Lecture Note

An Introduction to Formal Language Theory

- from Finite State Automaton to Turing Machine -

Akira Imada
Brest State Technical University

Last modified on 20 April 2017

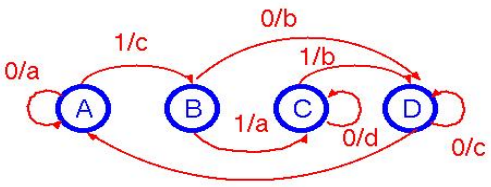
I. Finite State Automaton

What is Finite State Automaton?

Finite State Machine is a machine that makes an action depending on imput and its current state then change state to a new one. For example:

Current State	inputs	actions	New State
A	0	a	A
	1	c	B
B	0	b	D
	1	a	C
C	0	d	C
	1	b	D
D	0	c	D
	1	d	A

Transition Table



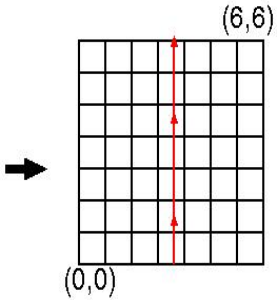
Its diagram

Simplest Example

Now let's think of a simple (FSA) which explors a grid world. This FSA has only one state A. It receives input from a cell in front. All cells are empty. So, input is always 0. Action is always "Go one step in front."

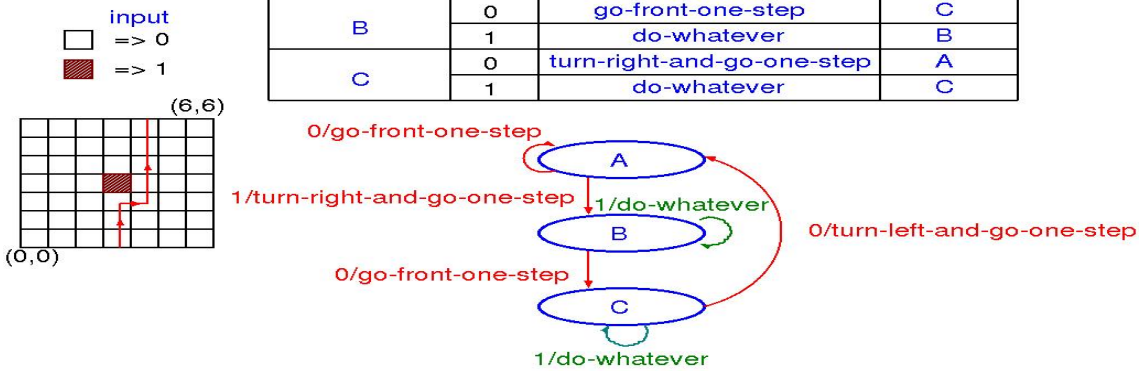
current state	input	action	new state
A	empty	go up one step	A

Or

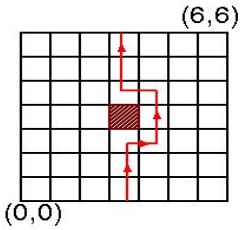


An FSA with 2 inputs 2 states and 4 actions

current state	input	action	new state
A	0	go-front-one-step	A
	1	turn-right-and-go-one-step	B
B	0	go-front-one-step	C
	1	do-whatever	B
C	0	turn-right-and-go-one-step	A
	1	do-whatever	C



Excersize

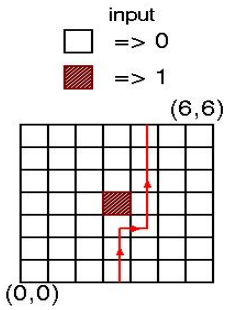


A little more complicated FSA

FSA which avoids obstacle then return to the original route

Try to design this FSA with input and actions being same as before but with more states

Representation of FSA with binary chromosome



current state	input	action	new state
A	0	go-front-one-step	A
	1	turn-right-and-go-one-step	B
B	0	turn-left-and-go-one-step	A
	1	do-whatever	B

action
go-front-one-step => 00
turn-right-and-go-one-step => 01
turn-right-and-go-one-step => 10
do-whatever => 11

0	0	00	0
	1	01	1
0	0	00	0
	1	01	1

(000011000011)
Let's call this chromosome of this FSA

Excersize

From random chromosom to FSA

1. *Create a binary chromosome with 12 genes at random.*
2. *Translate it to state transition table of FSA with 2 states, 2 inputs and 4 outputs.*
3. *Starting from (4,0) run the FSA on gridworld of 7x7 where only one obstacle locates at (3,3).*
4. *Report (1) chromosome, (2) table or diagram of FSA, and (3) trace of FSA.*
5. *Repeat from 1. to 4. above 10 times*

Excersize

An evolution of FSA

1. *Create a population of N random binary chromosome with 12 genes each corresponding to FSA*
2. *Make each of the FSA move 8 steps and record the final location as (X, Y).*
3. *evolve the population with fitness being $|X - 4| + |Y - 7|$ until fitness becomes stable.*
4. *Report evolution by graph of generation vs. fitness*

II. Formal Language

Yet another type of FSA?

