Molecular Implementation of Combinatory Computing for Nanostructure Synthesis and Control:

Progress on Universally Programmable Intelligent Matter

UPIM Report 6

Technical Report UT-CS-03-506

Bruce J. MacLennan*

Department of Computer Science
University of Tennessee, Knoxville
maclennan@cs.utk.edu

July 7, 2003

Abstract

Molecular combinatory computing makes use of a small set of chemical reactions that together have the ability to implement arbitrary computations. Therefore it provides a means of "programming" the synthesis of nanostructures and of controlling their behavior by programmatic means. We illustrate the approach by several simulated nano-assembly applications, and discuss a possible molecular implementation in terms of covalently structured molecular building blocks connected by hydrogen bonds.

*This report is an extended version of a paper to be presented at IEEE-Nano 03 [Mac03c] incorporating additional material from a presentation at JCIS 7 [Mac03b]. This report may be used for any non-profit purpose provided that the source is credited.

1 Introduction

We are investigating a systematic approach to nanotechnology based on a small number of molecular building blocks (MBBs). Central to our approach is the identification of a small set of such MBBs that is provably sufficient for controlling the nanoscale synthesis and behavior of materials. To accomplish this we have made use of *combinatory logic* [CFC58], a mathematical formalism based on network (graph) substitution operations suggestive of supramolecular interactions. This theory shows that two simple substitution operations (known as S and K) are sufficient to describe any computable process (Turing-computable function) [CFC58, sec. 5H]. Therefore, these two operations are, in principle, sufficient to describe any process of nanoscale synthesis or control that could be described by a computer program. In a molecular context, several additional housekeeping operations are required beyond S and K, but the total is still less than a dozen.

In addition, 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* [CFC58, ch. 4] proves that substitutions may be performed in any order or in parallel without affecting the computational result; this is very advantageous for molecular computation. (More precisely, the theorem states that *if* you get a result, you always get the 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.)

At IEEE-Nano 2002 we presented an overview of the strategy and potential of molecular combinatory computing [Mac02f]. In this report, in addition to a brief introduction to molecular combinatory computing, we discuss possible molecular implementations as well as our accomplishments in the (simulated) synthesis of membranes, channels, nanotubes, and other nanostructures.

2 Combinatory Computing

2.1 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. As previously remarked, one of the remarkable theorems of combinatory logic is that two simple substitution operations are sufficient for implementing any program (Turing-computable function). Therefore we use primarily the two primitive combinators, K and S (which exists in two variants Š and S proper).

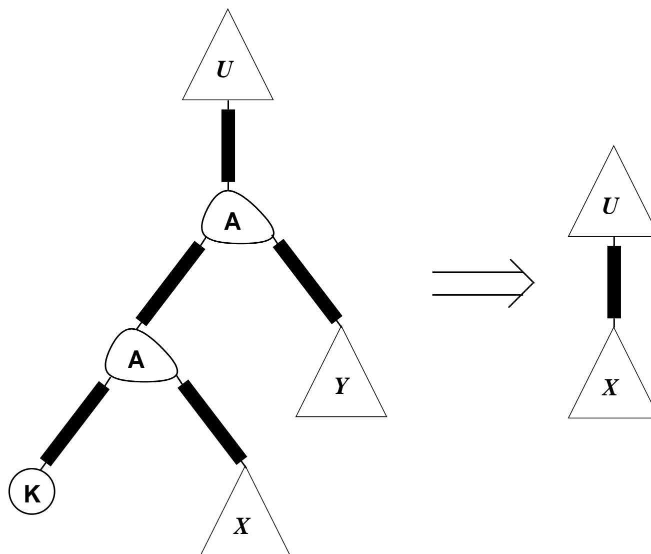

Figure 1: K combinator substitution operation. U, X, and Y represent any networks (graphs).

To understand these operations, consider a binary tree of the form $((\mathsf{K}X)Y)$, where X and Y are binary trees (Fig. 1). (The A nodes are implicit in the parentheses.) This configuration triggers a substitution reaction, which has the effect

$$((\mathsf{K}X)Y) \Longrightarrow X. \tag{1}$$

That is, the complex $((\mathsf{K}X)Y)$ is replaced by X in the supramolecular network structure; the effect of the operation is to delete Y from the network. The K group is released as a waste product, which may be recycled in later reactions. The tree Y is also a waste product, which may be bound to another primitive operator (D), which disassembles the tree so that its components may be recycled. The D primitive is the first of several house-keeping operations, which are not needed in the theory of combinatory logic, but are required for molecular computation. (Detailed descriptions can be found in a prior report [Mac02d].)

The second primitive operation is described by the rule:

$$(((\mathsf{S}X)Y)Z) \Longrightarrow ((XZ)(YZ)). \tag{2}$$

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 S (replicating) and $\check{\mathsf{S}}$ (sharing). The molecular implementations of the two are very similar (see Fig. 2).

