

# Predsluke 2D celičnih avtomatov

MAGISTRSKO DELO

Univerza v Ljubljani  
Fakulteta za računalništvo in informatiko

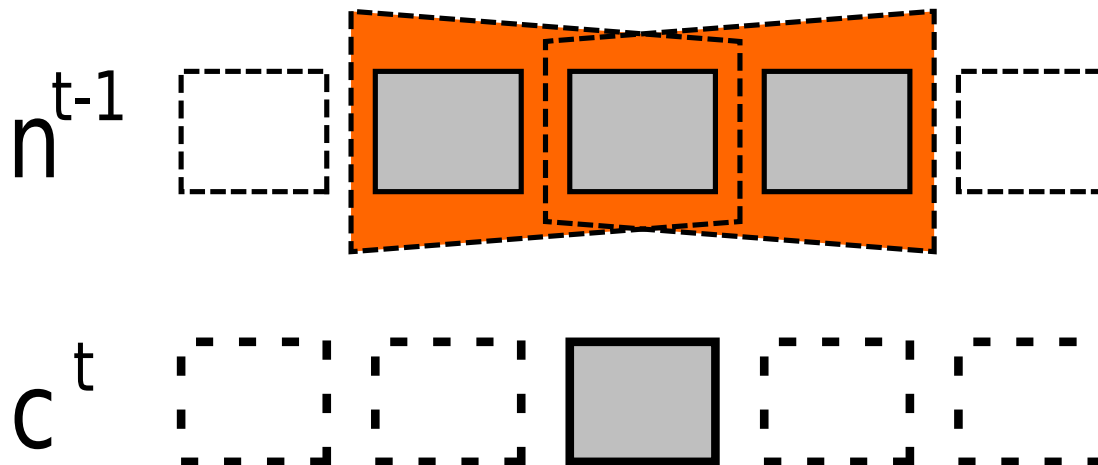
Mentor: prof. dr. Branko Šter  
Avtor: Izток Jeras

# Motivacija

- Umetno življenje
- Conwayeva igra življenja
  - Univerzalni konstruktor
  - Turingov stroj
- Evolucija
  - Langtonova zanka
  - Evoloop
- Holografski princip
  - Termodinamika, entropija in puščica časa

# 1D CA: pravilo, okolica in prekrivanje

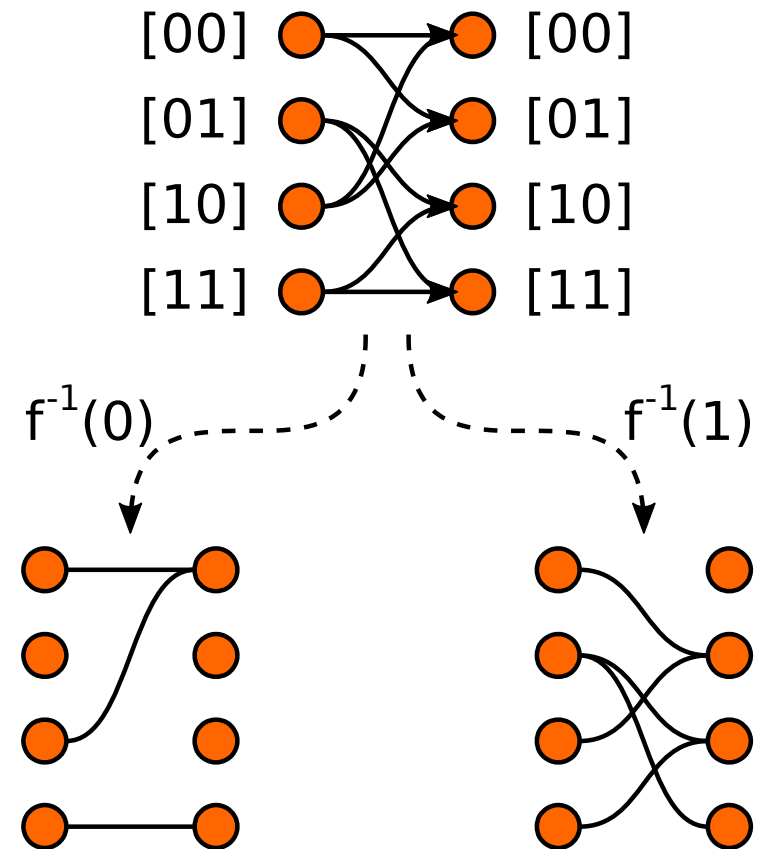
- 1D homogen časovno-prostorsko diskreten dinamični sistem
- Vrednost celice v sedanosti  $c_x^t$  je funkcija vrednosti njene okolice v preteklosti  $n_x^{t-1}$ 
$$c_x^t = f(n_x^{t-1})$$
- Predslike so stanja iz preteklosti, ki se preslikajo v dano sedanost, izpolnjujejo pogoja:
  - Vsaka okolica mora izpolnjevati tranzicijsko funkcijo
  - Sosednje okolice se morajo ujemati v prekrivanju



1D CA: nekaj splošnega o  
predslikah?

