"Cambium" Intermediate Process Language

Eric Mjolsness[1] and Bruce Shapiro[2]

(1) University of California, Irvine; emj@uci.edu
(2) Caltech Jet Propulsion Laboratory; bshapiro@caltech.edu

December 20, 2008

UCI ICS TR# 08-13

Abstract

Cambium is a mathematically well-defined process language intended as a multiparadigm intermediate representation for biological models that include developmental or other structure-alterning biological processes. The name stands for "Computerized Arrows for Mathematical Biology with Intercellular, Understandable Models". Cambium could find practical application in large-scale software environments that must be compatible with multiple modeling languages and simulators for complex and variable-structure dynamical systems. An example is the modeling of developmental biology at multiple spatial and temporal scales within the iPlant project.

1 Introduction

We propose a generic "arrow" syntax for representing processes, which intrincially has well-defined mathematical semantics, and to which and from which other process languages can be mapped. The target languages are those capable of simulating biological development at multiple scales, including cell division and tissue growth, and which therefore simulate variable-structure dynamical systems. These languages must also support the fine-scale modeling of cellular processes. Arrows in this syntax are given a compositional semantics.

1.1 Related work

The concept of an intermediate, shared modeling language for use in biological modeling has been realized in SBML and CellML. However, these model exchange languages are built on computational rather than mathematical foundations, and they do not currently handle the range of variable-structure systems encountered in developmental biology and "computational morphodynamics". SBML Level 2 doesn't support even fixed-topology spatially replicated models, except by replicating the model file. The importance of dynamical geometry and topology, and heterogeneous (multiparadigm), multiscale dynamics, means that mathematical methods are becoming increasingly important in computational biology and that a strong mathematical foundation is essential to developmental modeling and other spatial biological modeling. The relationship of SBML and CellML to Cambium should in the future be one of knowledge-guided intertranslation, since biological processes can be mapped into mathematical dynamics in a many-to-many manner. Such a biology/dynamics mapping is made explicit for example in the Sigmoid modeling environment [Cheng et al 2006]. To facilitate such intertranslation, Cambium should in the future have arrow types (a generic facility described in Section 2) corresponding to SBML (L2, and multiple specialized L3 extensions) and CellML constructs.

1.2 Notation

An (unordered) multiset is denoted $\{i * e,\ ...\}_*$ where i is an optional integer defaulting to one. Equivalently: Multiset$[i * e,\ ...]$. An (ordered) list, tuple, or sequence can be denoted by $\langle e,\ ...\rangle$ or $[e_j \mid j \in \mathcal{I}]$. Syntax trees denoting function evaluations $f(x,\ ...)$ or other name-list associations are represented as eg. $f[x,\ ...]$. Syntactic substitutions are delimited by $[\![...]\!]$.

2 Generic Arrow Notation

We propose a generic process syntax into which others can be mapped. The common form is

Arrow[type[attributes], LHS, MS, RHS, {Clauses[parameters]}]

or

Arrow[$\tau[\boldsymbol{\alpha}]$, L, M, R, {Clauses[$\boldsymbol{p}$]}]

The arrow type τ and its attributes determine which semantics is to be used. The left hand side input L, the continuously present part M, and the resultant right hand side R are in general labelled graphs with labelled edges; more specifically they can be (and often are) multisets or lists (tuples) of expressions in some sublanguage $\mathcal{L}$. The allowed Clauses and parameters can depend on $\tau[\boldsymbol{\alpha}]$.

Diagrammatic presentations of such a generalized arrow include the following:

$$\tau[\boldsymbol{\alpha}]: \quad L \xrightarrow[\text{rates } \boldsymbol{r}]{M} R, \quad \{\text{Clauses}[\boldsymbol{p}]\}$$

or

$$L \xrightarrow[\tau[\alpha]]{M;\ \boldsymbol{r}} R\ , \quad \{\text{Clauses}[\boldsymbol{p}]\}$$

A further convenient syntactic shorthand is to group arrows that shared type and attributes $\tau[\alpha]$, by placing multiple arrows within brackets preceded by the type: $\tau[\alpha]:\{\text{arrow}_1, \ldots \text{arrow}_k\}$. This is short for $\text{Arrow}[\tau[\alpha], *, \varnothing, *, \{\text{Constituents}[\text{Arrow}_1[\varnothing, , \ldots], \ldots \text{Arrow}_k[\varnothing, , \ldots]]\}]$ which can be transformed syntactically to $\text{Arrow}_1[\tau[\alpha], , \ldots], \ldots \text{Arrow}_k[\tau[\alpha], , \ldots]$.

As in Dynamic Grammars, the LHS, MS, and RHS are topological containers of some kind (with various options listed below), whose elementary constituents are parameter-bearing objects with an object type name (not to be confused with the semantics-determining arrow type name discussed above) and an optional sequence of parameters.

2.1 Definition of arrow types

New arrow types τ can be defined by following three steps:

(1) stating which attributes and combinations of attributes are allowed,

(2) defining the semantics for all allowed combinations of attributes (either directly or by systematically doing the third step), and

(3) curating semantics-preserving syntactic transformations, expressed as meta-rules using Cambium if possible, from generalzed arrows with selected attribute sets for the new type to other generalized arrow expressions for prevously defined arrow types.

By these steps, the mathematical well-definedness of Cambium is maintained as new arrow types are added. Two Cambium process arrows mean the same thing if and only if their semantics is the same. Various other equivalence classes (such as identity up to renaming) can be defined as well. The semantics of an initial set of arrow types and attributes is introduced in Section 4.

Definable types could include: *DynGram* [1], *DynGraphGram, LSystem* [2], *PSystem* [Spicher et al. 2007], *κ-calculus* [Danos et al. 2004], *Cellerator* [Shapiro et al 2003], *KMech* [Yang et al 2005], *Cellzilla* [Shapiro et al. 2007], *BNGL* [Hlavacek et al. 2006], *SBML L2* [Hucka et al. 2003], *CellML* [Cuellar et al 2003], *MCell* [Casanova et al. 2001], *CPM* [Chen et al 2007], *PPM* [Sbalzerini et al. 2006], *Sigmoid* [Cheng et al 2005], *Organism* [Jönsson 2006], ... Definable attributes for many of these these types include: *reversible, mass-action, bounded, regulatory* (for $L = \varnothing$), *fast*, *continuous/discrete* (in time), *deterministic/stochastic/quantum* (dynamics), among other attribute alternatives, many of which are summarized in Section 2.2 below.

2.2 Attributes

Selected generalized arrow types and their attributes are listed in Table 1. Examples will appear in Section 2.3 .

Table 1. Types and Attributes

Type	generic attributes (default values in **bold**)	more specific attributes	clauses
DynGram [1]	deterministic/**stochastic**/ quantum, **discrete**/continuous Events	**continuous**/discrete Global Time, **multiset**/graph topology, unbounded/bounded,	With/Solving SubjectTo(condition)
DynGraphGram or DGG	"	multiset/list/tree/**graph** topology	"
Procambium	**discrete**/continuous Events, deterministic/stochastic, reversible, fast, compound, massaction/bounded/ ratelaw, ODE/DAE/SDE/PDE/SPDE/ ...	multiset/list/tree/graph topology fireonce, Let, Define, sequence	"
LSystem [2]		multiset/**list**/graph topology	precondition
PSystem		multiset/**list**/**tree** topology	
Cellerator [3]	**deterministic**/stochastic, reversible, massaction/ratelaw	regulatory, enzymatic, compound, cascade, MM/Hill/GRN/GMCW	(ratelaw)
KMech	"	BiBi/PingPong/RSS/...	
Cellzilla	"	internal, external	
future	"	**dynamic**/ steadystate/ equilibrium	
Organism			
κ-calculus			
Other models	...	...	...

