APL-CAM Journal, Vol. 16, No.1, 16 January 1994, pp. 53-73.
Received: 21 December 1993.

GOD SAVE THE QUINs!

Gérard A. Langlet

CEA/DSM/DRECAM/SCM/LIT

C.E. Saclay

F-91191 Gif-sur-Yvette

Abstract:

This paper is intended to prepare a "Round Table" about Binary Algebra at APL94 in Antwerp. Based on the solutions of a complex puzzle, it shows the efficiency of matrix inversion when this latter is considered no more in floating-point arithmetics, but with pure binary or modulo2-integer algebra concepts, allied with the power of APL implementations on present-time micro-computers.

GOD SAVE THE QUINs!

Gérard A. Langlet

CEA/DSM/DRECAM/SCM/LIT, C.E. Saclay, F-91191 Gif-sur-Yvette

Abstract

This paper is intended to prepare a "Round Table" about Binary Algebra at APL94 in Antwerp. Based on the solutions of a complex puzzle, it shows the efficiency of matrix inversion when this latter is considered no more in floating-point arithmetics, but with pure binary or modul02-integer algebra concepts, allied with the power of APL implementations on present-time micro-computers.

The Quins

The Quin-puzzle is, inter alia, described by Dumontier in "Les Nouvelles d'APL", No 8 (Oct. 1993), under the name "Quinto", restricted to small examples with N $\in$ 3 5.

Given a square board with N×N sub-squares, clicking any such sub-square will invert its colour, as well as the colour of the four horizontal and vertical adjacent sub-squares, (provided these lie within the main square or board): every click inverts a swiss cross:

So, for a 3×3 board, the solution is almost trivial and may be deduced intuitively, by pure symmetry: What holds for one corner shall hold for all four corners. What holds for the sub-square that lies in the middle of the first row shall hold for the three other sub-quares which lie in the middle of the last row, and in the middle of the first and of the last columns; so, few combinations have to be tested before one finds that the st Andrew cross is the unique solution:

The pattern on the right is a proof that clicking the sub-squares delimited by the ×-cross fills the board.

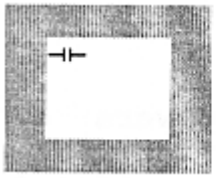
Conversely, if the colour-inverting pattern were the ×-cross, a solution would be (by duality) the former inverting pattern, a Swiss (or Savoy or +) cross ("sharing cross"?).

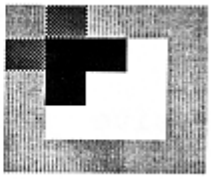
Now, given a 4×4 board, how can one find the solution? Then, for a N×N board, with N much greater than 4 or 5, how can we 1) know if there is a solution, 2) find it in the most convenient way?

Some special mathematical developments have to be thought of...

But, from the beginning, we should remain aware of the fact that such a puzzle - like many other ones - is purely binary; so, using anything else than pure binary algebra, then bit matrices, shall appear as a waste (of time, of thought, of computing power, of mathematics).

Let us go back to the 3x3 board and describe the 9 possible clicks by their effect on the whole board plus one virtual row and column on every side. In other words, the original 3x3 board is replaced by a 5x5 one. A click (symbolised by ⊣⊢ in the upper leftmost corner of the 3x3 board on the left only inverts 3 sub-squares in the real board (in solid colour) and 2 sub-squares outside (in half-tone grey in the 5x5 virtual board):





```
      |       |
  _ 0 1 0 0 0 _
    1 1 1 0 0
    0 1 0 0 0
  _ 0 0 0 0 0 _
    0 0 0 0 0
      |       |
```

This click corresponds to the binary matrix mask hereabove.

Once ravelled into a 25-item vector, the mask becomes:

```
0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
          ↑ ↑ ↑     ↑ ↑ ↑     ↑ ↑ ↑
```

The arrows point to the bits which belong to the 3×3 board.

The 2nd mask, for a click in the middle of the 1st row of the 3×3 board, corresponds to `¯1⌽` applied to the 1st mask:

```
0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
```

The 3nd mask, for a click right of the 1st row of the 3×3 board, corresponds to `¯1⌽` on the 2nd mask or to `¯2⌽` on the 1st mask:

```
0 0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
```

For the second row, we have, respectively, `¯5⌽ ¯6⌽ ¯7⌽` of the 1st mask, while for the 3rd row, the shifts become `¯10⌽ ¯11⌽ ¯12⌽` respectively.

Dropping the first 6 items and the last 6 items of all these 25-item vectors, then items which do not belong to the 3x3 board, one obtain the 9x9 matrix of the binary rule for all the inverters. Then, we go back to the solution of the MAGIC puzzle [cf. "Les Nouvelles d'APL" No 7 (May 1993) as well as "VECTOR" Vol. 10 No 1 (July 1993): two papers by M. Day, and G. Langlet).

M `⍴` M is a binary matrix with 81 items:

```
1 1 0 1 0 0 0 0 0
1 1 1 0 1 0 0 0 0
0 1 1 0 0 1 0 0 0
1 0 0 1 1 0 1 0 0
0 1 0 1 1 1 0 1 0
0 0 1 0 1 1 0 0 1
0 0 0 1 0 0 1 1 0
0 0 0 0 1 0 1 1 1
0 0 0 0 0 1 0 1 1
```

which is best displayed, using semi-graphics, as shown on the right, with the help of the paper- sparing function "FG", given in the Appendix, that will prove useful for greater values of N, formating the QUINs.

So, if I is the modulo2 inverse matrix of M, then the solution will be given by: `3 3⍴≠/I` which is the solution of an integer system of 9 equations modulo 2, knowing that the starting pattern is `3 3 ⍴ 0` (empty pattern, then `9⍴0` when ravelled), and that the goal is `3 3⍴1` (full pattern i.e. `9⍴1` when ravelled), the modulo2-difference or binary difference between both patterns being the same as the full pattern: All the second members of the equations are l's, hence the `≠/` shortcut.

If X is the unknown solution, ravelled, then `X≠.∧M` (or `2|X+.×M`) shall return a vector of l's.

In this case, a search for the successive matrix powers of M immediately shows (it is almost intuitive), that `M≠.∧M≠.∧M≠.∧M` is a unitary matrix, so that matrix inversion becomes useless: I has to equal `M≠.∧M≠.∧M` i.e. the cube of M. So, a very short ISO-APL expression gives the unique solution for `N≡3` (problem QUIN 3); it is `3 3⍴≠/M≠.∧M≠.∧M` once one has built matrix M, in fact directly as the reduction both along the rows and the columns, by the same binary mask B:

```
0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0 0 0
```

of a larger 25×25 matrix which takes care of the corners and of the sides, in which each row is a circular rightwards-shift by one item of the clicking effect of the Swiss cross for the 25 sub-squares of the larger matrix.

If N is the size of the puzzle, the size is N+2 for the virtual pattern matrix; however, the patterns may be obtained - cf. the QUIN function in the Appendix, even without circular shift.

When N grows, the number of items within M will always be `N*4`. Already, this will prohibit the use of floating-point arithmetics; e.g. with ⌹ (the domino), the result of which is always "real", requiring, in APL, 64 bits per item. Even if it were possible to use ⌹ as well as nice functions for computing the determinant - as proposed by M. Day for MAGIC, and by M. Dumontier for the Quinto, and if the workspace offered a very large size, the domino would produce WRONG results, because of floating-point catastrophic roundings (truncations) all along the huge number of operations which would be necessary: this domino is a black box for the APL user... (the same holds for the inverting subroutines of mathematical FORTRAN libraries).

