Complexity and Universality of Iterated Finite Automata

Jiang Zhang

Complex Systems Research Center
Academy of Mathematics and Systems Science
Chinese Academy of Sciences
No. 55 Zhong Guan Cun Dong Lu
Beijing, 100080, China
Zhangjiang@amss.ac.cn

The iterated finite automaton (IFA) was invented by Stephen Wolfram for studying the conventional finite state automaton (FSA) by means of *A New Kind of Science* methodology. An IFA is a composition of an FSA and a tape with limited cells. The complexity of behaviors generated by various FSAs operating on different tapes can be visualized by two-dimensional patterns. Through enumerating all possible two-state and three-color IFAs, this paper shows that there are a variety of complex behaviors in these simple computational systems. These patterns can be divided into eight classes such as regular patterns, noisy structures, complex behaviors, and so forth. Also they show the similarity between IFAs and elementary cellular automata. Furthermore, any cellular automaton can be emulated by an IFA and vice versa. That means IFAs support universal computation.

1. Introduction

The finite state automaton (FSA) or finite state machine is a very important model that has been widely used in computer science and industry [1, 2]. The automaton can perform very complex computational tasks with only finite internal states and fixed transition rules. Usually, there are two kinds of FSAs. Finite state acceptors (recognizers) only accept information and jump between different states but do not generate any output information. These machines are widely used as language recognizers [3]. Another class is called finite state transducers, which are able to generate output information as well as accept input information. They can be designed as controllers [2].

Complexity science arose sometime in the 1980s [4, 5] and stresses a different philosophy and methodology in the approach to complex system problems [4]. Stephen Wolfram's *A New Kind of Science* (NKS) is a representative of complex systems studies [6]. NKS mainly focuses on the complexity generated by different, very simple computational systems. Therefore, building the simplest systems, implement-

ing their computations, observing their behaviors, and drawing conclusions are main steps in the NKS approach. All kinds of computational systems, including Turing machines, substitution systems, and so on, were studied by this method in [6].

What kind of complex behaviors can the FSA perform? Can we use the methodology of NKS to study this specific system? Wolfram invented the iterated finite automaton (IFA) [7] to answer these questions. By adding a tape with finite size and some other constraints to the FSA, we can study the behavior just like one-dimensional cellular automata (CAs). Wolfram has enumerated all possible patterns of two-state two-color and three-state two-color IFAs. This paper mainly studies the complex behaviors of two-state three-color IFAs. We divide the patterns into eight groups roughly including regular patterns, noisy structures, complex behaviors, and so forth. Furthermore, two-state three-color IFAs show the similarity between IFAs and one-dimensional CAs. This similarity encouraged the author to study emulation relationships between IFAs and one-dimensional CAs. The result makes us conclude that IFAs as a whole family support universal computation.

Section 2 introduces the working mechanism of IFA systems. Section 3 investigates the complexity and classification of IFAs according to their generated patterns. Section 4 discusses the emulation approach of an IFA to a CA and a CA to an IFA. Then, the conclusions are drawn in Section 5.

2. Iterated Finite Automata

To illustrate what IFAs are and how they work, we first give some formal definitions.

Definition 1. A finite state transducer is a tuple, $\langle I, S, O, f, s_0 \rangle$, where I is a finite set of input symbols, S is a finite set of states, O is a finite set of output symbols, and $f : S \times I \to S \times O$ is a function. f can be represented by a set of transition rules. Each rule has the form $r : (s, i) \to (s', o)$, where $s, s' \in S$, $i \in I$, and $o \in O$. The initial state is $s_0 \in S$ [1].

Another representation of a finite state transducer is a graph in which vertices are states in S and directed edges are transitions between states, that is, are rules in f. There are two symbols on each edge denoting input and output information.

Example 1. Consider a specific finite state transducer, $\Gamma = \langle I, S, O, f, s_0 \rangle$, where $I = O = \{0, 1\}$, $S = \{1, 2, 3\}$, and $s_0 = 1$. The function f is a set of transition rules. This finite state transducer can also be represented by a graph as shown in Figure 1.

