A Formal Model of Universal Algorithmic Assembly and Molecular Computation

Bruce J. MacLennan, University of Tennessee - Knoxville, USA

ABSTRACT

In this paper, the author describes a systematic and general approach to nanostructure synthesis and control through autonomous molecular combinatory computing. Combinatory computing is based on simple network (graph) substitution operations, deriving from combinatory logic (Curry, Feys, & Craig, 1958), which are sufficient for any computation. When these operations are implemented by autonomous molecular processes, they provide a means for computing within supramolecular networks, which may be used to assemble these networks or control their behavior. Further, the Church-Rosser Theorem (Curry, Feys, & Craig, 1958) proves that substitutions may be performed in any order or in parallel without affecting the computational result; this is a very advantageous property for autonomous molecular computation. In addition to the theoretical foundations of molecular combinatory computing, the author discusses possible molecular implementations as well as accomplishments in the (simulated) synthesis of membranes, channels, nanotubes, and other nanostructures.

Keywords: *Algorithmic Assembly, Combinator, Combinatory Logic, Molecular Building Block, Molecular Computation, Nanofabrication, Nanotechnology, Nanotube, Self-Assembly*

INTRODUCTION

Our goal is a systematic and general approach to nanostructure synthesis and control through *molecular combinatory computing*. Combinatory computing is based on simple network (graph) substitution operations, deriving from combinatory logic (Curry, Feys, & Craig, 1958), which are sufficient for any computation; that is, these operations are *Turing-complete*. When these operations are implemented by molecular processes, they provide a means of computing within supramolecular networks, which may be used to assemble these networks or to control their behavior. Indeed, computer scientists have known for decades how to compile ordinary programs into combinator programs, and so this approach offers the prospect of compiling computer programs into molecular structures so that they may execute at the molecular level and with "molar" degrees of parallelism. Further, the *Church-Rosser Theorem* (Curry, Feys, & Craig, 1958, ch. 4) proves that substitutions may be performed in any order or in parallel without affecting the computational result; this is a very advantageous property for molecular computation. (More precisely, the theorem states that *if* you get a result, you always get the

DOI: 10.4018/jnmc.2010070104

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

same result. Some orders, however, may lead to nonterminating computations that produce *no* result. To date, we have found little need in molecular computing for such potentially nonterminating programs.)

COMBINATORY COMPUTING

Computational Primitives

Molecular combinatory programs are supramolecular structures in the form of binary trees. The interior nodes of the tree (which we call A nodes) represent the application of a function to its argument, the function and its argument being represented by the two daughters of the A node. The leaf nodes are molecular groups that function as primitive operations and data.

One of the remarkable theorems of combinatory logic is that two simple substitution operations are sufficient for implementing any program (Turing-computable function). One of these operations is called $\mathbf{K}$ and is described by the substitution rule:

$$((\mathbf{K}X)Y) \Rightarrow X. \qquad (1)$$

Here X and Y represent arbitrary binary trees and $\mathbf{K}$ represents a leaf of type $\mathbf{K}$; parentheses group the subtrees of an interior node (an A node). The interpretation of the rule is that wherever a subtree of the form $((\mathbf{K}X)Y)$ is found in the network, it may be replaced by X. This reaction is depicted in Figure 1, which also illustrates its similarity to a molecular substitution reaction. The effect of the operation is to delete Y from the network. The $\mathbf{K}$ group and Y subtree are released as waste products, which may be recycled in later reactions.

The second primitive operation is described by the rule:

$$(((\mathbf{S}X)Y)Z) \Rightarrow ((XZ)(YZ)). \qquad (2)$$

This rule may be interpreted in two ways, either as copying the subtree Z or as creating two links to a shared copy of Z (thus creating a graph that is not a tree). It can be proved that both interpretations produce the same computational result, but they have different effects when used for nanostructure assembly. For this reason we need both variants of the operation, which we denote $\mathbf{S}$ (replicating) and $\breve{\mathbf{S}}$ (sharing). The molecular implementations of the two are very similar (see Figure 2). If $\mathbf{C} = \mathbf{V}$ (a *sharing node*), then we have two links to a shared copy of Z:

$$(((\breve{\mathbf{S}}X)Y)Z) \Rightarrow ((XZ^{(1)})(YZ^{(0)})).$$

The structure created by this operator shares a single copy of Z; the notations $Z^{(1)}$ and $Z^{(0)}$ refer to two links to a “Y-connector” (called a $\mathbf{V}$ node), which links to the original copy of Z. (Subsequent computations may rearrange the locations of the two links.) The principal purpose of the $\breve{\mathbf{S}}$ operation is to synthesize non-tree-structured supramolecular networks.

On the other hand, if $\mathbf{C} = \mathbf{R}$ (a *replication node*), then other substitution reactions will begin the replication of Z, so that eventually the two links will go to two independent copies of Z:

$$(((\mathbf{S}X)Y)Z) \Rightarrow ((XZ)(YZ')).$$

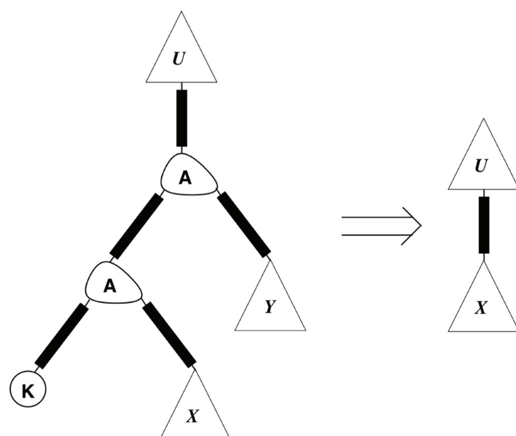
Here Z' refers to a new copy of the structure Z, which is created by the primitive replication ($\mathbf{R}$) operation. It can be shown that computation can proceed in parallel with replication without affecting the results of the process (a consequence of the Church-Rosser theorem) .

