Tutorial for BIOLOGO , A Domain-Specific Language For Morphogenesis

`http://www.simtk.org/projects/compucell3d`

Author:

Trevor Cickovski, University of Notre Dame, *tcickovs@nd.edu*
Jesús A. Izaguirre, University of Notre Dame, *izaguirr@cse.nd.edu*

September 22, 2006

We present a user guide for BioLogo , a domain-specific language for morphogenesis. BioLogo is based on a mathematical model of morphogenesis known as the Cellular Potts Model (CPM) [8], which models tissue and organ level cell pattern formation that occurs during morphogenesis. Working at the cellular level, this model is able to simulate cell adhesion, growth, division and death, as well as chemotactic and haptotactic responses to chemical gradients and cell differentiation into various types with specific properties that determine behavior. External chemical gradients can generally be established by solvers for partial differential equation (PDE) sets, such as Schnakenberg [12], Fitzhugh-Nagumo [14] or Hentschel-Glimm [9], to model reaction-diffusion. Or, chemicals can be secreted by cells at specific rates in response to some event.

BioLogo is intended to provide a higher level of abstraction for computational modeling of morphogenesis using the CPM coupled with chemical gradients. The language is XML-based, with its compiler an extension of an XML parser provided by the Xerces [2] libraries. Through BioLogo the user can specify CPM energy functions, chemical fields, and a model for cell differentiation. The BioLogo program, after compilation, passes through a code generator which generates dynamically loaded plugin extensions for the framework CompuCell3D, which is a three-dimensional framework for morphogenesis simulation. We have witnessed success using BioLogo and CompuCell3D in modeling cell sorting [3, 7], avian limb bud formation [13, 4], and *in vitro* capillary development [11].

Chapter 1

Introduction

Modeling morphogenesis involves several challenges. Finding a biologically significant model is first. The CPM provides a mathematical model that can accuractely model the patterning instabilities that occur during morphogenesis by operating at a cell-centered level of modeling. The CPM is grid-based, modeling cell shape changes and clustering in three-dimensional space. COMPUCELL3D [10, 4] employs the CPM along with PDE equation solvers that establish exterior chemical gradients, and models morphogenesis on a three-dimensional lattice. COMPUCELL3D addresses several challenges of morphogenesis modeling from a software perspective, implementing techniques for reducing speed and memory consumption, improving software maintenance and improving scalability for larger grid sizes.

The extensibility of COMPUCELL3D is strong for a programmer through its use of dynamically loaded plugin objects, which use an extension of the Proxy design pattern [6] and encapsulate functionality that is not a part of the core CPM model and should be included or excluded depending on the user. Through the use of keywords and references in a configuration file, the user can add or remove plugins from a particular simulation by adding or removing their reference in a COMPUCELL3D configuration file. A plugin object is then only allocated if referenced, using the Factory design pattern [1]. All plugin code is contained within a specific location, in a directory corresponding to the name of the plugin. Plugin additions can correspond to energy Hamiltonians, Cell Type Maps (which model cell differentiation), and exterior chemical fields, and typically do not require more than about five intermediate-level C++ classes. The addition of new features to COMPUCELL3D for a programmer is therefore not difficult. But for a non-programmer, COMPUCELL3D is rendered non-extensible by lack of knowledge of design patterns and C++ implementation of the necessary biological processes. BIOLOGO addresses this specific issue. By operating at a higher level of modeling and employing a syntax understandable to morphogenesis researchers and automatically generating appropriate plugins and compilation scripts for COMPUCELL3D, BIOLOGO provides a more convenient method of extending COMPUCELL3D for a non-programmer.

1.1 Useful Features

1.1.1 Cell Type Maps

Cell *types* provide a way of grouping cells with broadly similar behaviors (broad in a sense that each cell as a whole is different but may behave similarly) into the same category. During morphogenesis, cells *differentiate* into one of these known cell types which in turn describes its behavior. A Cell Type Map provides a method of modeling differentiation by associating a specific cell type as an attribute of every cell. In addition, even two cells of the same type may exist in different states, which is determined by a set of parameter values which also impacts behavior. Identical behavior in the same external environment is exhibited by two cells of the same type that are in the same state. State controls behavior at a finer granularity than type.

