# Universally Programmable Intelligent Matter (Exploratory Research Proposal)

UPIM Report 1

Technical Report UT-CS-02-486

Bruce J. MacLennan*

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

October 18, 2002

**Abstract**

*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. This report outlines a one-year exploratory research project to investigate the feasibility of UPIM.

## 1 Project Summary

*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

*This report may be used for any non-profit purpose provided that the source is credited.

or gaseous, although liquids and membranes are perhaps most typical. *Universally programmable* intelligent matter 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.

Accomplishing the goals of universally programmable intelligent matter will require the identification of a small set of molecular building blocks that is computationally universal. The *SK calculus* (a kind of combinatory logic) 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 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. However, the SK calculus may not be the best choice for programmable intelligent matter.

This exploratory research project has four principal objectives: (1) to develop a model of computation compatible with the constraints of molecular processes; (2) to identify at least two universal sets of building blocks for programmable intelligent matter; (3) to develop methods for interfacing with additional molecular building blocks for sensing conditions and causing effects in the external environment; (4) to develop prototype simulation software to investigate characteristics peculiar to molecular computation.

Some of the methods are theoretical: (1) the construction of a mathematical model of computation compatible with the constraints of molecular processes, and (2) a mathematical investigation of the properties (such as computational universality) of some simple graph operations resulting in at least two universal sets of building blocks. The theoretical investigation will be supplemented by (3) the development of simulation software to investigate stochastic and other novel factors affecting computation in a molecular context, and (4) the use of the simulator to demonstrate the use of programmable intelligent matter to implement several useful nanostructures, such as nanotubes and membranes with active channels and cilia.

Since the resulting building blocks for universal programmable matter are expected to be individually simple and few in number, this project will provide the information needed by chemists to identify or synthesize the substances sufficient to implement universally programmable intelligent matter. This will open the way toward the ability to produce materials with a desired nanostructure and behavior as easily as we program computers today. This project will take a first step toward a systematic approach to nanotechnology that will facilitate its rapid development.

# 2 Background

## 2.1 Approach to Universally Programmable Intelligent Matter

*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. In some applications the molecules self-assemble a desired nanostructure by “computing” the structure and then becoming inactive. In other, “interactive” 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. In particular, *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. 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.

*Term-rewriting systems* [10, 18] are simple computational systems in which networks are altered according to simple rewrite rules, which describe *substitutions* that have much in common with abstract chemical reactions. (One particular term-rewriting system, the lambda calculus, has been used already to model prebiotic chemical evolution [6, 7, 8].) Term-rewriting systems have been extensively investigated by mathematicians and computer scientists for several decades [4, 5].

One attractive feature of term-rewriting systems is that many of them have the *Church-Rosser property* [2], which means, roughly, that substitutions can be done in any order without affecting the computational result [4, ch. 4]. Therefore these term-rewriting systems have been investigated as a possible basis for massively-parallel computer architectures [5, 20]. This is an important property for a model of molecular computation, in which molecular processes take place stochastically.

One class of term-rewriting systems, the *combinatory logic systems* [3, 4, 9, 19], is very relevant for programmable intelligent matter, for it has been known since the early twentieth century that there are several small sets of substitution operations that can be used to program any Turing-computable function. One such *universal* set is the $\mathsf{SK}$ calculus, which comprises two simple substitution rules. The $\mathsf{K}$-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 trees.