Although $\mathbf{K}$ and $\mathbf{S}$ are sufficient for all computation, they cannot (even with $\breve{\mathbf{S}}$) create cyclic structures, for which we use an additional primitive operation $\breve{\mathbf{Y}}$, which creates a self-referential link through a $\mathbf{V}$-node. It is defined (MacLennan, 2002c):

$$(\breve{\mathbf{Y}}F) \Rightarrow y^{(1)} \quad \text{where} \quad y \equiv \left(Fy^{(0)}\right). \qquad (3)$$

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 1. combinator substitution operation. U, X, and Y represent any networks (graphs)



The operation creates an elementary cycle, which may be expanded by computation in the combinator tree F (Figure 3). Examples of the use of both $\breve{\mathbf{S}}$ and $\breve{\mathbf{Y}}$ are given below.

Molecular Extensions

The primitive substitutions already mentioned ($\mathbf{K}$, $\mathbf{S}$, $\breve{\mathbf{S}}$, $\breve{\mathbf{Y}}$) are adequate for describing the assembly of static structures, but for dynamic applications we will need additional operations that can respond to environmental conditions ("sensors") or have noncomputational effects ("actuators"). Many of these will be ad hoc additions to the basic computational framework, but we are developing general interface conventions to facilitate the development of a systematic nanotechnology (MacLennan, 2003b). For example, we may design a molecular group $\mathbf{K}_{-\lambda}$ that is normally inert, but is recognized (and therefore operates) as $\mathbf{K}$ in the presence of an environmental condition λ (e.g., light of a particular wavelength or a particular chemical species). Such a sensor may be used to control conditional execution, such as the opening or closing of a channel in a membrane (see Dynamic Structures, below).

NANOSTRUCTURE SYNTHESIS AND CONTROL

Assembly of Static Structures

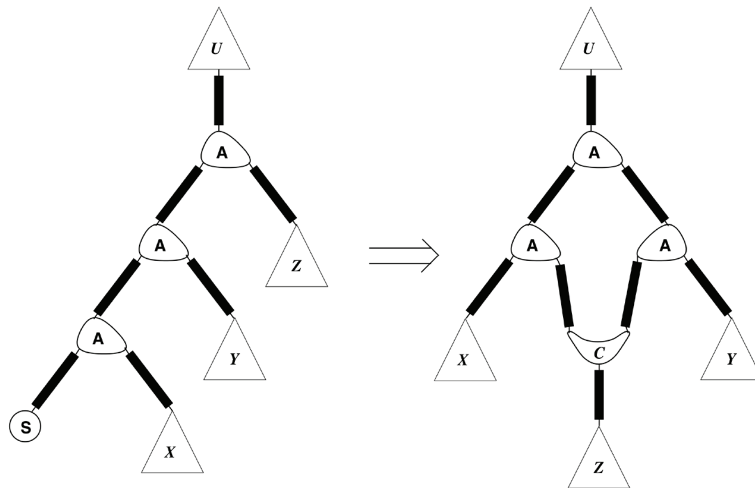
We have investigated the synthesis of a number of nanostructures by molecular combinatory computing. These include membranes and nanostructures of several different architectures. We have developed also systematic means to combine these into larger, heterogeneous structures, and to include active elements such as channels, sensors, and nano-actuators.

For a first example of nanostructure synthesis, we can consider a molecular combinatory program to assemble a membrane such as shown in Figure 4. This small example is produced by the program $\mathrm{xgrid}_{3,4}\,\mathbf{NNN}$ (where $\mathbf{N}$ is any inert group); in general, $\mathrm{xgrid}_{m,n}\,FYX$ computes an $m \times n$ membrane in which F, X, and Y are the terminal groups to be used on the left ends, right ends, and bottoms, respectively, of the chains. $\mathrm{xgrid}_{m,n}$ is defined:

$$\mathrm{xgrid}_{m,n} = \mathbf{B}(\mathbf{B}(\mathbf{Z}_{m-1}\mathbf{W}))(\mathbf{B}(\mathbf{Z}_{n-1}\mathbf{W})(\mathbf{Z}_m\breve{\Phi}_n)). \quad (4)$$

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 2. and $\breve{\mathbf{S}}$ combinator substitution operations. $\mathbf{C} = \mathbf{R}$ for $\mathbf{S}$ and $\mathbf{C} = \mathbf{V}$ for $\breve{\mathbf{S}}$. Note the reversed orientation of the rightmost A *group*



Unfortunately, space does not permit a complete explanation of this program, which may be found, with correctness proofs, in MacLennan (2002a). Equation 4 makes use of various standard combinators ($\mathbf{B}$, $\mathbf{W}$, etc.), which are defined in terms of $\mathbf{K}$, $\mathbf{S}$, or $\breve{\mathbf{S}}$; see the Appendix to this paper. The following comments may make Eq. 4 a little less opaque. One row of the membrane is created by

$$\breve{\Phi}_n F Y_1 \cdots Y_n X \Rightarrow F\left(Y_1 X^{(n-1)}\right)\cdots\left(Y_{n-1} X^{(1)}\right)\left(Y_n X^{(0)}\right),$$

where the parenthesized superscripts on X indicate that this group is shared by a chain of $n-1$ $\mathbf{V}$-nodes. The Y_j are the groups that will be linked to the bottoms of the (single-row) columns. This operation can be iterated m times to construct m linked rows, each with its own terminal group X_i:

$$\left(\breve{\Phi}_n\right)^m F Y_1 \cdots Y_n X_1 \cdots X_m$$
$$\Rightarrow F\left(Y_1 X_1^{(n-1)} \cdots X_m^{(n-1)}\right)\cdots\left(Y_n X_1^{(0)} \cdots X_m^{(0)}\right).$$