A Cell Type Map requires the following:

1. A set of parameter names and data types which make up the cell's *state*.
2. A method for initializing the cell state for each cell type.
3. A method for updating the cell state for each cell type.
4. A method for changing the cell type. In computer science terms, this is essentially a cell type automaton. There is a list of cell types, an initial cell type, and a list of rules for transitioning between cell types.

BIOLOGO provides the ability to specify one Cell Type Map for each unique simulation. By representing the Cell Type Map as a structured model with specific modules for the four above requirements, the user is saved the difficulty of encoding these requirements in C++ and interfacing them to COMPUCELL3D.

1.1.2 Chemical Fields

An arbitrary number of chemical fields can be superimposed on the CPM grid. These chemical fields may exist naturally in the external environment, or can be secreted by cells. In the former case, these chemicals are often modeled by partial differentiatial equation that simulate Reaction-Diffusion, following the idea of Turing [15] who introduced the idea that reacting and diffusing chemicals could form instabilities that could be modeled by a PDE RD approach, providing the basis for biological patterning. BIOLOGO provides statements for declaring all superimposed chemical fields, and initializing them with binary file input if they should be populated by PDE solvers, or just require a user-defined initial gradient. Declared fields can subsequently be referenced as three-dimensional arrays within BIOLOGO arithmetic and boolean expressions, using the bracket `[ ]` operators.

BIOLOGO can also be used to generate PDE solvers for population of chemical field concentrations. XML code can generate a solver which implements the finite difference method, or for more power embedded Python is supported. FiPy (`http:// www.ctcms.nist.gov`) offers a solid set of PDE solving libraries in Python.

1.1.3 Hamiltonians

The CPM follows the principle of energy minimization, which dictates that the system as a whole should tend towards a state of lower energy. A requirement of the CPM is to compute a change in energy ΔE that is incurred as a result of some change in the system, and accept this change with a specific probability that is inversely proportional to this energy change.

Each individual CPM Hamiltonian is a contributor to this calculation of ΔE. Through BIOLOGO the user can specify a method that returns a double-precision value, which implements the change in energy for this particular Hamiltonian. Within the method for calculating energy change, the user can straightforwardly reference cell attributes from the Cell Type Map such as cell type and also state variables, along with some predefined attributes that each cell is assumed to possess (examples are volume and surface area).

The user can choose to have their Hamiltonians be customizable. Each individual Hamiltonian will generate its own respective plugin for COMPUCELL3D. When plugins are referenced in a COMPUCELL3D configuration file, values for a predefined set of inputs can be provided which impact the behavior of the plugin. Through BIOLOGO the user can specify what the names of these input variables should be, and then correspondingly reference them in the method for calculating energy change. In this manner the user controls what aspects of the Hamiltonian should be customized and the role that each play in the CPM energy calculations.

Chapter 2

Syntax

2.1 Getting Started

Since BIOLOGO is an extension of the Extensible Markup Language (XML), a BIOLOGO file is actually an XML file, but with an extended syntax which we now describe. A BIOLOGO file always opens with the tag `<CompuCell3D>` and ends with the tag `</ CompuCell3D>`.

2.2 Cell Type Map

The template for a BIOLOGO Cell Type Map is shown in Program 1. A Cell Type Map is declared as a `cellmodel` and given a *modelname*. This *modelname* corresponds to the name of the resulting generated plugin for COMPUCELL3D.

The Cell Type Map opens with a section for declaration of state variables, which will in turn be attributes associated with each cell in the simulation. These are declared using BIOLOGO `declare` statement, described in section 2.5.

State variable declarations are followed by multiple declarations of cell types. Each cell type is given a type name. Within each cell type module, three modules are specified: (1) **creation**, which contains BIOLOGO statements that should be executed upon creation of a cell of this type, (2) **updatevaribles**, which contains BIOLOGO statements to execute to update state variables whenever this cell is selected for flipping, and (3) **updatecelltypes**, which contains the conditions that must pass for a cell to become this type. **updatecelltypes** thus implements the rules of the cell type automaton. The first specified cell type is assumed to be the initial cell type, unless a COMPUCELL3D PIF is used to initialize cell distributions (see COMPUCELL3D documentation; this input file allows for user-specified initial locations and cell types) The following events occur in sequence when a cell of a specific type is selected:

1. Attempt a type change. Sequentially execute the **updatecelltypes** modules of each cell type. When one passes, make the switch. If none pass, keep the same type.

