Evolving Artificial Cell Signaling Networks: Perspectives and Methods

James Decraene, George G. Mitchell and Barry McMullin

Artificial Life Laboratory
Research Institute for Networks and Communications Engineering
School of Electronic Engineering
Dublin City University, Glasnevin, Dublin, Ireland
james.decraene@eeng.dcu.ie

Summary. Nature is a source of inspiration for computational techniques which have been successfully applied to a wide variety of complex application domains. In keeping with this we examine Cell Signaling Networks (CSN) which are chemical networks responsible for coordinating cell activities within their environment. Through evolution they have become highly efficient for governing critical control processes such as immunological responses, cell cycle control or homeostasis. Realising (and evolving) Artificial Cell Signaling Networks (ACSNs) may provide new computational paradigms for a variety of application areas. In this paper we introduce an abstraction of Cell Signaling Networks focusing on four characteristic properties distinguished as follows: Computation, Evolution, Crosstalk and Robustness. These properties are also desirable for potential applications in the control systems, computation and signal processing field. These characteristics are used as a guide for the development of an ACSN evolutionary simulation platform. Following this we describe a novel class of Artificial Chemistry named Molecular Classifier Systems (MCS) to simulate ACSNs. The MCS can be regarded as a special purpose derivation of Hollands Learning Classifier System (LCS). We propose an instance of the MCS called the MCS.b that extends the precursor of the LCS: the broadcast language. We believe the MCS.b can offer a general purpose tool that can assist in the study of real CSNs in Silico The research we are currently involved in is part of the multi disciplinary European funded project, ESIGNET, with the central question of the study of the computational properties of CSNs by evolving them using methods from evolutionary computation, and to re-apply this understanding in developing new ways to model and predict real CSNs.

1 Introduction

Cell Signaling networks (CSNs) are bio-chemical systems of interacting molecules in cells [12, 17]. Typically, these systems take as inputs chemical signals generated within the cell or communicated from outside. These trigger a cascade of chemical reactions that result in changes of the state of the cell and (or)

generate some (chemical) output, such as prokaryotic chemotaxis, coordination of cellular division, or even to order the death of a cell (in the context of multi-cellular organisms).

As signal processing systems, CSNs can be regarded as special purpose computers [4]. In contrast to conventional silicon-based computers, the computation in CSNs is not realized by electronic circuits, but by chemically reacting molecules in the cell. The most important molecular components of CSNs are proteins and nucleic acids (DNA, RNA). There is an almost infinite variety of potential molecular species, each of which would have distinct chemical functionality and could engage in interactions with other molecules with varying degrees of specificity.

We distinguish CSNs as being networks made up of more than one distinct cell signaling *pathway*, which interact with each other.

An example of a simple chemotaxis signaling pathway is shown in Figure 1.A. Chemotaxis is a phenomenon where simple organisms such as bacteria move toward higher concentrations of specific chemicals in their surroundings. In this diagram, we distinguish six intracellular proteins (denoted as A, B, R, W, Y and Z) and the membrane receptors which can bind to the corresponding stimulatory element. The input level is determined by the concentration of bound molecules. This affects the output represented by the tumbling frequency which governs the bacteria direction. Figure 1.B shows, in schematic form, a simple Cell Signaling *Network* made up of two such interacting signal *pathways*.

We distinguish this work from previous work on real CSNs [12, 17] by focusing purely on Artificial Cell Signaling Networks (ACSNs). Through the use of evolutionary computing techniques we allow ACSNs to spontaneously emerge and adapt to the environment. Potentially of interest for the Biologist may be the insight that ACSNs gives as to how real CSNs evolved and how they operate. This synthetic biology approach allows us to incorporate the present knowledge of real CSNs into ACSNs. This biological understanding provided guiding points that directed the design of the MCS, these points also guide the evolution of ACSNs in silico. This may, for example, facilitate the prediction of missing signaling pathway information in real CSNs [15].

Given our motivation to maintain the biological plausibility of ACSNs, we are interested in investigating the use of ACSNs to implement computation, signal processing and (or) control functionality. This is motivated by preliminary studies which demonstrated that real CSNs could be considered for computational and engineering purposes:

- In [20], Lauffenburger presents his approach to cell signaling pathways which could be thought of and modelled as control modules in living systems.
- Yi et al. [26] demonstrated that CSNs may have some of the essential properties of an integral feedback control. This is a basic engineering strategy

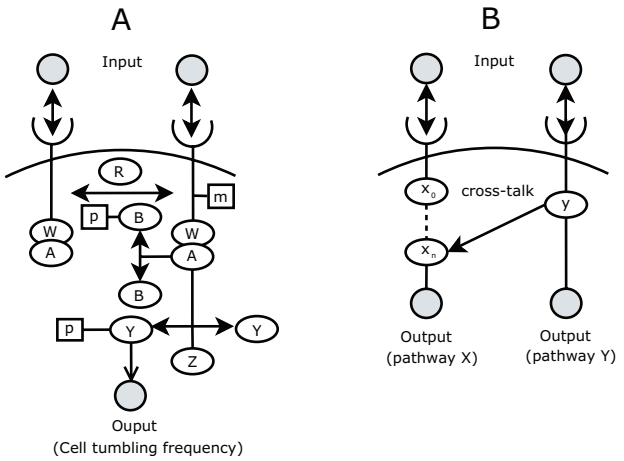

Fig. 1. A: Schematic representation of bacterial chemotaxis signaling pathway, adapted from [1]. The output is designated by the tumbling frequency which is determined from the input, the concentration level of ligand bound to the membrane receptors. This signal transduction is carried out by the reaction cascade depicted by the proteins A, B, R, W, X and Z. Details on chemical reactions can be found in [23]. B: A CSN composed from two distinct cell signaling pathways with unique input and output, an interaction between pathways occurs as molecule y interacts with x_n, this modulates the output of pathway X.

to ensure that a system outputs desired values independent of internal and external perturbations.

