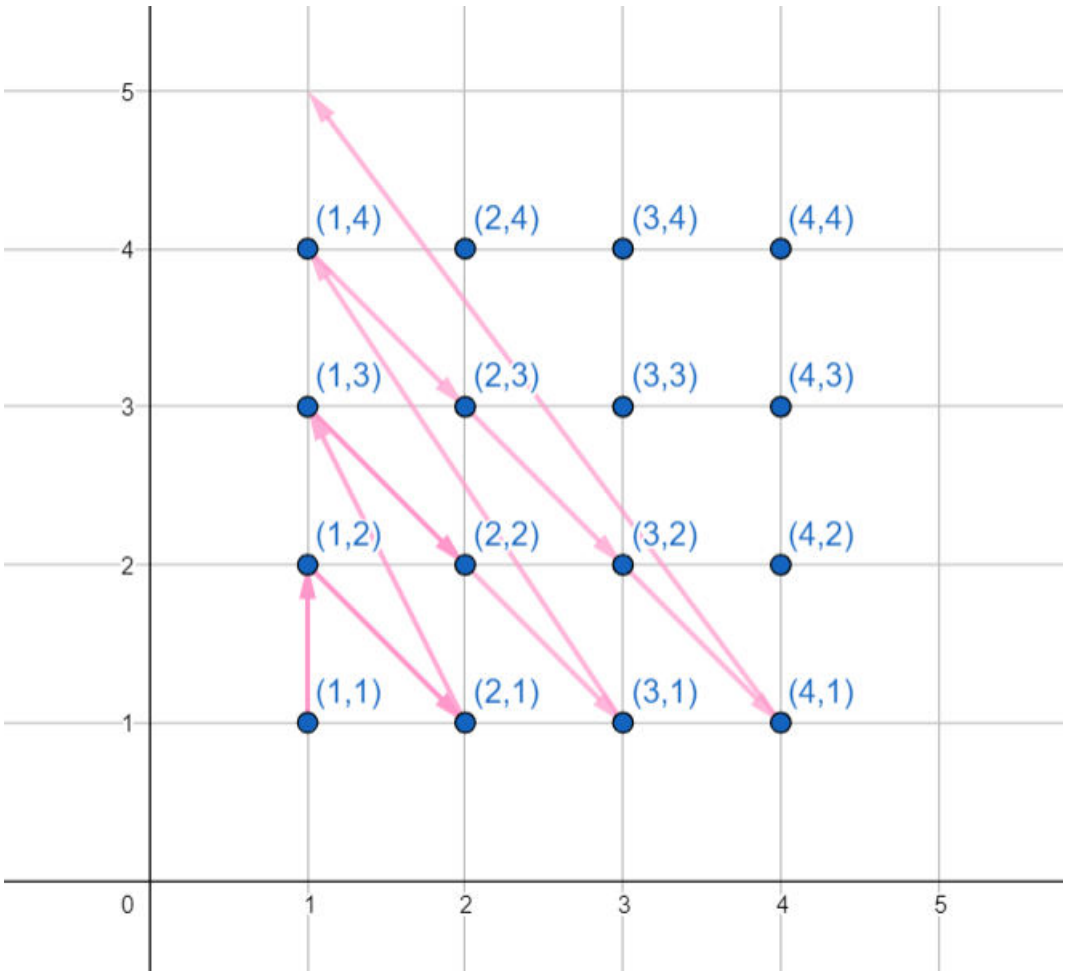
Bijecție între NxN și N:

Se deduce punând perechile de numere pe diagonale așa:



Funcția (de la NxN la N):

$$F: N \times N \to N \qquad F(x,y) = \frac{1}{2}(x+y)(x+y+1) + y$$

De la N la NxN (nu prea relevantă, dar am pus-o pentru curiozitatea de a înțelege cum poate o funcție de la (x,y) la z să meargă și de la z la (x,y)):

$$G: N \to N \times N, G(z) = (x,y)$$

$$w = \lfloor \frac{-1 + \sqrt{1 + 8z}}{2} \rfloor$$

$$t = \frac{w(w+1)}{2}$$

$$y = z - t$$

$$x = w - y$$

- metoda mai e cunoscută sub numele "diagonalizarea lui Cantor" (nu castor!!!)