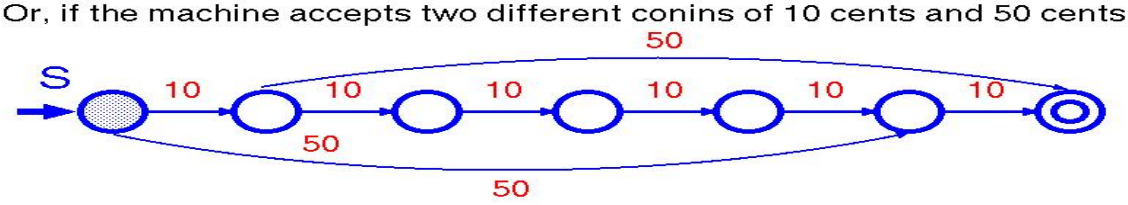
Next, we think of FSAs which accept or reject *a series of inputs*

Bending machine as an Example

Assume a bottle of cola is 60 cents.

When the bending machine accepts only 10 cents coin:

Or, if the machine accepts two different conins of 10 cents and 50 cents:



Production Rule

A legitimate string can be generated by a set of rules.

E.g.

Cat runs.

can be generated by two rules

S => N V
N => cat
V => run

in such a way as

S => N V => cat V => cat run

Where S, N, V are called non-Terminal and cat, run are called Terminal

Four model-languages as toy examples

(1) strings over {a,b,c} starting with a

E.g. abbc, aaaccb, acbac but not baac

(2) strings over {a,b,c,} with alphabetical order

E.g. abc, abbcc, aaabccc but not acb, cab,

(3) strings over {a,b} of the form $a^n b^n$

E.g. aaabbb, ab, aaaabbbb but not aab

(4) palindromes over {a,b}

E.g. bab, aabaa, abbbbbba but not baac

"Madam! I'm Adam." is an example in our language)

Production rule for strings starting with ”a”

Production rule to produce strings over {a,b,c} starting with "a" as an example

S => a A
A => a A
A => b A
A => c A
A => a
A => b
A => c

E.g.

S => a A => a b A => a b c

S => a A => a c A => a c c A => a c c c A => a c c c b

S => a A => a c A => a c b A => a c b a A => a c b a b A => a c b a b b

Production rule for strings with alphabetical order

Generating rule of strings over {a,b,c} with alphabetical order can be described as

S => a A
S => b B
S => b C
S => c C
S => a
S => b
S => c
A => a A
A => b B
A => c C
A => b
B => b B
B => b C
B => c C
C => c C
C => c

E.g.

S => a A => a a B => a a b C = a a b c C => a a b c c

S => a A => a b

S = b C => b c

Counter example:

Try to derive for example

a c b, c c a b b, b a c, ...

Production rule for strings with a form of $a^n b^n$

S => a S b
S => a b

E.g.

S => a S b => a a S b b => a a a b b b

Natural language with a form of $a^n b^n$

A natural language sometiemes have this form $A^n B^n$

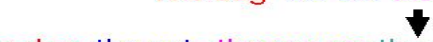
where, e.g.,

A = {dog, cat, mouse, monkey} and B = {die, chase, love chase}

The dog died.

The dog the cat chased died.

The dog the cat the mouse bite chased died.



The dog the cat the mouse the monkey loved bite chased died.

A more complicated example:

Anyone who feels that if so-many more students whom we havent actually admitted are siting in on the course than ones we have that the room had to be changed, then probably auditors will have to be excluded, is likely to agree that the curriculum needs revision.

Production rule for palindrome

S => a S a
S => b S b
S => a ... necessary when even characters are odd
S => b ... necessary when even characters are odd
S => ε ... necessary when even characters are even

E.g.

S => a S a => a b S b a => a b b b a
S => b S b => b a S a b => b a ε a b => b a a b

Counter example

Try to produce for example

a b a b, abbbbaa, etc

those are not palindromes

1. Regular Gramar

The regular-grammer is constructed rules whose form is

$$A \Rightarrow a\ B$$

where a is a termianal and A and B are non-terminal
B might be empty

Confirm that (1) strings starting with a and (2) alphabetical order are in this category while (3) $a^n\ b^n$ and palindrome are not

(1)

S	=>	a A
A	=>	a A
A	=>	b A
A	=>	c A
A	=>	a
A	=>	b
A	=>	b

(2)

S	=>	a A
S	=>	b B
S	=>	b C
S	=>	c C
S	=>	a
S	=>	b
S	=>	c
A	=>	a A
A	=>	b B
A	=>	c C
A	=>	b
B	=>	b B
B	=>	b C
B	=>	c C
C	=>	c C
C	=>	c

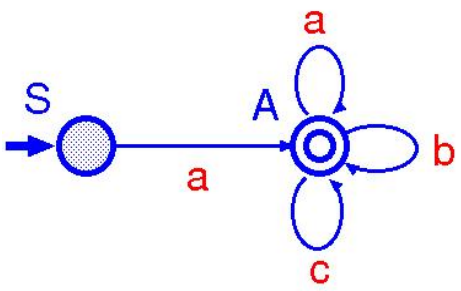
(3)

S	=>	a b
S	=>	a S b

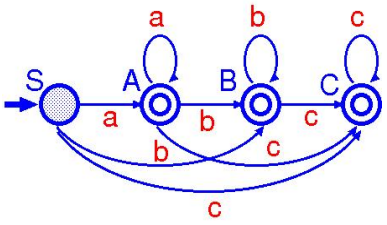
(4)

S	=>	a S a
S	=>	b S b
S	=>	a
S	=>	b
S	=>	ε

FSA that accepts or rejects a strings starting with "a"