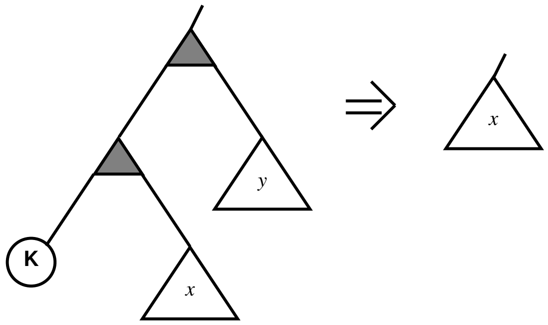


Figure 1: $\mathsf{K}$-Substitution

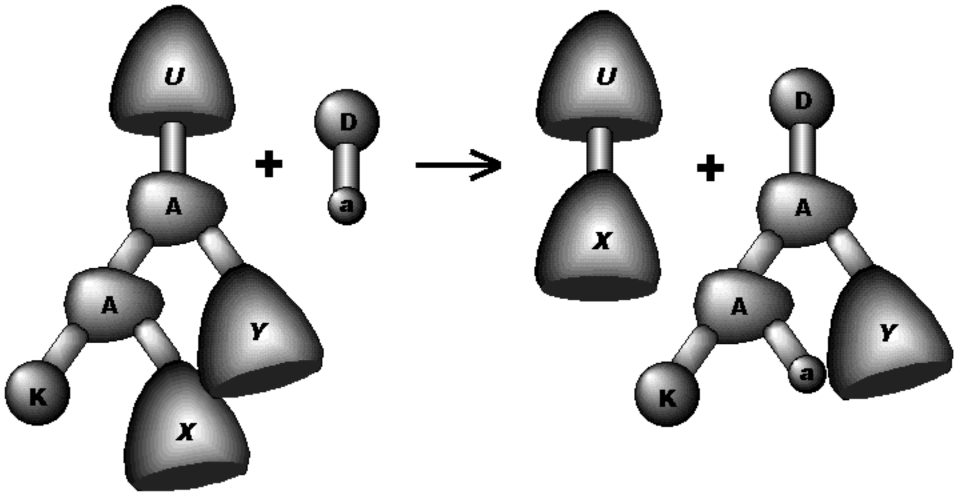


Figure 2: $\mathsf{K}$-Substitution as a Molecular Process

In effect, since the value of $(\mathsf{K}X)$, when applied to any $Y$, is $X$, the $\mathsf{K}$ 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 waste products:

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

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

The $\mathsf{S}$ 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). An important consequence of

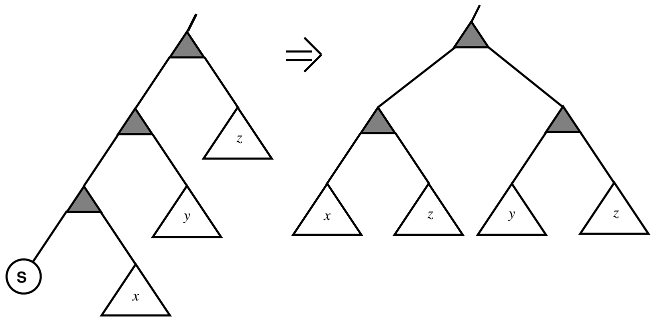


Figure 3: S-Substitution with Copying

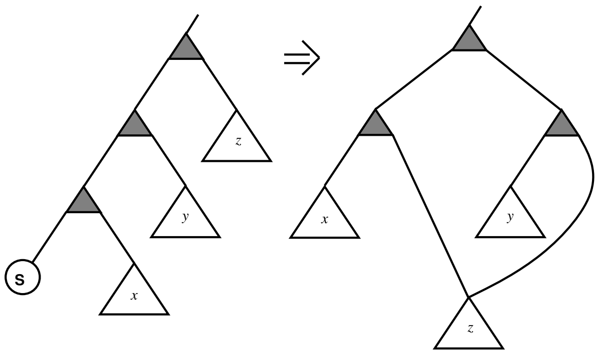


Figure 4: S-Substitution with Sharing

the Church-Rosser property is that the two interpretations lead to the same computational result, but the interpretations have practical differences, which are discussed later.

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. For example, the set of K with B′ and W is universal, where the latter two operators are defined,

$$\begin{aligned} (((\mathsf{B}'X)Y)Z) &\implies (Y(XZ)), \\ ((\mathsf{W}X)Y) &\implies ((XY)Y). \end{aligned}$$

A third universal set comprises K and these three combinators:

$$\begin{aligned} (((\mathsf{B}X)Y)Z) &\implies (X(YZ)), \\ ((\mathsf{C}^*X)Y) &\implies (YX), \\ (\mathsf{W}^*X) &\implies (XX). \end{aligned}$$

There are also other universal sets and, indeed, some general guidelines [4, 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 [5, 20]. 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.

