Stochastic Process Semantics for Dynamical Grammar Syntax

Eric Mjolsness
Department of Computer Science
University of California, Irvine
emj@uci.edu

UCI ICS TR# 05-14

Version 3 November 2005

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 mapped to semantics given in terms of a ring of operators, so that grammatical composition corresponds to operator addition or multiplication. The operators are generators for the time-evolution of stochastic processes. Within this modeling framework one can express data clustering models, logic programs, ordinary and stochastic differential equations, graph grammars, and stochastic chemical reaction kinetics. This mathematical formulation connects these apparently distant fields to one another and to mathematical methods from quantum field theory and operator algebra.

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 Stochastic Parameterized Grammars (SPG's) and their generalization to Dynamical Grammars (DG's) as such a unifying framework. To this end we define mathematically both the syntax and the semantics of this formal modeling language.

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

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

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. We will map such grammars to stochastic processes in both continuous time (Section 3.2) and discrete time (Section 3.5), and relate the two definitions (Section 3.8). A key feature of the semantics 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) a powerful technique for generating simulation algorithms, and 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 5.3), systems of ordinary differential equations (Section 5.4), and systems of stochastic differential equations (Section 5.4). Other frameworks that describe model classes that may overlap with those described here are numerous and include: branching or birth-and-death processes [1], marked point processes [2], MGS modeling language using topological cell complexes [3], interacting particle systems [4], BLOG probabilistic object model [5], adaptive mesh refinement with rewrite rules [6], stochastic pi-calculus [7], and colored Petri Nets [8]. The composition-preserving mapping $\Psi_{c/d}$ to an operator algebra of stochastic processes, however, appears to be novel.

The present paper is an expanded version of the summary presented in [9].

2 Syntax Definition

Consider the rewrite rule

$$A_1(x_1),\, A_2(x_2),\ \ldots,\, A_n(x_n) \to B_1(y_1),\, B_2(y_2),\ \ldots,\ B_m(y_m)\ \mathbf{with}\ \rho(\{x_i\}, \{y_j\}) \tag{1}$$

where the A_k and B_l denote symbols τ_a chosen from an arbitrary alphabet set $\mathcal{T} = \{\tau_a \mid a \in \mathcal{A}\}$ of "types". In addition these type symbols carry expressions for parameters x_i or y_j chosen from a base language $\mathcal{L}_P(i)$ defined below. The A's can appear in any order, as can the B's. Different A's and B's appearing in the rule can denote the same alphabet symbol τ_a, with equal or unequal parameter values x_i or y_j. 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 $\mathcal{F}$ of real-valued functions. 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. This interpretation will be formalized in the semantics. As an example,

$$\begin{gathered}\text{HydrogenAtom}(x),\ \text{HydrogenAtom}(y) \to \text{HydrogenMolecule}(z)\\ \mathbf{with}\ \ f(\,\|x-y\|)\exp\!\left(-\left(\,\|x-z\|^2 + \|y-z\|^2\right)\big/\,2\,\sigma^2\right)\end{gathered}$$

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

We now define $\mathcal{L}_P(i)$. Each 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 $\mathcal{D} = \{D_b \mid b \in \mathcal{B}\}$. Each D_b is a measure space with measure μ_b. Particular D_b 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_b in $V_a = \prod_{k=1}^{d_a} D_{b=\sigma(a\,k)}$ constitutes the type signature $\{\sigma_{a\,k} \in \mathcal{B} \mid 1 \leqslant k \leqslant d_a\}$ of type τ_a. (As an aside, polymorphic argument type signatures are supported by defining a derived type signature $\{\sigma_{a\,k\,b} = (D_b \subseteq D_{\sigma(a\,k)}) \in \{T, F\} \mid 1 \leqslant k \leqslant d_a,\ b \in \mathcal{B}\}$. For example we can regard $\mathbb{Z}$ as a subset of $\mathbb{R}$.) Correspondingly, parameter expressions x_i are tuples of length d_a, such that each component $x_{i\,k}$ is either a constant in the space $D_{b=\sigma(a\,k)}$, or a variable X_c ($c \in C$) that is restricted to taking values in that same space $D_{b(c)}$. 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 $x_{i\,k}$ take values in the same space $D_{b=\sigma(a\,k)}$. A *substitution* $\theta : c \mapsto D_{b(c)}$ 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 $\mathcal{L}_P(i)$.

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. 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 a on the left hand side followed by nondecreasing order of type index a on the right hand side of the rule. 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 $\rho_r = \rho_r^{\text{pure}}((x_i)) \Pr_r((y_j) \mid (x_i))$ of a conditional distribution on output parameters given input parameters $\Pr_r((y_j) \mid (x_i))$ and a total probability rate $\rho_r^{\text{pure}}((x_i))$ as a function of input parameters only.

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 called a parameterized *term,* which can match to a parameter-bearing *object* or *term instance* in a "pool" of such objects.) The caveat is that a particular type τ_i may appear 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 $\{ \ldots \tau_{a(i)}(x_i) \ldots \}_*$ (in which the same object $\tau_{a(i)}(x_i)$ may appear as the value of several different indices i) for both the LHS and RHS (Left Hand Side and Right Hand Side) of a rule:

$$\{\tau_{a(i)}(x_i) \mid i \in \mathcal{I}_L\}_* \to \{\tau_{a'(j)}(y_j) \mid j \in \mathcal{I}_R\}_* \ \textbf{with}\ \rho_r((x_i), (y_j)) \tag{2}$$

Here the same object $\tau_{a(i)}(x_i)$ may appear as the value of several different indices i under the mappings $i \mapsto (a(i), x_i)$ and/or $i \mapsto (a'(i), y_i)$. Finally we introduce the shorthand notation $\tau_i = \tau_{a(i)}$ and $\tau'_j = \tau_{a'(j)}$, and revert to the standard notation {} for multisets; then we may write

$$\{\tau_i(x_i)\} \to \{\tau'_j(y_j)\} \ \textbf{with}\ \rho_r((x_i), (y_j)) \tag{3}$$

In addition to the **with** clause of a rule following the LHS→RHS header, several other alternative clauses can be used as follows. "**under** $E(x, y)$" is translated into "**with** $\exp(-E(x, y))/Z$" where Z is the Boltzmann distribution partition function corresponding to $E(y)$ holding x constant. "**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 $f(x, y) = 0$. The semantics of "**via** Γ" will be defined in Section 3.3. The translation of "**solving** e" or "**solve** e" will be defined in terms of **with** clauses in Section 5.4. As a matter of definition, Stochastic Parameterized Grammars do not contain **solving/solve** clauses, but Dynamical Grammars may include them. 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 one. There exists a preliminary implementation of an interpreter for most of this syntax in the form of a *Mathematica* notebook, which draws samples according to the semantics of Section 3 below [10].