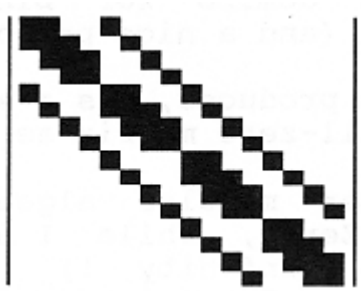
And the "almost-intuitive method by symmetry", which allowed to solve MAGIC or the QUIN for N¦3, raising M to some low power, cannot be general:

For a binary square matrix M with order `N×N` or `N*2` i.e. with shape `2⍴N×N`, then, if the matrix is invertible, the maximum power so that M raised to this power equals a unitary matrix, is: `¯1+2*N×N` or `¯1+2*N*2`: such a huge value, e.g. for `N≡20`, prohibits the attempts of replacing matrix inversion by products. Moreover, when matrix M is not invertible, no power of M will ever equal a unit matrix!

Indeed, a new method for binary matrix inversion had to be worked out...

More mathematical difficulties

Here is matrix M for $N \equiv 4$ together with its graphic look ↓↓↓

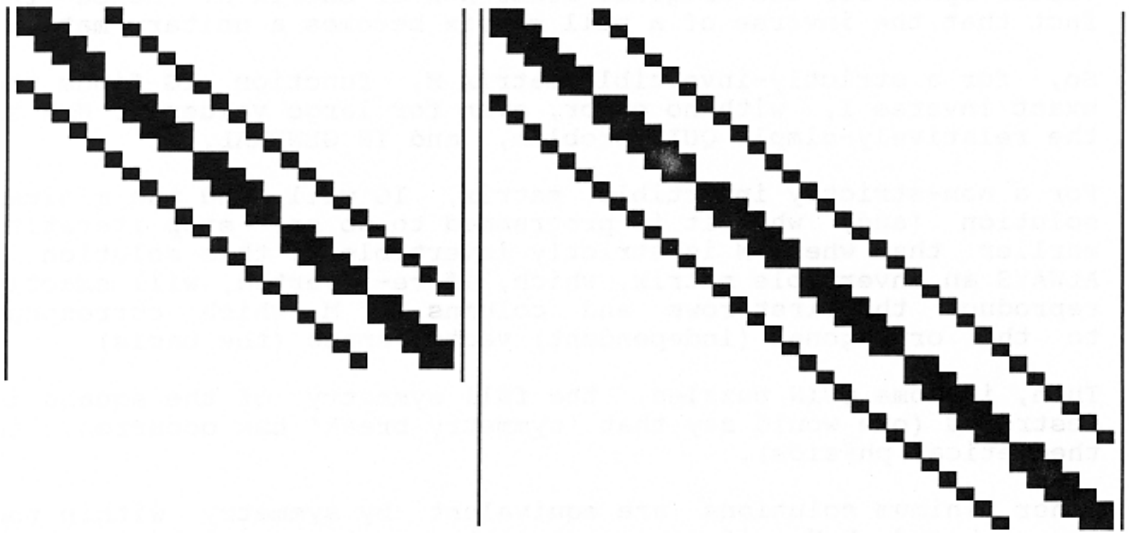


```
1  1  0  0  1  0  0  0  0  0  0  0  0  0  0  0
1  1  1  0  0  1  0  0  0  0  0  0  0  0  0  0
0  1  1  1  0  0  1  0  0  0  0  0  0  0  0  0
0  0  1  1  0  0  0  1  0  0  0  0  0  0  0  0
1  0  0  0  1  1  0  0  1  0  0  0  0  0  0  0
0  1  0  0  1  1  1  0  0  1  0  0  0  0  0  0
0  0  1  0  0  1  1  1  0  0  1  0  0  0  0  0
0  0  0  1  0  0  1  1  0  0  0  1  0  0  0  0
0  0  0  0  1  0  0  0  1  1  0  0  1  0  0  0
0  0  0  0  0  1  0  0  1  1  1  0  0  1  0  0
0  0  0  0  0  0  1  0  0  1  1  1  0  0  1  0
0  0  0  0  0  0  0  1  0  0  1  1  0  0  0  1
0  0  0  0  0  0  0  0  1  0  0  0  1  1  0  0
0  0  0  0  0  0  0  0  0  1  0  0  1  1  1  0
0  0  0  0  0  0  0  0  0  0  1  0  0  1  1  1
0  0  0  0  0  0  0  0  0  0  0  1  0  0  1  1
```

Since this matrix is still small, we can consult the gentle "domino-astrolog", just as a matter of curiosity:

```
      R←⌹M
DOMAIN ERROR
      R←⌹M
        ^
```

Here are the graphic outputs for matrices M with $N \equiv 5$ and $N \equiv 6$:



The "domino astrolog" also reports 'DOMAIN ERROR' for $N \equiv 5$, but works - it does not report any error - for N=6. We do not care about R itself; the astrolog was used as a failure test... ONLY.

Anyhow, for the moment, the fact that the determinant of M may be 0 or different from 0 remains... impredictible. But we have quickly found (without computing the

determinant), because ⌹ immediately returns an error - which is not, for these small values of N, a LIMIT error - that M may be NOT-invertible, in quite a lot of cases. So, in the GENERAL case, matrix inversion will fail... with or without our domino!

Solving the paradox

Function IG (matrix Inverse in the General case) is a revisited domino for binary - or, rather, integer-modulo2 matrices (and a nice proposal of extension for APL itself).

IG always produces, as a solution, an invertible matrix, even with an all-zero matrix as its argument!

In integer modul02 algebra, (in the set Z/2Z), 0 means number 'Zero', while 1 means 'Everything which is not 0', INCLUDING infinity!): 1 is THE infinite for 0, (its inverse)... There is no sign in such an algebra, so that there is only one type of infinity. When a null square matrix also represents 0, then a unitary matrix represents 1, and also the inverse of the null matrix: the secret lies within a definition.

The Algorithm which is used in the IG function, is based on APL-T.O.E. concepts (cf. APL92, St Petersburg): Every algorithm should reduce to combinations of ≠ and ≠\ (≠/ being a sub-case of ≠\). This had to hold also for matrix inversion, when M is a binary matrix, considered as containing (Z/2Z)-data.

IG is an optimiser: without knowing the value of the determinant (which is 1 for all strictly modul02-invertible matrices, and 0 otherwise), it will automatically find the minimum subset of orthogonal (independent) vectors of the matrix vector-space, and complete, if non-independent vectors are found, by vectors which will form a subsidiary independent vector-space for the original order $N \times N$ of matrix M. Hence the fact that the inverse of a null matrix becomes a unitary matrix.

So, for a strictly-invertible matrix M, function IG finds the exact inverse I, with no error, even for large values of $N \times N$ in the relatively-simple QUIN problem, and IN GENERAL.

For a non-strictly invertible matrix, IG will find the minimal solution (and, when it is programmed to do so, stop iterating earlier than when M is strictly invertible!); this solution is ALWAYS an invertible matrix, which, if re-inverted, will exactly reproduce the first rows and columns of M which correspond to the orthogonal (independent) vector-space (the basis).

Then, in some QUIN puzzles, the full symmetry of the square is destroyed (one would say that 'symmetry break' has occurred, in theoretical physics).

Other minimum solutions are equivalent by symmetry within the square board N×N, if one rotates the pattern which has been found, or if one reverses it (using, in APL, functions ⌽ ⊖ ⍉ and their combinations).

Of course, because all matrices M are diagonal-band sparse matrices, there are other known ways of inverting them.

The main goal of the present paper was not puzzle solving, but General Problem Solving, introducing an algorithm for the inversion of ALL binary/modulo2 matrices, which would NEVER signal any error, and find orthogonal sub-spaces, even in the case of classical non-inversibility, while all methods with uncontrolled roundings will undoubtfully fail, for large cases.

The QUINs are naive puzzles (at least, in "MAGIC"/"Inversion Diabolique", all masks, also named "inverters", are different). One can try to build more sophisticated QUINs, with a shape of the mask that will depend on each sub-square: in such a case, matrix M will no longer look like a band-matrix; hence the need for a general inverter. But this problem is nice for checking.

In order to prove that giant puzzle solving may be undertaken on a PC with an interpreted language (a few lines of APL code) in a small workspace, Appendix 1 presents the panoply of QUINs, while Appendix 0 presents the APL functions.