FSA that accepts or rejects a strings with alphabetical order



2. Context-free Grammar

The context-free grammer is constructed rules whose form is

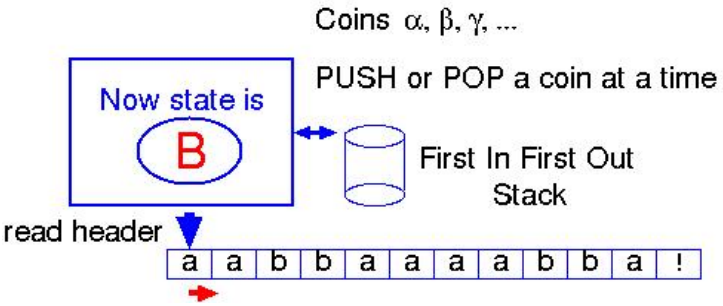
$$A \Rightarrow \alpha$$

where α is a non-empty sequece of either terminal or non-terminal

For example, grammar for strings over {a,b} that have a form $a^n b^n$ can be

S => a b
S => a S b

Pushdown Automaton - its diagram



A toy example of transition table

state	input	action	new state
A	a	push α	A
	b	pop	B
B	a	push β	A
	b	pop	B

$a^n b^n$ recognition by Pushdown Automaton

state	input	action	state
A	a	push	A
	b	stop and reject	–
	z	nothing	B
B	a	stop and reject	–
	b	pop	B
	!	accept only if stack is empty	–

E.x.

a	a	a	z	b	b	b	!

push => push => push => pop => pop => pop => check stack

accept because stack is empty

a	b	a	z	b	a	b	!

push => stop and reject

Palindrome recognition by Pushdown Automaton

state	input	top of stack	action	state
A	a	whatever	push α	A
	b	whatever	push β	A
	z	whatever	nothing	B
B	a	α	pop	B
		β	stop and reject	—
		ε	stop and reject	—
	b	α	stop and reject	—
		β	pop	B
		ε	stop and reject	—
	!		accept only if stack is empty	—

E.x.

a	b	b	z	b	b	a	!

push α => push β => push β => pop => pop => pop => check stack

accept because stack is empty

a	b	z	a	b	!

push α => push β => reject

Then what about odd number string?

Palindrome – when number of character is odd

state	input	top of stack	action	state
A	a	whatever	push α	A
	b	whatever	push β	A
	z	whatever	pop	B
B	a	α	pop	B
		β	stop and reject	—
		ε	stop and reject	—
	b	α	stop and reject	—
		β	pop	B
		ε	stop and reject	—
	!		accept only if stack is empty	—

E.x.

a	b	b	z	b	b	a	!

push α => push β => push β => pop => pop => pop => check stack

accept because stack is empty

a	b	b	a	z	b	b	a	!

Excersize

Pushdown Automaton which recognize $a^n b^n$

Programm a black box whose input is any digit of a series of 0 or 1 from keyboad, and if this input is a form of $0^n 1^n$*, then display "Yes" on the screen and "No" otherwise.*

Although another much simpler program can recognize $a^n b^n$ *more easily without using PDA, don't make it!*

Because our goal is to understand PDA!

Excersize

Pushdown Automaton which recognize palindrome

Programm a black box whose input is any even number digit of a series of 0 or 1 from keyboard, and if this input is a palindrome, then display "Yes" on the screen and "No" otherwise.

When the programm run with "abzba" from keyboard, such following outputs should be on the screen.

$(A\ x) \rightarrow (A\ yx) \rightarrow (B\ yx) \rightarrow (B\ y) \rightarrow (B\ \epsilon) \rightarrow$ *accept!*

or

for input "babzbaa"

$(A\ y) \rightarrow (A\ xy) \rightarrow (A\ yxy) \rightarrow (B\ yxy) \rightarrow (B\ xy) \rightarrow (B\ y) \rightarrow$ *reject!*

or

for input "bazba"

$(A\ y) \rightarrow (A\ xy) \rightarrow (B\ xy) \rightarrow$ *reject!*

Also much simpler coding exists, that is,
"if input string maches its reverse then accept otherwise reject,"
but notice that
our goal is to understand pushdown automaton,

not palindrome itself.

3. Context Sensitive Grammar

The context-sensitive grammer is constructed rules whose form is

$$\alpha A \beta => \gamma a \delta$$

where

A is a single non-terminal, a is a single terminal,

α, β, γ and δ are strings of non-terminals and terminals.

$a^n b^n c^n$ as an example of Context Sensitive Grammar

S	=> a S A B C
S	=> a A B C
A B C	=> b B
A B C	=> b c
A B C	=> c C
B	=> b B
C	=> c C
B	=> b
C	=> c

E.g.

S => a A B C => a b c

S => a S A B C => a(a A B C)A B C => a a(b B)A B C => a a b b(c C)=> a a b b c c

S => a S A B C => a(a S A B C)A B C => a a(a A B C)A B C => a a a(b B)A B C => a a a b b b A B C => a a a b b b(c C)=> a a a b b b c c c

S => a S A B C => a(a S A B C)A B C => a a(a S A B C) A B C => a a a(a A B C)A B C => a a a a(b B)A B C => a a a a b(b B)A B C
=> a a a a b b(b B)A B C => a a a a b b b b(c C)=> a a a a b b b b c c C => a a a a b b b b c c c C => a a a a b b b b c c c c

$a^n b^n c^n$ as an example of Context Sensitive Grammar with much simpler rule

S => a S B c
S => a b c
b B => b b
c B => B c

E.g.

