Bioastronomy 2002: Life Among the Stars
IAU Symposium, Vol. 213, 2004
R.P.Norris and F.H.Stootman (eds.)

Algorithmic Communication with Extraterrestrial Intelligence

B. S. McConnell
841 Corbett Ave, San Francisco CA 94131, U.S.A.
(brian@mcconnell.net)

Abstract. ACETI (algorithmic communication with extraterrestrial intelligence) builds upon mathematical languages to create a general purpose programming language. While the underlying framework may be quite simple, the programs derived from even a small instruction set, when run on a sufficiently fast computing substrate, may interact with their users in real-time and may exhibit complex behavior.

1. Introduction

ACETI (Algorithmic Communication with Extraterrestrial Intelligence) builds on mathematical languages such as Hans Freudenthal's Lingua Cosmica (Freudenthal 1960) to create a general purpose programming language that is derived from a small collection of basic symbols. The underlying framework is simple, and can be taught to anyone who is knowledgeable in basic math and logic. Yet, programs derived from this basic framework can exhibit complex and perhaps even intelligent behavior.

Such a framework is interesting for many reasons. This approach allows information to be transmitted with maximum efficiency, near the theoretical limits for a channel with finite bandwidth, while also encoding data to correct corrupted data (Shannon 1948). This technique also takes advantage of the fact that while the bandwidth of an inter-stellar communication channel is probably limited by background noise, power requirements, etc, the local bandwidth inside a computer, or network of computers, is likely to be much greater. Because of the computational requirements for SETI searches, there is likely to be a surplus of computing bandwidth at either end of an inter-stellar communication channel. Instead of sending a static dataset, such as a bitmap image, the sender can transmit algorithms that produce many different results depending on their starting conditions and interaction with the user. This idea was first proposed by Minsky (1971) who suggested using an algorithm to generate the Fibonacci series in an inter-stellar message.

ACETI is an interesting communication medium because it will allow the sender to create a message that is derived from a small set of basic symbols. These symbols can then be combined to define higher order instruction sets and more complex functions. While the behavior of the programs may be complex, the underlying language requires the recipient to understand only basic math and logic. Furthermore, the message may include programs that are designed

to process data elsewhere in the message, and possibly assist the recipient in parsing and deciphering the message.

The ability to combine different types of instruction sets is especially interesting, as each computing paradigm is well suited to different types of problems. Analog computing networks may be useful for simulating the behavior of highly networked systems that contain many feedback loops, although these can also be modeled digitally. Digital (Turing machine) programs will be useful for performing general-purpose numeric calculations, and will be able to handle a wide range of computing tasks. Quantum computing systems may also be useful for tackling certain problems that require exponential computing resources (e.g., factoring large numbers, simulating QM interactions). Since space is limited, this paper will focus on analog and digital systems.

Such a framework offers another even more interesting possibility, that the sender could transmit programs, such as simulations, that interact with the recipient in real-time. These programs would act as proxies on behalf of their senders, and could guide the recipient in analyzing and comprehending the contents of the message.

2. Digital Computing

Building an algorithmic communication system is fairly straightforward. The first step is to create an imaginary computer, or virtual machine, that recognizes a small collection of basic operators. These commands include functions such as:

- Basic math (e.g., addition, subtraction, shift, rotate, etc)
- Comparison (e.g., greater than, less than, equal, not equal)
- Conditional branching (e.g., If X is True, Jump to Y)
- Input/output (e.g., store or recall data from memory)

This is not a new idea. Freudenthal (1960) and Devito (1990) both proposed constructing mathematical languages for inter-stellar communication years ago. By adding two features to a mathematical language: conditional branching (if-then-else, subroutines, etc) and input/output (ability to read/write to virtual memory registers), it is possible to create a general purpose programming language that runs on this virtual machine.

To decode an algorithmic message, the recipient will first need to learn the basic symbols used in the programming language. Since these are all derived from basic math and logic functions, these can be taught in a primer that highlights sets of true and false examples (e.g., $1 + 2 = 3$, $2 \times 2 = 4$, etc). These symbols can be expressed as numbers, just as the low-level instructions for a CPU are expressed in numeric form.

The sender may describe the basic symbols in at least two ways. One option is to describe a set of abstract math and logic symbols, and allow the recipient to decide how best to execute these instructions in hardware. Another option is to describe a large network of logic circuits (e.g., NAND gates), and in effect, describe how to build a computing device that can execute programs

designed for it. Each approach has advantages. It should be possible to use both methods in the primer, and thus provide the recipient with two paths to decoding the basic instruction set. The recipient uses this information to build a table similar to the Intel 80 × 86 instruction set. This table associates each numeric symbol with its function or meaning (e.g., 12=ADD, 13=SUBTRACT, 16=STORE, 17=RECALL, etc). With this information, the recipient can then write software that executes programs derived from this basic instruction set.