- Deckard and Saura [6] used and evolved artificial biochemical networks capable of certain simple forms of mathematical computation such as a square root function.

One way to design ACNs to carry out such complex operations is to use artificial evolutionary techniques. A significant insight related to the evolution of signaling networks in silico, was suggested by Holland [15]. Holland proposed examining a simple agent-based model where the agents' behavior and adaptation was determined by the use of Learning Classifier System [13, 5]. Based on this machine learning approach Holland suggested that signaling networks could be modeled with LCS in a *top-down* fashion. We show how this work does not lend itself as a method for addressing our project.

We propose a variation of the LCS called the Molecular Classifier System which allows the emergence and evolution of signaling networks. This approach may be considered as viewing the evolution and design of signaling networks from a *bottom-up* manner complementing Holland's approach.

In section 2 we examine the nature of ACSNs by examining the critical issues that these raise. In section 3 we first present and discuss Holland's ap-

proach and following this we propose the structure of our ACSN evolutionary simulation platform, the Molecular Classifier System.

2 Artificial Cell Signaling Networks

As an abstraction of real CSNs, ACSNs are differentiated and simplified by some key properties. The selection of these particular characteristics is motivated by the will to employ Artificial Cell Signaling Networks for computational and control engineering purposes. Four issues are distinguished and presented: Computation, Evolution, Crosstalk and Robustness.

2.1 Computation

In the simplest cases, CSNs can be approximately modelled by systems of continuous differential equations, where the state variables are the concentrations of the distinct species of interacting molecules. As a "computational" device, this is most naturally compared to a traditional *analog* computer. Analog computers are precisely designed to model the operation of a target dynamical system, by creating an "analogous" system which shares (approximately) the same dynamics. Electronic analog computers (based on the "operational amplifier" as the core computational device) have long been displaced by digital computers, programmed to numerically solve the relevant dynamical equations, due to their much greater ease of programming and stability.

Nonetheless, there may be applications where a molecular level analog computer, in the form of a CSN, may have distinct advantages. Specifically, CSNs may offer capabilities of high speed and small size that cannot be realised with solid state electronic technology. More critically, where it is required to interface computation with chemical interaction, a CSN may bypass difficult stages of signal transduction that would otherwise be required. This could have direct application in so-called "smart drugs" and other bio-medical interventions.

While CSNs are typically treated in this "aggregate" manner, where the signal or information is carried by molecular concentration, one can also consider the finer grained behaviours of individual molecules are computational in nature. Thus a single enzyme molecule can be regarded as carrying out pattern matching to identify and bind target substrates, and then executing a discrete computational operation in transforming these into the product molecule(s). This has clear parallels with a wide variety of so-called "rewriting systems" in computational theory.

However, it also clearly differs in important ways, such as:

- Operation is stochastic rather than deterministic.

- Operation is intrinsically *reflexive* in that all molecules can, in principle, function as both "rules" (enzymes) and "strings" (substrates/products).

Dittrich [9] provides a more extended discussion of the potential of such "chemical computing".

2.2 Evolution

Evolutionary Algorithms (EAs) are non-deterministic search and optimisation algorithms inspired by the principles of neo-Darwinism. They have been applied successfully in a variety of fields [14, 11, 16]. Generally based on genetic operations such as crossover and mutation, EAs initially generate a wide range of candidate solutions. Over time, through selection, this can be reduced to an optimized set. Evolutionary computation can therefore deliver useful results without requiring a priori knowledge of the entire search space [11, 16].

Such techniques are relevant to the study of ACSNs because:

- The complex, and unpredictable, interactions between different components of CSNs, make it very difficult to design them "by hand" to meet specific performance objectives.
- However, natural evolution shows that in suitable circumstances, effective CSNs functionality can be achieved through evolutionary processes.

For example, Deckard and Saura [6] used such evolutionary techniques to construct (simulated) biochemical networks capable of certain simple forms of signal-processing. In this model (called Lakhesis), computational "nodes" represent molecule species with an attribute for concentration. Connections between nodes designate reactions defined by the type and rate of the reaction.

Another ESIGNET project contribution is given by Lenser et al. [24], in this System Biology approach, a multi-level EA is proposed to evolve biochemical networks (represented in SBML) for performing pre-specified tasks such as reconstructing real CSNs. The base level EA searches for optimal network topologies and the second level explores the kinetic parameters.

2.3 Crosstalk

"Crosstalk" phenomena happen when signals from different pathways become mixed together. This arises very naturally in CSNs due to the fact that the molecules from all pathways may share the same physical reaction space (the cell). Depending on the relative specificities of the reactions there is then an automatic potential for any given molecular species to contribute to signal levels in multiple pathways. An example is shown in Fig. 1.

In traditional communications and signal processing engineering, crosstalk is regarded as a defect—an *unintended* or *undesigned* interaction between signals, that therefore has the potential to cause system malfunction. This can also clearly be the case of crosstalk in CSNs. However, in the specific

case of CSN's, crosstalk also has additional potential functionality, which may actually be constructive:

- Even where an interfering signal is, in effect, adding uncorrelated "noise" to a functional signal, this may sometimes improve overall system behaviour. This is well known in conventional control systems engineering in the form of so-called "dither". Compare also, [2, 25] on constructive biological roles of noise.
- The crosstalk mechanism provides a very generic way of creating a large space of possible modifications or interactions between signaling pathways. Thus, although many cases of crosstalk may be immediately negative in their impact, crosstalk may still be a key mechanism in enabling incremental evolutionary search for more elaborate or complex cell signaling networks.

2.4 Robustness

It is argued that key properties in biochemical networks are to be robust, this is so as to ensure their correct functioning [3]. Similar works include research carried out at the Santa Fe institute in studying Cytokine signaling networks to design distributed autonomous networks, that are robust to small perturbations and responsive to larger ones [18]. Potential applications are distributed intelligent systems such as large fleets of robots working together, for automated response in computer security, for mobile computing networks, etc.