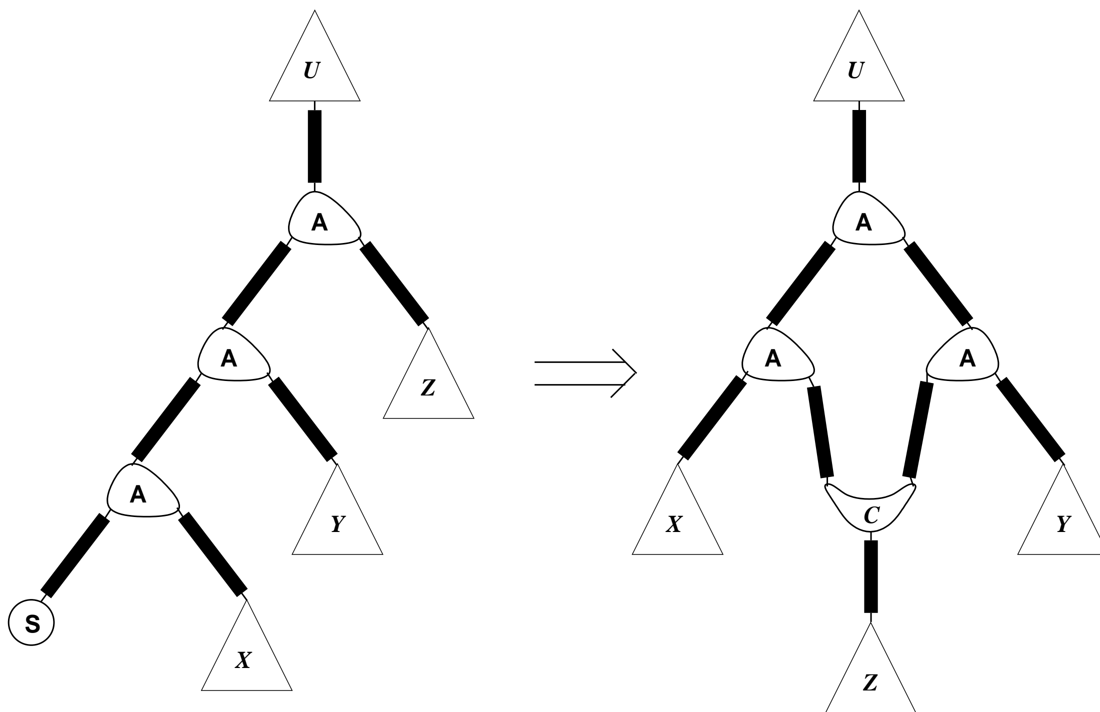

Figure 2: S and Š combinator substitution operations. $C = \mathsf{R}$ for S and $C = \mathsf{V}$ for Š. Note the reversed orientation of the rightmost A group.

If $C = \mathsf{R}$ (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:

$$(((\mathsf{S}X)Y)Z) \Longrightarrow ((XZ)(YZ')). \tag{3}$$

Here Z' refers to a new copy of the structure Z, which is created by a primitive replication (R) operation. The R operation progressively duplicates Z, "unzipping" the original and new copies [Mac02d]. The properties of combinatory computing allow this replication to take place while other computation proceeds, even including use of the partially completed replicate (a consequence of the Church-Rosser theorem).

The Š variant of the S operation is essential to molecular synthesis [Mac02d]. Thus, if $C = \mathsf{V}$ (a *sharing node*), then we have two links to a shared copy of Z (Fig. 2):

$$(((\check{\mathsf{S}}X)Y)Z) \Longrightarrow ((XZ^{(1)})(YZ^{(0)})). \tag{4}$$

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 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 Š operation is to synthesize non-tree-structured supramolecular networks.

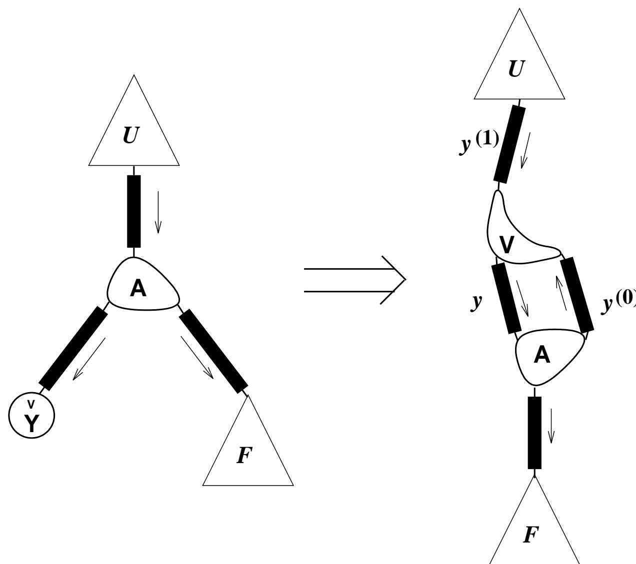

Figure 3: Y̌ combinator primitive substitution operation. Arrows indicate link direction; note elementary cycle between A and V groups.

Finally, we use the Y̌ operator to create elementary cyclic structures, which can be expanded into larger cycles. It is defined [Mac02d]:

$$(\check{\mathsf{Y}}F) \Longrightarrow y^{(1)} \quad \text{where } y \equiv (Fy^{(0)}). \tag{5}$$

See Fig. 3. The operation creates an elementary cycle, which may be expanded by computation in combinator tree F. Examples of the use of both Š and Y̌ are given in Sec. 3.

2.2 Molecular Extensions

The primitive substitutions already mentioned $(\mathsf{K}, \mathsf{S}, \check{\mathsf{S}}, \check{\mathsf{Y}})$ 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 [Mac03a]. For example, we may design a molecular group $\mathsf{K}_{-\lambda}$ that is normally inert, but is recognized (and therefore operates) as 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 Sec. 3.5.1).