To illustrate the idea, we will present, with little explanation, a program for computing a nanotube. A single ring of the nanotube is computed by:

$$\begin{aligned} \mathsf{Ring}(X, Y) &= R, \\ \text{where rec } R &= \mathsf{RingAux}(X, Y, R), \\ \mathsf{RingAux}(X, \mathsf{nil}, R) &= R, \\ \mathsf{RingAux}(x : X, y : Y, R) &= (x, y) : \mathsf{RingAux}(X, Y, R). \end{aligned}$$

The function is given two molecular chains, $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_m)$. It creates a ring of the $x_i$, each of which linked to the next $x$ group in the ring, as well as to the corresponding $y_i$ (see Fig. 5). A nanotube is computed by creating a series of linked rings:

$$\begin{aligned} \mathsf{Tube}(\mathsf{nil}, X, Y) &= \mathsf{Ring}(X, Y), \\ \mathsf{Tube}(a : N, X, Y) &= \mathsf{Ring}[X, \mathsf{Tube}(N, X, Y)]. \end{aligned}$$

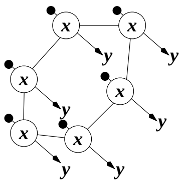


Figure 5: Example: Single Ring of a Nanotube

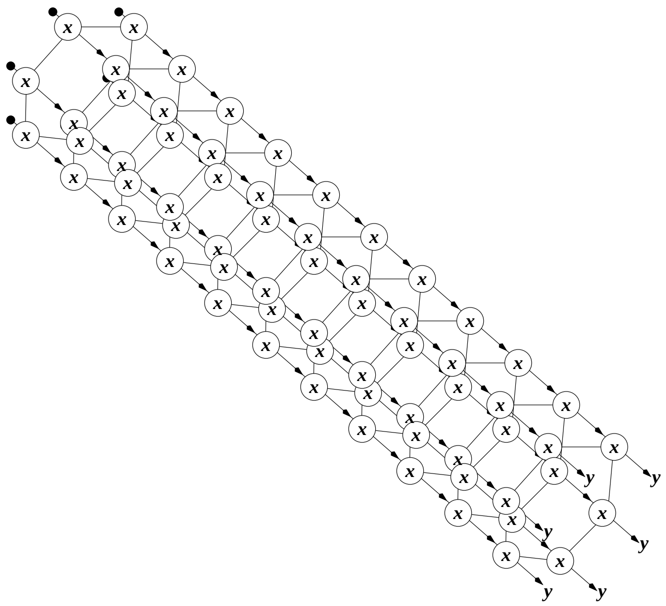

Figure 6: Example: Nanotube Synthesized by UPIM

If $N = (a_1, \ldots, a_{n-1})$ is any chain of length $n - 1$, then $\mathsf{Tube}(N, X, Y)$ will compute a nanotube of length $n$, with $x$ groups forming the sides of the tube, and $y$ groups forming its terminus (Fig. 6). Our discussion of this example is of necessity short, but it will serve to demonstrate how simple programs can generate useful nanostructures.

Intriguing as the SK calculus is as a basis for universally programmable intelligent matter, it also illustrates some of the problems that the proposed research is intended to address. As previously mentioned, term-rewriting systems assume that a network can be copied for free, as illustrated in Fig. 3. This is certainly a bad assumption for molecular computation, in which the time to replicate a structure is at least proportional to the logarithm of its size. It is also extremely wasteful, since when programs are compiled to use only the S and K combinators, one often observes that an S operation replicates a structure, which is almost immediately discarded by a K. There are ways to avoid much of this needless replication (at the expense of introducing additional primitive combinators), but considerable replication will remain.

The obvious solution is to use the sharing implementation (Fig. 4), since this does not require any copying. It is the solution adopted in many implementations of SK on conventional computers, in which one may have any number of pointers to a single data structure. However, this option does not seem to be possible in molecular computing, in which each link connects only two groups of atoms.

