Calculating ancestors in one-dimensional cellular automata

Juan Carlos Seck Tuoh Mora*
Centro de Investigación Avanzada en Ingeniería Industrial
Universidad Autónoma del Estado de Hidalgo
Ciudad Universitaria, Carr. Pachuca-Tulancingo Km 4.5 Pachuca, Hidalgo 42184, México

Genaro Juárez Martínez†
Departamento de Ingeniería Eléctrica, Sección Computación
CINVESTAV-IPN.
Av IPN 2508, Col San Pedro Zacatenco, México D.F. 07360, México

Harold V. McIntosh ‡
Departamento de Aplicación de Microcomputadoras
Instituto de Ciencias, Universidad Autónoma de Puebla
Apartado postal 461, 72000 Puebla, Puebla, México

Revised March 2004

Abstract

One-dimensional cellular automata are dynamical systems characterized by discreteness (in space and time), determinism and local interaction. We present a procedure in order to calculate ancestors for a given sequence of states, this procedure is based on a special kind of graph called *subset diagram*. We use this diagram to specify subset tables for calculating ancestors which are not Garden-of-Eden sequences, hence the process is able to yield ancestors in several generations. Some examples are illustrated using the cellular automaton Rule 110 which is the most interesting automaton of two states and three neighbors.

1 Introduction

A cellular automaton consists of an array of cells where each takes a value from a finite set of states. Every cell updates its value depending on the state of its neighboring cells, hence the global behavior of the automaton depends on the local interaction of the cells. Cellular automata were invented by John von Neumann to investigate self-reproducing systems [10] [1].

A relevant work by Edward F. Moore analyze Garden-of-Eden sequences [9], that is, sequences of states which can not be produced by the automaton and may only appear in the initial array of the system. Given

*jseck@ciaii.edu.mx, juanseck@yahoo.com.mx
†genaro@enigma.red.cinvestav.mx, genarojm@correo.unam.mx
‡mcintosh@servidor.unam.mx, http://delta.cs.cinvestav.mx/~mcintosh

a sequence of states in a cellular automaton, one question is if it has an ancestor sequence. We can extend this problem in time, that is, looking backwards for ancestors in several steps [4, 13].

This paper presents a procedure for calculating ancestors of a sequence of states in several steps, this procedure is based on graph tools such as de Bruijn and subset diagrams and their matrix representations. From these diagrams, we obtain data structures known as subset tables, which are useful to yield ancestors which are not Garden-of-Eden sequences.

We improve this procedure using de Bruijn diagrams of second and third order. This procedure is illustrated calculating ancestors in several steps in the binary one-dimensional cellular automaton Rule 110. We use this rule because it is the most simple cellular automaton able to realize universal computation [2, 12, 8], therefore we desire to find ancestors of several interesting structures produced by this automaton such as large triangles.

The paper has the following organization, Section 2 describes the basic concepts and the graph tools used for analyzing one-dimensional cellular automata. Using these graphs, Section 3 presents data structures known as subset tables which will be useful to define a procedure for calculating non Garden-of-Eden ancestors. Section 4 exposes how de Bruijn and subset diagrams of higher order improve the calculation of ancestors for larger number of steps. Section 5 shows examples of this process calculating ancestors of large triangles in Rule 110 and the final section provides the concluding remarks of the paper.

2 Basic concepts

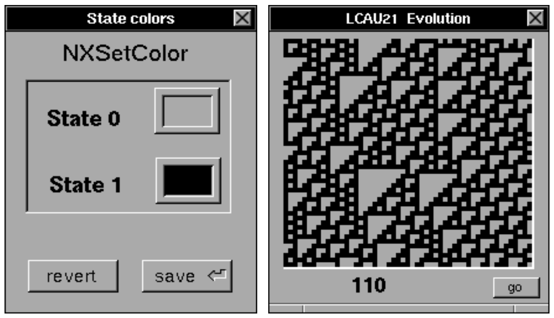
A one-dimensional cellular automaton consists of a one-dimensional array of cells and a finite set of states K; the cardinality of K is presented by $k \in \mathbb{Z}^+$. Initially each cell takes a value from K, forming the initial configuration of the automaton. We shall only study finite configurations, in this case we put together the initial cell and the last one of the initial configuration forming a ring. For $n \in \mathbb{Z}^+$ let K^n be the set of sequences with n states and let K^* be the whole set of finite sequences.

For $m \in \mathbb{Z}^+$ there is a mapping $\varphi : K^m \rightarrow K$ where every $w \in K^m$ is a neighborhood of the automaton, m is the neighborhood size and φ is the evolution rule; this mapping is applied over each neighborhood in the initial configuration, where each neighborhood shares $m-1$ cells with the contiguous ones at both sides. In this way, the initial configuration yields a new configuration with the same number of cells. This process is indefinitely repeated, inducing a mapping between configurations of the automaton, hence the global behavior or evolution of the automaton depends on the local features of the evolution rule.