2. If a type change was made, change the cell to be the new type.

3. Execute the **updatevariables** module of the current cell type.

BIOLOGO statements are all described in section 2.5.

```
<cellmodel  name="modelname">

    ..... declarations of state variables .....

    <celltype  name="type1">
        <creation>
            ... statements to execute upon creation of a cell of this type ...
        </creation>
        <updatevariables>
            ... statements to execute to update state variables ...
        </updatevariables>
        <updatecelltypes>
            ... condition(s) to pass for a cell to become type1 ...
        </updatecelltypes>
    </celltype>

    <celltype  name="type2">
        <creation>
            ... statements to execute upon creation of a cell of this type ...
        </creation>
        <updatevariables>
            ... statements to execute to update state variables ...
        </updatevariables>
        <updatecelltypes>
            ... condition(s) to pass for a cell to become type2 ...
        </updatecelltypes>
    </celltype>

    ... repeat for any other cell types ...

</cellmodel>
```
Program 1: Template for a BIOLOGO Cell Type Map.

2.3 Chemical Fields

Superimposed chemical fields in BioLogo are associated with specific Hamiltonians (described in Section 2.4). One of several methods can be used to superimpose a field. A field can be populated from binary input file(s), statically (one input file) or dynamically (multiple input files, read at specific frequencies). Or, a field can evolve itself under specific instructions given within the Hamiltonian. Finally, a field can follow the approach of Turing [15], governed by a set of reaction-diffusion partial differential equations (PDEs).

2.3.1 Field Type 1: Static, From File

File-populated fields are specified using a BioLogo `<Input>` tag. To declare a static field that should be initialized through a file, the following template is used:

<**Input** `name=`*"inputname"* `type=`*"file"* `/>`
<**Field** `name=`*"fieldname"* `fieldtype=`*"datatype"* `filename=`*"inputname"* `/>`

When a Hamiltonian is declared, a plugin is generated for CompuCell3D and is then referenced accordingly in the CompuCell3D configuration file. This `Input` tag allows the user to specify the name of the file used to populate this chemical field. The user input for the file name is given by *inputname* in the CompuCell3D configuration file, and the dimensions by a default input **FieldDim**. The *fieldname* attribute gives the name of the chemical field which can in turn be referenced in all Hamiltonians and the Cell Type Map using standard 3D array bracket (`[ ]`) accessor operators. *fieldtype* specifies the type of data contained by this field, which since this field is file-populated can be any BioLogo data type (see Section 2.5) with the exception of *pixel* or *cell*. The *filename* attribute hooks the two, attaching the file *inputname* with the field *fieldname*. For some Hamiltonian *H*, the above could be referenced in the CompuCell3D configuration file as follows:

<**Plugin** `name=`*"H"*
<**inputname**>*chemical.dat*</**inputname**>
<**FieldDim** *x="71" y="36" z="211"*></**FieldDim**>
</**Plugin**

Subsequently, *chemical.dat* will populate *fieldname* as a $71x36x211$ field of floating point values. The file is assumed to contain values on individual lines, with *z* as the innermost loop, *x* as outermost.

2.3.2 Field Type 2: Dynamic, From File

A dynamic file-populated field uses a similar BioLogo tag, but instead of specifying *inputname* as type *file*, *inputname* should be type *fileprefix*:

<**Input** `name=`*"inputname"* `type=`*"fileprefix"* `/>`
<**Field** `fieldname=`*"fieldname"* `type=`*"datatype"* `filename=`*"inputname"*`/>`

By specifying the input as a *fileprefix*, upon plugin reference in the CompuCell3D configuration file *inputnamefreq* will be a default parameter for the read frequency. Subsequently, at each appropriate step, if it is time to repopulate the chemical field based on the user-specified frequency, the file *inputname + step # + .dat* is read. In the COMPUCELL3D configuration file, for the same Hamiltonian *H*:

```
<Plugin name="H">
<inputname>chemical</inputname>
<inputnamefreq>20</inputnamefreq>
<FieldDim x="71" y="36" z="211"></FieldDim>
</Plugin>
```

Now every 20 steps, *fieldname* will be repopulated by the files *chemical0.dat, chemical20.dat, chemical40.dat, etc.* The field dimensions are still $71x36x211$.

