Combinatory Logic For Autonomous Molecular Computation

Bruce J. MacLennan
Department of Computer Science, University of Tennessee, Knoxville

1 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 [2], which are sufficient for any computation. 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* [2, 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.

2 Combinatory Computing

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 K and is described by the substitution rule:

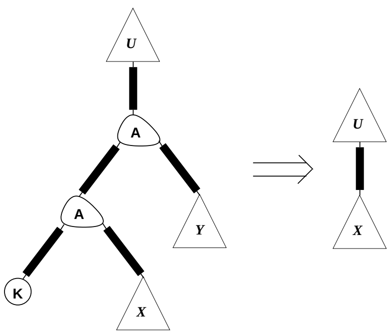

Figure 1: K combinator substitution operation.

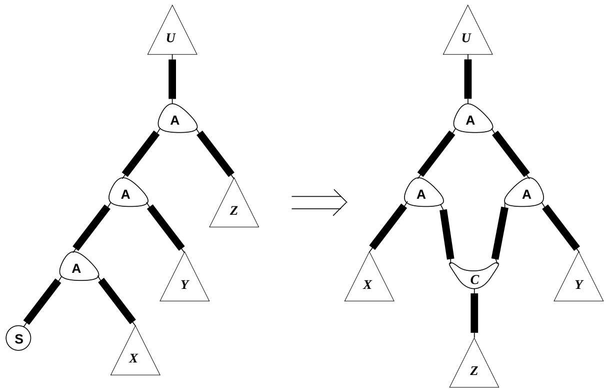

Figure 2: S and Š combinator substitution operations.

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

Here X and Y represent arbitrary binary trees and K is a leaf labeled K; parentheses group the subtrees of an interior node (which we designate A). The interpretation of the rule is that wherever a subtree of the form $((\mathsf{K}X)Y)$ is found in the network, it may be replaced by X. This reaction is depicted in Fig. 1, which also illustrates its similarity to a molecular substitution. The effect of the operation is to delete Y from the network.

The second primitive operation is described by the rule:

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

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 Š (sharing). The molecular implementations of the two are very similar (see Fig. 2). If $C = \mathsf{V}$ (a *sharing node*), then we have two links to a shared

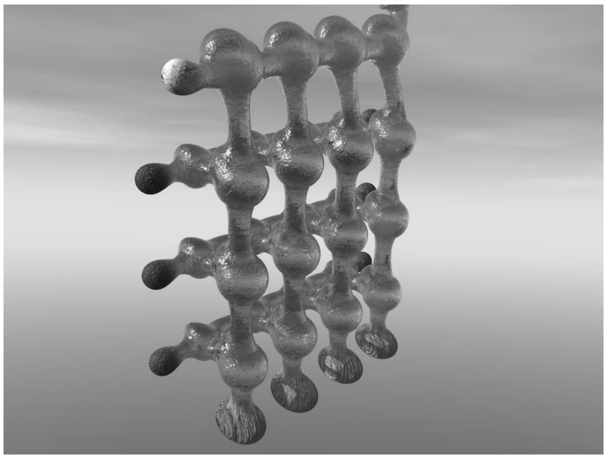

Figure 3: Visualization of cross-linked membrane.

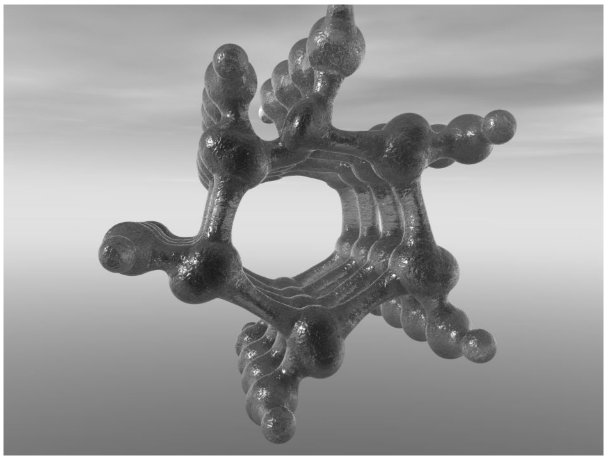

Figure 4: Visualization of small cross-linked nanotube.

copy of Z. On the other hand, 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. It can be shown that computation can proceed in parallel with replication without affecting the results of the process.

Although K and S are sufficient for all computation, they cannot (even with $\check{\mathsf{S}}$) create cyclic structures, for which we use an additional primitive operation $\check{\mathsf{Y}}$, which creates a self-referential link through a V node [6].

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 non-computational 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 [8]. 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.