- între N^N și N nu există bijecție

Automat finit = o mașină folosită pentru recunoașterea pattern-urilor, care poate fi într-o singură stare (dintr-un număr FINIT de stări) la un moment din timp ("finite state machine")

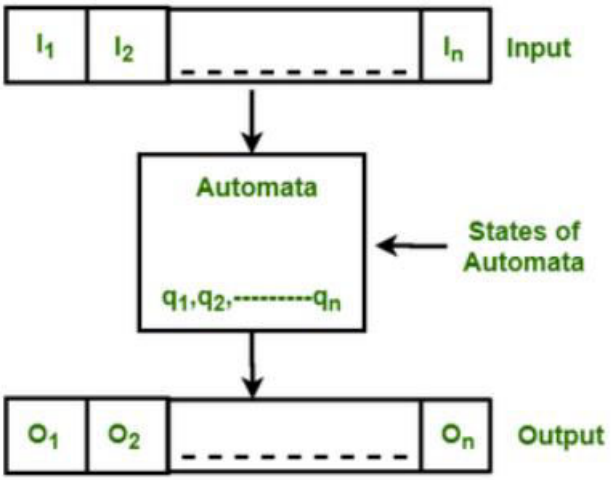
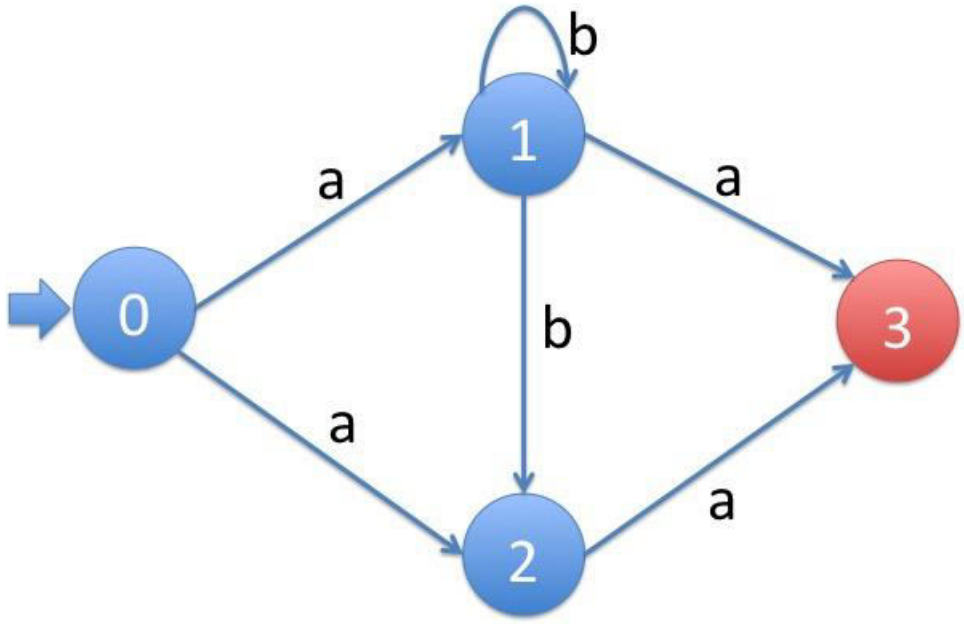
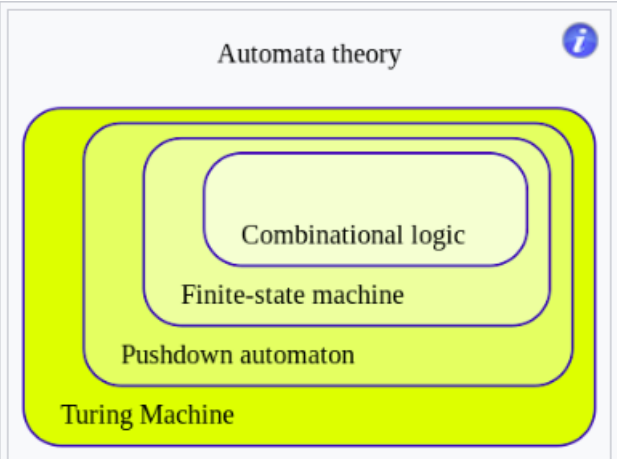


Figure: Features of Finite Automata



- este definit prin:

- Q: mulțime finită de stări (practic nodurile din desen - 0, 1, 2, 3)
- Σ: mulțime finită de simboluri (a, b)
- q_0: starea de început (unică - 0, marcată prin săgeată)
- F: mulțime de stări finale (pot fi mai multe - 3)
- δ: funcție de tranziție (ne zice cu ce simbol putem trece din q_x în q_y)