When the allowed topological containers and/or term parameters in an arrow type are sufficiently flexible, then it becomes possible to represent meta-rules: arrows representing syntactic transformations of other arrows. These are useful for example in step 3 of the semantic definition process outlined in Section 2.1 above.

2.3 Examples

2.3.1 Receptor/Ligand interaction

Consider the reaction that pairs receptor R at cell i and ligand L from neighboring cell j into receptor-ligand complex $R \diamond L$ at cell i. A human-readable notation for this reaction might be

$$R_i + L_j \rightleftharpoons (R \diamond L)_i$$

and its Cellzilla presentation looks similar to this. The $i - j$ neighborhood relation is not explicit but we can call it "$\text{nbr}_{j\,i}$". Recall from Section 1.2 the "$\{..\}_*$" notation for multisets. Then the reaction can be represented in Cambium as follows:

$$\text{Arrow}[\text{Cellzilla}[\text{det},\ \text{massaction},\ \text{reversible},\ \text{external}],\ \{R[i], L[j]\}_*, \varnothing, \{(R \diamond L)[i]\}_*,\ \{\text{With}[k_f, k_r], \text{Using}[\text{nbr}, i, j,\ \text{Cell}]\}]$$

which should be equivalent to:

$$\text{Arrow}[\text{DynGram}[\text{det},\ \text{massaction},\ \text{reversible}],\ \{R[i], L[j]\}_*, \{\text{nbr}[j, i]\}_*, \{(R \diamond L)[i]\}_*,\ \{\text{With}[k_f, k_r], \text{Using}[\text{Cell}, i, j]\}]$$

which in turn could be automatically translated to

$$\text{Arrow}\Big[\text{DynGram}[\text{ratelaw}, \text{ODE}], \{\text{Cell}[\langle R_1, L_1, (R \diamond L)_1\rangle, i], \text{Cell}[\langle R_2, L_2, (R \diamond L)_2\rangle, j]\}_*,$$
$$\{\text{nbr}[j, i]\}_*, \{\text{Cell}[\langle R_1 + d\,R_1, L_1, (R \diamond L)_1 + d\,(R \diamond L)_1\rangle, i], \text{Cell}[\langle R_2, L_2 + d\,L_2, (R \diamond L)_2\rangle, j]\}_*,$$
$$\Big\{\text{Solving}\Big[\frac{d\,R_1}{d\,t} = -k_f\,R_1\,L_2 + k_r(R \diamond L)_1,$$
$$\frac{d\,L_2}{d\,t} = -k_f\,R_1\,L_2 + k_r(R \diamond L)_1,\ \frac{d\,(R \diamond L)_1}{d\,t} = k_f\,R_1\,L_2 - k_r(R \diamond L)_1\Big]\Big\}\Big]$$

or more directly, omitting the Cell compartment object as in Section 3.2 below,

$$\text{Arrow}\Big[\text{DynGram}[\text{ratelaw}, \text{ODE}], \{\text{Real}[R, i], \text{Real}[(R \diamond L), i], \text{Real}[L, j]\}_*,$$
$$\{\text{nbr}[j, i]\}_*, \{\text{Real}[R + d\,R, i], \text{Real}[(R \diamond L) + d\,(R \diamond L), i], \text{Real}[L + d\,L, j]\}_*,$$
$$\Big\{\text{Solving}\Big[\frac{d\,R}{d\,t} = -k_f\,R\,L + k_r(R \diamond L),\ \frac{d\,L}{d\,t} = -k_f\,R\,L + k_r(R \diamond L),\ \frac{d\,(R \diamond L)}{d\,t} = k_f\,R\,L - k_r(R \diamond L)\Big]\Big\}\Big]$$

These translations differ in representing enclosing compartments explicitly or only implicitly.

On the other hand the stochastic dynamics representation

$$\text{Arrow}[\text{DynGram}[\text{stoch},\ \text{massaction},\ \text{reversible}], \{R[i], L[j]\}_*, \{\text{nbr}[j, i]\}_*, \{(R \diamond L)[i]\}_*,\ \{\text{With}[k_f, k_r]\}]$$

is also valid. This process may best be understood in terms of DGG notation:

DGG[stoch, massaction, reversible] :

$$\left(\begin{array}{ccc} i & \overset{\text{nbr}}{\longrightarrow} & j \\ \uparrow\ \sqsubseteq & & \uparrow\ \sqsubseteq \\ R & & L \end{array}\right) \xrightarrow{i \overset{\text{nbr}}{\longrightarrow} j} \left(\begin{array}{ccc} i & & \overset{\text{nbr}}{\longrightarrow}\ j \\ \uparrow\ \sqsubseteq & & \\ R \diamond L & & \end{array}\right), \qquad \text{With}[k_f, k_r]$$

i.e.

$$\left(\begin{array}{ccc} i & \overset{\text{nbr}}{\longrightarrow} & j \\ \uparrow\ \sqsubseteq & & \uparrow\ \sqsubseteq \\ R & & L \end{array}\right) \underset{\text{DGG[stoch, massaction]}}{\xleftrightarrow{k_f, k_r}} \left(\begin{array}{ccc} i & & \overset{\text{nbr}}{\longrightarrow}\ j \\ \uparrow\ \sqsubseteq & & \\ R \diamond L & & \end{array}\right)$$

or

DGG[stoch, massaction, reversible] :

$$\left(\begin{array}{ccccc} & & i & & \overset{\text{nbr}}{\longrightarrow}\ j \\ & \overset{\sqsubseteq/\varnothing}{\nearrow} & & \overset{\varnothing/\sqsubseteq}{\nwarrow} & \uparrow\ \sqsubseteq\ /\varnothing \\ R/\varnothing & & & \varnothing/R\diamond L & L/\varnothing \end{array}\right), \qquad \text{With}[k_f, k_r]$$

Here the label A/B means A is in the LHS, B is in the RHS; Other labels are in the MHS, unless they are link labels and one of their anchoring nodes is missing from the input set (LHS $\bigcup$ MHS) or from the output set. Alternatively, as for example in the ratelaw, ODE version above,

DGG[stoch/det, massaction, reversible] :

$$\left(\begin{array}{ccccc} & & i & & j \\ & & \uparrow\ \text{index} & & \uparrow\ \text{index} \\ & & \text{Cell} & \overset{\text{nbr}}{\longrightarrow} & \text{Cell} \\ & \overset{\text{contains}/\varnothing}{\swarrow} & & & \downarrow\ \text{contains}/\varnothing \\ R/\varnothing & & \overset{\varnothing/\text{contains}}{\searrow} & & \\ & & \varnothing/R\diamond L & & L/\varnothing \end{array}\right), \qquad \text{With}[k_f, k_r]$$

2.3.2 Transport across a membrane

DGG[stoch, massaction, reversible] :

$$\left(\begin{array}{ccccc} i & & \overset{\partial^+}{\longrightarrow} & b & \overset{\partial^-}{\longleftarrow}\ j \\ \uparrow\ \sqsubseteq\ /\varnothing & & & \uparrow\ \sqsubseteq & \uparrow\ \varnothing/\sqsubseteq \\ A/\varnothing & & & T & \varnothing/A \end{array}\right), \qquad \text{With}[k, k]$$

The boundary relationship $\partial^{\pm}$ is signed, so we know $i \neq j$. The sign is arbitrary, so symmetry requires the forward and reverse rates to be equal.

2.3.3 Auxin signal transduction

The first use case for the iPlant Computational Morphodynamics preproject is as follows [courtesy of Alistair Middleton, Nottingham CPIB].

