Molecular Combinatory Computing for Nanostructure Synthesis and Control

Bruce MacLennan
Department of Computer Science
University of Tennessee
Knoxville, TN 37996–3450
Email: maclennan@cs.utk.edu

***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.**

I. 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 [1], a mathematical formalism based on network 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. 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. An additional advantage of this approach is that the substitutions can be done in any order or in parallel without affecting the result, which makes it an ideal model for autonomous molecular computation.

At IEEE-Nano 2002 we presented an overview of the strategy and potential of molecular combinatory computing [2]. In this summary, 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.

II. Overview

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, which are represented by the two daughters of the A node. The leaf nodes are molecular groups that function as primitive operations. As previously remarked, we primarily make use of two primitive combinators, K and S (which exists in two variants $\check{\mathsf{S}}$ and S proper). To understand these operations, consider a binary tree of the form $((\mathsf{K}X)Y)$, where X and Y are binary trees. (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 K group is released as a waste product, which may be recycled in later reactions. The tree Y is also a waste product, but it is 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 [3].)

The second primitive computational operator is S, which has the effect:

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

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. The properties of combinatory computing allow this replication to take place while other computation proceeds, even including use of the partially completed replicate.

A variant of the S operation is essential to molecular synthesis [3]:

$$(((\check{\mathsf{S}}X)Y)Z) \Longrightarrow ((XZ^{(1)})(YZ^{(0)})). \tag{3}$$

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. The principal purpose of the $\check{\mathsf{S}}$ operation is to synthesize non-tree-structured supramolecular networks.

Finally, we use the $\check{\mathsf{Y}}$ operator to create elementary cyclic structures, which can be expanded into larger cycles. It is defined [3]:

$$(\check{\mathsf{Y}}X) \Longrightarrow Z^{(1)} \quad \text{where } Z \equiv (FZ^{(0)}). \tag{4}$$

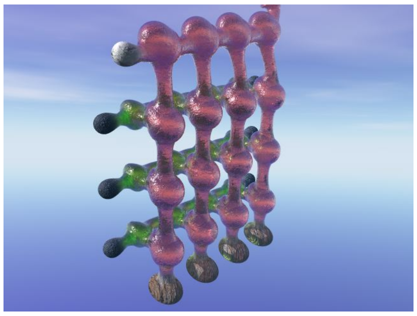

Fig. 1. Visualization of cross-linked membrane produced by $\mathsf{xgrid}_{3,4}\mathsf{NNN}$.

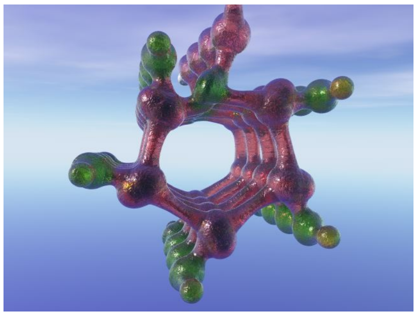

Fig. 2. Visualization of small nanotube, end view, produced by $\mathsf{xtube}_{5,4}\mathsf{N}$.

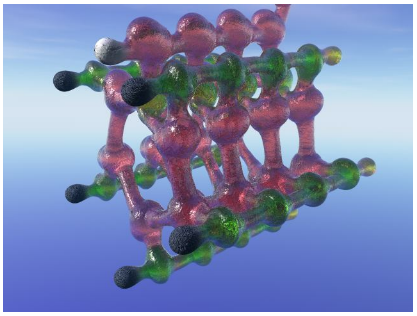

Fig. 3. Visualization of small nanotube, side view, produced by $\mathsf{xtube}_{5,4}\mathsf{N}$.

Examples of the use of both $\check{\mathsf{S}}$ and $\check{\mathsf{Y}}$ are given in Sec. III.

III. 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.

A. Membranes

For our first example we will discuss the synthesis of a cross-linked membrane, such as shown in Fig. 1. Such a structure is produced by the combinator program $\mathsf{xgrid}_{3,4}\mathsf{NNN}$, where xgrid is defined:

$$\mathsf{xgrid}_{m,n} = \mathsf{B}(\mathsf{B}(\mathsf{Z}_{m-1}\mathsf{W}))(\mathsf{B}(\mathsf{Z}_{n-1}\mathsf{W})(\mathsf{Z}_m\check{\Phi}_n)), \quad (5)$$

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 [4].

The size of $\mathsf{xgrid}_{m,n}$, the program structure to generate an $m \times n$ membrane, can be shown [4] to be $20m + 28n + 73$ primitive groups (AKS$\check{\mathsf{S}}$). 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 (5) 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 [3] for explanation). Similar compressions can be applied to the other parts dependent on m and n. Furthermore, as will be explained in Sec. III-C, large membranes can be synthesized by iterative assembly of small patches.