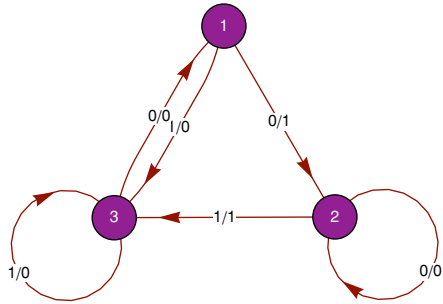

Figure 1. A finite state transducer.

In Figure 1, the edge from 1 to 2 represents the transition rule: if the machine is in state 1 and accepts an input 0, then its state will turn into 2 and generate an output 1.

Definition 2. An IFA is a pair: $\Omega = \langle \Gamma, \alpha \rangle$. Γ is a finite state transducer with the same input and output set, that is, $\Gamma = \langle I, S, I, f, s_0 \rangle$. Usually, we set $I = \{0, 1, \ldots, c-1\}$ (the number of input and output symbols [colors] is c), and the finite set $S = \{1, 2, \ldots, s\}$ (the number of states is s). This specific finite state transducer is also called a *machine* or a *machine head*. $\alpha = I^n$ is a tape with n cells, each member in α is $\langle a_1, a_2, \ldots, a_n \rangle$ indicating cells with different colors; that is, the i^{th} cell's color is $a_i \in I$. So any member of α is a configuration of the tape.

Now we will discuss how a finite state transducer Γ operates on the tape α. Assume that the initial configuration of α is $\langle a_1^0, a_2^0, \ldots, a_n^0 \rangle$. The machine Γ with the initial state s_0 starts to read information from the tape. The color of the first cell on the tape is a_1^0. According to the input a_1^0 and state s_0, the machine will look up the rules table and give out a pair of symbols representing the output and the next state $\langle s', y \rangle$, respectively. So, the color of the first cell will be updated as $a_1^1 = y$, and the machine Γ will move to the second cell.

Definition 3. The process of the finite state transducer Γ operating on a cell of the tape including reading the input information from the cell, updating the cell's color, and moving to the next cell is called a *step* of the IFA.

The machine Γ will repeat this process step by step until it reaches the last cell of the tape. Then the IFA has finished a turn.

Definition 4. For IFA $\Omega = \langle \Gamma, \alpha \rangle$, n (the number of cells in α) updating steps of Γ on the tape α from the first cell to the last cell one by one is called a *turn* of this IFA.

After one turn, the configuration of the tape becomes $\langle a_1^1, a_2^1, \ldots, a_n^1 \rangle$. Then the finite state transducer will return to its initial state s_0, move to the first cell again, and repeat the whole process to start the second turn. At last, we obtain a sequence of configurations of the tape with different turns; this sequence can build up a two-dimensional pattern of the IFA.

Definition 5. A *pattern* of an IFA is a sequence of tape configurations: $\langle C_1, C_2, \ldots, C_T \rangle$, where $C_i \in I^n$.

For example, consider the finite state transducer in Example 1 is working on a tape with five cells, then three turns are as shown in Figure 2.

In Figure 2, a two-dimensional pattern can be obtained once we integrate the pictures in the last step row by row together to show the behavior of this IFA.

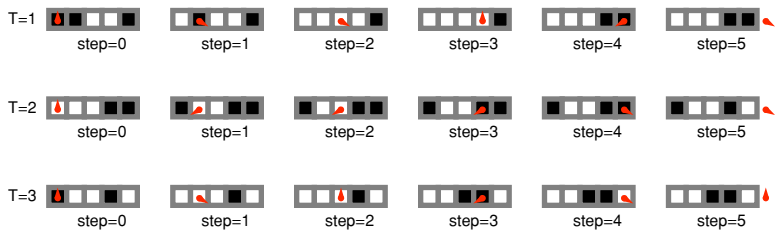

Figure 2. Three turns of the example IFA. The red arrow represents the machine head. Different head directions stand for different states.

