

UNIVERZA V LJUBLJANI  
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Iztok Jeras

**Predslike 2D celičnih avtomatov**

MAGISTRSKO DELO  
NA UNIVERZITETNEM ŠTUDIJU

Ljubljana, 2016



UNIVERZA V LJUBLJANI  
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Iztok Jeras

**Predslike 2D celičnih avtomatov**

MAGISTRSKO DELO  
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Branko Šter

Ljubljana, 2016



To magistrsko delo je ponujeno pod licenco *Creative Commons Attribution-ShareAlike 4.0 International*<sup>1</sup> ali v slovenščini *priznanje avtorstva in deljenje pod enakimi pogoji 4.0 mednarodna*<sup>2</sup>. Po želji se lahko uporabnik poslužuje novejših različice. To pomeni, da se tako besedilo, slike, grafi in druge sestavine dela kot tudi rezultati magistrskega dela lahko prosto distribuirajo, reproducirajo, uporabljajo, dajejo v najem, priobčujejo javnosti in predelujejo, pod pogojem, da se jasno in vidno navede avtorja in naslov tega dela in da se v primeru spremembe, preoblikovanja ali uporabe tega dela v svojem delu, lahko distribuira predelava le pod licenco, ki je enaka tej. Uporabljena je mednarodna licenca, čeprav obstaja verzija prilagojena za slovenski pravni red, to pa zato, ker slovenska verzija ni vzdrževana. Podrobnosti licence so dostopne na spletni strani <http://creativecommons.org> ali na Inštitutu za intelektualno lastnino, Streliška 1, 1000 Ljubljana.



Izvorna koda programske opreme, razvite za potrebe magistrskega dela, je ponujena pod licenco *Unlicense*<sup>3</sup>. To pomeni, da se lahko prosto uporablja, distribuira in/ali predeluje, brez kakšnih koli obveznosti. Avtor se odpoveduje vsem pravicam in tako omogoča uporabnikom izvirne kode, da se izognejo preverjanju pravnih obveznosti.

*Besedilo je oblikovano z urejevalnikom besedil  $\text{\LaTeX}$ .  
Slike in grafi so narisani s pomočjo programa za vektorske ilustracije Inkscape<sup>4</sup>.*

---

<sup>1</sup><http://creativecommons.org/licenses/by-sa/4.0/>

<sup>2</sup><http://www.ipi.si/sl/creative-commons-cc/o-uporabi-licence>

<sup>3</sup><http://unlicense.org/>

<sup>4</sup><https://inkscape.org>





Številka: 159-MAG-RI/2016  
Datum: 06. 04. 2016

Iztok JERAS, univ. dipl. inž. el.

**Ljubljana**

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **Pred slike 2D celičnih avtomatov**

**Preimages of 2D cellular automata**

Tematika naloge:

Raziščite problem iskanja predslik 2D celičnih avtomatov s pomočjo De Bruijn-ovih diagramov. Pri tem se opirajte na znane rešitve za 1D problem. Poiščite algoritem za določitev obstoja predslik in algoritem za njihovo preštevanje ter izpis.

Algoritem naj bo matematično formuliran z enačbami in grafično predstavitevijo ter implementiran s programsko opremo. S stališča procesne kompleksnosti primerjajte novo razviti algoritem z obstoječimi algoritmi, ki se uporabljajo za iskanje stanj tipa 'Garden of Eden' v celičnem avtomatu 'Game of Life'.

Mentor:

prof dr. Branko Šter



Dekan:

prof. dr. Nikolaj Zimic





# IZJAVA O AVTORSTVU

## magistrskega dela

Spodaj podpisani **Iztok Jeras**, z vpisno številko **63030393**, sem avtor magistrskega dela z naslovom:

**Pred slike 2D celičnih avtomatov**

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal samostojno pod vodstvom mentorja **prof. dr. Branka Štera**,
- so elektronska oblika magistrskega dela, naslova (slov., angl.), povzetka (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela
- in soglašam z javno objavo elektronske oblike magistrskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 30. avgust 2016

Podpis avtorja:



# Zahvala

Zahvaljujem se prof. dr. Andreju Dobnikarju za pomoč pri pisanju člankov o algoritmih za štetje in izpis predslik pri enodimenzijskih celičnih avtomatih.



# Kazalo

<b>Povzetek</b>	<b>1</b>
<b>Abstract</b>	<b>3</b>
<b>1 Uvod</b>	<b>5</b>
1.1 Motivacija . . . . .	5
1.1.1 Celični avtomati kot model vesolja . . . . .	5
1.1.2 Informacijska dinamika . . . . .	5
1.1.3 Atraktorjevo korito . . . . .	7
1.2 Algoritmi za štetje in izpis preslik CA . . . . .	8
1.2.1 Implementacija algoritma . . . . .	10
1.2.2 Primeri in ilustracije . . . . .	10
<b>2 Definicija CA</b>	<b>11</b>
2.1 Definicija 1D CA . . . . .	11
2.1.1 Okolica . . . . .	12
2.1.2 Prekrivanje okolic . . . . .	12
2.1.3 Lokalna tranzicijska funkcija in pravilo . . . . .	13
2.1.4 Robni pogoji . . . . .	14
2.1.5 Globalna tranzicijska funkcija . . . . .	14
2.1.6 Preslike . . . . .	14

2.2	Definicija 2D CA . . . . .	15
2.2.1	Okolica . . . . .	15
2.2.2	Prekrivanje okolic . . . . .	16
2.2.3	Lokalna tranzicijska funkcija in pravilo . . . . .	18
2.2.4	Robni pogoji . . . . .	19
2.2.5	Globalna tranzicijska funkcija . . . . .	19
2.2.6	Predsluke . . . . .	20
2.2.7	Lokalna in globalna tranzicijska funkcija . . . . .	20
<b>3</b>	<b>Konstrukcija mreže predslik</b>	<b>21</b>
3.1	De Bruijnov diagram . . . . .	21
3.2	Mreža predslik 1D CA . . . . .	21
3.3	Mreža predslik 2D CA . . . . .	22
3.4	Mreža . . . . .	23
3.5	Robni pogoji . . . . .	24
<b>4</b>	<b>Algoritem za štetje in izpis predslik</b>	<b>26</b>
4.1	Procesiranje v eni dimenziji . . . . .	26
4.2	Procesiranje v drugi dimenziji . . . . .	28
4.3	Izpis predslik . . . . .	28
4.4	Nezmožnost procesiranja z linearno zahtevnostjo . . . . .	29
<b>5</b>	<b>Primerjava z znanimi algoritmi</b>	<b>31</b>
5.1	Iskanje stanj GoE za GoL . . . . .	31
5.2	Iskanje predslik GoL na splošno . . . . .	32
5.3	Polni algoritmi za štetje in izpis predslik . . . . .	33

<b>6</b>	<b>Sklepne ugotovitve</b>	<b>34</b>
6.1	Analiza 2D CA s pomočjo končnih avtomatov . . . . .	34
6.2	Izboljšave podanega algoritma . . . . .	35
6.3	Dokazovanje . . . . .	35
<b>A</b>	<b>De Bruijnov diagram za GoL</b>	<b>37</b>
<b>B</b>	<b>Primer delovanja algoritma za CA z okolico quad</b>	<b>38</b>
<b>C</b>	<b>Opis algoritma Dona Woodsa</b>	<b>42</b>
	<b>Seznam slik</b>	<b>44</b>
	<b>Literatura</b>	<b>45</b>