For a cellular automaton of neighborhood size m, each sequence w in K^m *evolves* or has a mapping into a state a in K, thus we say that w is an ancestor of a. This is extensive for larger sequences, for $n \in \mathbb{Z}^+$, every sequence $w \in K^{n+m}$ evolves into a sequence $v \in K^{n+1}$ applying φ over each of the $n+1$ neighborhoods forming w; and w is an ancestor of v.

Given an evolution rule φ, it may exists a sequence $w \in K^*$ such that w has not ancestors and it can only appear as part of the initial configuration, then we say that w is a Garden-of-Eden sequence.

An example of the evolution of a one-dimensional cellular automaton of 2 states and neighborhood size 3 is presented in Figure 1. This figure shows the evolution rule in a table, where one column represents the neighborhoods of the automaton an the other the evolution of every neighborhood. This automaton is known as Rule 110 due to the binary number formed by the evolution of the neighborhoods.



Neighborhood	Evolution
000	0
001	1
010	1
011	1
100	0
101	1
110	1
111	0

Figure 1: Evolution of the automaton Rule 110.

The evolution rule of a cellular automaton can be represented using a directed graph called *de Bruijn diagram* [3] [7] [5]. In this diagram, the nodes represent partial neighborhoods, the directed edges between nodes represent complete neighborhoods and each directed edge is labeled by the evolution of the corresponding neighborhood. For a cellular automaton of k states and neighborhood size m, a formal description of the de Bruijn diagram is the following one:

1. The nodes of the diagram are the elements of the set K^{m-1}.
2. For $a, b \in K$ and $w \in K^{m-2}$, there exists a directed edge from node aw to node wb representing the complete neighborhood $awb \in K^m$.
3. Every directed edge representing awb is labeled by $c \in K$ if and only if $\varphi(awb) = c$.

The de Bruijn diagram of the automaton Rule 110 is presented in Figure 2, light edges represent neighborhoods evolving into 0 and dark edges are neighborhoods evolving into 1.

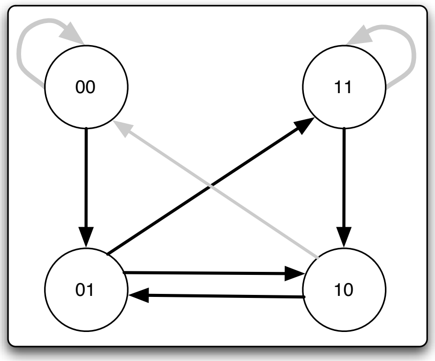


Figure 2: De Bruijn diagram of the automaton Rule 110.

De Bruijn diagrams are relevant because they provide a graph presentation for the evolution rule of a one-dimensional cellular automaton; in a de Bruijn diagram a given path represents a sequence of states produced by the evolution of the automaton and the nodes in the path describe the ancestor of the sequence. Thus the analysis of ancestors may be realized studying the paths of the de Bruijn diagram.

A given sequence of states may have several, one or no ancestor, hence this sequence has analogously several, one or no path representing it in the de Bruijn diagram. Therefore if we want to know if a given sequence is a Garden-of-Eden one, we have to review if there is no path labeled with this sequence in the corresponding de Bruijn diagram.

In order to define a deterministic procedure for knowing if some sequence is a Garden-of-Eden one, we shall use the de Bruijn diagram to generate the subset diagram [7, 11]. The subset diagram is another useful directed graph which provides a functional representation of the corresponding de Bruijn diagram, this feature will be relevant for detecting Garden-of-Eden sequences. The formal construction of a subset diagram is the following one:

1. The nodes of the subset diagram are all the possible subsets of de Bruijn nodes, starting from the empty set up to the subset containing all the de Bruijn nodes.
2. For $a, b, c \in K$, $w \in K^{m-2}$ and two subsets A, B in the subset diagram, there is a directed edge from A to B labeled by c if for each de Bruijn node $wb \in B$ there exists another de Bruijn node $aw \in A$ such that $\varphi(awb) = c$.

The subset diagram of the automaton Rule 110 is presented in Figure 3, light arrows represent edges labeled by state 0 and dark arrows are edges labeled by state 1.