Obviously, the pattern of an IFA is determined by the machine and the tape. In this paper, we only investigate the initial configuration with all blank cells, that is, $a_1^0 = a_2^0 = \cdots = a_n^0 = 0$. Another convention is to treat all machines with the same number of states s and the same number of input/output symbols c as a class of IFAs which is denoted as a pair (s, c). There are $(s \times c)^{s \times c}$ possible rules in the class (s, c) IFAs. Different rules in the same class determine the patterns of the IFA. Thus, we can assign a coding number for each IFA in the class (s, c). The concrete method for coding IFAs is shown in Appendix A.

3. Complexity of Iterated Finite Automata

3.1 The Complexity of (2,2) and (3,2) Iterated Finite Automata

Wolfram had studied the complexity of (2,2) and (3,2) IFAs in [7]. Most (2,2) IFAs exhibit trivial structures such as blank tapes and cyclic patterns, except for some nested structures. But for (3,2) IFAs,

more nested structures were found, and some random structures were discovered. Some selected (2,2) and (3,2) IFA patterns are shown in Figure 3.

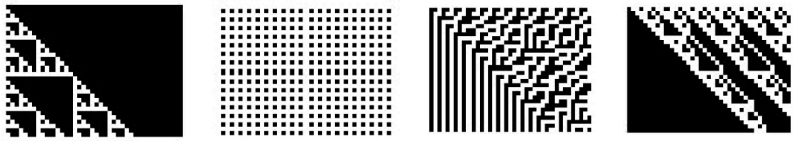

Figure 3. Typical patterns in (2,2) and (3,2) IFAs. The number below the pattern is its rule number.

3.2 The Complexity of (2,3) Iterated Finite Automata

The author investigated the complex behaviors of (2,3) IFAs by systematic searching. There is a total of $6^6 = 46\,656$ possible (2,3) IFA rules. After filtering out a large number of trivial patterns that are cyclic structures and homogenous colors, there are still 1580 IFAs. Then the author divided them into eight classes, roughly according to their behaviors (see Figure 4).

1. *Regular patterns (Reg)*. The patterns in this class exhibit some regularity such as big triangles and stripes. Although most of them are trivial, there are a few exceptional regular patterns which have some nested structures, such as 3651 and 31741 in Figure 2.
2. *Noisy structures (Noisy)*. The patterns in this class are very random and full of noise from the first appearance. But some regular and symmetric local structures can also be found.
3. *Nested triangles (Nested Tri)*. Nested triangles are very common in worlds of simple programs, thus it is not astonishing that nested triangles are very easily found in IFAs.
4. *Random triangles (Ran Tri)*. This kind of structure also contains numerous triangles, but the global patterns composed by these local triangles are not obviously nested and exhibit randomness in some sense.
5. *Nested structures (Nested S)*. There are many nested structures that are not triangles that are put in this class. Some basic structures as building blocks are contained by themselves.
6. *Thin guys (Thin G)*. Some structures have random and complex interesting behaviors only in a very thin area on the left part of the pattern.

