Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 265 (2010) 123–144
www.elsevier.com/locate/entcs

Towards Measurable Types for Dynamical Process Modeling Languages

Eric Mjolsness[1]

Department of Computer Science
University of California
Irvine, California, USA

Abstract

Process modeling languages such as "Dynamical Grammars" are highly expressive in the *processes* they model using stochastic and deterministic dynamical systems, and can be given formal semantics in terms of an operator algebra. However such process languages may be more limited in the types of *objects* whose dynamics is easily expressible. For many applications in biology, the dynamics of spatial objects in particular (including combinations of discrete and continuous spatial structures) should be formalizable at a high level of abstraction. We suggest that this may be achieved by formalizing such objects within a type system endowed with type constructors suitable for complex dynamical objects. To this end we review and illustrate the operator algebraic formulation of heterogeneous process modeling and semantics, extending it to encompass partial differential equations and intrinsic graph grammar dynamics. We show that in the operator approach to heterogeneous dynamics, types require integration measures. From this starting point, "measurable" object types can be enriched with generalized metrics under which approximation can be defined. The resulting measurable and "metricated" types can be built up systematically by type constructors such as vectors, products, and labelled graphs. We find conditions under which functions and quotients can be added as constructors of measurable and metricated types.

Keywords: biological models, dynamical systems, master equation, measureable type, metricated type, operator algebra, process modeling language, stochastic processes, stochastic semantics

1 Introduction

Modeling complex biological systems is a substantial challenge that requires the integration of ideas from many subfields of science, mathematics and computing. For applications in "computational morphodynamics" [4], the local dynamics of form in biology and elsewhere, the dynamics of spatial objects (including mixtures of discrete and continuous spatial structures) requires integration of models from geometry, physics, biochemistry, dynamical systems, and stochastic processes. An arena where this integration can occur is in formally defined modeling languages that incorporate heterogeneous dynamics: discrete, continuous, deterministic, stochastic, and spatial paradigms for dynamical systems. As shown for example by the

[1] Email: emj@uci.edu

doi:10.1016/j.entcs.2010.08.008

"Dynamical Grammars" (DG) process modeling language, these objectives can be achieved by defining formal semantics in terms of an operator algebra of stochastic processes. DG's comprise a language with defined syntax and semantics [14], first implemented as an embedded language by the "Plenum" package [21] for the *Mathematica* computer algebra system.

In pursuit of biological applications, a natural sequence of generalizations to process models have been encountered. The simplest processes are essentially chemical reactions, in which a few input objects meet and are converted into some other output objects. These reaction proceswes are assembled into network models. Chemical reactions can be modeled deterministically, for example using differential equations that evolve real-valued concentrations, or they can be modeled more accurately as stochastic processes. Essentially the same models are used in elementary population genetics, where the "molecules" are actually organisms in one or more genetically defined species. In both cases, the next escalation involves giving state attributes (such as position or internal state) to the reacting objects. As modeled in DG's, the resulting state attributes can evolve in discrete stochastic jumps or continuously via differential equations. In this way, local information processing within and between neighboring cells can be very flexibly modeled, but the evolution of the cell-cell neighborhood relationships themselves requires further topological and geometric expressiveness.

A useful plateau of biological expressiveness is reached with the encoding of graph grammars within DGs, which enables the simulation of variable-structure discrete spatial models such as stem cell niche models with biomechanics [21]. Further spatial abstraction and expressiveness, such as manifold and non-manifold continuous dynamic geometries encountered in developing biological tissues and organs, will require improved support for labelled graphs and continuous spatial object types, perhaps by introducing powerful type construction mechanisms as proposed below. However, this is not as easy to do in an operator algebraic process modeling language as in a programming language owing to the need for integration of operators over the values of each type, and possibly for distance functions that can quantify the approximation of one value by another, as we will show. So we seek type constructors for spatial modeling which are amenable to the operator algebra formulation of dynamics.

This paper is organized as follows. In the remainder of Section 1 we exhibit a dynamical grammar for a biological model, and define notation. In Section 2 we review the operator algebra approach to defining the semantics of process modeling languages, as it has been applied in the particular case of Dynamical Grammars. In Section 2.2 we review the existing DG mechanisms for generalized reactions or rules, including systems of differential equations, acting on parameterized terms; in each case we exhibit the operator expressions for the modeling language semantics. We also show how type polymorphism may be expressed in DGs. In Section 2.3 we consider extensions to DG semantics for graph grammars and for general (possibly stochastic) partial differential equations, the latter by means of operator functional integrals as an infinite limit. In Section 2.4 we summarize dynamical process se-

mantics and formulate the integration criterion for new type constructors, which requires that types must be measurable.In Section 3 we consider extensible types built by means of primitive types and type constructors. Section 3.2 describes the existing DG type constructors (vectors, products, and labelled graph container objects encoded using product types). As a step forward in Section 3.3 we consider intrinsic labelled graph types, followed by the more difficult cases of functions and quotients. Finally we collect a set of well known observations that together define a sufficient condition for the formation of measurable product types, and likewise formulate a set of properties for object types that in principle allow them to be subjected to quotient operations. These properties include measurability, but also various quantification conditions that are weaker than metrizability. These points are summarized in Section 4.

1.1 Syntax

Briefly, a *dynamical grammar* consists of a header followed by an unordered set or multiset "{...}" of generalized reactions or rules, each representing a process. Each rule has a left hand side (LHS), an arrow, a right hand side (RHS), followed by a keyword (in this paper, keywords are in boldface), followed by additional algebraic syntax depending on the keyword. LHS and RHS consist of multisets of parameterized terms. The header begins with the keyword "grammar", followed by the name of the grammar (so that grammars can invoke one another recursively using rules with appropriate keywords), followed optionally by allowed input and output object multisets in the form of an LHS→RHS rule.

1.1.1 Example: Olfactory epithelium stem cell niche model

A model similar to that of [21] for the regeneration of odorant-sensing neurons in mouse olfactory epithelium can be expressed as a dynamical grammar. Let the parameterized term "cell$[\chi, \mathbf{x}, V, \phi]$" represent a cell with discrete cell type label $\chi \in \{1, 2, 3 = \chi_{\max}\}$, d-dimensional position $\mathbf{x}$, volume v, and growth inhibitor concentration ϕ. Then the grammar could be written as follows:

grammar Epithelium {

/* cell replication with or without differentiation : */

$$\text{cell}[\chi : \mathbb{N}, \mathbf{x} : \mathbb{R}^d, V : \mathbb{R}, \phi : \mathbb{R}] \longrightarrow \text{cell}[\chi + \Delta\chi_1, \mathbf{x} + \Delta\mathbf{x}, V/2, \phi],$$
$$\text{cell}[\chi + \Delta\chi_2, \boldsymbol{x} - \Delta\mathbf{x}, V/2, \phi]$$
$$\textbf{with} \quad \hat{\rho}(V) P(\Delta\chi_1 : \mathbb{N}, \Delta\chi_2 : \mathbb{N} | \chi, \phi) \mathcal{N}(\Delta\mathbf{x} : \mathbb{R}^d; cV^{1/d})$$
$$\times \Theta(\chi < \chi_{\max}) \Theta(\Delta\chi_1 \in \{0, 1\}) \Theta(\Delta\chi_2 \in \{0, 1\})$$

/* cell death : */

$$\text{cell}[\chi_{\max}, \mathbf{x}, V, \phi] \longrightarrow \varnothing \quad \textbf{with} \quad \gamma$$

/* cell growth, dependent on cell type χ : */

$$\begin{aligned}&\text{cell}[\chi, \mathbf{x}, V, \phi] \longrightarrow \text{cell}[\chi, \mathbf{x}, V + dV, \phi] \\ &\textbf{solving} \quad \left\{\tfrac{dV}{dt} = k\Theta(\chi < \chi_{\max}) + k\varsigma(V)\Theta(\chi = \chi_{\max})\right\}\end{aligned}$$