There are some possible solutions to this problem, but they will not be discussed at this time. We raise the copying/sharing issue to show that the constraints of molecular computing are different from those of electronic computing. Therefore, while term-rewriting systems, combinatory logic, and the SK calculus in particular are suggestive of how universally programmable intelligent matter might be implemented, we must be prepared to develop new models of computation that are compatible with the constraints of molecular processes. Developing such a model is a principal objective of the proposed research.

To extend the range of application of universally programmable intelligent matter 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). For example, 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.

To some extent the sensors and effectors are ad hoc additions to the basic computational framework (e.g., SK). However, they are fundamentally incompatible with it in one sense, for the time when their reactions take place is usually important. They are termed *imperative* operations and do not have the Church-Rosser property. Therefore, programs incorporating them must have means for controlling the time of their execution. Fortunately, these issues have been addressed long ago in the design and implementation of functional programming languages [12, and references therein], and the results of those investigations can be applied to universally programmable intelligent matter.

An issue not previously 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 finding a solution to this problem is one of the objectives of the proposed research project, a possible solution 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 genetic engineering. The replicated program chains are converted back into (now molecular) networks by a simple set of substitution rules, implemented chemically. (The problem of replication error is discussed later.)

## 2.2 Potential of Universally Programmable Intelligent Matter

Before presenting the specific objectives of this research, it will be worthwhile to discuss briefly some of its potential applications. First, however, it is necessary to contrast universally programmable intelligent matter with some other more or less related ideas.

*Programmable matter* [14, 15, 16, 17] is an approach to computation based on lattice-like arrays of simple computational elements; cellular automata (such as Conway's "game of life") are examples. Although techniques from the "programmable matter paradigm" certainly will be applicable to universally programmable intelligent matter, there are differences in objective: programmable matter seems to be intended primarily for implementation on electronic digital computers, and computational universality does not seem to be a goal.

*Complex adaptive matter* (CAM) has been under investigation at Los Alamos National Laboratory as an approach to adapting matter or materials to a desired functionality by a quasi-evolutionary process comprising amplification, noise or variation, and filtering. Again, the goals are different, but there are several intersections with the proposed research. First, like ours, their focus is on molecular processes rather than electronic computation. Second, CAM techniques might be used for synthesizing functional groups implementing universal sets of molecular operators. Third, such a universal set might provide building blocks for the quasi-evolutionary CAM process. (Indeed, we have already begun investigating statistical properties of "soups" of SK complexes, which might be used as the raw materials of the CAM process [13, 21].)

*Smart Matter* is being developed at the Xerox Palo Alto Research Center. Its long-term goals are quite similar to those of intelligent matter, as defined in this proposal. However, the current focus seems to be on small (but not molecular) building blocks that combine sensor, computation, and actuation functions. That is, the goal is MEMS (Micro-Electro-Mechanical Systems) rather than nanotechnology. Certainly, however, intelligent matter, in our sense, will benefit from the more general techniques of distributed control and embedded computation that might come out of the Smart Matter project.

Returning now to possible applications of universally programmable intelligent matter, these applications 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. We will briefly mention a few potential applications of each kind.

Programs are ideally suited to creating complex data structures, which can be converted to complex physical structures by means of universally programmable intelligent matter. 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 universally programmable intelligent matter 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.

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. Universally programmable intelligent matter 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.

To tie the foregoing ideas together we may present the typical process of developing an application of universally programmable intelligent matter.

1. Write a program in an appropriate high-level programming language to create the desired nanostructure or to exhibit the desired interactive behavior at the nanoscale. Debug and simulate the execution of the program on a conventional computer.

2. Compile the program into a combinator tree (e.g., a network of $\mathsf{S}$, $\mathsf{K}$ and other combinators).

3. Simulate (on a conventional computer) the substitutions on the network, but subject to molecular constraints (e.g., including reactant concentrations, substitution errors, etc.).