# Seznam uporabljenih kratic in simbolov

<b>1D</b> .....	enodimenzionalen
<b>2D</b> .....	dvodimenzionalen
<b>3D</b> .....	tridimenzionalen
<b>CA</b> .....	cellular automata (celični avtomat)
<b>GoL</b> .....	Conway's Game of Life (Conwayeva igra življenja)
<b>GoE</b> .....	Garden of Eden (rajski vrt)
<b>trid</b> .....	okolica CA, sestavljena iz treh celic na heksagonalni mreži
<b>quad</b> .....	okolica CA, sestavljena iz štirih celic na kvadratni mreži
<b>DFS</b> .....	depth first search
<b>SAT</b> .....	boolean satisfiability problem
$C$ .....	poljubna konstanta
$S$ .....	nabor stanj celice
$ S $ .....	število možnih stanj celice
$c$ .....	stanje posamezne celice (celoštevilska vrednost)
$c_{x,y}$ .....	vrednost celice na koordinatah $(x, y)$ znotraj 2D polja celic
$c^t$ .....	vrednost celice v sedanjosti
$c^{t+1}$ .....	vrednost celice v prihodnosti (en korak)
$N_x$ .....	velikost pravokotnega 2D polja celic v dimenziji X
$N_y$ .....	velikost pravokotnega 2D polja celic v dimenziji Y
$N$ .....	število celic v 1D ali 2D polju
$M_x$ .....	velikost pravokotne 2D okolice celice v dimenziji X
$M_y$ .....	velikost pravokotne 2D okolice celice v dimenziji Y
$M$ .....	število celic v 1D ali 2D okolici



$n$ .....	stanje okolice posamezne celice (celoštevilska vrednost)
$n_{x,y}$ .....	vrednost okolice celice na koordinatah $(x, y)$ znotraj 2D polja celic
$n^{t-1}$ .....	vrednost okolice celice v preteklosti (en korak)
$n^t$ .....	vrednost okolice celice v sedanjosti
$f$ .....	tranzicijska funkcija, ki definira časovno evolucijo avtomata
$f^{-1}$ .....	obratna tranzicijska funkcija
$o_x$ .....	prekrivanje okolic v dimenziji X
$o_y$ .....	prekrivanje okolic v dimenziji Y
$o_{xy}$ .....	prekrivanje okolic v diagonalni smeri



# Povzetek

Medtem ko je računanje predslik 1D celičnih avtomatov dobro raziskan in podrobno dokumentiran problem, je to področje pri 2D celičnih avtomatih manj raziskano. Za 1D problem poznamo algoritme za štetje in izpis predslik s procesno zahtevnostjo linearno odvisno od velikosti problema. Možno je tudi določiti, ali je 1D celični avtomat reverzibilen in kakšen je regularen jezik vseh stanj brez predslik. Pri 2D problemu sicer poznamo nekaj algoritmov za iskanje predslik, vendar so slabo teoretično raziskani. Vemo tudi, da je problem reverzibilnosti 2D celičnih avtomatov na splošno neodločljiv.

Mreža predslik, ki sem jo razvil za potrebe analize predslik 1D celičnih avtomatov, se je izkazala za praktično orodje pri razlagi algoritmov in pri dokazovanju njihove pravilnosti. Tukaj opišem, kako se mrežo predslik tvori za 2D celične avtomate. Če je bila ta mreža za 1D problem navaden graf, je za 2D problem razširjena v tretjo dimenzijo. Predslike so namesto poti v grafu ploskve na mreži. Robni pogoj se spremeni iz uteži za vozlišča, ki zaključujejo pot v 1D problemu, v uteži za sklenjeno pot okoli ploskve predslike v 2D problemu.

Pri razvoju algoritma se je izkazalo, da štetja predslik 2D celičnega avtomata ni mogoče izvesti v linearni odvisnosti od velikosti problema. Zahtevnost podanega algoritma narašča eksponentno z velikostjo problema v eni dimenziji. Podani algoritem se ne razlikuje dosti od obstoječih. Celični avtomat razdeli na vrstice in išče predslike za posamezno vrstico od prve do zadnje. Vrstične rezultate nato združi v rešitev za celoten 2D problem. Podobno kot pri algoritmu za 1D probleme, je analiza razdeljena v dva prehoda. V prvem prehodu po vrsticah algoritem prešteje predslike, v drugem opcijskem prehodu pa predslike izpiše. Drugi prehod poteka v obratni smeri kakor prvi prehod, tudi procesiranje posamezne vrstice poteka iz drugega zornega kota.

Glavni napredek algoritma vidim v uporabi učinkovitih rešitev za 1D problem pri analizi vrstic ter v uporabi progresivnega kodiranja vmesnih rezultatov, kar lahko zmanjša porabo pomnilnika.

## Ključne besede:

celični avtomati, predslike, predhodniki, računska zahtevnost, reverzibilnost, rajski vrt, Conwayeva igra življenja, trid, quad



# Abstract

While computing preimages of 1D cellular automata is a well researched and documented problem, for 2D cellular automata there is less research available. For the 1D problem we know algorithms for counting and listing preimages where computational complexity is a linear function of the size of the problem. It is possible to determine whether a 1D cellular automaton is reversible, and what is the Garden of Eden sequence regular language. For the 2D problem we know a few algorithms, but they are poorly theoretically researched. We also know that the reversibility problem is in general undecidable for 2D cellular automata.

The preimage network, first developed for 1D cellular automata, was proved to be a useful tool for explaining algorithms and for constructing proofs. Here I explain how to construct the preimage network for 2D cellular automata. While for the 1D problem this network is a normal graph, for 2D it was extended into the third dimension. Preimages are transformed from paths in the graph in 1D into surfaces on the network in 2D. Edge conditions are transformed from weights for vertices ending a path in the 1D problem into weights for the closed path around a preimage surface in the 2D problem.

While developing the algorithm, it proved impossible to count preimages of 2D cellular automata with processing requirements growing linearly with problem size. Instead, processing requirements grow exponentially with the size in one of the dimensions. The described algorithm does not differ much from the existing ones. The cellular automaton is split into rows, the preimage list is first determined for each row from the first to the last. The row results are then combined into the result for the whole 2D problem. In a similar fashion to the 1D approach, the algorithm splits into two passes. In the first pass preimages are counted, in the second optional pass preimages are listed. The second pass is performed in the opposite direction, while rows are also observed from the opposite side.

I see the main advantage of the described algorithm in using existing solutions for row processing. Solutions proved to be effective in solving the 1D problem. Using progressive encoding of intermediate solutions also enables reducing memory consumption.

## Key words:

cellular automata, preimages, predecessors, ataviser, computational complexity, reversibility, Garden of Eden, Conway's Game of Life, trid, quad



# Poglavje 1

## Uvod

### 1.1 Motivacija

#### 1.1.1 Celični avtomati kot model vesolja

Ker lahko vsak univerzalen sistem modelira vsak drug univerzalen sistem, lahko predpostavimo, da lahko z univerzalnimi celičnimi avtomati (CA) modeliramo vesolje. Samo modeliranje vesolja je še izven našega dosega, poizkuša pa se vsaj približati teorijo CA in teoretično fiziko. S strani informacijske teorije in termodinamike je predvsem zanimiv model gravitacije kakor entropijske sile (Entropic gravity [37]), ki predpostavlja, da je 3D vesolje projekcija procesov, ki se odvijajo na 2D ploskvi. Z druge strani pa CA tudi omogočajo opazovanje abstraktnega kopiranja informacij (replikacija) in evolucije [34]. Oba sta pomembna informacijska pojava v našem vesolju.