In despite of limitations imposed by IBM, and of memory-adress limits in the PC-AT machines, QUIN 15 (forming a binary matrix with order 225 ie 50625 items) can be solved even with TryAPL2 for free (because TryAPL2 codes binary items in bits).

Middle-order QUINs were obtained (and saved) using APL.68000 on the Macintosh and the Atari, high-order ones with APL*PLUS II on 386 (Compaq) and 486 (Dell) PC-compatible machines.

QUIN 47 requires matrix inversion of a non-invertible matrix with 4,879,681 items. The invertible matrix of QUIN 48 contains 5,308,416 items. The inaccurate result of the domino, if ⌹ could work for a matrix with 2304 rows and columns (?), would require 42,467,328 bytes of storage for its result alone. This number is already beyond the capacity of all usual PC's and large enough to make people reluctant to lose their time with floating-point arithmetics. And 48 is still a small number...

Some patterns are amusing (cf. the 'writing-machine' for $N \equiv 9$, the 'priest' for $N \equiv 44$) .

All are, anyhow, aesthetical.

Reference

Some more details will be given in a paper entitled "La Quintescence du Quinto" especially about matrix powers, in "Les nouvelles d'APL" No 9 (Dec. 1993). Other references are given in the text. A fully-documented software will be available at the Software Exchange Booth, at APL94, Antwerp, Belgium, for several APL implementations.

Thanks

Louis Metayer shall be thanked for help, getting the higherorder QUINs on a fast 486 machine with a 66 MHz clock (and he is still going on, in order to increase his collection).

IBM and all the team which realised TryApl2 as a free APL interpreter also deserve a special mention: for the first time, the capabilities of APL & APL2 can be shown with a powerful and pleasant implementation which handles binary items in bits, so that many teachers of mathematics, students and physicists can discover the courses about binary/modulo2 matrix algebra we have developed, with plenty of consequences in physics and computer science, as well as in biology, especially in genetics. (For the moment, this (free) course is in French)/

Appendix 0 Functions

```
      ∇R←QUIN N;E;F;G;H;L;P;Q;V
[1] Q←N×P←N+1+1◇ F←1+1↑L←G,G←×/H←P,P◇ E←N,N◇ V←G↑QρP↑Nρ1◇ L←-P+1
[2] M←H↑3 3ρ0 1 0 1 1 1 ◇ M←,M ◇ M←F↑M ◇ M←LρM ◇ M←V≠M ◇ V←L⌽V
[3] M←V/M ◇ X←IG M ◇ S←EρX←≠/X ◇ R←FG S ◇ R←R,'|',FG Eρ≠≠X≠M
∇
```

M and X are global. The function is a sequence of short ISO-APL parenthesis-free expressions, so as to prevent "ws FULL". It can become a one-line function.

```
      ∇M←IG M;D;I;J;K;L;N;S;V
[1] I←⎕IO≠J←l ◇ L←⍳D←1↑S←ρM ◇ K←ρI↓V←D↑1 ◇ M←M,SρV,0
[2] 'I←,M[N←V/L;] ◇ J←V≠M[;I⍳1] ◇ J←ρI←J/L
      ◇ M[I;]←M[I;]≠M[J←J/N;] ◇ V←0,¯1↓V' wh 'K≠N'
[3] V←M[;L] ◇ V←S↑V ◇ V←1,V ◇ V←≠\V ◇ V←∧\V ◇ V←+/V ◇ V←⍋V
[4] N←0,D ◇ M←N↓M ◇ M←M[V;]
∇
```

Several versions were tested. IG can also become a one-line fn. Lines [3] and [4] can be made much shorter; again, decomposing into elementary statements is an insurance against "WS FULL". ⍋ on arrays is not used although it could be: a) because it is not an ISO-standard, b) because it may produce WRONG results, on large matrices, due to - again - the deficiencies of floating arithmetics, c) because, on permutation matrices, ≠\ is much faster (there are some fundamental connections between ≠\ and ⍋⍋: "grade" and "grade grade" themselves ARE connected to binary matrix inversion, so that the problem is recursive!).

The main part of CP time is consumed by line [2]; so, efforts to use ≠ only as THE PRIMITIVE in the algorithm were successful; (diadic ⍳ can also be replaced by ≠ and compression by a mask) .

```
      ∇ ΔE wh ΔL;ΔC                  ⍝ the "while" function
[1] ΔC←⍎⍕ΔL ◇ ⍎ΔC/ΔE ◇ →ΔC/1        ⍝ executes ΔE while ⍎ΔL
      ∇                              ⍝ remains TRUE.
```

Note. The "wh"-loop condition may be replaced by a condition that will stop the iteration as soon as matrix M is not invertible; this is detectable in `I⍳1` and best realised by error-trapping, replacing "wh" by a "while_no_error_occurs" fn; then, lines [3] and [4] will, as an example, return a unitary matrix as the generalised inverse of a null-matrix in the extreme case of non-strict-inversibility. A problem with the domino (already noticed by the late Gilles Martin at APL80, Noordwijkerhout/Leiden (NL», is that one loses some useful results which are hopelessly replaced by the fatal "DOMAIN ERROR" message.

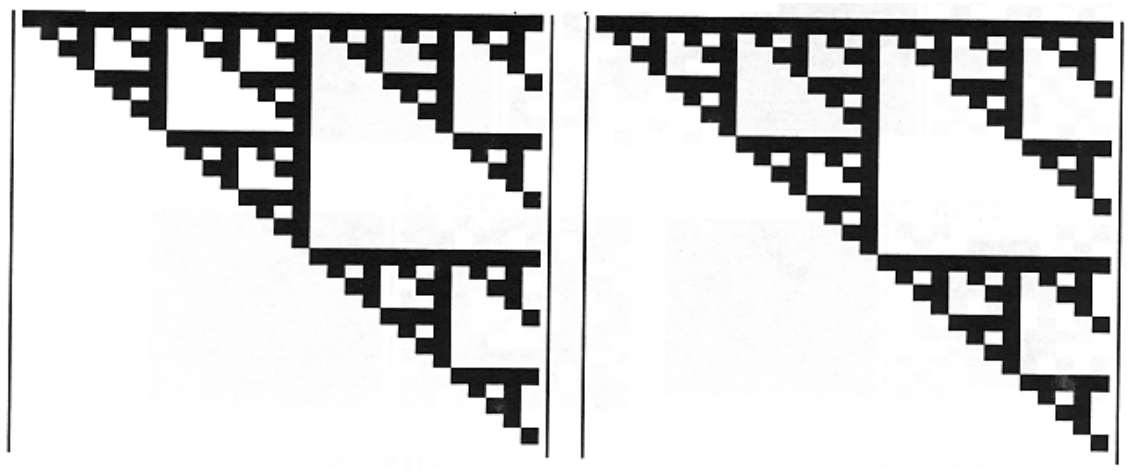
Such a WHILE-loop may be semantically considered as a scanning operation along the columns of M. One can as well scan the rows of M. The ≠ while-scanning is a temporal form of differencescanning in parallel (≠\ is more present in the IG function than one may think!). within the function, the original M is stepwise-transformed into a permutation matrix, while in the reciprocal space, a unitary matrix suffers from the same scanning, and, stepwise, becomes the inverse matrix of M, except for a permutation of its rows which is rearranged at the end by the reciprocal (just transposed) permutation matrix, obtained as the left part of M. One can also use two matrices instead of a single "CinemaScope" M within IG. Then, the APL version becomes slightly slower, while the risk of "WS FULL" decreases.