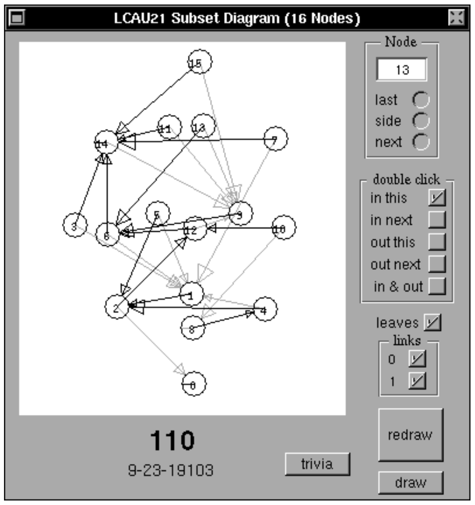


Figure 3: Subset diagram for the automaton Rule 110. Subsets are named as: $0 = \emptyset$, $1 = 00$, $2 = 01$, $4 = 10$, and $8 = 11$; these names are added for unions forming larger subsets.

If the de Bruijn diagram has k^{m-1} nodes, then the corresponding subset diagram has $2^{k^{m-1}}$ nodes. As we said before the subset diagram provides a functional presentation of the de Bruijn diagram, hence in order to detect a Garden-of-Eden sequence we just have to review if it defines a path from the full subset to the empty one in the subset diagram. For instance, if $w \in K^*$ defines such a path in the subset diagram, it means that in the de Bruijn diagram there are no paths labeled by w, implying that this sequence does not have ancestors.

The subset diagram is useful for reviewing if some sequence has no ancestors in one generation; if we want to calculate ancestors for more generations we can calculate all the ancestors of the sequence and select those which are not Garden-of-Eden sequences according with the subset diagram, repeating the process over these ancestors. In order to improve this process, we shall only take the part of the subset diagram useful for detecting Garden-of-Eden sequences. In the next section we shall use this part of the subset diagram to define subset tables which will be used for implementing a process to calculate non Garden-of Eden ancestors in several generations.

3 Procedure for calculating ancestors in several generations

Our process for calculating Garden-of-Eden ancestors needs the part of the subset diagram which is able to detect paths from the full subset to the empty one. The next procedure illustrates how to use the de Bruijn diagram associated with the automaton for obtaining this part of the subset diagram:

Procedure 1 (Part of the subset diagram which detects Garden-of-Eden sequences)

1. *Take all the nodes in the de Bruijn diagram and form the full subset, save this subset in a list L.*
2. *For $a, b, c \in K$, $w \in K^{m-2}$ and every new subset A in L, take the subset B of de Bruijn nodes such that for each node $wb \in B$ there exists $aw \in A$ such that $\varphi(awb) = c$; if B is not in L add B to it.*
3. *If there are no new subsets in L then stop the process, otherwise go to step* 2.

Procedure 1 calculates the component of the subset diagram beginning from the full subset; one important observation is that if the procedure does not generate the empty set then the automaton does not have Garden of Eden and therefore it is surjective. Procedure 1 forms the list L presented in Table 1 and the component presented in Figure 4 for the automaton Rule 110; this component has eight nodes and one of them is the empty set.

subsets	state 0	state 1
15	9	14
14	9	14
9	9	6
6	1	14
1	1	2
2	0	12
12	9	6
0	0	0

Table 1: List L of subsets in Rule 110 obtained by Procedure 1.

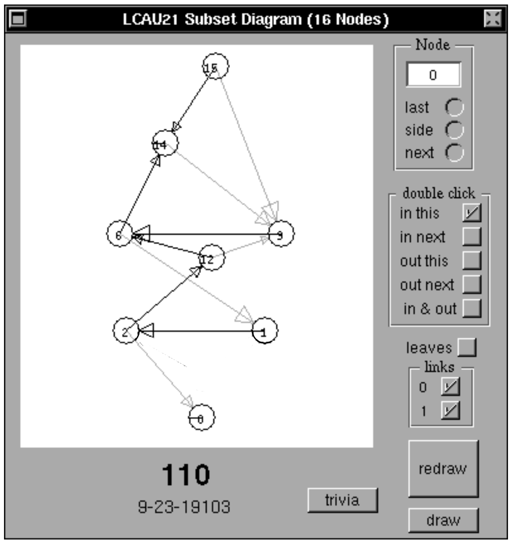


Figure 4: Component beginning from the full subset for the automaton Rule 110, light arrows represent edges labeled with state 0 and dark arrows are edges labeled with state 1.

Based on the subsets generated by Procedure 1, we shall specify *subset tables* which are special data structures representing the relation of each subset with the others. The table corresponding with a given subset B has the following structure:

- The rows of the table are the de Bruijn nodes forming B.
- Every column of the table represents a pair of elements, the first one is a subset A which reaches B by a particular edge and the second is the label $c \in K$ of this edge.
- For a row $wb \in K^{m-1}$ and a column (A, c) in the table, the entry $(wb, (A, c))$ represents a set of pairs where the first element of each pair is a de Bruijn node $aw \in K^{m-1}$ in A such that $\varphi(awb) = c$ and the second element is the state a.

