APL-CAM Journal, Vol. 14, No.3, 16 JUly 1992, pp. 407-411.
Received: 28 April 1992.

DNA-STYLE PROGRAMMING IN APL2 FOR GENETIC SIMULATION

Gerard A. Langlet
B.P. 36, F-78354-Jouy en Josas

Abstract

DNA-style Programming is a programming style that mimics the way strands of the DNA 4-character-code agglutinate in order to produce the best and efficient routines we happen to know, the ones you and me are programmed with. starting from a few well-chosen characters, short words are formed by catenation; these words are then embedded in an upper-scale alphabet, this process being able to build a fully recursive structure. APL2 is a suitable language for handling strands, with, moreover, a real pedadogic power: the concept of DNA-style programming, applied to bitmaps, allows to mimic pattern replication in a way which looks much like the meiose of cells as observed by biologists. Such models are a consequence of the theory of binary integration.

Introduction

The theory of binary integration, exposed in previous papers, results from algorithmic compression: It was observed, experimentally, that many algorithms, after reduction to their nucleus, converged towards a unique idiom, named in APL jargon "different scan", symbolised by `≠\` , and whose other names are "asymmetric parity (or spin) propagation", or "binary integration", i.e. the counterpart, in the Boolean field, of numeric cumulation `(+\)` or discrete integration. `≠\` is also known in image and signal processing, to produce the "inverse Gray code".

Studies of the properties of `≠\` gave rise to the embryo of a new theory of information, the pariton being the matrix of all iteratively integrated binary sequences of any finite bit sequence, this construct being mathematically proved as periodical. Genetic processes cannot be described by continuous functions. Any information may be converted with full precision into a sequence of bits, and processed with no loss of information, by combination of a very few number of primitive applications, as shown in [1] and [2]. In the general case, all Boolean functions, except NOT, XOR and NOT-XOR, may alter the content of the message they process, absorbing or half-absorbing information, and should be discarded, for this reason, from any attempt of modelling periodic systems. A periodic system is able to reproduce itself in space (crystals are the best example, but all elementary particles also belong to this category) or in time (all living species, as long as no mutation occur; mutations can occur, coming from outside the periodic system, which is in fact a sub-system of the universe, by a collision e.g. of a large asteroid with a planet).

NOT, XOR and NOT-XOR are symbolised in APL by ~ ≠ and = respectively. One easily checks that, for any `B`, `~B` is equivalent to `1≠B` , and that, for any `A` and `B`, `A=B` is

`~A≠B` i.e. `1≠A≠B` so that all the acceptable primitives for the modelling of periodic systems may be replaced by `≠` alone.

The hereabove-mentioned proof that successive iterates of `≠\` build a periodic system for any binary sequence, strengthens the conjecture (set after pure experimental work on algorithms), that `≠` and its derived function `≠\` should allow to describe correctly all periodic systems and perhaps will lead to understand complex ones. Moreover, there can be no other solution: It is impossible to build a "plus" or "minus" function, without using at least a half-absorbing primitive (AND) in order to propagate carries; then "times", "divide", "power" and other applications, derive from "plus". In some special cases, however, such as `A+A` or `2×A` which is equivalent, the absorbing effect of AND disappears and the result, expressed in base 2, is just a bit-shift so that information is preserved. Shifts do not destroy the shape of information, provided a 0 is inserted at the right place. In physics and mathematics, this is named "invariance by translation". We shall not discuss here "invariance by symmetry", provided in APL by combination of the monadic symbols `⌽ ⊖` and `⍉` although nice automata may also be built using them.

In order to program a genetic automaton, we MAY use, as elementary bricks, the scalar `≠` as well as the asymmetric spinor `≠\` on all axes of propagation. In APL, which handles orthogonal arrays, `≠\` is just an easy notation, equivalent to `≠\[⎕IO]` , for scan along the first axis, i.e. rows in a matrix. We shall be allowed circular shifts, provided we first catenate 0's in a convenient way; in fact, `1⌽` i.e. one-step shifts will suffice, because information-bits, as well as particles, interact first with their closest neighbour only. Then, the result of an interaction (by `≠`) is transmitted to the second neighbour which has become the closest neighbour of the result: this is exactly what `≠\` does intrinsically, acting as a pseudo-wave maker. Successive iterations of `≠\` produce tides, indeed periodic, with also sub-periods, on bit sequences: this is exactly what you will observe on a beach where quanta are no more bits, but water molecules, or, at a larger scale, droplets, and also with sub-periods.

DNA programming style

Having already shown that, in a pari ton, information self-organises into helices and double helices, we shall now explore another way which will bring us closer to optimal programming, i.e. the DNA code, with the help of APL2-syntax.