#### 1.1.2 Informacijska dinamika

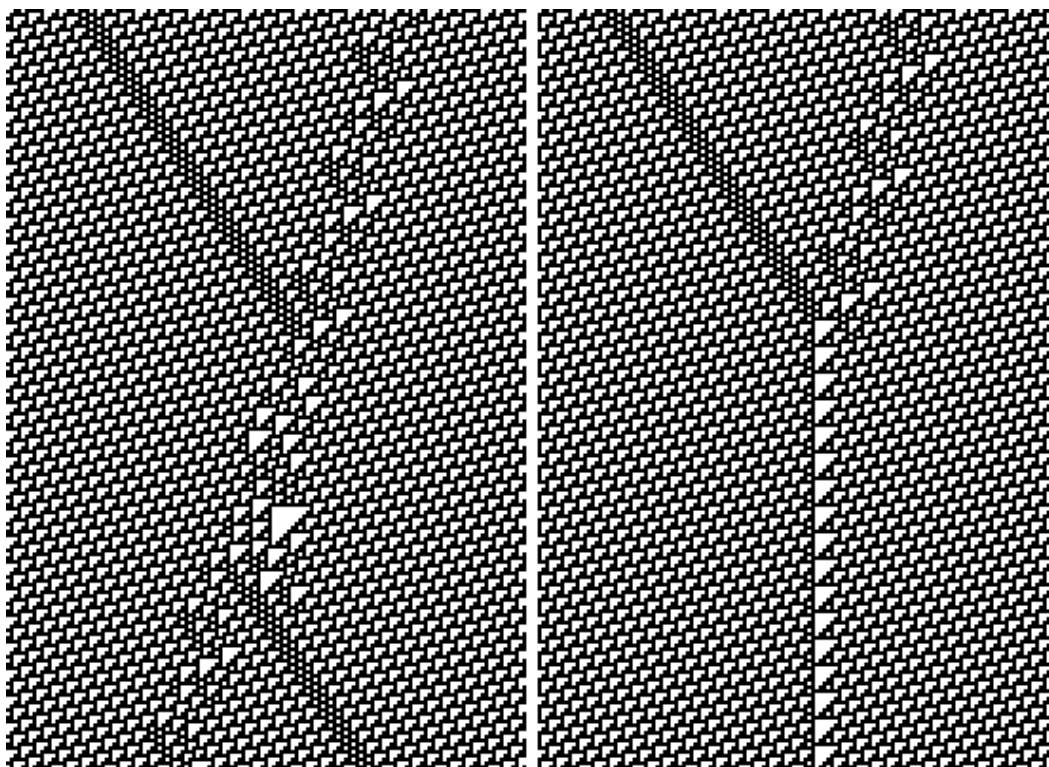
Informacijsko dinamiko CA se najpogosteje opisuje samo kakor reverzibilno ali ireverzibilno, obstaja tudi nekaj člankov, ki opazujejo entropijo sistema. Pri reverzibilnem avtomatu se vsa informacija ohranja, vsako stanje ima natanko eno prihodnost in eno preteklost. Za reverzibilen celični avtomat lahko tudi definiramo pravilo oziroma funkcijo, s katero preprosto procesiramo avtomat v preteklost namesto prihodnost. Za reverzibilne CA je torej računanje predslik (preteklosti) trivialno in ne potrebuje tukaj opisanega algoritma. Za ireverzibilne CA tako reverzno pravilo ne obstaja. Trenutno stanje ima lahko nobeno, eno, ali več predslik. Opisani algoritem je namenjen štetju in izpisu predslik za dano sedanje stanje CA. Za take avtomate pravimo, da s časom izgubljajo informacijo, ker ne moremo enolično določiti preteklega stanja.

Pogosto je tudi opazovanje dinamike delcev pri GoL (Game of Life [5] ali slovensko igra življenja) (slika 1.1) in elementarnem pravilu 110 [9] (slika 1.2). Pištole, trki delcev in podobni konstrukti obstajajo tako v GoL kakor pri elementarnem pravilu 110 in na splošno

pri vsakem univerzalnem celičnem avtomatu. Pištola (slika 1.1) oddaja delce, izvor in oddani delci skupaj rastejo v neskončnost. S tem se informacija potrebna za opis sistema povečuje. Slika 1.2 prikazuje dve različni interakciji med dvema delcema. V levem primeru se delca srečata, si nekaj časa delita skupen prostor in nato nadaljujejeta pot vsak zase. V desnem primeru se delca srečata in po trku nastane nov delec. Veliko poklicnih in amaterskih raziskovalcev proučuje delce v GoL in njihovo dinamiko. S pomočjo osnovnih gradnikov je mogoče skonstruirati kompleksnejše sisteme, med katerimi so najzanimivejši Turingov stroj [33] in univerzalni konstruktor [15].



Slika 1.1: Gosperjeva pištola z nekaj izstreljenimi drsalci.



Slika 1.2: Trk para delcev v pravilu 110. Razlika v začetnem stanju med slikama je v razdalji (fazi) med delcema.

Celične Avtomate sta si zamislila matematika John von Neumann in Stanislaw Ulam, da bi lahko oblikovala teorijo univerzalnega konstruktorja [10], ki je sposoben iz zapisa

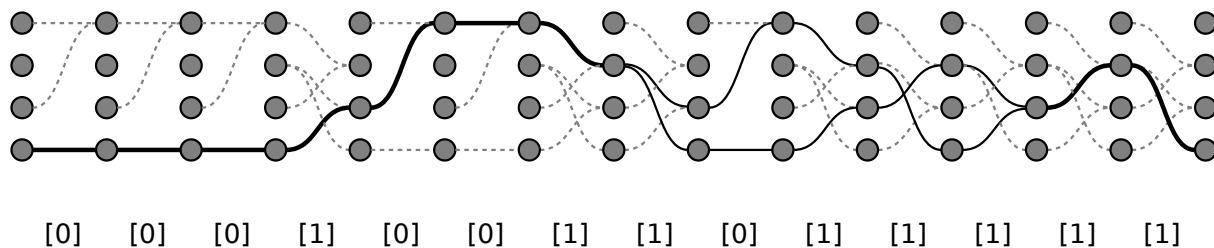


na traku narediti kopijo samega sebe. Christopher Langton je vzel del univerzalnega konstruktorja in ga poenostavil v preprosto zanko [8], ki kopira samo sebe na podlagi informacije zapisane v notranjosti telesa (slika 1.3).

2	2	2	2	2	2	2	2								
2	1	7	0	1	4	0	1	4	2						
2	0	2	2	2	2	2	2	2	0	2					
2	7	2							2	1	2				
2	1	2							2	1	2				
2	0	2							2	1	2				
2	7	2							2	1	2				
2	1	2	2	2	2	2	2	2	1	2	2	2	2	2	2
2	0	7	1	0	7	1	0	7	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

Slika 1.3: Langtonova zanka. Številke so stanja posamezne celice. Spodaj desno iz zanke raste nova zanka.

Zgoraj naštet primeri nekako opisujejo informacijsko dinamiko v CA. Ne obstaja pa še splošna teorija dinamike informacij v CA, ki bi dinamiko opisala kvantitativno in prostorsko. V svojem članku [25] in prispevkih na konferencah [20, 22, 24] sem grafično upodobil pred slike trenutnega stanja za 1D problem. Iz upodobitve (slika 1.4) je videti, da se ponekod izgubi več informacije kakor drugod, kar kaže na možnost izpeljave splošne teorije dinamike informacij. Na žalost se ta možnost še ni udejanila. Podobno je možno grafično upodobiti pred slike 2D CA, ter iz grafov sklepati o izgubi informacije v 2D CA.