The subset table divides the subset B into de Bruijn nodes and each row indicates how a de Bruijn node of B is related with another subset A by means of reversing the direction of the edge labeled by $c \in K$. The general form of a subset table is presented in Table 2.

	source subset, labeled edge
ending subset, de Bruijn node	de Bruijn node in the source subset/left state of the neighborhood

Table 2: Structure of a subset table.

Remember that in the subset diagram every directed edge labeled by $c \in K$ means that the subsets defining the edge specify neighborhoods which evolve in c as well, thus the idea of the next procedure is obtaining ancestors by means of reversing the direction of the edges in the subset diagram.

Procedure 2 (Calculating ancestors for a given sequence $w \in K^*$)

1. *Starting from the full set, trace out a path corresponding with the sequence w.*
2. *If the final subset of this path is the empty set then the sequence belongs to the Garden of Eden, hence stop the process. Otherwise take each de Bruijn node in the final subset, every node specifies the rightmost $m-1$ cells of every possible ancestors. In this way if there are several de Bruijn nodes in the final subset, there are several possible ancestors as well.*
3. *Read now the directed edges running backwards, and take from each subset the de Bruijn nodes contained within this trajectory.*
4. *Write down the leftmost state in each the Bruijn node passed, thereby developing the ancestor from right to left, and stopping at the full set.*

Procedure 2 gets a possible ancestor of w taking the leftmost state of the de Bruijn node while passing from the destination subset to the source one. Several de Bruijn nodes within the same subset in the directed edge represent alternate choices and a recursive process should be specified to get all the possible ancestors.

In order to implement Procedure 2 we shall use the subset tables defined in Table 2 for running backwards the paths in the subset diagram. For instance, the subset tables for the automaton Rule 110 are in Table 3.

Subset table 14

	15,1	14,1	6,1
14,01	00/0 , 10/1	10/1	10/1
14,10	01/0 , 11/1	01/0 , 11/1	01/0
14,11	01/0	01/0	01/0

Subset table 9

	15,0	14,0	9,0	12,0
9,00	00/0 , 10/1	10/1	00/0	10/1
9,11	11/1	11/1	11/1	11/1

Subset table 6

	9,1	12,1
6,01	00/0	10/1
6,10	11/1	11/1

Subset table 12

	2,1
12,10	01/0
12,11	01/0

Subset table 2

	1,1
2,01	00/0

Subset table 1

	6,0	1,0
1,00	10/1	00‘/0

Table 3: Subset tables of the automaton Rule 110.

These tables shall be scanned repeatedly to get the ancestors of a given sequence; the next procedure explains how to calculate the ancestors of a sequence $w \in K^*$:

Procedure 3 (Calculating ancestors of $w \in K^*$ by means of subset tables)

1. *Starting from the full set, trace out a path corresponding with the sequence w.*
2. *If w defines a path from the full subset to the empty set, then it is a Garden-of-Eden sequence and stop the process, otherwise take the sequence of subsets in the path labeled by w and the final subset A.*
3. *Take the subset table of A, for every row $(A, va \in K^{m-1})$ in this table, the de Bruijn node va specifies the $m-1$ rightmost states of a possible ancestor of w.*
4. *Take the entry $(bv \in K^{m-1}, b \in K)$ corresponding with row $(A, va \in K^{m-1})$ and column $(B, d \in K)$ associated with the previous subset B in the path, add b to the left of the ancestor that we are calculating and go to row $(B, bv \in K^{m-1})$ of the subset table B.*
5. *If B is the full subset then stop the process, otherwise go to step 4 now using row $(B, bv \in K^{m-1})$ in the subset table B.*

We present now an example of Procedure 3 using the subset tables of the automaton Rule 110 described in Table 3 and the sequence 1010:

1. The sequence 1010 yields the path 15, 14, 9, 6, 1 in the subset diagram.
2. Since the final subset of the path is not the empty set, we take the final subset table 1.
3. Let us take row $1, 00$ in the subset table 1, thus the rightmost states of the ancestors of 1010 are 00.

4. Let us take column $6, 0$ associated with subset table 6 and the last state of the path 1010, add 1 to the left of the ancestor and we pass to row $6, 10$ in the subset table 6.
5. In row $6, 10$ let us take now column $9, 1$, add 1 to the left of the ancestor and we take now row $9, 11$ in the subset table 9.
6. In row $9, 11$ let us take column $14, 0$, add 1 to the left of the ancestor and we pass to row $14, 11$ in the subset table 14.
7. In row $14, 11$ let us take column $15, 1$, and add 0 to the left of the ancestor. The procedure stops since the full subset has been reached and the ancestor of 1010 is 011100.

