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

Iterated Finite Automaton (IFA) is invented by Wolfram for studying the conventional Finite State Automata (FSA) by means of A New Kind of Science (NKS) methodology. An IFA is a composition of a finite state automaton and a tape with limited cells. The complexity of behaviors generated by various FSA operating on different tapes can be visualized by two dimensional patterns. Through enumerating all possible 2-state and 3-color IFA, this paper shows that there are a variety of complex behaviors in these simple computational systems. These patterns can be divided into 8 classes roughly such as regular patterns, noisy structures, complex behaviors, and so forth. Also they show the similarity between iterated finite automata and elementary cellular automata. Furthermore, any cellular automaton can be emulated by an IFA and vice versa. That means IFA supports universal computation.

1. Introduction

Finite state automaton (FSA) or finite state machine (FSM) 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 only with finite internal states and fixed transition rules. Usually, there are two kinds of FSA. Finite state acceptors (recognizers) only accept information and jump between different states but not generate any output information. These machines are widely used as language recognizers[3]. Another class is called finite state transducers which is 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]. It stresses a different philosophy and methodology to approach complex system problems[4]. Stephen Wolfram’s A New Kind of Science (NKS) is a rep-

This paper is based on the research in 2007 NKS summer school.

†Electronic mail address: Zhangjiang@amss.ac.cn"
resentative of complex systems studies[6]. NKS mainly focuses on the
complexity generated by different very simple computational systems.
Therefore, building the simplest systems, implementing their computa-
tions, observing their behaviors and drawing conclusions are main
steps of NKS approach. All kinds of computational systems including
Turing Machines, Substitution Systems, and so on were studied by this
method in his famous book "A New Kind of Science"[6].

What kind of complex behaviors can the FSA perform? Can we
use the methodology of NKS to study this specific system? Wolfram
invented Iterated Finite Automata (IFA) [7] to answer these questions.
By adding a tape with finite size and some other constraints for FSA,
we study their behaviors just like one dimensional cellular automata.
Wolfram has enumerated all of possible patterns of 2-state 2-color and
3-state 2-color IFA. This paper mainly studies the complex behaviors
of 2-state 3-color IFA. We divide the patterns into 8 groups roughly
including regular patterns, noisy structures and complex behaviors,
etc. Furthermore, 2-state 3-color IFA show the similarity between IFA
and 1 dimensional Cellular Automata(CA). That encourages the au-
thor to study emulation relationships between IFA and 1-d CA. The
result makes us conclude that IFA as a whole family supports universal
computation.

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

2. Iterated Finite Automata

To illustrate what is iterated finite automata and how they work, we
should give some formal definitions at first.

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 also a
finite set of output symbols. And \( f : S \times I \rightarrow S \times O \) is a function. \( f \)
can be represented by a set of transition rules. Each rule has the form
\( r : (s, i) \rightarrow (s', o) \), where \( s, s' \in S, i \in I, \) and \( o \in O. \ s_0 \in S \) is the
initial state.[1]

Another representation of a finite state transducer is graph in which
vertices are states in \( S \) and directed edges are transitions between
states, i.e., 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 be also represented by a graph (See Figure 1).

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

Definition 2. An iterated finite automaton is a pair: $\Omega = (\Gamma, \alpha)$. Where $\Gamma$ is a finite state transducer with the same input and output set, i.e., $\Gamma = (I, S, I, f, s_0)$. 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. And $\alpha = I^n$ is a tape with $n$ cells, each member in $\alpha$ is $\langle a_1, a_2, \ldots, a_n \rangle$ indicating cells with different colors, i.e., the $i^{th}$ cell’s color is $a_i \in I$. So any member of $\alpha$ is a configuration of the tape.

Now we will discuss how a finite state transducer $\Gamma$ operates on the tape $\alpha$. Assume that the initial configuration of $\alpha$ is $\langle a_1^0, a_2^0, \ldots, a_n^0 \rangle$. The machine $\Gamma$ 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 output and 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 $\Gamma$ will move to the second cell.