A Stochastic Parameterized Grammar (SPG) Γ consists of (minimally) a collection of such rules with common type set $\mathcal{T}$, base space set $\mathcal{D}$, type signature specification σ, and probability rate language $\mathcal{L}_R$. After defining the semantics of such grammars, it will be possible to define semantically equivalent classes of SPG's that are untyped or that have richer argument languages $\mathcal{L}_P(i)$.

3 Semantic Maps

We provide a semantics function $\Psi_c(\Gamma)$ in the form of an algebraic construction that results in a *stochastic process*, if it exists, or a special "undefined" element if the stochastic process doesn't exist. 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. In each case the stochastic process specifies the time evolution of a probability distribution over the contents of a "pool" of grounded parameterized terms $\tau_a(x_a)$ that can each be present in the pool with any allowed multiplicity from zero 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.

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 from more elementary linear mappings. To do so we need to define the states.

A state of the "pool of term instances" is defined as an integer-valued function n: the "copy number" $n_a(x_a) \in \{0, 1, 2, \ldots\}$ of parameterized terms $\tau_a(x_a)$ that are grounded (have no variable symbols X_c), for any combination $(a, x_a) \in \mathcal{V} = \coprod_{a \in \mathcal{A}} a \otimes V_a$ of type index $a \in \mathcal{A}$ and parameter value $x_a \in V_a$. We denote this state by the "indexed set" notation for such functions, $\{n_a(x)\}$. Each type τ_a may be assigned a maximum 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 state of the full system at time t is defined as a probability distribution on all possible values of this (already large) pool state: $\Pr(\{n_a(x_a) \mid (a, x_a) \in \mathcal{V}\}; t) \equiv \Pr(\{n_a(x_a)\}; t)$. The probability distribution that puts all probability density on a particular pool state $\{n_a(x_a)\}$ is denoted $|\{n_a(x_a)\}\rangle$.

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

$$\begin{gathered} \frac{d}{dt} \Pr(\{n_a(x)\};\ t) = \sum_{\{m_a(x)\}} H_{\{n\}\{m\}} \Pr(\{m_a(x)\};\ t), \quad \text{i.e. in matrix notation} \\ \frac{d}{dt} \Pr(t) = H \cdot \Pr(t) \end{gathered} \tag{4}$$

which has the formal solution

$$\Pr(t) = \exp(t H) \cdot \Pr(0). \tag{5}$$

For discrete-time semantics $\Psi_d(\Gamma)$ there is an linear map $\hat{H}$ which evolves unnormalized probabilities forward by one rule-firing time step. The probabilities must of course be normalized, so that after s discrete time steps the probability is:

$$\Pr(s) = c_n \hat{H}^s \cdot \Pr(0) = \left(\hat{H}^s \cdot \Pr(0)\right) \big/ \left(\mathbf{1} \cdot \hat{H}^s \cdot \Pr(0)\right) \tag{6}$$

which, taken over all $s \geqslant 0$ and $\Pr(\{n_a(x)\};\ 0)$, defines $\Psi_d(\Gamma)$. In both cases the long-time evolution of the system may converge to a limiting distribution $\Psi_c^*(\Gamma) \cdot \Pr(0) = \lim_{t \to \infty} \Pr(\{n_a(x)\}; t)$ which is a key feature of the semantics, but we do not define the semantics $\Psi_{c/d}(\Gamma)$ as being only this 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.

3.1 Operator algebra

The simplest basis operators $\{B_\alpha\}$ are elementary creation operators $\{\hat{a}_a(x) \mid a \in \mathcal{A} \wedge x \in V_a\}$ and annihilation operators $\{a_a(x) \mid a \in \mathcal{A} \wedge x \in V_a\}$ that increase or decrease each copy number $n_a(x)$ in a particular way (reviewed in [11]):

$$\begin{aligned} \hat{a}_a(x) \mid \{n_b(y)\}\rangle &= \mid \{n_b(y) + \delta_K(a, b)\, \delta(x, y)\}\rangle \\ a_a(x) \mid \{n_b(y)\}\rangle &= n_a(x) \mid \{n_b(y) - \delta_K(a, b)\, \delta(x, y)\}\rangle \end{aligned} \tag{7}$$

where

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

and where δ is the Dirac delta (generalized) function appropriate to the (product) measure μ on the relevant value space V. These two operator types then generate $N_a(x) = \hat{a}_a(x)\, a_a(x)$

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

and they satisfy

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

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 omit the type and value subscripts which are all assumed equal and discrete below, then

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

$$\{a,\ \hat{a}\} \equiv \text{" anticommutator "} \equiv a\, \hat{a} + \hat{a}\, a = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I\,;\ \hat{a}\, a\ = N \equiv \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

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

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

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 $Q(N \mid n^{(\max)})$ of degree $n^{(\max)}$-1 in N with rational coefficients,

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

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

$$[a(x),\ \hat{a}(y)] = \delta(x-y)\left[I + N\, Q\left(N \,\middle|\, n^{(\max)}\right)\right] \tag{8}$$

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

3.2 Continuous-time semantics

For a grammar rule number "r" of the form of (Equation 3) 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:

$$\hat{O}_r = \rho_r\,((x_i),\,(y_j))\left[\prod_{i\in \mathrm{rhs}(r)} \hat{a}_{a(i)}\,(x_i)\right]\left[\prod_{j\in \mathrm{lhs}(r)} a_{b(j)}\,(y_j)\right] \tag{9}$$

If there are variables $\{X_c\}$, we must sum or integrate over all their possible values in $\otimes_c D_{b(c)}$:

$$\hat{O}_r = \int_{D_{b(1)}} \cdots \int_{D_{b(c)}} \cdots \left(\prod_c d\,\mu_{b(c)}(X_c)\right) \rho_r\,((x_i\,(\{X_c\})),\,(y_j\,(\{X_c\})))\left[\prod_{i\in \mathrm{rhs}(r)} \hat{a}_{a(i)}\,(x_i(\{X_c\}))\right]\left[\prod_{j\in \mathrm{lhs}(r)} a_{b(j)}\,(y_j\,(\{X_c\}))\right] \tag{10}$$

Thus, syntactic variable-binding has the semantics of multiple integration.