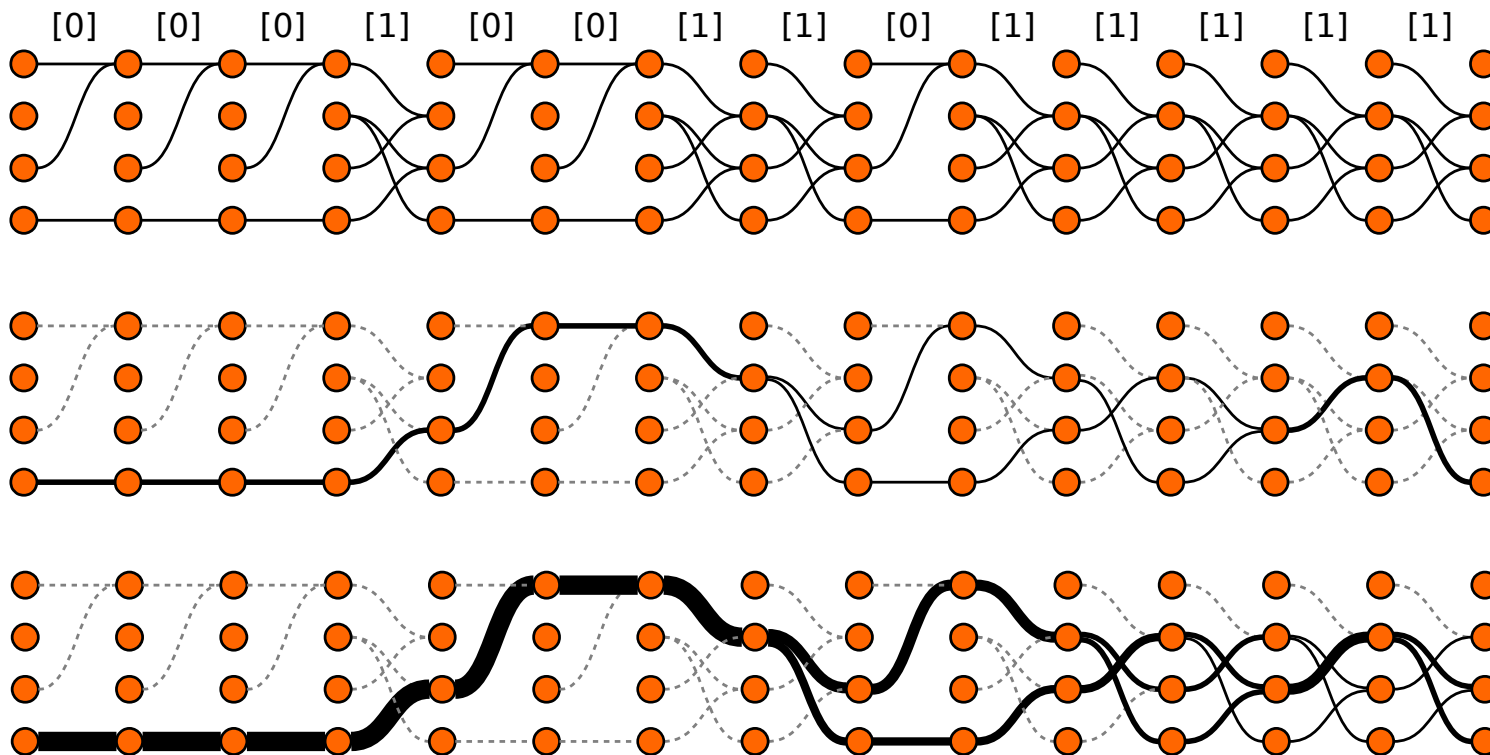
# 1D CA: De Bruijnov graf

- Prekrivanja so volišča
- Okolice so poti med vozlišči
- Graf se razdeli na grafe za posamezno stanje, vključuje samo okolice, ki pripeljejo v dano stanje
- De Bruijnov graf predstavlja inverzno tranzicijsko funkcijo  $f^{-1}(c)$
- Primeri kažejo pravilo 110