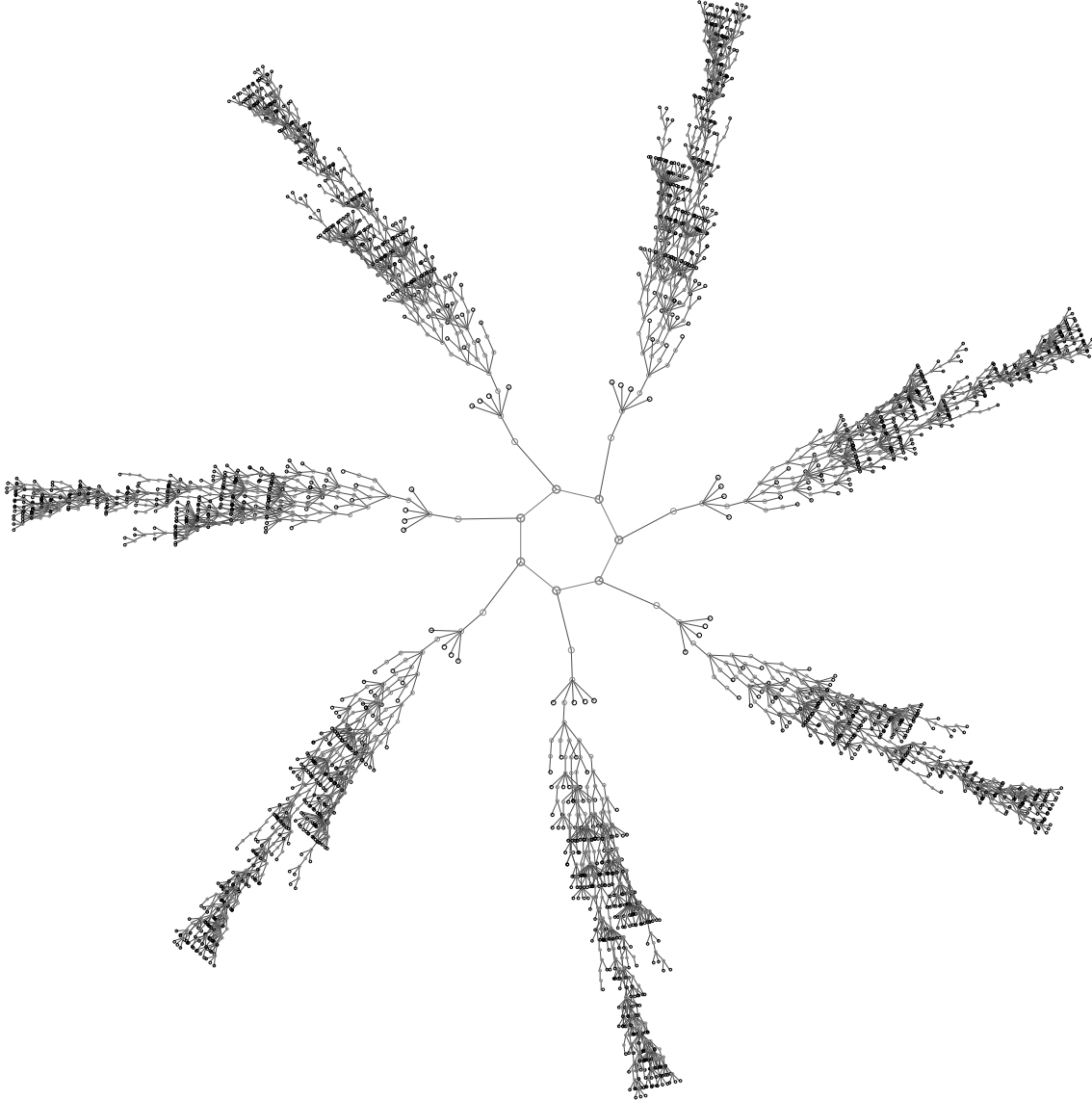
Slika 1.4: Mreža pred slik 1D CA pravilo 110 za tiho ozadje s cikličnim zaporedjem 00010011011111. Vsaka od dveh poudarjenih poti med levim in desnim predstavlja pred sliko. V enem delu sta si pred slike enaki, drugod se razlikujeta. Laično lahko sklepamo, da se tam kjer se pred slike razlikujeta izgubi 1 bit informacije.

### 1.1.3 Atraktorjevo korito

Pomembno orodje za analizo časovno in prostorsko diskretnih dinamičnih sistemov je atraktorjevo korito (slika 1.5). Andrew Wuensche že leta proučuje atraktorjeva korita naključnih binarnih mrež in celičnih avtomatov [41, 42]. To je graf, kjer so vozlišča stanja cikličnega končnega CA, usmerjene poti pa povezujejo vsako stanje z njegovim časovnim naslednikom. Vsa stanja se lahko zberejo v eno ali več korit, odvisno od sistema.

V vsakem CA, ki ni lokalno injektiven, se pojavljajo stanja brez pred slik, imenovana GoE (*Garden of Eden* ali rajski vrt) [29, 30]. Stanja GoE so listi v grafu korita.

Sam atraktor je cikel v grafu korita. To je časovno periodično stanje v katerem se končna časovna evolucija vsakega končnega diskretnega dinamičnega sistema. Cikel lahko vsebuje eno ali več stanj, njihovo število imenujemo perioda atraktorja. Atraktor predstavlja tudi stabilne objekte v neskončnem CA. Za GoL obstajajo katalogi takih objektov, kjer so ti urejeni po velikosti, periodi in hitrosti premikanja. Kakor primer bi navedel mrtvo ozadje (perioda 1) in drsalca (perioda 4).



Slika 1.5: Atraktorjevo korito za elementarno pravilo 110 in konfiguracijo 00010011011111 znotraj atraktorja. Korito vsebuje osrednji cikel atraktorja ter drevesa, ki rastejo iz cikla in se končujejo z listi (stanja GoE). Slika je narejena s programom [www.ddlab.com](http://www.ddlab.com).

## 1.2 Algoritmi za štetje in izpis predslik CA

Doslej sem že razvil napredne algoritme za štetje in izpis predslik 1D CA [25]. Skozi zgodovino so taki algoritmi napredovali, tako da je padala njihova računska zahtevnost in opisna/implementacijska zahtevnost:

1. 'brute force' algoritmi [4]  $O(C^N)$
2. DFS [7], Backtacking [2], SAT [3] algoritmi
3. kombinirani algoritmi, kjer štetje in izpis predslik nista strogo ločena
4. optimalni algoritmi, kjer sta izpis in štetje strogo ločena

Algoritme za reševanje SAT problemov poznam bolj slabo, tako da nisem čisto prepričan v katero kategorijo bi jih postavil.

Bistvo optimalnega algoritma je, da je proces štetja predslik optimalen. Za 1D CA je to doseženo, saj je zahtevnost štetja linearno odvisna od velikosti problema  $O(N)$  ( $N$  je število opazovanih celic). Raziskave algoritma za štetje predslik 2D CA sem se lotil s predpostavko, da je tudi tu možno opraviti štetje predslik z linearno zahtevnostjo. Izkaže se, da 2D problem ni tako preprost. Predstavljeni so primeri, iz katerih je razvidno, da algoritem z linearno zahtevnostjo ne more pravilno opisati vseh situacij. Zahtevnost opisanega algoritma sicer raste eksponentno z velikostjo ene od dimenzij polja CA  $O(C^{N_x})$ , in linearno z velikostjo druge dimenzije  $O(N_y)$  ( $C$  je konstanta odvisna od števila stanj celice in velikosti okolice, ne pa tudi od velikosti polja). Ker pa nisem dokazal optimalnosti opisanega algoritma za štetje, dopuščam možnost, da obstaja algoritem z nižjo kompleksnostjo.