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

$$\hat{O}_r = \rho_r\,((x_i),\,(y_j))\left[\prod_{i\in \mathrm{rhs}(r)\setminus \mathrm{lhs}(r)} \hat{a}_{a(i)}\,(x_i)\right]\left[\prod_{j\in \mathrm{lhs}(r)} N_{b(j)}\,(y_j)\right] \tag{11}$$

Unfortunately $\hat{O}_r$ doesn'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 conserves probability:

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

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

$$H = \sum_r O_r = \sum_r \hat{O}_r - \sum_r \mathrm{diag}\left(\mathbf{1}^T \cdot \tilde{O}_r\right) = \hat{H} - D \tag{12}$$

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 9) (Equation 10) 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 12 or Equation 10) corresponds to independently firing processes; ring operator multiplication (as in Equation 9) corresponds to obligatory event co-occurrence.

3.3 Recursion among grammars

If the limiting distribution $\Psi_c^*(\Gamma)$ exists for all initial states $\Pr(0)$, it defines a new operator $B^*(\Gamma) = \lim_{t\to\infty} \exp t\, H(\Gamma)$. 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 $P(\{\tau_{a(i)} \mid i \in \mathcal{I}'\})$:

$$O_r = (P(\{\tau_{a'(j)} \mid j \in \mathcal{I}_R\}))\, B^*(\Gamma)\, (P(\{\tau_{a(i)} \mid i \in \mathcal{I}_L\})) \tag{13}$$

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

$$\{\tau_{a(i)}(x_i) \mid i \in \mathcal{I}_L\} \to \{\tau_{a'(j)}(y_j) \mid j \in \mathcal{I}_R\} \text{ \textbf{via} } \Gamma \tag{14}$$

in a different grammar Γ' or even within the same grammar Γ, recursively. This is how one continuous-time grammar can "call" another one. 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 $B(\Gamma \mid t) = \exp t\, H(\Gamma)$ in place of $B^*(\Gamma)$ in Equation 13. In this case the syntax of Equation 14 can be " ... **via** $\Gamma(t)$".

3.4 Execution algorithms

The meaning of the operator exponential is given by the Taylor series expansion for the exponential, or more generally by the Trotter product formula as follows:

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

This formula can be used to derive "forward Euler" types of simulation algorithms. It is an analog of "operator splitting" in numerical integration. More advanced methods such as the Gillespie simulation algorithm (suitably generalized to handle parameterized types using the factorization $\rho_r((x_i), (y_j)) = \rho_r^{\text{pure}}((x_i)) \Pr_r((y_j) \mid (x_i))$) can be derived from the time-ordered product expansion of $\exp[t(H_0 + H_1)]$ (Section 3.7 below).

3.5 Discrete-time SPG semantics

The operator $\hat{H}$ describes the flow of probability per unit time, over an infinitesimal time interval, into new states resulting from a single rule-firing of any type. If we condition the probability distribution on a single rule having fired, setting aside the probability weight for all other possibilities, the normalized distribution is

$$c_1\, \hat{H} \cdot p_0 = \left(\hat{H} \cdot p_0\right) / \left(\mathbf{1} \cdot \hat{H} \cdot p_0\right)$$

For a second rule firing it is therefore

$$\tilde{c}_2\, \hat{H} \cdot \left[\left(\hat{H} \cdot p_0 \right) / \left(\mathbf{1} \cdot \hat{H} \cdot p_0 \right) \right] = \frac{\hat{H} \cdot \left[\left(\hat{H} \cdot p_0 \right) / \left(\mathbf{1} \cdot \hat{H} \cdot p_0 \right) \right]}{\left(\mathbf{1} \cdot \hat{H} \cdot \left[\left(\hat{H} \cdot p_0 \right) / \left(\mathbf{1} \cdot \hat{H} \cdot p_0 \right) \right] \right)} = \frac{\left(\hat{H} \cdot \left(\hat{H} \cdot p_0 \right) \right) \left(\mathbf{1} \cdot \hat{H} \cdot p_0 \right)}{\left(\mathbf{1} \cdot \hat{H} \cdot p_0 \right) \left(\mathbf{1} \cdot \hat{H} \cdot \left(\hat{H} \cdot p_0 \right) \right)}$$

$$= \left(\hat{H}^2 \cdot p_0 \right) \Big/ \left(\mathbf{1} \cdot \hat{H}^2 \cdot p_0 \right) = c_2\, \hat{H}^2 \cdot p_0$$

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

$$c_n\, \hat{H}^n \cdot p_0 = \left(\hat{H}^n \cdot p_0 \right) / \left(\mathbf{1} \cdot \hat{H}^n \cdot p_0 \right) \tag{15}$$

where $\hat{H} = \sum_r \hat{O}_r$ as before. This expression depends on a normalization constant $c_n = 1 \Big/ \left(\mathbf{1} \cdot \hat{H}^n \cdot p_0 \right)$. 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 $c_n \neq c^n$. This is a critical distinction between stochastic grammar and Markov chain models, for which $c_n = c^n$.

An execution algorithm is directly expressed by (Equation 15).

3.6 Relation to fixed points

The Ergodic Theorem gives conditions under which a stochastic processes 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.

3.7 Time-ordered product expansion

An indispensable tool for studying such stochastic processes in physics is the time-ordered product expansion [12-13]. We use the following form:

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

$$= \sum_{n=0}^{\infty} \left[\int_0^t d\, t_1 \int_{t_1}^t d\, t_2 \cdots \int_{t_{n-1}}^t d\, t_n \exp((t - t_n)\, H_0)\, H_1 \exp((t_n - t_{n-1})\, H_0) \cdots H_1 \exp(t_1\, H_0) \right] \cdot p_0 \tag{16}$$

where H_0 is a solvable or easily computable part of H, so the exponentials $\exp(t\, H_0)$ can be computed or sampled more easily than $\exp(t\, H)$. See [14] for an elementary probabilistic derivation of this form. This expression can be used to generate Feynman diagram expansions, in which n denotes the number of interaction vertices in a graph representing a multi-object history [11]. If we apply (Equation 16) with

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

we derive the well-known Gillespie algorithm for simulating chemical reaction networks [15], which can now be applied to SPG's. However many other decompositions of H are possible, one of which is used in Section 5.4 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 16) has the advantage of being recursively self-applicable.

Thus, (Equation 16) entails a systematic approach to the creation of novel simulation algorithms.