AM-SME (Sigmoid) source code:

```
reactionRates = List[Rule[r1, 1.0000000000],
   Rule[r2, 0.1000000000], Rule[r3, 0.1000000000], Rule[r4, 1.0000000000],
   Rule[r5, 0.5000000000], Rule[r6, 1.0000000000], Rule[r7, 1.0000000000],
   Rule[r8, 0.0100000000], Rule[r9, 1.0000000000], Rule[r10, 1.0000000000],
   Rule[r11, 1.0000000000], Rule[r12, 1.0000000000], Rule[r13, 1.0000000000]];
initialConditions = List[Rule[AuxIAAProtein, 0.0100000000], Rule[AuxIAAmRNA, 0],
   Rule[AuxinSCFTIR1, 0.0100000000], Rule[AuxinSCFTIR1AuxIAA, 0],
   Rule[Auxin, 0.1000000000], Rule[SCFTIR1, 0.3000000000], Rule[AuxIAAmod, 0]];
cmodel = Union[List[List[ShortRightArrow[AuxIAAmRNA, ∅], r1]],
   List[List[ShortRightArrow[AuxIAAmod, ∅], r2]],
   List[List[ShortRightArrow[AuxIAAProtein, ∅], r3]],
   List[List[ShortRightArrow[AuxIAAmRNA, List[AuxIAAmRNA, AuxIAAProtein]], r4]],
   List[List[ShortRightArrow[AuxinSCFTIR1AuxIAA, List[AuxinSCFTIR1, AuxIAAmod]],
     r5]], List[List[RightArrowLeftArrow[
      List[AuxIAAProtein, AuxinSCFTIR1], AuxinSCFTIR1AuxIAA], r6, r7]],
   List[List[RightArrowLeftArrow[List[Auxin, SCFTIR1], AuxinSCFTIR1], r8, r9]],
   List[List[RightArrowLeftArrow[Auxin, ∅], r10, r11]],
   List[List[RightArrowLeftArrow[SCFTIR1, ∅], r12, r13]]];
```

Sigmoid Model Explorer screen shot of this model, at the start of the iPlant Computational Morphodynamics preproject:

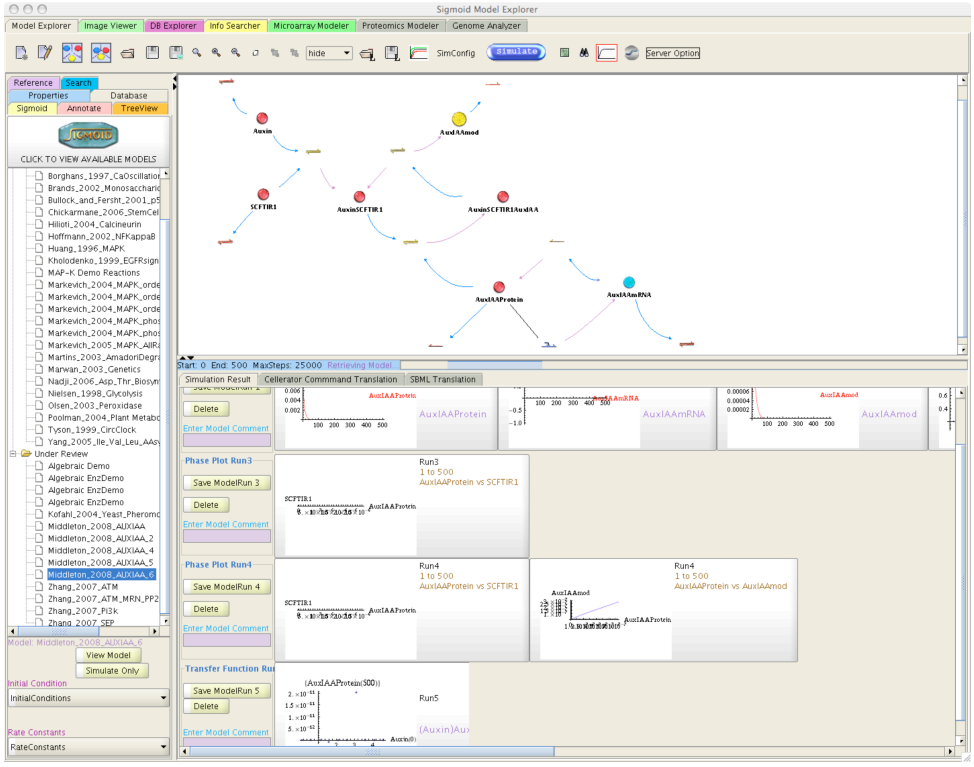

Figure 1. Screen shot of auxin signal transduction "toy" model running in Sigmoid, www.sigmoid.org .

Cambium representation thereof (produced by the first running code for Cambium, lightly edited for consistency with this document):

```
{arrow[{Procambium[Let], {}, {1.`}, {r1}}],
 arrow[{Procambium[Let], {}, {0.1`}, {r2}}],
 ...,
 arrow[{Procambium[Let], {}, {0.01`}, {AuxIAAProtein}}],
 arrow[{Procambium[Let], {}, {0}, {AuxIAAmRNA}}],
 ...,
 arrow[{Procambium[ratelaw, ODE],
   multiset[{AuxIAAmod, 1}], ∅, ∅, solving[{AuxIAAmod′ == -AuxIAAmod r2}]}],
 arrow[{Procambium[ratelaw, ODE], multiset[{AuxIAAmRNA, 1}],
   ∅, ∅, solving[{AuxIAAmRNA′ == -AuxIAAmRNA r1}]}],
 ...,
 arrow[{Procambium[ratelaw, ODE], multiset[{Auxin, 1}, {SCFTIR1, 1}],
   ∅, multiset[{AuxinSCFTIR1, 1}], solving[{Auxin′ == -Auxin r8 SCFTIR1,
     AuxinSCFTIR1′ == Auxin r8 SCFTIR1, SCFTIR1′ == -Auxin r8 SCFTIR1}]}],
 arrow[{Procambium[ratelaw, ODE], multiset[{AuxinSCFTIR1, 1}], ∅,
   multiset[{Auxin, 1}, {SCFTIR1, 1}], solving[{Auxin′ == AuxinSCFTIR1 r9,
     AuxinSCFTIR1′ == -AuxinSCFTIR1 r9, SCFTIR1′ == AuxinSCFTIR1 r9}]}]
}
```

2.3.4 Clavata/Wuschel model

The second use case for the iPlant Computational Morphodynamics preproject is the following model [4]. Using the Cellzilla augmentation of Cellerator notation,

```
internal[i] := {
  {{X[i]} ↦ CLV3[i], GRN[Abs[Tip[i][t]] r_CLV3, {T_X,CLV3}, 1, 0, sigma]},
  {{WUS[i], L1Signal[i], BottomSignal[i]} ↦ CLV1[i],
   GRN[r_CLV1, {T_WUS,CLV1, T_L1,CLV1, T_Bottom,CLV1}, 1, 0, sigma]},
  {{Y[i], L1Signal[i], BottomSignal[i]} ↦ WUS[i],
   GRN[r_WUS, {T_Y,WUS, T_L1,WUS, T_Bottom,WUS}, 1, 0, sigma]}, {WUS[i] → WUS[i] + X[i], k_X,f},
  {CLV3[i] → ∅, k_CLV3},
  {WUS[i] → ∅, k_W},
  {X[i] → ∅, k_X,r},
  {CLV1[i] → ∅, k_CLV1},
  {CLV1[i] + CLV3[i] ⇄ CLV13[i], k1, k2}, {CLV13[i] → CLV13[i] + Y[i], k_f,Y},
  {Y[i] → ∅, k_r,Y},
  {L1[i] → L1[i] + L1Signal[i], k_L11},
  {L1Signal[i] → ∅, k_L12},
  {BottomMarker[i] → BottomMarker[i] + BottomSignal[i], k_B1},
  {BottomSignal[i] → ∅, k_B2},
  {Tip[i] → Tip[i], 0}}

external[i, j] := {{CLV3[i] → CLV3[j], D_CLV3}, {L1Signal[i] → L1Signal[j], D_L1},
  {BottomSignal[i] → BottomSignal[j], D_Bottom}, {X[i] → X[j], D_X}}
```