S => a b c

S => a S B c => a (a b c) B c => a a b(B c) c => a a(bb)c c => a a b b c c

S => a S B c => a (a S B c) B c => a a(a b c)B c B c => a a a b(B c)c B c => a a a(b b)c c B c
=> a a a b b c(B c)c => a a a b b(B c)c c => a a a b(b b)c c c => a a a b b b c c c

$a^n b^n c^n$ as an example of Context Sensitive Grammar yet another one

S	=>	a A b c
S	=>	a b c
A b	=>	b A
A c	=>	B b c c
a B	=>	a a
a B	=>	a a A
b B	=>	B b

E.g.

S => a A B C => a b c

S => a S A B C => a(a A B C)A B C => a a(b B)A B C => a a b b(c C)=> a a b b c c

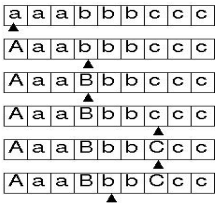
S => a S A B C => a(a S A B C)A B C => a a(a A B C)A B C => a a a(b B)A B C => a a a b b b A B C => a a a b b b(c C)=> a a a b b b c c c

S => a S A B C => a(a S A B C)A B C => a a(a S A B C) A B C => a a a(a A B C)A B C => a a a a(b B)A B C => a a a a b(b B)A B C
=> a a a a b b(b B)A B C => a a a a b b b b(c C)=> a a a a b b b b c c C => a a a a b b b b c c c C => a a a a b b b b c c c c

Diagram of Linear Bounded Automaton

$a^n b^n c^n$ recognition by Linear Bounded Automaton

1. Starting from leftmost cell, scan input to right, replace the first "a" with A.
2. Move to the first "b" and replace it with B.
3. Move right to the first "c" and replace it with C.
4. Move left scanning "B" and again move to left till the first "b"
5. Repeat 3-4.



Formal Grammar – Summary so far

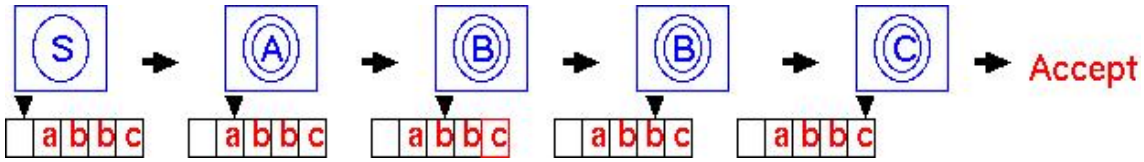
Grammar	Machine to recognize	Form of rules	Examples
Regular Grammar	Finite State Automaton	A => a B	strings starting "a" strings alphabetical order
Context Free Grammar	Pushdown Automaton	A => α	$a^n b^n$ palindrome
Context Dependent Grammar	Least Bounded Automaton	α A β => γ a δ	$a^n b^n c^n$
Free Grammar	Turing Machine		

A is a single non-terminal, a is a single terminal, and $\alpha, \beta, \gamma, \delta$ are strings of no-terminal and terminals which could be empty.

Diagram of FSA for comparison

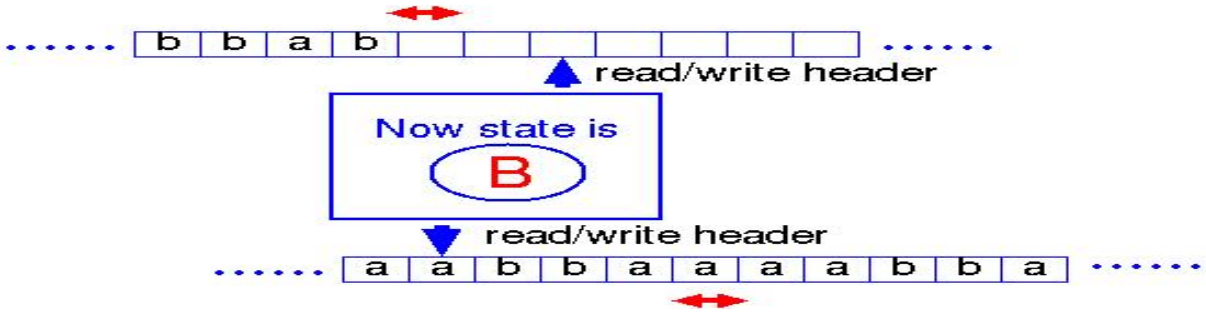
Now notice that FSA can be shown with LBA-like diagram!

E.g. – strings of alphabetical order by FSA



4. Free Grammar

Diagram of Turing Machine



Appendix I – Coding issues

1. Programming to simulate a FSA

```
IF (x, y, direction) == (4, 3, up) OR (5, 4, left) OR (4, 5, down) OR (3, 4, right) THEN   input = 1
                                ELSE   i = 0
                            x = 4, y = 0, direction = up
                    WHILE counter < 9   {
                                    counter++
        IF (state, direction, input) == (A, up, 0) THEN (action, direction, state) = (y++, up, A)
        IF (state, direction, input) == (A, up, 1) THEN (action, direction, state) = (x++, left, B)
                                    ⋮
                        PRINT  (x, y, direction, state)
                                etc
                                    }
```

2. Translation of chromosome into transition table

Create chromosome with 12 binary genes at random
($x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}$)

print ("A 0"); if ($2^* x_1 + x_2$) == 0 print ("Go one step") else if ($2^* x_1 + x_2$) == 1 print ("Turn left")
else if ($2^* x_1 + x_2$) == 2 print ("Turn left") else if ($2^* x_1 + x_2$) == 3 print ("Do nothing");
If x_3 == 0 print ("A") else print("B");