Procedure 3 can be used for calculating all the ancestors of a given sequence, then we can take the subset diagram to analyze them and conserve only those ancestors which are not Garden-of Eden sequences. In this way we present the next procedure for calculating the ancestors of $w \in K^*$ in n generations backwards.

Procedure 4 (Calculating the ancestors of w in n generations)

1. *Apply procedure 3 for obtaining the ancestors of w.*
2. *Using the subset diagram, keep those ancestors which are not Garden-of-Eden sequences.*
3. *For each ancestor conserved in the previous step, apply step 1.*
4. *The procedure stops up to we get n iterations.*

Procedure 4 was coded in C language using the data structure depicted in Figure 5, this structure represents subset tables and the connections among them.

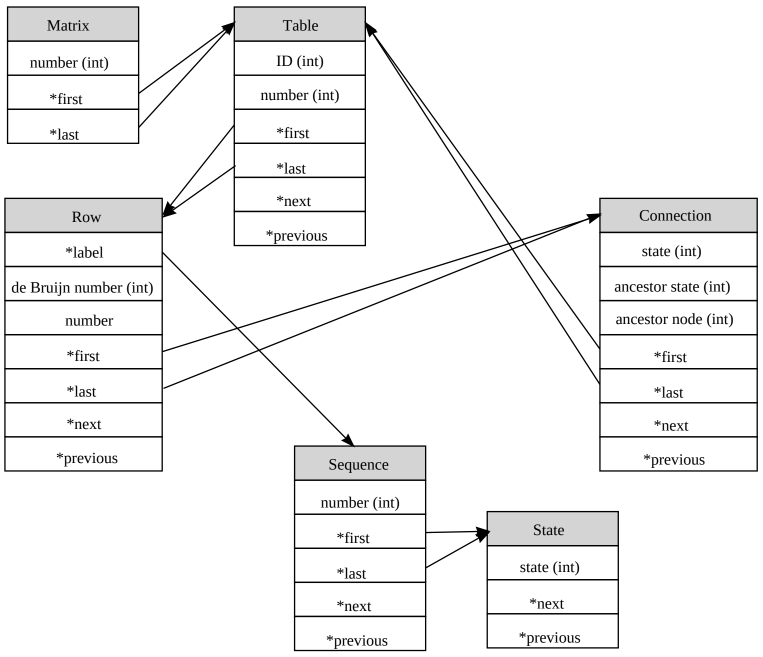


Figure 5: Data structure representing the relation of the subset tables.