Proces izpisa predslik ima vedno eksponentno rastočo komponento. Ker število vseh stanj sistema raste eksponentno z velikostjo problema, posledično tudi povprečje števila predslik skozi vsa stanja raste eksponentno. Optimalen algoritem za izpis predslik je časovno in pomnilniško linearno odvisen od števila predslik. Moja algoritma izpis predslik 1D in 2D CA sta optimalna in omogočata izpis predslik brez slepih poti, kar je posledica podrobne analize problema ob štetju.

Ostli znani algoritmi spadajo v drugo kategorijo. Večinoma izkoriščajo le lokalno znanje znanje o sistemu (lokalno štetje predslik), ne pa tudi globalnih števec. Zato zahajajo v slepe poti in se morajo vračati v prejšnje stanje ali pa opuščajo neustrezne rešitve, katerim so že namenili procesni čas.

Pomembno vlogo pri razvoju opisanega algoritma ima mreža predslik. To je grafična upodobitev problema, katere cilj je lažje razumevanje problema in rešitve. Osnova za oblikovanje mreže predslik so De Bruijnovi diagrami. Posamezen De Bruijnov diagram je mreža predslik ene celice. Mreže posameznih celic se nato povezujejo, tako da na koncu opisujejo celotno polje celic.

McIntoch in njegovi učenci so bili pobudniki uporabe De Bruijnovi diagramov za analizo 1D CA. Uporabljali so jih za štetje in izpis predslik ter za analizo delcev in njihovih interakcij. Paulina Léon in Genaro Martínez (McIntoshev učenec) [27] sta začetnika apliciranja De Bruijnove diagramov na 2D CA, vendar jih ne uporabljata za štetje in izpis predslik.

Največ raziskav s področja predslik 2D CA je bilo opravljenih ravno s ciljem iskanja stanj GoE v avtomatu GoL. S stališča algoritma za štetje predslik je stanje GoE tako stanje, katerega število predslik je nič. Algoritem za štetje predslik je možno pretvoriti v manj zahteven algoritem za preverjanje ali je dano stanje stanje GoE, tako da se operacije nad celimi števili pretvori v logične operacije nad Boolovimi stanji.

Opisan algoritem na koncu primerjam z ostalimi doslej znanimi algoritmi, ki večinoma temeljijo na DFS principu.

### 1.2.1 Implementacija algoritma

Algoritem je implementiran kot računalniški program v jeziku C [21]. Knjižnica GMP <sup>1</sup> je uporabljena za zapis celih števil, večjih od 64 bitov. Poleg samega algoritma za štetje in izpis predslik sem pripravil tudi orodje za simulacijo binarnega CA z okolico quad [23], ki temelji na simulatorju za GoE [40]. Simulator se lahko zažene kar v internetnem brskalniku.

### 1.2.2 Primeri in ilustracije

Večina primerov in ilustracij opisuje binarni 2D CA z majhno okolico quad (štiri celice na kvadratnem polju, Toffoli 2008 [36]). Večina primerov in sploh ilustracije za GoL bi bile zaradi  $2^{3 \cdot 3} = 512$  možnih stanj okolice preveč kompleksne in nepregledne. V nasprotju ima binarna okolica quad le  $2^{2 \cdot 2} = 16$  možnih stanj.

Želel sem sicer uporabiti kako določeno pravilo za okolico quad, tako ki bi omogočalo zanimivo dinamiko delcev, vendar na tem področju še ni kaj dosti raziskav. Obstaja le dokaz univerzalnosti za binarni avtomat z okolico trid (Powley 2008 [32]).

V slikah je uporabljena izometrična projekcija, saj je za potrebe analize osnovnemu 2D polju dodana tretja dimenzija, ki opisuje prostor predslik (mreža predslik).

---

<sup>1</sup>The GNU Multiple Precision Arithmetic Library <https://gmplib.org/>

# Poglavje 2

## Definicija CA

Omejujemo se na celične avtomate, ki so:

- prostorsko diskretni (celice),
- časovno diskretni (korak),
- homogeni (pravila so enaka za vse celice in vse celice se posodobijo naenkrat) in
- deterministični (novo stanje je odvisno le od trenutnega stanja).

Vsaka celica ima diskretno vrednost  $c$  iz nabora stanj celice  $S$ . Stanja so oštevilčena.

$$c \in S \quad \text{in} \quad S = \{0, 1, \dots, |S| - 1\} \quad (2.1)$$

Za potrebe implementacije algoritma je pomembno, da smo stanje elementov CA kakor so pravilo, vrednost okolice in podobno sposobni indeksirati. Uporabljena je metoda, ki je v osnovi bila namenjena zapisu pravila. Posplošeno, so vrednosti celic  $c(i)$  zložene v niz  $A = [c(0), c(1), \dots, c(|A| - 1)]$  dolžine  $|A|$ , ki je interpretiran kakor  $|A|$  mestno pozitivno celo število  $num$  v  $S$ -iškem številskem sestavu (za podane primere dvojiškem).

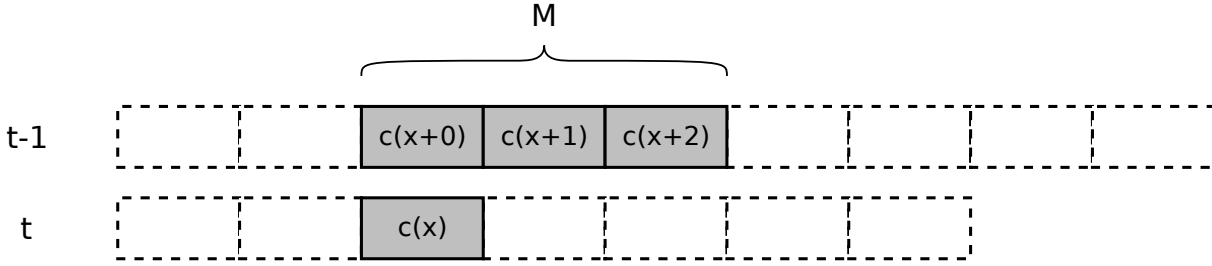
$$num = \sum_{i=0}^{|A|-1} |S|^i \cdot c(i) \quad (2.2)$$

### 2.1 Definicija 1D CA

Definicija 1D CA je tukaj na hitro povzeta po [25]. Za 1D CA se celice združujejo v regularen niz. Na splošno je dolžina niza lahko neskončna, bolj običajni pa so končni nizi dolžine  $N$ .

### 2.1.1 Okolica

Za potrebe enostavnosti implementacije se okolica celice širi le v pozitivni smeri (slika 2.1). Torej je za celico  $c(x)$  njena okolica velikosti  $M$  enaka  $n(x) = \{c(x), c(x+1), \dots, c(x+M-1)\}$ . V praksi se podana definicija okolice razlikuje od bolj običajne centrirane okolice le v tem, kako je časovno sosledje stanj poravnano v grafični upodobitvi. Nekoliko neobičajno definicijo sem uporabil, ker ne potrebuje ločene interpretacije za okolice sode velikosti. Hkrati sem se tako izognil uporabi negativnih števil v implementaciji algoritma.



Slika 2.1: Okolica 1D CA velikosti  $M = 3$ .