/* symmetric cell-to-cell diffusion of growth inhibition signal ϕ : */

$$\begin{aligned}&\text{cell}[\chi_1, \mathbf{x}_1, V_1, \phi_1], \text{cell}[\chi_{\max}, \mathbf{x}_2, V_2, \phi_2] \\ &\longrightarrow \text{cell}[\chi_1, \boldsymbol{x}_1, V_1, \phi_1 + d\phi_1], \text{cell}[\chi_{\max}, \mathbf{x}_2, V_2, \phi_2] \\ &\textbf{solving} \quad \left\{\tfrac{d\phi_1}{dt} \quad = D(\|\mathbf{x}_1 - \boldsymbol{x}_2\|, (V_1 V_2)^{1/(2d)}) \quad (\phi_2 - \phi_1)\right\}\end{aligned}$$

/* signal production, dependent on cell type χ, and degradation : */

$$\begin{aligned}&\text{cell}[\chi, \mathbf{x}, V, \phi] \longrightarrow \text{cell}[\chi, \mathbf{x}, V, \phi + d\phi] \\ &\textbf{solving} \quad \left\{\tfrac{d\phi}{dt} = k'\Theta(\chi = \chi_{\max}) - \lambda\phi\right\}\end{aligned}$$

}

Here $\hat{\rho}, D$, and ς are nonnegative monotonic bounded functions of their arguments; P and N are conditional probability distributions (in particular $\mathcal{N}(.;\sigma)$ is the normal distribution with mean zero and standard deviation σ); $c, k, k', \gamma,$ and λ are nonnegative real-valued constants; and $\Theta(Q)$ is the 0/1-valued Heaviside step function on a predicate Q. "$x : \tau$" introduces a parameter x with type τ, and comments are set off as /* ... */. The first rule states that cell division cuts cell volume in half, leaves growth inhibitor concentration fixed, jostles the positions slightly while preserving center of mass, and leaves cell type either constant or increased by one step along an irreversible path from "stem cell" to "transit-amplifying cell" to "neuron". This process generates discrete events in continuous time happening **with** a specified probability per unit time. There are also processes that occur over a continuous duration of time, given by the differential equations shown in the **solving** clauses. An additional **solving** rule (not shown) can change the position of cells in response to crowding by their neighbors.

Many of the foregoing rules could be split up into multiple rules. For example the first rule could equivalently be replaced by the more elementary rules:

$$\begin{aligned}&\text{stemcell}[\mathbf{x} : \mathbb{R}^d, V : \mathbb{R}, \phi : \mathbb{R}] \to \text{stemcell}[\boldsymbol{x} + \Delta\mathbf{x}, \frac{V}{2}, \phi], \text{stemcell}[\mathbf{x} - \Delta\mathbf{x}, \frac{V}{2}, \phi] \\ &\textbf{with} \quad \hat{\rho}(V) P_{\text{stemcell}}(0|\phi) \mathcal{N}(\Delta\mathbf{x}; V^{1/d}) \\ &\text{stemcell}[\mathbf{x}, V, \phi] \to \text{TAcell}[\mathbf{x} + \Delta\boldsymbol{x}, V/2, \phi], \text{stemcell}[\mathbf{x} - \Delta\mathbf{x}, V/2, \phi] \\ &\textbf{with} \quad \hat{\rho}(V) P_{\text{stemcell}}(1|\phi) \mathcal{N}(\Delta\mathbf{x}; V^{1/d}) \\ &\text{stemcell}[\mathbf{x}, V, \phi] \to \text{TAcell} :: \text{cell}[\mathbf{x} + \Delta\boldsymbol{x}, V/2, \phi], \text{TAcell}[\mathbf{x} - \Delta\mathbf{x}, V/2, \phi] \\ &\textbf{with} \quad \hat{\rho}(V) P_{\text{stemcell}}(2|\phi) \mathcal{N}(\Delta\mathbf{x}; V^{1/d})\end{aligned}$$

and likewise three more rules for TAcells giving rise to TAcells and/or neurons.

1.2 Notation

Notation is as follows. $f(x, y, ...)$ represents a function evaluated with ordered arguments $x, y, ...$ On the other hand $\tau[x, y, ...]$ with square brackets represents an object of type τ that retains all the information in parameters x, y, and so on. It may be thought of as a syntax tree with root node τ and ordered children x, y, ... Also $\{x, y, ...\}$ is an unordered set, and $\{x, y, ...\}_*$ with an asterisk subscript is an unordered multiset, i.e. a function from the set $\{x, y, ...\}$ to the positive natural numbers specifying "how many times" each element occurs in the set. 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 (tuples or lists in the finite case) using square brackets: $[x(i)||i \in \mathcal{I}]$. More generally, $[x(i)|P(x(i), i)||i \in \mathcal{I}]$ imposes the image of a preexisting ordering of the index set $\mathcal{I}$ onto any elements $x(i)$ selected for inclusion by the predicate P. Similarly, $f([x])$ is a shorthand for $f([x_i||i \in \{1, ...n\}])$ $= f([x_i|1 \leqslant i \leqslant n||i \in \mathbb{N}])$ which is in turn a shorthand for $f(x_1, ...x_n)$. $\Theta(P) \equiv 1$ if P is true, otherwise 0 is the Heaviside function on predicates. $\delta_{ij} \equiv \Theta(i = j)$ is the Kronecker delta function. $\mathcal{N}(.; \sigma)$ is the normal distribution with mean zero and standard deviation σ. In case of type ambiguity (eg. for type inference or dynamic typing) we use the notation $x : \tau$ to indicate that variable x has type τ, and the notation $f([x]) : \tau$ to indicate that f returns a value of type τ. Similarly, $\tau_1 :: \tau_2$ indicates that τ_1 is a subtype of τ_2. Metalanguage syntax is as follows. Macro expansion or evaluation is indicated by the function-like notation: $\mu[\![...]\!]$. Substitution of an expression s for an unbound variable x in a term t is denoted $t\langle\!\langle x \mapsto s \rangle\!\rangle$.

2 Defining Processes

We now describe the semantics of processes. Most of the technical descriptions in this section up through Section 2.2, and also Section 3.2.2, are lightly edited excerpts from [14], which is the primary source for the ideas presented in these sections of the paper.

2.1 Time-evolution operator semantics

The "master equation" for the continuous-time evolution of probabilities is:

$$\frac{dp(t)}{dt} = W \cdot p(t) \tag{1}$$

where $p(t)$ is the joint probability distribution over all possible states of the system at time t. The *semantics* of a model is given by the operator W, which specifies a stochastic process. These stochastic processes can be specialized to deterministic dynamics as well as shown below. The model operator W can be *composed* by simply adding up operators W_r that correspond to parallel, interacting subprocesses

indexed by r:

$$W = \sum_r W_r \tag{2}$$

Simple process operators W_r can be built out of *products* of elementary process operators by which objects of specified type and state are created or destroyed. The semantics of a product of operators is an infinitely rapid sequence of changes taking zero time. Scalar multiplication can speed up or slow down a given process. Thus, we are concerned with at least the algebraic structure of a ring of operators that act linearly on probability vectors. All of this is analogous to the operator algebras encounted in quantum mechanics and quantum field theory, except that only classical probabilities need be used. Quantum semantics would be obtained by inserting a factor of $\sqrt{-1}$ in Equation 1, obtaining the Schrödinger equation.

The semantics of a modeling *language* such as Dynamic Grammars is given by the procedure by which its process expressions can be converted into time evolution operators W_r. The resulting continuous-time dynamics can be related to a discrete-time dynamics in which composition is somewhat more complicated.

Stochastic process operator semantics for modeling languages, including the master equation for time evolution and measures for product types, was proposed in [13] and elaborated in detail in [14]. The master equation was proposed independently for a "small stochastic process algebra" in [3]. Stochastic semantics for the "kappa" rule-based modeling language [5,6] was also proposed in [7].