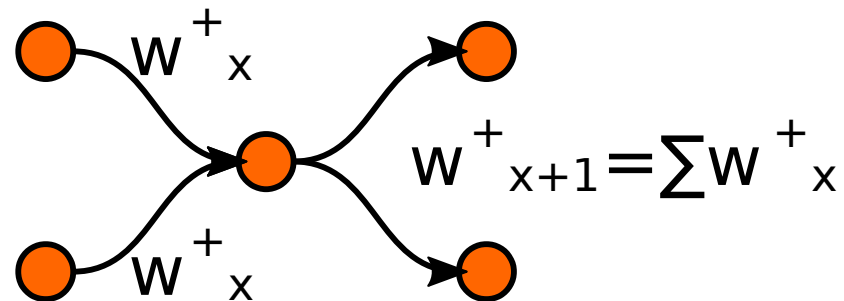
# 1D CA: mreža predslik

- Z veriženjem De Bruijnovih grafov tvorimo mrežo
- Iščemo vse zvezne poti, ki povezujejo levi in desni rob
- Ostale poti postopno izločujemo

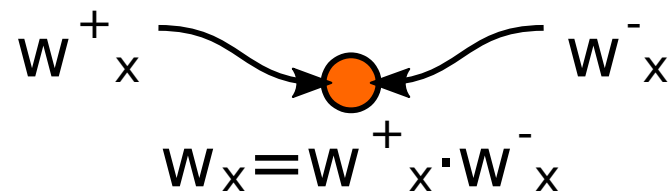


# 1D CA: uteži v mreži predslik

- V teoriji grafov je utež  $w$  enaka številu poti skozi vozlišče
- Izhodna utež  $(x+1)$  je enaka vsoti vhodnih uteži  $(x)$



- Uteži se računajo v dveh prehodih, naprej (+) in nazaj (-) glede na smer poti v grafu
- Skupna utež je zmnožek uteži obeh prehodov (+ in -)



# 1D CA: matrične enačbe

- Za De Brujnov graf vsakega stanja lahko zapišemo matriko
- Podobno veriženju grafov, lahko množimo matrike
- Vsota elementov matrike je število predslik
- Vsota elementov na diagonali je število predslik za ciklični rob
- Vmesne rezultate lahko uporabimo za izračun uteži v mreži

$$D(0) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad D(1) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$D(\alpha) = \sum_{x=0}^{N-1} D(c_x) = D(c_0) D(c_0) \dots D(c_0)$$



# 1D CA: algoritem za izpis predslik

- Pomnilnik alociramo z vnaprej znanim številom predslik (štetje iz desne proti levi)
- Z utežmi za  $x=0$  inicializiramo vrednosti
- Za vsako okolico na položaju  $x$  uteži določajo koliko poti nadaljuje z katerim stanjem celice  $x+1$

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

# 1D CA: graf podmnožic in regularni jezik GoE stanj

- Binarna preslikava iz vektorja uteži  $\mathbf{b}_x$  trenutnih vozlišč v naslednji vektor uteži  $\mathbf{b}_{x+1}$

$$\mathbf{b}_{x+1} = \mathbf{b}_x D(c_x)$$

- Nabor preslikav definira končni avtomat
- Če je začetno stanje polno in končno stanje prazno, avtomat sprejme GoE besede, in definira regularni jezik GoE stanj

Primer za pravilo 110:

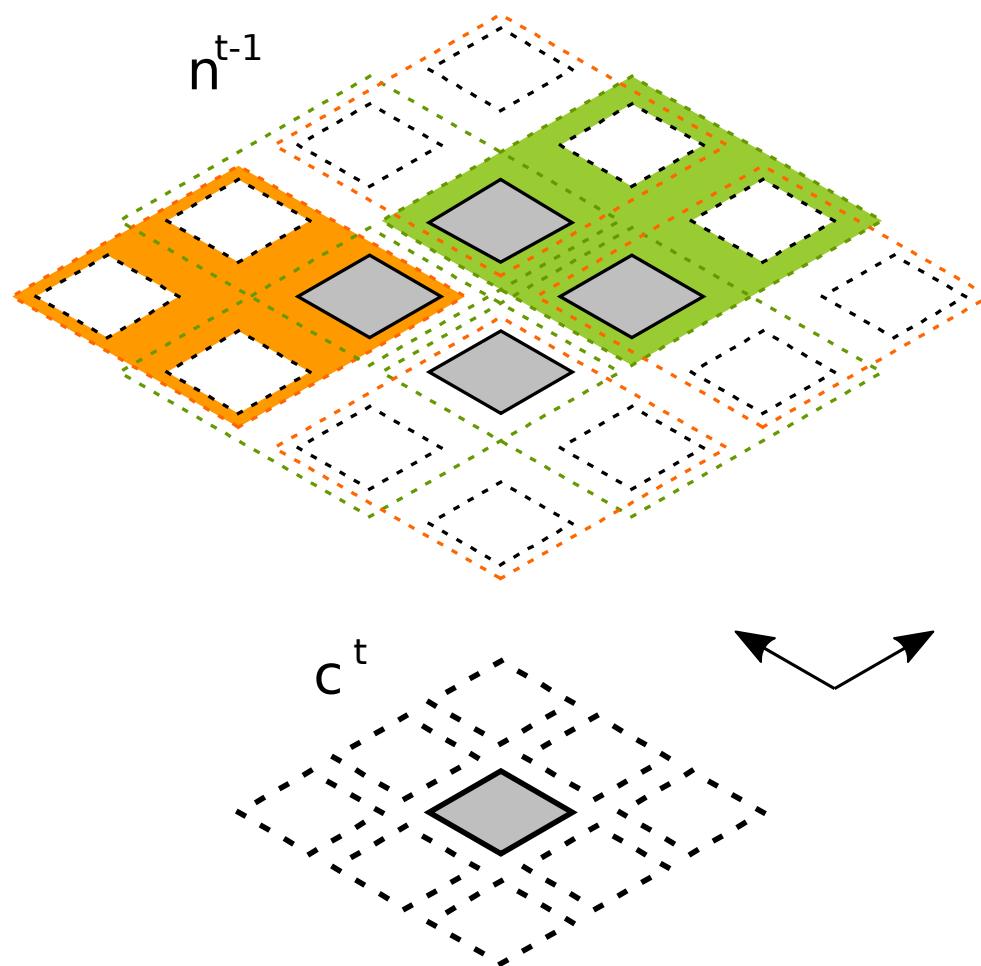
```
aut = Automaton(
    (('0000', '0000', 0), ('0000', '0000', 1),
     ('0001', '0001', 0), ('0001', '0010', 1),
     ('0010', '1000', 0), ('0010', '0100', 1),
     ('0011', '1001', 0), ('0011', '0110', 1),
     ('0100', '0000', 0), ('0100', '0011', 1),
     ('0101', '0001', 0), ('0101', '0011', 1),
     ('0110', '1000', 0), ('0110', '0111', 1),
     ('0111', '1001', 0), ('0111', '0111', 1),
     ('1000', '1000', 0), ('1000', '0100', 1),
     ('1001', '1001', 0), ('1001', '0110', 1),
     ('1010', '1000', 0), ('1010', '0100', 1),
     ('1011', '1001', 0), ('1011', '0110', 1),
     ('1100', '1000', 0), ('1100', '0111', 1),
     ('1101', '1001', 0), ('1101', '0111', 1),
     ('1110', '1000', 0), ('1110', '0111', 1),
     ('1111', '1001', 0), ('1111', '0111', 1)),
    initial_states = ['1111'],
    final_states   = ['0000'],
    input_alphabet = [0,1])
aut_min = aut.minimization()
list(aut_min.language(5))

[[0, 1, 0, 1, 0]]
```

# Atraktor in korito

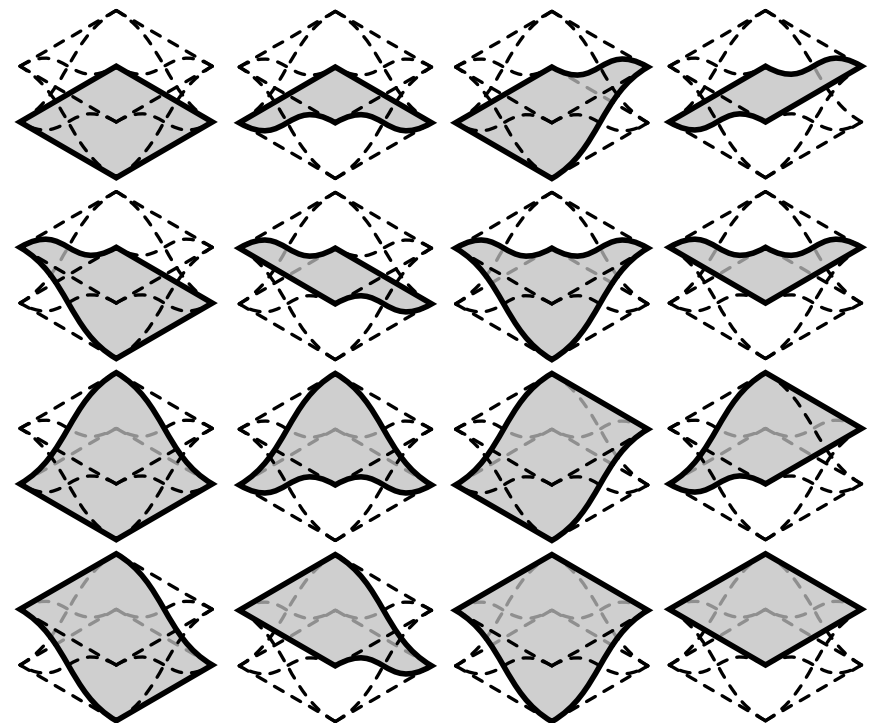
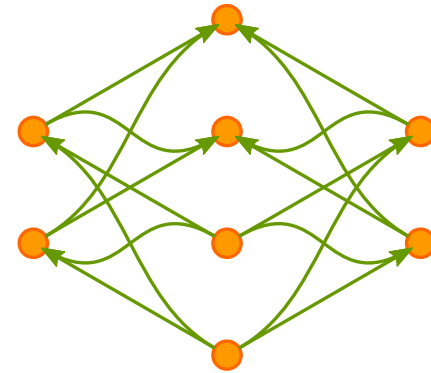
# 2D CA: okolica in prekrivanje