3 Nanostructure Synthesis and Control

For a first example of nanostructure synthesis, we can consider a molecular combinatory program to assemble a membrane such as shown in Fig. 3. This is produced by $\mathsf{xgrid}_{3,4}\mathsf{NNN}$ (where N is any inert group); $\mathsf{xgrid}_{m,n}$, which computes an $m \times n$ membrane, is defined:

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

Unfortunately, space does not permit an explanation of this program, which may be found, with correctness proofs, is a prior report [4]. The above definition makes use of various combinators (B, W, etc.), which are defined in terms of K, S, or $\check{\mathsf{S}}$. When all these definitions are expanded, $\mathsf{xgrid}_{m,n}$ is found to be a binary tree of size $20m + 28n + 73$ primitive groups $(\mathsf{A}, \mathsf{K}, \mathsf{S}, \check{\mathsf{S}})$ [4]. Therefore, the program for an $m \times n$ membrane is of size $\mathcal{O}(m + n)$. Perhaps remarkably, standard recoding techniques [4, 6] can be successively applied to reduce this to $\mathcal{O}(\log m + \log n)$, to $\mathcal{O}(\log\log m + \log\log n)$, etc.

We have also developed combinatory programs for other membrane architectures, such as hexagonal grids [4]. Further, while the "mesh size" of these membranes is determined by the length of the link group, simple variants of the programs produce meshes that are multiples of this fundamental length [8].

The $\check{\mathsf{Y}}$ (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 Fig. 4 [4]. This is created by $\mathsf{xtube}_{5,4}\mathsf{N}$, where

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

The size of this program is $102m + 44n - 96$ primitive groups.

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" [8]. Any such patch may be joined either horizontally or vertically with another patch of compatible dimensions, to yield a patch combining the two. In this way patches may be hierarchically assembled into larger patches. Furthermore, combinatory computing permits the patch assembly operations to 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. Similar techniques may be used to assemble heterogeneous nanotubes from smaller segments, Y-connectors, etc.

Rather than computing to a stable state, *dynamic structures* remain potentially active, ready to respond to environmental conditions [8]. An example is a channel in a membrane, which may open or close in response to a change in the environment. 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 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 Sec. 5), but they can work cooperatively to generate larger forces.

4 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 **true** or **false** (usually K and (SK)). Therefore, by producing enough replicates of P and a sufficient variety of potential solutions X, we may evaluate the potential solutions with molar parallelism.

5 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?

We are investigating the representation of 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 structural reorganization. (Hydrogen bond strength is in the range 2–40 kJ/mol.) Hydrogen bonds are 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 [9, 11].

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 their design [1].

It appears that our primitive groups (A, K, R, S, $\check{\mathsf{S}}$, V, $\check{\mathsf{Y}}$ and the links) 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 the size of our building blocks might be 90 atoms ($\mathsf{K}, \mathsf{S}, \check{\mathsf{S}}, \check{\mathsf{Y}}$) to 150 atoms ($\mathsf{A}, \mathsf{R}, \mathsf{V}$).

To implement the substitutions 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 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. We are investigating the use of graded electrostatic potentials to effect this relocation. The final molecule, a *synthesase*, recognizes the configuration created by the permutase, and binds to the waste structures, releasing the desired product from the permutase. For example, S- or $\check{\mathsf{S}}$-synthesase will remove the S- or $\check{\mathsf{S}}$-permutase from the structure on the right in Fig. 2. Here again we are depending on the extensive synthetic precedent for covalently-structured molecules with strategically located hydrogen bonding sites.

It is necessary to point out that 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 demonstrations of continuously operating nanomachines driven by electromagnetic radiation or chemical fuel [3, 10], which demonstrate the feasibility of nonterminating reactions. In our case, the most likely sources of fuel are the various species of analysase, permutase, and synthesase molecules.

6 Conclusions

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 have given several examples of the use of molecular combinatory programming to create useful nanostructures. We also presented a possible implementation based on enzyme-mediated reorganization of large networks of molecular building blocks linked by hydrogen bonds. Additional information about this project can be found in reports and articles archived at the project website: `http://www.cs.utk.edu/~mclennan/UPIM`.

7 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.

References

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

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

[3] N. Koumura, R. W. J. Zijlstra, R. A. van Delden, N. Harada, and B. L. Feringa. Light-driven monodirectional molecular rotor. *Nature*, 401:152–5, 1999.

[4] B. J. MacLennan. Membranes and nanotubes: UPIM report 4. Tech. Rep. CS-02-495, Dept. of Computer Science, Univ. of Tennessee, Knoxville, 2002.

[5] —. Molecular combinator reference manual. Tech. Rep., Dept. of Computer Science, Univ. of Tennessee, Knoxville, 2002.

[6] —. Replication, sharing, deletion, lists, and numerals: UPIM report 3. Tech. Rep. CS-02-493, Dept. of Computer Science, Univ. of Tennessee, Knoxville, 2002.

[7] —. Universally programmable intelligent matter (exploratory research proposal): UPIM report 1. Tech. Rep. CS-02-486, Dept. of Computer Science, Univ. of Tennessee, Knoxville, 2002.

[8] —. Sensors, patches, pores, and channels: UPIM report 5. Tech. Rep. forthcoming, Dept. of Computer Science, Univ. of Tennessee, Knoxville, 2003.

[9] 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. *J. of American Chemical Society*, 113(12):4696–4698, 1991.

[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*, 406:605–8, 2000.

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