3.8 Relation between semantic maps

Proposition. Given the stochastic parameterized grammar (SPG) rule syntax of Equation 22,

(a) There is a semantic function Ψ_c mapping from any continuous-time, context sensitive, 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.

(b) There is a semantic function Ψ_d mapping any discrete-time, sequential-firing, context sensitive, stochastic parameterized grammar Γ via a time evolution operator $\hat{H}(\Gamma)$ 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, n.

(c) The short-time limit of the density $\Psi_c(\Gamma)$ conditioned on $t \to 0$ and conditioned on n is equal to $\Psi_d(\Gamma)$.

(d) 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 $\Gamma_{\text{rl-tree}}$ whose asymptotic probability distribution is that of the resource-limited context-free feature tree $\mathcal{T}(N, q, \phi)$.

Proof: (a): Section 3.2. (b): Section 3.5. (c) Equation 16 (details in Appendix). (d) Section 4.1 below.

Corollary. The following diagram commutes:

$$\begin{tikzcd}
& \hat{H}(\Gamma) \arrow[dl, "\Psi_c"'] \arrow[dr, "\Psi_d"] & \\
\mathcal{P}(H(\Gamma) \mid t, n) \arrow[rr, "(\Delta t \to 0)"] & & \mathcal{P}\left(\hat{H}(\Gamma) \mid n\right)
\end{tikzcd}$$

Here n = number of rule firings, t = continuous time, Δt = elapsed continuous time of execution.

3.9 Discussion: Transformations 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 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) and functors acting on categories. In the case of SPG's, two possibilities for the preserved property are immediately 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: $\Psi^*(\Gamma) = \Psi^*(\Gamma')$. Preserving the full semantics would be required of a simulation algorithm. Alternatively, an inference algorithm could preserve a joint probability distribution on unobserved and observed random variables, in the form of Bayes' rule,

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

where (in, internal, out) are collections of parameterized terms that are inputs to, internal to, and outputs from the grammar Γ respectively.

4 Examples

4.1 Cluster trees

Here is a simple cluster-generating grammar:

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

nodeset(x) $\rightarrow$ node(x), {child(x) $\mid$ $1 \leqslant i \leqslant n$} **with** $q(n)$, $n \geqslant 0$.

child(y) $\rightarrow$ nodeset(x) **with** $\phi(x \mid y)$

}

Since there is only one term on each LHS, it is “context free”. Here is its behavior:

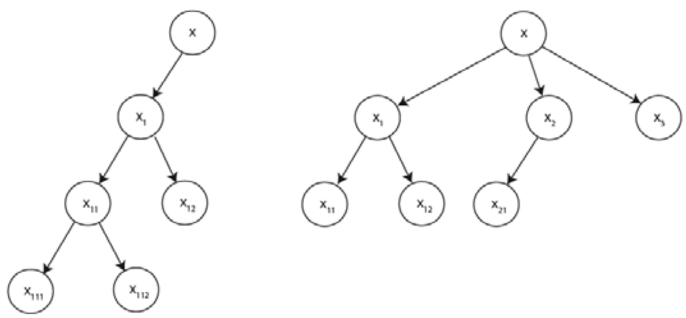

Figure 1: Two feature trees generated by the *clustergen* stochastic parameterized grammar. $\Pr = q(1)\,q(2)^2\,q(0)^3$ $\times\phi\,(x_1 \mid x)\,\phi(x_{11} \mid x_1)\,\phi(x_{12} \mid x_1)$ $\times\phi(x_{111} \mid x_{11})\,\phi(x_{112} \mid x_{11})$. (b) $\Pr = q(3)\,q(2)\,q(1)\,q(0)^4$ $\times\phi\,(x_1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

Here is its discrete-time semantics (omitting for simplicity the node labels x, and just keeping the tree structure):

$$\begin{aligned}\hat{H} &= \sum_{k=0}^{\infty} q(k)\,\hat{a}^k\,a = g(\hat{a})\,a \\ H &= g(\hat{a})\,a - N\end{aligned} \tag{17}$$

$$\begin{aligned}\hat{H}^2 &= g(\hat{a})^2\,a^2 + g(\hat{a})\,g'(\hat{a})\,a; \\ \hat{H}^3 &= g\,(\hat{a})^3\,a^3 + 3\,(g\,(\hat{a}))^2\,g'\,(\hat{a})\,a^2 + g\,(\hat{a})\,(g'\,(\hat{a}))^2\,a + (g\,(\hat{a}))^2\,g''\,(\hat{a})\,a;\ \ldots\end{aligned} \tag{18}$$

where

$$g(z) = \sum_{n=0}^{\infty} z^n\,q(n)$$

In this model, every power of $\hat{H}$, and the continuous-time evolution $\exp t\,H$, can be formally expressed and computed using 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

$$J(x; x_0) = \int_{x_0}^{x} \frac{du}{g(u)-u} \quad \text{and} \quad K(x; x_0) = \int_{x_0}^{x} \frac{du}{g(u)} \tag{19}$$

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

$$\frac{d}{dJ} = \frac{dx}{dJ}\,\frac{d}{dx} = \frac{1}{dJ/dx}\,\frac{d}{dx} \leftarrow\!\shortmid H$$

$$e^{tH}\, f(x) \mapsto e^{t\,(d/dJ)}\, f\big(J^{-1}(J(x))\big) = f\big(J^{-1}(t + J(x))\big) \tag{20}$$

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 18) using K, so that

$$\begin{aligned} e^{s\hat{H}}\, f(x) &\mapsto f\big(K^{-1}(s + K(x))\big) = f\bigg(x + s\,g(x) + \frac{s^2}{2}\,g(x)\,g'(x) + \frac{s^3}{3!}\Big(g(x)\,(g'(x))^2 + (g(x))^2\,g''(x)\Big) + \ldots\bigg) \\ &= f(x) + s\,g(x)\,\partial_x f(x) + \ldots \leftarrow\!\shortmid (I + s\,g(\hat{a})\,a + \ldots)\,f(x), \end{aligned} \tag{21}$$

from which we can recalculate (Equation 18). 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.

The following grammar is *equivalent* to clustergen, in its conditional distributions $\Pr(\{\text{node}(x_I) \mid 1 \leqslant I \leqslant N\},\ \mid N)$. It constitutes a valid *grammar transformation* of clustergen:

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

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

children$(x, n, N) \to$ child(x, N'), children$(x, n-1, N-N')$ **with** $R(N' \mid n, N)$

children$(x, 0, N) \to \varnothing$

child$(y, N) \to$ nodeset(x, N) **with** $\phi(x \mid y)$

}

The functions R and r can be computed by reversion of series using generating functions [14].