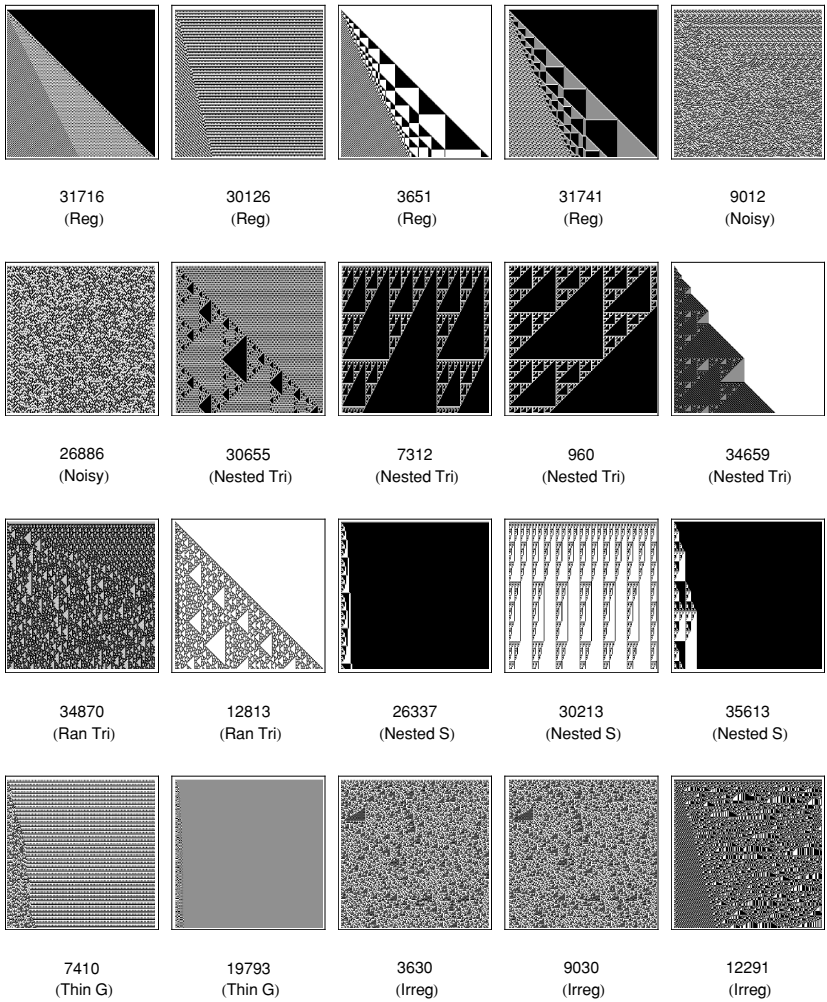

Figure 4. Complex behaviors in (2,3) IFAs with different classes.

7. *Irregular patterns (Irreg)*. A large number of IFAs are very difficult to be classified as one of the groups listed. There may be some regular local patterns, but the global structures are irregular. Whereas, compared to the noisy structures they are not so random. So we classify these IFAs as an alternative group.

8. *Complex structures*. Although it is very difficult to distinguish the complex structures that fall into this group from irregular patterns, some IFAs are selected as a new class because their behaviors are so complex that the sophisticated computation may be supported by the communication between different local areas. As examples, two of them are selected and shown in Figure 5.

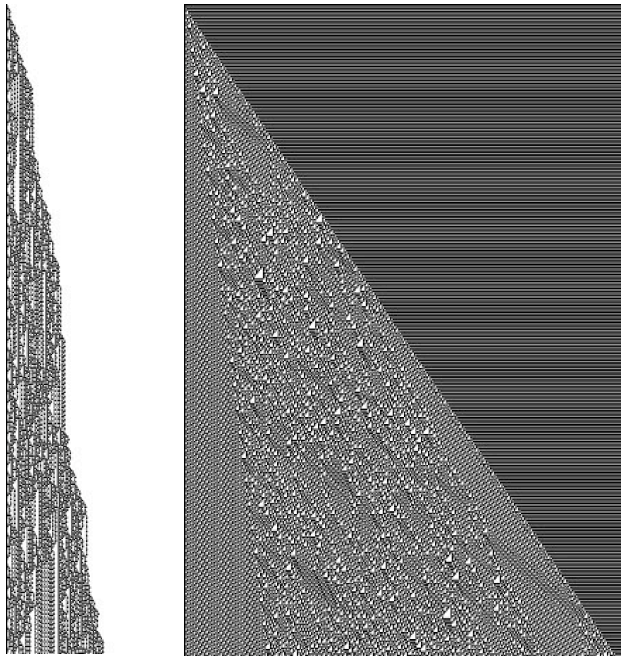

Figure 5. Complex patterns may support sophisticated computations.

The complexity of patterns in class 8 leads us to guess that IFAs may support universal computation. This hypothesis will be further confirmed by the facts mentioned in the next subsection.

3.3 Similarity between Iterated Finite Automata and Cellular Automata

From the patterns investigated earlier, it is not difficult to see that there is much similarity between IFAs and CAs.