Note. APL2 implementations are now even available at low-cost (MicroAPL's APL.68000 Level II on Macintosh and Atari, Girardot's APL90 on Macintosh, IBM's APL2 on all PC's and, especially IBM's TRYAPL2. Although this latter implementation has some restrictions for array and workspace size, it is very fast and codes binary arrays in bits; moreover, the diskette is free, may be copied and distributed at no charge). APL2-syntax is now on the way to become the heart of the new ISO-standard; since IBM recently adopted the statement separator (which also exists in the genetic code), we may now use it in almost all APL implementations.

Here is a small genetic program in APL2. All variables could be localised with no problem; in fact, we think that the standard should make automatically local all 1-character possible variable names, without explicit declaration anymore.

```
    ∇ N AUTOMATON T
[1] ⎕IO ifno'N'⋄ (G D R U H V)←'()T≠⍀⌿'⋄ (S H V)←(R U)(U H)(V V)
[2] W←G H R D U V⋄ E←N⌷S(S V)(S H)W(S W)⋄ Vdo∈'MAT T←'E'1⌽T←0,T'
[3]                                          ⍝ That's all, folks
    ∇
```

Syntax: *N* is optional *N*∈⍳5; *T* is a binary matrix.

ifno is a function which assigns the value of its left argument to the name given in its right one. Here, *N* becomes ⎕*IO* in the absence of *N* ; this function replaces ⍎(*O*=⎕*NC*'*N*')/'*N*←⎕*IO*' .

vdo is the "vital do" which means "iterate the expression as much as you can"; this suppresses control structures in programs and is closer to biology: we are driven by such a notoptional facility; ***vdo*** is a computer-analog or synonym of *live*, the program submitted to execution being always the same, an almost-perfect optimiser; ***vdo*** sets an infinite loop which stops in case of any error (fatal disease), of interrupt/break (kill), of WS FULL (not enough air) or of QUOTA EXHAUSTED (the weight of years, i.e. too many bits to manage; some APL2-like implementations still offer the nice SYMBOL TABLE FULL diagnostic ...). The APL2-model may be given by the following function (left); on the right, stands the APL*PLUS II one:

```
∇ vdo ∆dE                          ∇ vdo dE;OALX;⎕ELX
[1] '→' ⎕EA ∆E ⋄ →1                [1] ⎕ALX←⎕ELX←'→'⋄ ⍎∆E ⋄ →1
∇                                  ∇
```

PLOT is any of your favourite graphic or semi-graphic programs, showing a black-and-white image of bitmaps, or isoplots (outlining). You may also replace it by ⎕← (or a monadic +). We may suggest:

```
    ∇ PLOT T
[1] 2/' ' [T]                              ⍝ ⎕IO is already 0
    V
```

Five different genetic automata are indeed available, according to the value of *N* (Which, just like in computer chess-games, defines a scale of complexity). Level a (assuming a-origin) corresponds to ≠ used alone. Level 4 compares (with scalar ≠) the results of binary integration on both axes, by ≠⍀ for rows, and by ≠⌿ for columns. Many more levels can be added, e.g. for automata expanding both ways [3] (this one expands horizontally only - vertical expansion would result from the simultaneous replacement of "⊖" by "⌽" and of "," by "⍪" or ",[⎕IO]"), or for more integrations (along e.g. diagonals or various combinations of axes, if *T* is a 3-D or a multidimensional array, or an enclosed array that mimics a full population with different animal or vegetal heterogeneous shapes).

So, level a clones *T* after intermediate transforms. Level 4 produces clones of the initial *T* and, in addition, pseudo-hybrids, because *T* is somewhat "married" with its transposed matrix, this resulting from comparison of integrations on two orthogonal axes. These pseudo-hybrids give in turn pseudo-hybrids with the clones, etc ... The population increases and doubles suddenly (simulating the chaotic behaviour of a baby-boom!).

Note. with TRYAPL2, starting from a 16x16 matrix, (e.g. **ROBY** or the **RAW_BEE**, see the Appendix), one may watch hundreds of generati;ns. The automaton also runs "as is" in APL.68000 Level II. with APL*PLUS II, strand notation is accepted, but multiple assignment is not; so, after a slight change in the program, one may go for lunch and come back after several thousand iterations which will have built a surprising collection of clones and hybrids the intimate structure of which will require many days for a deeper study of all the details and to set up an inventory of the charming creatures.