In the usual case we want all the terminal groups to be identical on the right ends, $X_i = X$, and on the bottoms, $Y_j = Y$. That is, to generate our membrane we want:

$$\breve{\Phi}_n^m F \underbrace{Y \cdots Y}_{n} \underbrace{X \cdots X}_{m}.$$

To accomplish this, we make use of the $\mathbf{W}$ combinator, which duplicates the argument of a function. To get m copies, we iterate $\mathbf{W}$ $m-1$ times using $\mathbf{W}^{n-1} = \mathbf{Z}_{m-1}\mathbf{W}$:

$$\mathbf{Z}_{m-1}\mathbf{W}\left(\breve{\Phi}_n^m F \underbrace{Y \cdots Y}_{n}\right) X \Rightarrow \breve{\Phi}_n^m F \underbrace{Y \cdots Y}_{n} \underbrace{X \cdots X}_{m}.$$

We make n copies of Y in the same way:

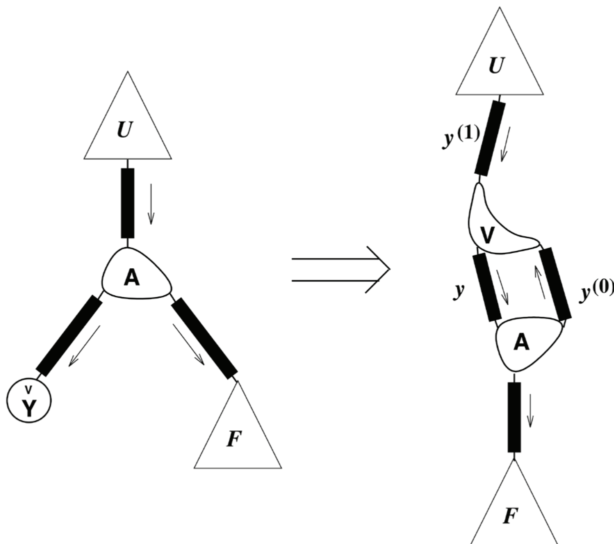
$$\mathbf{Z}_{m-1}\mathbf{W}\left(\mathbf{Z}_{n-1}\mathbf{W}\left(\breve{\Phi}_n^m F\right) Y\right) X \Rightarrow \mathbf{Z}_{m-1}\mathbf{W}\left(\breve{\Phi}_n^m F \underbrace{Y \cdots Y}_{n}\right) X.$$

Therefore, the definition of xgrid is given by:

$$\text{xgrid}_{m,n} F Y X = \mathbf{Z}_{m-1}\mathbf{W}\left(\mathbf{Z}_{n-1}\mathbf{W}\left(\breve{\Phi}_n^m F\right) Y\right) X.$$

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 3. combinator primitive substitution operation. Arrows indicate link direction; note resulting elementary cycle between A *- and* $\mathbf{V}$ *-groups.*



The additional $\mathbf{B}$ combinators in Eq. 4 are a result of extracting F, Y, and X out of the parentheses in the preceding equation in order to get a definition of the $\mathrm{xgrid}_{m,n}$ structure independent of these variables (MacLennan, 2002a).

When all the definitions in the Appendix are expanded, $\mathrm{xgrid}_{m,n}$ is found to be a binary tree of size $20m + 28n + 73$ primitive groups ($\mathbf{A}$, $\mathbf{K}$, $\mathbf{S}$, $\breve{\mathbf{S}}$) (MacLennan, 2002a). Therefore, the program for an $m \times n$ membrane is of size $O\left(m + n\right)$. This does not seem to be unreasonable, even for large membranes, but it can be decreased more if necessary. For example, if $m = 10^k$, then $\mathbf{Z}_m$ in Eq. 4 can be replaced by $\mathbf{Z}_k\mathbf{Z}_{10}$, reducing the size of this part of the program from $O\left(10^k\right)$ to $O\left(k\right)$ (MacLennan, 2002c). Similar compressions can be applied to the other parts dependent on m and n. Therefore, by these recoding techniques the size of the program can be successively reduced to $O\left(\log m + \log n\right)$, to $O\left(\log\log m + \log\log n\right)$, etc. Furthermore, as will be explained later, large membranes can be synthesized by iterative assembly of small patches.

The $\breve{\mathbf{Y}}$ (cycle forming) operation can be used to connect the lower and upper margins of the cross-linked membrane to generate a nanotube such as shown in Figures 5 and 6. This is created by $\mathrm{xtube}_{m,n}\,\mathbf{N}$, where

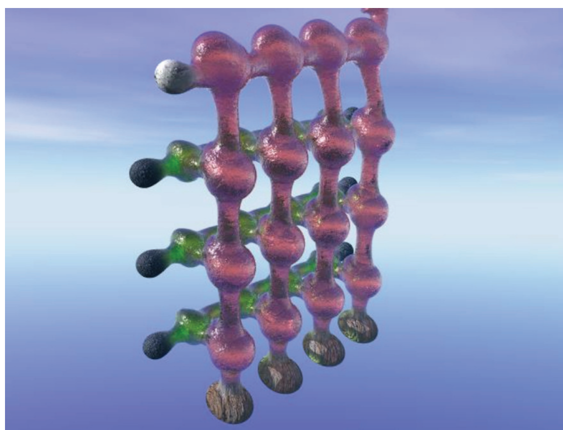
$$\mathrm{xtube}_{m,n} = \breve{\mathbf{W}}^{m-1}\left(\mathbf{W}^{n-1}\left(\breve{\Phi}_n^m\mathbf{N}\right)\left(\mathbf{B}^m\breve{\mathbf{Y}}\left(\mathbf{C}_{[m]}\mathbf{I}\right)\right)\right).$$

The program is explained in MacLennan (2002a); its size is $102m + 44n - 96$ primitive groups ($\mathbf{A}$, $\mathbf{K}$, $\mathbf{N}$, $\mathbf{S}$, $\breve{\mathbf{S}}$, $\breve{\mathbf{Y}}$).