Building higher-level programs would be accomplished by constructing libraries of intermediate modules that are derived from the basic instruction set. These are built up in layers, and can be reused in other programs. For example, a function to calculate the sine of an angle requires addition, division, multiplication and conditional branching. This sine function, once written, can then be referenced in shorthand form by other modules and programs.

A collection of intermediate modules would probably be fairly large. These modules would handle a wide range of functions including: higher-level math, memory management, compression, error correction, and so forth. The toolkit would be analogous to the large collection of reusable subroutines that are included with modern programming languages. Although many layers of modules may be used to build a high-level program, all of these functions are ultimately reduced to the basic instruction set.

Such programs can be used to describe any number of systems, including: mathematical concepts such as the Fibonacci series (Shannon 1948), simulation of an organic system such as an ant colony (McConnell 2001), and complex physical processes such as molecular folding (McConnell et al. 2004). The range of problems that can be calculated using this type of computing is quite large, as proven by the diversity of programs available for modern personal computers and handheld computing devices.

3. Analog Computing

The general approach outlined in this paper can also be used to transmit instructions to build analog computers. In fact, such analog computing networks can be used to describe digital computing circuits that, in turn, run digital programs described elsewhere in a message. The sender could describe an analog network that models the behavior of a memory device or CPU, thus providing the recipient with a blueprint for emulating or translating such systems into hardware.

Analog computing offers a number of advantages compared to their digital counterparts, one of which is the ability to model richly interconnected systems that contain positive and negative feedback loops (e.g., nervous systems, climate models, etc). While these systems can be modeled in digital algorithms, the computation requirements for modeling these systems can be quite large. This is especially true in systems where a computation's outcome may be affected by multiple feedback loops.

Analog programs are not written as series of procedural commands, as are digital programs. The author describes a real or virtual network of electronic components that, when linked together, transform, sum and compare the information fed into them. The inputs to these programs are analog electrical

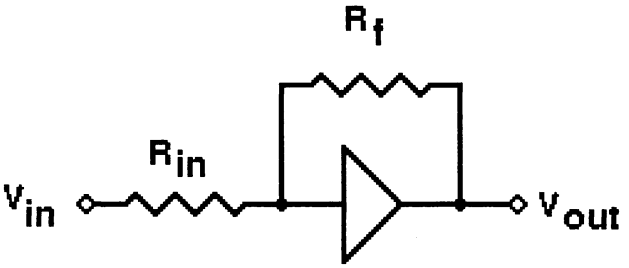

Figure 1. An operational amplifier, one of the basic components in an analog computer. In this circuit, $V_{out} = R_f/R_{in} \times V_{in}$. This is the equivalent of the function $y = Ax$ [credit: Ken Bigelow (Devito 1990)].

signals, either from an external device or from the output terminals of another analog computing circuit. Like their digital counterparts, the primitives in an analog computer can be reduced to a small lexicon of basic components. One difference between an analog computer and stored-program digital computer is that the instructions for an analog program are embedded in the circuit design for the computer.

Analog computers can combine a number of mathematical functions, including: addition, subtraction, multiplication, integration, differentiation and logarithmic functions. Each of these functions can be implemented via fairly simple circuits, which can then be linked to each other to form large analog computing networks as well as digital computing devices.

To transmit an analog program, we must also describe the underlying circuitry. Each discrete circuit in an analog computer can be represented as an ideal mathematical function. When placed in a N-dimensional coordinate space, these ideal circuits can then be linked to each other using sets of simple $P_{(x,y,z)} = F_{addition}(P_{(x,y,z)}, P_{(x,y,z)})$ statements. The first step in creating a system for transmitting analog programs is to create a coordinate space. One can think of this as a virtual breadboard. Each ideal circuit in the analog computer (an adder, for example) will have two or more terminals. In a physical device, these terminals would be wires or leads that are affixed to a circuitboard or breadboard. In this system, the terminals are represented as 2-D, 3-D or N-dimensional coordinates.