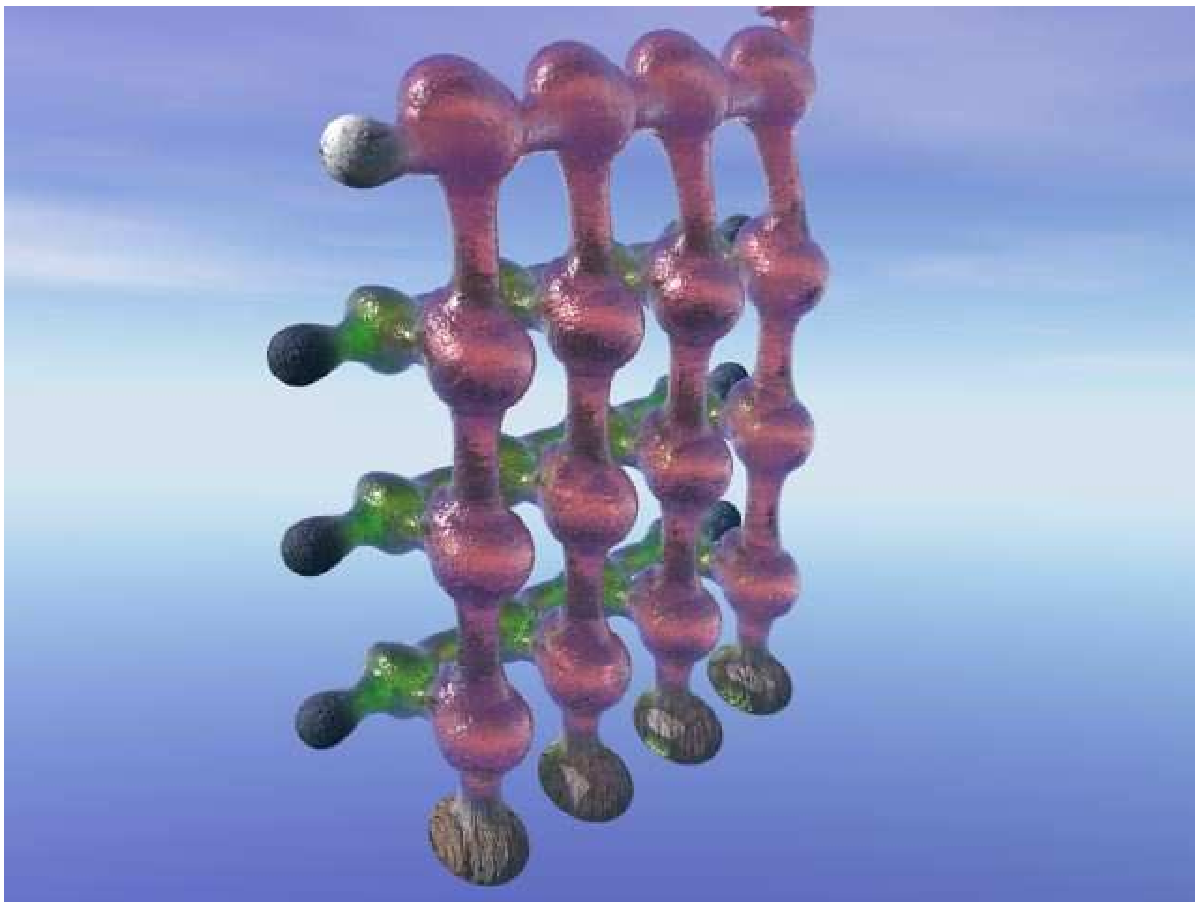

Figure 4: Visualization of cross-linked membrane produced by $\mathsf{xgrid}_{3,4}\mathsf{NNN}$. A groups are red, V groups green; other groups are inert.

3 Examples

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.

3.1 Membranes

For our first example we will discuss the synthesis of a cross-linked membrane, such as shown in Fig. 4. Such a structure is produced by the combinator program $\mathsf{xgrid}_{3,4}\mathsf{NNN}$, where xgrid is defined:

$$\mathsf{xgrid}_{m,n} = \mathsf{B}(\mathsf{B}(\mathsf{Z}_{m-1}\mathsf{W}))(\mathsf{B}(\mathsf{Z}_{n-1}\mathsf{W})(\mathsf{Z}_m\check{\Phi}_n)), \tag{6}$$

which is an abbreviation for a large binary tree of A, K, S, and $\check{\mathsf{S}}$ groups. Unfortunately, space does not permit an explanation of this program or a proof of its correctness, both of which may be found in a technical report [Mac02a]. However, all the combinators that it uses are defined in the appendix to this report. In the formula $\mathsf{xgrid}_{m,n}XYZ$, the parameters m and n are the height and width of the membrane, respectively. X, Y, and Z are the terminal groups to be used on the left ends, right ends, and bottoms of the chains. (In the expression $\mathsf{xgrid}_{3,4}\mathsf{NNN}$, N is any inert group.)

The size of $\mathsf{xgrid}_{m,n}$, the program structure to generate an $m \times n$ membrane, can be shown [Mac02a] to be $20m + 28n + 73$ primitive groups (A, K, S, $\check{\mathsf{S}}$). 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 Z_m in Eq. 6 can be replaced by $\mathsf{Z}_k\mathsf{Z}_{10}$, reducing the size of this part of the program from $\mathcal{O}(10^k)$ to $\mathcal{O}(k)$ (see [Mac02d] for explanation). 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 $\mathcal{O}(\log m + \log n)$, to $\mathcal{O}(\log\log m + \log\log n)$, etc. Furthermore, as will be explained in Sec. 3.4, large membranes can be synthesized by iterative assembly of small patches.