The final part of this section presents the C code implementing both Procedure 3 and Procedure 4, the first procedure uses the data structure presented in Figure 5 in order to calculate the ancestors of a given sequence by means of subset tables.

```
/* Recursive implementation of Procedure 3 which takes a sequence of states b2 and its
corresponding path b1 in the subset diagram  in order to form the ancestors of b2 by
means of the subset tables represented by the global matrix structure subsetMatrix*/
void expandSubsetSequence(sequenceList a, sequence b1, sequence b2, subsetMatrix c,
                          int pos, int no)
{
 int i,aux2;
 sequence aux;
 state d1;
 state d2;
 table e;
 row f;
 state g;
 conecction h;
 d1=b1->last; /* Take the elements both from the sequence and the path*/
 d2=b2->last;
 for(i=0;i<pos;i++)
 {
  d1=d1->previous;
  d2=d2->previous;
 }
 e=c->first;  /* Select the current subset table*/
 for(i=0;i<c->number;i++)
 {
  if(compareSequences(e->ID,d1)==1)
   break;
  e=e->next;
 }
 aux2=1;
 f=e->first; /* Select the adequate row in the associated subset table */
 for(i=0;i<e->number;i++)
 {
  if(pos==0) /* Copy the de Bruijn node of the final subset into the ancestor */
  {
   final->number=0; /* final is a global sequence which keeps the ancestor */
   g=f->label->last;
   for(j=0;j<f->label->last;j++)
   {
    aux2=checkGarden(final,g->est); /* Check if final is not a Garden-of-Eden sequence */
    if(aux2==0)
     break;
    insertState(final,g->est);
    g=g->previous;
   }
  }
  if(((pos==0)||(f->deBruijnNumber==no))&&(aux2==1)) /*Select the next subset table*/
  {
   h=f->first;
   for(j=0;j<f->number;j++)
   {
    aux->number=0;
    if(h->last->number==0)
    {
     copySequence(subset->first->subsetNumber,aux); /* We reach the full subset */
    }
    else
    {
     copySequence(h->last->ID,aux); /* We take the subset of the following subset table*/
    }
    if(compareSequences(aux,d1->previous)==1)
    { /* Compare if we have selected the correct subset with the correct labeled edge*/
     if(h->edo==d2->est)
     {  /* Check if final is not forming a Garden-of-Eden sequence */
      aux2=checkGarden(final,h->ancestorState);
      if(aux2==1) /* Add another state to the leftmost part of "final" */
      {
       insertState(final,h->ancestorState);
       if(pos<(b2->number)-1)
       { /* Recursive process */
        expandSubsetSequence(a,b1,b2,c,pos+1,h->ancestorNode);
       }
       else
       {  /* We have completed the ancestor, numcal is a global variable counting
             the number of generated ancestors*/
         numcal++;
         insertSequence(a,final);
       }
      }
     }
    }
    if(j<(f->number)-1)
     h=h->next;
   }
  }
  if(i<(e->number)-1)
   f=f->next;
 }
}

/* Implementation of Procedure 4 for calculating ancestors */
int calculatingAncestors(sequence chain)
{
 int i,j;
 int aux;
 sequence a, subsetSequence;
 for(i=0;i<numgen;i++)
 { /*Cycle which calculates ancestors in "numgen" generations */
  {
   numcal=0;
   if(i==0)
   { /*Calculates the path "chain" in the subset diagram */
    subsetSequence=subsetAncestors(chain);
    if(subsetSequence->number>0)
    {
     /* "chain" is not a Garden-of-Eden sequence, then we calculate its ancestors using
         subset tables, these ancestors are conserved in "ancestorList" , the global
         variables used in the procedure are: ancestorList.- list of sequences,
         subsetmatrix.- list of subset tables */
     expandSubsetSequence(ancestorList,subsetSequence,chain,subsetMatrix,0,0);
    }
    else
    { /* "secuencia" is a Garden-of-Eden sequence */
     printf("\nThe sequence belongs to the Garden of  Eden\n\n");
     return 0;
    }
   }
   else
   { /* "aux"  keeps the number of ancestors calculated by expandSubsetSequence() */
    aux=ancestorList->number;
    a=ancestorList->first; /* "a" points to the first ancestor in the list */
    for(j=0;j<aux;j++)
    { /* Cycle which takes all the ancestors in the list */
     subsetSequence=subsetAncestors(a);
     if(subsetSequence->number>0)
     {
      /* "a" is not a Garden-of-Eden sequence, we calculate its ancestors using subset
          tables, these ancestors are conserved in "ancestorList2" which is a global
          variable keeping a list of sequences */
      expandSubsetSequence(ancestorList2,subsetSequence,a,subsetMatrix,0,0);
     }
     a=a->next;
    }
   }
   if(ancestorList2->number==0) /*Check if "ancestorList2" has elements*/

    return 0;
  }   /* "ancestorList2" is not empty, then it takes the place of "listaAncestros" */
  changeList(ancestorList,ancestorList2);
 }
 return 1;
}
```

4 Improving the calculation of ancestors

Procedure 4 iteratively calculates non Garden-of-Eden ancestors for a given sequence of states in n generations. In order to improve this procedure, we will take de Bruijn diagrams representing the evolution of the automaton in two and three generations; these diagrams are respectively called de Bruijn diagrams of second and third order. Based on these diagrams , we shall take their corresponding subset diagrams and applying Procedure 4, we will yield ancestors for 2 and 3 steps backwards in each iteration. For instance, the matrix presentation of the de Bruijn diagram of second order for the automaton Rule 110 is in Table 4. In this table rows are sequences $aw \in K^{m+1}$, columns are sequences $wb \in K^{m+1}$ and each entry represents the evolution of the sequence $awb \in K^{m+2}$ in two generations.

Nodes	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	0	1														
0001			1	1												
0010					1	0										
0011							0	1								
0100									0	1						
0101											0	0				
0110													1	0		
0111															1	0
1000	0	1														
1001			1	1												
1010					1	0										
1011							0	1								
1100									0	1						
1101											0	0				
1110													1	1		
1111															1	0

Table 4: Matrix presentation for the de Bruijn diagram of second order in Rule 110.

The de Bruijn diagram of third order for the automaton Rule 110 has 64 nodes, therefore it is impractical to show its matrix representation. We take de Bruijn diagrams of third order in order to improve the calculations of Procedure 4, the component of the subset diagram calculated with Procedure 1 for the automaton Rule 110 has 361 nodes, therefore there are 360 subset tables useful for calculating ancestors in three generations for each iteration of the procedure. In the next section we show some results concerning to calculating ancestors in Rule 110, in particular we are interested in searching ancestors for the large triangles produced by this cellular automaton.