4. On a computer, flatten the combinator tree into a string representing a sequence of DNA bases.

5. Use this string to guide the synthesis of a DNA sequence.
6. Replicate the DNA sequence until the required number of copies of the program are produced.
7. Use the translation or tree-building substitutions to construct a molecular combinator tree from each DNA string. (An intermediate RNA stage could be used, if required.)
8. Supply reactants for the computational substitutions (e.g., $\mathsf{S}, \mathsf{K}, \mathsf{A}, \mathsf{D}$ groups), and allow the reaction to proceed to equilibrium.
9. If the application is static, wash out or otherwise eliminate any remaining reaction waste products.
10. If the application is static, substitute permanent replacement groups for computational groups by ordinary chemical processes (if required by the application).

## 2.3 Problems

In this section we will summarize some of the problems that need to be solved in order to make universally programmable intelligent matter a reality, and which therefore define the objectives for the proposed exploratory research project.

We need a model of computation that respects the constraints of molecular processes. For example, as explained above, our model cannot assume that replication is free, or that a functional group can be linked to ("pointed at") by an unlimited number of other groups. Such a model of molecular computation might be a modification of the network-substitution model, or it might be a completely different model. (For example, we are investigating a model based on permutation of link participants.)

Assuming that we stay with something like the combinatory logic network-substitution model, then we need a solution to the replication/sharing problem. One promising solution, which we have been investigating, is to implement a "lazy" replication operation. With such an operation the two "copies" can begin to be used even before the replication is complete. (The Church-Rosser property guarantees the safety of such simultaneous use and replication.) Discarding such a partially replicated structure causes the replication process to be prematurely terminated, thus decreasing wasted resources.

We need one or more universal sets of molecular primitives. Practical experience with combinator programming has shown that it is usually more efficient to use more than the theoretically minimum set of combinators. For example, by including the identity function or $\mathsf{I}$ combinator $[(\mathsf{I}X) \Longrightarrow X]$, many of the self-canceling replications and deletions can be eliminated. However, we will have to keep in mind that the efficiency tradeoffs of molecular computing are not the same as those of conventional computing (e.g., substitutions take place asynchronously and in parallel, but require physical resources). Other combinators, which are not necessary from a mathematical perspective, must be included because of the structures they create. For example, the $\mathsf{Y}$ combinator is used to construct cyclic structures (i.e., self-sharing structures), whereas its definition in terms of S and K constructs potentially infinite branching structures, which are mathematically equivalent to cycles, but not physically equivalent.

As previously discussed, interactive applications of universally programmable intelligent matter will require sensor and effector operations. The sensors will assume two or more different configurations, or react in two or more different ways, depending on some external condition (presence of light, a chemical species, etc.). For example, a sensor could produce either of two different molecular groups, such as K (representing **true**, which selects the first alternative) and (KI) (representing **false**, which selects the second), or their equivalents. Therefore, we need to determine a general way of incorporating sensors and effectors into programs. However, a more serious problem is controlling the execution time of sensors and effectors in a computational model in which substitutions can take place at any time and in parallel. There are ways of delaying substitutions in combinatory networks (for example, by "abstracting" [1, 4] a dummy variable from them), but it is not clear whether this is the best approach, or whether we should use a different model that provides more direct control over time of execution.

Certainly, one of the strengths of the combinatory logic approach to molecular computation is the Church-Rosser property, which means that substitutions can take place in any order without affecting the result of computation, but there are some issues that must be resolved. For example, if a program has two alternative branches, selected by a conditional operation, then we may have substitutions taking place in both branches simultaneously. Even though it will not affect the result of computation, it may be wasteful to process a program structure that will not be needed. However, there is a worse problem. If programs are written in the natural recursive way (such as our Ring and Tube examples above), then it is possible that all the reaction resources could go to recursive expansion of conditional arms that will end up being discarded. There are ways to solve this problem within the combinatory logic framework. (For example, we can delay the substitutions within a network by "abstracting" [1, 4] from it an argument, which is provided only when we want the substitutions to proceed.) However, there may be more direct solutions to the problem, such as those used in ordinary programming language implementations, which can be adapted to the molecular context.