2.2 Current DG Semantics

We first review those aspects of Dynamical Grammar semantics that have been previously defined and, with the exception of polymorphism, implemented in Plenum.

2.2.1 Chemical reactions

Consider the chemical reaction:

$$\sum_{a=1}^{A_{\max}} m_a^{(r)} A_a \quad \xrightarrow{k_{(r)}} \quad \sum_{b=1}^{A_{\max}} n_b^{(r)} A_b \tag{3}$$

Applications of this kind of process model are legion in biochemistry, population biology, and cellular systems biology.

We can translate this information into a stochastic process obeying the master equation, by defining a suitable operator W algebraically in terms of elementary annihilation and creation operators $a_{a(i)}$ and $\hat{a}_{b(j)}$ obeying the Heisenberg algebra $a_i \hat{a}_j - \hat{a}_j a_i = \delta_{ij}$ or variants thereof. The general principle is : destroy all the objects on the left hand side (LHS) of the rule, and instantaneously thereafter, create all the objects on the RHS. The operator expression of this principle for chemical reactions is:

$$\hat{O}_r = k_{(r)} \left\{ \prod_{j \in \mathrm{rhs}(r)} \hat{a}_{b(j)} \right\} \left\{ \prod_{i \in \mathrm{lhs}(r)} a_{a(i)} \right\} \quad . \tag{4}$$

This operator is off-diagonal and represents the flow of probability into a new state. We must also represent the compensating flow of probability out of the old state, $D_r = \mathrm{diag}(\mathbf{1}^T \cdot \hat{O}_r)$ (where the matrix notation is that $\mathbf{1}^T$ is the transpose of the column vector all of whose elements take the value one, and "diag" converts the resulting row vector into a diagonal matrix), resulting in a net operator $W_r = \hat{O}_r - D_r$. If there are many reactions indexed by r in a reaction network, their operators W_r add up as in Equation 2. The result is a stochastic model of mass action kinetics for each reaction and for the whole reaction network.

An alternative notation for the reaction in Equation 3 could be a multiset rewrite rule: $\{m_a^{(r)} \times A_a | m_a^{(r)} > 0\}_* \xrightarrow{k_{(r)}} \{n_b^{(r)} \times A_b | n_b^{(r)} > 0\}_*$, where $n \times x$ means that object x occurs n times in the multiset.

It is conventional in biochemical network models to reduce higher-order reactions to those with just one or two inputs (unimolecular or bimolecular reactions) by splitting up reactions, so that $\sum_a m_a^{(r)}$ is a small integer. Likewise $\sum_b n_b^{(r)}$ is typically a small integer. Thus, the total amount of computational work that has to be done "instantaneously" in Equation 4 is a small constant. The same principle can be applied in the more expressive parameterized reactions below. The analogous expressions represent interaction vertices for Feynman diagrams in quantum field theory, where they also have low total degree: $\sum_a (m_a^{(r)} + n_a^{(r)})$ is usually 2, 3, or 4.

2.2.2 Parameterized reaction

A minimal generalization of chemical reaction notation is to allow the chemical species or "types" to be indexed by static parameters in a reaction

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

This syntax can be used to formulate dynamical models of stateful objects like cells, molecular complexes, or covalently modified proteins such as those with multiple phosphorylation sites. Here $\{...\}_*$represents a multiset, and **with** is a keyword introducing the probability per unit time ρ_r that the specified discrete event will occur instantaneously at a particular moment in real-valued time. Assuming that the parameter expressions x, y contain no variables X_c, the time-evolution operator for each individual reaction is:

$$\hat{O}_r = \rho_r([x_a], [y_b]) \left\{ \prod_{b \in \mathrm{rhs}(r)} \hat{a}_{j(b)}([y_b]) \right\} \left\{ \prod_{a \in \mathrm{lhs}(r)} a_{i(a)}([x_a]) \right\} \tag{6}$$

Note that there are now separate creation and annihilation operators for every possible value of the parameter list $[x_a]$ - all acting on the same very large Fock space, defined in Section 3.2.2 below.

However, a much more useful rule would be a rule schema with many possible values for some of its parameters, obtained by making the parameter expressions be a function of some variables. If there are variables $\{X_c\}$, we must sum or integrate

over all their possible values in a suitable measure space $\bigotimes_c D_{\beta(c)}$:

$$\hat{O}_r = \int_{D_{j(1)}} \cdots \int_{D_{j(k)}} \cdots \left(\prod_k d\mu_{j(k)}(X_k) \right) \rho_r\Big([x_a([X_k])], [y_b([X_k])] \Big) \times \left\{ \prod_{b \in \mathrm{rhs}(r)} \hat{a}_{j(b)}\left(y_b([X_k])\right) \right\} \left\{ \prod_{a \in \mathrm{lhs}(r)} a_{i(a)}\left(x_a([X_k])\right) \right\} \quad (7)$$

For example, some object types could be parameterized by position and velocity vectors; others by rotation matrices. Different measures would be required to integrate over these different kinds of parameters.

The nonnegative real-valued reaction rate $\rho_r([x_i], [y_j])$ is given by an algebraic expression in a typed language $\mathcal{L}_R$, and denotes a 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. Depending on the norm used in the Banach space $\mathcal{F}$ it may be possible to formulate rate functions that grow without bound as a function of their arguments, and allow in principle for an infinite amount of computation to be done in a finite time. In that case, not all models formulate-able with Dynamical Grammars are effectively simulate-able in finite time on Turing machines.

We assume that there is a typed language $\mathcal{L}_P$ that constrains the parameterized terms x_i occurring in the rules. The essential feature of $\mathcal{L}_P$ is that it includes a set of function symbols with defined input and output type signatures. These function symbols represent type-supported operations on the typed constants, variables X_k, and typed parameter expressions x_a in $\mathcal{L}_P$. The typed parameter expressions x_a may appear in the parameter lists of LHS and RHS terms $\tau_{a(i)}[x_i]$, and as arguments to the rate functions ρ_r. In this way, algebraic data types make contact with algebraic time-evolution operators. Rate functions $\rho_r([x_i], [y_j])$ ultimately relate the domain-specific types to nonnegative real numbers that specify process rates.

Each parameterized term $\tau_{a(i)}[x_i]$ or $\tau_{a'(j)}[y_j]$ 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 type τ_a.

2.2.3 Polymorphism

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

σ_{ab}:

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

Some freedom is present in the choice of σ, with which various different polymorphism schemes could be implemented. Let $l_{ab}(k)$ be the map $l_b(k)$ whose existence is assured by $\sigma_{ab} = 1$ (if it is so). Let $\tilde{\lambda}_{ab} : V_b \to V_a$ be the associated linear projection operator, which drops parameters in b having no counterpart in a, We parameterize the nullspace of $\tilde{\lambda}$ by $z \in V_b{}' = \prod_{m\in\{1,\ldots d_b\}\wedge m\notin \mathrm{Im}(l_b)} D_{\beta=\gamma(bl)}$, and let $\pi_{V_b{}'} : V_b \to V_b{}'$ be the projection operator complementary to $\tilde{\lambda}$. We define $\lambda : V_a \otimes V_b{}' \to V_b$ so that $\lambda \circ (\tilde{\lambda}, \pi_{V_b{}'}) = \mathrm{id}(V_b)$. We will eliminate extraneous values of V_a from consideration with Heaviside step functions $\Theta(x_i([X_c]) \in \mathrm{Im}(\tilde{\lambda}_{ab})) \in \{0, 1\}$.