3.1.1 Hexagonal Membrane

For another example, consider the hexagonally structured membrane in Fig. 5. It is constructed by the combinator program $\mathsf{hgridt}_{2,3}\mathsf{N}$, where [Mac02a]:

$$\begin{aligned}
\mathsf{Arow}_n &= \mathsf{B}\check{\mathsf{W}}_{[n-1]} \circ \mathsf{B}^{[n]}, && (7)\\
\mathsf{Vrowt}_n &= \check{\mathsf{W}}_{[n]} \circ \mathsf{KI} \circ \mathsf{K}_{(2n-2)} \circ \mathsf{B}^{[n-1]} \circ \mathsf{C}^{[n]}\mathsf{IN} \circ \mathsf{CIN}, && (8)\\
\mathsf{drowt}_n &= \mathsf{Vrowt}_n \circ \mathsf{Arow}_n, && (9)\\
\mathsf{hgridt}_{m,n} &= \mathsf{Z}_{n-1}\mathsf{W}(\mathsf{Z}_m\mathsf{drowt}_n\mathsf{N}). && (10)
\end{aligned}$$

The size of the program is $\mathcal{O}(m + n)$ primitive combinators.

3.2 Nanotubes

Next we consider the synthesis of nanotubes, such as shown in Figs. 6 and 7. This is accomplished by using the $\check{\mathsf{Y}}$ combinator to construct a cycle between the upper and lower margins of the cross-linked membrane (Fig. 4). The program is [Mac02a]:

$$\mathsf{xtubep}_{m,n} = \check{\mathsf{W}}^{m-1}(\mathsf{W}^{n-1}(\check{\Phi}_n^m\mathsf{N})(\mathsf{B}^m\check{\mathsf{Y}}(\mathsf{C}_{[m]}\mathsf{I}))). \quad (11)$$

The size of $\mathsf{xtubep}_{m,n}$ is $102m + 44n - 96$ primitive groups (A, K, N, S, $\check{\mathsf{S}}$, $\check{\mathsf{Y}}$).

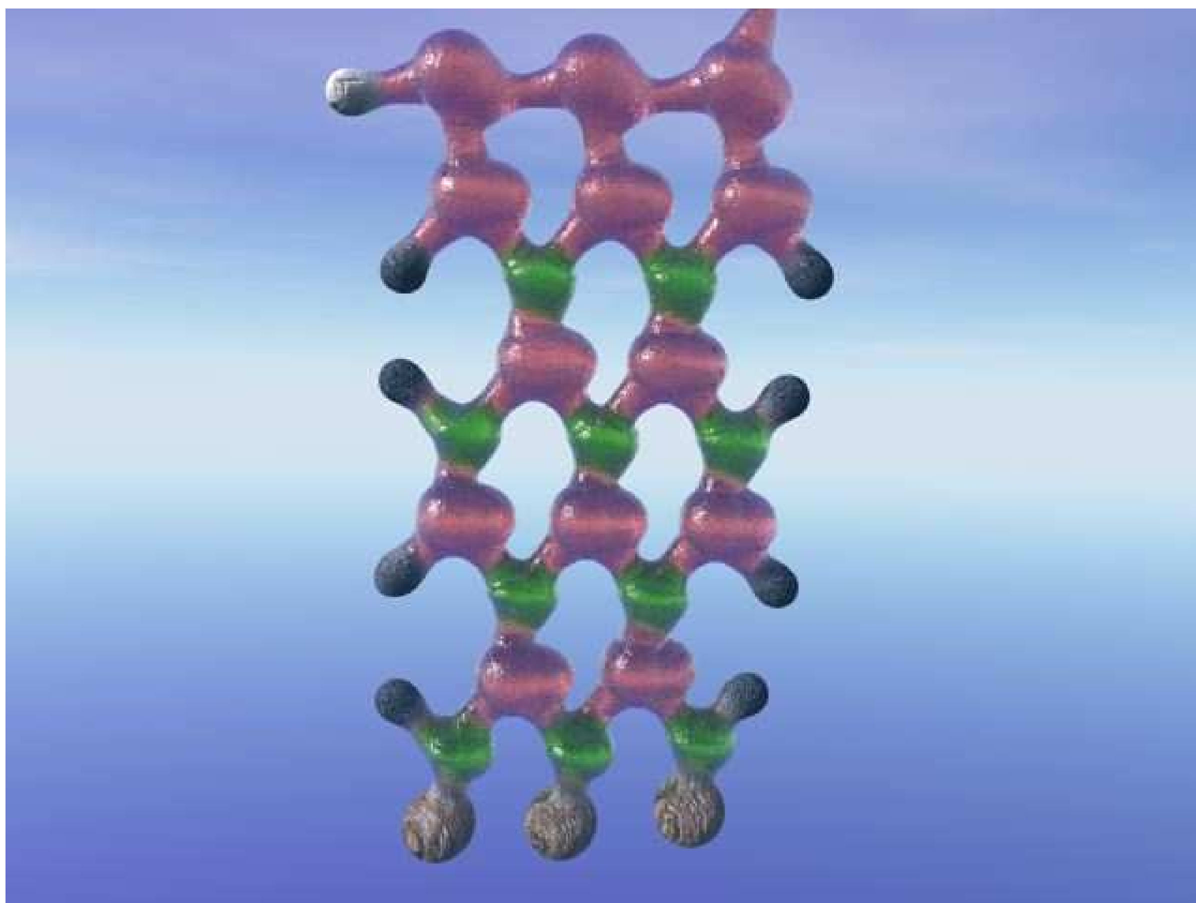

Figure 5: Visualization of small hexagonal membrane synthesized by $\mathsf{hgridt}_{2,3}\mathsf{N}$.