Function FG is trivial, a good exercise for the reader with the semi-graphic characters ' ▄▀ ' of the PC-⎕AV. (Such graphic outputs required, on the Macintosh, to modify the APL.68000 font).

```
      ∇R←FG B;K;L;N;⎕IO
[1] N[⎕IO←0]←K←K+2|K←1↑N←⍴B ⋄ B←N↑B ⋄ L←1≠K←K⍴1 0 ⋄ R←K⌿B←N↑B
[2] R←' ▄▀'[2⊥R,[¯.1]L⌿B] ⋄ ⍝ R←'|',R,'|'
∇
```

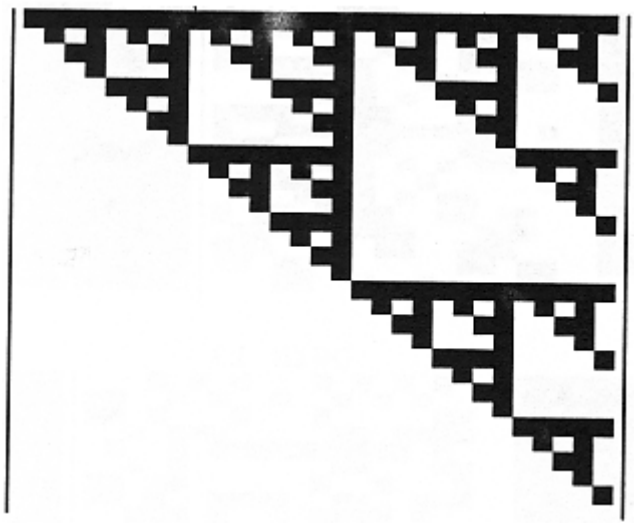
This is one of the possible versions. Removing the "⍝" allows to catenate a full vertical bar left and right to show that the output corresponds to a matrix, Formatted Graphically. Example:

```
      A←FG M←⌽GENITON 29 ⋄ A,' ',FG IG M
```



This example clearly illustrates the properties of "genitons", when ⌽ or ⊖ mirrored: these self-similar (Sierpinski) matrices are modulo2-self-matrix-inverses for all orders from 2 to infinity; the example with order 29 is chosen just to fit the page width. For such matrices (this was unknown before the "APL-TOE" paper at APL92), IG can be tested nicely, but is of... no use. The successive rows of this matrix M may also be obtained as the 29 successive iterates of ≠\ with initial B set to 32↑1:

```
      M←0⍴B←32↑1 ⋄ 'M←M,¯29↑B←≠\B' do 29 ⋄ FG IG M←⌽29 29⍴M
```



⍝ e.g. with the "do" function:

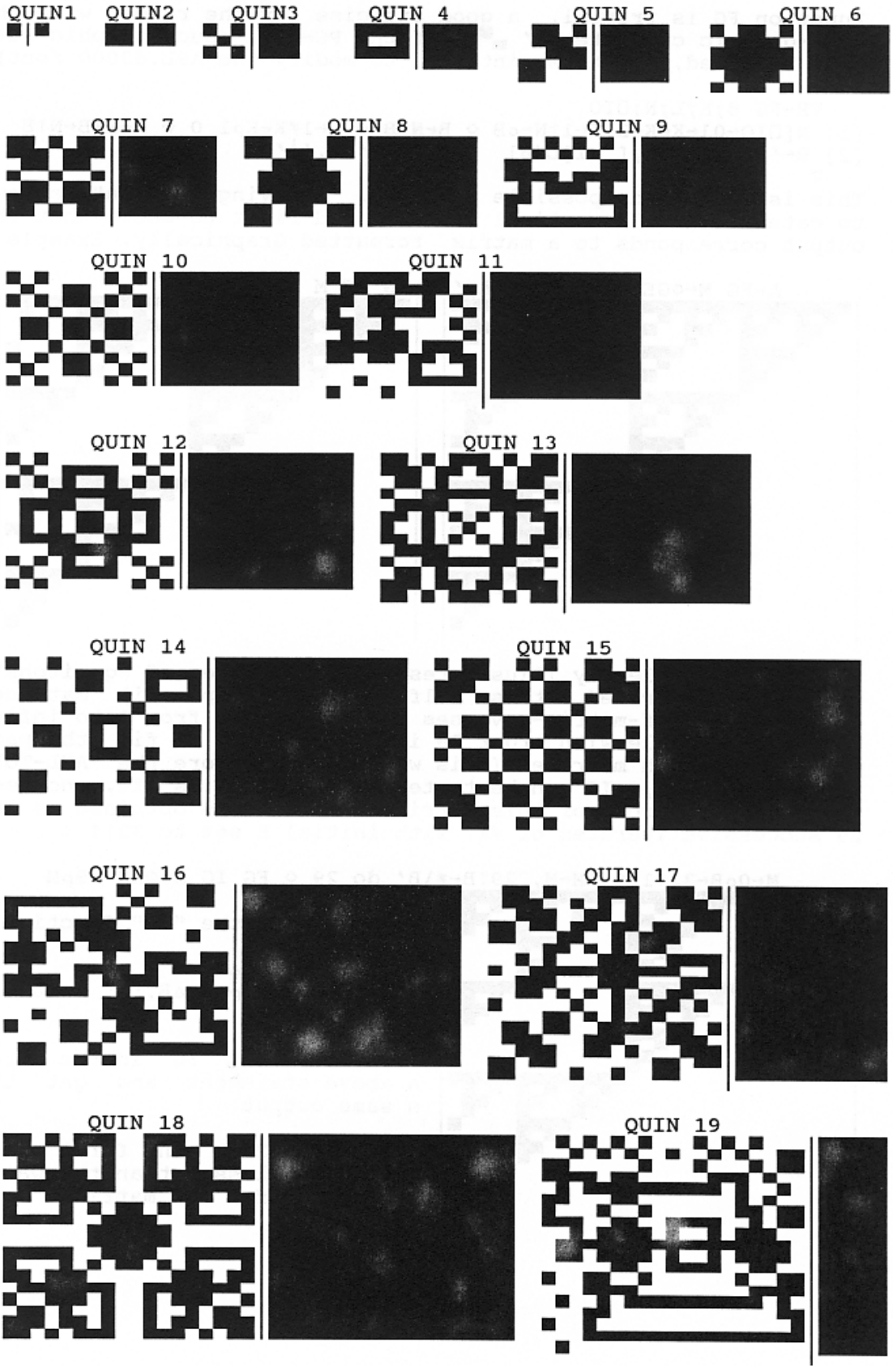
```
⍝      ∇ ΔE do ΔN
⍝ [1] ΔN←ΔN×⍴ΔE←ΔE,'⋄' ⋄ ⍎ΔN⍴ΔE

⍝      ∇
```

⍝ One may omit IG in the here
⍝ above statement and get the
⍝ same output
⍝ 32 is 2*⌈2⊛29 the first power
⍝ of 2 not lesser than the wanted order for the matrix.

```
                ≠\≠\≠\
```

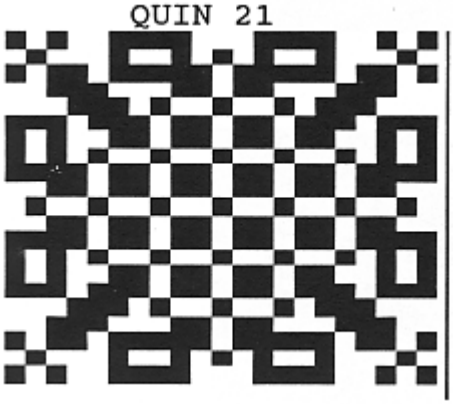
Appendix 1 The dynasty of QUINs



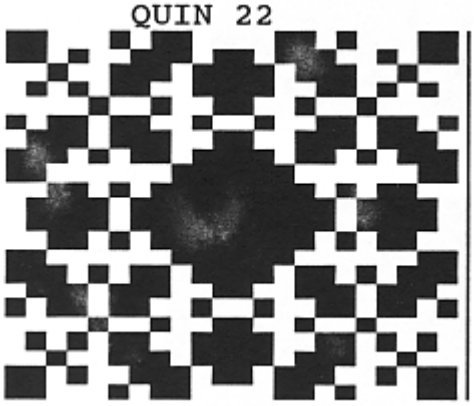
NB. À partir de ce point, pour cette version électronique, les zones de contrôle noires à droite des Quin ont été omises.