Such models have considerable utility for problem formulation in pattern recognition, image analysis, and machine learning.

4.1.1 Dirichlet and Chinese Restaurant processes

The stick-breaking construction of a Dirichlet process can be expressed with this discrete-time grammar (following [16]):

grammar (discrete-time) *DP* (start$(N) \to \{\text{cluster}(i, \theta_k, \pi_k) \mid 1 \leqslant k < \infty\}$) {

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

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

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

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) $\to$ {sample(x) | $1 \leqslant k \leqslant N$}) {

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

samples(N), C = {cluster(i, θ_k, π_k) | $1 \leqslant k < \infty$} $\to$

samples($N-1$), C, sample' (θ_k)

with π_k

subject to $N > 0$

sample' ($\hat{\theta}$) $\to$ 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 [14]. However, *clustergen* determines a more general family of distributions. For example one can control the histogram of cluster sizes.

4.2 Biochemical reaction networks

Given the chemical reaction network syntax

$$\left\{ m_a^{(r)} A_a \,\middle|\, 1 \leqslant a \leqslant A_{\max} \right\} \xrightarrow{k_{(r)}} \left\{ n_b^{(r)} A_b \,\middle|\, 1 \leqslant a \leqslant A_{\max} \right\}, \tag{22}$$

define an index mapping

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

and likewise for $b(j)$ as a function of $\left\{ n_b^{(r)} \right\}$. Then (Equation 22) can be translated to the following equivalent grammar syntax for the multisets of parameterless terms

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

whose semantics is the time-evolution generator

$$\hat{O}_r = k_{(r)} \left[\prod_{i \in \mathrm{rhs}(r)} \hat{a}_{a(i)} \right] \left[\prod_{j \in \mathrm{lhs}(r)} a_{b(j)} \right]. \tag{23}$$

This generator is equivalent to the stochastic process model of mass-action kinetics for the chemical reaction network (Equation 22).

5 Reductions

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

5.1 Logic programs

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

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

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

$$p_1,\ \ldots,\ p_n \to q,\ p_1,\ \ldots,\ p_n \tag{24}$$

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

$$\hat{H} = \sum_r \hat{O}_r = \sum_r \left[\prod_{i \in \mathrm{rhs}(r) \setminus \mathrm{lhs}(r)} \hat{a}_{a(i)}\right] \left[\prod_{j \in \mathrm{lhs}(r)} N_{b(j)}\right] \tag{25}$$

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 24).

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

$$p_1 \wedge \ldots \wedge p_n\ \wedge \neg\, r_1 \wedge \ldots \wedge \neg\, r_m \Rightarrow q\ ,\quad n,\, m \geqslant 0$$

or even more general cardinality constraint atoms $0 \leqslant l \leqslant |\,Z\,| = \sum_{i \in A} \Theta(p_i) \leqslant u \leqslant \infty$ [17]. These constraints can be expressed in operator algebra by expanding the basis operator set $\{B_\alpha\}$ beyond the basic creation and annihilation operators. For example the cardinality of a set Z of positive literals $\{p_i \mid i \in A\}$ is computed by the diagonal operator

$$N_Z = \log_2\left(\bigotimes_i (I_i + \Theta(i \in A)\, N_i)\right)$$

and further thresholding functions can be applied element-by-element to the nonzero diagonal terms of such an operator:

$$\left(l \leqslant \sum_{i \in A} \Theta(p_i) \leqslant u\right) = \ \Theta_{l\,u}\left(\log_2\left(\bigotimes_i (I_i + \Theta(i \in A)\, N_i)\right)\right).$$

Neither $\log_2$ nor $\Theta_{l\,u}$ are exactly within the operator polynomial ring generated by creation and annihilation operators alone, though sufficient approximations may be.

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

5.2 Graph grammars

Graph grammars are composed of local rewrite rules for graphs (see for example [18]). We now express a class of graph grammars in terms of SPG’s.

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 sub-graphs 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 $\{L_\lambda \mid \lambda \in \Lambda\}$. Unlike types but like variables X_c, the label symbols $L_{\lambda(i)}$ actually denote nonnegative integer values - unique addresses or object identifiers.

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

$$\begin{gathered}\left\{L_{\lambda(i)} := \tau_i\left(x_{a(i)}; \left(L_{N(i,\sigma)} \,\middle|\, \sigma \in 1..\sigma^{\max}_{a(i)}\right)\right) \,\middle|\, i \in \mathcal{I}\right\} \\ \to \{L_{\lambda(i)} \mid i \in \mathcal{I}_1 \subseteq \mathcal{I}\} \bigcup \left\{L_{\lambda'(j)} := \tau_j\left(x'_{a'(j)}; \left(L_{N'(j,\sigma)} \,\middle|\, \sigma \in 1..\sigma^{\max}_{a'(j)}\right)\right) \,\middle|\, j \in \mathcal{J}\right\} \\ \textbf{with}\ \rho_r\left(\left\{x'_{a'(j)}\right\} \,\middle|\, \{x_{a(i)}\}\right)\end{gathered} \tag{26}$$

(compare to (Equation 2)). Note that the fanout of the graph is limited by $\sigma^{\mathrm{cur}}_i \leqslant \sigma^{\max}_{a(i)}$. Let

$$\begin{gathered}\mathcal{I} = \mathcal{I}_1 \bigcup \mathcal{I}_2 \ \text{and}\ \mathcal{I}_1 \bigcap \mathcal{I}_2 = \varnothing \\ \mathcal{J} = \mathcal{J}_1 \bigcup \mathcal{J}_2 \ \text{and}\ \mathcal{J}_1 \bigcap \mathcal{J}_2 = \varnothing \\ \mathcal{J}_1 = \{j \in \mathcal{J} \wedge (\exists\, i \in \mathcal{I}_2 \mid \lambda(i) = \lambda'(j)\} \\ \mathcal{J}_2 = \{j \in \mathcal{J} \wedge (\nexists\, i \in \mathcal{I}_2 \mid \lambda(i) = \lambda'(j)\} \\ \mathcal{I}_3 = \{i \in \mathcal{I}_2 \wedge (\nexists\, j \in \mathcal{J}_1 \mid \lambda(i) = \lambda'(j)\} \subseteq \mathcal{I}_2)\end{gathered}$$