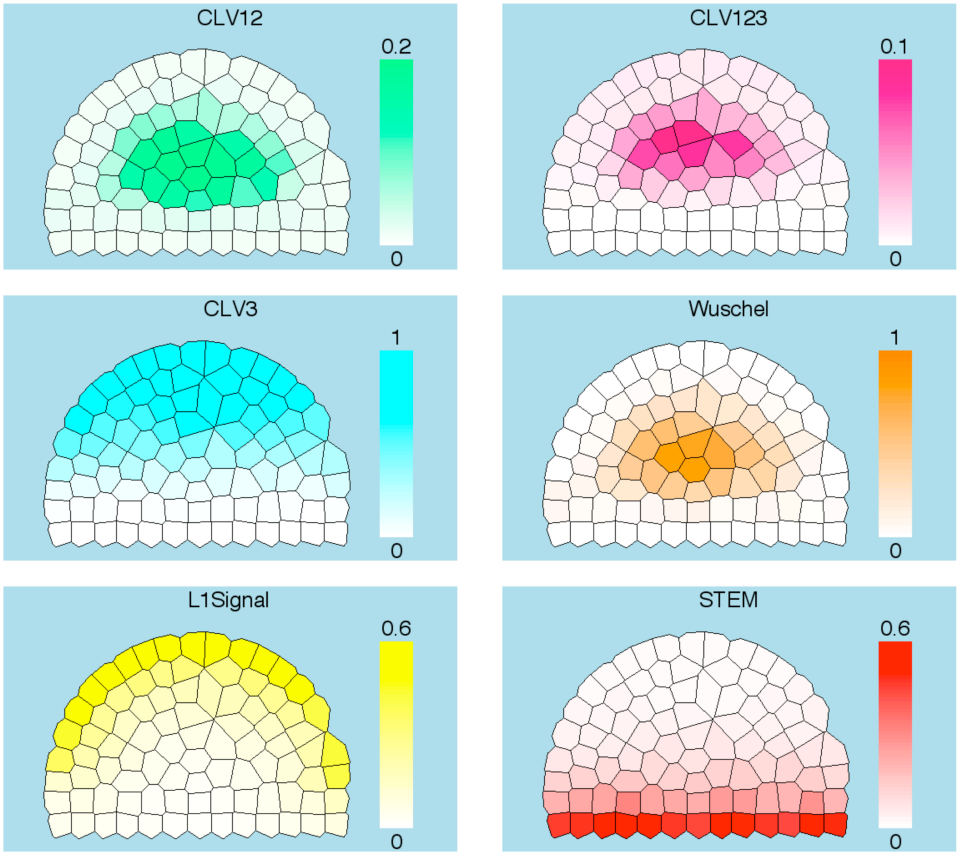

Figure 2. Stable state resulting from the Cellzilla CLV/WUS model, with Voronoi diagram topology.

This can be transcribed into Cambium as something like

```
{
    arrow[Procambium[GRN], ∅,
      MultiSet[{X[i][t], 1}], MultiSet[{CLV3[i][t], 1}], ∅],
    arrow[Procambium[GRN], ∅, MultiSet[{BottomSignal[i][t], 1},
      {L1Signal[i][t], 1}, {WUS[i][t], 1}], MultiSet[{CLV1[i][t], 1}], ∅],
    ...,
    arrow[Procambium[MassAction],
     MultiSet[{WUS[i][t], 1}], ∅,
     MultiSet[{WUS[i][t], 1}, {X[i][t], 1}], ∅],
    ...,
    arrow[Procambium[MassAction], MultiSet[{X[i][t], 1}],
     {ConnectionMatrix[i, j]},
     MultiSet[{X[j][t], 1}], with[ConnectionMatrix[i, j]]]
}
```

Thence to a model whose mathematical meaning is built out of ODE's:

$$\left\{
\begin{array}{l}
\texttt{arrow}\left[\texttt{Procambium[ratelaw, ODE]},\right. \\
\quad \left.\varnothing, \texttt{MultiSet[\{X[i][t], 1\}]}, \texttt{MultiSet[\{CLV3[i][t], 1\}]},\right. \\
\quad \texttt{solving}\left[\right. \\
\texttt{ODETerm}\left[\texttt{CLV3[i][t]}, \frac{1}{2}\,\texttt{Abs[Tip[i][t]]}\, r_{\text{CLV3}} \left(1 + \frac{T_{\text{X,CLV3}}\, \texttt{X[i][t]}}{\sqrt{1 + T_{\text{X,CLV3}}^2\, \texttt{X[i][t]}^2}}\right)\right]\right]\right], \\
\texttt{arrow}\left[\texttt{Procambium[ratelaw, ODE]},\right. \\
\quad \texttt{MultiSet[\{BottomSignal[i][t], 1\},} \\
\texttt{\{L1Signal[i][t], 1\}, \{WUS[i][t], 1\}],} \\
\quad \texttt{\{\}, MultiSet[\{CLV1[i][t], 1\}], solving}\left[\texttt{ODETerm}\left[\texttt{CLV1[i][t]},\right.\right. \\
\frac{1}{2}\, r_{\text{CLV1}} \\
\left(1 + (T_{\text{Bottom,CLV1}}\, \texttt{BottomSignal[i][t]} + T_{\text{L1,CLV1}}\, \texttt{L1Signal[i][t]} + T_{\text{WUS,CLV1}}\, \texttt{WUS[i][}\right. \\
\quad \left.\left.\left.\left.\texttt{t]}) \Big/ \left(\sqrt{\left(1 + (T_{\text{Bottom,CLV1}}\, \texttt{BottomSignal[i][t]} + T_{\text{L1,CLV1}}\, \texttt{L1Signal[i][t]} + \right.\right.\right.\right. \\
\quad \left.\left.\left.\left.\left.\left.T_{\text{WUS,CLV1}}\, \texttt{WUS[i][t]})^2\right)\right)\right]\right]\right], \\
\texttt{...}, \\
\texttt{arrow[Procambium[ratelaw, ODE],} \\
\quad \texttt{MultiSet[\{WUS[i][t], 1\}], } \varnothing, \\
\quad \texttt{MultiSet[\{WUS[i][t], 1\}, \{X[i][t], 1\}],} \\
\quad \texttt{solving[ODETerm[WUS[i][t], 0], ODETerm[X[i][t], } k_{\text{X,f}}\, \texttt{WUS[i][t]]]]}, \\
\texttt{...}, \\
\texttt{arrow[Procambium[ratelaw, ODE],} \\
\texttt{MultiSet[\{X[i][t], 1\}], \{ConnectionMatrix[i, j]\},} \\
\quad \texttt{MultiSet[\{X[j][t], 1\}],} \\
\quad \texttt{\{solving[ODETerm[X[i][t], -ConnectionMatrix[i, j] * } D_{\text{X}}\, \texttt{X[i][t]],} \\
\quad \texttt{ODETerm[X[j][t], ConnectionMatrix[i, j] * } D_{\text{X}}\, \texttt{X[i][t]]]\}]}
\end{array}
\right\}$$

2.3.5 L-system example: *Anabaena* model

Here is a differential L-system model for 1D growth of *Anabaena catenula* [5]:

axiom: $F_h(s_{\max}, c_{\max})\, F_v(s_{\max}, c_{\max})\, F_h(s_{\max}, c_{\max})$

$F(s_l, c_l) < F_v(s, c) > F(s_r, c_r)$:

if $s < s_{\max}$ & $c > c_{\min}$

solve $d\,c/d\,t = D\,(c_l + c_r - 2\,c) - \mu\,c$

$d\,s/d\,t = r\,s$

if $s = s_{\max}$ & $c > c_{\min}$

produce $F_v(k\,s_{\max}, c)\, F_v((1-k)\,s_{\max}, c)$

if $c = c_{\min}$

produce $F_h(s, c)$

$F_h(s, c)$:

solve $d\,s/d\,t = r\,s(s_{\max} - s)$

$d\,c/d\,t = r\,c\,(c_{\max} - c)$

This model was translated into DG's in [1].