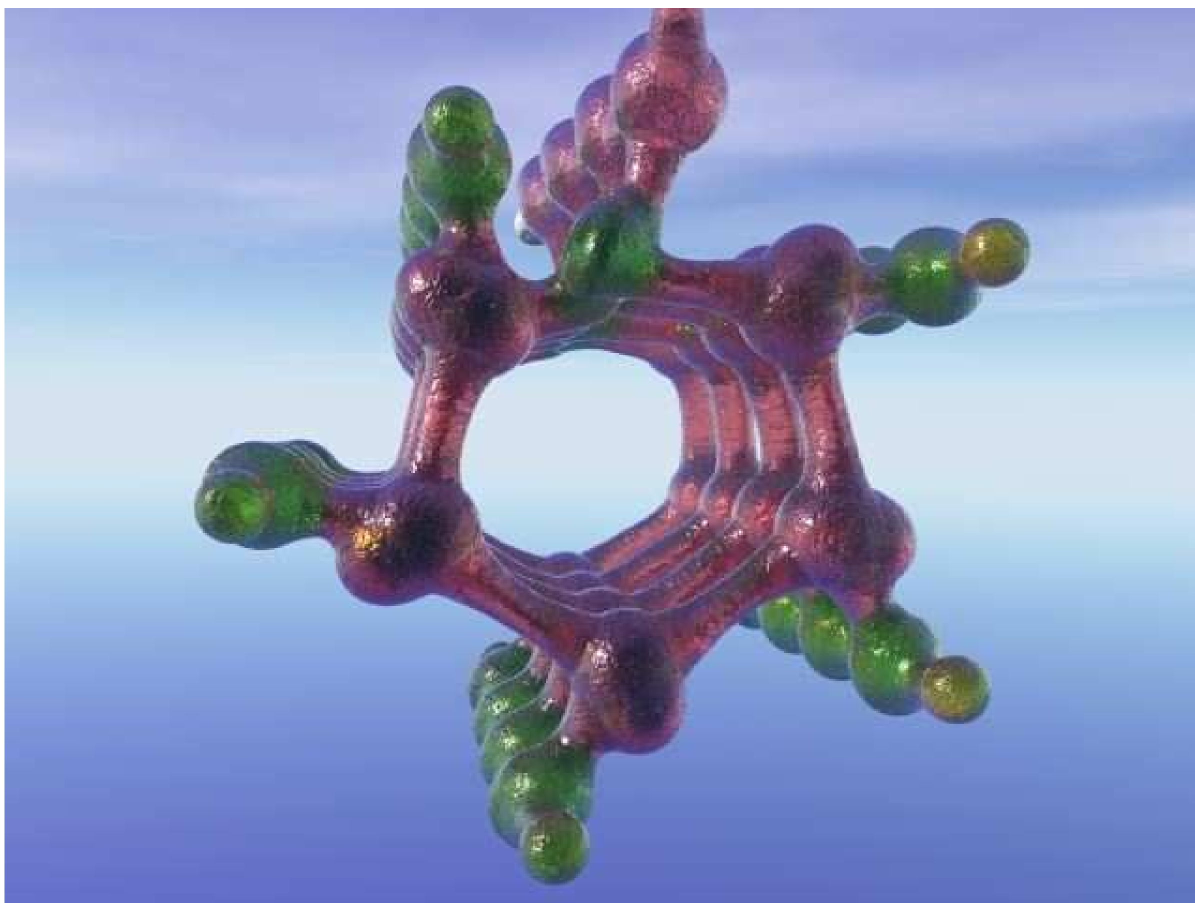

Figure 6: Visualization of small nanotube, end view, produced by $\mathsf{xtube}_{5,4}\mathsf{N}$.