Then the time-evolution operator expression Equation 7 for the rule of Equation 5 becomes

$$\begin{aligned}\hat{O}_r = \int_{D_{\beta(1)}} \ldots \int_{D_{\beta(c)}} \ldots \left(\prod_c d\mu_{\beta(c)}(X_c)\right) \rho_r\Big([x_i([X_c])], [y_j([X_c])] \Big) \\ \times \left(\prod_{i\in \mathrm{lhs}(r)} \Theta(x_i([X_c]) \in \mathrm{Im}(\tilde{\lambda}_{ab}))\right) \left\{\prod_{j\in \mathrm{rhs}(r)} \hat{a}_{a(j)}(y_j([X_c]))\right\} \\ \times \left\{\prod_{i\in \mathrm{lhs}(r)} \left[\sum_d \sigma_{b(i)d} \int_{V_d{}'} dz\ a_d(\lambda(x_i([X_c]), z))\right]\right\}. \quad (8)\end{aligned}$$

Note that $\sigma_{ab} = 1$ is reflexive and can be chosen to be antisymmetric, hence defining a partial ordering "$\leqslant$" on types. We say τ_b is a *subtype* of τ_a. The type ordering is related to *substitutability*: If $\tau_b \leqslant \tau_a$ then expressions of type τ_b can be substituted for variables of type τ_a in language $\mathcal{L}_P$ without type violation, at least in covariant contexts such as the foregoing rule semantics. In this semantics, subtyping is used only on the LHS and not the RHS of a rule. Subtyping polymorphism is not supported in the Plenum implementation of Dynamical Grammars; instead, subtyping of "cell types" was hand-coded using graph grammar rules (see Section 2.3.1 below) along with extra parameter-bearing objects representing the subtype memberships.

This version of Equation 8 is corrected from that of [14], which included σ but omitted consideration of the map $\tilde{\lambda}$.

2.2.4 Graph grammar rules

In [14], labelled graphs were encoded using parameterized terms by devoting the first parameter of each term to an integer-valued Object Identifier (OID), and using some of the other parameters to hold the OIDs of other graph-linked objects. In this way graph-grammar rules could be systematically translated into parameterized-grammar rules.

Already the OID translation of graph grammar rules may be used to implement conventional Abstract Data Types (ADTs) such as lists in terms of pointer data structures such as doubly linked lists. Using the Heisenberg algebra of creation

and annihilation operators, it is possible to algebraically verify the ADT relation between inserting and removing an item in such a list: insertion followed by deletion should yield the identity operation.

2.2.5 Differential equations (ODE, SDE)

The system of Langevin equations

$$\frac{dx_i}{dt} = v_i([x_k]) + \eta_i(t) \tag{9}$$

are stochastic if the continous stochastic process $\eta_i(t) \neq 0$ given by Stochastic Differential Equations (SDEs); otherwise they specialize to a system of ordinary differential equations (ODEs). It may be recast as a time-evolution operator by using differential operators:

$$\hat{O}_{\text{drift}} = -\int d\{x\} \int d\{y\}\, \hat{a}([y])a([x]) \left(\sum_i \nabla_{y_i} v_i([y]) \prod_k \delta(y_k - x_k) \right) \tag{10}$$

$$\hat{O}_{\text{diffusion}} = \int d\{x\} \int d\{y\}\, \hat{a}([y])a([x]) \left(\sum_{ij} \nabla_{y_i} \nabla_{y_j} D_{ij}([y]) \prod_k \delta(y_k - x_k) \right) \tag{11}$$

Consequently, process reactions or rules that syntactically incorporate ordinary differential or stochastic equations may be given semantics. The ODE version of this possibility has been implemented in Plenum [21] and results in a very flexible kind of hybrid system for biological models.

In the current Plenum implementation of DGs only very specific partial differential equations (PDEs) are supported: diffusion equations with constant, isotropic D.

2.2.6 Algorithms for simulation and learning

Probably the most surprising aspect of the operator formulation of dynamics is that finite, computationally tractable algorithms can be found for sampling from the resulting probability distributions and indeed can be derived systematically. For stochastic discrete event processes, the Dyson series or "time-ordered product expansion" can be used to systematically derive simulation algorithms. In particular if one considers off-diagonal elements as a perturbation, this method can be used to rederive Gillespie's Stochastic Simulation Algorithm [22,14] and its generalization to parameterized terms. Other operator splittings yield other algorithms including hybrid ODE/discrete-event solvers [21]. In addition, parameter inference algorithms can be derived for this power series approach [20,22]. Thus even when all operators used are infinite objects, finite and effective algorithms can be derived from them.

2.3 Proposed DG-like Semantics

2.3.1 Graph grammar rules

In [14], labelled graphs were encoded using parameterized terms using unique integer-valued Object Identifiers (OIDs). However, there was no "intrinsic" graph

type constructor or graph grammar dynamics in the language. We now suggest such a dynamics.

Suppose that with consistent node indexing $i_1, i_2, ...$ we wish to rewire the small graph g as the new small graph g', where a, b range over the same sets of nodes. Represent these graphs by 0/1-valued adjacency matrices, and suppose the node labels are $[\lambda_a|a]$ and $[\lambda'_a|a]$ before and after rewriting. Then the rewriting operator deletes all the old edges and nodes labels, if they exist in exactly the pattern required by g and λ, and replaces them with the corresponding new edges and node labels:

$$\hat{O}_r = \frac{1}{k!} \sum_{\{i_1,...i_k\}} [\prod_{c,d \in \mathrm{rhs}(r)} (\hat{a}_{i_c i_d})^{g'_{cd}}][\prod_{c \in \mathrm{rhs}(r)} \hat{a}_{i_c \lambda'_c}][\prod_{a,b \in \mathrm{lhs}(r)} (a_{i_a i_b})^{g_{ab}}][\prod_{a \in \mathrm{lhs}(r)} a_{i_a \lambda_a}] \tag{12}$$

This semantics automatically generalizes to multigraphs as well, by allowing $g_{ab} \in \mathbb{N}$. The summation implies a search for matching graph structures which is considerably reduced if the input graph g has relatively unique node labels λ.

In this way, graph rewrite rules may be made intrinsic to a graph type or type constructor rather than just being encoded by OIDs. Graph type constructors can be used to build other container types including lists, trees, and so on. One danger with an expression such as Equation 12 is that the product of operators may grow to more than just a handful of operators, so that the amount of computational work that must be done "instantaneously" at each event is bounded by a larger constant. The number of operators multiplied together grows with the size of the graph fragments being rewritten.

2.3.2 Partial differential equations (PDE's) and stochastic PDE's

We may translate partial differential equations and stochastic partial differential equations of general form into the operator algebra, by relating PDE's and SPDE's to large systems of ODE's and SDE's, and taking the limit symbolically. Nontrivial analysis may be needed to confirm whether the indicated limits really exist or not in any given case [9,10].

Consider the following (possibly stochastic) PDE :

$$\frac{\partial \Phi(x)}{\partial t} = F[\Phi](x) = F(\Phi(x), \frac{\partial \Phi(x)}{\partial x}, ..., \frac{\partial^n \Phi(x)}{\partial x^n}) + \eta(t). \tag{13}$$

where x may be a scalar or a vector, and likewise for Φ. We define a translation to (Equation 9 and Equation 10) using Table 1.