The *Anabaena* model could be encoded first in Cambium, directly and without significant translation, as something like

{

$$\text{Arrow[LSystem[axiom]}, \varnothing, \varnothing, \langle F_h(s_{\max}, c_{\max}), F_v(s_{\max}, c_{\max}), F_h(s_{\max}, c_{\max})\rangle, \varnothing]\ ,$$

$$\text{Arrow[LSystem[continuous\_event]},\ \langle F(s_l, c_l), F_v(s, c), F(s_r, c_r)\rangle, \varnothing, \langle F(s_l, c_l), F_v(s, c), F(s_r, c_r)\rangle,\ \{\text{Precondition}[s < s_{\max} \wedge c > c_{\min}],\ \text{Solve}[d\,c/d\,t = D\,(c_l + c_r - 2\,c) - \mu\,c,\ d\,s/d\,t = r\,s]\ \}]\ ,$$

$$\text{Arrow[LSystem[discrete\_event]}, \langle F(s_l, c_l), F_v(s, c), F(s_r, c_r)\rangle, \varnothing,\ \langle F(s_l, c_l), F_v(k\,s_{\max}, c), F_v((1-k)\,s_{\max}, c), F(s_r, c_r)\rangle,\ \{\text{Precondition}[s = s_{\max} \wedge c > c_{\min}]\}]\ ,$$

$$\text{Arrow[LSystem[discrete\_event]}, \langle F(s_l, c_l), F_v(s, c), F(s_r, c_r)\rangle, \varnothing,\ \langle F(s_l, c_l), F_h(s, c), F(s_r, c_r)\rangle,\ \{\text{Precondition}[c = c_{\min}]\}]\ ,$$

$$\text{Arrow[LSystem[continuous\_event]}, \langle F_h(s, c)\rangle, \varnothing, \langle F_h(s, c)\rangle,\ \{\text{Solve}[d\,s/d\,t = r\,s(s_{\max} - s),\ d\,c/d\,t = r\,c\,(c_{\max} - c)]\ \}]$$

}

It could then be translated into Procambium type arrows or some other attributed arrow type. A possible translation from this intermediate form to a Procambium intermediate form might be:

{

$$\text{Arrow[Procambium[fire\_once, fast], } \varnothing, \varnothing, \langle F(\text{" } h \text{ "}, s_{\max}, c_{\max}), F(\text{" } v \text{ "}, s_{\max}, c_{\max}), F(\text{" } h \text{ "}, s_{\max}, c_{\max})\rangle, \varnothing] \text{ ,}$$

$$\begin{gathered}\text{Arrow[Procambium[ratelaw, ODE], } \langle F(\sigma, s_1, c_1), F(\text{" } v \text{ "}, s, c), F(\sigma, s_2, c_2)\rangle, \varnothing, \\ \langle F(\sigma, s_1, c_1), F(\text{" } v \text{ "}, s, c), F(\sigma, s_2, c_2)\rangle, \\ \{\text{SubjectTo}[s < s_{\max} \wedge c > c_{\min}], \\ \text{Solving}[d\,c/d\,t = D\,(c_1 + c_2 - 2\,c) - \mu\,c,\ d\,s/d\,t = r\,s]\ \}] \text{ ,}\end{gathered}$$

$$\begin{gathered}\text{Arrow[Procambium[det, fast], } \langle F(\sigma, s_1, c_l), F(\text{" } v \text{ "}, s, c), F(\sigma, s_2, c_2)\rangle, \varnothing, \\ \langle F(\sigma, s_1, c_1), F(\text{" } v \text{ "}, k\,s_{\max}, c), F(\text{" } v \text{ "}, (1-k)\,s_{\max}, c), F(\sigma, s_2, c_2)\rangle \text{ ,} \\ \{\text{SubjectTo}[s = s_{\max} \wedge c > c_{\min}]\}] \text{ ,}\end{gathered}$$

$$\begin{gathered}\text{Arrow[Procambium[det, fast], } \langle F(\sigma, s_1, c_1), F(\text{" } v \text{ "}, s, c), F(\sigma, s_2, c_2)\rangle, \varnothing, \\ \langle F(\sigma, s_1, c_1), F(\text{" } h \text{ "}, s, c), F(\sigma, s_2, c_2)\rangle, \{\text{SubjectTo}[c = c_{\min}]\}] \text{ ,}\end{gathered}$$

$$\begin{gathered}\text{Arrow[Procambium[ratelaw, ODE], } \langle F(\text{" } h \text{ "}, s, c)\rangle, \varnothing, \langle F(\text{" } h \text{ "}, s, c)\rangle, \\ \{\text{Solving}[d\,s/d\,t = r\,s(s_{\max} - s),\ d\,c/d\,t = r\,c\,(c_{\max} - c)]\ \}]\end{gathered}$$

}

Alternatively the *Anabaena* model could be translated from some other arrow type into the second form above, thence to the first one, and then executed directly by invoking the appropriate L+C simulator [2].

2.3.6 Meta-rule

The rule-to-rule transformations foreseen in Section 2.1 (step 3) may themselves be expressible in Cambium. For example:

$$\begin{gathered}\text{Procambium[meta] :} \\ \text{Arrow[LSystem[continuous\_event], LHS, MS, RHS, \{Precondition[}P\text{], Solve[odesystem]\}]} \longrightarrow \\ \text{Arrow[Procambium[ratelaw, ODE], LHS, MS, RHS, \{SubjectTo[}P\text{], Solving[odesystem]\}]}\end{gathered}$$

or, in standardized form,

$$\begin{gathered}\text{Arrow[Procambium[meta],} \\ \text{Arrow[LSystem[continuous\_event], LHS, MS, RHS, \{Precondition[}P\text{], Solve[odesystem]\}], } \varnothing, \\ \text{Arrow[Procambium[ratelaw, ODE], LHS, MS, RHS, \{SubjectTo[}P\text{], Solving[odesystem]\}], } \varnothing]\end{gathered}$$

The "meta" attribute implies at least the "fast" attribute described below, since it is to fire before any ordinary or slow rules.

3 Cambium Syntax

We will generally use “reaction”, “generalized reaction”, and “rule” interchangably.

3.1 Sequence and fire-once constructs

Fire-once rules can only fire once within the execution of a sequence, and are denoted by the attribute fireonce. They may also have the attribute fast. The semantics of these attributes is discussed in Section 4.3 .

General syntactic form for a sequence:

$$\text{Arrow[Procambium[sequence], } * \text{, } \varnothing \text{, } * \text{, \{Constituents[} \\ \text{reaction}_1 \text{ ,} \\ \text{...} \\ \text{reaction}_k \\ \text{]\}]}$$

Here reaction_1 ...reaction_k are more Cambium arrows.

We could in addition recognize a “pretty” shorthand form such as

$$\langle\ \text{reaction}_1 \text{ ,} \\ \text{... ,} \\ \text{reaction}_k\ \rangle$$

in which angle brackets replace braces in the usual aggregation of rules to form a grammar. Then at least in Dynamical Grammars, nested combinations of grammars and sequences would be syntactically simplified:

$$\text{grammar(...) } \langle \text{r1, \{r2, r3, } \langle \text{r4, r5} \rangle \text{, r6\}, \{r7, r8\}} \rangle \text{, or inversely} \\ \text{grammar(...) \{r1, } \langle \text{r2, r3, \{r4, r5\}, r6} \rangle \text{, } \langle \text{r7, r8} \rangle \text{\},}$$

would make sense. Nested sequences and nested sets each can be flattened out without loss of generality, provided that sets of rules are combined as multisets and then converted to sets. A related alternative is to recognize a category of process aggregate entities - such as models, grammars, sequences, and perhaps programs - that are not Arrows but whose constituents may be.