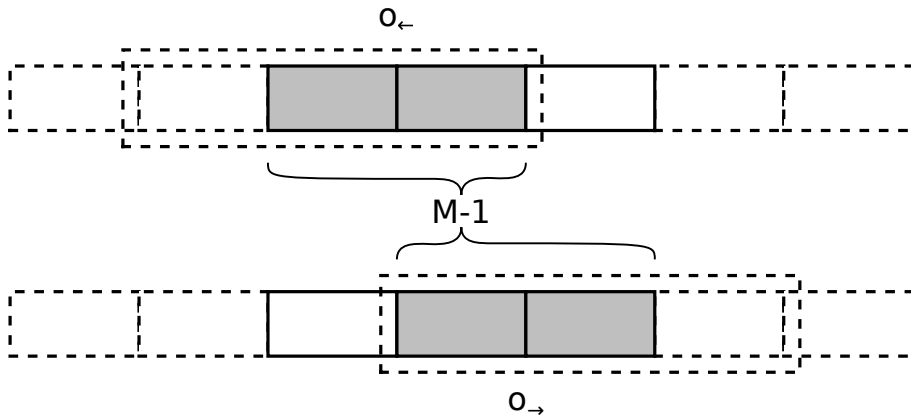
Indeks okolice  $n(x)$  in nabor možnih okolic sta:

$$n(x) = \sum_{i=0}^{M-1} |S|^i \cdot c(x+i) \quad (2.3)$$

$$n \in \{0, 1, \dots, |S|^M - 1\} \quad (2.4)$$

### 2.1.2 Prekrivanje okolic

Dana okolica  $n(x)$  se za  $M-1$  celic prekriva z okolicama sosednjih celic (slika 2.2). Na levi imamo prekrivanje  $o_{\leftarrow}(x)$  med okolicama  $n(x-1)$  in  $n(x)$ . Na desni imamo prekrivanje  $o_{\rightarrow}(x)$  med okolicama  $n(x)$  in  $n(x+1)$ .



Slika 2.2: Prekrivanje okolic 1D CA velikosti  $M - 1 = 2$ .

Indeksa levega prekrivanja  $o_{\leftarrow}(x)$  ter desnega prekrivanja  $o_{\rightarrow}(x)$  prekrivanj in nabor možnih prekrivanj so:

$$o_{\leftarrow}(x) = \sum_{i=0}^{i=M-2} |S|^i \cdot c_{x+i} \quad (2.5)$$

$$o_{\rightarrow}(x) = \sum_{i=1}^{i=M-1} |S|^i \cdot c_{x+i} \quad (2.6)$$

$$o \in \{0, 1, \dots, |S|^M - 2\} \quad (2.7)$$

### 2.1.3 Lokalna tranzicijska funkcija in pravilo

Preslikava sedanje okolice  $n^t(x)$  v prihodnjo istoležno celico  $c^{t+1}(x)$  je definirana s tranzicijsko funkcijo  $f$ , ki vsaki vrednosti okolice pripiše vrednost celice:

$$c^{t+1}(x) = f(n^t(x)) \quad (2.8)$$

Za potrebe iskanja predslik je zanimiva obratna funkcija  $f^{-1}$ , ki ob podanem stanju trenutne celice  $c^t(x)$  vrne množico okolic  $n^{t-1}(x)$ , ki se preslikajo v to vrednost:

$$f^{-1}(c^t(x)) = \{n^{t-1}(x) \in S^m \mid f(n^{t-1}(x)) = c^t(x)\} \quad (2.9)$$

Tranzicijsko funkcijo je možno definirati s pravilom  $r$ . Pravilo je indeks izbrane funkcije znotraj celotnega nabora  $|S|^{|S|^M}$  funkcij. Definirano je kakor celo število v  $S$ -iškem številskem sestavu, kjer so cifre zaporedje vrednosti celic v katere tranzicijska funkcija preslika vsako od  $|S|^M$  možnih okolic:

$$r = \sum_{n=0}^{n=|S|^M-1} |S|^n \cdot f(n) \quad (2.10)$$

$$r \in \{0, 1, \dots, |S|^{|S|^M} - 1\} \quad (2.11)$$

Globalna tranzicijska funkcija se pogosto zapisuje kakor tranzicijska tabela, ki navaja vrednost tranzicijske funkcije  $f(n)$  za vsako okolico  $n$ .

### 2.1.4 Robni pogoji

Neskončen niz celic nima roba in zanj ne potrebujemo definicije robnih pogojev. Hkrati pa v praksi ne moremo računati z neskončnostjo, zato obravnavamo končne nize celic.

Če smatramo opazovani končen niz za del neskončnega niza, potem se moramo opredeliti, kako bomo obravnavali vrednosti celic izven opazovanega niza. Tukaj je opisan samo odprt robni pogoj, kjer celice izven roba niso definirane, oziroma lahko zasedejo poljubno vrednost. Za niz dolžine  $N$  je potemtakem definiranih le  $N - (M - 1)$  okolic.

Drugi običajen robni pogoj je ciklični, kjer je niz celic sklenjena zanka. Ciklično koordinato celice  $x_{\odot}$  izračunamo iz  $x$  po modulu  $N$ .

$$x_{\odot} = x \bmod N \quad (2.12)$$

### 2.1.5 Globalna tranzicijska funkcija

Globalna tranzicijska funkcija  $F$  s pomočjo lokalne tranzicijske funkcije  $f(n)$  preslika vsako okolico  $n^t(x)$  iz niza celic  $\alpha^t$  v času  $t$  v celico  $c^{t+1}(x)$  iz niza celic  $\beta^{t+1}$  v času  $t + 1$ .

$$\beta^{t+1} = F(\alpha^t) \quad (2.13)$$

Za odprti robni pogoj vrne globalna funkcija za vhodni niz dolžine  $N$  izhodni niz dolžine  $N - (M - 1)$ , za ciklični robni pogoj pa niz dolžine  $N$ .

### 2.1.6 Predslike

Vsaka predslika  $\beta^{t-1}$  danega niza celic  $\alpha^t$  mora ustrezati dvema pogojema:

1. Za vsako celico  $c^t(x)$  mora okolica  $n^{t-1}(x)$  ustrezati inverzni tranzicijski funkciji  $f^{-1}$ :

$$\forall x : c^t(x) = f(n^{t-1}(x)) \quad (2.14)$$

2. Vsak par okolic  $\{n^{t-1}(x), n^{t-1}(x + 1)\}$  se mora ujemati v prekrivanju.

$$\forall x : o^{t-1}(x) = o_{\rightarrow}^{t-1}(x) = o_{\leftarrow}^{t-1}(x + 1) \quad (2.15)$$

Opazovanje prekrivanja okolic je glavni element algoritmov za iskanje predslik.



## 2.2 Definicija 2D CA

Za 2D CA se celice združujejo v regularno 2D polje. Mreža polja je lahko pravokotna, šestkotna ali celo kvazikristalna. Tukaj se bomo omejili na pravokotno mrežo. Na splošno je velikost polja lahko neskončna, bolj običajna pa so končna polja, definirana kakor pravokotnik velikosti  $N_x \times N_y$ . Skupno število celic v končnem polju je  $N = N_x \cdot N_y$ .

Za potrebe indeksiranja elementov CA, so ploskve razdeljene v vrstice, nato so vrstice sestavljene v niz, ki je interpretiran kakor število. Cifre v številu si sledijo od spodaj levo do zgoraj desno znotraj okolice (sliki 2.3 in 2.4).