This syntax may be translated to the following ordinary non-graph grammar rule (where NextOID is a variable, and OIDGen and Null are types reserved for the translation):

$$\begin{gathered}\{\tau_{a(i)}(L_{\lambda(i)}, x_{a(i)}, (L_{N(i,\sigma)} \mid \sigma \in 1..\sigma^{\mathrm{cur}}_i)) \mid i \in \mathcal{I}\}, \text{OIDGen(NextOID)} \\ \to \{\tau_{a(i)}(L_{\lambda(i)}, x_{a(i)}, (L_{N(i,\sigma)} \mid \sigma \in 1..\sigma^{\mathrm{cur}}_i)) \mid i \in \mathcal{I}_1\} \bigcup \\ \left\{\tau_{a'(j)}\left(L_{\lambda'(j)}, x'_{a'(j)}, \left(L_{N'(j,\sigma)} \,\middle|\, \sigma \in 1..\sigma^{\mathrm{cur}}_j\right)\right) \,\middle|\, j \in \mathcal{J}_1 \wedge (i \in \mathcal{I}_2) \wedge (\lambda(i) = \lambda'(j))\right\} \bigcup \\ \left\{\tau_{a'(j)}\left(L_{\lambda'(j)}, x'_{a'(j)}, \left(L_{N'(j,\sigma)} \,\middle|\, \sigma \in 1..\sigma^{\mathrm{cur}}_j\right)\right) \,\middle|\, j \in \mathcal{J}_2\right\} \bigcup \{\text{Null}(L_{\lambda(i)}) \mid i \in \mathcal{I}_3\} \\ \bigcup \{\text{OIDGen(NextOID} + |\mathcal{J}|)\} \\ \textbf{with}\ \rho_r\left(\left\{x'_{a'(j)}\right\} \,\middle|\, \{x_{a(i)}\}\right) \prod_{j \in \mathcal{J}_2} \delta_K\,(L_{\lambda'(j)},\ \text{NextOID} + j - 1)\end{gathered}$$

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:

$$\begin{gathered}\tau_a(L_{\lambda(1)}, x, (L_{N(1,\sigma)} \mid \sigma \in 1..\sigma^{\mathrm{cur}}_1)),\ \text{Null}(L_{\lambda(2)}) \\ \to \tau_a(L_{\lambda(1)}, x, (L_{N(1,\sigma)} \mid (\sigma \in 1..\sigma^{\mathrm{cur}}_1) \wedge (N(1,\sigma) \neq \lambda(2)))),\ \text{Null}(L_{\lambda(2)}) \\ \textbf{with}\ \ \rho_{\mathrm{cleanup}} \sum_{\sigma \in 1..\sigma^{\max}} \delta_K(L_{N(1,\sigma)},\ L_{\lambda(2)})\end{gathered}$$

5.3 String rewrite rule grammars

Strings may be encoded as one-dimensional graphs using either a singly or doubly linked list data structure. String rewrite rules

$$(\tau_{a(i)}(x_i) \mid i \in \mathcal{I}_L) \to (\tau_{a'(j)}(y_j) \mid j \in \mathcal{I}_R)\ \textbf{with}\ \rho_r((x_i), (y_j)) \tag{27}$$

(note ordering of arguments) are emulated as graph rewrite rules, whose semantics are defined above. This form is capable of handling many L-system grammars [19]. If ρ_r is not supplied it defaults to 1.

5.4 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 $\eta(t) = 0$):

$$d x_i = v_i(\{x_k\})\, d t + \sigma(\{x_k\})\, d W \quad \text{or} \tag{28}$$

$$\frac{d x_i}{d t} = v_i(\{x_k\}) + \eta_i(t) \tag{29}$$

under some conditions on the noise term $\eta(t)$ the dynamics can be expressed [13] as a Fokker-Planck equation for the probability distribution $P(\{x\}, t)$:

$$\frac{\partial P(\{x\}, t)}{\partial t} = -\sum_i \frac{\partial}{\partial x_i} v_i(\{x\})\, P(\{x\}, t) + \sum_{i j} \frac{\partial^2}{\partial x_i\, \partial x_j} D_{i j}(\{x\})\, P(\{x\}, t) \tag{30}$$

Let $P(\{y\}, t \mid \{x\}, 0)$ 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 $t = 0$,

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

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

$$\rho_{\text{drift}}(\{y_i\} \mid \{x_i\}) = -\sum_i \frac{\partial}{\partial y_i} v_i(\{x\}) \prod_i \delta(y_i - x_i) \tag{31}$$

$$\rho_{\text{diffusion}}(\{y_i\} \mid \{x_i\}) = \sum_{i j} \frac{\partial^2}{\partial y_i\, \partial y_j} D_{i j}(\{x\}) \prod_i \delta(y_i - x_i) \tag{32}$$

from which by (Equation 10) we construct the evolution generator operators $O_{\text{FP}} = O_{\text{drift}} + O_{\text{diffusion}}$, where

$$O_{\text{drift}} = -\int d\{x\} \int d\{y\}\, \hat{a}(\{y\})\, a(\{x\}) \left(\sum_i \nabla_{y_i} v_i(\{y\}) \prod_k \delta(y_k - x_k) \right) \tag{33}$$

$$O_{\text{diffusion}} = \int d\{x\} \int d\{y\}\, \hat{a}(\{y\})\, a(\{x\}) \left(\sum_{i j} \nabla_{y_i} \nabla_{y_j} D_{i j}(\{y\}) \prod_k \delta(y_k - x_k) \right) \tag{34}$$

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 $\sum_{i j} \frac{\partial^2}{\partial x_i\, \partial x_j} D_{i j}(\{x\})$ in the Fokker-Planck partial differential equation (Equation 30), 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 $\langle w|\exp(t\,O_{\mathrm{FP}})\,|z\rangle$. For example,

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

Computing higher powers yields

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

which is a formal solution of the drift-only differential equation $(d\,x_i)/d\,t = v_i(\{x_k\})$.

Diffusion/drift rules can be combined with chemical reaction rules to describe reaction-diffusion systems [11,20]. The foregoing approach can be generalized to encompass partial differential equations (PDE's) and stochastic partial differential equations (SPDE's) [14]. 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 28), (Equation 29), (Equation 31), (Equation 32). A special "**solve**" or "**solving**" keyword may be used to introduce such ODE/SDE rule clauses in the SPG syntax. This syntax can 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. 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 16) for $\exp(t\, H)$ 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 [21] which exhibits simulation algorithms, in [22] which introduces the "**solve**" keyword, and in [23] which specifies a dynamical grammar modeling framework for developmental biology.