5 Examples

The one-dimensional cellular automaton Rule 110 is right now a relevant topic of study, as we can see in several works by Cook [2], Wolfram [12], Wuensche [13] and McIntosh [8]. The reason is that Rule 110 has very interesting properties in spite of its simplicity (only two states and neighborhood size 3), for instance, it is possible to simulate a cyclic tag system in this automaton for implementing an universal computing device [8, 6, 12]. Rule 110 is characterized by forming triangles through its evolution, that is, the evolution rule has the property of covering the space with triangles of different sizes formed by the cells of the configurations. An evolution sample from a random configuration in Rule 110 is depicted in Figure 6.

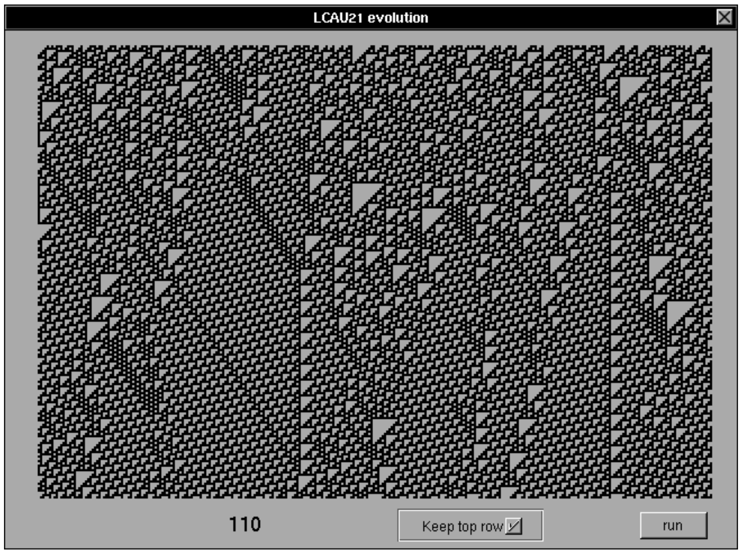
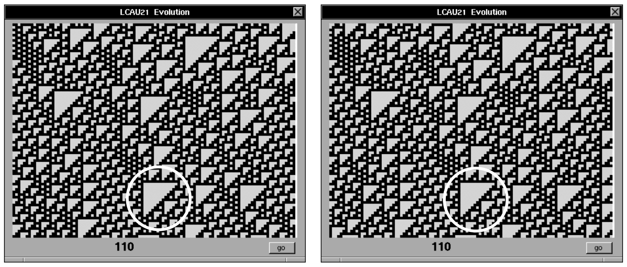


Figure 6: Common evolution of the automaton Rule 110.

Given the property of the automaton in constructing triangles during its evolution, one interesting question is if it is able to produce large triangles after several generations. In order to obtain some examples for analyzing this feature, we use Procedure 4 for calculating ancestors of triangles with different sizes (from 13 up to 18 cells in each side of the triangle). For this particular case, we use the following parameters:

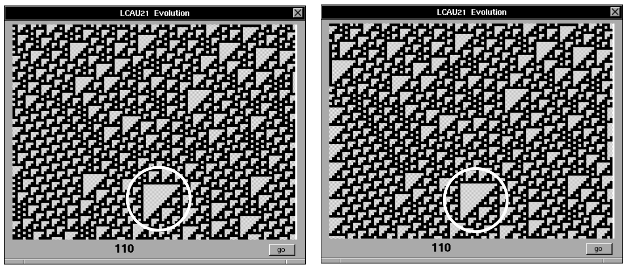
- Return up to 60 generations.
- Keep up to 600 ancestors in each generation.
- Use a de Bruijn diagram of third order.

Some ancestors calculated with Procedure 4 for these triangles are presented in the following figures, in each case the ancestors specify the initial configuration and the desired triangle yielded by the evolution of the automaton is in a white circle. Notation T_n means that the triangle is formed by n cells with state 0 on each side.



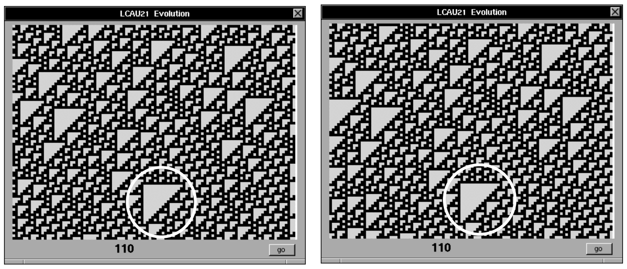
T13

Figure 7: Two ancestors for a triangle T_{13}.