Alon et al. have demonstrated from studying *Escherichia coli* chemotaxis that molecular interactions can exhibit robustness [1, 20]. In this case it means that after a change in the stimulus concentration (input), the tumbling frequency (output) managed to reach a steady state that is equivalent to the pre-stimulus level. This is illustrated in Figure 2.

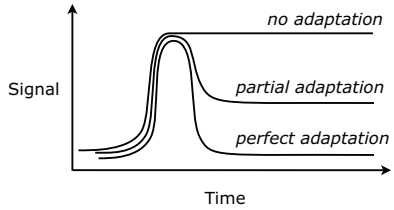

Fig. 2. Representation of dynamic responses of a system to a stimuli adapted from [20]. No adaptation is observed when the system response attains a new steady state following the change in input. Partial adaptation describe a partial recovery, the difference between the initial state and the new state is lower than the one observed in the previous case. Perfect adaptation is met when the system is able to come back to its initial state

Such properties are highly desirable in dynamic engineered systems when subjected to internal and external uncertainty and perturbation.

3 An evolutionary approach to implement ACSNs

In the following, we first examine a specific class of Evolutionary Algorithm called Learning Classifier System (LCS) devised by Holland in 1976 [13]. In 2001 Holland [15] identified the possibility of using LCS to implement signaling networks (biochemical circuits). However Holland's work was never actually implemented. We use Holland's proposition as the seminal point for the development of the first Classifier System based ACSN implementation, our Molecular Classifier System (MCS).

3.1 Learning Classifier Systems

Learning Classifier Systems are systems constructed from condition-action rules called *classifiers*. The classifiers can be viewed as IF/THEN statements in the form IF "rule" THEN "action". The condition section of the classifier examines all of the messages in the system and identifies those that satisfy the *rules* conditions. Once this is accomplished the action part instructs that a message is to be sent. Holland's initial work was modified a number of times and at present many different varieties of learning classifier systems are available [19].

In Holland's LCS the system receives an input from its environment as a binary encoded data. This is then stored in an internal data store termed the *message list*, see Figure 3. The LCS then evaluates the input and determines an appropriate response, indicated by the action. This action typically alters the current state of the environment. Any desired behaviour that is exhibited is then rewarded through a scalar reinforcement. The system iterates the cycle of response, reinforcement and discovery for each discrete time-step.

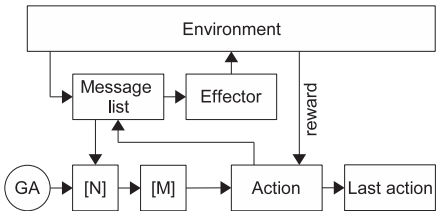

Fig. 3. Schematic of Holland's Learning Classifier System

The rule-base consists of a population of N classifiers. Both parts of the classifier are randomly initialized. The rule conditions and actions (the classifiers) can be characterized by strings formed from a ternary alphabet 0,1,#.

The use of the # provides a single character wildcard which allows for the potential matching of a greater number of strings e.g. 10# would match two potential inputs 100 or 101. The use of the wildcard character also provides for string processing at the action stage, for example: in responding to the input 110, the rule IF 1#0 THEN 0#1 would produce the action 011. Each classifier also has an associated fitness measure, quantifying the *usefulness* of a rule in attracting external reward.

On receiving an input message, a typical LCS processes as follows: initially the input message rule-base is scanned and all rules whose condition matches the external message are added to the "match set" denoted as [M], see Figure 3. Secondly any other rules matching messages in the message list are also added to [M]. Rules that contribute significantly to the targeted learning task may then be reinforced through the use of *bidding techniques*. For a comprehensive introduction to Learning Classifier System, see [5].

In [15] Holland proposed an agent-based model where the agents' behavior and adaptation are determined by the use of Learning Classifier System. This work provided an existence proof that LCS could be used to evolve a simple repertoire of condition-action rules to a more complex goal directed set of rules. In typical biochemical networks, interactions between molecules follow the same condition-action mechanisms. Thus Holland suggested that this approach could be used to simulate and evolve signaling networks. His proposition to design signaling networks was to start with a LCS-based "over-general" model of a biological phenomenon (e.g. transformation of a healthy cell to a cancer cell, see Table 1). Then a general phenomenon can be refined

Table 1. Over-general model of the transformation of healthy cell to cancer cell

Rule	Condition	Action
(1)	If healthy cell and DNA damage	Then apoptosis or immortality
(2)	If immortality	Then stable existence or genetic instability
(3)	If genetic instability	Then ephemeral clonal expansion or robust clonal expansion

through several iterations. At each iteration, the details of the occurring interactions are detailed, see Table 2. These iterations were continued until the desired CSN level was reached, where the biomolecular elements are specified (e.g. protein ligand, receptor, ions etc.), see Table 3 for an example of such a rule. This refining process clearly shows the top-down methodology to design signaling networks.

Despite this, the LCS-based approach to specify CSNs sounded promising, actual implementation was never performed. Importantly, this approach does not meet the requirements of our project. First, we do not distinguish a demarcation between rules and messages, in our context, the chemical operations are reflexives. Secondly, Holland's suggestion was to initially model known

Table 2. Refinement of rule 1

Rule	Condition	Action
(1.1)	IF healthy cell and DNA damage	Then apoptosis or mutation for resistance to apoptosis
(1.2)	IF resistance to apoptosis	Then susceptibility to growth inhibitory signals or mutation for loss of susceptibility to growth inhibitory
(1.3)	IF loss of susceptibility to immortality	THEN selective growth advantage and growth inhibitory signals

Table 3. A biomolecular level rule

Rule	Condition	Outcome
(x.x.x.x)	If apropos growth factor	Then gf receptor activated

real CSNs, however from our bottom-up perspective, we require the ACSNs to evolve from very simple networks to more complex networks that exhibit the known real CSNs properties. As a consequence we propose a variation of Holland's LCS to fulfill the requirements of our project.