5.5 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 [14]), and consequently they can support compositional description of complex situations such as multiple object tracking in the presence of cell division in biological imagery [24]. Second, the reduction of many divergent styles of model to a common SPG syntax and operator algebra semantics enables new possibilities for hybrid model forms. For example one could combine logic programming with probability distribution models, or discrete-event stochastic and differential equation models as discussed in Section 5.4, in possibly new ways.

As a third point of AI relevance, from SPG probabilistic domain models it is possible to derive *algorithms* for simulation (as in Section 3.7) 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 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 3.9) 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. 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. Second, a representationally adequate computerized modeling language can be of great assistance in constructing mathematical models in science, as demonstrated for biological regulatory network models by Cellerator [25] 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. 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 ([3,14,19,23,26]). 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 develop 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 [14] generalize many other such representations. For the actual grammar itself, there exists a bipartite graph $\{G_{r\,a}, G_{a\,r}\}$ of types τ_a (indexed by a) and rules (indexed by r), in which type node a links to rule r ($G_{r\,a} = 1$) iff some term of type τ_a occurs in the LHS multiset of rule r, and rule r links to type node a ($G_{a\,r} = 1$) 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 [8] in which CPN *tokens* (our grounded term instances or objects) contain *values* (our vector of parameter values). However our 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, and firing rate functions.

Acknowledgements. Useful discussions with Guy Yosiphon, Pierre Baldi, Ashish Bhan, Michael Duff, Sergei Nikolaev, Bruce Shapiro, Padhraic Smyth, Michael Turmon, and Max Welling are gratefully acknowledged. The work was supported in part by a Biomedical Information Science and Technology Initiative (BISTI) grant (number R33 GM069013) from the National Institute of General Medical Sciences, by the National Science Foundation's Frontiers in Biological Research (FIBR) program, award number EF-0330786, 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.

References

[1] Athreyea, K. B., & Ney, P. E. (1972). *Branching Processes*. Springer-Verlag; Dover.

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

[3] Giavitto, J., & Michel, O. (2001). *MGS: a Programming Language for the Transformations of Topological Collections*. CNRS Université d'Evry Val d'Essonne, Evry. 61-2001, http://mgs.lami.univevry.fr/PUBLICATIONS/publication.html#Documentation .

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

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

[6] Smith, C., Prusinkiewicz, P., & Samavati, F. (2003). *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 3062* (pp. 313–327). Springer-Verlag. http://www.algorithmicbotany.org/papers/localspec.agtive2003.pdf .

[7] Phillips, A., & Cardelli, L., *A Correct Abstract Machine for the Stochastic Pi-Calculus*. BioConcur 2004. Retrieved from http://www.luca.demon.co.uk/Bibliography.htm#GeneNetworks .

[8] Jensen, K. (1997). *Coloured Petri Nets I, II, III*. Springer-Verlag.

[9] Mjolsness, E. (2006). *Stochastic Process Semantics for Dynamical Grammar Syntax: An Overview*. Ninth International Symposium on Artificial Intelligence and Mathematics, Online proceedings, to appear. Also arXiv preprint cs.AI/0511073, 20 Nov 2005, or http://computableplant.ics.uci.edu/papers/#frameworks .

[10] Stochastic Parameterized Grammar interpreter software, Guy Yosiphon and Eric Mjolsness, UCI School of Information and Computer Science, 2005.

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

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

[13] Risken, H. (1984). *The Fokker-Planck Equation*. Berlin: Springer.

[14] Mjolsness, E. (2005). *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 .

[15] Gillespie, D. J. (1976). **Comput. Physics**, 403–434.

[16] Gharamaini, Z., *Non-parametric Bayesian Methods*. Uncertainty in Artificial Intelligence (UAI) Tutorial notes. Retrieved from http://www.gatsby.ucl.ac.uk/~zoubin/talks/uai05tutorial-b.pdf .

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

[18] Cuny, J., Ehrig, H., Engels, G., & Rozenberg, G. (1994). *Graph Grammars and their Applications to Computer Science*. Springer.

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

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

[21] Federl, P., & Prusinkiewicz, P. (2004). *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 3037* (pp. 65–72). Berlin: Springer.

[22] Prusinkiewicz, P., Hammel, M. S., & Mjolsness, E. (1993). *Animation of Plant Development*. SIGGRAPH '93 Conference Proceedings.

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

[24] Gor, V., Bacarian, T., Elowitz, M., & Mjolsness, E. (2005). *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

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

[26] Fracchia, F. D. (1996). *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). Berlin: Springer-Verlag,.

7 Appendix

7.1 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 of discrete rule firings, n, has the generating function

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

so that

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

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:

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

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

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

$$=\sum_{n=0}^{\infty}\frac{z^n}{n!}\left[\partial_z{}^n \exp\left(t\left(\hat{H}\, z-D\right)\right)\right]_{z=0}\cdot p_0$$

$$=\sum_{n=0}^{\infty}\frac{z^n}{n!}\left[\partial_z{}^n \sum_{k=0}^{\infty}\frac{\left(t\left(\hat{H}\, z-D\right)\right)^k}{k!}\right]_{z=0}\cdot p_0$$

$$=\sum_{n=0}^{\infty}\frac{z^n}{n!}\left[\sum_{k=n}^{\infty}\frac{1}{k!}\sum_{\{0\leqslant i_p\leqslant k-n\}\wedge\sum_{p=0}^{n} i_p=k-n} n!\,(-t\,D)^{i_n}\; t\,\hat{H}(-t\,D)^{i_{n-1}}\cdots t\,\hat{H}\,(-t\,D)^{i_0}\right]\cdot p_0$$

$$=\sum_{n=0}^{\infty} z^n\left[\sum_{k=0}^{\infty}\frac{1}{(k+n)!}\sum_{\{0\leqslant i_p\leqslant k\}\wedge\sum_{p=0}^{n} i_p=k} t^n\,(-t\,D)^{i_n}\;\hat{H}(-t\,D)^{i_{n-1}}\cdots\hat{H}\,(-t\,D)^{i_0}\right]\cdot p_0$$

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

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

Thus

$$\lim_{t\to 0}\left[\, s_n\,/\,1\cdot s_n\right]=\hat{H}^n\cdot p_0\Big/\left(1\cdot\hat{H}^n\cdot p_0\right)$$