For example, the pattern of IFA 3507 is similar to the elementary CA (one-dimensional CA with two colors and two neighbors, ECA) 30. The similarity can only be shown by flipping the pattern of ECA 30 left to right and shearing the white cells in the left of the big triangle (see Figure 6). Another example is IFA 26337. The nested structure shows similarities compared to ECA 225. The pattern of ECA 225 should also be flipped and sheared (see Figure 6).

These similarities encourage us to propose that IFAs may emulate ECAs.

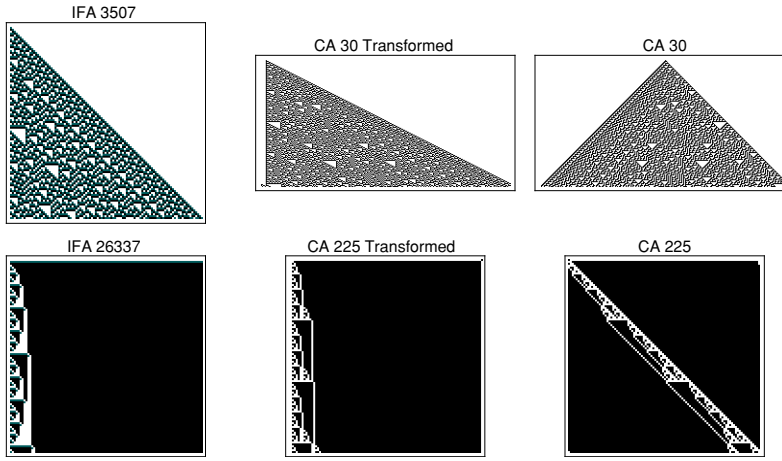

Figure 6. Similarity between IFAs and ECAs.

4. Universality of Iterated Finite Automata

4.1 Iterated Finite Automata that Emulate Elementary Cellular Automata

The complexity of IFA behaviors and the similarity between IFAs and ECAs make us propose that any behavior of an ECA can be emulated by an IFA. This subsection presents a method for constructing a concrete IFA that emulates the given ECA using a well-known procedure.

It is very natural to point out that the tape of cells in an IFA can be regarded as the cells in the one-dimensional CA. But the major difference between IFAs and CAs from their working mechanism is the former updates the cells on the tape step by step; however the latter updates all of the cells simultaneously. This difference recurs as the construction method of a specific Turing machine to emulate a given ECA in [6]. Actually, the parallel systems can be emulated by a serial system once enough states are provided. Hence, we can construct an IFA to emulate the given ECA.

For any ECA (two-color, two-neighbor), we can construct a specific four-state two-color IFA to emulate that ECA. At the beginning, the IFA tape is configured to duplicate the configurations of cells in the ECA. The finite state machine will scan the cells on the tape one by one. There are four possible combinations of two adjacent cells. Therefore, four states can memorize these combinations. It is not difficult to construct a rule table for the FSA that performs the same computation as the ECA. Then the IFA produces the same outcome on the third cell as the second cell of the ECA.

To articulate the process just mentioned, we construct a concrete IFA that emulates ECA 110 (see Figure 7).

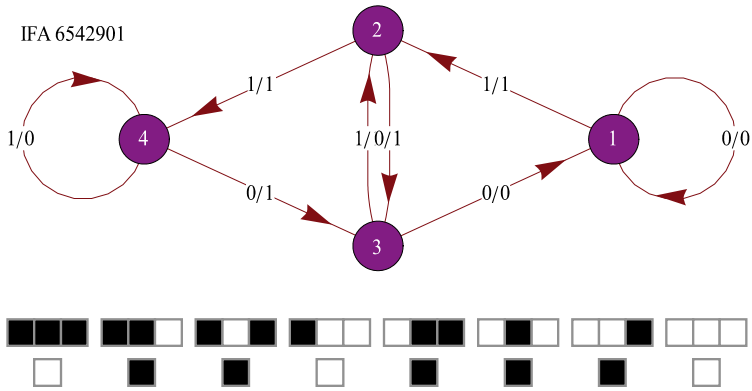

Figure 7. IFA 6542901 in (4,2) class that can emulate ECA 110. For comparison, the rule icon of ECA 110 is shown.

