Universally Programmable Intelligent Matter
Summary

Bruce J. MacLennan
Department of Computer Science, University of Tennessee, Knoxville, TN 37996, USA

***Abstract* — We explain how a small set of molecular building blocks will allow the implementation of "universally programmable intelligent matter," that is, matter whose structure, properties, and behavior can be programmed, quite literally, at the molecular level.**

I. Definitions

Intelligent matter is any material in which individual molecules or supra-molecular clusters function as agents to accomplish some purpose. Intelligent matter may be solid, liquid, or gaseous, although liquids and membranes are perhaps most typical. *Universally programmable* intelligent matter (UPIM) is made from a small set of molecular building blocks that are universal in the sense that they can be rearranged to accomplish any purpose that can be described by a computer program. In effect, a computer program controls the behavior of the material at the molecular level. In some applications the molecules self-assemble a desired nanostructure by "computing" the structure and then becoming inactive. In other applications the material remains active so that it can respond, at the molecular level, to its environment or to other external conditions. An extreme case is when programmable supra-molecular clusters act as autonomous agents to achieve some end.

Although materials may be engineered for specific purposes, we will get much greater technological leverage by designing a "universal material" which, like a general-purpose computer, can be "programmed" for a wide range of applications. To accomplish this, we must identify a set of molecular primitives that can be combined for widely varying purposes. The existence of such universal molecular operations might seem highly unlikely, but there is suggestive evidence that it may be possible to discover or synthesize them.

II. Approach

Accomplishing the goals of UPIM will require the identification of a small set of molecular building blocks that is computationally universal. The SK *calculus* (a kind of combinatory logic [1]-[4]) is a formal system that demonstrates that such sets exist. It is capable of universal computation, but makes use of only two simple operations on networks (graphs), which are suggestive of molecular processes. Computer scientists have investigated the SK calculus extensively for several decades as a basis for massively parallel computer architectures, and the translation of high-level functional computer programs into SK structures is well understood [5]-[8]. Although the SK calculus may not be the best choice for programmable intelligent matter, it is a place to start.

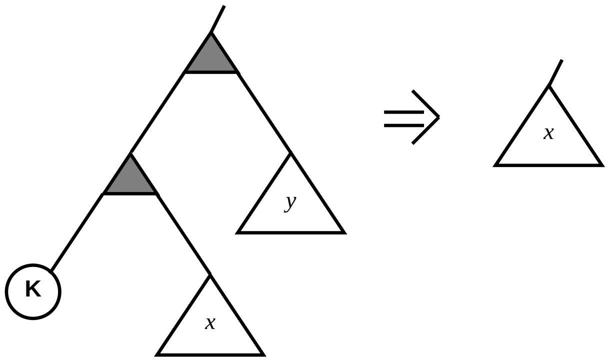

Figure 1: K-substitution.

The SK calculus is defined by two simple substitution rules. The K-substitution is expressed by this rewrite rule,

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

which describes the transformation shown in Fig. 1, in which X and Y represent any networks. In effect, since the value of $(\mathsf{K}X)$, when applied to any Y, is X, the K operation, when applied to X yields the constant function $(\mathsf{K}X)$. This is the interpretation, but the computational effect is entirely expressed in the substitution in Fig. 1.

It will be apparent that this substitution rule suggests a molecular process, but the equivalent depiction in Fig. 2 makes the similarity more apparent. It can be put in the style of a chemical reaction, including reaction resources and

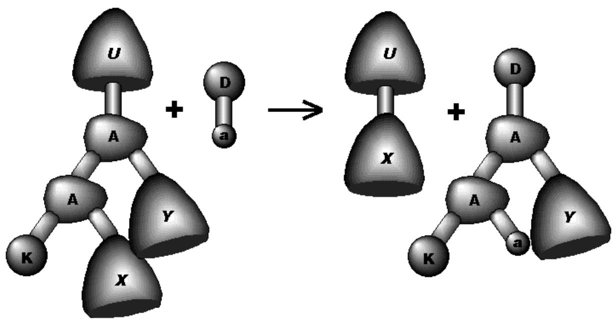

Figure 2: K-substitution as a molecular process.

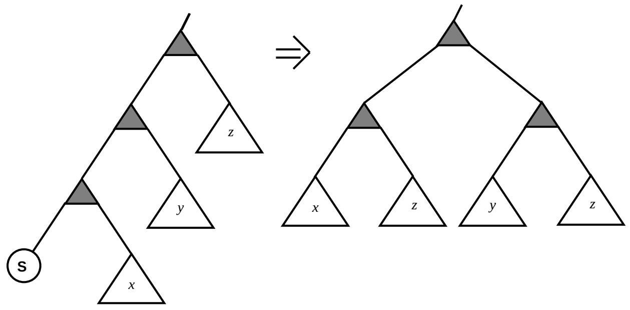

Figure 3: S-substitution with copying.