These membranes and nanotubes have a *unit mesh*, that is, the dimensions of the basic cells are determined by the size of the primitive groups ($\mathbf{A}$, $\mathbf{V}$) and the links between them. It is relatively straight-forward to modify the preceding definitions to have larger mesh-dimensions (multiples of the cells). In addition, various pendant groups can be incorporated into the structure (MacLennan, 2003b).

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 4. Visualization of cross-linked membrane generated by $\mathrm{xgrid}_{3,4}\,\mathbf{NNN}$. *The vertical chains and the uppermost horizontal chain are composed of* $\mathbf{A}$ *nodes; the other horizontal chains are composed of* $\mathbf{V}$ *nodes.*



Finally, we consider $\mathrm{hgrid}_{m,n}\,X$, a program for assembling an $m \times n$ hexagonal grid such as shown in Figure 7, in which the upper parts of the hexagons are $\mathbf{A}$-nodes and the lower parts are $\mathbf{V}$-nodes. In brief (MacLennan, 2002a), a row of n $\mathbf{V}$-nodes is created by $\mathrm{vrow}_n = \breve{\mathbf{W}}_{[n]} \circ \mathbf{BB}^{[n-1]}$. A row of n $\mathbf{A}$-nodes is created by $\mathrm{arow}_n = \mathbf{B}\breve{\mathbf{W}}_{[n-1]} \circ \mathbf{B}^{[n]}$. A double row, comprising an $\mathbf{A}$ and a $\mathbf{V}$ row, is assembled by $\mathrm{drow}_n = \mathrm{vrow}_n \circ \mathrm{arow}_n$. The hexagonal membrane is then created by iterating drow_n m times: $\mathrm{drow}_n^m = \mathbf{Z}_m\,\mathrm{drow}_n$. The arguments of this function are the groups that occupy the lower fringe. Since $\mathrm{hgrid}_{m,n}\,X$ uses X for all of these groups, it has to duplicate it $n-1$ times. Therefore it is defined:

$$\mathrm{hgrid}_{m,n} = \mathbf{Z}_{n-1}\mathbf{W}\left(\mathbf{Z}_m\,\mathrm{drow}_n \mathrm{N}\right).$$

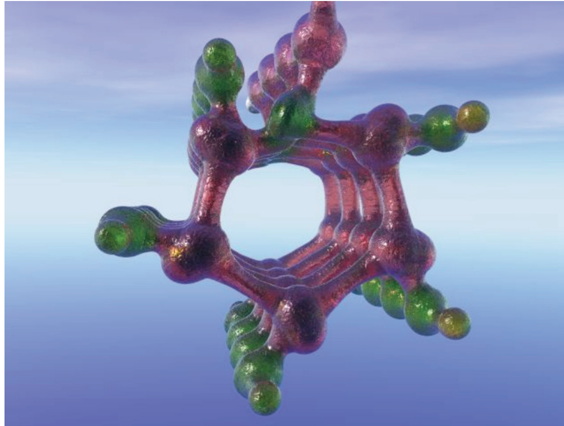
The size of this program is $10m + 142n - 57 = O\left(m+n\right)$ primitive groups. See MacLennan (2002a) for a complete derivation, correctness proofs, variations, and size analysis.

The preceding examples have shown how we can assemble membranes and nanotubes that are homogeneous in structure, but often it is required to generate heterogeneous structures. For example, we may want a membrane with pores or channels distributed through it is some regular way. To accomplish this we have developed a general rectangular "patch format" (MacLennan, 2003b).

Any such patch may be joined either horizontally or vertically with another patch of compatible dimensions, to yield a patch combining the two. For example, if P is an $m \times n$ patch and Q is an $m \times n'$ patch, then $\mathbf{B}^{n+1}QP$ is an $m \times \left(n + n'\right)$ patch with Q to the right of P. Similarly, if P is $m \times n$ and Q is $m' \times n$, then $P \circ \mathbf{B}^m Q$ is the $\left(m + m'\right) \times n$ patch with P below Q. (In combinatory logic, $X \circ Y = \mathbf{B}XY$.) In this way patches may be hierarchically assembled into larger patches. Furthermore, combinatory computing permits the patch assembly operations to be iterated, thus creating large membranes with complex hierarchical structures. This allows us to build upon a basic library of elementary membrane patches, pores, and other nanostructural units.

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 5. Visualization of end of small cross-linked nanotube generated by $\mathrm{xtube}_{5,4}\ \mathbf{N}$



Nanotubes can also be synthesized in patch format to allow end-to-end connection. If T is a patchable tube synthesizer of length m and U is one of length m', both of the same circumference n, then $U \circ T$ is a patch synthesizer that connects U to the right of T. This operation is easily iterated, for

$$T^k = \underbrace{T \circ T \circ \cdots \circ T}_{k}$$

is k replicates of T connected end-to-end (and thus of length km). This operation can also be expressed $\mathbf{Z}_k T$. If, as is commonly the case, the size of the synthesizer T is $O(m+n)$, then the size of $\mathbf{Z}_k T$ is $O(k+m+n)$.