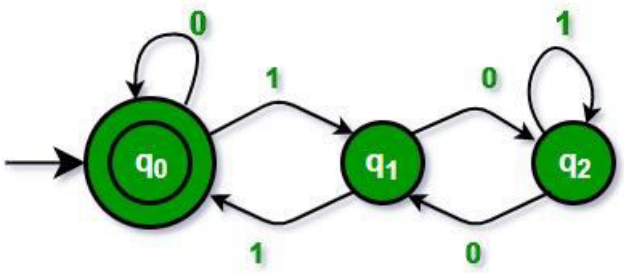
Classes of automata

Automate pentru a testa divizibilitatea:

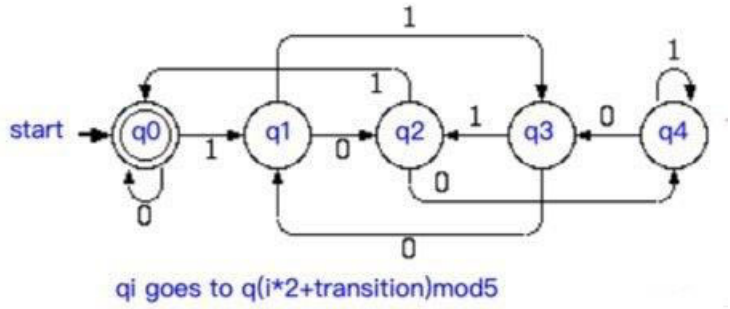
- nu prea importante

- au la bază scrierea numerelor în bază 2

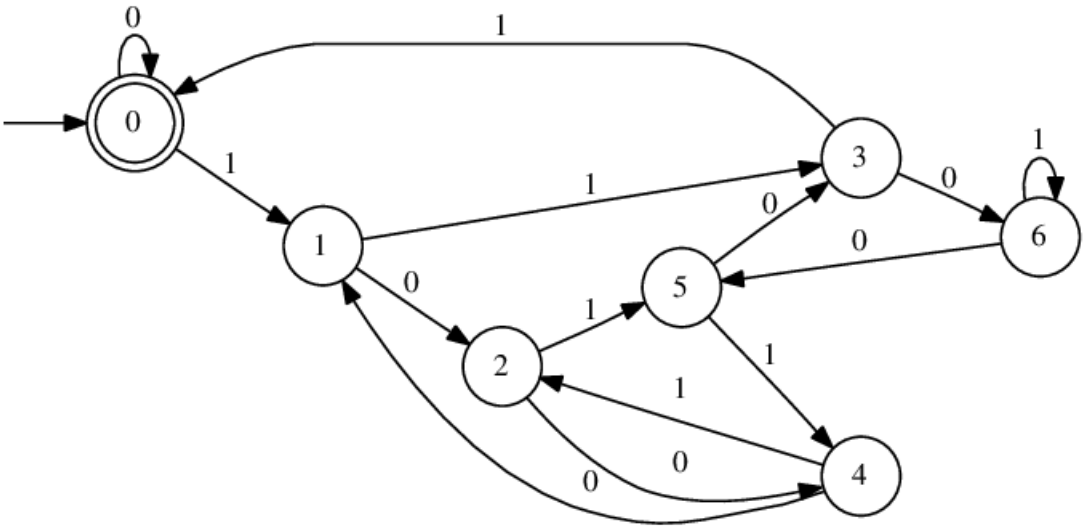
- divizibilitate cu 3:



- divizibilitate cu 5:



- divizibilitate cu 7:



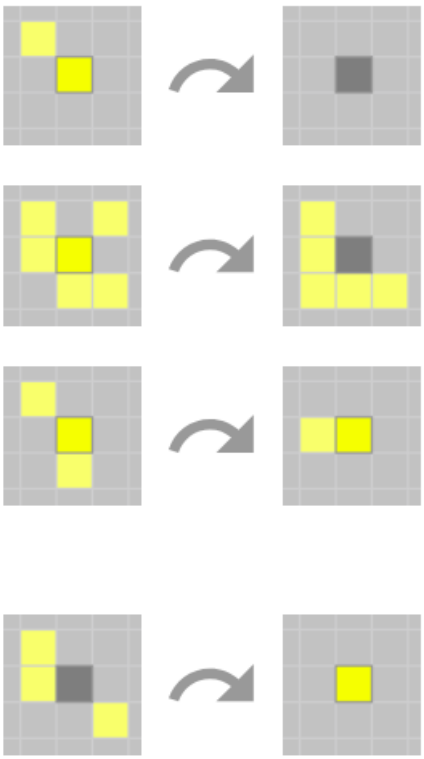
Automat celular - (mai teoretic) constă dintr-o rețea regulată de „celule” fiecare conținând o „stare” aleasă dintr-un set finit și care se poate schimba în timp. Starea unei celule la momentul t + 1 este o funcție a stării la momentul t a unui număr finit de celule numit „vecinătatea” sa. Cu fiecare nouă unitate de timp, aceleași reguli sunt aplicate simultan tuturor celulelor din rețea, producând o nouă „generație” de celule în funcție în întregime de generația anterioară.

(mai ușor de vizualizat) Ne putem imagina o matrice ("rețea regulată de celule") în care unele celule sunt colorate cu o culoare. La momentul următor de timp, fiecărei celule din matrice îi e aplicată o funcție care îi schimbă (sau nu) culoarea și obținem o configurație nouă (bazată pe cea precedentă). Acest lucru se repetă.

Game of Life - "joc" matematic care este un automat celular. Fiecare celulă are 2 stări, moartă sau vie. Fiind într-o "matrice", o celulă are 8 vecini (includem diagonalele). Regulile sunt:

- o celulă **vie** cu 1 sau niciun vecin viu moare (de singurătate)
- o celulă **vie** cu 4 sau mai mulți vecini vii moare (de suprapopulare)
- o celulă **vie** cu 2 sau 3 vecini vii ramâne vie
- o celulă **moartă** cu 3 vecini vii primește viață

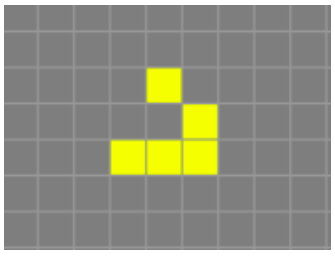
Exemple pentru fiecare regulă în ordinea în care sunt enumerate mai sus:



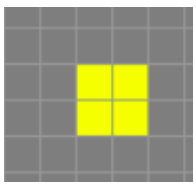
Jocul primește o configurație de celule la început și aplică regula la infinit. Unele configurații se termină/mor, adică toate celulele ajung moarte (și evident nu mai pot învia), pe când alte configurații ciclează (după un număr de pași se ajunge din nou la o configurație care a mai fost) sau merg la infinit.

Un simulator care ajută pentru vizualizare se găsește aici: https://playgameoflife.com/

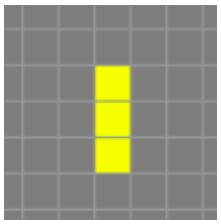
De exemplu, această configurație "moare" în aproximativ 50 de pași:



Această configurație nu se modifică niciodată:



Aceasta are doar 2 stări care se repetă infinit:



Matematica a studiat în detaliu diverse categorii de configurații. Numele de "game of life" provine din faptul că poate fi asemănat cu viața (cică).

Codificarea lui Wolfram = sistem numeric pentru regulile automatelor celulare uni-dimensionale. Practic se bazează pe observația că putem pune regulile unui automat celular într-un tabel (exemplu imediat).

Regula 110 este un automat celular cât de cât similar cu Game of Life:

Current pattern	111	110	101	100	011	010	001	000
New state for center cell	0	1	1	0	1	1	1	0

Aceasta este codificarea lui Wolfram pentru regulile automatului celular "Regula 110". Dacă ne uităm pe rândul de jos din tabel, putem construi reprezentarea binară 01101110, care corespunde numărului 110 în baza 10 (este pur și simplu fascinant!).

Regula 110 este Turing-complete! => orice calcul sau program poate fi simulat cu el.

Dacă tot suntem aici, **Turing-completeness**:

Un sistem de reguli pentru manipularea datelor (un limbaj de programare, un automat celular, un set de instrucțiuni, etc.) se numește Turing-complete dacă poate fi folosit pentru a simula o mașină Turing. O mașină Turing poate fi folosită pentru a calcula orice, așadar Turing-completeness încearcă să exprime puterea imensă a sistemului respectiv. (O să ne mai întâlnim cu noțiunea si la pavaje Wang!)