QUIN 20

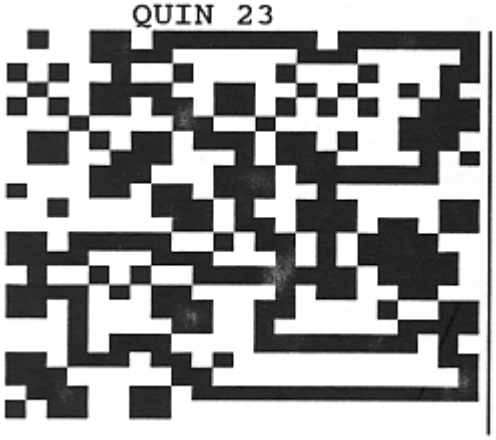
QUIN 21



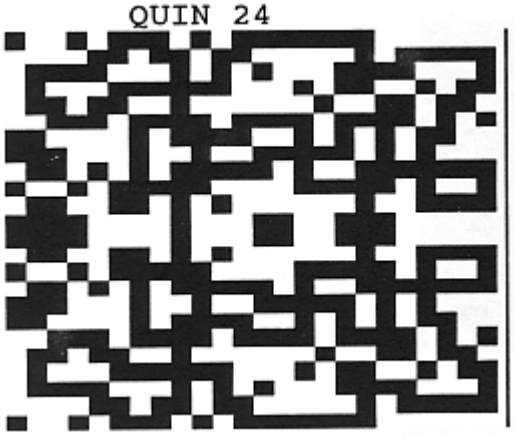
QUIN 22



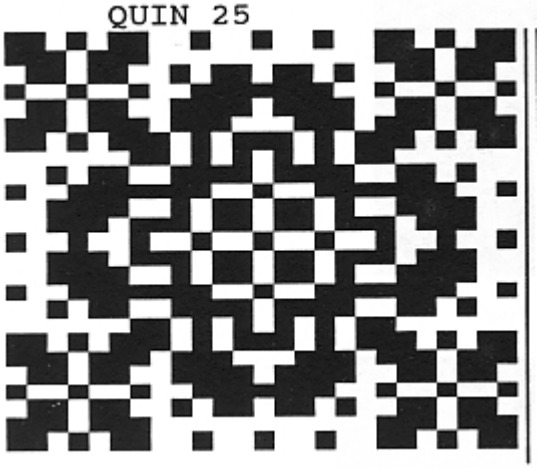
QUIN 23



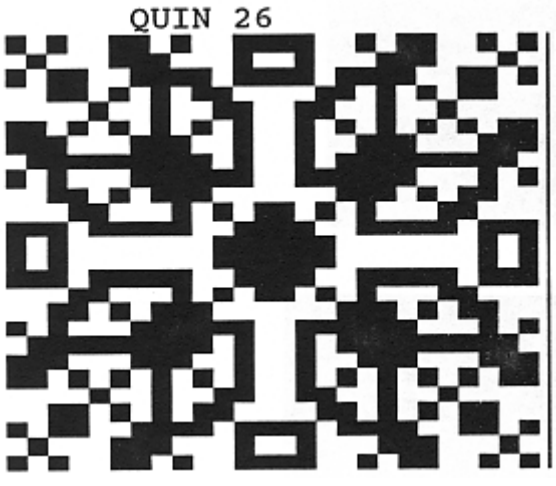
QUIN 24



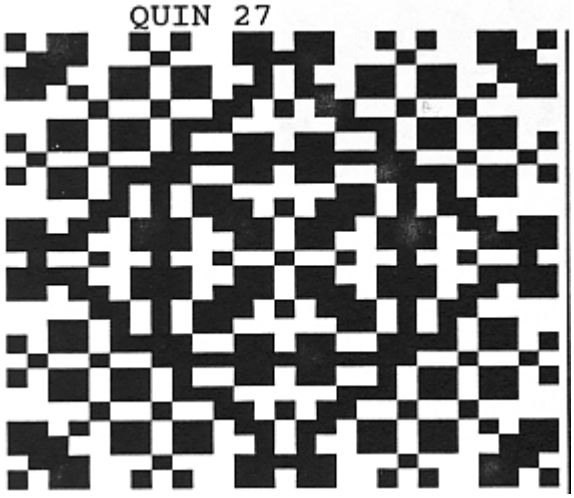
QUIN 25



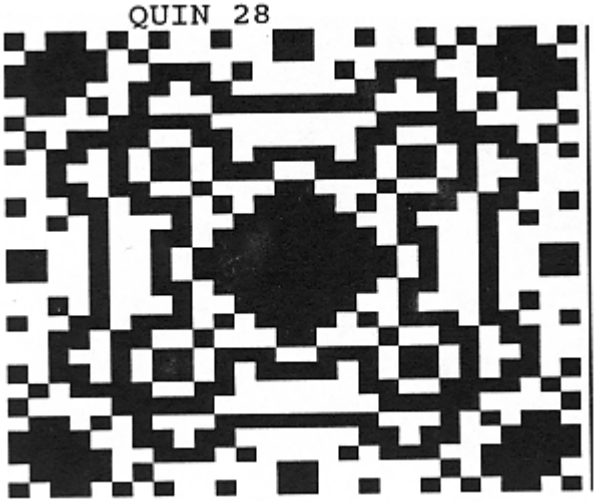