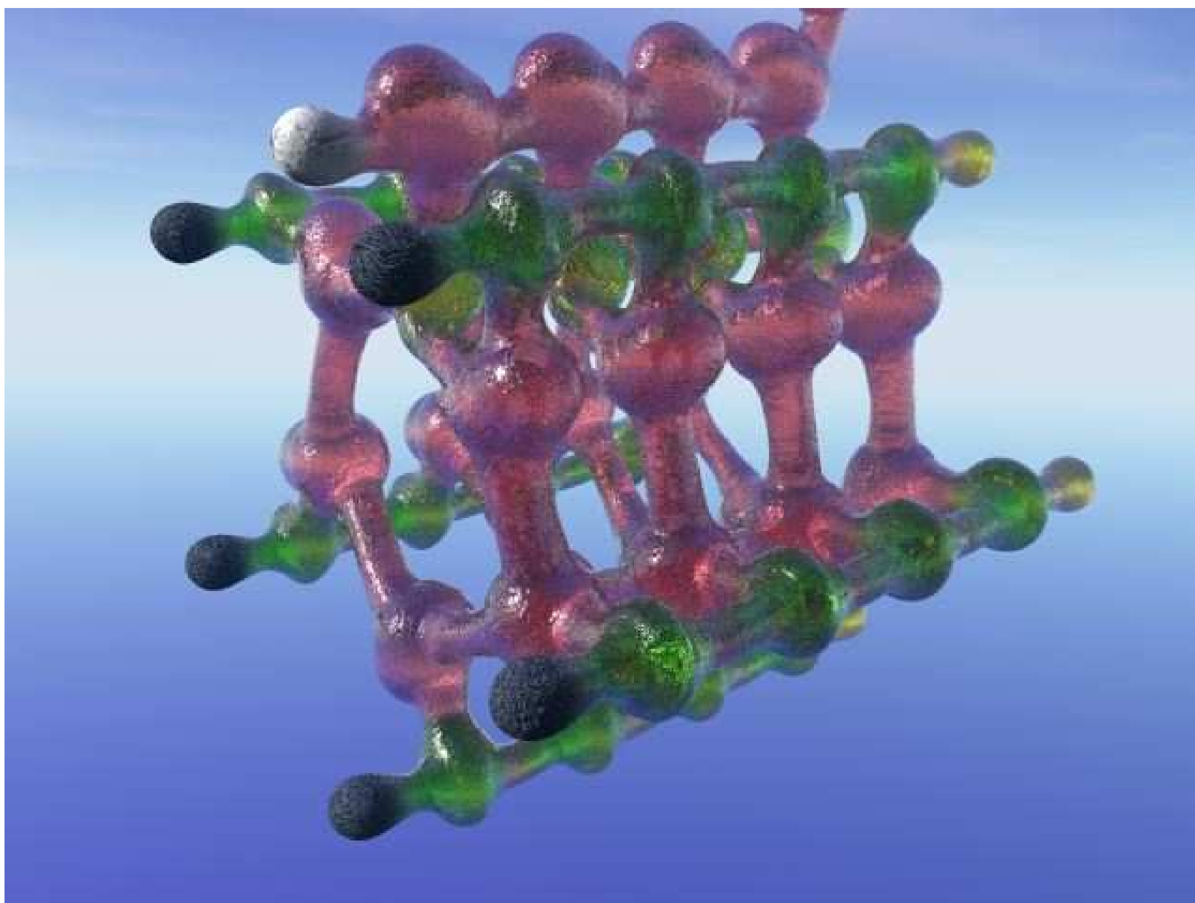

Figure 7: Visualization of small nanotube, side view, produced by $\mathsf{xtube}_{5,4}\mathsf{N}$.

3.3 Non-unit Meshes

The membranes and nanotubes previously described are said to have a *unit mesh*, that is, the dimensions of the basic (square or hexagonal) cells are determined by the size of the primitive groups (A, 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. (See a forth-coming report [Mac03a] for details.)

3.4 Assembly of Heterogeneous Structures

Large membranes will not be homogeneous in structure; often they will contain pores and active channels of various sorts embedded in a matrix. One way of assembling such a structure is by combining rectangular patches as in a patchwork quilt. To accomplish this we have defined a uniform interface for such patches (see a forthcoming report [Mac03a] for details).

A combinatory program P constructs an $m \times n$ patch if it has the following *patch synthesizer* "signature":

$$PFY_1 \cdots Y_n X_1 \cdots X_m \Longrightarrow FV_1 \cdots V_m U_1 \cdots U_n.$$

F is any combinatory operator (especially another patch synthesizer). $X_1, \ldots, X_m$ will be horizontal connections from the patch to the right (or terminal groups if this is the right-most patch); similarly $Y_1, \ldots, Y_n$ are vertical connections from the patch below. Whatever rectangular structure is created by P (e.g., a cross-linked or hexagonal grid), $V_1, \cdots V_m$ represents the horizontal connections from its left side, and $U_1, \ldots, U_n$, the vertical connections from its top. The V_k and U_j connections are passed to F, which is typically another patch synthesizer.

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 $\mathsf{B}^{n+1}QP$ is an $m \times (n + n')$ patch with Q to the right of P. Similarly, if P is $m \times n$ and Q is $m' \times n$, then $P \circ \mathsf{B}^m Q$ is the $(m + m') \times n$ patch with P below Q.

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 is k replicates of T connected end-to-end (and thus of length km). This operation can also be expressed $\mathsf{Z}_k T$. If, as is commonly the case, the size of the synthesizer T is $\mathcal{O}(m+n)$, then the size of $\mathsf{Z}_k T$ is $\mathcal{O}(k+m+n)$. Similarly, rectangular patches can be iteratively assembled, both horizontally and vertically, to hierarchically synthesize large, heterogeneous membranes. This allows us to build upon a basic library of elementary membrane patches, pores, and other nanostructural units.

3.5 Active Elements

Rather than computing to a stable state, *dynamic structures* remain potentially active, ready to respond to environmental conditions [Mac03a]. Unfortunately, space does not permit a detailed discussion of the synthesis of membranes with pores and channels; the following brief remarks must suffice.

3.5.1 Pores and Active Channels

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.

Channels open or close under control of *sensor molecules*, which can respond to conditions, such as electromagnetic radiation or the presence of chemical species of interest. This is most simply accomplished by synthesizing a molecular group, which we denote $\mathsf{K}_{-\lambda}$, that responds to condition λ by reconfiguring into a K combinator.

Given a sensor, "one-shot" channels — which open and stay open, or close and stay closed — are easy to implement. In the former case, the sensor triggers the dissolution of the interior of its patch (perhaps using the deletion operator D to disassemble it). In the latter case, the sensor triggers a synthesis process to fill in a pore. Reusable channels (which open and close repeatedly) are more complicated, since, in order to reset themselves, they need a supply of sensor molecules that are protected from being triggered before they are used.

3.5.2 Nano-Actuators

Nano-actuators have some physical effect depending on a computational process. Certainly, many of these will be synthesized for special purposes. However, we have been investigating actuators based directly on the computational reactions. To give a very simple example, we may program a computation that synthesizes a chain of some length; we may also program a computation that collapses a chain into a single link. The two of these can be used together, like opposing muscle groups, to cause motion under molecular program control. The force that can be exerted will depend on the bond strength of the nodes and links (probably on the order of 50 kJ/mol; see Sec. 4.1.2). However, these forces can be combined additively, as individual muscle fibers work cooperatively in a muscle.

3.6 Computational Applications

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 either **true** or **false**: usually K and (SK) [Mac02d]. 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.

4 Possible 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 spent some time trying to develop at least 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?

4.1 Combinator Networks

4.1.1 Requirements

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.