Dynamic Structures

Rather than computing to a stable state, *dynamic structures* remain potentially active, ready to respond to environmental conditions (MacLennan, 2003b). Unfortunately, space does not permit a detailed discussion of the synthesis of membranes with pores and channels; the following brief remarks must suffice.

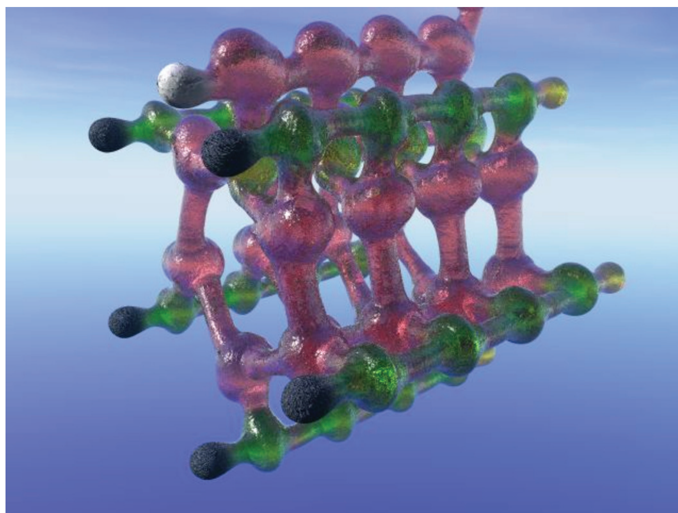
A rectangular *pore* is simply a patch in which the interior is an open space. These pores can be combined with other patches to create membranes with pores of a given size and distribution (all in terms of the fundamental units, of course). Pores can be included in the surfaces of nanotubes as well.

An example of an active structure is a channel in a membrane, which may open or close in response to a change in the environment. This is most simply accomplished by synthesizing a molecular group, which we denote $\mathbf{K}_{-\lambda}$, that responds to condition λ by reconfiguring into a $\mathbf{K}$ combinator. The simplest channels are "one-shot," that is, once opened they remain open, or once closed, remain closed. A one-shot channel that closes when triggered can be implemented by using a sensor molecule to trigger the assembly of a membrane patch covering a pore. A channel that opens works similarly, discarding the covering membrane patch. Resettable channels, which can be opened and closed any number of times, are more complicated, since they need a supply of sensor molecules that are protected from being triggered before they are used.

Nano-actuators can, of course, be designed as ad hoc extensions to the combinatory framework, but we are also investigating purely computational implementations of actuators. For example, under program control, we can synthesize a chain of molecular units; similarly under program control, we can collapse such a chain into a single unit. By using such processes in complementary pairs (like opposing

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 6. Visualization of side of small cross-linked nanotube generated by $\mathtt{xtube}_{5,4}\ \mathbf{N}$



muscle groups) we have computational control of physical motion. The force exerted by each such nano-actuator depends on the linking bond strength (perhaps 50 kJ/mol; see Networks, below), but they can work cooperatively to generate larger forces.

Molecular combinatory computing is not limited to nanostructure synthesis and control, but may be applied to more conventional computational problems. Suppose we want to attack an NP problem with *molar parallelism* (that is, with a degree of parallelism on the order of 10^{23}). Further suppose we have a polynomial-time program p to test the correctness of a potential solution x . As previously remarked, the program p can be compiled into a molecular combinator tree P ; similarly a potential solution x can be encoded as a combinator tree X . Then the tree (PX) will evaluate the potential solution, reducing to the molecular combinator representation of true or false (usually $\mathbf{K}$ and $(\mathbf{SK})$, respectively) (MacLennan, 2002c). Therefore, by producing enough replicates of P and a sufficient variety of potential solutions X_k , we may evaluate the potential solutions with molar parallelism.

MOLECULAR IMPLEMENTATION

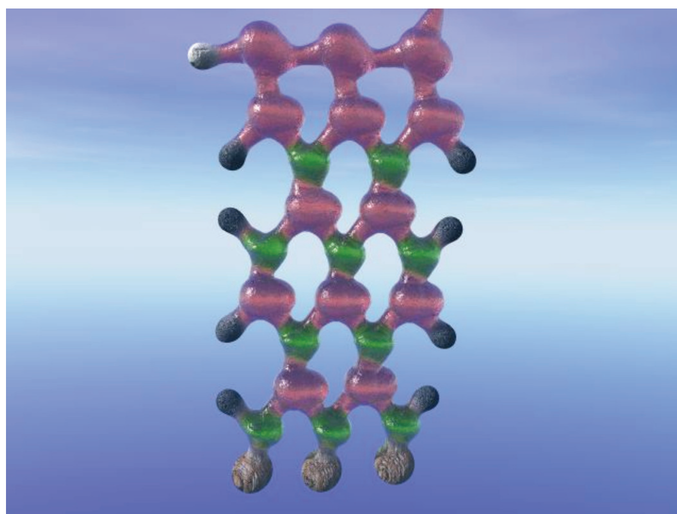
Of course, all the advantages of molecular combinatory computing are illusory unless a molecular implementation of the combinatory operations can be discovered or developed. Therefore we have begun investigating one feasible molecular implementation. The two principal problems are: (1) How are the combinator networks represented molecularly? (2) How are the substitution operations implemented molecularly?

Networks

Combinatory computing proceeds by making substitutions in networks of interconnected nodes. These networks constitute both the medium in which computation takes place and the nanostructure created by the computational process. Therefore it is necessary to consider the molecular implementation of these networks as well as the processes by which they may be transformed according to the rules of combinatory computing.