With this table of translations, the drift and diffusion operators for PDE's and SPDE's become

$$O_{\mathrm{drift}} = -\int \int \mathcal{D}\Phi \mathcal{D}\Phi' \; \hat{a}(\Phi') a_\tau(\Phi) \left(\int dx \frac{\delta}{\delta \Phi'(x)} F[\Phi'](x) \, \Delta(\Phi' - \Phi) \right) \tag{14}$$

and

$$O_{\mathrm{diffusion}} = D \int \int \mathcal{D}\Phi \mathcal{D}\Phi' \hat{a}(\Phi') a_\tau(\Phi) \left(\int dx \frac{\delta^2}{\delta \Phi'(x)^2} \Delta(\Phi' - \Phi) \right). \tag{15}$$

Table 1
Ordinary vs. Partial differential objects

Ordinary differential object	Partial differential object
d/dt	$\partial/\partial t$
$i \in \mathbb{N}$	$x \in \mathbb{R}$
x_i	$\Phi(x)$
y_i	$\Phi'(x)$
$\partial/\partial x_i$(partial derivative)	$\delta/\delta\Phi(x)$ (functional derivative)
D (homog. scalar diffusion coef.)	D (homog. scalar diffusion coef.)
$\delta(\boldsymbol{y}-\mathbf{x}) = \prod_i \delta(y_i - x_i)$	$\Delta(\Phi'-\Phi) = \prod_x \delta(\Phi'(x) - \Phi(x))$
$\int dx g(x)$ (ordinary integral)	$\int \mathcal{D}\Phi G[\Phi]$ (functional integral)
$a_\tau(\mathbf{x}) = a_\tau([x_i])$	$a_\tau(\Phi) = a_\tau(x \mapsto \Phi(x))$

where

$$\begin{aligned}\Delta(\Phi'-\Phi) &= \lim_{\sigma\to 0} \prod_x \mathcal{N}(\Phi'(x) - \Phi(x); \sigma) \\ &\equiv \lim_{\sigma\to 0} \exp \int dx \log(\mathcal{N}(\Phi'(x) - \Phi(x); \sigma))\end{aligned}$$

This gives another potential application of the time-ordered product expansion which can be used to create simulation algorithms.

With suitable PDE's it becomes possible to represent dynamically changing manifolds, either by differential equations for the metric as in General Relativity, or for an explicit embedding into a higher dimensional space, or for an implicit embedding given by a function $f(\mathbf{x}) = 0$ (a level set method).

2.4 Discussion of dynamics

Table 2 summarizes the increasing DG capabilities called for by various keywords that can appear in the generalized reaction or rule syntax. Recursive process models are available through the "**via**" and "**substituting**" keywords and their semantics, which are analogous to subroutine calls and macro substitutions respectively. An essential point in the semantics is that repeatedly, greater expressivity is achieved by taking *limits* that yield object types and processes of higher (finite or infinite) cardinality. Examples include infinite limits of the maximum number of molecules of each type, the number of values each parameter can have, the precision of a numerical parameter, and the cardinality of allowed index sets for collections of parameters.

Equation 7 and Equation 10 above involve *sums or integrals* over the values taken by a variable of some particular type. Equation 12 can be regarded as a sum over variables whose types are nodes and links in a graph data type. Technically, the values of the integrands are operators in the Fock space defined in Section 3.2.2 below, which means that even integrals over infinite domains integrate up to operators whose nonzero real-valued elements are each the summed reaction rates ρ_r of all fully redundant copies of some rule. This quantity can be restricted to be finite, usually just a process rate times a small integer. Such integrals are typically encountered in the Lagrangian functionals of quantum field theory, where the integration parameters are taken to include particle momenta.

Table 2
DG keywords and the notation they introduce

Importance	Keyword	has expression ...	semantics
Essential			
	with	prob. rate	discrete transitions
	solving	differential eq	differential operator
Expressive			***limits* of essentials**
	subject to	constraint	delta function factor
	via	sub-grammar call	W'=exp(TW)
	solving	functional diff eq	functional diff oper
Convenient			
	substituting	macro gram. call	semantics/expansion
	under	Boltzman energy	related to **with**
	[...; ...]	sequential events	$\exp(tW_2)\exp(tW_1)$

What is essential for each semantic operator above is the capability to integrate over the various required domains of integration, i.e. over the values taken by the typed variables. The reason is that *variable-binding in the process syntax corresponds to integration in the operator semantics.* Integration in turn requires a measure with which to integrate, defined on a measure space. Thus, the operator algebra approach to dynamics *requires measurable data types.*

3 Defining Object Types

3.1 Language

Type-specific subsets of the language $\mathcal{L}_P$ of Section 2.2.2 may have variables, function symbols, relation symbols, and quantifiers as usual, or they may be more constrained for a particular type. Logic may be classical or intuitionistic. Real-valued function expressions are needed to specify process rates. Relations are functions taking values in a truth-value space Ω, which for classical logic is the Boolean algebra on $\{T, F\}$. Constraints on processes can be specified by predicates. The typed language $\mathcal{L}_P$ shares some similarities with for example the typed "local language" $\mathcal{L}$ in [2]. However, a type expression component of the language $\mathcal{L}_P$ is not yet formalized since we don't yet know what function and power type constructors may be measurable.

We will allow axioms expressed in $\mathcal{L}_P$ to be associated with types and type constructors. For example, the axioms for object types arising in "universal algebra" (such as groups and rings but not fields) would be equational laws, universally quantified.

3.2 Existing types

3.2.1 Primitive types

The primitive types for modeling include numbers: minimally, the integers $\mathbb{Z}$ and the real numbers $\mathbb{R}$ (or finitely computable approximations thereof). In each case it is important that there is a standard algebraic structure (a ring or field supporting arithmetic operations), a standard topology, a standard measure, and a standard

distance metric. Specifically the integers have the discrete topology, the uniform measure, and can be integrated over by summation; the real numbers have the topology generated by open intervals, the associated Borel measure, and Lebesgue integration; and both have distances defined by $d(x,y) = |x-y|$ that generalize to Euclidean distance in vector spaces over the respective primitive types. For many applications including quantum mechanical ones, the complex field $\mathbb{C}$ should also be taken as a primitive type. Integration is essential for the operator representation of dynamics, and distance measures enable controlled approximations.

Thus the expressions in the language $\mathcal{L}_P$ involving a primitive type τ include variables of type τ (denoted $x, y, ...,$ or $x : \tau$ etc.), functions such as $+, -, *, /$, and distance $\mathrm{dist}(x,y)$, the equality relation $=$, as well as the integration linear functional $\int$. Given integration it is possible to define distributions or generalized functions including the delta "function" satisfying $\int \delta(x,y) f(x) dx = f(y)$. This is the Kronecker delta function for integers and the Dirac delta function for reals.

3.2.2 Vector and product types

Type constructors generate new types from old ones. Standard type constructors in mathematically defined programming languages may include type sums, products, powers, and function types, and we will also include quotient types. Here we describe the type constructors already present in Stochastic Parameterized Grammars and therefore in Dynamical Grammars: vector and product types.

Vector spaces $\mathbb{R}^d$ or $\mathbb{Z}^d$ of fixed finite integral dimension $d > 0$ over $\mathbb{R}$ or $\mathbb{Z}$ have addition, subtraction, equality, scalar multiplication, distance and integration defined as usual. Distance is defined by the additivity of squared Euclidean distances; integration is defined by multiple integration. In addition, linear transformations on vector spaces may be defined by their action on a vector basis. This fact distinguishes a vector space from a general product type, though it is a specialization or subtype thereof. Other non-primitive types that support $+, -, =, \mathbb{R}$-scalar multiplication, $\int$, and $\mathrm{dist}(.,.)$ can also serve as the substrate for d-dimensional vectors.

In defining Dynamical Grammars and SPGs (Stochastic Parameterized Grammars: DG's without differential equations or differential operators), the notation for instances of product type τ was $\tau[x_1, ..., x_n]$ where the parameters x_i could be values or instances of different types according to a type signature. The type signature for τ is of course the same for all instances. In particular the parameters could be instances of the foregoing primitive types and/or vector types over primitive types. In the case $n = 1$, we have an alternative notation of an object x of type τ. Generically $\tau[x_1, ..., x_n]$ are called parameterized objects.

For product types $\prod_{i \in \mathcal{I} \subseteq \mathbb{N}} \tau_i$, a Fock space was constructed in [14] within which one can represent the probability distribution over numbers of objects of each type, taking into account the indistinguishability of objects of the same type and same parameter values, as follows. Each value space V_a is a measure space, with a σ-algebra of "events" on which probability is to be defined. A probability distribution on a measure space X is just a (nonnegative) measure P on the σ-algebra for which $P(X) = 1$. We may construct a probabilistic version of a many-particle "symmetric

Fock space" following [15]. Given a nonnegative integer n_a we may define the set of states that have a total of n_a "copies" of grounded parameterized term $\tau_a(x_a)$:

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

Here $\mathcal{S}(n)$ is the symmetric group on n items. The quotient is taken with respect to equivalence classes of Cartesian-product members that differ only by a permutation of n_a items. A new σ-algebra is induced on the space $f_a(n_a)$ by the Cartesian product operation and 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.