Conclusion

This simple models are based on theoretical considerations only. No ad-hoc assumptions are made (e.g. about various number of neighbours as in Conway's automaton which, surprisingly, produces babies with 3 genitors). No number except 0 or 1, or arithmetic function is used. (Does Nature know or need arithmetic functions?). Combinations of I-step shifts, ≠ and integrations by ≠\ on different axes already produce a very coherent but complex universe of strange "beings". Adding symmetries, which also preserve information, will lead to increase complexity, then the variety of produced shapes. If now, we let two or several populations from different origins interact, and, for extra fun or intellectual curiosity, allow mutations, what shall we obtain?

Anyway, it is a pleasure to work in APL and APL2, which have the immense privilege of offering all the necessary tools in order to imagine, write and simplify in parallel various concise, pedagogic and easy-to-test models (now even for free) on all cheap computers; the noticeable properties of binary integration could not have been found, studied and applied without Iverson's powerful notation which contains, as mathematically complete, the full kit of elementary bricks and idiomaticconstructs from which it was, however, difficult, to extract the really important quarks of information processing, i.e. those which may be applied directly to biology and physics, dropping everything which may alter the informational content of any measurement or genetic patrimony.

If the proposed models indeed mimic what biologists Observe, and for which there are either no other satisfactory models, or only models based on various postulates and tautologic statistical assumptions, let us add that ≠ and ≠\ PRESERVE the entropy of information, although they act exactly like Maxwell's demon; a happy end would be brought to the controversy about the entropic paradox of living processes, Which, by the increase of organisation they induce, seemed to be somehow in contradiction with the second principle of thermodynamics. The proposed models could appear as a step towards the proof that *all living processes are indeed* ***iso-entropic***.

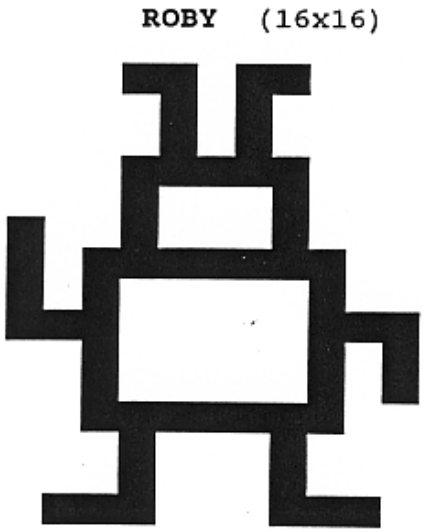
Appendix: Data for the Automaton

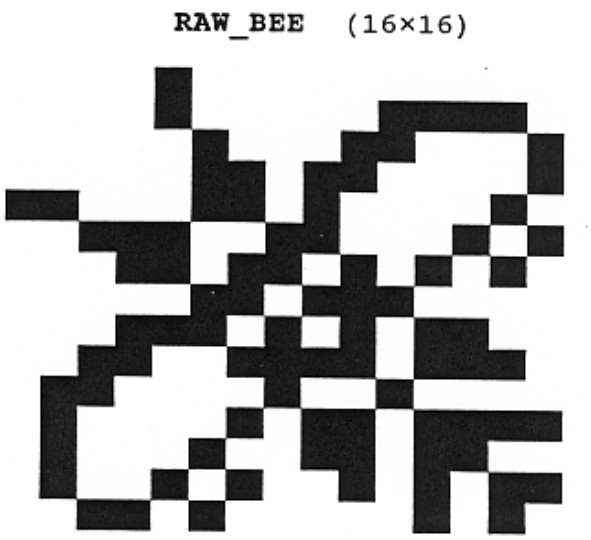
Any Boolean matrix can be used. However we suggest to test first square matrices the dimension of which is a power of 2 e.g. 16x16.

A unit matrix such as `U←16 16⍴17↑1` will suffice. `⌽U` is another good test. We also suggest: `16 16⍴16⍴1` (a plain horizontal line of pixels).

If you have a scanner, you may try your own photograph or your mother-in-law's. Here are two nice candidates, **ROBY** the robot and the **RAW_BEE**, which is the cursor-sprite of the ATARI when the machine is running (other possibilities would be

WINDOWS's sand-glass, the Macintosh's watch or characters from the screen or printer's fonts).





One may also catenate horizontally or vertically these matrices, in order to "marry" them. ROBY is asymmetric, but the RAW_BEE is diagonal-symmetric. For $N=\square IO$, every step whose number is an odd multiple of 8 produces siamese twins. For even multiples of 8 (i.e. multiples of 16), the original reappears.

References

[l] G.A. Langlet, "Physique et Algèbre de Boole", Les Nouvelles d'APL No 3, (Avril 1992) [in French].

[2] G.A. Langlet, "The Fractal Laws" of Genetics, SUbmitted to BIOMATH (1992).

[3] G.A. Langlet, "Towards the Ultimate APL-TOE", accepted at APL92 (St Petersburg), APL Quote-Quad, ACM (USA), Vol. 22, (July 1992).