Similar problems arise in the implementation of interactive intelligent matter, that is, intelligent matter that does not compute to a stable state, but remains active. A rewriting system, such as the SK calculus, is by its nature "compute once" because the program is consumed in the process of computation. Indeed, this similarity to a chemical reaction is one of its virtues in the context of molecular computation. However, it is a problem in the context of interactive intelligent matter, since it means that a program, once executed, does not exist to execute a second time. There are at least two potential solutions to this problem. The standard solution, used in the network-substitution implementation of programming languages, is to use a combinator such as Y (mentioned above) to replicate a program structure whenever it is needed. Another solution would be a molecular process to "interpret" a fixed program structure, much as in an ordinary digital computer. However, aside from the fact that this gets quite far away from network substitution and its advantages, it has serious problems of its own, which come from having multiple functional groups simultaneously "reading" (and thus linking to) the program structure.

Although there are many similarities between term-rewriting systems and molecular processes, there are also important differences, which must be addressed in a theory of molecular computing. For example, the relative rates of reactions can be controlled by the concentrations of the reactants and other conditions, such as temperature. This can be exploited for nonstandard uses of network substitution. For example, we may have two or more conflicting sets of substitution rules for a set of molecular operators, and we can determine which are applied, or their relative probability of being applied, by controlling the concentrations of the reactants needed for the alternate rule sets. This permits probabilistic control of the nanostructures generated. Or we may have different substitutions performed in different stages of a process. Computation can also be controlled by external fields or other gradients for various purposes, such as creating oriented structures. Such considerations raise problems and potentials that are new for models of computation.

One characteristic of molecular computation which distinguishes it from electronic computation is the high probability of error in molecular processes. Therefore, we will need to develop means (both chemical and computational) for decreasing the probability of such errors, for correcting them when they occur, or for assuring that results are insensitive to them. Further, unpublished preliminary investigations indicate that a certain fraction of random SK trees will result in nonterminating, expansive, chain reactions [13, 21]. (The probability of termination decreases with increasing random tree size.) This is a potential problem, since it suggests that a sizable fraction of substitution errors could result in runaway chain reactions that could use up all the reaction resources.

Network substitution is based on the mathematical definition of a graph: dimensionless nodes linked by edges; normally the geometrical arrangement of the nodes and length of the edges is irrelevant. However, molecular groups occupy finite volumes, and there are constraints on the locations of bonds and lengths of linking groups, which are some of the constraints that need to be accommodated in a theory of molecular computing (e.g., compare Figs. 1 and 2). Folding of program networks could also interfere with substitution operations, and so we will have to find chemical means of keeping networks extended.

Interactive applications must be provided with an adequate supply of reactants to assure that substitutions can take place when they are supposed to. Certainly, some of the reactants can come from the recycled products of previous reactions, but others will require fresh raw materials. The same considerations will apply in static applications that involve long or complex chain reactions. There are several possible solutions to this problem, but the choice depends on the specifics of the UPIM application. For example, if the networks are attached to solid substrates (e.g., membranes, sponges or particles), then reactants may be made to flow over them (thus also clearing away waste products). If the networks are in colloidal suspension or free-floating in a fluid, then reactants are easy to add; waste products might be removed by osmosis, filtering, precipitation, etc.

Thse are just a few of the ways in which universally programmable intelligent matter differs from conventional computation and use of term-rewriting systems.

# 3 Objectives

Given the preceding discussion of some of the problems that must be addressed in order to make universally programmable intelligent matter a success, we can now present the objectives of the proposed one-year exploratory research project.

## 3.1 Mathematical Model of Molecular Computation

The first objective is to develop a mathematical model of molecular computation that, on the one hand, incorporates the most important constraints on molecular processes and, on the other, is at a sufficiently high level of abstraction to avoid irrelevant complexities of the processes. As already remarked, the model must respect the fact that only two functional groups can be connected by a linking group, and that links are of (approximately) fixed length and make their connections at (approximately) fixed bond angles. At this time we anticipate that the model will be based on mathematical graph theory, incorporating the geometrical constraints on link length and orientation. It is unclear to what extent energy relationships need to be incorporated into the model. The goal will be to develop a mathematical framework that is independent of particular primitive substitutions (e.g., S and K) and other operations (e.g., sensors and effectors).