2.3.3 Field Type 3: Secretion/Resorption

Other types of fields may be governed by simple secretion/resorption rules that should be executed at every CPM Monte Carlo step. The format of the BIOLOGO *secrete* tag is as follows:

```
<secrete field="fieldname" location="l" amount="a" condition="c" />
```

We are assuming *fieldname* to have been declared within some Hamiltonian *H* as containing numerical values (not pixels or cells). The above statement says that at every Monte Carlo step, if *c* (a boolean expression) is true, increase the value at location *l* (type **pixel**) by amount *a* (numerical value). The **resorb** tag operates in the same fashion, except location *l* of *fieldname* will decrease by *a*.

2.3.4 Field Type 4: Partial Differential Equations

Turing [15] modeled reaction and diffusion of chemical fields through a set of PDEs. One can superimpose multiple interdependent fields and evolve them through the same set of PDEs, for example in the Hentschel-Glimm [9] model there is an activator and inhibitor chemical.

BIOLOGO makes this possible with **Evolver**s. Currently, evolvers only work in two dimensions (zero *y-dimension*) but we are working on extending them to three. The syntax is as follows:

```
<Evolver name="evolvername" />
... inputs ...
... fields ...
<DiffEq fieldname="field1" />
<Term exp="exp1" condition="c1" />
<Term exp="exp2" condition="c2" />
... more terms ...
</DiffEq>
<DiffEq fieldname="field2" />
... terms ...
</DiffEq>
... more differential equations ...
</Evolver>
```

There is one **DiffEq** matched up with each field declared. The best way to illustrate this is by example. We can take the simple Schnakenberg equations [12]:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \gamma(a - u + u^2 v) + \Delta u, \\ \frac{\partial v}{\partial t} &= \gamma(b - u^2 v) + d\Delta v. \end{aligned} \tag{2.1}$$

We could add these equations to a simulation in BIOLOGO as follows:

```
<Evolver name=''Schnakenberg''>
<Input name=''gamma'' type=''float'' />
<Input name=''a'' type=''float'' />
<Input name=''b'' type=''float'' />
<Input name=''d'' type=''float'' />

<Field name=''u'' type=''float'' />
<Field name=''v'' type=''float'' />

<DiffEq fieldname=''u''>
<Term exp=''gamma*(a-u+u*u*v)'' condition=''true'' />
<Term exp=''delta(u)*u'' condition=''true'' />
</DiffEq>

<DiffEq fieldname=''v''>
<Term exp=''gamma*(b-u*u*v)'' condition=''true'' />
<Term exp=''d*delta(v)'' condition=''true'' />
</DiffEq>
</Evolver>
```

Program 2: Example Schnakenberg equation solver represented in BIOLOGO.

In this example, there are two fields declared, *u* and *v* and four input constants *gamma*, *a*, *b* and *d*. Note that all `condition` values are *true* in this case (the default as well), because all terms are included in the Schnakenberg equations no matter what. This will not always be the case however, and so we provide the `condition` attribute which can conditionally add or remove terms from equations.

Embedded Python

A PDE solver can also include embedded Python modules between <**Python**> tags. This option offers a tradeoff for more power at the cost of some performance. We illustrate the use of FiPy libraries to implement the spinoff of the Gamba-Serini RD equations [5], used in [11] to model *in vitro* capillary formation:

$$\frac{\partial c}{\partial t} = \alpha\delta_{\sigma_x,0} - (1 - \delta_{\sigma_x,0})\epsilon c + D\nabla^2 c, \tag{2.2}$$

We can model this using standard BIOLOGO XML as in Program 3 or equivalently with embedded Python using Program 4.

2.4 Hamiltonians

A Hamiltonian is declared using BIOLOGO `Hamiltonian` tags. Each Hamiltonian is given a name, input variables, associated chemical fields, and finally a step function which calculates an energy change. This energy change calculation is performed at every CPM flip attempt to determine if the selected cell's proposed

```
<PDESolver name="GambaSerini" normalize="false">
  <Input name="alpha" type="float" />
  <Input name="epsilon" type="float" />
  <Input name="DiffConst" type="float" />
  <Field name="c" type="float" />
  <DiffEq fieldname="c">
    <Term exp="(1-Kronecker)*alpha - epsilon*c*Kronecker + DiffConst*Laplacian(c)" condition="true" />
  </DiffEq>
</PDESolver>
```