Suppose that the initial tape is 0100…, then ECA 110 will give the output: #10…. (The first cell is undetermined in this case.) The four-state two-color IFA 6542901 can emulate this ECA. At the beginning, the internal state of this finite state machine is 1. It will accept the input 0 from the tape and keep that state according to its rule. Then it moves to the second cell where a black cell is encountered. Then it will transit to the state 2 and give an output which should be neglected. Actually, the input information 01 has been stored by the state 2. Next, another 0 is fed on the IFA. It will transit to 3 and give an output 1. At this time, the IFA output on the third cell is exactly the same as the ECA. After that, when given the input 0, the IFA will transit to 1 again and generate an output 0. This output is the same as the output on the third cell of ECA 110. Thereafter, they can perform exactly the same as the preceding cells.

Another issue is the boundary condition. Because ECAs have cyclic boundary conditions, nevertheless, IFAs can only move on the tape in one direction. This can be solved by enlarging the size of the IFA. The number of cells in an IFA is determined by the number of steps we want to emulate times two, plus the number of cells in the ECA for cases when the number of steps is divided by the number of cells. (For more complicated cases, please refer to Appendix C.) For example, the specific IFA that was just constructed can emulate ECA 110 for 100 steps by duplicating the initial (100) cells of ECA 110 three times ($300 = 2 \times 100 + 100$) (see Figure 8).

Mathematica code for constructing IFA rules and initial conditions to emulate the ECA are listed in Appendices B and C.

Because ECAs are known to be universal, the IFA as a class, which can emulate any ECA, support universal computation.

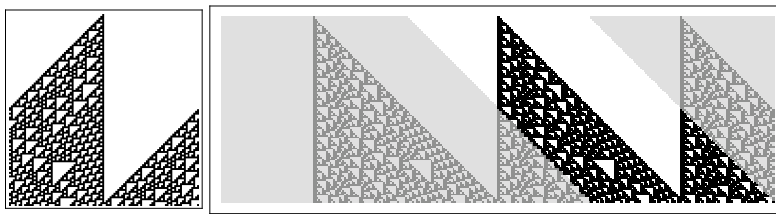

Figure 8. Comparison of patterns between ECA 110 and IFA 6542901 for 100 steps. The middle area without the gray mask shows the same behavior as ECA 110. Notice that to solve the issue of boundary conditions, the number of cells in the IFA is three times the number of cells in the ECA.

4.2 Cellular Automata that Emulate Iterated Finite Automata

CAs are known to be universal computational systems, therefore, they can emulate any other computational system including IFAs. The explicit approach for a CA that emulates an IFA is constructed here. The basic idea is to use additional colors that correspond to the multiple states of the IFA. The n (the number of cells) steps of the CA is equivalent to a turn of an IFA. As an example, (2,3) IFA 3615 emulated by a constructed CA is shown in Figure 9.

The code for transformations of rules and initial conditions are shown in Appendices D and E.

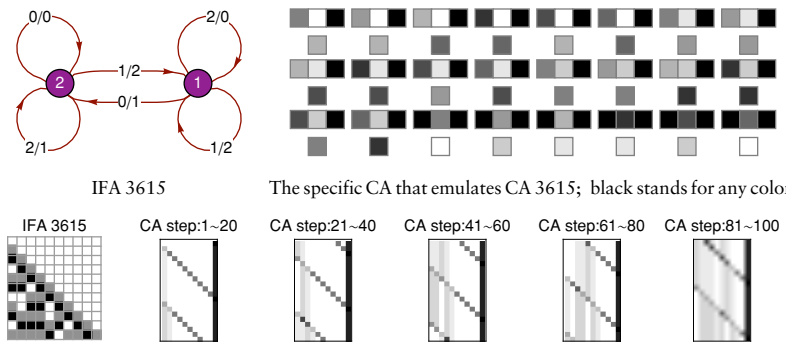

Figure 9. A CA emulating (2,3) IFA 3615. One step of the IFA is emulated by 10 steps of the IFA.