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 7. Visualization of a small part of a hexagonal grid produced by $\mathrm{hgrid}_{2,3}\,\mathbf{N}$. *The upper parts of the hexagons are* $\mathbf{A}$ *-nodes and the lower parts are* $\mathbf{V}$ *-nodes*



The first requirement is that nodes and linking groups need to be stable in themselves, but the interconnections between them need to be sufficiently labile to permit the substitutions. Second, the node types ($\mathbf{A}$, $\mathbf{K}$, $\mathbf{S}$, etc.) need to identifiable, so that the correct substitutions take place. In addition, for more secure matching of structures, the link ($\mathbf{L}$) groups should be identifiable. Further, it is necessary to be able to distinguish the various binding sites on a node. For example, an $\mathbf{A}$ node has three distinct sites: the result site, an operator argument, and an operand argument (MacLennan, 2002c).

Our current approach is to implement the networks as hydrogen-bonded covalently-structured molecular building blocks (MBBs). Hydrogen bonds are used because they are labile in an aqueous environment, balancing reasonable stability with the flexibility required for continual structural reorganization. Hydrogen bonding as a means of connecting and identifying covalently-bonded subunits is the basis, of course, for recognition and ligation in DNA and related molecules. Further, large, tree-like structures called *dendrimers* have been assembled from hydrogen-bonded MBBs (Simard, Su, & Wuest, 1991; Zimmerman, Zeng, Reichert, & Kolotuchin, 1996). Nevertheless, hydrogen bonds are not very stable in aqueous solution, so there may be a delicate balance between stability and lability. Covalently-structured MBBs are used because they provide a comparatively rigid framework in which to embed hydrogen bonding sites, and because there is an extensive synthetic precedent for engineering molecules with the required shape and with appropriately located hydrogen bonding sites (Allis & Spencer, 2003).

It is necessary to be able to distinguish the "head" and "tail" ends (MacLennan, 2002c) of the $\mathbf{L}$ groups (i.e., our graph edges are directed), and we estimate that two or three H-bonds are required to do this securely. (For comparison, thymine and adenine have two H-bonds, cytosine and guanine have three.) Therefore, if we take 20 kJ/mol as the strength of a typical H-bond (the strengths of which vary from 2 to 40 kJ/mol), then the total connection strength of a link will be about 50 kJ/mol.

Hydrogen bonding can also be used for recognizing different kinds of nodes by synthesizing them with unique arrangements of donor and acceptor regions. Currently (MacLen-

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

nan, 2002c), we are using eleven different node types ($\mathbf{A}$, $\mathbf{D}$, $\mathbf{K}$, $\mathbf{L}$, $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{R}$, $\mathbf{S}$, $\breve{\mathbf{S}}$, $\mathbf{V}$, $\breve{\mathbf{Y}}$), so it would seem that arrangements of five H-bonds would be sufficient (since they accommodate 16 complementary pairs of bond patterns). (The $\mathbf{D}$ -node controls the disassembly and recycling of waste products, and $\mathbf{P}$ and $\mathbf{Q}$ are inert place-holders, see MacLennan, 2002c.)

As previously remarked, it is necessary to be able to distinguish the three binding sites of an $\mathbf{A}$ -node. However, since the "tail" of any $\mathbf{L}$ -group must be able to bind to either of the $\mathbf{A}$'s argument sites, they must use the same H-bond pattern. Therefore, at least part of the discrimination of the $\mathbf{A}$'s binding sites must be on the basis of the orientation of the $\mathbf{A}$ node. Fortunately, the orientation specificity of H-bonds allows this.

Therefore, it appears that our primitive groups will require two or three H-bonds at each attachment site and four or five for secure identification of node type. By comparison with thymine (23 atoms) and adenine (26 atoms), which have two H-bonds each, we anticipate that our primitive combinators ($\mathbf{K}$, $\mathbf{S}$, $\breve{\mathbf{S}}$, $\breve{\mathbf{Y}}$) might be 90 atoms in size, $\mathbf{L}$ -groups about 120, and ternary groups ($\mathbf{A}$, $\mathbf{R}$, $\mathbf{V}$) about 150.

Substitutions

We state briefly the requirements on a molecular implementation of the primitive combinator substitutions. First, there must be a way of matching the network configurations that enable the substitution reactions. For example, a $\mathbf{K}$ -substitution (Eq. 1) is enabled by a leftward-branching tree of the form $((\mathbf{K}X)Y)$, and an $\mathbf{S}$ -substitution (Eq. 2) is enabled by a leftward-branching tree of the form $(((\mathbf{S}X)Y)Z)$ (see Figures 1 and 2). So also for the other primitive combinators ($\mathbf{D}$, $\mathbf{R}$, $\breve{\mathbf{S}}$, $\breve{\mathbf{Y}}$). Second, the variable parts of the matched structures (represented in the substitution rules by italic variables such as X and Y), which may be arbitrarily large supramolecular networks, must be bound in some way. Third, a new molecular structure must be constructed, incorporating some or all of these variable parts. Further, reaction waste products must be recycled or eliminated from the system, for several reasons. An obvious one is efficiency; another is to avoid the reaction space becoming clogged with waste. Less obvious is the fact that discarded molecular networks (such as Y in Eq. 1) may contain large executable structures; by the laws of combinatory logic, computation in these discarded networks cannot affect the computational result, but they can consume resources.