Program 3: Gamba-Serini spinoff in standard XML.

```
<PDESolver name="GambaSeriniSCRIPT" normalize="false">
  <Input name="alpha" type="float" />
  <Input name="epsilon" type="float" />
  <Input name="DiffConst" type="float" />
  <Field name="c" type="float" />
  <Python>
    diffterm = ImplicitDiffusionTerm(coeff = DiffConst)
    secretion = alpha*kronecker
    resorption = ImplicitSourceTerm(coeff = epsilon*(1-kronecker))
    eq = TransientTerm() == secretion - resorption + diffterm
    eq.solve(c, dt=dt)
  </Python>
</PDESolver>
```

Program 4: Example of embedded Python.

flip should occur. Predefined BIOLOGO variables occurring in Hamiltonians included: *pt* (type `pixel`, representing the CPM randomly selected pixel, *oldcell* (type `cell`, representing the CPM selected cell at point *pt*), *newcell* (type `cell`, representing the CPM candidate cell), *potts.cellfield* (field of type `pixel`, representing the central CPM lattice.

The Hamiltonian template is shown in Program 5.

Upon this reference in the BIOLOGO file, a plugin for COMPUCELL3D will be generated with the name *Hamiltonian name*. *input1* and *input2* are variables whose values should be specified by the user in the plugin reference in the COMPUCELL3D configuration file. Once declared using `Input` tags, *input1* and *input2* can be referenced in expressions within the energy change calculation (and also in Cell Type Maps if this plugin is specified as included). It is these inputs that enable customizability of each Hamiltonian.

The only part of *Hamiltonians* that has not been covered is the **Equation** module. This specifies how energy is calculated with this Hamiltonian. An equation can either be a **neighborsum**, **cellsum** or **pixelsum**. There formats are as follows:

1. <**neighborsum** `exp="`*arithmetic expression*`"` `limit="`*integer expression*`"` `condition="`*boolean expression*`"` `/>`

2. <**cellsum** `exp="`*arithmetic expression*`"` `condition="`*boolean expression*`"` `/>`

3. <**pixelsum** `exp="`*arithmetic expression*`"` `condition="`*boolean expression*`"` `/>`

```
<Hamiltonian  name="Hamiltonian name">
   <Input name="input1" type="datatype1" />
   <Input name="input2" type="datatype2" />
   ...  other input variables ...
   <Input name="file1" type="file" fieldname="field 1 name" fieldtype="data type of field 1" />
   ...  other input declarations ...
   <Field name="field 2 name" type="data type of field 2" />
   ...  other field declarations ...
   <Step>
      ...  Chemical field secretion and resorption commands ...
   </Step>
   <Equation>
      ...  Energy Hamiltonian equation ...
   </Equation>
</Hamiltonian>
```

Program 5: Template for a BIOLOGO Hamiltonian.

A **neighborsum** describes interactions between all neighboring cells *sigma* and *sigmaP* in the CPM lattice. For all *sigma* and *sigmaP* within a Euclidean distance of `limit`, `exp` will be added to the energy sum if `condition` is true. This tag also defines a variable *distance* which can be used in expressions, which is the distance between *sigma* and *sigmaP*. A **cellsum** is slightly simpler, just looping over all cells *sigma*. `exp` is added to the energy sum if `condition` is true. A **pixelsum** loops over all pixels *pt*.

As an example the CPM volume energy from Eq. **??** can be represented in BIOLOGO as a **cellsum**:

```
<Hamiltonian name="Volume">
<Input name="TargetVolume" type="int" />
<Input name="LambdaVolume" type="double" />
<Equation>
<cellsum exp="LambdaVolume*(sigma.volume - TargetVolume)*(sigma.volume - TargetVolume)"
        condition="true" />
</Equation>
</Hamiltonian>
```

2.5 BIOLOGO Statements

BIOLOGO possess the ability to declare variables, copy values, and can implement conditional blocks and both conditional and counting loops. BIOLOGO also allows the passing of arithmetic and boolean expressions in infix notation using XML tag attributes and the scripting of some C++ functions within expression attributes. Through these capabilities BIOLOGO can implement functionality common to many other high level languages, and through other unique capabilities, can abstract more complex functionality such as cell type changes, pixel neighbor loops, and contributions to the CPM ΔE.