Products types with parameters of the same product type, or otherwise recursively defined product types, are accommodated in DG's indirectly by way of graph grammars, with graph links represented by equality of unique integer-valued object identifiers (OIDs) defined at various positions in the parameter list as described in Section 2.3.1. This encoding was needed due to the lack of function or power types that could more naturally represent relationships.

3.2.3 Labelled graph types

A *de facto* type constructor is given by the OID encoding of labelled graphs described in Section 2.3.1. Here we exhibit a syntax extension for such labelled graphs.

The OID label or address symbols $\mathrm{Oid}_{\lambda(i)}$ denote OID-typed variables taking unique values in a discrete domain such as the nonnegative integers. 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. Unique OIDs are maintained, so that $\lambda(i)$ and $\lambda'(j)$ are injective maps on nonnegative integers $i \in \mathcal{I}$ and $j \in \mathcal{J}$. A rule in a graph grammar then takes the form

$$\begin{aligned} &\left\{ \mathrm{Oid}_{\lambda(i)} := \tau_i[x_{a(i)}; [\mathrm{Oid}_{N(i,\sigma)} | \sigma \in 1..\sigma^{\max}_{a(i)}]] | i \in \mathcal{I} \right\} \\ &\longrightarrow \left\{ \mathrm{Oid}_{\lambda(i)} | i \in \mathcal{I}_1 \subseteq \mathcal{I} \right\} \cup \left\{ L_{\lambda'(j)} := \tau_j[x'_{a'(j)}; [L_{N'(j,\sigma)} | \sigma \in 1..\sigma^{\max}_{a'(j)}]] | j \in \mathcal{J} \right\} \\ &\qquad \textbf{with} \quad \rho_r\left(\left[x'_{a'(j)}\right], \left[x_{a(i)}\right]\right) \end{aligned} \tag{16}$$

as explained in [14]. Nodes in the LHS and RHS graphs are parameterized types $\tau_i[x_{a(i)}]$. Links in these two graphs are specified by repetition of the same value for an Oid variable occurring on the left of a ":=" symbol and on the right of one or more ":=" symbols, all on the same side (either left or right) of the rule arrow "$\longrightarrow$".

Such rules have been used in models of the regulated growth and cellular differentiation of the filamentous cyanobacterium *Anabaena catenula*, the root of the plant *Arabidopsis thaliana*, and the mouse olfactory epithelium [21]. Translation of such rules to the syntax and semantics of Equation 5 is shown in [14]. It uses both product types such as $\tau_i[\mathrm{Oid}_{\lambda(i)}, x_{a(i)}, [\mathrm{Oid}_{N(i,\sigma)} | \sigma \in 1..\sigma_i^{\mathrm{cur}}]]$ and vector types such as $[\mathrm{Oid}_{N(i,\sigma)} | \sigma \in 1..\sigma_i^{\mathrm{cur}}]$. However, such a translation is not type-safe as the resulting parameterized terms could possibly conflict with others of the same type name and signature, not involved in representing graphs.

Rules of this form allow graphs whose nodes are parameterized terms $\tau_i[x_{a(i)}]$ of one or more types $\tau_i \in \mathcal{T}$ to undergo local rewriting operations, conditioned on their graph connections to other such terms. Thus, we have a *de facto* "graph type constructor" that takes in a set of types $\mathcal{T}$ and produces a new container type. Using polymorphism it may be possible, as in many programming languages, to replace the set of types $\mathcal{T}$ with a single base type τ_{base} for the nodes in the graph. And of course given graphs one can *implement* many other container types, such as trees and arbitrary-length lists, though not in a type-safe manner. So it would be preferable to have graph types supported rather than just encoded in the modeling language, as we discuss in the next section.

3.3 Proposed type constructors

Function types $\tau_1 \to \tau_2$ are needed for dynamics of geometry among other application areas, and quotient types $\tau/\sim$ are needed for mathematical abstraction. There are a number of obstacles to creating function and quotient types with the properties needed for the operator algebra formulation of dynamics. In this section we review a few well-known concpts that collectively may indicate a way forward for function and quotient types.

We will begin, however, with labelled graph types.

3.3.1 Labelled graph types

Given a set of types $\mathcal{T}$, for example a base type τ_{base} and all of its subtypes, we would like to create a graph (or tree or list) type whose nodes are labelled by objects whose types are in $\mathcal{T}$. The semantics of Section 2.3.1 shows the kind of transformations required; what is still needed is suitable syntax for specifying labelled graphs on the LHS and RHS "directly" and in a permutation-invariant way, rather than through a redundant encoding. Similar to an abstract data type, such labeled graphs would not be "built" out of sets or pointers but rather manipulated algebraically. Beginning with the primitive "$\alpha \xrightarrow{\lambda} \beta$" for a pair of nodes labelled by α and β connected by a link labelled by λ, we can use a "Merge" operation (a macro taking any number of arguments) which equates nodes that share labels to

build up small graphs directly, so that for example

$$\text{Merge}[\![\alpha \xrightarrow{\lambda} \beta, \beta \xrightarrow{\mu} \gamma, \gamma \xrightarrow{\nu} \alpha]\!] \;=\; \alpha \xrightarrow{\lambda} \beta \;. \qquad (17)$$

Here the labels $\alpha, \beta, \lambda, \mu, ...$ represent constants or variables, taking typed values for which equality can be tested (for example they may represent integers). Merge$[\![...]\!]$ is a macro, which is evaluated before the DG model is simulated, mapped to its semantics, or analysed. Other basic graph-producing macros may be defined as well. Labels can be further controlled with a relabelling macro operation "$G\langle\!\langle nodelabelmap; linklabelmap \rangle\!\rangle$" where G is a labelled graph and the label maps specify substitutions acting on label expressions in $\mathcal{L}_P$. For example, *nodelabelmap* for integer-valued labels might take the form $i \mapsto f[\![i]\!]$. Such maps can be used to erase label distinctions among nodes and edges, potentially increasing the automorphism group of a labelled graph. For very small graphs, 2D layouts such as the RHS of Equation 17 can be written directly. With such a language we obtain symbolic expressions representing labelled graphs, in such a way that the expressions can occur in the LHS or RHS of a rewrite rule. In the absence of link labels the semantics of such a rule can be given by Equation 12. In the presence of link labels, a similar operator expression can be given or else the node- and link-labelled graphs can first be translated into purely node-labelled bipartite graphs.

As an example of such a graph rewrite rule, one might have a pair of cells sharing a common "face" which is to be divided into two different subfaces:

$$\left(1 \to 3 \leftarrow 2\right) \langle\!\langle [\text{cell}(c_1), \text{cell}(c_2), \text{face}(\phi_0)]\,; \partial \rangle\!\rangle$$

$$\longrightarrow \left(\begin{array}{ccc} & 4 & \\ \nearrow & & \nwarrow \\ 1 & & 2 \\ \searrow & & \swarrow \\ & 5 & \end{array} \right) \langle\!\langle [\text{cell}\,(c_1)\,, \text{cell}\,(c_2)\,, \varnothing,\ \text{face}\,(\phi_1)\,,\ \text{face}\,(\phi_2)]\,; \partial \rangle\!\rangle$$

A Dynamical Grammar that also uses the graph rewriting capabilities called for here and in Section 2.3.1 may be termed a "Dynamical Graph Grammar". Examples of related frameworks in which one of the graph link types is used to formalize containment relations between biological structures include P-systems [18] and stochastic bigraphs [11]. In addition Finite Element Method (FEM) geometries and other discretized manifold or nonmanifold geometries (eg. cell complexes) can be represented with suitable labelled graphs. However, to represent continuous limits of such geometries also requires function types, discussed in the next section. For reflective or meta-modeling, labeled graphs can be used to represent commutative diagram specifications of axioms, and also graphical models of probability distributions. Simple meta-rules were demonstrated in [21]. Like any discrete space,

discretely labelled graphs can be given a counting measure.