Such a compound arrow could also be used to preserve the grouping of arrows in the Anabaenda L-system example of Section 2.3.5 .

There is a question about what to do with the LHS and RHS of the main reaction in a sequence. This is analogous to the treatment of LHS and RHS for entire grammars in Dynamical Grammars, where a filtering operation is introduced to minimize interdependence. We could take the LHS and RHS to be the entire current pool (hence "*") by default, for sequential and parallel aggregates (denoted $\langle$, ...$\rangle$ and { , ...} respectively), and reserving some heavy-duty aggregate like “grammar” for use in filtering the pool to introduce modularity. Then we would have syntactic constructs such as:

$$\text{FilteredProcess[Procambium[model || grammar], LHS, MS, RHS, CompoundProcess[...]]}$$

and

CompoundProcess[Procambium[sequence || set], {
reaction$_1$,
...
reaction$_k$
}]

3.2 Initial condition specification

Suppose we want to initialize the concentration of Wuschel in many cells indexed by i to the values stored in the array $W_i[0]$. Poetically, we wat to use the values of one sequence of variables to set the values of another sequence of variables:

$$\text{Let} \quad [\text{Winit}[i] \mid 1 \leqslant i \leqslant i_{\max}] \mapsto [\text{Wus}[i] \mid 1 \leqslant i \leqslant i_{\max}]$$

or more prosaically,

$$\text{Arrow}[\text{Procambium}[\text{Let}], \varnothing, \{[\text{Winit}[i] \mid 1 \leqslant i \leqslant i_{\max}]\}, \{[\text{Wus}[i]\ , i] \mid 1 \leqslant i \leqslant i_{\max}]\}, \varnothing\}]$$

Even this requires an extension of notation. Safest would be the pedestrian

$$\{\text{Arrow}[\text{Procambium}[\text{Let}], \varnothing, \{\text{Winit}[i]\}, \{\text{Wus}[i]\}, \varnothing] \mid 1 \leqslant i \leqslant i_{\max}\} \tag{1}$$

"Let" is intrinsically fast and fire-once, so we could equivalently write Procambium[Let, fast, fireonce] in the foregoing. We now discuss the possible interpretations of Equation 1 in terms of meaningful Dynamical Grammar constructs.

A number of language questions are raised even by this example. This is because of a clash of paradigms between conventional programming, where symbols have current values and arrays are declared and sized ahead of their use, and DG modeling. For example under the semantics of Dynamical Grammars, the only hidden or implicit value associated with a parameterized object is a nonnegative copy number. Therefore the array references Winit[i] and Wus[i] only have implicit values if they are nonnegative integers, interpreted as numbers of identical copies. This would never be true for Winit, which by definition comprises externally supplied data, and would not be true for Wus in a model that represents concentrations.

Such arrays of value-bearing variables may however be modeled within DG as eg. Real[Winit[i], " Winit ", i] or perhaps Real[Winit[i], " Winit ", i, $i-1$, $i+1$] where the inner Winit[i] and the index expressions are actual values, rather than syntax trees, and "Winit" on the other hand is a name. If the type Real or NatNum is constrained to have copy number zero or one, then the pool of active terms can be kept consistent in either asserting some unique value for Winit[i] , or no value. The initialization of Equation 1 above could then be interpreted as

$$\{\text{Arrow}[\text{DynGram}[\text{fire\_once}, \text{fast}], \varnothing, \{\text{Real}[w, \text{" Winit "}, i]\}, \{\text{Real}[w, \text{" Wus "}, i]\}, \varnothing\}] \mid 1 \leqslant i \leqslant i_{\max}\}$$

or

$$\{\text{Arrow}[\text{DynGram}[\text{fire\_once}, \text{fast}], \varnothing, \{\text{eval}(\text{Winit}[i])\}, \{\text{Real}[\text{eval}(\text{Winit}[i]), \text{" Wus "}, i]\}, \varnothing\}] \mid 1 \leqslant i \leqslant i_{\max}\}$$

where eval(...) is called during a preprocessing phase from within some larger computational evironment. Of course, the real number returned by eval(Winit[i]) is not depleted from the pool by the second interpretation

above - if we assume it is in the pool in the first place which the first interpretation doesn't require. Either way, the interpretation of Wus[i] as Real[Wus[i], "Wus", i] would also make sense of the Cellzilla notation for reactions among real-valued concentration variables, even in the absence of any enclosing compartment object such as Cell in Section 2.3.1 that can play this role.

Without this interpretation, if we are somehow writing arrows automatically within an environment that defines the Winit array, then

$$\{\text{Arrow}[\text{DynGram}[\text{fire\_once}, \text{fast}], \varnothing, \varnothing, \text{Wus}[\text{eval}(\text{Winit}[i]), i], \varnothing\}] \mid 1 \leqslant i \leqslant i_{\max}\}$$

would at least evaluate to the the right thing if the real-valued concentration "Wus[i]" is regarded as a hidden first argument in the Wus syntax tree Wus[w, i] .

Initialization of integer copy numbers is syntactically different from initialization of real-valued concentrations or any other parameters. In that case Wus[i] needs no special interpretation other than as the nonnegative copy number n_{Wus} , and Equation 1 can be interpreted as being equivalent to:

$$\{\text{Arrow}[\text{DynGram}[\text{fire\_once}, \text{fast}], \varnothing, \{\text{eval}(\text{Winit}[i])\}, \{\text{eval}(\text{Winit}[i]) * \text{Wus}[i]\}_*, \varnothing\}] \mid 1 \leqslant i \leqslant i_{\max}\}$$

In general the pre-execution invokation of eval( ) can be made implicit by use of suitable notation. Examples include ordinary parentheses for function evaluation $f(\ldots)$ as distinct from abstract syntax trees $f[\ldots]$, and the use of double brackets $[\![\ldots]\!]$ to denote preexecution substitution of labels as in $f[x, y, z, y][\![a, b, a]\!]$ which evaluates to $f[a, b, a, b]$.

Another application of discrete rule sequences is in creating prior probability distributions on structured images for use as generative models in statistical image analysis. Such "visual grammars" [NIPS 1989] have been the subject of much recent progress .in their practical application [NIPS workshop 2007].

3.3 Textual forms

The following examples have been autoproduced in both textual and *Mathematica* compatible formats, starting from Cellerator style notation. From *Mathematica* expressions one can produce mathml/xml formatted syntax trees, as foreseen in the current iPlant Computational Morphology preproject plan [M. Vaughn, private communication].