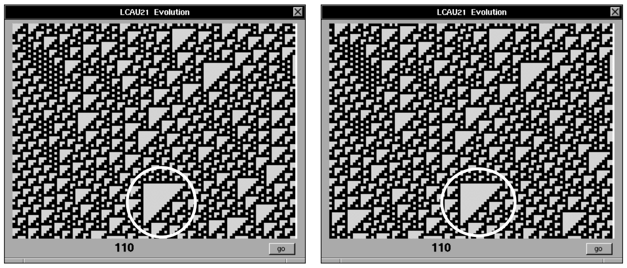
T14

Figure 8: Two ancestors for a triangle T_{14}.



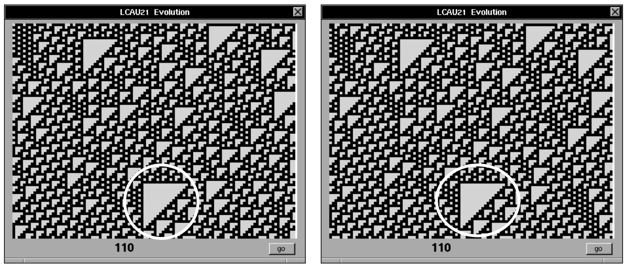
T15

Figure 9: Two ancestors for a triangle T_{15}.



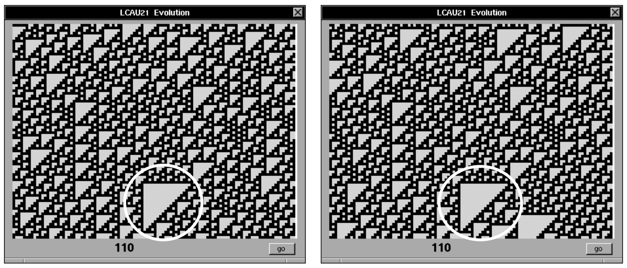
T16

Figure 10: Two ancestors for a triangle T_{16}.



T17

Figure 11: Two ancestors for a triangle T_{17}.



T18

Figure 12: Two ancestors for a triangle T_{18}.

6 Concluding remarks

Calculating ancestors in several generations for a given sequence of states is a classical problem in cellular automata theory, however there are few algorithms available to realize this process. We have presented a procedure based on graph tools for calculating non Garden-of-Eden ancestors. The procedure presented in this paper has a simple computing implementation because it is based on simple graphs as de Bruijn and subset diagrams.

For the examples presented in the previous section, the procedure yields a few dozens of generations backwards; in general for cellular automata with a small number of states or a small neighborhood size, this process is able to yield ancestors in dozens of generations. Of course, the procedure is limited by the computer memory used in calculating and saving graphs and subset tables. In any case, the expected results of this procedure depend on the properties of each particular cellular automaton.

One problem with the procedure is that we could have a huge number of possible ancestors for a given sequence, and an exhaustive search is not practical. In this case we have implemented a random selection of the ancestors in order to continue with the process.

If we want to obtain more generations backwards, we must define another kind of methods instead of an exhaustive search, for instance a procedure based on some analytic representation for the evolution of a cellular automaton could be able to return hundreds of generations.

References

[1] E. F. Codd. *Cellular Automata.* Academic Press, New York, 1968.

[2] Matthew Cook. Universality in elementary cellular automata. personal communication.

[3] Solomon Wolf Golomb. *Shift Register Sequences.* University of Southern California, Holden-Day Inc., 1967.

[4] Howard Gutowitz and Christophe Domain. The topological skeleton of cellular automaton dynamics. *Physica D*, 103:155–168, 1997.

[5] Erica Jen. Enumeration of preimages in cellular automata. *Complex Systems*, 3(5):421–456, 1989.

[6] Genaro Juárez Martínez. Un camino para construir configuraciones complejas en la regla 110. available in http:// delta. cs. cinvestav. mx/ ˜mcintosh, 2002.

[7] Harold V. McIntosh. Linear cellular automata via de Bruijn diagrams. available in http:// delta. cs. cinvestav. mx/ ˜mcintosh, 1991.

[8] Harold V. McIntosh. A concordance for rule 110. available in http:// delta. cs. cinvestav. mx/ ˜mcintosh, 2002.

[9] Edward F. Moore. Machine models of self-reproduction. In Arthur W. Burks, editor, *Essays on Cellular Automata*, pages 204–205. University of Illinois Press, 1970.

[10] John von Neumann. *Theory of Self-Reproducing Automata.* University of Illinois Press, Urbana and London, 1966. edited by Arthur W. Burks.

[11] Burton H. Voorhees. *Computational analysis of one-dimensional cellular automata.* World Scientific, Sigapore, New Jersey, London, Hong Kong, 1996.

[12] S. Wolfram. *A new kind of science.* Wolfram Media, Inc., Champaign Illinois, USA, 2002.

[13] Andrew Wuensche. Finding gliders in cellular automata. In A.Adamatzky, editor, *Collision-Based Computing.* Springer, 2002.