## 3.2 Universal Sets of Operators

The second objective will be to identify at least two universal sets of primitive operators. SK is, of course, a primary candidate, but there are others, as mentioned above, which may be more suitable from the perspective of simplicity or similarity to molecular processes. In addition, whichever mathematically complete set is chosen, we will have to extend it to solve the replication/sharing problem, as well as to handle creation of cyclic structures (e.g., Y), destruction/recycling of discarded structures (e.g., D) and other bookkeeping and practical necessities. To the extent possible, these extended operations will be kept independent of the choice of universal operators.

## 3.3 Interface Operators

We cannot anticipate the various molecular sensor and effector operators that may be discovered or synthesized in the future. However, at this stage we can develop general interface techniques that allow these operators to be integrated into the combinatory logic framework. For example, sensor substitutions may produce either of two groups, which function as the truth values **true** or **false**. Accommodation of imperative sensor and effector operations will also require a solution to the problem of controlling their execution time. Fortunately, we know of two potential solutions (based on *bracket abstraction* [1, 4] and *token passing*), and there are doubtless others.

## 3.4 Simulation Tool

Some issues in universally programmable intelligent matter can be settled mathematically. For example, it is relatively easy to prove that a set of primitive combinators is universal (assuming that the result has not already been proved in the literature). It will also be straightforward to prove some results on possible network geometries and on the control of imperative operators. However, some other results will be difficult to obtain by analysis, and for these investigations we intend to develop a simulation tool.

The basic functionality of the simulator will be to perform substitutions based on provided rewrite-rules for the primitive combinators, including sensors and effectors. There must be means for reading in simulated molecular networks or chains and for displaying them during simulation or at its end.

The simulator also needs to be able accommodate factors that are important in molecular computing. For example, substitutions should be performed randomly, with probability of substitution being proportional to reactant concentrations. Reactant depletion and replenishment should be modeled, so that we can investigate their use to control computation. The simulator must also allow us to investigate the effects of substitution error by controlling the error rates of the operations. The simulator should allow simulation of spatially fixed networks (such as those anchored to substrates or embedded in gels) as well as free-floating networks (such as those in fluids or colloids).

## 3.5 Demonstration Applications

We intend to use the simulator to implement several simple applications to investigate the potential of universally programmable intelligent matter. At this time, we are considering the following demonstrations:

1. nanotubes of specified diameter and length,
2. membranes with pores of specified diameter and density;
3. membranes with cilia that flex in a specified direction upon command;
4. a rotating flagellum.

These are not complex problems from the standpoint of pure computation, but we need to understand the issues that arise when they are implemented molecularly.

# 4 Plan

## 4.1 Mathematical Model

The mathematical model must be established before any of the other tasks can progress. The actual mathematical definition will be straightforward once it has been decided what should be included in it and what left out. We have allocated 3 months to the development of the model.

## 4.2 Primitive Operators

Once the model is established we can begin to select a set of basic operators, as small and simple as possible, but including those necessary for computational universality, cyclic processing, bookkeeping, and typical sensors and effectors. If we decided to explore new universal sets of combinators, then we will have to prove their universality, but that is generally straightforward (just use them to program a known universal set). More difficult will be settling on at least one feasible solution to the replication/sharing problem and to an approach to the control of interactive intelligent matter. In the process of evaluating the primitives, several of the demonstration applications will be programmed. This task is estimated to take 4–6 months and will be conducted by the PI.

## 4.3 Simulator

Once the mathematical model is complete (Sec. 4.1), specification and implementation of the simulation can begin and proceed in parallel with the selection of primitive operators (Sec. 4.2). A Graduate Research Assistant will implement the simulator, a task which is estimated to take 4–6 months.