2.5.1 Basic Functionality

Arithmetic and Boolean Expressions

BIOLOGO expressions use infix notation. They can use the following symbols: `+`, `-`, `*`, `/`, `%`, and `()` for arithmetic operators; `greater`, `less`, `greaterequal`, `lessequal`, `and`, `or`, `equal`, and `notequal` for boolean operators. Single quotes (') encompass characters and strings. In all other ways, standard C++ notation can be used for expressions, allowing reference of C++ predefined functions (for example, `drand48()`). Examples of valid BIOLOGO arithmetic expressions:

- `5 + 4 * 3`
- `(7 % 5) / 2`
- `drand48() - 1`

Examples of valid BIOLOGO boolean expressions. These will always evaluate to one of two quantities, `true` or `false`. For example, in these cases, the first two evaluate to `true` while the last is `false`:

- `((5 equal 5) and (7 greater 2))`
- `10 notequal 8`
- `((10 less 8) or (7 lessequal 2))`

Variable Declarations

BIOLOGO variables can be declared to be one of several BIOLOGO types and can subsequently be used to store a value of this type. Variables can then be referenced in BIOLOGO arithmetic and boolean expressions, and their current value will be substituted upon expression evaluation. A variable is declared using the BIOLOGO **declare** tag, of the following format for an integer:

<**declare** ><**int** `name="`*variable name*`"` `value="`*infix integer expression*`"/>`</**declare**>

The *value* attribute is completely optional but can contain any valid BIOLOGO expression of type integer. Expressions can contain combinations of variables and constants. BIOLOGO data types include the following:

1. **int**: An integer.
2. **float**: A floating-point value.
3. **double**: A double-precision value.
4. **char**: A character.
5. **string**: A string of characters.
6. **pixel**: A data object representing a point. This type of object contains data members `x`, `y` and `z` which contain integer coordinate values. These data members can be referenced in expressions using the **.** operator, with the form *variablename.x* for example, similar to C++ or Java.

7. **cell**: A data object represented a simulated cell. This type of object by default contains data members `type`, `volume`, `surfacearea`, `targetvolume` and `targetsurface`. This object would also contain any cell state variables specified by the user in the Cell Type Map.

Copying Values

The value of a valid BIOLOGO expression can be copied into a defined variable of a compatible type using a BIOLOGO `copy` tag:

<**copy** name="*variable name*" value="*expression*" />

The types do not necessarily have to correspond exactly, for example, `int`, `float` and `double` values can be copied amongst each other, with appropriate truncation occurring by the same rules as C++ or Java. Here are examples of valid BIOLOGO copy statements, assuming *x* and *y* are declared integer variables:

- <**copy** name="*x*" value="*3*" />
- <**copy** name="*x*" value="*(3 % y)+5*" />
- <**copy** name="*x*" value="*x*y*" />

Conditional Blocks

BIOLOGO statements that are contained within conditional blocks should be executed only if some specific condition passes. This is implemented by a `if-elseif-else` set of modules, which are analagous to conditional blocks in higher level languages. The template for a BIOLOGO conditional block is as follows:

```
<if condition="boolean expression 1">
   ....  BioLogo statements ....
</if>

<elseif condition="boolean expression 2">
   ....  BioLogo statements ....
</elseif>

<elseif condition="boolean expression 3">
   ....  BioLogo statements ....
</elseif>

....  any other elseif modules ....

<else>
   ....  BioLogo statements ....
</else>
```

Boolean expression conditions within conditional blocks are checked sequentially until one passes. The BIOLOGO statements within the module of the first passed condition are executed. If no conditions pass, the **else** module is executed. All **elseif** and **else** modules are optional.

Loops

Loops provide the ability to execute the same sets of BIOLOGO statements multiple times without rewriting them. There are two different types of BIOLOGO loops: conditional and counting. Conditional loops execute contained BIOLOGO statements while some boolean condition is true, and stop the moment it becomes false. Counting loops execute a specific number of times, defining a loop counter variable (LCV) which is initialized to a specific value, and is incremented or decrement by a specific amount at each loop iteration until it hits another specific value and the loop stops.