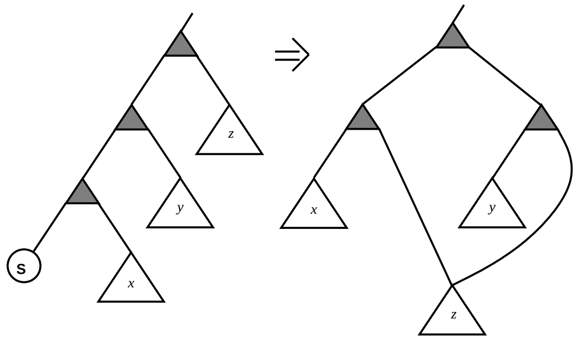

Figure 4: S-substitution with sharing.

waste products:

$$U\mathsf{A}_2\mathsf{K}XY + \mathsf{Da} \longrightarrow UX + \mathsf{DA}_2\mathsf{Ka}Y.$$

Here A, K, D, and a are functional groups, and U, X, and Y represent arbitrary molecular networks. D is a disposal operator and a is a computationally inert place-holding group.

The S operator is only slightly more complicated; it is defined by the rewrite rule,

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

There are two ways of interpreting it as a network substitution, depending on whether we make a new copy of Z (Fig. 3) or share a single copy (Fig. 4). However, the Church-Rosser property [9]-[10] shows that the two interpretations lead to the same computational result, but the interpretations have practical differences, which cannot be addressed in this summary.

It is important to stress the significance of the SK calculus: these two simple operations are capable of computing anything that can be computed on any digital computer. This is certainly remarkable, and so it is surprising that there are quite a few other universal sets of combinators. There are even some general guidelines [2, sec. 5H] for universality (i.e., the combinators must be able to delete, duplicate, and permute). The existence of multiple universal sets is very fortunate, because it implies that when we begin to search for molecular implementations of these operations, we will have a greater probability of finding reactions implementing at least one universal set of substitutions.

The combination of the parallel computation permitted by the Church-Rosser property and the simplicity of the SK calculus has led computer scientists to investigate it as a basis for parallel computer architecture [6]-[7]. There are simple algorithms for translating functional computer programs into SK networks, and considerable effort has been devoted to optimizing them. Therefore, if we can identify molecular processes corresponding to a universal set of combinators (SK, for example), then we can at least see the possibility of writing a computer program and translating it into a molecular process.

III. Extensions

To expand the range of application of UPIM and for other practical purposes, it is advisable to extend the set of primitive operations beyond those minimally necessary for computational universality (e.g., S and K). First, we might want to add *sensor operations* that respond differently in different environmental conditions. For example, they might be sensitive to light or to the presence of some chemical. The results of these tests could be used to control conditional execution of the program. In addition to such external input to the program, it is also useful to have means for external output, which can be accomplished with *effector operations*. These reactions, when they take place, cause some noncomputational effect, such as the release of a chemical, the emission of light, or physical motion. They are one of the ways that intelligent matter can have an effect beyond its own internal computational reconfiguration.

Another issue that must be addressed is the production of a molecular combinator network (e.g., an SK tree) from a macroscopic program, and the subsequent replication of a large number of copies. Although the best approach is one of the objectives of our research, a possible method can be presented at this time. Arbitrary combinator trees can be represented uniquely as parenthesized strings, such as "$(((\mathsf{SK})\mathsf{S})(\mathsf{SK}))$." Therefore, such a string could be encoded by chain of four molecular groups (s, k, p, q), such as "$pppskqsqpskqq$" for the previous example. Thus we proceed in stages. The program is compiled into SK trees (or other combinators); the trees are flattened into parenthesized strings; and the strings are encoded in molecular chain structures (e.g., DNA sequences), which are synthesized and replicated by standard techniques from biotechnology. The replicated program chains are converted back into (now molecular) networks by a simple set of substitution rules, implemented chemically.

IV. Applications

Finally, it will be worthwhile to discuss briefly some of the possible applications of UPIM, which may be static or dynamic (or interactive). By a *static* application we mean one in which the intelligent matter computes into an equilibrium state, and is inactive thereafter. Therefore static applications are most often directed toward generating some specialized material with a computationally defined nanostructure. On the other hand, *dynamic* or *interactive* applications never terminate, but always remain ready to respond to their environment in some specified way; they are the truly "smart" materials.

A. Static Applications

Programs are ideally suited to creating complex data structures, which can be converted to complex physical structures by means of UPIM. Networks, chains, tubes, spheres, fibers, and quasi crystalline structures are all straightforward to compute. The network resulting from such a computation will be composed of computational groups (e.g., $\mathsf{S}, \mathsf{K}, \mathsf{A}$) as well as inert groups, which are manipulated by the computation but do not affect it. Typically, in these applications the computational phase will be followed by a chemical phase in which the computational groups are replaced by substances appropriate to the application (a sort of "petrification"). In addition to the examples already mentioned, such an approach could be used to synthesize membranes with pores or channels of a specified size and arrangement (determined either deterministically by the program or stochastically by molecular processes).