### 2.2.1 Okolica

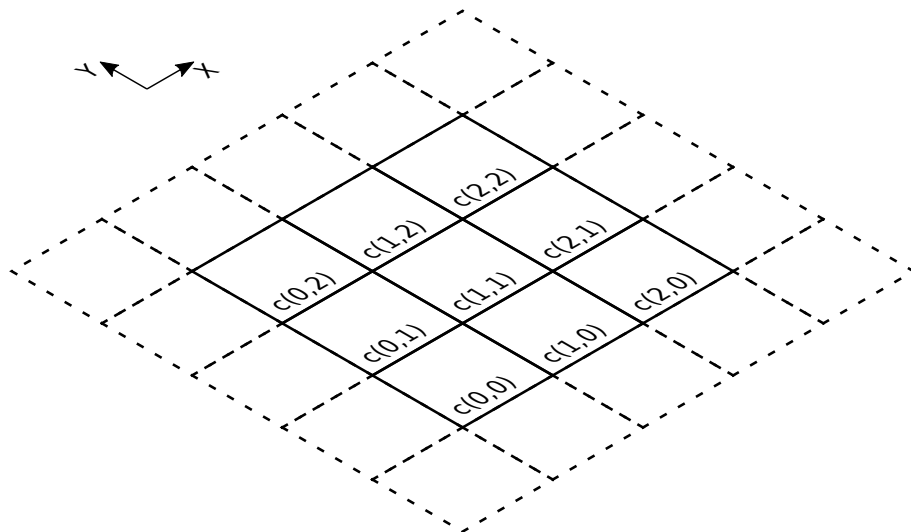
Prihodnje stanje neke celice  $c_{x,y}$  s koordinatami  $(x,y)$  je odvisno od trenutnega stanja pripadajoče okolice  $n_{x,y}$  (slika 2.5). Tudi pri obliki okolice se bom omejil na pravokotnik velikosti  $M_x \times M_y$ . Število celic v okolici je  $M = M_x \cdot M_y$ . Na voljo je  $|S|^M$  možnih okolic.

$$n \in S^M \quad \text{in} \quad S^M = \{0, 1, \dots, |S|^M - 1\} \quad (2.16)$$

Indeks okolice je (sliki 2.3 in 2.4):

$$n(x, y) = \sum_{\substack{i=0 \\ j=0}}^{i=M_x-1 \\ j=M_y-1} |S|^{M_x j + i} \cdot c_{x+i, y+j} \quad (2.17)$$

$$n \in \{0, 1, \dots, |S|^{M_x M_y} - 1\} \quad (2.18)$$

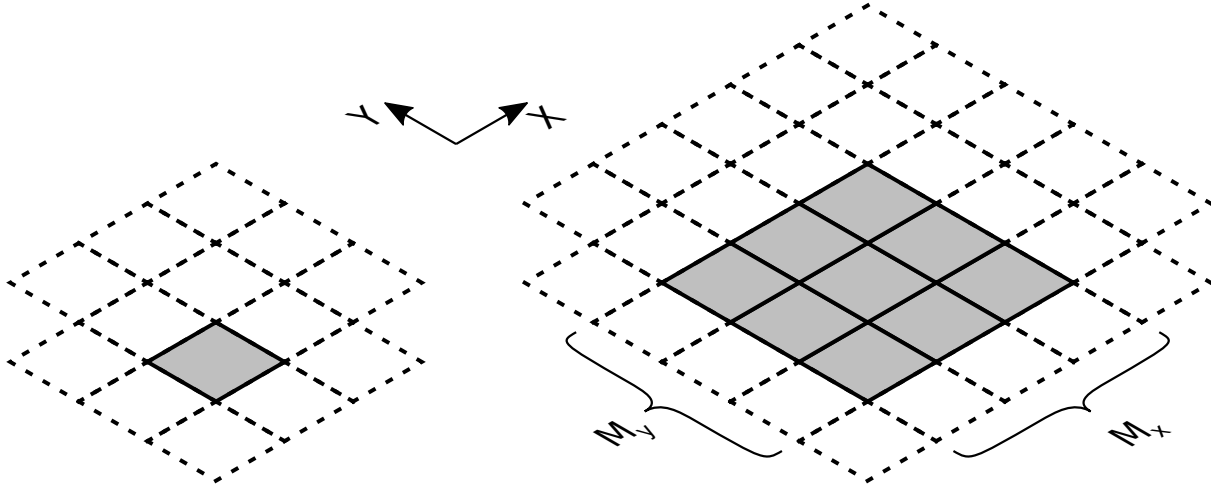


Slika 2.3: Indeksiranje okolice celice z dimenzijama  $M_x = M_y = 3$ .



Slika 2.4: Indeksiranje okolice celice z dimenzijama  $M_x = M_y = 2$ .

Prostorski odnos med okolico in celico v prihodnjem stanju avtomata, ki jo ta okolica določa, ni podrobno definiran. Običajno se smatra, da je celica v sredini okolice, ampak za opisani algoritem to ni nujno pomembno. Moja implementacija algoritma celice, okolice in samo polje indeksira tako, da je začetna ali referenčna celica v spodnjem levem kotu.



Slika 2.5: Celica  $c(x, y)$  in pripadajoča okolica  $n(x, y)$  z dimenzijama  $M_x = M_y = 3$ .

### 2.2.2 Prekrivanje okolic

Celice se v smeri dimenzije X prekrivajo za ploskev velikosti  $(M_x - 1) \times M_y$  (sliki 2.6 in 2.7). To ob indeksiranju da nabor:

$$o_{\leftarrow}(x, y) = \sum_{\substack{i=M_x-2 \\ j=M_y-1 \\ i=0 \\ j=0}}^{i=M_x-1} |S|^{(M_x-1)j+i} \cdot c_{x+i, y+j} \quad (2.19)$$

$$o_{\rightarrow}(x, y) = \sum_{\substack{i=M_x-1 \\ j=M_y-1 \\ i=1 \\ j=0}}^{i=M_x-1} |S|^{(M_x-1)j+i} \cdot c_{x+i, y+j} \quad (2.20)$$

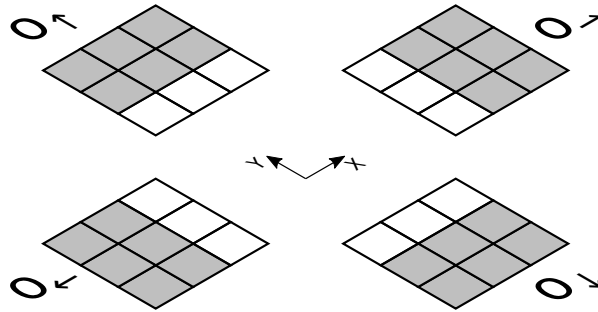
$$o_{\leftrightarrow} \in \{0, 1, \dots, |S|^{(M_x-1)M_y} - 1\} \quad (2.21)$$

Celice se v smeri dimenzije Y prekrivajo za ploskev velikosti  $M_x \times (M_y - 1)$  (sliki 2.6 in 2.7). To ob indeksiranju da nabor:

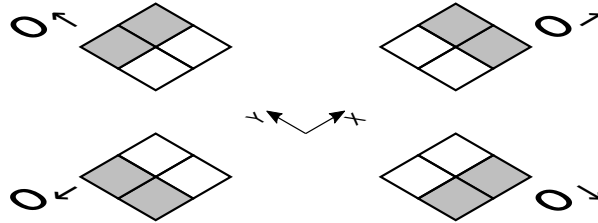
$$o_{\downarrow}(x, y) = \sum_{\substack{i=0 \\ j=0}}^{i=M_x-1 \\ j=M_y-2} |S|^{M_x j + i} \cdot c_{x+i, y+j} \quad (2.22)$$

$$o_{\uparrow}(x, y) = \sum_{\substack{i=0 \\ j=1}}^{i=M_x-1 \\ j=M_y-1} |S|^{M_x j + i} \cdot c_{x+i, y+j} \quad (2.23)$$

$$o_{\downarrow} \in \{0, 1, \dots, |S|^{M_x(M_y-1)} - 1\} \quad (2.24)$$



Slika 2.6: Prekrivanje okolic sosednjih celic v smeri dimenzij X in Y, za velikost okolice  $M_x = M_y = 3$ . Okolice se prekrivajo v 6 celicah od 9.



Slika 2.7: Prekrivanje okolic sosednjih celic v smeri dimenzij X in Y, za velikost okolice  $M_x = M_y = 2$ . Okolice se prekrivajo v 2 celicah od 4.