Conditional loops can be either `do` or `while`. A `do` loop executes until the passed condition is true, and the `while` loop executes until the passed condition is false. Each has the same template:

```
<do condition="boolean expression">
   .... BioLogo statements ....
</do>
```

```
<while condition="boolean expression">
   .... BioLogo statements ....
</while>
```

A counting loop template, using the BIOLOGO <**for**> tag, is shown below:

```
<for variable="defined variable" from="arithmetic expression" to="arithmetic expression"
      step="arithmetic epxression">
   .... BioLogo statements ....
</for>
```

Upon executing of this loop, the passed *variable* will be initialized to the evaluation of the expression in the *from* attribute. Each iteration of the loop will increment the *variable* by the expression in the *step* attribute. Execution of the loop will stop when *variable* exceeds the value in the *to* expression. Note that this loop assumes that *variable* will increase at each iteration. It is also possible to allow *variable* to decrease by the expression in the *step* attribute, and that is performed by using an attribute *downto* in place of *to*.

2.5.2 Unique Capabilities

BIOLOGO also provides some abstractions for some common tasks in the CPM model. These statements are less general than the basic statements, and typically have specific places within the BIOLOGO file where they are valid. The three examples are: changing cell type, pixel neighbor loops and CPM energy change contributions. Obviously changing cell type is only applicable within the `updatecelltypes` module of Cell Type Maps. Similarly, the energy change contribution can only be applied within Hamiltonina `Step` modules. Pixel neighbor loops can be used in Cell Type Maps or Hamiltonians, but generally are more useful within Hamiltonians.

Changing Cell Type

In a Cell Type Map, all cells are governed by sets of rules for changing cell type. A change of cell type is performed using the BIOLOGO <**changeif**> tag. Its template is as follows:

<**changeif** `currenttype="`*defined cell type*`"` `condition="`*boolean expression*`"` `/>`

These **updatecelltypes** module which contains this statement will itself be contained within a BIOLOGO **celltype** declaration. If the passed condition is true, and the cell's current type is the value passed into *currenttype*, the cell becomes the enclosed celltype. When a cell is selected by the CPM, all **updatecelltypes** moduels are executed sequentially. When the first **changeif** statement results in a change of cell types, no more changes are attempted. If no successful **changeif** statements are encountered, the **updatevariables** module of the current **celltype** is executed.

Pixel Neighbor Loops

For convenience we have abstracted the ability to, given a pixel in the lattice, to loop over all its neighbors within a specific Euclidean distance and perform some operation on each neighbor. This would be useful, for example, in the definition of an adhesion-related Hamiltonian which must determine the level of adhesivity between a lattice location and all neighboring points. A neighbor loop is formed using the BIOLOGO <**forneighbors**> tag. This tag accepts as attributes a loop counter `variable` which this time is of type **pixel** and holds the current neighbor. Attribute `point` contains the pixel of reference, `grid` is a field of pixels (typically in a simulation there will only be one, the central CPM lattice), `depth` is a Euclidean distance limit for neighbors, and the user can provide a variable in the `distance` attribute to hold the current distance between point `variable` and point `point`. Each of these variables can subsequently be referenced within the body of the neighbors loop.

<**forneighbors** `variable="`*declared variable of type pixel*`"` `point="`*declared variable of type pixel*`"` `grid="`*declared field of type pixel*`"` `distance="`*declared variable of type double*`"` `depth="`*arithmetic expression of type double*`"` `/>`

As a simple example, suppose we have a two-dimensional 3x3 `grid` (we can extend the neighbor loop to three dimensions using the same principles). Further suppose our `point` of reference is (1,1), and we set `depth` to 1.5. This loop would execute 8 times, with the following values at each iteration:

- **Iteration 1:** `variable` = (0,1), `distance` = 1
- **Iteration 2:** `variable` = (1,0), `distance` = 1
- **Iteration 3:** `variable` = (1,2), `distance` = 1
- **Iteration 4:** `variable` = (2,1), `distance` = 1
- **Iteration 5:** `variable` = (0,0), `distance` = 1.414
- **Iteration 6:** `variable` = (0,2), `distance` = 1.414
- **Iteration 7:** `variable` = (2,0), `distance` = 1.414

- **Iteration 8:** `variable` = (2,2), `distance` = 1.414

When another neighbor is considered, its `distance` will be beyond the `depth` of 1.5, and the loop execution will stop.