There are also energetic constraints on the substitution reactions, for any system of molecular computation must be fueled if it is universal (has the power of a universal Turing machine). This is because a spontaneous chemical reaction decreases Gibbs free energy, and so must eventually reach equilibrium, but a computation may be nonterminating. To have the power of universal computation we must have the potential of nonterminating programs. Fortunately we have several recent concrete examples of how such nonterminating processes may be fueled. For example, Koumura, Zijlstra, van Delden, Harada, and Feringa (1999) have demonstrated continuous (nonterminating) unidirectional rotary motion driven by ultraviolet light. In the four-phase rotation, alternating phases are by photochemical reaction (uphill) and by thermal relaxation (downhill). Also, Yurke, Turberfield, Mills, Simmel, and Neumann (2000) have demonstrated DNA "tweezers," which can be cycled between their open and closed states so long as an appropriate "DNA fuel" is provided. LaBean, Yan, Park, Feng, Yin, Li, Ahn, Liu, Guan, and Reif (2003) have demonstrated autonomous DNA "robots" powered by ATP consumption of a ligase. These all provide plausible models of how molecular combinatory computation might be powered. We can conclude that the individual steps of a combinator substitution must be either energetically "downhill" or fueled by external energy or reaction resources. In our case, the most likely sources of fuel are the various species of "substitutase" molecules, considered next.

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

To implement the substitution processes we are investigating the use of enzyme-like covalently-structured molecules to recognize network sites at which substitutions are allowed, and (through graded electrostatic interactions) to rearrange the hydrogen bonds to effect the substitutions. Again, the rich synthetic precedent for covalently-structured molecules makes it likely that the required enzyme-like compounds, which we call substitutase molecules, can be engineered. We anticipate the use of three enzyme-like covalently-structured molecules for each primitive combinator; they implement three stages in each substitution operation. The first of these molecules, which we call analysase, is intended to recognize the pattern enabling the substitution and to bind to the components of the matching subtree. For example, $\mathbf{K}$ -analysase binds to a structure of the form $((\mathbf{K}X)Y)$ (see Figure 1), in particular to links to the variable components U , X , and Y . The second stage, which is implemented by a permutase molecule, physically relocates some of the components to prepare them for the last stage. To this end, we are investigating the use of graded electrostatic interactions to move the bound variable parts into position for the correct substitution product. The permutase molecule also includes any fixed combinators (e.g., $\mathbf{R}$, $\mathbf{V}$) that are required for the product, which are bound to other product components at this time. At the end of this stage, the product network is essentially complete, but still bound to the permutase molecule. The final molecule, a synthesase, recognizes the configuration created by the permutase, and binds to the waste structures, displacing and releasing the desired product from the permutase. For example, $\mathbf{S}$ - or $\breve{\mathbf{S}}$ -synthesase will remove the ($\mathbf{S}$ - or $\breve{\mathbf{S}}$ -) permutase and release the structure on the right in Figure 2. Here again we are depending on the extensive synthetic precedent for covalently-structured molecules with strategically located hydrogen bonding sites.

Discussion

Finally, we will review some of the issues that must be resolved and problems that must be solved before molecular combinatory computing can be applied to nanotechnology. First, of course, it will be necessary to synthesize the required MBBs for the nodes and links, and to synthesize the required substitutase molecules; fortunately, there is every reason to believe that this is well within the capabilities of the state of the art of synthetic chemistry (Allis & Spencer, 2003). Another issue is error control: substitutions will not take place with perfect accuracy, and we know that some substitution errors can result in runaway reactions (MacLennan, 1997; YarKhan, 2000). Therefore we must develop means to prevent errors or to correct them soon after they occur. A third issue is that the supramolecular networks may get quite dense during computation, and we are concerned about the ability, and probability, of the substitutase molecules reaching the sites to which they should bind (i.e., what are the steric constraints on the processes). Nevertheless, the enormous potential of molecular combinatory computing makes these problems worth solving.

CONCLUSION

A small set of simple network-substitution operations are sufficient to implement any computation that can be performed on a digital computer. These operations, which may be performed in any order or in parallel, provide an ideal model for autonomous molecular computation. We displayed several simulated applications to nanostructure synthesis, and indicated how it may be applied to the assembly of large, active, heterogeneous structures. Finally, we discussed a possible molecular implementation based on networks of covalently-structured MBBs connected by H-bonds, and substitution operations implemented by endothermic reactions with synthetic “substitutase” molecules. Additional information about this project can

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

be found in reports and articles (MacLennan, 2002a, 2002b, 2002c, 2002d, 2003a, 2003b) archived at the project website: http://web.eecs.utk.edu/~mclennan/UPIM.

ACKNOWLEDGMENTS

This research is supported by Nanoscale Exploratory Research grant CCR-0210094 from the National Science Foundation. It has been facilitated by a grant from the University of Tennessee, Knoxville, Center for Information Technology Research. The author's research in this area was initiated in 1997 when he was a Fellow of the Institute for Advanced Studies of the Collegium Budapest.

REFERENCES

Allis, D. G., & Spencer, J. T. (2003). Nanostructural architectures from molecular building blocks . In Goddard, W. A., Brenner, D. W., Lyshevski, S. E., & Iafrate, G. J. (Eds.), *Handbook of Nanoscience, Engineering, and Technology*. Boca Raton, FL: CRC Press.

Curry, H. B., Feys, R., & Craig, W. (1958). *Combinatory Logic* (*Vol. I*). Amsterdam, The Netherlands: North-Holland.

Koumura, N., Zijlstra, R. W. J., van Delden, R. A., Harada, N., & Feringa, B. L. (1999). Light-driven monodirectional molecular rotor. *Nature*, *401*, 152–155. doi:10.1038/43646

LaBean, T. H., Yan, H., Park, S. H., Feng, L., Yin, P., Li, H., et al. (2003). Overview of new structures for DNA-based nanofabrication and computation. In P. P. Wang & A. Menzies (Eds.), *Proceedings of the 7th Joint Conference on Information Sciences* (pp. 1475-1478). Durham, NC: JCIS/Association for Intelligent Machinery.