5. Conclusions

Any iterated finite automaton (IFA) is a composition of a conventional finite state transducer and a tape. This extended definition of the finite state automaton (FSA) not only facilitates our investigation by means of Wolfram's *A New Kind of Science* methodology but also shows great complexity and universality.

This paper mainly discusses the complexity of a specific class of IFA which has only two states and three possible cell colors. After enumerating all possible (2,3) IFAs, we show the spectrum of complex patterns generated by them. These patterns can be divided roughly into eight classes including regular patterns, noisy structures, nested triangles, and so forth. Among those classes, an important class (complex) is selected to show their ability of propagating information between local areas and the potential capability of supporting sophisticated computations. Furthermore, the similarity between simple IFAs and elementary cellular automata (ECAs) is pointed out.

Any ECA can be emulated by a (4,2) IFA. That means IFAs support universal computation. This conclusion not only adds a new member to the universal computational system family but also confirms the computational equivalent principle again which states that any nontrivial computational process may support universal computation.

IFAs, as another instance of a computational universe, have lots of unknown properties which should be further studied. For example, can we construct a much simpler IFA to emulate an ECA? Actually, some IFAs in the (2,3) class have shown their abilities to perform sophisticated and similar computation as ECAs. Is there a (2,3) IFA which emulates ECA 110? This plausible conclusion is worthy of further study.

Acknowledgments

Thanks to Stephen Wolfram, Todd Rowland, Eric Rowland, and Jason Cawley for discussions during the NKS 2007 summer school. This paper is based on my research there. Thanks for the support of Guozhi Xu Post Doctoral Research Foundation and National Natural Science Foundation of China (No. 60574068).

Appendix

This Appendix contains *Mathematica* code and examples for demonstrating topics discussed in the text.

A. Constructing an IFA Rule Table from its Code Number

`ToFARule` will output the list of IFA rules using as input the code number of the rule and the number of states and colors.

```
In[1]:= ToFARule[n_Integer,{s_Integer,k_Integer}]:=
        (*n:rule number,s:number of states,k:number of colors*)
            Flatten[MapIndexed[{1,-1}#2+{0,k}→
            Mod[Quotient[#1,{k,1}],{s,k}]+{1,0}&,
            Partition[IntegerDigits[n,sk,s k],k],{2}]]
```

Example:

```
In[2]:= ToFARule[26853, {2, 3}]

Out[2]= {{1, 2} → {2, 0}, {1, 1} → {1, 2}, {1, 0} → {2, 1},
         {2, 2} → {1, 1}, {2, 1} → {2, 2}, {2, 0} → {2, 0}}
```

B. Transforming Rules from ECA to IFA

`RuleTransform` will output the rules of the IFA when given the ECA rule code number.

```
In[3]:= RuleTransformer[ECARule_Integer]:=Module[{rule,i,rTab},
            rule=Reverse[IntegerDigits[ECARule,2,8]];
            Table[rTab=IntegerDigits[i,2,3];{FromDigits[Take[rTab,2],2]
                Last[rTab]}→{FromDigits[Take[rTab,-2],2]+1,
                rule[[i+1]]},{i,7,0,-1}]]

In[4]:= RuleTransformer[110] (*Transform ECA 110 to IFA*)

Out[4]= {{4, 1} → {4, 0}, {4, 0} → {3, 1},
         {3, 1} → {2, 1}, {3, 0} → {1, 0}, {2, 1} → {4, 1},
         {2, 0} → {3, 1}, {1, 1} → {2, 1}, {1, 0} → {1, 0}}
```

C. Transforming Initial Conditions from ECA to IFA

`ICTransformer` will give the correct IFA initial conditions when given as input the initial condition list of an ECA and the number of steps to be emulated.

```
In[5]:= ICTransformer[CAIC_List,steps_Integer]:=
        (*CAIC:cells configuration of the ECA,
        steps:number of steps to be emulated*)
            Module[{sz,tab,tab1,tab2},
                sz=Length[CAIC];
                tab=If[IntegerPart[steps/sz]>0,
                    Nest[Join[#,CAIC]&,CAIC,IntegerPart[steps/sz]-1],{}
                tab1=Join[Take[CAIC,-Mod[steps,sz]],tab];
                tab2=Join[tab,Take[CAIC,Mod[steps,sz]]];
                Join[{0},tab1,CAIC,tab2]]
```