A number of applications are suggested by the requirements of implementing small, autonomous robots. Some of these will be controlled by very dense analog neural networks, but to achieve densities comparable to mammalian cortex (15 million neurons per square cm., with up to several hundreds of thousands of connections each), we will need to be able to grow intricately branching dendritic trees at the nanoscale. Generation of such structures is straightforward with UPIM (e.g., using L-systems [11]). The sensor and effector organs of microrobots will also require very fine structures, which UPIM can be programmed to generate.

Of course, we should not neglect the potential of UPIM to do conventional computation, such as solving NP-complete problems by massively parallel computation. For example, we might replicate many copies of a program to test a potential solution, then mix them in a reaction vessel with structures representing possible solutions, and wait for equilibrium to determine actual solutions. The advantage of our approach to this kind of search problem over others, such as DNA computation, is that our nanoscale test molecules are programmable.

B. Dynamic Applications

Dynamic intelligent matter is interactive in the sense that it is continually monitoring its environment and capable of responding according to its program. That is, it is in a state of temporary equilibrium, which can be disrupted by changes in the environment, resulting in further computation and behavior as the material seeks a new equilibrium.

For example, a membrane with channels, such as mentioned above, could be made active by having the channels open or close in response to environmental conditions, including control commands transmitted optically or chemically. The program located in each channel is simple: in response to its sensor state it executes one or the other of two effectors, blocking the channel or not. The sensor and the medium would determine whether the channel is sensitive to global conditions (e.g., overall chemical environment or ambient illumination) or to its local environment (e.g., molecules or light in its immediate vicinity).

Similarly, unanchored or free-floating molecular clusters (e.g., in colloidal suspension) may react to their environment and change their configuration, thus affecting physical properties of the substance, such as viscosity or transparency. Or they might polymerize or depolymerize on command.

Unanchored supramolecular networks might also operate as semiautonomous agents to recognize molecules or molecular configurations, and act upon them in some intended way (e.g. binding toxins or pollutants). However, such applications will require the agents to operate in a medium that can supply the reactants needed for computation.

These sorts of active intelligent matter will find many applications in autonomous microrobots. For example, active membranes can serve as sensory transducers, responding to conditions in the environment and generating electrical, chemical, or other signals. They can also be programmed to self-organize into structures capable of preprocessing the input (e.g., artificial retinas or cochleas). Further, it is a simple modification of a membrane with channels to make a membrane with cilia that flex on command. By means of local communication, the cilia may be made to flex in coordinated patterns. Similarly we may fabricate artificial muscles, which contract or relax by the coordinated action of microscopic fibers. UPIM may also provide a systematic approach to self-repair of autonomous robots and other systems, since if a robot's "tissues" were created by computational processes, then they can remain potentially active, ready to restore an equilibrium disrupted by damage. Less ambitiously, materials can be programmed to signal damage or other abnormal conditions.

Acknowledgement

Preparation of this report has been facilitated by a grant from the University of Tennessee, Knoxville, Center for Information Technology Research.

References

[1] H. B. Curry, "Grundlagen der kombinatorischen Logik," *American Journal of Mathematics*, vol. 52, pp. 509–536, 789–834, 1930.

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

[3] J. R. Hindley, B. Lercher, and J. P. Seldin, *Introduction to Combinatory Logic*, Cambridge: Cambridge University Press, 1972.

[4] M. Schönfinkel, "Über die Bausteine der mathematischen Logik," *Math. Annalen*, vol. 92, pp. 305–316, 1924.

[5] S. K. Abdali, "An abstraction algorithm for combinatory logic," *Journal of Symbolic Logic*, vol. 41, no. 1, pp. 222–224, March 1976.

[6] J. H. Fasel and R. M. Keller (eds.), *Graph Reduction, Proceedings of a Workshop, Santa Fe, New Mexico, USA, September 29 – October 1, 1986*, Berlin: Springer Verlag, 1987.

[7] D. A. Turner, "A new implementation technique for applicative languages," *Software — Practice and Experience*, vol. 9, pp. 31–49, 1979.

[8] B. J. MacLennan. *Functional Programming: Practice and Theory*, Reading: Addison-Wesley, 1990.

[9] A. Church and J. B. Rosser, "Some properties of conversion," *Trans. American Math. Soc.*, vol. 39, pp. 472–482, 1936.

[10] B. K. Rosen, "Tree manipulation systems and Church-Rosser theorems," *Journal of the ACM*, vol. 20, no. 1, pp. 160–187, January 1973.

[11] A. Lindenmeyer and P. Prusinkiewicz, "Developmental models of multicellular organisms: A computer graphics perspective," In C. G. Langton, editor, *Artificial Life*, volume VI of *SFI Studies in the Sciences of Complexity*, pages 221–250, Redwood City: Addison-Wesley, 1989.