MacLennan, B. J. (1997). *Preliminary Investigation of Random SKI-Combinator Trees* (Tech. Rep. No. CS-97-370). Knoxville, TN: Department of Computer Science, University of Tennessee, Knoxville.

MacLennan, B. J. (2002a). *Membranes and Nanotubes: Progress on Universally Programmable Intelligent Matter — UPIM Report 4* (Tech. Rep. No. CS-02-495). Knoxville, TN: Department of Computer Science, University of Tennessee, Knoxville.

MacLennan, B. J. (2002b). *Molecular Combinator Reference Manual — UPIM Report 2* (Tech. Rep. No. CS-02-489). Knoxville, TN: Department of Computer Science, University of Tennessee, Knoxville.

MacLennan, B. J. (2002c). *Replication, Sharing, Deletion, Lists, and Numerals: Progress on Universally Programmable Intelligent Matter — UPIM Report 3* (Tech. Rep. No. CS-02-493). Knoxville, TN: Department of Computer Science, University of Tennessee, Knoxville.

MacLennan, B. J. (2002d). *Universally Programmable Intelligent Matter (Exploratory Research Proposal) — UPIM Report 1* (Tech. Rep. No. CS-02-486). Knoxville, TN: Department of Computer Science, University of Tennessee, Knoxville.

MacLennan, B. J. (2003a). *Molecular Combinator Reference Manual*. Retrieved from http://www.cs.utk.edu/ ~mclennan/UPIM/CombRef.pdf.

MacLennan, B. J. (2003b). *Sensors, Patches, Pores, and Channels: Progress on Universally Programmable Intelligent Matter — UPIM Report 5* (Tech. Rep. No. CS-03-513). Knoxville, TN: Department of Computer Science, University of Tennessee, Knoxville.

Simard, M., Su, D., & Wuest, J. D. (1991). Use of hydrogen bonds to control molecular aggregation. Self-assembly of three-dimensional detworks with large chambers. *Journal of the American Chemical Society*, *113*(12), 4696–4698. doi:10.1021/ja00012a057

YarKhan. A. (2000). *An Investigation of Random Combinator Soups*. Knoxville, TN: Department of Computer Science, University of Tennessee, Knoxville.

Yurke, B., Turberfield, A. J., Mills, A. P. Jr, Simmel, F. C., & Neumann, J. L. (2000). A DNA-fuelled molecular machine made of DNA. *Nature*, *406*, 605–608. doi:10.1038/35020524

Zimmerman, S. C., Zeng, F. W., Reichert, D. E. C., & Kolotuchin, S. V. (1996). Self-assembling dendrimers. *Science*, *271*, 1095. doi:10.1126/science.271.5252.1095

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Bruce J. MacLennan has a BS in mathematics (with honors, 1972) from Florida State University, and an MS (1974) and PhD (1975) in computer science from Purdue University. He was a Senior Software Engineer with Intel Corporation (1975–9), after which he joined the Computer Science faculty of the Naval Postgraduate School (Monterey, CA) as Assistant Professor (1979–83), Associate Professor (1983–7), and Acting Chair (1984–5). Since 1987 he has been a member of the Electrical Engineering and Computer Science faculty of the University of Tennessee, Knoxville. MacLennan's research includes the application of molecular computing to nanostructure synthesis and control and the development of novel models of computation intended to better exploit physical processes for computation. Prof. MacLennan has more than 60 refereed journal articles and book chapters and has published two books. He has made more than 60 invited or refereed presentations.

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

APPENDIX

For completeness, we include the definitions of all combinators used in this paper (except the primitives $\mathbf{K}$, $\mathbf{N}$, $\mathbf{S}$, $\breve{\mathbf{S}}$, and $\breve{\mathbf{Y}}$). For additional explanation, see the combinatory logic literature (Curry, Feys, & Craig, 1958) as well as our previous reports (MacLennan, 2002d, 2003a). For convenience, the composition operator may be used as an abbreviation for the $\mathbf{B}$ combinator: $X \circ Y = \mathbf{B}XY$. In combinatory logic, omitted parentheses are assumed to nest to the left, so for example $\mathbf{S}(\mathbf{KS})\mathbf{K} = \left(\left(\mathbf{S}\left(\mathbf{KS}\right)\right)\mathbf{K}\right)$.

$$
\begin{aligned}
\mathbf{B} &= \mathbf{S}\left(\mathbf{KS}\right)\mathbf{K} \\
\mathbf{C} &= \mathbf{B}\left(\mathbf{BS}\right) \\
\mathbf{I} &= \mathbf{SKK} \\
\breve{\mathbf{S}}_1 &= \breve{\mathbf{S}} \\
\breve{\mathbf{S}}_{n+1} &= \mathbf{B}\breve{\mathbf{S}}_n \circ \breve{\mathbf{S}} \\
\mathbf{W} &= \mathbf{CSI} \\
\breve{\mathbf{W}} &= \mathbf{C}\breve{\mathbf{S}}\mathbf{I} \\
\mathbf{Z}_0 &= \mathbf{KI} \\
\mathbf{Z}_{n+1} &= \mathbf{SBZ}_n \\
\breve{\Phi}_n &= \breve{\mathbf{S}}_n \circ \mathbf{K}
\end{aligned}
$$

For any X,

$$
\begin{aligned}
X^1 &= X \\
X^{n+1} &= X \circ X^n \\
X_{(0)} &= X \\
X_{(n+1)} &= \mathbf{B}X_{(n)} \\
X_{[1]} &= X \\
X_{[n+1]} &= \mathbf{B}X_{[n]} \circ X \\
X^{[1]} &= X \\
X^{[n+1]} &= X \circ \mathbf{B}X^{[n]}
\end{aligned}
$$

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.