Each element in the analog computer will be represented by a series of expressions, such as the example below:

$$P_{(1,3)} = \mathrm{Fadd}(P_{(1,1)}, P_{(1,2)})$$
$$P_{(2,3)} = \mathrm{Fadd}(P_{(2,1)}, P_{(2,2)})$$
$$P_{(3,3)} = \mathrm{Fadd}(P_{(3,1)}, P_{(3,2)})$$
$$P_{(3,4)} = \mathrm{Fmult}(P_{(1,3)}, P_{(2,3)})$$
$$P_{(3,5)} = \mathrm{Fmult}(P_{(3,4)}, P_{(3,3)})$$

NOTE: The statements are shown in mnemonic form for easy reading. The operators would be represented in the actual message as numeric symbols, pulse trains, etc.

https://doi.org/10.1017/S0074180900193696 Published online by Cambridge University Press

Since the recipient of such a message will not know the contents of the transmission, the sender will also need to transmit a primer that describes the basic message format and primitives. For analog programs, the sender will need to describe: the coordinate system used to link components, delimiters that demarcate nested expressions, the basic functions used in the networks (e.g., addition, subtraction, etc). Since even very complex analog computing networks can be derived from a small base vocabulary of symbols, the primer need only describe a dozen or so basic elements. These basic elements can be chained together to perform much more complex calculations, yet the message itself will consist of nothing more than a long series of $x = fn(a, b)$ statements.

The recipient of such a message will have the option of running these programs in a simulation on a fast digital computer, or implementing the analog program in hardware (a better option for richly interconnected systems with feedback between nodes).

A simple analog program, such as a program that calculates the rate of change of an input (dA/dt), will not require much computation. This type of program can therefore be run as a simulation on a fast digital computer. The recipient can simply write software that parses the X=F(A,B) statements used to describe an analog program, and then run a simulation to model the behavior of the system. If the analog network is small, or if the recipient has very fast computers, this is the easiest way to probe the behavior of these programs without building customized hardware.

But what if the instruction set describes a very large network of richly interconnected nodes, such as a synthetic nervous system (even a relatively simple insect-like nervous system)? Even though each node performs relatively simple calculations, it also influences the behavior of many nearby nodes that in turn may influence the behavior of their neighbors. This type of problem can easily overtax the capabilities of a centralized stored-program computer. In this case, the recipient may translate the instruction set into hardware that can run the analog program efficiently.

One disadvantage analog computers have, relative to digital programs, is their inability to represent precise numerical quantities, such as integers. This is a problem when computing sequential series (a FOR.. NEXT loop for example). When modeling complex, networked systems, one is often more interested in the rate of change of the variables in a simulation, and the feedback loops between these variables and between elements in the simulation. In such a situation, the noise introduced by analog computers is usually overwhelmed by the feedback effects, and in many cases, is actually useful because it simulates the random variations found in real-world systems.

4. Local Communication and Intelligent Proxies

Algorithmic communication will be useful not only because of the ability to transmit data more reliably and efficiently, but also because of the potential to allow real-time interaction between sender and recipient. While the message itself cannot be transmitted faster than light, the sender can transmit sophisticated programs that the recipient runs on a local computing device or simulator. If these programs are sufficiently sophisticated, this will eliminate the need for

the recipient to send most queries back to the sender (and wait many years for a reply). The programs act as proxies on behalf of their senders.

While it is difficult to predict the level of intelligence these programs might exhibit, it is reasonable to assume that a detectable civilization will be at least as proficient in computing as we are today. Therefore these programs would, at a minimum, match the capabilities of programs in widespread use today. If one is comfortable assuming that a detectable civilization will be more experienced in this respect, it is probably reasonable to conclude that these proxies could exhibit some types of intelligence when run on a fast enough computing grid.

This raises some interesting implications should SETI succeed in acquiring an information-bearing signal from another civilization. The traditional assumption has always been that any message we receive will be in the form of a static message, similar to the Arecibo message sent in 1975. An algorithmic system that is designed to run on a large computing grid will not be a static message. This is an interesting possibility because it would enable real-time communication between the proxy agents and their recipients, and because such agents could accelerate the process of comprehending an inter-stellar message. If provided with a fast enough computing substrate, these systems may themselves be considered intelligent.

References

Devito, C. 1990, Journal of the British Interplanetary Society, 43, 561

Freudenthal, H. 1960, LINCOS: Design of a Language for Cosmic Intercourse, (North Holland Publishing Co.)

McConnell, B. S. 2001, in SPIE Proceedings : The SETI in the Optical Spectrum, 4273, 194

McConnell, B. S. et al. 2004, Between Worlds, MIT Press

Minsky, M. 1971, presentation, US-USSR SETI conference, Byurakan, Armenia

Shannon, C. E. 1948, Bell System Technical Journal, http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html