QUIN 26



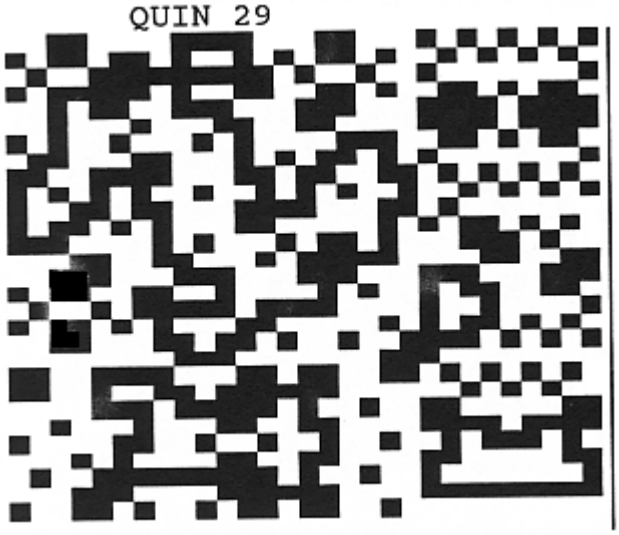
QUIN 27



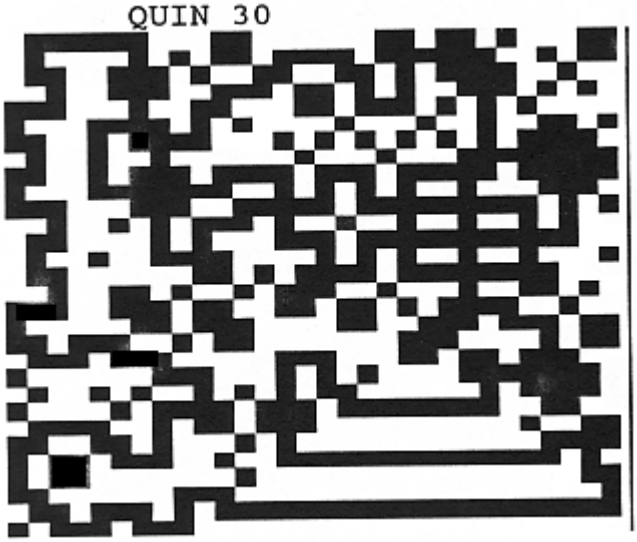
QUIN 28



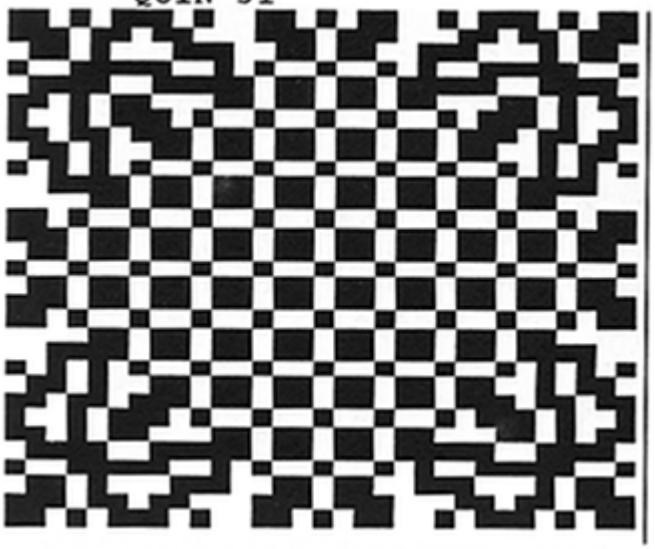
QUIN 29



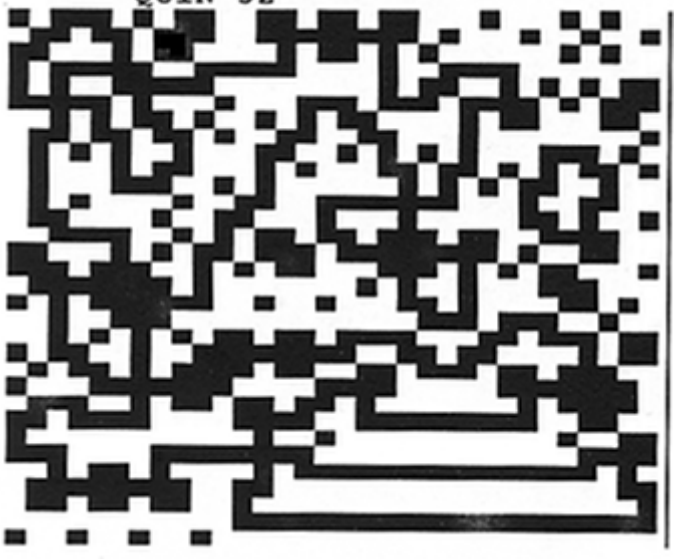
QUIN 30



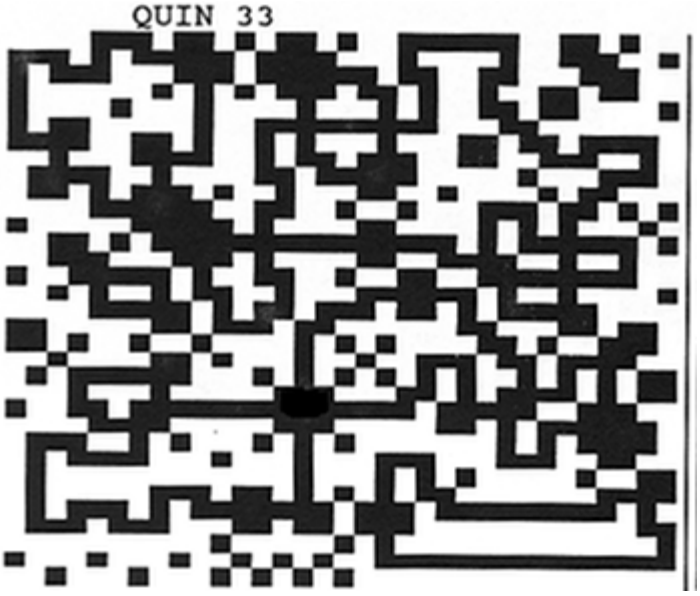
QUIN 31