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 (A, K, S, etc.) need to identifiable, so that the correct substitutions take place. In addition, for more secure matching of structures, the link (L) groups should be identifiable. Further, it is necessary to be able to distinguish the various binding sites on a node. For example, an A node has three distinct sites: the result site, an operator argument, and an operand argument [Mac02d].

4.1.2 Hydrogen-Bonded Covalent Subunits

Our current approach is to implement the nodes and linking groups by covalently-structured molecular building blocks (MBBs) and to interconnect them by hydrogen

bonds. We use a covalent framework for the nodes and links 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 of the required shape and with appropriately located hydrogen bonding sites [AS03]. This is in fact the structural basis of both DNA and proteins (hydrogen bonding as a means of connecting and identifying covalently-bonded subunits).

Hydrogen bonds are used to interconnect the MBBs because they are labile in aqueous solution, permitting continual disassembly and re-assembly of structures. (H-bond strengths are 2–40 kJ/mol.) Further, other laboratories have demonstrated the synthesis and manipulation of large hydrogen-bonded tree-like structures (*dendrimers*) [SSW91, ZZRK96]. Nevertheless, hydrogen bonds are not very stable in aqueous solution, so there may be a delicate balance between stability and lability.

It is necessary to be able to distinguish the "head" and "tail" ends [Mac02d] of the 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, 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 [Mac02d], we are using eleven different node types (A, D, K, L, P, Q, R, S, Š, V, Ÿ), so it would seem that arrangements of five H-bonds would be sufficient (since they accommodate 16 complementary pairs of bond patterns). (Actually, linear patterns of from one to four bonds are sufficient – 15 complementary pairs – but the slight savings does not seem worth the risk of less secure identification.)

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

A number of hydrogen-bonding sites can be located in a small area. For example, thymine (23 atoms) and adenine (26 atoms) have two H-bonds; amino acids are also small, on the order of 30 atoms and as few as 10. On the basis of the above considerations, we estimate — very roughly! — that our primitive combinators (K, S, Š, Ÿ) might be 90 atoms in size, L groups about 120, and ternary groups (A, V, R) about 150.

4.2 Substitution Reactions

4.2.1 Requirements

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 K-substitution (Eq. 1) is enabled by a leftward-branching tree of the form $((\mathsf{K}X)Y)$, and an S-substitution (Eq. 2) is enabled by a leftward-branching tree of the form $(((\mathsf{S}X)Y)Z)$ (see Figs. 1 and 2). So also for the other primitive combinators (D, R, $\check{\mathsf{S}}$, $\check{\mathsf{Y}}$).

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.

Finally, there are energetic constraints on the substitution reactions, to which we now turn.

4.2.2 Fundamental Energetic Constraints

On the one hand, any system that is computationally universal (i.e., equivalent to a Turing machine in power) must permit nonterminating computations. On the other, a spontaneous chemical reaction will take place only if it decreases Gibbs free energy; spontaneous reactions tend to an equilibrium state. Therefore, molecular combinatory computing will require an external source of energy or reaction resources; it cannot continue indefinitely in a closed system.

Fortunately we have several recent concrete examples of how such nonterminating processes may be fueled. For example, Koumura et al. [KZvD+99] 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 et al. [YTM+00] have demonstrated DNA "tweezers," which can be cycled between their open and closed states so long as an appropriate "DNA fuel" is provided. Both of these 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 reactions resources. In our case, the most likely sources of fuel are the various species of "substitutase" molecules (see next).

4.2.3 Use of Synthetic Substitutase Molecules

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 kinds of substitutase 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 operation and to bind to the components of the matching subtree. For example, K-analysase binds to a structure of the form $((\mathsf{K}X)Y)$ (see Fig. 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., R, V) that are required for the product, and 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, S- or Š-synthesase will remove the (S- or Š-) permutase and release the structure shown on the right in Fig. 2.

4.2.4 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; fortunately, there is every reason to believe that this is well within the capabilities of the state of the art of synthetic chemistry [AS03]. Also, it will be necessary to synthesize the required substitutase molecules; again, there is every reason to believe that this is well within the capabilities of the state of the art of synthetic chemistry.

A second problem is error control: substitutions will not take place with perfect accuracy, and we know that some substitution errors can result in runaway reactions [Mac97, Yar00]. 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 substi-

tutase 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.

5 Conclusions

After briefly reviewing the concept of molecular combinatory computing, we displayed several simulated applications to nanostructure synthesis. We also 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. Unfortunately, we have had to omit much explanation, discussion, and analysis, but it can be found in other publications from our project [Mac02a, Mac02c, Mac02d, Mac02e], archived at the project website: `http://www.cs.utk.edu/~mclennan/UPIM`.

6 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 when he was a Fellow of the Institute for Advanced Studies of the Collegium Budapest (1997).

A Definitions

For completeness, we include the definitions of all combinators used in this report (except the primitives K, N, S, $\check{\mathsf{S}}$, and $\check{\mathsf{Y}}$). For additional explanation, see the combinatory logic literature [CFC58, e.g.] as well as our previous reports [Mac02b, Mac02c, Mac02e]. For convenience, the composition operator may be used as an abbreviation for the B combinator: $X \circ Y = \mathsf{B}XY$. In combinatory logic, omitted parentheses are assumed to nest to the left, so for example $\mathsf{S(KS)K} = ((\mathsf{S(KS)})\mathsf{K})$.