3.3.1 Auxin signal transduction

The initial conditions are as discussed above.

```
arrow[Let,∅,{1.},{r1}], arrow[Let,∅,{0.1},{r2}], arrow[Let,∅,{0.1},{r3}], arrow[Let,{},{1.},{r4}],
arrow[Let,∅,{0.5},{r5}], arrow[Let,∅,{1.},{r6}], arrow[Let,∅,{1.},{r7}], arrow[Let,{},{0.01},{r8}],
arrow[Let,∅,{1.},{r9}], arrow[Let,∅,{1.},{r10}], arrow[Let,∅,{1.},{r11}], arrow[Let,{},{1.},{r12}],
arrow[Let,∅,{1.},{r13}],
arrow[Let,∅,{0.01},{AuxIAAProtein}], arrow[Let, ∅, {0},
    {AuxIAAmRNA}], arrow[Let,∅,{0.01},{AuxinSCFTIR1}],
arrow[Let,∅,{0},{AuxinSCFTIR1AuxIAA}], arrow[Let,∅,{0.1},{Auxin}],
```

```
arrow[Let,∅,{0.3},{SCFTIR1}],
arrow[Let,∅,{0},{AuxIAAmod}],
arrow[
 Procambium[ratelaw,ODE],
 multiset[{AuxIAAmod, 1}],
 multiset[],
 multiset[],
 solving[{
  AuxIAAmod'=-(AuxIAAmod*r2)}]
],
arrow[
 Procambium[ratelaw,ODE],
 multiset[{AuxIAAmRNA, 1}],
 multiset[],
 multiset[],
 solving[{
  AuxIAAmRNA'=-(AuxIAAmRNA*r1)}]
],
arrow[
 Procambium[ratelaw,ODE],
 multiset[{AuxIAAmRNA, 1}],
 multiset[],
 multiset[{AuxIAAmRNA, 1}, {AuxIAAProtein, 1}],
 solving[{
  AuxIAAmRNA'=0,
  AuxIAAProtein'=AuxIAAmRNA*r4}]
],
...
```

3.3.2 MathML/XML format

Automatic conversion of the forgoing formats to MathML (and hence to XML) is also available, since the alpha version software is written in *Mathematica*.

3.4 Mechanical models

Cambium can represent dynamics on labelled graphs and therefore discretized cell complexes whose compartments are labelled by algebraic or discretized representations of functions such as distances, level sets that define compartment boundaries, diffusable substance concentrations, and stress and strain fields. With such representations, finite element methods and other approaches to mechanical modeling can in principle be supported.

3.5 Programming constructs

In addition to finite sequences, loops of sequentially executed rules are possible. The semantics for loops are outlined in Section 4.4, although no special syntax is proposed here. Subroutine and macro calls are already present in Dynamic Grammars, using the **via** and **substituting** keyword clauses. By these means we can introduce any degree of programming required for Cambium to serve its functions in intermodel translation and simulation.

A major difference between Cambium and conventional programming languages is the default aggregation of reaction rules within unordered sets, whose compositional semantics is obtained just by summing the individual rules' time-evolution operators. This semantics corresponds naturally to the continuous-time, asynchronous, locally independent, causal, and highly parallel world of physics and biology.

The definition of general and parallel programming languages by means of transformations on multi-sets is studied in detail in [McEvoy, 1997].

4 Cambium Semantics

Most Cambium semantics can be specified by syntactic reduction to Dynamical Grammars and/or Dynamical Graph Grammars, which have well-defined semantics in terms of time-evolution operators and the Master Equation or the Chapman-Kolmogorov equation.

Novel semantics are required for new attributes and constructs including fire-once rules, Let and Define rules, and Sequence compound arrows. Let and Define rules are needed for initialization; they are fast, fire-once rules. Fire-once rules fire exactly once per occurence in the execution of a Sequence. A Sequence may or may not be fire-once. We must define all of these behaviors in terms of time-evolution operators.

4.1 Conventional reaction processes

The semantics of Dynamic Grammars and Dynamic Graph Grammars both assume a hidden integer-valued variable for each parameterized object: the current number of identical copies of that object, with the given parameter values. If continuous variables are required they must be treated as parameters, which can take values in any measure space so that integration over parameter values can be defined.

For each combination of object type and parameter values, with maximal copy number $n_{\max} = \infty$, define the creation and annihilation matrices that increase or decrease copy number n as follows:

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

Using this notation, the time-evolution operators for a generalized reaction should destroy all parameterized objects on the LHS and then, with no delay, create all parameterized objects on the RHS [1]:

$$\hat{W}_r = \rho_r([x_a],[y_b]) \left[\prod_{b\in \text{rhs}(r)} \hat{a}_{j(b)}(y_b)\right] \left[\prod_{a\in \text{lhs}(r)} a_{i(a)}(x_a)\right] \tag{2}$$

Equivalently any unchanged objects in both LHS and RHS can be grouped together in the "MS". The time-evolution operator makes it explicit that such objects are only counted, not altered, by a reaction event:

$$\hat{W}_r = \rho_r([x_a],[y_b],[z_c]) \left[\prod_{c\in \text{rhs}(r)} \hat{a}_{j(c)}(z_c)\right] \left[\prod_{b\in \text{mhs}(r)} N_{j(b)}(y_b)\right] \left[\prod_{a\in \text{lhs}(r)} a_{i(a)}(x_a)\right] \tag{3}$$

Here $N = \hat{a}\,a$ is the number operator, which is diagonal and therefore doesn't change any particle numbers; it just provides information about how many particles of a given type are present.

Given the operators $\hat{W}_r$ for individual generalized reactions, the full time-evolution operator is

$$W = \sum_r W_r, \text{ where}$$

$$W_r = \hat{W}_r - D_r \text{ and } D_r = \text{diag}\left(\mathbf{1}\cdot\hat{W}_r\right)$$

and the Master Equation for evolution of state probabilities is $d\,p/d\,t = W\cdot p$. From this equation one can derive the Chapman-Kolmogorov equation and the Gillespie Stochastic Simulation algorithm, among others. Its formal solution is simply $p(t) = \exp(t\,W)\cdot p(t)$.

The semantics of Equation 2 and Equation 3 can be extended to graph grammars either by reduction of graph grammars to stochastic parameterized grammars [1], or directly in terms of labelled graphs in the LHS and RHS [DGG, in preparation].

Likewise, the semantics of models specified partly or entirely by differential equations can be expressed in the same kind of operator algebra by using differential operators.

4.2 Fast rules

Rules with attribute "fast" act as if their time-evolution operators W_r were all multiplied by a constant Λ, and then a limit $\Lambda \to \infty$ were taken on the solutions $p(t)$ to the Master Equation. Rules with attribute "fast[α]" likewise are multiplied by Λ^α before this limit is taken. So it is possible to have fast fast fast rules (i.e. fast[3] rules), but also fast[$^1/_3$] and fast[-3] rules. In general

$$W(\Lambda) = \sum_r \Lambda^{\alpha(r)}\, W_r$$

The default speedup exponent for each rule is of course $\alpha(r) = 0$. Since α is part of an attribute of rule r, and not an object parameter, it is a constant whose value can only be changed by a metarule.

4.3 Fire-once rules

Execution of a reaction or rule may be controlled by adding a required input token such as "start" to the LHS, but not the RHS. If this token has a maximum copy number of one, then the reaction can only fire once. The token itself is just an unparameterized object not used elsewhere except to set up the initial unfired condition. Reactions not already in this form, but declared to be fire-once, could be transformed into this form.

The semantics of this situation can be modeled by a "fire-once matrix" in the two-dimensional state space given by the presence or absence of the special token:

$$\hat{F} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix},\ D_F = \text{diag}\left(\mathbf{1}\cdot\hat{F}\right) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

$$F = \hat{F} - D_F = \begin{pmatrix} -1 & 0 \\ 1 & 0 \end{pmatrix}.$$

We will take the tensor product of 2x2 matrices like these, representing the evolution of the present/absent state of the start token, with the potentially much larger time-evolution matrices of the previous section, representing the evolution of the rest of the system under the influence of a particular rule. The tensor product means that the two changes happen in different spaces, without influencing each other, but simultaneously.