3.3.2 Function types

Product, function, and power types can all substantially *raise* the cardinality of the (finite or infinite) objects they represent or approximate, and can therefore be computationally problematic. Integrating a functional over a domain consisting of real-valued functions introduces new complications not present in finite-dimensional multiple integration, since Lebesgue measure is not available in infinte-dimensional vector spaces. Instead one may use the Wiener measure which is defined in terms of a diffusion stochastic process, or more generally the "abstract Wiener space" measure on any separable Banach space [8,17], if a suitable norm has been defined. There are several plausible topologies for function spaces, including strong (derived from the norm) and weak. Linear operators, required for meta-modeling and a natural next step in the type progression, have even more topologies that generalize the topology of finite matrices: norm, weak, strong, ultraweak, ultrastrong, and so on. So ideas of topology, measure, and integration each split up into several different generalizations in infinite dimensional settings.

Thus each function type requires in principle a verification that a measure suitable for integration can be defined. If the domain of the function is $\mathbb{Z}$ or $\mathbb{Z}^d$ then the uniform measure and the discrete topology can be used. If the domain is $\mathbb{R}$ or $\mathbb{R}^d$ then we can use the Wiener measure which is natural when spatial locality is reflected in processes that permit local diffusion. More generally if a suitable norm, separable Banach space, and embedded separable Hilbert space can be defined, then we can use the abstract Wiener measure. In this way, definitions of norm and inner product (or distance) can serve as essential stepping stones towards defining measure, in the case of types in infinite dimensional function spaces.

Since a separable topological space is one that contains a countable dense subset, and since the product of at most c separable spaces is separable (where $c = |\mathbb{R}|$ is the cardinality of the continuum) [16], the function types admitted above cannot be iterated indefinitely without further constraint to restore separability.

3.3.3 Quotient types

Quotient types, formed by taking equivalence classes modulo an equivalence relation, have the potential to *lower* cardinality, mitigating the problems introduced by function types. They are essential to defining abstractions. But quotients may introduce computational problems when equivalence is hard to determine. Thus, the relation of equality ($=$) between instances of a type may be augmented by a separate, coarser internal equivalence relation ($\approx$) to keep the equality of abstract types computationally tractable by representing separately the accumulated equivalences that can't easily be computed. A motivating example of a quotient space is the space of differential manifolds, which is usually defined in terms of atlases of coordinate charts (with transition functions), modulo a "compatibility" equivalence relation between atlases.

Coarser and finer equivalence relations may be defined using distance metrics

and related concepts. A real-valued distance metric obeys the axioms of (A1) non-negativity $d(x,y) \geqslant 0$, (A2) identity $d(x,y) = 0 \Leftrightarrow x = y$, (A3) symmetry $d(x,y) = d(y,x)$, and (A4) the triangle inequality $d(x,z) \leq d(x,y) + d(y,z)$. Implicitly there is also (A5) finiteness $d(x,y) < +\infty$. These axioms can be relaxed in various useful ways. Dropping (A3) results in a "quasimetric", which can be symmetrized to give a metric by either + or maximum operations. Relaxing (A2) to $x = y \Rightarrow d(x,x) = 0$ gives a "pseudometric". Dropping (A5) results in an "extended" metric. Several of these omissions may be combined.

The extended pseudoquasimetric (or extended quasipseudometric) was advocated in [12] for its categorical properies under the quotient operation. One key point is that the Hausdorff metric is a symmetrized version of a pseudometric between sets in a metric or pseudometric space; such sets can be taken to be the equivalence classes under an equivalence relation, $\sim$. In this way, the pseudometric property can be preserved when one takes the quotient by an equivalence relation. The same argument applies in the more general case of a quasi-pseudometric. Unfortunately pure metrics, satisfying axiom (A2) along with the others, are not generally preserved under quotients. Also metric spaces are first-countable (have a countable local base topology), in contrast to unrestricted function spaces such as $\mathbb{R} \to \mathbb{R}$. Consequently, we won't insist on a tight relationship between distance metrics and topology.

To show that the triangle inequality persists for asymmetric Hausdorff distances between equivalence classes $[\![x]\!]$, $[\![y]\!]$ etc of variables x, y, etc. is standard: $D([\![x]\!],[\![z]\!]) \equiv \sup_{x\in[\![x]\!]}\inf_{z\in[\![z]\!]} d(x,z) \leqslant \sup_{x\in[\![x]\!]}\inf_{y\in[\![y]\!]}\inf_{z\in[\![z]\!]}(d(x,y) + d(y,z)) \leqslant \sup_{x\in[\![x]\!]}\inf_{y\in[\![y]\!]} d(x,y) + \inf_{y\in[\![y]\!]}\inf_{z\in[\![z]\!]} d(y,z) \leqslant \sup_{x\in[\![x]\!]}\inf_{y\in[\![y]\!]} d(x,y) + \sup_{y\in[\![y]\!]}\inf_{z\in[\![z]\!]} d(y,z) \equiv D([\![x]\!],[\![y]\!]) + D([\![y]\!],[\![z]\!])$.

3.3.4 Properties of metricated types

In addition, distance metrics and the foregoing relaxations of them can serve to define topologies and measures, as we have seen, and can directly serve our criterion of model approximation. So we may propose the following set of properties as essential to objects types τ simulatable with dynamical grammars:

- (P1) a set of functions and relations within the language $\mathcal{L}_P$ whose type signatures include type τ, among them equality (=) defined on objects of the type τ;
- (P2) a list of immediate supertypes $\{\tau'\}$ - i.e. information to determine τ's position in the type ordering $\tau \leqslant \tau'$;
- (P3) a measure μ_τ, and an associated extremal distribution $\delta_\tau(x,y)$;
- (P4) an equivalence relation, $\approx_\tau$;
- (P5) functions $d_{+,\tau}(x,y)$ and $d_{-,\tau}(x,y)$ such that:
  - (P5a) $d_{+,\tau}(x,y) \geqslant d_{-,\tau}(x,y)$;
  - (P5b) $d_{\pm,\tau}(x,y)$ are extended pseudoquasimetrics, i.e. they satisfy axioms (A1), (A4), and $(x =_\tau y) \Rightarrow (d_{\pm,\tau}(x,y) = 0)$;
  - (P5c) the upper function $d_{+,\tau}(x,y)$satisfies $(d_{+,\tau}(x,y) = 0) \Rightarrow (x \approx_\tau y)$;

(P5d) the lower function $d_{-,\tau}(x,y)$satisfies $(x \approx_\tau y) \Rightarrow (d_{-,\tau}(x,y) = 0)$;

(P6) a set of type-specific axioms in $\mathcal{L}_P$.

Such a type τ will be called *metricated*, to distinguish from the stronger claim of "metrizable" (with a Borel measure) and the weaker claim of "measurable".

Note that if $d_+(x,y) = d_-(x,y)$ for metricated type τ then $d(x,y) = 0 \Leftrightarrow x \approx y$, which is a version of axiom (A2) with equality replaced by $\approx$. Also, (A1) and (A4) and $(x =_\tau y) \Rightarrow (d_{\pm,\tau}(x,y) = 0)$ imply that $(d_{+,\tau}(x,y) = 0) \wedge (d_{+,\tau}(y,x) = 0)$ and $(d_{-,\tau}(x,y) = 0) \wedge (d_{-,\tau}(y,x) = 0)$ are equivalence relations. It is easy to supply trivial lower and upper bound candidate distance-like functions as a default, in case no serious use is to be made of them, for example in defining the required measure.