CR

⋮

If x_{12} == 0 print ("A") else print("B");

⬇ This create depending on chromosom, e.g.,

A	0	Go one step	A
A	1	Turn left	B
B	0	Go one step	B
B	1	Turn right	A

3. PDA for even number palindromes

```
Initialize
    state = A;
    stack = {}
main() {
        while x =! 9   {
        accept input x from kyeboard;                          % assumming input is 0,1,8,9
                IF state == A
                    IF x == 0 THEN push a;
                        ELSE IF x == 1 THEN push b;
                        ELSE state = B;                        % iuput is 8 (center of string)
                 ELSE                                          % state is B
                    IF {x == 0 && stack == a} THEN pop
                        ELSE IF {x == 0 && top_of_stack == b} THEN stop and reject
                        ELSE IF {x == 1 && top_of_stack == b} THEN pop
                 PRINT ({state, whole stack} "=>")
                 }
       IF stack == empty  THEN                                 % input of string has all done
                PRINT "accepted!"
       ELSE
                PRINT "rejected!"
     }
```

Appendix II – Cell Automaton

What is Cell Automaton?

Cell Automaton is constructid with an any dimension of grid

Each cell in the grid takes one state out of k states

Each of these cells are determined being affected by the states of neighbor cells

One Dimensional Cell Automaton

Each cell has a state either '0' or '1'

The state of a cell is determined by the state of itself and its right and left cell

E.g.

The cell of state '1' becomes '0' at time t+1 if its right is '0' and left is '1' at time t.

Let's call this (0 1 1) => 0

We have 8 such rules to specify states of all cells at time t+1

Hence there are 256 such rules.

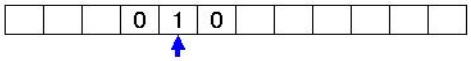
Each rule is indexed by its decimal representation of the binary

Excersize

Apply the Rule 30 to 20 1-D cells starting with all '0' state. Display the evolution from t = 1 to 20

One Dimensional Cell Automaton

Assuming 2 states of "0" and "1"
the state of each cell changed according to its right cell and left cell
E.g.



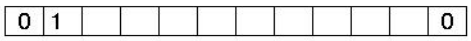
The state of 5-th cell from the leftmost changed according to the state of 4-th and 6th using a given rule such as

0 0 0 => 1
0 0 1 => 1
0 1 0 => 1
0 1 1 => 1
1 0 0 => 0
1 0 1 => 1
1 1 0 => 0
1 1 1 => 0

Now that the state of 5-th cell is "1" and the 4-th is "0" and 6th is "0" the 5-th change from 1 to 0

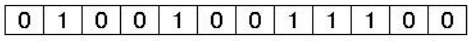
Then question is, what about the following 10-th cell?

We also assume the cells are troidal which means the left-hand neighbor of the 1-st cell, e.g., is the last cell

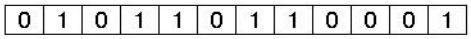


So, in this case, the 1-st cell of "0" changed to "1" according to the 2-nd line of the rule 0 0 1 => 1

Thus, the following pattern at t = 0



evolves to be as follows at t = 0



Then question is, what about the pattern at time 1?

One Dimensional Cell Automaton (continued)

The rule in the previous page

0 0 0 => 1
0 0 1 => 1
0 1 0 => 1
0 1 1 => 1
1 0 0 => 0
1 0 1 => 1
1 1 0 => 0
1 1 1 => 0

is called Rule 47 because $(0\ 0\ 1\ 0\ 1\ 1\ 1\ 1)_2 = (47)_{10}$

Then what is the Rule 30?

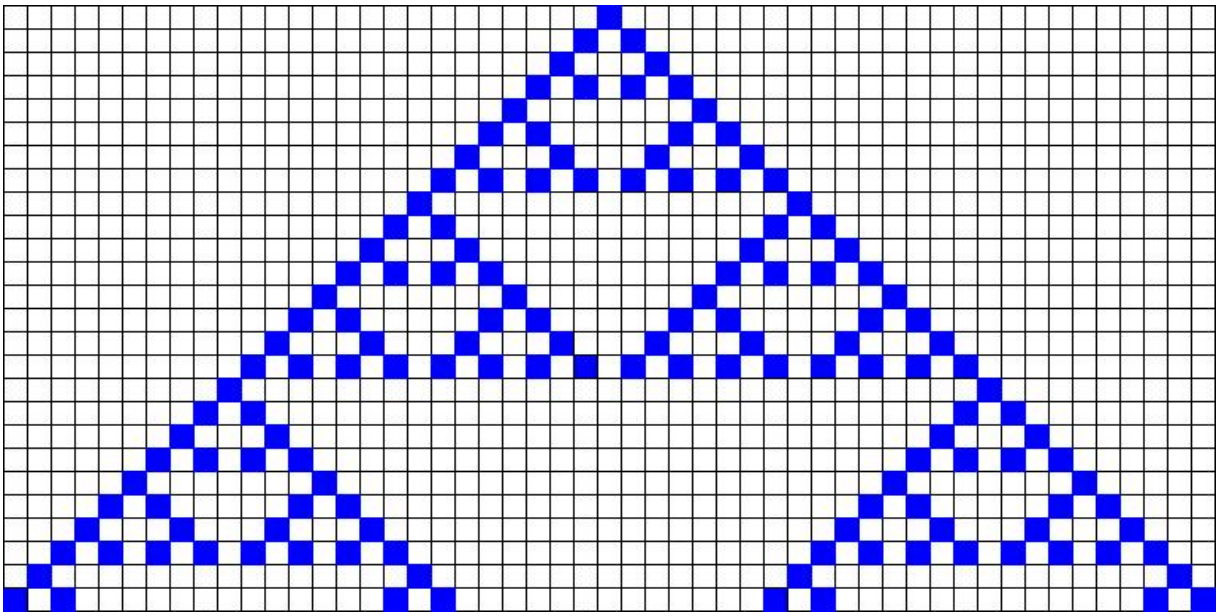
Or

How many such rules exist?