Example: The initial condition of an ECA is a random list with 10 {0, 1}, and the step to be emulated is 10.

```
In[6]:= ICTransformer[Table[RandomInteger[{0, 1}], {10}], 10]

Out[6]= {0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
         1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1}
```

D. Transforming Rules from IFA to One-Dimensional CA

`CARuleTransformer` will output the correct rules for a one-dimensional CA that emulates an IFA when given the IFA's code number and the number of states and colors.

```
In[7]:= CARuleTransformer[n_Integer,s_Integer,k_Integer]:=
          Module[{IFARule,rule1,rule2,k1,rule3,rule4},
            IFARule=ToFARule[n,{s,k}]/.Rule→List;
            rule1=
              Flatten[Table[{{IFARule[[i,1,1]] k+IFARule[[i,1,2]]
                x_/;x<k,__}→
              IFARule[[i,2,1]] k+x,{__,
              IFARule[[i,1,1]] k+IFARule[[i,1,2]],__}→
              IFARule[[i,2,2]]},{i,1,Length[IFARule]}]];
            rule2=
              Join[rule1,
              FilterRules[{#→#[[2]]}&/@Tuples[Range[0,(s+1) k-1]
                Except[rule1]]];
            k1=(s+1) k;
            rule3=
              ReplacePart[#,{0}→
                Rule]&/@({Append[#,k1],Append[#,0]/.rule2}&/@
                Tuples[Range[0,(s+1) k-1],2]);
            rule4=
              ReplacePart[#,{0}→
              Rule]&/@({Prepend[#,k1],Prepend[#,0]/.rule2}&/@
              Tuples[Range[0,(s+1) k-1],2]);
            Join[{{x_/;x<k,k1,__}→k1,{x_/;x≥k,k1,__}→
              k1+1,{k1+1,x_,__}→k+x,{__,k1+1,__}→
              k1,{__,x_,k1+1}→x},rule2,rule3,rule4]]
```

Example: Transform the (2,3) IFA 3615 into CA rules.

```
In[8]:= CARuleTransformer[3615, 2, 3] // Short
```

```
Out[8]//Short=
{{x$_ /; x$ < 3, 9, __} → 9, {x$_ /; x$ ≥ 3, 9, __} → 10,
 {10, x$_, __} → 3 + x$, {__, 10, __} → 9, <<784>>,
 {9, 8, 5} → 8, {9, 8, 6} → 8, {9, 8, 7} → 8, {9, 8, 8} → 8}
```

E. Transforming Initial Conditions from IFA to CA

`CAInitialTransformer` will generate the initial conditions of the one-dimensional CA that can emulate the given IFA when given as input the initial conditions and the number of states and colors of an IFA.

```
In[9]:= CAInitialTransformer[ini_,s_,k_]:=Join[ini,{(s+1) k+1}]
```

```
In[10]:= CAInitialTransformer[Table[0, {10}], 2, 3]
```

```
Out[10]= {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10}
```

References

[1] H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, 2nd ed., New York: Prentice-Hall, 1998.

[2] A. Gill, *Introduction to the Theory of Finite-State Machines*, New York: McGraw-Hill, 1962.

[3] J. Carroll and D. Long, *Theory of Finite Automata with an Introduction to Formal Languages*, New York: Prentice-Hall, 1989.

[4] M. Waldrop, *Complexity: The Emerging Science at the Edge of Order and Chaos*, New York: Simon & Schuster, 1992.

[5] J. H. Holland, *Hidden Order: How Adaptation Builds Complexity*, Menlo Park, CA: Addison-Wesley Publishing Company, 1995.

[6] S. Wolfram, *A New Kind Of Science*, Champaign, IL: Wolfram Media, Inc., 2002.

[7] S. Wolfram. "Iterated Finite Automata." (November 17, 2003) www.stephenwolfram.com/publications/informalessays/iteratedfinite.