Planck Equation.* Springer-Verlag, New York Berlin Heidelberg (1984)

[34] Gillespie, D. J.: *A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions*, Journal of Computational Physics **2**, 403-434 (1976).

[35] Shachter, R.: *Evaluating influence diagrams.* Operations Research, **33**, 871–882. (1986)

[36] Kanazawa, Koller, D., & Russell, S.: *Stochastic Simulation Algorithms for Dynamic Probabilistic Networks.* Proceedings of Uncertainty in Artificial Intelligence 95. http://citeseer.ist.psu.edu/kanazawa95stochastic.html (1995)

[37] Mjolsness, E.: *Variable-Structure Systems from Graphs and Grammars.* UC Irvine School of Information and Computer Sciences, Irvine. UCI ICS TR# 05-09, http://computableplant.ics.uci.edu/papers/ vbl-Struct_GG_TR.pdf (2005)

[38] Ghahramani, Z.: *Non-parametric Bayesian Methods.* Paper presented at The 21st Conference on Uncertainty in Artificial Intelligence (UAI 2005), University of Edinburgh Edinburgh, Scotland, July 26th-July 29th 2005. http://www.gatsby.ucl.ac.uk/~zoubin/talks/uai05tutorial-b.pdf . Cited in Sep 06 (2006)

[39] Geman, S., & Geman, D.: *Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images.* IEEE Transactions on Pattern Analysis and Machine Intelligence, **6**, 721–741. (1984)

[40] MacQueen, J. B.: *Some Methods for classification and Analysis of Multivariate Observations.* Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability (1967)

[41] Dempster, A., Laird, N., & Rubin, D.: *Maximum likelihood from incomplete data via the EM algorithm.* Journal of the Royal Statistical Society, **Series B**, 1–38. (1977)

[42] Hart, C., et al.: *A mathematical and computational framework for quantitative comparison and integration of large-scale gene expression data.* Nucleic Acids Res, **33**, 2580–2594. (2005)

[43] Jönsson, H., et al.: *An auxin-driven polarized transport model for phyllotaxis.* Proc. Natl. Acad. Sciences USA, **103**(5), 1633–1638. Retrieved 13 January 2006 from http://www.pnas.org/cgi/content/abstract/103/5/1633 (2006)

[44] Federl, P., & Prusinkiewicz, P.: *Solving differential equations in developmental models of multicellular structures expressed using L-systems.*. In M. Bubak, G. van Albada, P. Sloot & J. Dongarra (Ed.), *Proceedings of Computational Science. ICCS 2004, II. Lecture Notes in Computer Science vol.3037*, pp. 65–72. Springer-Verlag, New York Berlin Heidelberg (2004)

[45] Jacquez, J. A., & Simon, C. P.: *The Stochastic SI Model with Recruitment and Deaths. I. Comparison with the Closed SIS Model.* Mathematical Biosciences, **117**, 77–125. (1993)

[46] Cenzer, D., Marek, V. W., & Remmel, J. B. *Using logic programs to reason about infinite sets.* Eighth International Symposium on Artificial Intelligence and Mathematics, http://rutcor.rutgers.edu/~amai/aimath04/accepted.html (2004)

[47] Cuny, J., Ehrig, H., Engels, G., & Rozenberg, G. *Graph Grammars and their Applications to Computer Science.* Springer-Verlag, New York Berlin Heidelberg (1994)

[48] Mjolsness, E. *Symbolic Neural Networks Derived from Stochastic Grammar Domain Models.* In R. Sun & R. Alexandre (Ed.), *Connectionist Symbolic Integration.* Lawrence Erlbaum Associates (1997)

[49] Bhan, A., & Mjolsness, E.: *Static and Dynamic Models of Biological Networks.* Complexity, **11**(6), 57–63. (2006)

[50] Turing, A. M.: *The chemical basis of morphogenesis.* Phil. Trans.Roy. Soc. Lond., **B237**, 37–72. (1952)

[51] Gor, V., Bacarian, T., Elowitz, M., & Mjolsness, E.: *Tracking Cell Signals in Fluorescent Images.* Computer Vision Methods for Bioinformatics (CVMB) workshop, at Computer Vision and Pattern Recognition (CVPR). http://computableplant.ics.uci.edu/CVPR-2005.pdf .Cited in Jun 2005. (2005)

[52] Shapiro, B. E., et al.: *Cellerator: extending a computer algebra system to include biochemical arrows for signal transduction simulations* . Bioinformatics, **19**, 677–678. (2003)

[53] Fracchia, F. D.: *Integrating lineage and interaction for the visualization of cellular structures.*. In J. Cuny, H. Ehrig, G. Engels & G. Rozenberg (Ed.), *Graph grammars and their application to computer science; Fifth International Workshop, Lecture Notes in Computer Science 1073,* (pp. 521–535). Springer-Verlag, New York Berlin Heidelberg (1996)