Excersize

Starting with 20 all "0" cells, evolve the cells from t = 0 to t = 20, by applying Rule 30

An example



Coding of a 1-D Cell Automaton

Assuming Rule is

(000)=>a(1)
(001)=>a(2)
(010)=>a(3)
(011)=>a(4)
(100)=>a(5)
(101)=>a(6)
(110)=>a(7)
(111)=>a(8)

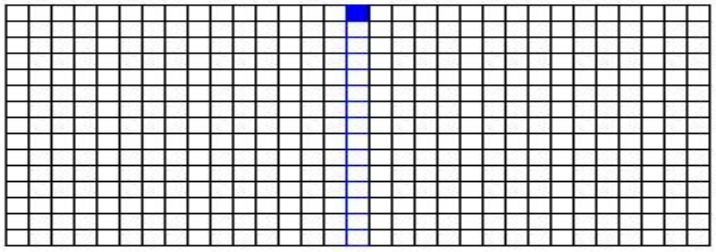
E.g. for Rule-30 (a(8)a(7)a(6)a(5)a(4)a(3)a(2)a(1))=(00011110)

```
initialize( ){
    for i = 1 to 20 x(i) = 0
    t = 0}
main( ){
    while t < 20 {
        for i = 1 to 20 {
             if      i =  1 then i - 1 = 20   % when cell is left  most
             else if i = 20 then i + 1 = 0    % when cell is right most
             else {
                   if      x(i -1) = 0 & x(i) = 0 & x(i + 1) = 0 then y(i) = a(1)
                   else if x(i -1) = 0 & x(i) = 0 & x(i + 1) = 1 then y(i) = a(2)
                   else if x(i -1) = 0 & x(i) = 1 & x(i + 1) = 0 then y(i) = a(3)
                   else if x(i -1) = 0 & x(i) = 1 & x(i + 1) = 1 then y(i) = a(4)
                   else if x(i -1) = 1 & x(i) = 0 & x(i + 1) = 0 then y(i) = a(5)
                   else if x(i -1) = 1 & x(i) = 0 & x(i + 1) = 1 then y(i) = a(6)
                   else if x(i -1) = 1 & x(i) = 1 & x(i + 1) = 0 then y(i) = a(7)
                   else if x(i -1) = 1 & x(i) = 1 & x(i + 1) = 1 then y(i) = a(8)
                   }
         for i = 1 to 20 {
             print y(i)
             x(i) = y(i)
         carriage return
         t++
       }}
```

Excersize – 1-D Cell Automaton

Starting with one-dimensional 31 cells with only center being 1,

evolve them till 15th generation.



The rules to be tried are:

18, 50, 57, 73, 99, 110, 125, 145, 149

Also it might be interesting to try

106, 120, 134, 137, 141

Conway's LIFE Game

Rule of survival or death

1. Any alive cell will die if its alive neighbours are fewer than two (<= too lonly to lbe aive)
2. Any alive cell will remain alive if its alive neighbours are two or three
3. Any alive cell will die if its alive neighbours are more than three (<= too crowdy to be alive)
4. Any dead cell will become alive only if its alive neighbours are exactly three, otherwise remain dead

Conway's LIFE Game

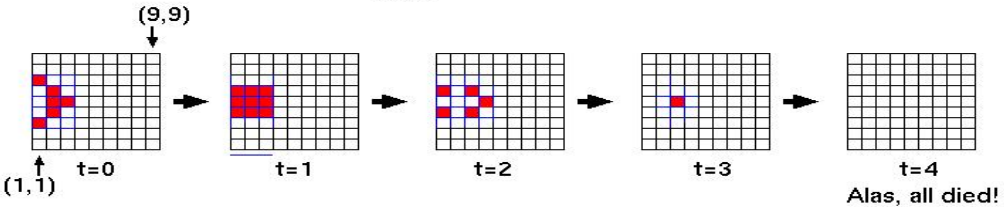
Rule

1. Any alive cell will die if its live neighbours are fewer than two (too lonly to live)
2. Any alive cell will remain live if its live neighbours are two or three
3. Any alive cell will die if its live neighbours are more than three (too crowdy to be alive)
4. Any dead cell will become alive only if its alive neighbours are exactly three, otherwise remain die