B. Nanotubes

Next we consider the synthesis of nanotubes, such as shown in Figs. 2 and 3. This is accomplished by using the $\check{\mathsf{Y}}$ combinator to construct a cycle between the upper and lower termini of the cross-linked membrane (Fig. 1). The program is [4]:

$$\mathsf{xtubep}_{m,n} = \check{\mathsf{W}}^{m-1}(\mathsf{W}^{n-1}(\check{\Phi}^m_n\mathsf{N})(\mathsf{B}^m\check{\mathsf{Y}}(\mathsf{C}_{[m]}\mathsf{I}))). \quad (6)$$

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

C. 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 [5] for details).

It is simple to assemble patches into larger structures. 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 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.

D. Active Elements

1) Pores and Active Channels: Space does not permit a detailed discussion of the synthesis of membranes with pores and channels [5]; 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.

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 write $\mathsf{K}_{-\lambda}$, which 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 (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 to be able to prevent sensor molecules from being prematurely triggered.

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. IV). However, we also know that these forces can combine additively, like individual muscle fibers in a muscle.

IV. Possible Molecular Implementation

A. Combinator Networks

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 [3].

2) Hydrogen-Bonded Covalent Subunits: Our current approach is to implement the nodes and linking groups by covalently-structured MBBs and to interconnect them by hydrogen bonds. We use a covalent framework for the nodes and links because they are stable and because there is an extensive synthetic precedent for engineering molecules of the required shape and with appropriately located hydrogen bonding sites [6]. 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. (Bond strengths are 2–40 kJ/mol.) Further, other laboratories have demonstrated the synthesis and manipulation of large hydrogen-bonded tree-like structures (*dendrimers*) [7], [8].

We estimate that two or three H-bonds will be necessary at each end of an L group. Therefore, if we take 20 kJ/mol as the strength of a typical H-bond, then the total connection strength 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, we are using 11 different node types (A, D, K, L, P, Q, R, S, Š, V, Ÿ), so it would seem that arrangements of 5 H-bonds would be sufficient (since they accommodate 16 complementary pairs of bond patterns).

A number of hydrogen-bonding sites can be located in a small area. For example, in DNA thymine (23 atoms) and adenine (26 atoms) have two H-bonds, while cytosine and guanine have three. Similarly, 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 100 atoms in size, L groups about 130, and ternary groups (A, V, R) about 180.

B. Substitution Reactions

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

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. 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. [9] 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. [10] 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.

3) Use of a Synthetic Substitutase: 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 MBBs makes it likely that the required enzyme-like compounds, which we call *substitutase molecules*, can be engineered.

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 [6]. 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. 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.

V. Conclusion

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 [3], [4], [11], [12].

Acknowledgment

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.

References

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

[2] B. J. MacLennan, "Universally programmable intelligent matter summary," in *IEEE Nano 2002*. IEEE Press, 2002, pp. 405–8.

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

[4] ——, "Membranes and nanotubes: Progress on universally programmable intelligent matter — UPIM report 4," Dept. of Computer Science, University of Tennessee, Knoxville, Tech. Rep. CS-02-495, 2002, available at http://www.cs.utk.edu/ ~library/TechReports/ 2002/ut-cs-02-495.ps.

[5] ——, "Sensors, patches, pores, and channels: Progress on universally programmable intelligent matter — UPIM report 5," Dept. of Computer Science, University of Tennessee, Knoxville, Tech. Rep. forthcoming, 2003.

[6] D. G. Allis and J. T. Spencer, "Nanostructural architectures from molecular building blocks," in *Handbook of Nanoscience, Engineering, and Technology*, W. A. Goddard, D. W. Brenner, S. E. Lyshevski, and G. J. Iafrate, Eds. CRC Press, 2003, ch. 16.

[7] M. Simard, D. Su, and J. D. Wuest, "Use of hydrogen bonds to control molecular aggregation. Self-assembly of three-dimensional networks with large chambers," *Journal of American Chemical Society*, vol. 113, no. 12, pp. 4696–4698, 1991.

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

[9] N. Koumura, R. W. J. Zijlstra, R. A. van Delden, N. Harada, and B. L. Feringa, "Light-drive monodirectional molecular rotor," *Nature*, vol. 401, pp. 152–5, 1999.

[10] B. Yurke, A. J. Turberfield, A. P. Mills Jr, F. C. Simmel, and J. L. Neumann, "A DNA-fuelled molecular machine made of DNA," *Nature*, vol. 406, pp. 605–8, 2000.

[11] B. J. MacLennan, "Universally programmable intelligent matter (exploratory research proposal) — UPIM report 1," Dept. of Computer Science, University of Tennessee, Knoxville, Tech. Rep. CS-02-486, 2002, available at http://www.cs.utk.edu/ ~library/TechReports/ 2002/ut-cs-02-486.ps.

[12] ——, "Molecular combinator reference manual — UPIM report 2," Dept. of Computer Science, University of Tennessee, Knoxville, Tech. Rep. CS-02-489, 2002, available at http://www.cs.utk.edu/ ~library/TechReports/ 2002/ut-cs-02-489.ps.