Given a time-evolution operator $W = \hat{W} - D$ from the conventional (not fire-once) semantics of an arrow, which for simplicity we assume has no terminal states, we construct the combined dynamics $\tilde{W}$:

$$\hat{\tilde{W}} = \hat{F} \otimes \hat{W} \quad \Rightarrow \quad \tilde{W} = \hat{F} \otimes \hat{W} - D_F \otimes D\ .$$

We now find the future evolution of the entire system, $\exp\left(t\,\tilde{W}\right)$, from the Taylor series:

$$\exp\left(t\,\tilde{W}\right) = I + t\left(\hat{F} \otimes \hat{W} - D_F \otimes D\right) + \frac{t^2}{2}\left(\hat{F} \otimes \hat{W} - D_F \otimes D\right)^2 + \frac{t^3}{3!}\left(\hat{F} \otimes \hat{W} - D_F \otimes D\right)^3 + \ldots\ .$$

Using the algebra

$$\hat{F}^2 = 0,\ \hat{F}\,D_F = \hat{F},\ \text{and}\ D_F\,\hat{F} = 0\ ,$$

we calculate

$$\exp\left(t\,\tilde{W}\right) = I + t\left(\hat{F} \otimes \hat{W} - D_F \otimes D\right) + \frac{t^2}{2}\left(-\hat{F} \otimes \hat{W}\,D + D_F \otimes D^2\right) + \frac{t^3}{3!}\left(\hat{F} \otimes \hat{W}\,D^2 - D_F \otimes D^3\right) + \ldots$$

whence

$$\exp\left(t\,\tilde{W}\right) = I + \hat{F} \otimes \hat{W}(I - \exp(-t\,D))\,/\,D - D_F \otimes (I - \exp(-t\,D)).$$

In the long-time (fast subprocess) limit, this is just

$$\lim_{t\to\infty} \exp\left(t\,\tilde{W}\right) = \hat{F}\otimes\left(\hat{W}/D\right) + (I - D_F)\otimes I$$

The second term may be eliminated by starting in the "0" (start) state. The factor of $\hat{F}$ in the first term just says the rule eventually fires. The subtantive part of the resulting limit is just

$$\mathcal{M} = \hat{W}/D$$

as expected. For the treatment of the case in which W may have terminal states, see the discrete-time semantics of [1].

Clearly this token-consumption transformation could be generalized to fire-n-time reactions by changing the maximum copy number of the input token to n.

4.4 Sequences, loops, and NFSA control

The token mechanism can be generalized to finite Sequences by adding a special output token for each rule, which is the required input token for the next one. A similar calculation starting from

$$\hat{F} = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 0 & & 0 & 0 \\ 0 & 1 & 0 & & 0 & 0 \\ \vdots & & \ddots & \ddots & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} = \hat{a}_{\text{counter}},\ \hat{F}^k = 0$$

shows that for a Sequence of k fast rules,

$$\mathcal{M} = \left(\hat{W}_k/D_k\right)\ldots\left(\hat{W}_2/D_2\right)\left(\hat{W}_1/D_1\right)\ ,$$

again as expected.

Looping may be obtained by having the final output token in a sequence be the same as the first input token, rather than having $k+1$ different tokens. A loop can be exited, or a sequence can branch, by having several rules share an input token. The choice of branch taken can be regulated by parameter-dependent relative reaction rates, which depend on non-token parameterized objects. Convergence in execution paths is obtained by sharing output tokens. By these means, control by any Finite State Automaton (FSA) (no branching) or Nondeterministic Finite State Automaton (NFSA) can be imposed on a collection of rules. Sequences and loops are inherently non-parallel computationally, but parallelism may be obtained by adding an integer parameter p (an arbitrary process number) to all of the tokens; p must match agree between input and output tokens for each rule in the FSA, as for example rules of the form: token(p, k), LHS $\rightarrow$ token(p, $k+1$), RHS. Then sequences with different values of p will interleave arbitrarily in time, but for each value of p, processes will remain sequentialized.

5 Acknowledgements

We have benefitted from discussions with Premyslaw Prusinkiewicz. This work was prepared in part for the "iPlant" preproject in Computational Morphodynamics. Work supported in part by the National Science Foundation's Frontiers in Biological Research (FIBR) program, award number EF-03307686, supporting the "Computable Plant" project, and by the Beckman Network Modeling Center (BNMC) at the California Institute of Technology.

References

[1] Mjolsness, E., & Yosiphon, G. (2007, January). *Stochastic Process Semantics for Dynamical Grammars*. Annals of Mathematics and Artificial Intelligence, **47**(3-4).

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

[3] Jönsson, H., et al. (2005). *Modeling the Organization of the WUSCHEL Expression Domain in the Shoot Apical Meristem*. In *Procedings of Intelligent Systems in Molecular Biology* .

[4] Federl, P., & Prusinkiewicz, P. (2004). *Solving Differential Equations in Developmental Models of Multicellular Structures Expressed Using L−systems*. Proceedings of Computational Science.

[5] Prusinkiewicz, P., Karwowski, R., & Lane, B. (2007). *The L+C plant modelling language*.. In J. Vos (Ed.), *Functional-Structural Plant Modelling in Crop Production* .

References for modeling languages and software:

Hlavacek WS, et al. (2006) Rules for modeling signal-transduction systems. Science's STKE 2006, re6.

Antoine Spicher, Olivier Michel, Mikolaj Cieslak, Jean-Louis Giavitto, Przemyslaw Prusinkiewicz. Stochastic P systems and the simulation of biochemical processes with dynamic compartments. Biosystems 2007. See also http://ppage.psystems.eu/ .

Danos and Laneve, Formal molecular biology. Theoretical Computer Science 325(1), 69-110, 2004.

Hugh McEvoy, "Coordinating multiset transformers". PhD thesis, University of Amsterdam, 1997.

Nan Chen, James A. Glazier, Jesús A. Izaguirre, and Mark S. Alber, "A parallel implementation of the Cellular Potts Model for simulation of cell-based morphogenesis", Comput Phys Commun. 2007 June; 176(11-12): 670–681. doi: 10.1016/j.cpc.2007.03.007.

I. F. Sbalzarini, J. H. Walther, M. Bergdorf, S. E. Hieber, E. M. Kotsalis, and P. Koumoutsakos. PPM – a highly efficient parallel particle-mesh library. J. Comput. Phys., 215(2):566–588, 2006.

L Systems: http://www.algorithmicbotany.org

MGS: http://mgs.ibisc.univ-evry.fr/

H. Jönsson, "Organism" simulator: private communication 2005.

OpenAlea: http://openalea.gforge.inria.fr/dokuwiki/doku.php

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

"The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models", M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano , A. P. Arkin , B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J.-H. Hofmey, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Nov`ere, L. M. Loew, D. Lucio , P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, Bioinformatics Vol 19, no 4, pp 524-531, 2003.

Yang, C-R., Shapiro, B.E., Mjolsness, E.D., and Hatfield, G.W.. Bioinformatics, "An enzyme mechanism language for the mathematical modeling of metabolic pathways.", vol. 21 no. 6, pages 774–780, March 2005.

"Platforms for Modeling in Systems Biology: Recent Developments in MathSBML and Cellerator", Bruce E. Shapiro, James Lu, Michael Hucka, Eric D. Mjolsness, Poster and Proceedings extended abstract, ICSB 2007: The Eighth International Conference on Systems Biology, Long Beach California, Oct 2-4 2007.

Cuellar, A.A., Lloyd, C.M., Nielsen, P.F., Bullivant, D.P., Nickerson, D.P. and Hunter, P.J. An Overview of CellML 1.1, a Biological Model Description Language. SIMULATION: Transactions of The Society for Modeling and Simulation International. 2003 Dec;79(12):740-747. See also http://www.cellml.org.

Jianlin Cheng, Lucas Scharenbroich, Pierre Baldi, Eric Mjolsness, "Sigmoid: Towards an Intelligent, Scalable, Software Infrastructure for Pathway Bioinformatics and Systems Biology", IEEE Intelligent Systems, May/June 2005. See also http://www.sigmoid.org.

Casanova, H., et al., Distributing MCell simulations on the Grid. Intl. J. High Perf. Comp. Appl., 2001. 15: p. 243-257.