When t = 0

(1,3) is alive now and has one alive neighbour => die by rule-1
(1,4) is dead now and has three alive neighbours => becomes alive by rule-4
(1,5) is dead now and has three alive neighbours => becomes alive by rule-4
(1,6) is dead now and has three alive neighbours => becomes alive by rule-4
(1,7) is alive now and has one alive neighbour => die by rule.1
(1,8) is dead now and has one alive neighbour => remains dead by rule-4
(1,9) is dead now and has no alive neighbour => remain dead by rule-4 <= Note that our grid is troidal!
(2,3) is dead now and has two alive neighbours => remains dead by rule-4
(2,4) is alive now and has three alive neighbours => remains alive by rule-2
(2,5) is alive now and has three alive neighbours => remains alive by rule-2
(2,6) is alive now and has three alive neighbours => remains alive by rule-2
(2,7) is dead now and has two alive neighbours => remains dead by rule.4
(2,8) is dead now and has one alive neighbour => remains dead by rule-4
(2,9) is dead now and has no alive neighbour => remain dead by rule-4
⋮
(9,9) is dead now and has no alive neighbour => remain dead by rule-4

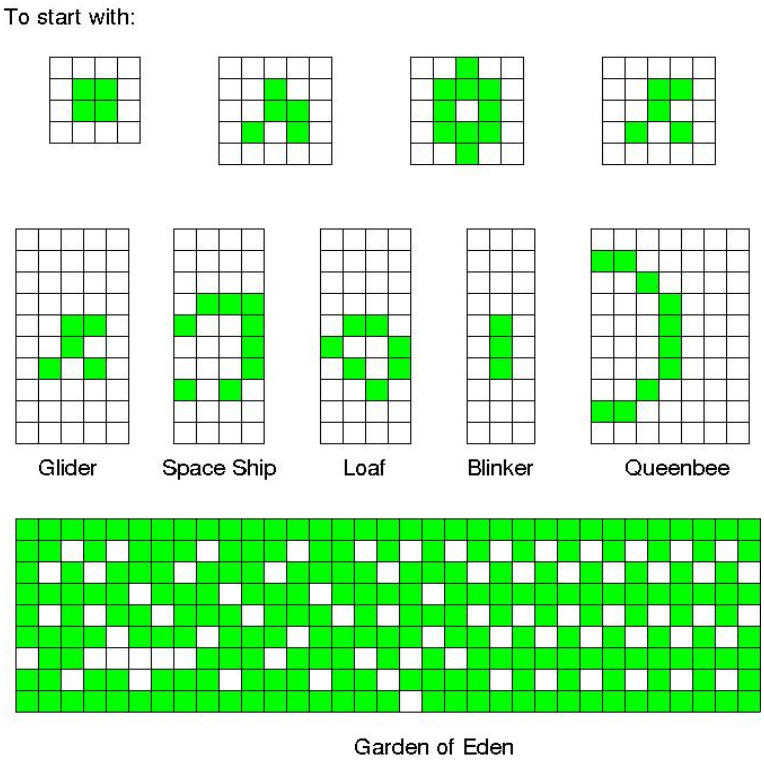
Thus



LIFE Game – How to code

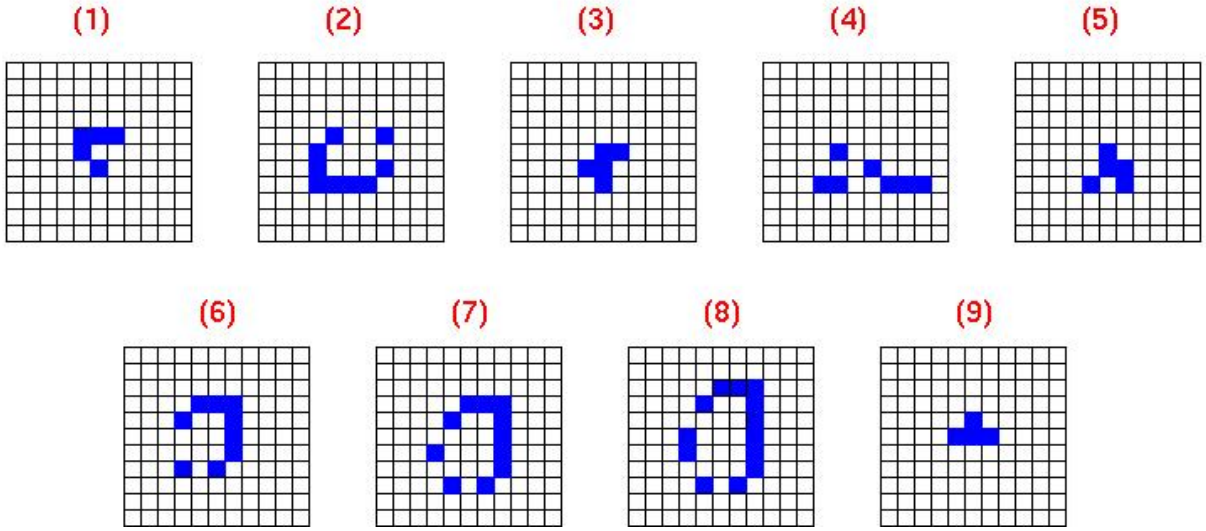
```
initialize ( )
   for i, j = 1 to n {x(i, j) = 0 or 1 according to the initial pattern}

main ( ) {
   while t <= t_final {
         for i, j = 1 to n {                                          % treatment when cell is at boarder
            if (i, j) = 1 then (i-1, j-1) = n
            if (i, j) = n then (i+1, j+1) = 1

         counter = the number of 1 of eight neighbor cells of x(i, j) % count how many neighbors are alive

         if x(i, j) = 1 & counter < 2      then y(i, j) = 0           % renew x(i, j) according to the 4 rules
         if x(i, j) = 1 & counter = 2 or 3 then y(i, j) = 1
         if x(i, j) = 1 & counter > 3      then y(i, j) = 0
         if x(i, j) = 0 & counter = 3      then y(i, j) = 1

         for i, j = 1 to n {
            if y(i, j) = 1 then color the cell blue otherwise empty   % renew the pattern
            x(i, j) = y(i, j) }                                       % renew all the cells for t = t+1

         t++
    }
```