3.2 The Molecular Classifier System

We define the Molecular Classifier System (MCS) as a class of string-rewriting based Artificial Chemistries. This approach is inspired by Hollands Learning Classifier Systems (LCS). In Hollands LCS, a demarcation is distinguished between *rules* and *messages*, however as mentioned earlier operations in a biochemical networks are intrinsically *reflexive* in the sense that all molecules can function as both rules (enzymes) and messages (substrates/products). The MCS addresses this issues by removing this rules/messages demarcation found in the LCS.

The behavior of the condition/binding properties and action/enzymatic functions is specified by a "chemical" language defined in the MCS. The chemical language defines and constrains the complexity of the chemical reactions that may be represented and simulated with the MCS. For example, a MCS model using a limited number of computational functions may only fatefully represent very simplistic chemical reactions.

Before describing the nature of the enzymatic functions (action part of a molecule), the binding properties of the molecules must be identified. We have thus far identified the following potential properties: In the MCS approach, a reaction between molecules may only occur if the informational string of a first molecule satisfies/binds with the conditional part of a second molecule. The second molecule may be the same as the first molecule leading to self-binding. The condition part refers to the binding properties of a molecule whereas action refers to the computational ("enzymatic") function. This pat-

tern matching occurring implies a notion of *specificity* or "binding strength". A molecule having a high specificity would have less chance to react with another one. Whereas a molecule having a low specificity is likely to bind to another more often. Therefore we could translate this into an effect on reaction rate/kinetics.

When two molecules can bind and consequently react to each other, the action part of one of the molecules is used to carry out the enzymatic operations upon the binding molecule (substrate). This operation results in producing another offspring (product). This is analogous to the action part of a LCS rule used by Holland [15]. When a reaction occurs, the symbols contained in the MCS action part are processed in a sequential order (parsed from left to right). The outcome (product) of the reaction depends on the nature of the symbols' functionality.

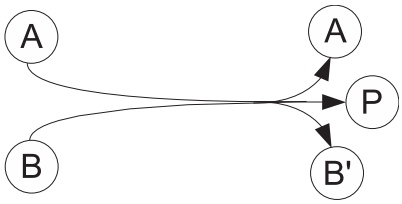

Fig. 4. Schematic of a reaction in the MCS: When a molecule A can react with a molecule B, the action statement of molecule A is "executed" upon the informational string of the binding molecule B. A is viewed as an enzyme and B as a substrate, thus A's structure is not affected by the reaction whereas B's structure is degraded and a product P is generated. A's action statement operators take as inputs the symbols of B's string. An offspring molecule P is generated as a result of these operations

In [21], a minimalist approach to the MCS was proposed to investigate protocell computation. In this study, a protocell is modelled as a container for artificial molecules (Molecular Classifier Systems). The latter may interact with each other to generate new molecular offspring. The chemical language used in this instance of a MCS for protocells employs a minimal set of computational components to reduce the conceptual gap between artificial and real chemistry. To represent, simulate and evolve ACSNs, more computational functions are necessary, nevertheless the definitive set of operations is still under investigation as we are trying to understand what are the minimal operational requirements to allow a primitive ACSN to spontaneously emerge. In the remainder of this section, we present a candidate solution based on a variant of the Holland Broadcast Language.

3.3 The Holland Broadcast Language

The Broadcast Language is a programing formalism introduced by Holland in 1975 [14, 7], which can be thought of as the precursor for the LCS, the latter being a simplification of the Broadcast Language. The Broadcast Language was originally intended to solve some undesirable issues arising out of the use of Genetic Algorithms (GAs) [14, 11]. Holland argued that GAs provide an efficient method of adaptation, however in the case of long-term adaptation, the efficiency of GAs could be limited by the representation used to encode the problem. This problem representation is usually fixed and influences the complexity of the fitness function. During long-term evolution, this may limit the performances of the GA. To overcome this limitation, Holland proposed to adapt the problem representation used by the fitness function. Adapting the representation may then generate correlations between the problem representation and the GA performance.

A key property shared between the MCS and the Broadcast Language is the removal of any demarcation between messages and rules. A second beneficial property is the ability of the Broadcast Language to provide a straightforward representation to a variety of natural models such as the operon-operator model (a Genetic Regulatory Network model).

The Broadcast Language basic components are called *broadcast units* which can be viewed as condition/action rules. Whenever a broadcast unit conditional statement is satisfied, the action statement is executed. This means that whenever a broadcast unit detects in the environment the presence of (a) specific signal(s), including themselves, then the broadcast unit would broadcast an output signal.

Some broadcast units may broadcast a signal that may constitute a new broadcast unit. Similarly, a broadcast unit can be interpreted as a signal detected by another broadcast unit. Broadcast units may also process a given signal, in the sense that, a broadcast unit may output a signal that is some modification of the detected/input signal. As a result, a broadcast unit may create new broadcast units or detect and modify an existing broadcast unit. A set of broadcast units, combined as a string, designates a *broadcast device*.

Table 4. Comparison of biological and broadcast language terminology

Biology	Broadcast Language
sequence of amino acids from $\{A, R, N, D, C, E, \ldots\}$	string of symbols from $\Lambda = \{0,\ 1,\ *,\ :,\ \Diamond,\ \nabla,\ \blacktriangledown,\ \triangle,\ p,\ '\}$
substrate	input signal
product	output signal
protein with no enzymatic function	null unit
enzyme	broadcast unit
protein complex	broadcast device
cellular milieu	list of strings from Λ

As a summary, the above table presents a comparison between the biological and the broadcast language terminology.