Definition 3. The process of the finite state transducer $\Gamma$ operating on a cell of the tape including reading the input information from the cell, updating the cell’s color and move to the next cell is called a step of the IFA.
And then, the machine $\Gamma$ will repeat this process step by step until it reaches the last cell of the tape. Then the IFA finish a turn.

**Definition 4.** For IFA $\Omega = (\Gamma, \alpha)$, $n$ (the number of cells in $\alpha$) updating steps of $\Gamma$ on the tape $\alpha$ 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: $(a_1^1, a_2^1, ..., a_n^1)$. 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 2-dimensional pattern of the IFA.

**Definition 5.** A pattern of an IFA is a sequence of tape configurations: $(C_1, C_2, ..., C_T)$, where $C_i \in \mathcal{I}$.

For example, consider the finite state transducer in Example 1 is working on a tape with 5 cells, then three turns are shown as Figure 2.

![Figure 2](image_url)

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

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

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, i.e., $a_1^0 = a_2^0 = ... = a_n^0 = 0$. Another convention is we treat all machines with the same $s$, the number of states and $c$, the number of input/output symbols as a same class IFA which is denoted as a pair $(s, c)$. There are $(s \times c)^s \times c$ possible rules in the class $(s, c)$ IFA. 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 IFA is shown in Appendix A.
3. Complexity of IFA

3.1 The Complexity of (2,2) IFA and (2,3) IFA

Wolfram had studied the complexity of (2,2) IFA and (2,3) IFA in [7]. Most (2,2) IFA exhibit trivial structures such as blank tapes and cyclic patterns except some nested structures. But for (2,3) IFA, more nested structures were found and some random structures were discovered. Some selected (2,2) IFA patterns and (2,3) IFA patterns are shown in Figure 3.

![Figure 3](image)

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

3.2 The Complexity of (2,3) IFA

The author investigated the complex behaviors of (2,3) IFA by systematic searching. There are total $6^3 = 46656$ possible (2,3) IFA rules. After filtering out a large number of trivial patterns which are cyclic structures and homogenous colors, there are still 1580 IFA. Then the author divided them into 8 classes roughly according to their behaviors (See Figure 4), they are:

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 also can 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 easy to be found in IFA.

4. Random Triangles (Ran Tri)
   Another kind of structure also contains numerous triangles but the
global patterns composed by these local triangles are not obviously nested but exhibit randomness in some sense.

(5) Nested Structures (Nested S)

There are many nested structures but not triangles which are classified as this class. Some basic structures as building blocks are contained by themselves.

(6) Thin Guys (Thin G)

**Figure 4.** Complex behaviors in (2,3) IFA with different classes.
Some structures have random and complex interesting behaviors only in a very thin area on the left part of the pattern.

(7) Irregular Patterns (Irreg)

A large number of IFA are very difficult to be classified as one of the groups listed above. 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 IFA as an alternative group named irregular patterns.

(8) Complex structures

Although it is very difficult to distinguish the complex structures fallen into this group from irregular patterns, some IFA 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 to be shown in Figure 5.

The complexity of patterns in the complex structure class makes us to guess that IFA may support universal computation. This hypothesis will be further confirmed by the facts mentioned in the next subsection.

3.3 Similarity between IFA and CA

From patterns investigated above, it is not difficult to see that there is much similarity between IFA and CA. For example, the pattern of IFA 3507 is similar to the elementary CA (1-dimensional CA with 2 colors and 2 neighbors, ECA) 30. The similarity can only be shown by flipping the pattern of ECA30 left to right and shear 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 ECA225. Also the pattern of ECA225 should be flipped and sheared. (See Figure 6)

These similarities encourage us to propose that IFA may emulate ECA.