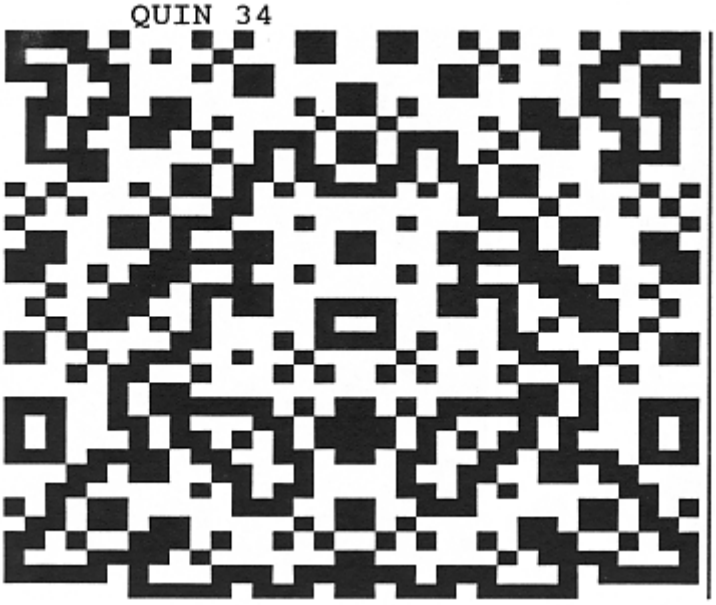
QUIN 32



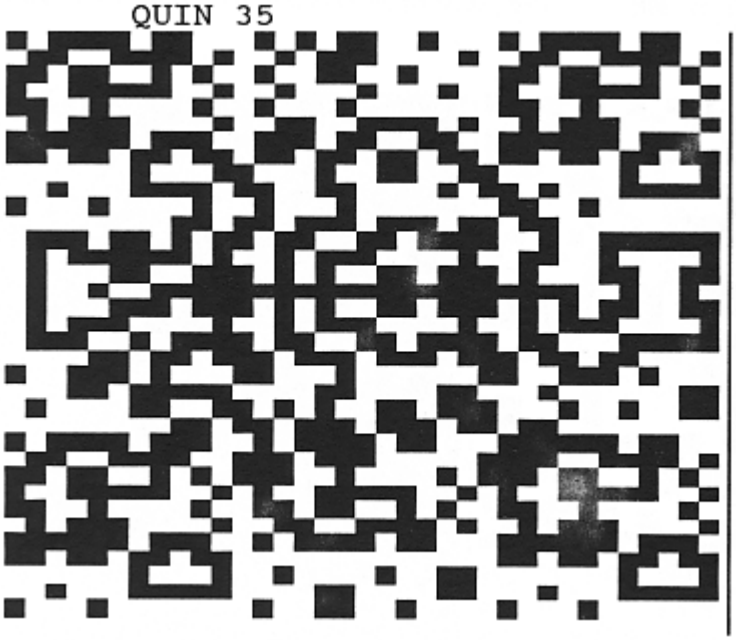
QUIN 33



QUIN 34

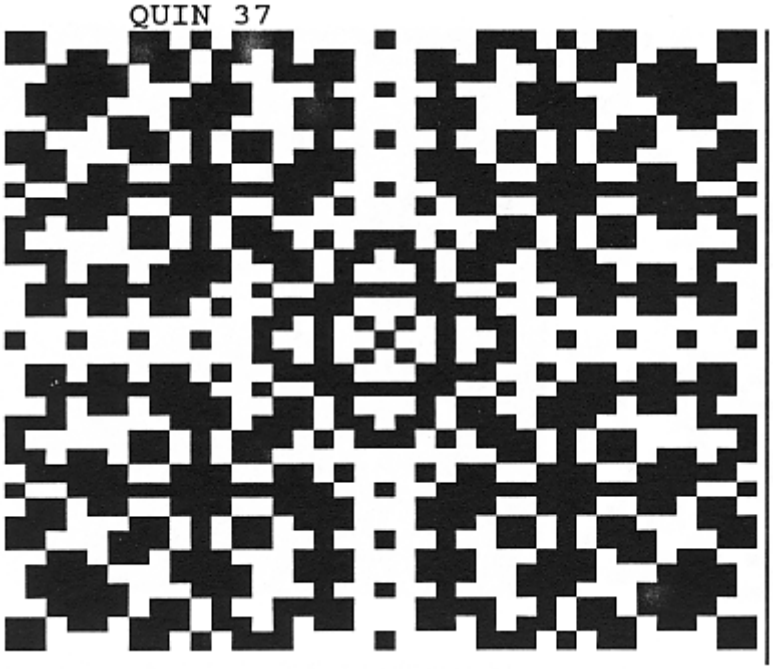


QUIN 35

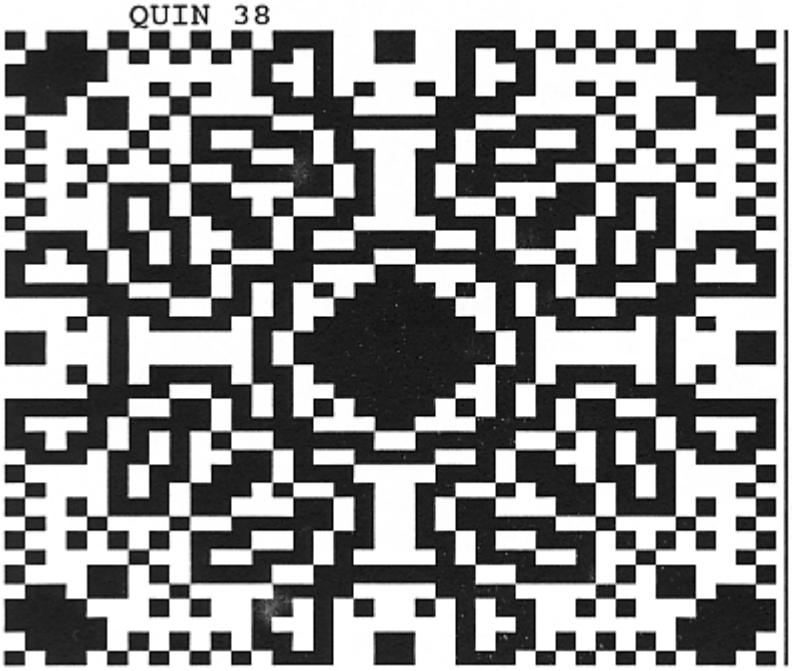


QUIN 36

QUIN 37

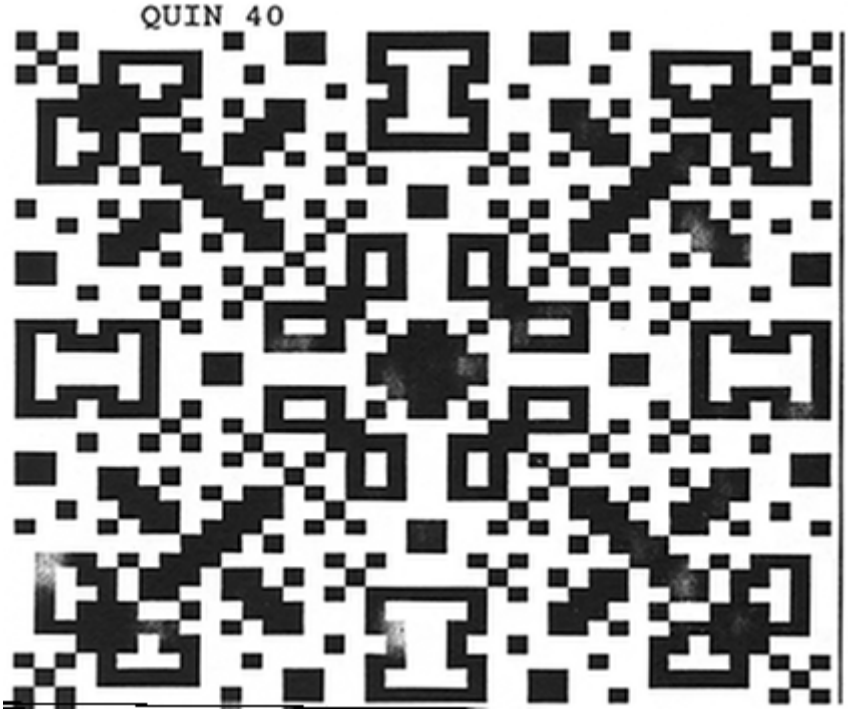


QUIN 38