The Broadcast Language alphabet Λ is finite and contains ten *symbols*, Λ^* is the set of strings over Λ. The symbols constitute the atomic elements of the language.

$$\Lambda = \{0,\ 1,\ *,\ :,\ \Diamond,\ \triangledown,\ \blacktriangledown,\ \triangle,\ p,\ '\}$$

Let I be an arbitrary string from Λ^*, in I, a symbol is said to be quoted if it is preceded by a symbol $'$. A broadcast unit I_n is an arbitrary string from Λ^* which contains neither unquoted $*$ or unquoted : .

The finite collection of broadcast devices can be described by its *state* S at each timestep t. For example $S(0) = \{011 : *\triangle 011 : 11, 101, 100, 0111 : 01 :\}$ describes the set of broadcast devices at timestep $t = 0$, which corresponds to the initial state of the collection. Four types of broadcast unit can be distinguished, any broadcast units that do not follow one of the four schemes (see below) are null units. Broadcast units may engage in the following interactions based on discrete timesteps:

1. $*I_1 : I_2$
   If a signal of type I_1 is detected at time t then the signal I_2 is broadcast at time $t + 1$.

2. $* : I_1 : I_2$
   If there is no signal of type I_1 present at time t then the signal I_2 is broadcast at time $t + 1$.

3. $*I_1 :: I_2$
   If a signal of type I_1 is detected at time t then a *persistent* string of type I_2 (if any) is removed from the environment at the end of time t.

4. $*I_1 : I_2 : I_3$
   If a signal of type I_1 and a signal of type I_2 are both present at time t then the signal S_3 is broadcast *at the same time* t unless the string I_3 contains unquoted symbols $\{\triangledown, \blacktriangledown, \triangle\}$ or singly quoted occurrence of $*$, in which case the string I_3 is broadcast a time $t + 1$.

The symbols

The interpretation of each symbol in $\Lambda = \{0,\ 1,\ *,\ :,\ \Diamond,\ \triangledown,\ \blacktriangledown,\ \triangle,\ p,\ '\}$ is now presented. In particular cases, some symbols may not be interpreted by a given broadcast unit, these *ignored* symbols are simply overlooked. However, they remain important as they may get activated at a later stage where broadcast units undergo recombination.

$\{0, 1\}$ 0 and 1 are the basic elements to specify a signal. A string such as 010110 can be regarded as the signature of a particular signal. This signature can be employed by a broadcast unit to detect and identify a signal.

For example: let $I_1 = *10111 : 00$ be a broadcast unit and $I_2 = 10111$ a signal, both strings are present at time t in the environment: $S(t) = \{*10111 : 00, 10111\}$. At time t, I_2 is detected by I_1, this triggers the activation of broadcast unit I_1, as a result: $S(t+1) = \{*10111 : 00, 10111, 00\}$.

$*$ This symbol indicates that the subsequent *symbols* until the next unquoted $*$ (if any) are to be interpreted as a broadcast unit. If a broadcast device I does not contain any unquoted $*$ then I is a null unit.

: This symbol is used as a punctuation mark to differentiate the arguments of a broadcast unit. The symbol : (position and frequency) determines the type of the broadcast unit as presented earlier. If more than two unquoted : are found in a broadcast unit then the third : and anything to the right of it are ignored.

$\Diamond$ When this symbol is met in the argument of a broadcast unit, it indicates that a signal detected by the broadcast unit may present any symbol at this position. This specific symbol occurring in the detected signal does not affect its acceptation or rejection by the broadcast unit.

For example, with $S(t) = \{*10\Diamond 11 : 00, 10011\}$ we obtain at time $t+1$ $S(t+1) = \{*10\Diamond 11 : 00, 10011, 00\}$, $*10\Diamond 11$ broadcasts 00 if a signal containing $10 \ldots 11$ is present ($\ldots$ indicates any arbitrary symbol from Λ).

Also if $\Diamond$ occurs at the rightmost position in the argument of a broadcast unit, then it indicates that a signal detected by the broadcast unit may present any suffix without affecting acceptance or rejection.

For example, with $S(t) = \{*1011\Diamond : 00, 101101101\}$ we obtain $S(t+1) = \{*1011\Diamond : 00, 101101101, 00\}$, $*1011\Diamond : 00$ would broadcast 00 if any signal containing the prefix 1011 is detected.

$\triangledown$ When this symbol occurs in the arguments of a broadcast unit, it designates any arbitrary initial (prefix) or terminal (suffix) strings of symbols. This allows one to pass a string of symbols from the input signal to the broadcast signal ($\approx$ unit processing).

For example, with $S(t) = \{*10\triangledown : \triangledown, 10011\}$ we obtain at $t+1$: $S(t+1) = \{*10\triangledown : \triangledown, 10011, 011\}$. In this case $\triangledown$ designates the suffix 011 occurring in the input signal 10011. whereas if $S(t) = \{*10\triangledown : \triangledown, 100100101\}$ then we obtain at $t+1$: $S(t+1) = \{*10\triangledown : \triangledown, 100100101, 0100101\}$. If several occurrences of $\triangledown$ are found in the output argument of a given broadcast unit, then they all designate the same substring.

$\blacktriangledown$ This symbol is similar to $\triangledown$ but can also concatenate different inputs signals.

For example, with $S(t) = \{*10\triangledown : 11\blacktriangledown : 000\triangledown\blacktriangledown, 10111, 1100\}$ we obtain at $t+1$: $S(t+1) = \{*10\triangledown : 11\blacktriangledown : 000\triangledown\blacktriangledown, 10111, 1100, 00011100\}$. In this case $\triangledown$ designates the suffix 111 occurring in the input signal 10111 and $\blacktriangledown$ designates the suffix 00 found in the detected signal 1100. The format of the broadcast signal is $000\triangledown\blacktriangledown$, therefore we replace and concatenate $\triangledown$ and $\blacktriangledown$ accordingly and we obtain the output signal 00011100.