7.2 Time-ordered operator expansion

We continue the calculation of $S(z)$ from the previous section. The general expansion formula for S is given by the time-ordered product (Equation 2.14 of [11], equation 4.29 of [13]) which we can derive by elementary probabilistic means as follows.

$$=\sum_{n=0}^{\infty} z^n\, t^n\left[\sum_{k=0}^{\infty}\sum_{\{0\leqslant i_p\leqslant k\}\wedge\sum_{p=0}^{n} i_p=k}\frac{\prod_{p=0}^{n}(i_p)!}{\left(\sum_{p=0}^{n} i_p+n\right)!}\,\frac{(-t\,D)^{i_n}}{(i_n)!}\,\hat{H}\,\frac{(-t\,D)^{i_{n-1}}}{(i_{n-1})!}\cdots\hat{H}\,\frac{(-t\,D)^{i_0}}{(i_0)!}\right]\cdot p_0$$

$$=\sum_{n=0}^{\infty} z^n\, t^n\left[\sum_{\{0\leqslant i_p\leqslant\infty\}}\frac{\prod_{p=0}^{n}(i_p)!}{\left(\sum_{p=0}^{n}(i_p+1)-1\right)!}\,\frac{(-t\,D)^{i_n}}{(i_n)!}\,\hat{H}\,\frac{(-t\,D)^{i_{n-1}}}{(i_{n-1})!}\cdots\hat{H}\,\frac{(-t\,D)^{i_0}}{(i_0)!}\right]\cdot p_0$$

$$=\sum_{n=0}^{\infty} z^n\, t^n\left[\sum_{\{0\leqslant i_p\leqslant\infty\}}\frac{\prod_{p=0}^{n}\Gamma(i_p+1)}{\Gamma\left(\sum_{p=0}^{n}(i_p+1)\right)}\,\frac{(-t\,D)^{i_n}}{(i_n)!}\,\hat{H}\,\frac{(-t\,D)^{i_{n-1}}}{(i_{n-1})!}\cdots\hat{H}\,\frac{(-t\,D)^{i_0}}{(i_0)!}\right]\cdot p_0$$

Now we use the Multinomial-Dirichlet normalization integral

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

Accordingly,

$$S(z) =$$

$$\sum_{n=0}^{\infty} z^n \; t^n \left[\sum_{\{0 \le i_p \le \infty\}} \int_0^1 d\theta_0 \cdots \int_0^1 d\theta_n \, \delta\!\left(\sum_{i=1}^{n} \theta_p - 1 \right) \left(\prod_{p=0}^{n} (\theta_p)^{i_p} \right) \frac{(-t\,D)^{i_n}}{(i_n)!} \, \hat{H} \, \frac{(-t\,D)^{i_{n-1}}}{(i_{n-1})!} \cdots \hat{H} \, \frac{(-t\,D)^{i_0}}{(i_0)!} \right] \cdot p_0$$

$$= \sum_{n=0}^{\infty} z^n \; t^n \left[\sum_{\{0 \le i_p \le \infty\}} \int_0^1 d\theta_0 \cdots \int_0^1 d\theta_n \, \delta\!\left(\sum_{i=1}^{n} \theta_p - 1 \right) \frac{(-\theta_n\, t\,D)^{i_n}}{(i_n)!} \, \hat{H} \, \frac{(-\theta_{n-1}\, t\,D)^{i_{n-1}}}{(i_{n-1})!} \cdots \hat{H} \, \frac{(-\theta_0\, t\,D)^{i_0}}{(i_0)!} \right] \cdot p_0$$

$$= \sum_{n=0}^{\infty} z^n \; t^n \left[\int_0^1 d\theta_0 \cdots \int_0^1 d\theta_n \, \delta\!\left(\sum_{i=1}^{n} \theta_p - 1 \right) \right.$$

$$\left. \sum_{\{0 \le i_0 \le \infty\}} \frac{(-\theta_n\, t\,D)^{i_0}}{(i_n)!} \, \hat{H} \sum_{\{0 \le i_1 \le \infty\}} \frac{(-\theta_{n-1}\, t\,D)^{i_1}}{(i_{n-1})!} \cdots \hat{H} \sum_{\{0 \le i_n \le \infty\}} \frac{(-\theta_0\, t\,D)^{i_n}}{(i_0)!} \right] \cdot p_0$$

$$= \sum_{n=0}^{\infty} z^n \; t^n \left[\int_0^1 d\theta_0 \cdots \int_0^1 d\theta_n \, \delta\!\left(\sum_{i=1}^{n} \theta_p - 1 \right) \exp(-\theta_n\, t\,D) \, \hat{H} \exp(-\theta_{n-1}\, t\,D) \cdots \hat{H} \exp(-\theta_0\, t\,D) \right] \cdot p_0$$

$$= \sum_{n=0}^{\infty} z^n \left[\int_0^t d\tau_0 \cdots \int_0^t d\tau_n \, \delta\!\left(\sum_{i=1}^{n} \tau_p - t \right) \exp(-\tau_n\, D) \, \hat{H} \exp(-\tau_{n-1}\, D) \cdots \hat{H} \exp(-\tau_0\, D) \right] \cdot p_0$$

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

$$\exp\!\left(t\left(\hat{H} - D\right)\right)$$

$$= \sum_{n=0}^{\infty} \left[\int_0^t d\tau_0 \cdots \int_0^t d\tau_n \, \delta\!\left(\sum_{i=1}^{n} \tau_p - t \right) \exp(-\tau_n\, D) \, \hat{H} \exp(-\tau_{n-1}\, D) \cdots \hat{H} \exp(-\tau_0\, D) \right] .$$

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

$$\exp\!\left(t\left(\hat{H} - D\right)\right) \cdot p_0 =$$

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

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 $\exp\!\left(t\left(\hat{H} - D\right)\right)$ 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 $\hat{H}$.

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 $\sqrt{-1}$ which would turn our probabilities into the complex-valued probability factors of quantum mechanics. It can be accomplished for any decomposition of H into a solvable part H_0 (here, $-D$) plus a more difficult term H_1 (here, $\hat{H}$):

$$\exp(t\,(H_0 + H_1)) \cdot p_0 \;=$$
$$\sum_{n=0}^{\infty} \left[\int_0^t d\,t_1 \int_{t_1}^t d\,t_2 \cdots \int_{t_{n-1}}^t d\,t_n \exp((t - t_n)\,H_0)\,H_1 \exp((t_n - t_{n-1})\,H_0) \cdots H_1 \exp(t_1\,H_0) \right] \cdot p_0 \tag{35}$$