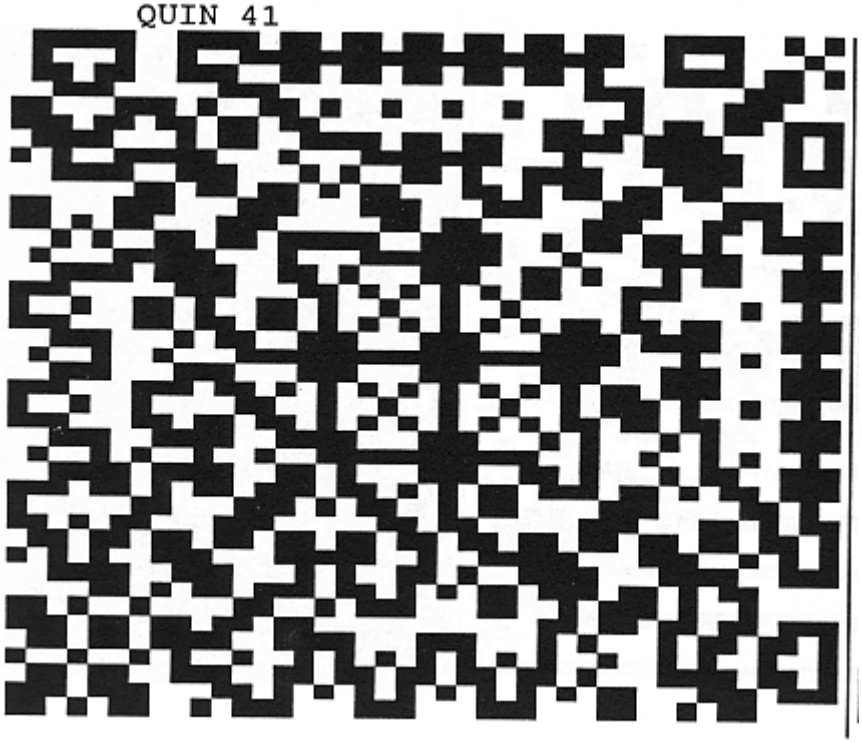


QUIN 39

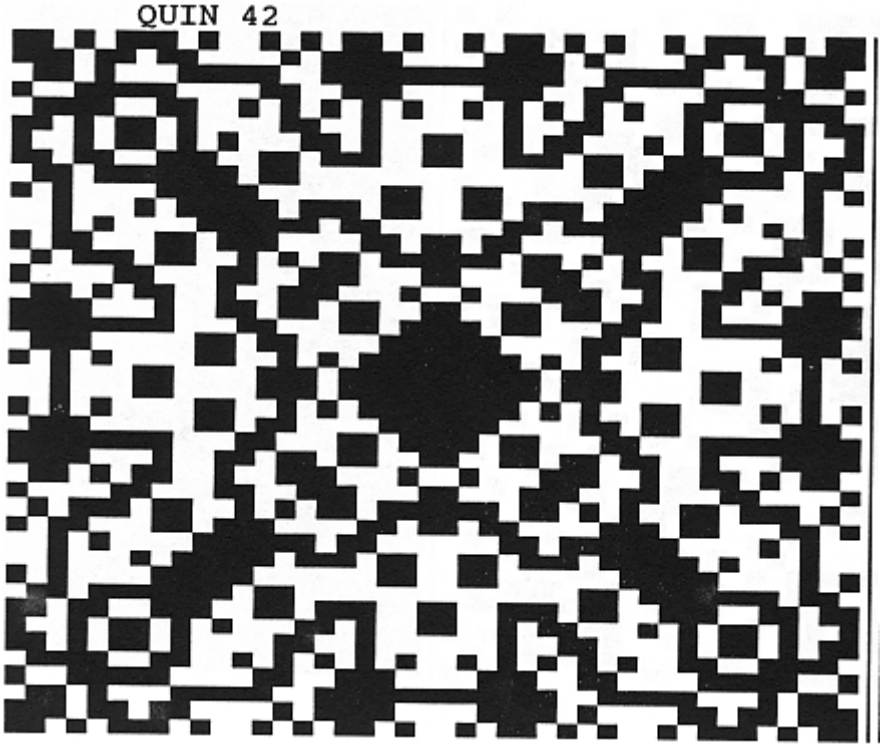
QUIN 40



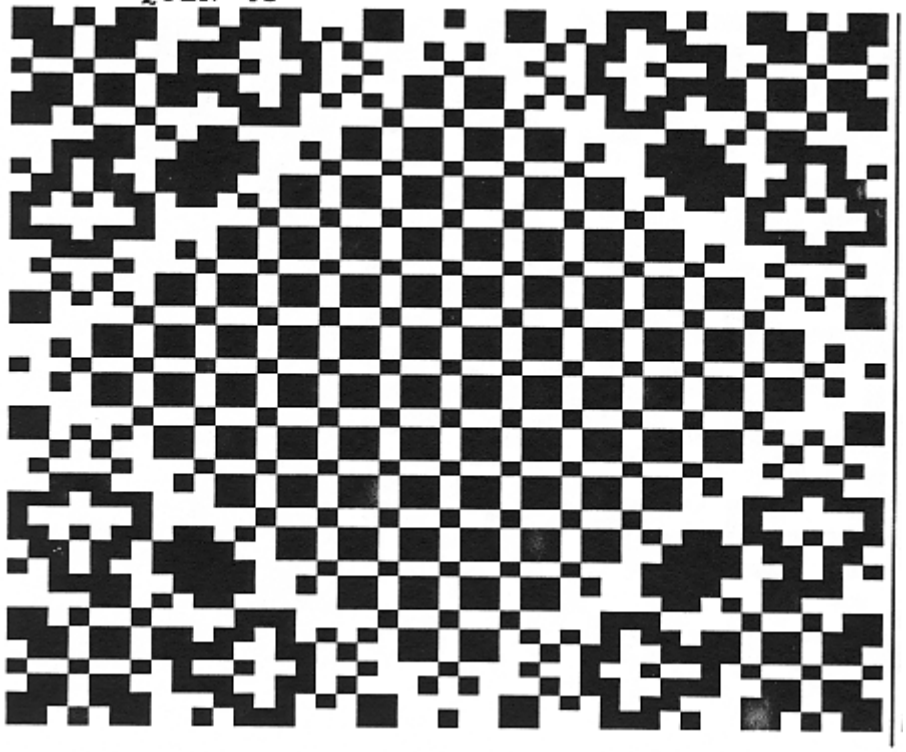
QUIN 41



QUIN 42

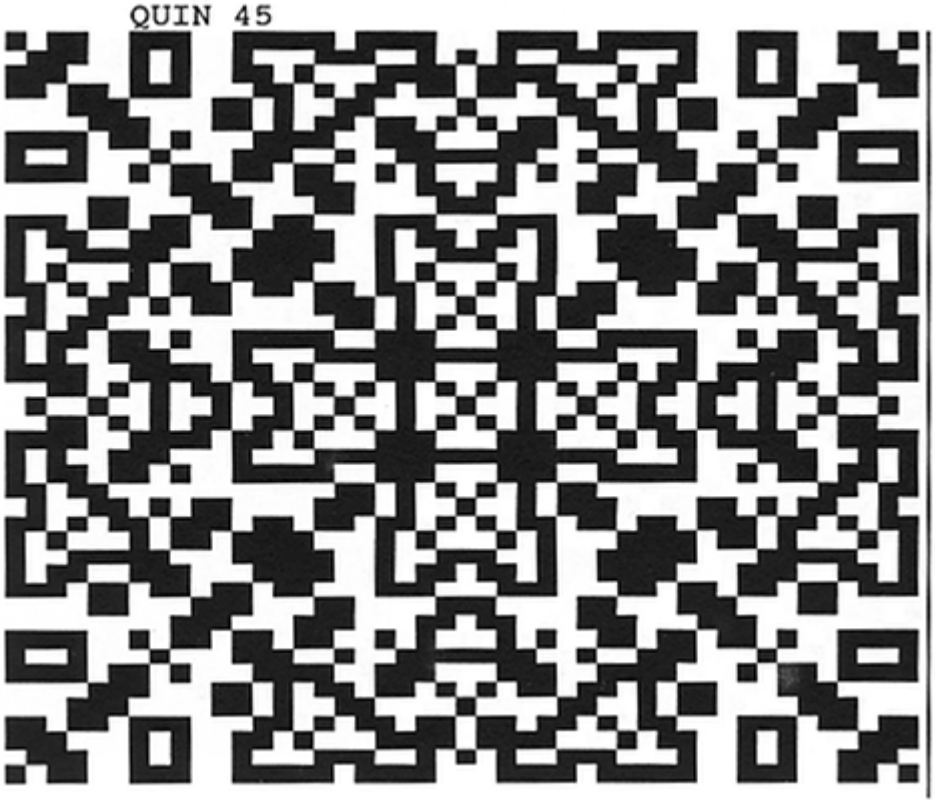


QUIN 43

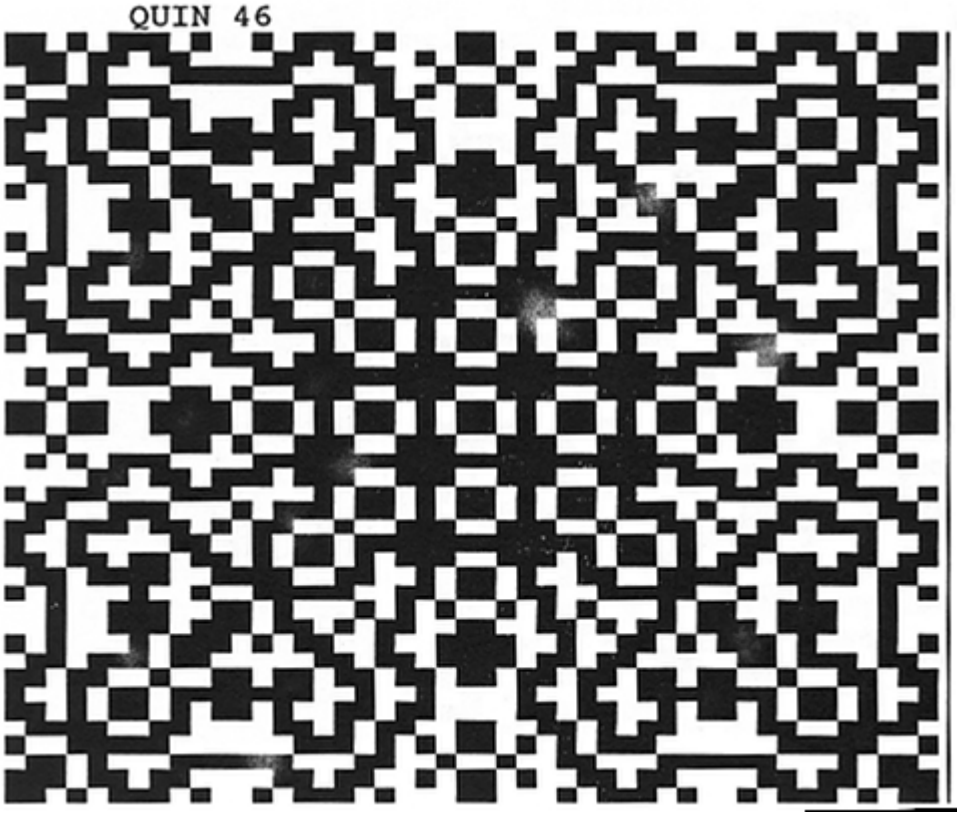


QUIN 44

QUIN 45



QUIN 46



QUIN 47

QUIN 48