Conway's LIFE Game



Excersize – The Game of Life

With the initial pattern being one of the below, evolve them from the center of the 50x50 grid from t=0 to t=60 applying the Conway's 4 rules of the Game of Life



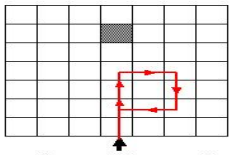
Examination - A model solution (1st page)

Name ( )

1. Following the transition table of the FSA shown below, draw the trace of machine that starts the center of botom of the grid-world shown with its state being A.

empty = 0

obstacle = 1



Start with state A

state	input	action	next state
A	0	Move 4 steps forward	B
	1	Turn to left and 1 step forward	A
B	0	Turn to right and 2 step forward	B
	1	Turn to left and 1 step forward	A

2. Create a transition table of the FSA that recognizez (accepts) strings on alphabet {a,b,c,d) with an alphabetical order, such as aacccdd.

state	input	new state	
A	a	A	
	b	B	
	c	C	
	d	D	
B	a		=> reject
	b	B	
	c	C	
	d	D	
C	a		=> reject
	b		=> reject
	c	C	
	d	D	
D	a		=> reject
	b		=> reject
	c		=> reject
	d	D	

If complete the input of the string and not so far rejected => accept

3. Using the Rule shown below, derive the string aaaabbbbcccc.

S => a b c
S => a S B c
b B => b b
c B => B c

S => a S B c => a (a S B c) B c => a a (a S B c) B c B c => a a a (a b c) B c B c B c => a a a a b (B c) c B c B c
=> a a a a (b b) c (B c) (B c) c => a a a a b b (B c)(B c)c c => a a a a b (b b) c B c c c => a a a a b b b B c c c c
=> a a a a b b (b b) c c c c => a a a a b b b b c c c c

Examination - A model solution (2nd page)

4. (i) What is Rule 100 in one-dimensional automaton?

As 4 + 32 + 64 = $(100)_{10}$ we know $(100)_{10}$ = $(0\ 1\ 1\ 0\ 0\ 1\ 0\ 0)_2$

So Rule 100 is

(0 0 0) = 0
(0 0 1) = 0
(0 1 0) = 1
(0 1 1) = 0
(1 0 0) = 0
(1 0 1) = 1
(1 1 0) = 1
(1 1 1) = 0

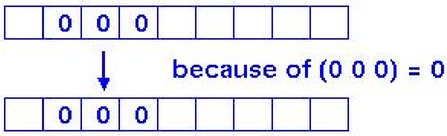
(ii) Then starting with (0,0,0,0,1,0,0,0,0), evolve this 8 cells by the Rule-100 until t =7. Show the result below by filling all cells with "0" or "1".

As all we must use in this case are (0 0 0) =0, (0 0 1) = 0, (0 1 0) = 1, (1 0 0) = 0

0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0

One of the easiest case!

Needless to say but, e.g.,



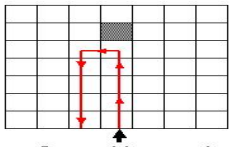
Additional examination - A model solution

Additional Examination Name ( )

1. Following the transition table of the FSA shown below, draw the trace of machine that starts the center of botom of the grid-world shown with its state being A.

empty = 0

obstacle = 1



Start with state A

state	input	action	next state
A	0	Move 1 steps forward	B
	1	Turn to left and 1 step forward	A
B	0	Turn to right and 1 step forward	B
	1	Turn to left and 1 step forward	A

2. Create a transition table of the FSA that recognizez (accepts) strings on alphabet {a,b,c,d) with an alphabetical order, such as aacccdd.

state	input	new state	
A	a	A	
	b	B	
	c	C	
B	a		=> reject
	b	B	
	c	C	
C	a		=> reject
	b		=> reject
	c	A	

If complete the input of the string and not so far rejected => accept

3. Using the Rule shown below, derive the string a a a b b b c c c.

S => a b c
S => a S B c
b B => b b
c B => B c

S => a S B c => a (a S B c) B c => a a (a b c) B c B c => a a a b (B c) c B c => a a a (b b) c c B c => a a a b b c (B c) c
=> a a a b b (B c) c c => a a a b (b b) c c c => a a a b b b c c c

4. Starting with (0,0,0,0,1,0,0,0,0) at t=0, evolve this 9 cells by the Rule-62 untill t = 7.

As $(62)_{10}$ = $(0\ 0\ 1\ 1\ 1\ 1\ 1\ 0)_2$ the Rule-62 is

(0 0 0) = 0
(0 0 1) = 1
(0 1 0) = 1
(0 1 1) = 1
(1 0 0) = 1
(1 0 1) = 1
(1 1 0) = 0
(1 1 1) = 0

So

0	0	0	0	1	0	0	0	0
0	0	0	1	1	1	0	0	0
0	0	1	1	0	0	1	0	0
0	1	1	0	1	1	1	1	0
1	1	0	1	1	0	0	0	1
0	0	1	1	0	1	0	1	1
1	1	1	0	1	1	1	1	0
1	0	0	1	1	0	0	0	1