$\triangle$ This symbol is employed in the same manner as $\triangledown$ and $\blacktriangledown$ but designates an arbitrary *single* symbol whose position can be anywhere in the argument of a given broadcast unit.
For example, with $S(t) = \{*11\triangle 0 : 1\triangle, 1100\}$ the output signal 10 is broadcast and this produces at $S(t+1) = \{*11\triangle 0 : 1\triangle, 1100, 10\}$. Whereas if $S(t) = \{*11\triangle 0 : 1\triangle, 1110\}$ the output signal 11 is broadcast producing $S(t+1) = \{*11\triangle 0 : 1\triangle, 1100, 11\}$.

p When this symbol occurs at the first position of a string, it designates a *persistent* string. This string would then persist over time until it is deleted, even if the string is not an active broadcast unit. A null device occurring at time t which is not persistent exists only for one timestep and is removed at the end of time t.

$'$ This symbol is used to *quote* a symbol in the arguments of a broadcast unit. When a symbol is said to be quoted, it acts as a simple literal, i.e. a $'\triangle$ would only match $\triangle$.
For example, with $S(t) = \{*11'\triangle 0 : 11, 11\triangle\}$, the input signal $11\triangle$ is detected by the broadcast unit and thus the output string 11 is broadcast producing at $t+1 : S(t+1) = \{*11'\triangle 0 : 11, 11\triangle, 11\}$.

The broadcast language presented by Holland in the original text omitted a number of interactions between broadcast devices, which could in certain cases present us with ambiguities regarding the expected action to be performed. Holland discussed some of these semantical conflicts [14], while the remaining ambiguities were addressed in [7, 8]. However in the context of this paper, this is not an important consideration.

Example: Building a NAND gate

In this section we describe the construction of a NAND gate using the broadcast language. This is intended to demonstrate how the broadcast language can be considered as a logical universal computational formalism. Moreover using the Boolean abstraction, it is also possible to build qualitative models of natural networks such as Genetic Regulatory Networks. With the Boolean abstraction, a molecule is considered as a logical expression having two different possible states. One possible state is the ON state meaning that the molecule is present in the environment. When a molecule state is OFF, this indicates that the particular molecule is not present in the environment (cell).

In the remainder of this section we first present a simple example in which a NAND gate is constructed within a static environment (the inputs values do not change over time), then a second example follows in which the same gate is adapted to be used with a dynamic system:

1. We consider a NAND gate having for inputs signals A and B and for output signal C. To construct this logical gate with the broadcast language, we first represent each signal A, B, C as null broadcast devices (substrates): $A = p001$, $B = p010$ and $C = p000$.

We then declare the following active broadcast devices (enzymes): $I_1 = *p001 : 011$ and $I_2 = *p010 : 100$, these devices emit signaling molecules $S_1 = 011$ and $S_2 = 100$ upon detecting A and B respectively. Similarly, we define $I_3 = * : p001 : 101$ and $I_4 = * : p010 : 110$ which would emit $S_3 = 101$ and $S_4 = 110$ if A or B are not detected.
Finally the following broadcast devices are employed to output C according to the intermediary states of signaling molecules S_1,S_2,S_3 and S_4: $I_5 = *011 : 110 : p000$, $I_6 = *100 : 101 : p000$ and $I_7 = *101 : 110 : p000$. Using these broadcast devices, it is possible to obtain the state of C according to the states of input signals A and B, 2 time steps are necessary to propagate and process the signals A and B.

2. Within a dynamic system, some modifications are necessary to maintain our NAND gate. These modifications are intended so as to allow the output signal C to degrade over time. First, the broadcast devices I_1,I_2 and I_3 are modified as follows: As currently defined, those broadcast devices output the signal $p000$ (which designates the persistent signal C) when satisfied, these output signals are replaced with an additional signaling molecule $S_5 = 111$. As a result, we obtain the broadcast devices: $I'_5 = *011 : 110 : 111$, $I'_6 = *100 : 101 : 111$ and $I'_7 = *101 : 110 : 111$.
We then declare a broadcast device $I_8 = *111 : p000$ that upon detecting S_5 would emit the output signal C. Finally we declare a broadcast device $I_9 = p000 :: p000$ that deletes (degrades) C upon detecting C. We note that as soon as a signal C appears at time t, it would be removed by I_9 at the end of time t. To counter balance that effect, we double the concentration of broadcast devices I_8 so that the production rate of C is higher than its degradation rate.

In Fig. 5 we present a simulation using such a NAND gate specified with the broadcast language. In this simulation, the inputs A and B are manually switched ON/OFF at different timesteps. We detail the states of the system at timestep 0,1 and 2:

$$\begin{aligned} S(0) = \{&A = p001\ ,\ I_1 = *p001 : 011,\ I_2 = *p010 : 100,\ I_3 = * : p001 : 101,\\ &I_4 = * : p010 : 110,\ I'_5 = *011 : 110 : 111,\ I'_6 = *100 : 101 : 111,\\ &I'_7 = *101 : 110 : 111,\ I_8 = *111 : p000,\ I_8 = *111 : p000, I_9 = p000 :: p000\} \end{aligned}$$

At $t = 0$, the system is initialized with above broadcast devices, A is ON, and both B and C are OFF. We note that both broadcast devices I_1 and I_4 are satisfied leading to the production of S_1 and S_4 at $t = 1$:

$$\begin{aligned} S(1) = \{&A = p001\ ,\ I_1 = *p001 : 011,\ I_2 = *p010 : 100,\ I_3 = * : p001 : 101,\\ &I_4 = * : p010 : 110,\ I'_5 = *011 : 110 : 111,\ I'_6 = *100 : 101 : 111,\\ &I'_7 = *101 : 110 : 111,\ I_8 = *111 : p000,\ I_8 = *111 : p000, I_9 = p000 :: p000,\\ &S_1 = 011, S_4 = 110\} \end{aligned}$$