4. Universality of IFA

4.1 IFA emulates ECA

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

It is very natural to point out that the tape of cells in IFA can be regarded as the cells in the one-dimensional cellular automaton. But the major difference between IFA and CA from their working mechanism is the former updates the cells on the tape step by step; however
Figure 5. Complex patterns may support sophisticated computations.

the latter updates all of the cells simultaneously. This difference recurs us 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 (2-color, 2-neighbor), we can construct a specific 4-state 2-color IFA to emulate that ECA. At the beginning, the tape of IFA is configured to duplicate the configurations of cells in ECA. And the finite state machine will scan the cells on the tape one by one. There are 4 possible combinations of two adjacent cells. Therefore, 4 states can memorize these combinations. It is not difficult to construct a rule table for the finite state automaton that performs the same computation as the ECA. Then the IFA produces the same outcome...
on the third cell as the ECA’s second cell.
To articulate the process mentioned above, we construct a concrete IFA to emulate ECA110. (See Figure 7)

Suppose that the initial tape is 0100..., then ECA110 will give an output: #10... (The first cell is undetermined in this case). A 4-state 2-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. Then it will keep the state according to its rule. And 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 output of IFA on the third cell is exactly same as the ECA. After that, by 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 ECA110. Thereafter, they can perform exactly the same as the preceding cells.

Another issue is the boundary condition. Because ECA have cyclic boundary conditions, nevertheless, IFA 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 IFA is determined by the number of steps we want to emulate times 2 plus the number of cells of ECA in cases of the number of steps is divided by the number of cells (For more complicated cases, please refer the Appendix C). For example, the specific IFA that had been constructed above can emulate the ECA110 100 steps by duplicating the initial (100) cells of ECA110 3 times ($300 = 2 \times 100 + 100$). (See Figure 8)

![Figure 8. Comparison of patterns between ECA110 and IFA6542901 for 100 steps. The middle area without gray mask shows same behavior as ECA110. Notice that to solve the issue of boundary conditions, the number of cells in IFA is 3 times of the number of cells in the ECA.](image)

The Mathematica code for constructing IFA rule and initial condition to emulate the ECA are listed in Appendix B and C.

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

4.2 CA emulating IFA

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

![Figure 9](image)

The codes for transformations of rules and initial conditions are shown in the Appendix D and E.
Any iterated finite automaton is a composition of a conventional finite state transducer and a tape. This extended definition of finite state automata not only facilitates our investigation by means of Wolfram’s 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 2 states and 3 possible cell colors. After enumerating all possible (2,3) IFA, we show the spectrum of complex patterns generated by them. These patterns can be divided roughly into 8 classes including regular patterns, noisy structures, nested triangles and so forth. Among those classes, an important class (complex class) 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 IFA and elementary CA is pointed out.

Any elementary CA can be emulated by a (4,2) IFA. That means IFA supports the 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 non-trivial computational process may support universal computation.

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

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

5. Conclusions
Acknowledgments

Thanks Stephen Wolfram, Todd Rowland, Eric Rowland and Jason Cawley for discussions in the NKS 2007 summer school. Thanks for the support of Guozhi Xu Post Doctoral Research Foundation and National Natural Science Foundation of China (No.60574068).

Appendix

A. Mathematica code for constructing an IFA rules table from its code number

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}] ]

B. Mathematica Code for rule transformation from ECA to IFA

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

C. Code for initial condition transformation from ECA to IFA

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]
]
D. Code for rule transformation from ECA to IFA

CARuleTransformer[n_Integer, s_Integer, k_Integer, s0_] :=
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], 3],
    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_, __} -> s0 k + x, {__, k1 + 1, __} ->
    k1, {__, x_, k1 + 1} -> x}, rule2, rule3, rule4]
]

E. Code for initial condition transformation from ECA to IFA

CAInitialTransformer[ini_, s_, k_, s0_] := Join[ini, {(s + 1) k + 1}]

References


Complex Systems, volume (year) 1–1+