Chapter 3

Installation and Usage

BIOLOGO is a part of the COMPUCELL3D tarballs. To download BIOLOGO , download the appropriate COMPUCELL3D tarball for your machine. Linux and MacOS versions are available. To view the results of extended COMPUCELL3D simulations, Qt is required. Linux tarballs include versions which support Qt 3 (installed on most RedHat machines) and Qt 4 (the most recent version).

An initial unzipping and untarring of the `.tar.gz` file will produce a folder. Change to that folder and you should find a `BioLogo/` directory. Now change to the directory `BioLogo/BioLogo` and you will find a script `install.sh`. Run this script and the BioLogo framework will compile.

The same folder will contain some example BIOLOGO XML files and an execution script `BIOLOGO.sh`. To run BIOLOGO :

```
./BIOLOGO.sh <BIOLOGO File>
```

This will produce the C++ extensions for COMPUCELL3D, which can subsequently be compiled and run with the rest of the framework.

Chapter 4

Concluding Remarks

We thank you for using our product and wish you the best in your endeavors. Please submit all questions, bug fixes, etc. to *tcickovs@nd.edu*.

Bibliography

[1] A. Alexandrescu. *Modern C++ design: Generic programming and design patterns applied.* Addison-Wesley, Reading, Massachusetts, 2001.

[2] Apache. Xerces C++ parser. `http://xml.apache.org/xerces-c/index.html`, 2003.

[3] D. A. Beysens, G. Forgacs, and J. A. Glazier. Cell sorting is analogous to phase ordering in fluids. *Proc. Natl. Acad. Sci. USA*, 97:9467–9471, 2000.

[4] T. Cickovski, C. Huang, R. Chaturvedi, T. Glimm, H. G. E. Hentschel, M. S. Alber, J. A. Glazier, S. A. Newman, and J. A. Izaguirre. A framework for three-dimensional simulation of morphogenesis. *IEEE/ACM Trans. on Comp. Bio. and Bioinformatics*, 2(4): 273–288, 2005.

[5] A. Gamba, D. Ambrosi, A. Coniglio, A. DeCandia, S. DiTalia, E. Giraudo, G. Serini, L. Preziosi, and F. Bussolino. Percolation morphogenesis and Burgers dynamics in blood vessels formation. *Phys. Rev. Lett.*, 90(11): 118101, 2003.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, Massachusetts, 1995.

[7] J. A. Glazier and F. Graner. Simulation of the differential adhesion driven rearrangement of biological cells. *Phys. Rev. E*, 47(3): 2128–2154, 1993.

[8] F. Graner and J. A. Glazier. Simulation of biological cell sorting using a two-dimensional extended potts model. *Phys. Rev. Lett.*, 69(13): 2013–2016, 1992.

[9] H. G. E. Hentschel, T. Glimm, J. A. Glazier, and S. A. Newman. Dynamical mechanisms for skeletal pattern formation in the vertebrate limb. *Proc. R. Soc. Lond. B Biol. Sci.*, 271(1549): 1713–1722, 2004.

[10] J. A. Izaguirre, R. Chaturvedi, C. Huang, T. Cickovski, J. Coffland, G. Thomas, G. Forgacs, M. S. Alber, G. Hentschel, S. A. Newman, and J. A. Glazier. CompuCell, a multimodel framework for simulation of morphogenesis. *Bioinformatics*, 20:1129–1137, 2004.

[11] Roeland M. H. Merks, Sergey V. Brodsky, Michael S. Goligorsky, Stuart A. Newman, and James A. Glazier. Cell elongation is key to *in silico* replication of *in vitro* vasculogenesis and subsequent remodeling. *Dev. Biol.*, 289(1): 44–54, 2006.

[12] J. D. Murray. Mathematical biology, second corrected edition. In *Biomathematics*, volume 19, pages 156,376,406,472,739, Berlin, Heidelberg, 1993. Springer-Verlag.

[13] S. A. Newman and H. L. Frisch. Dynamics of skeletal pattern formation in developing chick limb. *Science*, 205(4407):662–668, 1979.

[14] Alexandre Panfilov and Pauline Hogeweg. Spiral breakup in a modified fitzhugh-nagumo model. *Phys. Lett. A*, 176:295–299, 1993.

[15] A. Turing. The chemical basis of morphogenesis. *Phil. Trans. Roy. Soc. London*, B 237:37–72, 1952.