At $t = 1$, I'_5 is activated due to the presence of S_1 and S_4. I'_5 is a type 4 broadcast device that is able to output S_5 signal during same timestep. As a result, both I_8 broadcast devices are now activated and produce two instances of C molecule at $t = 2$. As S_1,S_4 and S_5 are not persistent, these signals are removed at the end of $t = 1$. However, as A is still ON and B is OFF, S_1 and S_4 are again produced at $t = 2$.

$$
\begin{aligned}
S(2) = \{&A = p001\ ,\ I_1 = *p001 : 011,\ I_2 = *p010 : 100,\ I_3 = * : p001 : 101,\\
&I_4 = * : p010 : 110,\ I'_5 = *011 : 110 : 111,\ I'_6 = *100 : 101 : 111,\\
&I'_7 = *101 : 110 : 111,\ I_8 = *111 : p000,\ I_8 = *111 : p000, I_9 = p000 :: p000,\\
&C = p0000, S_1 = 011, S_4 = 110\}
\end{aligned}
$$

At the beginning of $t = 2$, two instances of C are contained in the system. However the I_9 broadcast device is now satisfied by instances of the C molecule resulting in the removal of one instance of C.

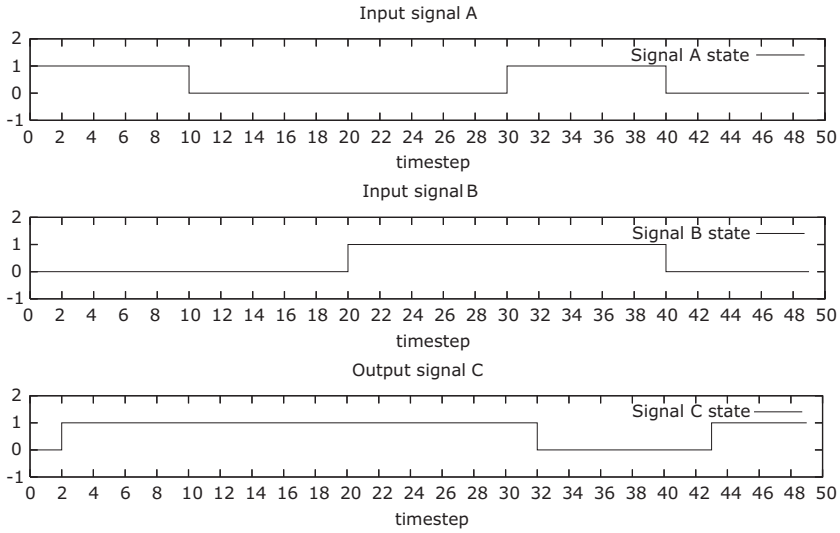

Fig. 5. NAND gate specified with our implementation of the broadcast language, details can be found in [7]. The output signal C state is initialized as OFF (0), at timestep 10,20,30 and 40, inputs A/B are manually switched ON/OFF (1/0), We note that the propagation time needed to process the switching of inputs A or B differs according to the nature of the switching involved and present states of A,B and C.

3.4 Fusing MCS and the Broadcast Language

In [8], it was demonstrated that the Broadcast Language can model Genetic Regulatory Networks (GRNs). This was due to the ability of the Broadcast Language to mirror Boolean networks which illustrates the wide ranging processing power that Broadcast Systems are capable of. Nevertheless, it was also highlighted that the Broadcast Language is limited regarding the representation and simulation of CSNs. To address this issue, we propose to combine the MCS concept with the Broadcast Language in a new system termed "MCS.b". The MCS.b complements the broadcast language (syntax and semantics) and extends it by including the following refinements:

- Instead of processing all broadcast devices sequentially and deterministically during a time step, the MCS.b processes as follows: at each time step t, we pick n pairs of broadcast devices at random. For each pair of devices, one of the broadcast devices is designated (at random) as the *catalyst device* and the second one as the *substrate device*. If the conditional statement of the catalyst device is satisfied by the signal of the substrate device, then the action statement of the catalyst device is executed upon the substrate device.
- n is a constant and designates the number of pairs of broadcast devices that will interact during a timestep. It is also plausible to consider n as the temperature in real chemistry. Temperature has an important role in chemical reactions, indeed molecules at higher temperature have a greater probability to collide with one another. In the broadcast language "universe", in order to increase the "temperature", one may increment the integer number n.
- In the broadcast language specification given by Holland, additional rules were required to resolve some ambiguities raised by the interpretation of broadcast devices. To facilitate this, the MCS.b simplifies the interpretation of broadcast units by preserving broadcast units of type 1 only.
- Similarly the notion of non-persistent devices is removed: by default all devices are considered as persistent molecules.
- As type 3 broadcast units and non-persistent devices no longer exist in this proposal, no molecule can be deleted from the population. However the deletion of molecules is needed to obtain evolutionary pressure. Our suggestion is as follows: each time two molecules react together, we pick a molecule at random and delete it from the population.

By combining the strength of both the MCS and Broadcast Language, we expect the MCS.b to be capable of modeling, simulating and evolving ACSNs in a more fateful manner. At present, we have conducted a number of preliminary experiments examining the spontaneous emergence of collective autocatalytic sets among others. This was expected to be trivial as this phenomenon was already demonstrated with other Artificial Chemistry Systems (such as Tierra, Alchemy, etc.). Initial results suggest that the MCS.b

performs as expected, however before these results can be presented to the research community, validation against empirical biological data is required.

4 Future work

In keeping with the four presented ACSN characteristic properties, we will focus on the following areas:

4.1 Computation

As part of the ESIGNET project, we will investigate the computational power of the MCS.b. This will include an examination of the MCS.b for Turing Completeness. As one of our project goals is to evolve ACSNs for computational purposes, incorporating completeness may be regarded as a crucial issue.