Given a new equivalence relation $\sim$ defined on a metricated type τ , the quotient type $\tau/\sim$ is defined as follows. *Measure* $\mu_{\tau/\sim}$: the pushforward measure, as for example when Lebesgue measure on $\mathbb{R}$ maps to (Lebesgue) measure on the unit circle S^1. *Equivalence relation* $\approx_{\tau/\sim}$: closure of $(\sim, \approx_\tau)$, so that $(x \sim y) \Rightarrow (x \approx_{\tau/\sim} y)$ and $(x \approx_\tau y) \Rightarrow (x \approx_{\tau/\sim} y)$. If $\sim$ is coarser than $\approx_\tau$, then this closure $\approx_{\tau/}$ is just $\sim$. Upper and lower *extended pseudoquasimetrics*: can be taken as the nonsymmetric version of Hausdorff distance, possibly loosened for tractability: $d_{\pm,\tau/}([\![x]\!],[\![y]\!]) \gtrless \sup_{x\in[\![x]\!]}\inf_{y\in[\![y]\!]} d_{\pm,\tau}(x,y)$, for equivalence classes $[\![x]\!]$ and $[\![y]\!]$ under $\sim$, subject to axioms (P5). *Equality* $=_{\tau/\sim}$: must be an equivalence relation on τ, satisfying $(x =_\tau y) \Rightarrow (x =_{\tau/\sim} y) \Rightarrow (x \approx_{\tau/\sim} y)$. For example, we may take $(x =_{\tau/\sim} y) \equiv ((d_{+,\tau}(x,y) = 0) \wedge (d_{+,\tau}(y,x) = 0))$ since the latter is an equivalence relation and since $(x =_\tau y) \Rightarrow (d_{+,\tau}(x,y) = 0) \Rightarrow (x \approx_\tau y) \Rightarrow (x \approx_{\tau/\sim} y)$.

This approach to quotient types is in the spirit of "setoids" [1], though with ideas of "generalized metric space" (explored much further in e.g. [19]) added. The idea is that computing equivalence or "actual" distance may be intractable, but computing some upper and lower bounds on distance can be made tractable. It may also be that verifying proofs of equivalence in particular cases is much more tractable than deciding equivalence.

Thus we see that an object type must support measure and integration, and object type constructors can do this through the use of several kinds of norms and distances including extended pseudoquasimetrics. The generic object data type may take values in some measurable, quasimetric space. Conditions sufficient for the construction of function types and quotient types are given above.

4 Conclusions

Process modeling languages with operator algebra semantics can be augmented with type constructors to create objects and processes at successively larger scales as well as greater levels of abstraction. Vector, product, and graph type constructions are straightforwardly available, and recursively related processes can be defined as well. However, function types and quotient types are more subtle. We give conditions under which they can be defined, but we do not know how often these constructions can be iterated before the conditions are necessarily violated.

A dynamical model may now be defined by a "workspace" containing a combi-

nation of dynamical grammar specifications, which define processes, and nontrivial object type declarations. These grammars and user-defined types are syntactic objects upon which a semantics is defined. They can be regarded as constant values (not varying over time) denoted by their names, in which case there is a clear separation between syntactic expressions and dynamical objects. But there could also be time-varying dynamical grammars and/or types, whose discrete-time or continuous-time dynamics is given by suitable metagrammars.

Acknowledgement

Discussions with Christophe Godin, Przemek Prusinkiewicz, and Guy Yosiphon were helpful in this work. Research was supported by NIH R01 GM086883 and P50 GM76516.

References

[1] G. Barthe, V. Capretta, and O. Pons. *Setoids in type theory.* Journal of Functional Programming, 13(2):261–293, March 2003.

[2] J. S. Bell, *Toposes and local set theories*, Chapter 3, Oxford U. Press 1988. Dover reprint 2008.

[3] L. Cardelli. *A process algebra master equation.* Proc. Fourth International Conference on the Quantitative Evaluation of Systems, QEST 2007.

[4] V. Chickarmane, A. H. K. Roeder, P. T. Tarr, A. Cunha, C. Tobin, E. M. Meyerowitz. *Computational Morphodynamics: A modeling framework to understand plant growth*, Annu. Rev. Plant Biol. 61:65-67, 2010.

[5] V. Danos, J. Feret, W. Fontana, and J. Krivine. *Scalable simulation of cellular signaling networks.* In Zhong Shao, editor, APLAS, volume 4807 of Lecture Notes in Computer Science, pages 139{157. Springer, 2007.

[6] V. Danos, C. Laneve. *Formal molecular biology.* Theoretical Computer Science 325(1), 69–110 (2004).

[7] J. Feret, H. Koeppl, T. Petrov. *Stochastic fragments: A framework for the exact reduction of the stochastic semantics of rule-based models.* 2009. URL http://infoscience.epfl.ch/record/142570?ln=fr, last accessed 6/2010.

[8] Gross, Leonard (1967). *Abstract Wiener spaces.* In *Proc. Fifth Berkeley Sympos. Math. Statist. and Probability* (Berkeley, Calif., 1965/66), Vol. II: Contributions to Probability Theory, Part 1. Berkeley, Calif.: Univ. California Press. pp. 31–42. MR0212152

[9] G. Johnson and M. Lapidus, *The Feynman integral and Feynman's operational calculus.* Oxford U. Press 2000.

[10] H. Kleinert, *Path integrals in quantum mechanics, statistics, polymer physics, and financial markets.* World Scientific 2009.

[11] J. Krivine, R. Milner, and A. Troina, 2008. *Stochastic Bigraphs.* Electron. Notes Theor. Comput. Sci. (ENTCS) 218 (Oct. 2008), 73-96. DOI= http://dx.doi.org/10.1016/j.entcs.2008.10.006.

[12] F. W. Lawvere. (1973, 2002), *Metric spaces, generalised logic, and closed categories*, In Reprints in Theory and Applications of Categories, 1, pp. 1–37, 2002.

[13] E. Mjolsness. *Stochastic process semantics for dynamical grammar syntax: an overview.* In: Ninth International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida, 4-6 January 2006 (http://anytime.cs.umass.edu/aimath06/ last accessed 6/2010); also arXiv:cs.AI/0511073v1, 20 Nov 2005.

[14] E. Mjolsness, G. Yosiphon. *Stochastic process semantics for dynamical grammars.* Annals of Mathematics and Artificial Intelligence 2006, 47:329-395.

[15] M. Reed and B. Simon. *Methods of modern mathematical physics: Functional analysis I.* New York: Academic Press, 1972.

[16] K. A. Ross and A. H. Stone, *Products of Separable Spaces*, The American Mathematical Monthly, Vol. 71, No. 4 (Apr., 1964), pp. 398-403

[17] D. Stroock (2008). *Abstract Wiener Space, Revisited*. Communications on Stochastic Analysis, Vol. 2, No. 1 (2008) 145-151.

[18] A. Spicher, O. Michel, M. Cieslak, J-L. Giavitto, and P. Prusinkiewicz (2008) *Stochastic P systems and the simulation of biochemical processes with dynamic compartments*. BioSystems 91: 458-472.

[19] S. Vickers, *Localic Completion of Generalized Metric Spaces I*, Theory and Applications of Categories, Vol. 14, No. 15, pp 328-356, 2005.

[20] Y. Wang, S. Christley, E. Mjolsness and X. Xie. *Parameter inference for discretely observed stochastic kinetic models using stochastic gradient descent*, submitted manuscript.

[21] G. Yosiphon. *Stochastic parameterized grammars: Formalization, inference and modeling applications*. PhD thesis, Department of Computer Science, University of California, Irvine, May 2009. URL: http://computableplant.ics.uci.edu/~guy/downloads/DGPublications.html, last accessed June 2010.

[22] G. Yosiphon, E. Mjolsness. *Towards the inference of stochastic biochemical network and parameterized grammar models*. In *Learning and Inference in Computational Systems Biology*. Edited by Lawrence ND, Girolami M, Rattray M, Sanguinetti G: MIT Press; 2009.