## 4.4 Demonstration Applications

Once the simulator is running we will be able to begin to test the demonstration applications. These will be "hand coded" in the combinatory primitives, unless we happen to decide on a set of combinators (such as SKI) for which translators are already available. At first we will run them under ideal (error-free) conditions; as time permits we will begin to investigate the effects of substitution errors, reactant concentrations, etc. This task, in which both the PI and a GRA will participate, is estimated to take 3 months.

# 5 Summary

In this section we summarize the relevance of universally programmable intelligent matter to the research themes of the Nanoscale Science and Engineering Program. Universally programmable intelligent matter will provide a systematic, general approach to the production of **nanoscale systems** since it will permit the synthesis of any structure describable by a computer program. Further, because the materials can be active, we may have dynamic control of permeability, viscosity and other physical characteristics. Universally programmable intelligent matter will contribute to **nanoscale devices and system architecture** by providing a systematic way of synthesizing nanostructures useful for computation and control, including sensors and actuators. More importantly, it will provide a new technology for computing that will be better suited to the molecular scale than are conventional technologies. If successful, universally programmable intelligent matter will provide a systematic approach to **manufacturing processes at the nanoscale** by allowing the creation and assembly of nanosystems to be controlled by molecular programs. Finally, universally programmable intelligent matter will allow us to create systems and materials inspired by **biosystems at the nanoscale**, such as active membranes, very dense neural networks, sensors, and motor systems such as cilia and flagella.

# References

[1] S. K. Abdali. An abstraction algorithm for combinatory logic. *Journal of Symbolic Logic*, 41(1):222–224, March 1976.

[2] A. Church and J. Barkley Rosser. Some properties of conversion. *Trans. American Math. Soc.*, 39:472–482, 1936.

[3] H. B. Curry. Grundlagen der kombinatorischen Logik. *American Journal of Mathematics*, 52:509–536, 789–834, 1930.

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

[5] Joseph H. Fasel and Robert M. Keller (eds.). *Graph Reduction, Proceedings of a Workshop, Santa Fe, New Mexico, USA, September 29 – October 1, 1986*. Springer Verlag, Berlin, 1987. Lecture Notes in Computer Science, vol. 279.

[6] Walter Fontana and Leo W. Buss. 'The arrival of the fittest': Toward a theory of biological organization. *Bulletin of Mathematical Biology*, 56(1):1–64, 1994.

[7] Walter Fontana and Leo W. Buss. What would be conserved if 'the tape were played twice'? *Proceedings National Academy of Science USA*, 91:757–761, 1994.

[8] Walter Fontana and Leo W. Buss. The barrier of objects: From dynamical systems to bounded organizations. In J. Casti and A. Karlqvist, editors, *Boundaries and Barriers*, pages 56–116. Addison-Wesley, 1996.

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

[10] C. Hoffman and M. J. O'Donnell. Pattern matching in trees. *Journal of the ACM*, 29(1):68–95, January 1982.

[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. Addison-Wesley, 1989.

[12] Bruce J. MacLennan. *Functional Programming: Practice and Theory*. Addison-Wesley, 1990.

[13] 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.

[14] B. Mayer, G. Koehler, and S. Rasmussen. Simulation and dynamics of entropy driven, molecular self-assembly processes. *Physical Review E*, 55(4):4489–4499, 1997.

[15] B. Mayer and S. Rasmussen. The lattice molecular automata (LMA): A simulation system for constructive molecular dynamics. *International Journal Modern Physics*, 9(1), 1998.

[16] S. Rasmussen and J. Smith. Lattice polymer automata. *Ber. Bunsenges. Phys. Chem.*, 98(9):1185–1193, 1994.

[17] Steen Rasmussen, Carsten Knudsen, and Rasmus Feldberg. Dynamics of programmable matter. In C. Langton et al., editor, *Artificial Life II*, volume X of *SFI Studies in the Sciences of Complexity*, pages 211–254. Addison-Wesley, 1991.

[18] Barry K. Rosen. Tree manipulation systems and Church-Rosser theorems. *Journal of the ACM*, 20(1):160–187, January 1973.

[19] M. Schönfinkel. Über die Bausteine der mathematischen Logik. *Math. Annalen*, 92:305–316, 1924.

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

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