4.2 Evolution

An ACSN implies several cell signaling pathways interacting with each other. In order to evolve such a system of signaling networks *controlling* each other, it will be necessary to evaluate different Evolutionary Computational (EC) techniques. Because from biology it is natural to have a *hierarchical* system it may prove beneficial to investigate multi-level EC systems e.g. Hierarchical Genetic Algorithms [10].

4.3 Crosstalk

To obtain a better understanding of the crosstalk phenomenon and more specifically about the positive and negative effects of crosstalk. We will would like to see if it is possible to specify a network topology that allows optimal control of crosstalk effects.

A small world topology [22] may be of interest, as we may observe an analogy between CSNs and small world networks. This class of network, and more specifically scale-free networks are characterized by possessing nodes acting as “highly connected hubs”. Although most nodes in these networks are of low degree. For example, a highly connected node could be referring to an ATP molecule that shares the same high degree of connectivity in real biochemical networks.

4.4 Robustness

We will investigate the ability of ACSNs to create and sustain specific internal conditions such as homeostasis. We would like to exhibit such robust behavior in simulated ACSNs, and how through evolutionary changes, robustness can be refined. Another consequent issue is to quantify the robustness of such systems to external shocks and changes of conditions.

5 Conclusion

In this paper we introduced an abstraction of Cell Signaling Networks focusing on four characteristic properties distinguished as follows: Computation, Evolution, Crosstalk and Robustness. We indicated how these attributes can be highly desirable properties for potential applications in the control systems, computation and signal processing field. Following this we described a novel class of Artificial Chemistry named Molecular Classifier Systems (MCS) which was inspired by Hollands Learning Classifier System (LCS). To simulate and evolve ACNS, we proposed an instance of the MCS called the MCS.b that extends the precursor of the LCS: the broadcast language. We completed the paper by examining further work that is required to conclusively validate our approach.

Acknowledgement

This work was funded by ESIGNET (Evolving Cell Signaling Networks in Silico), an European Integrated Project in the EU FP6 NEST Initiative (contract no. 12789).

References

1. U. Alon, M. G. Surette, N. Barkai, and S. Leibler. Robustness in bacterial chemotaxis. *Nature*, 397(6715):168–171, January 1999.
2. A. M. Arias and P. Hayward. Filtering transcriptional noise during development: concepts and mechanisms. *Nature Reviews Genetics*, 7(1):34–44.
3. N. Barkai and S. Leibler. Robustness in simple biochemical networks. *Nature*, 387(6636):913–917, June 1997.
4. D Bray. Protein molecules as computational elements in living cells. *Nature*, 376(6538):307–312, Jul 1995.
5. L. Bull and T. Kovacs. Foundations of Learning Classifier Systems: An Introduction. *Foundations of Learning Classifier Systems*, 2005.
6. A. Deckard and H. M. Sauro. Preliminary studies on the in silico evolution of biochemical networks. *Chembiochem*, 5(10):1423–1431, October 2004.
7. J. Decraene. The Holland Broadcast Language. Technical Report ALL-06-01, Artificial Life Lab, RINCE, School of Electronic Engineering, Dublin City University, 2006.
8. J. Decraene, G. G. Mitchell, B. McMullin, and C. Kelly. The holland broadcast language and the modeling of biochemical networks. In Marc Ebner, Michael O'Neill, Anikó Ekárt, Leonardo Vanneschi, and Anna Isabel Esparcia-Alcázar, editors, *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, Valencia, Spain, 11 - 13 April 2007. Springer.
9. P. Dittrich. Chemical computing. In Jean-Pierre Banâtre, Pascal Fradet, Jean-Louis Giavitto, and Olivier Michel, editors, *UPP*, volume 3566 of *Lecture Notes in Computer Science*, pages 19–32. Springer, 2004.

10. B. Freisleben. Metaevolutionary approaches. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, pages C7.2:1–8. Institute of Physics Publishing and Oxford University Press, Bristol, New York, 1997.
11. D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, January 1989.
12. Ernst J. M. Helmreich. *The Biochemistry of Cell Signalling*. Oxford University Press, USA, 2001.
13. J.H. Holland. Adaptation. *Progress in theoretical biology*, 4:263–293, 1976.
14. J.H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.
15. J.H. Holland. Exploring the evolution of complexity in signaling networks. *Complexity*, 7(2):34–45, 2001.
16. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press, December 1992.
17. G. Krauss. *Biochemistry of Signal Transduction and Regulation*. John Wiley & Sons, 2003.
18. S. Forrest L. Segel. Robustness of cytokine signalling networks. http://www.santafe.edu/research/signallingnetworks.php.
19. P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors. Springer-Verlag, April 2001.
20. D.A. Lauffenburger. Cell signaling pathways as control modules: complexity for simplicity? *Proc. Natl. Acad. Sci. USA*, 97(10):5031–3, 2000.
21. B. McMullin, C. Kelly, D. OBrien, G. G. Mitchell, and J. Decraene. Preliminary Steps toward Artificial Protocell Computation. In *Proceedings of the 2007 International Conference on Morphological Computation*, 2007. To appear.
22. M. E. J. Newman. Models of the small world: A review, May 2000.
23. R. C. Stewart and F. W. Dahlquist. Molecular components of bacterial chemotaxis. *Chem. Rev.*, 87:997–1025, 1987.
24. T. Lenser, T. Hinze, B. Ibrahim, and P. Dittrich. Towards Evolutionary Network Reconstruction Tools for Systems Biology. In *Fifth European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, 2007. To appear.
25. D. Volfson, J. Marciniak, W. J. Blake, N. Ostroff, L. S. Tsimring, and J. Hasty. Origins of extrinsic variability in eukaryotic gene expression. *Nature*, December 2005.
26. T. M. Yi, Y. Huang, M. I. Simon, and J. Doyle. Robust perfect adaptation in bacterial chemotaxis through integral feedback control. *Proc Natl Acad Sci U S A*, 97(9):4649–4653, April 2000.