- Za primer je uporabljena okolica iz 4 celic imenovana **quad**
- Sosednji okolici v smeri dimenzij X ali Y (**zelená**) se prekrivajo za dve celici
- Sosednji okolice v diagonalni smeri (**oranžna**) se prekrivajo za eno celico
- Za tvorjenje mreže predslik so potrebni drugačni elementi kakor pri 1D problemu



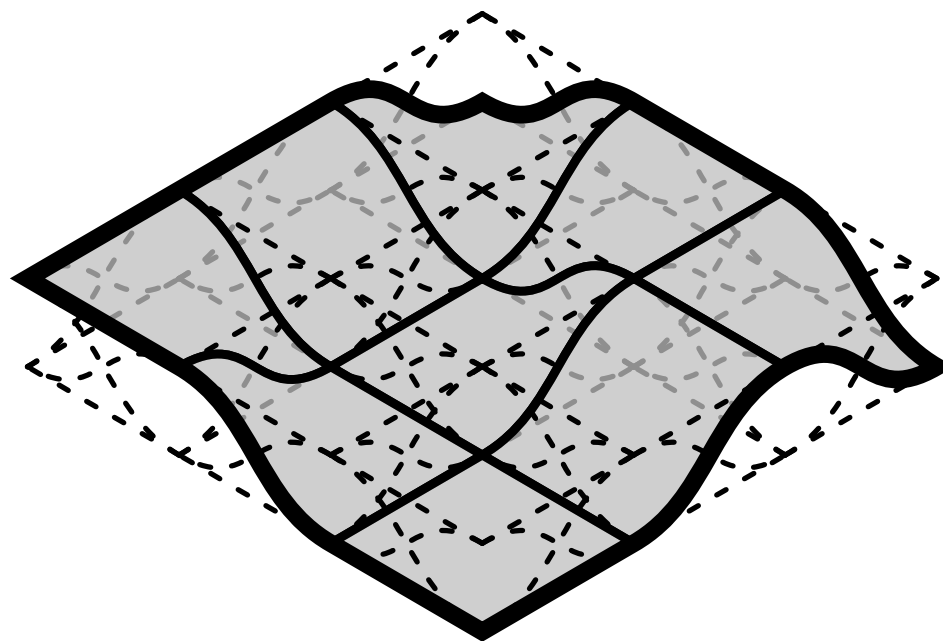
# 2D CA: De Brujnov graf

- De Bruijnov graf dobi dodatno dimenzijo v primerjavi z 1D CA
- Prekrivanje v smeri dimenzij X ali Y (**zelena**) je predstavljeno s potmi med vozlišči
- Prekrivanje v diagonalni smeri (**oranžna**) je predstavljeno z vozlišči
- Okolice (siva) so predstavljene s ploskvami, katerih vogali so vozlišča in robovi poti med vozlišči, prikazane so vse možne okolice/ploskve



# 2D CA: mreža predslik

- Z tlakovanjem De Bruijnovih grafov tvorimo mrežo
- Iščemo vse zvezne ploskve, ki pokrivajo celotno površino mreže
- Ostale ploskve postopno izločujemo
- Poudarjena ploskev prikazuje predsliko z vrednostjo



1110

1001

0011

0010

2D CA: štetje predslik znotraj vrstice

2D CA: štetje predslik po vrsticah



2D CA: izpis predslik

2D CA: stanja Garden of Eden

# Viri in programska oprema