$$\mathsf{B} = \mathsf{S(KS)K} \tag{12}$$
$$\mathsf{C} = \mathsf{B(BS)} \tag{13}$$
$$\mathsf{I} = \mathsf{SKK} \tag{14}$$
$$\check{\mathsf{S}}_1 = \check{\mathsf{S}} \tag{15}$$
$$\check{\mathsf{S}}_{n+1} = \mathsf{B}\check{\mathsf{S}}_n \circ \check{\mathsf{S}} \tag{16}$$
$$\mathsf{W} = \mathsf{CSI} \tag{17}$$
$$\check{\mathsf{W}} = \mathsf{C}\check{\mathsf{S}}\mathsf{I} \tag{18}$$
$$\mathsf{Z}_0 = \mathsf{KI} \tag{19}$$
$$\mathsf{Z}_{n+1} = \mathsf{SBZ}_n \tag{20}$$
$$\check{\Phi}_n = \check{\mathsf{S}}_n \circ \mathsf{K} \tag{21}$$

For any X,

$$X^1 = X \tag{22}$$
$$X^{n+1} = X \circ X^n \tag{23}$$
$$X_{(0)} = X \tag{24}$$
$$X_{(n+1)} = \mathsf{B}X_{(n)} \tag{25}$$
$$X_{[1]} = X \tag{26}$$
$$X_{[n+1]} = \mathsf{B}X_{[n]} \circ X \tag{27}$$
$$X^{[1]} = X \tag{28}$$
$$X^{[n+1]} = X \circ \mathsf{B}X^{[n]} \tag{29}$$

References

[AS03] Damian G. Allis and James T. Spencer. Nanostructural architectures from molecular building blocks. In William A. Goddard, Donald W. Brenner, Sergey Edward Lyshevski, and Gerald J. Iafrate, editors, *Handbook of Nanoscience, Engineering, and Technology*, chapter 16. CRC Press, 2003.

[CFC58] H. B. Curry, R. Feys, and W. Craig. *Combinatory Logic, Volume I.* North-Holland, Amsterdam, 1958.

[KZvD+99] Nagatoshi Koumura, Robert W. J. Zijlstra, Richard A. van Delden, Nobuyuki Harada, and Ben L. Feringa. Light-driven monodirectional molecular rotor. *Nature*, 401:152–5, 1999.

[Mac97] Bruce J. MacLennan. Preliminary investigation of random SKI-combinator trees. Technical Report CS-97-370, Dept. of Computer Science, University of Tennessee, Knoxville, 1997. Available at http://www.cs.utk.edu/~library/TechReports/ 1997/ut-cs-97-370.ps.Z.

[Mac02a] Bruce J. MacLennan. Membranes and nanotubes: Progress on universally programmable intelligent matter — UPIM report 4. Technical Report CS-02-495, Dept. of Computer Science, University of Tennessee, Knoxville, 2002. Available at http://www.cs.utk.edu/ ~library/TechReports/ 2002/ut-cs-02-495.ps.

[Mac02b] Bruce J. MacLennan. Molecular combinator reference manual. Technical report, Dept. of Computer Science, University of Tennessee, Knoxville, 2002. Latest edition available at http://www.cs.utk.edu/ ~mclennan/UPIM/CombRef.ps.

[Mac02c] Bruce J. MacLennan. Molecular combinator reference manual — UPIM report 2. Technical Report CS-02-489, Dept. of Computer Science, University of Tennessee, Knoxville, 2002. Available at http://www.cs.utk.edu/ ~library/TechReports/ 2002/ut-cs-02-489.ps.

[Mac02d] Bruce J. MacLennan. Replication, sharing, deletion, lists, and numerals: Progress on universally programmable intelligent matter — UPIM report 3. Technical Report CS-02-493, Dept. of Computer Science, University of Tennessee, Knoxville, 2002. Available at http://www.cs.utk.edu/ ~library/TechReports/ 2002/ut-cs-02-493.ps.

[Mac02e] Bruce J. MacLennan. Universally programmable intelligent matter (exploratory research proposal) — UPIM report 1. Technical Report CS-02-486, Dept. of Computer Science, University of Tennessee, Knoxville, 2002.

Available at http://www.cs.utk.edu/ ~library/TechReports/ 2002/ut-cs-02-486.ps.

[Mac02f] Bruce J. MacLennan. Universally programmable intelligent matter: Summary. In *IEEE Nano 2002*, pages 405–8. IEEE Press, 2002.

[Mac03a] Bruce J. MacLennan. Sensors, patches, pores, and channels: Progress on universally programmable intelligent matter — UPIM report 5. Technical Report forthcoming, Dept. of Computer Science, University of Tennessee, Knoxville, 2003.

[Mac03b] Bruce J. MacLennan. Combinatory logic for autonomous molecular computation. In *Proceedings, 7th Joint Conference on Information Sciences*, in press, 2003.

[Mac03c] Bruce J. MacLennan. Molecular combinatory computing for nanostructure synthesis and control. In *IEEE Nano 2003*. IEEE Press, in press, 2003.

[SSW91] Michel Simard, Dan Su, and James D. Wuest. Use of hydrogen bonds to control molecular aggregation. Self-assembly of three-dimensional networks with large chambers. *Journal of American Chemical Society*, 113(12):4696–4698, 1991.

[Yar00] Asim YarKhan. An investigation of random combinator soups. Technical report, Dept. of Computer Science, University of Tennessee, Knoxville, 2000. Unpublished report.

[YTM+00] Bernard Yurke, Andrew J. Turberfield, Allen P. Mills Jr, Friedrich C. Simmel, and Jennifer L. Neumann. A DNA-fuelled molecular machine made of DNA. *Nature*, 406:605–8, 2000.

[ZZRK96] S. C. Zimmerman, F. W. Zeng, D. E. C. Reichert, and S. V. Kolotuchin. Self-assembling dendrimers. *Science*, 271:1095, 1996.