Celice se v diagonalni smeri prekrivajo za ploskev velikosti  $(M_x - 1) \times (M_y - 1)$  (sliki 2.8 in 2.9). To ob indeksiranju da nabor:

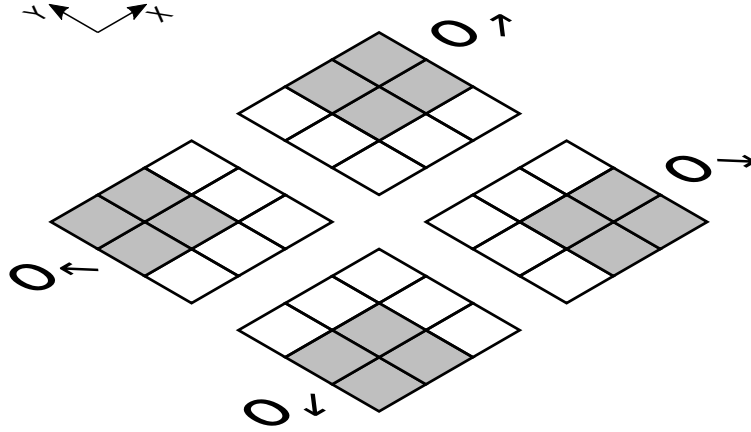
$$o_{\swarrow} = \sum_{\substack{i=0 \\ j=0}}^{i=M_x-2 \\ j=M_y-2} |S|^{(M_x-1)j+i} \cdot c_{x+i, y+j} \quad (2.25)$$

$$o_{\searrow} = \sum_{\substack{i=1 \\ j=0}}^{i=M_x-1 \\ j=M_y-2} |S|^{(M_x-1)j+i} \cdot c_{x+i, y+j} \quad (2.26)$$

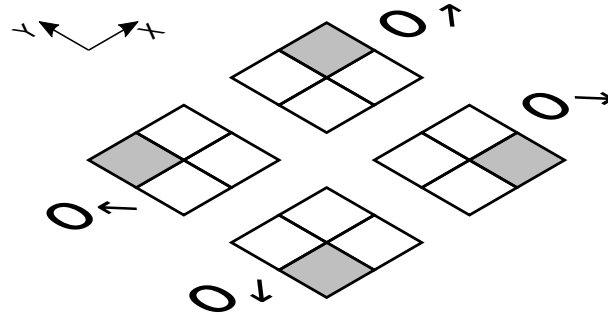
$$o_{\nwarrow} = \sum_{\substack{i=M_x-2 \\ j=M_y-1}}^{i=0 \\ j=1} |S|^{(M_x-1)j+i} \cdot c_{x+i,y+j} \quad (2.27)$$

$$o_{\nearrow} = \sum_{\substack{i=M_x-1 \\ j=M_y-1}}^{i=1 \\ j=1} |S|^{(M_x-1)j+i} \cdot c_{x+i,y+j} \quad (2.28)$$

$$o_{\times} \in \{0, 1, \dots, |S|^{(M_x-1)(M_y-1)} - 1\} \quad (2.29)$$



Slika 2.8: Prekrivanje okolic sosednjih celic v diagonalni smeri, za velikost okolice  $M_x = M_y = 3$ . Okolice se prekrivajo v 4 celicah od 9.



Slika 2.9: Prekrivanje okolic sosednjih celic v diagonalni smeri, za velikost okolice  $M_x = M_y = 2$ . Okolice se prekrivajo v eni celici od 4.

### 2.2.3 Lokalna tranzicijska funkcija in pravilo

Preslikava sedanje okolice  $n^t(x, y)$  v prihodnjo istoležno celico  $c^{t+1}(x, y)$  je definirana s tranzicijsko funkcijo  $f$ , ki vsaki vrednosti okolice pripiše vrednost celice:

$$c^{t+1}(x, y) = f(n^t(x, y)) \quad (2.30)$$

Za potrebe iskanja predslik je zanimiva obratna funkcija  $f^{-1}$ , ki ob podanem stanju trenutne celice  $c^t(x, y)$  vrne množico okolic  $n^{t-1}(x, y)$ , ki se preslikajo v to vrednost:

$$f^{-1}(c^t(x, y)) = \{n^{t-1}(x, y) \in S^m \mid f(n^{t-1}(x, y)) = c^t(x, y)\} \quad (2.31)$$

Tranzicijsko funkcijo je možno definirati s pravilom  $r$ . Pravilo je indeks izbrane funkcije znotraj celotnega nabora  $|S|^{|S|^{M_x M_y}}$  funkcij. Definirano je kakor celo število v  $S$ -iškem številskem sestavu, kjer so cifre zaporedje vrednosti celic v katere tranzicijska funkcija preslika vsako od  $|S|^{M_x M_y}$  možnih okolic:

$$r = \sum_{n=0}^{n=|S|^{M_x M_y}-1} |S|^n \cdot f(n) \quad (2.32)$$

$$r \in \{0, 1, \dots, |S|^{|S|^{M_x M_y}} - 1\} \quad (2.33)$$

Globalna tranzicijska funkcija se pogosto zapisuje kakor tranzicijska tabela, ki navaja vrednost tranzicijske funkcije  $f(n)$  za vsako okolico  $n$ .

## 2.2.4 Robni pogoji

Neskončno polje nima roba in zanj ne potrebujemo definicije robnih pogojev. Hkrati pa v praksi ne moremo računati z neskončnostjo, zato obravnavamo končna polja celic.

Če smatramo opazovano končno polje za del neskončnega polja, potem se moramo opredeliti, kako bomo obravnavali vrednosti celic izven opazovanega polja. Tukaj je opisan samo odprt robni pogoj, kjer celice izven roba niso definirane, oziroma lahko zasedejo poljubno vrednost. Za polje velikosti  $N_x \times N_y$  je potemtakem definiranih le  $(N_x + (M_x - 1)) \times (N_y + (M_y - 1))$  okolic.

Drugi običajen robni pogoj je ciklični, kjer je polje celic torus. Ciklično koordinato celice  $x_{\odot}, x_{\odot}$  izračunamo iz  $x, y$  po modulu  $N_x$  oziroma  $N_y$ .

$$\begin{aligned} x_{\odot} &= x \bmod N_x \\ y_{\odot} &= y \bmod N_y \end{aligned} \quad (2.34)$$

## 2.2.5 Globalna tranzicijska funkcija

Globalna tranzicijska funkcija  $F$  s pomočjo lokalne tranzicijske funkcije  $f(n)$  preslika vsako okolico  $n^t(x, y)$  iz niza celic  $A^t$  v času  $t$  v celico  $c^{t+1}(x)$  iz niza celic  $B^{t+1}$  v času  $t + 1$ .

$$B^{t+1} = F(A^t) \quad (2.35)$$

Za odprti robni pogoj vrne globalna funkcija za vhodno polje velikosti  $N_x \times N_y$  izhodno polje velikosti  $(N_x + (M_x - 1)) \times (N_y + (M_y - 1))$ , za ciklični robni pogoj pa polje velikosti  $N_x \times N_y$ .

## 2.2.6 Predslike

Vsaka predslika  $B^{t-1}$  danega polja celic  $A^t$  mora ustrezati naslednjim pogojem.

1. Za vsako celico  $c^t(x, y)$  mora okolica  $n^{t-1}(x, y)$  ustrezati inverzni funkciji  $f^{-1}$ :

$$\forall x, y : c^t(x, y) = f(n^{t-1}(x, y)) \quad (2.36)$$

2. Vsak par okolic v dimenziji X  $\{n^{t-1}(x, y), n^{t-1}(x+1, y)\}$  se mora ujemati v prekrivanju.

$$\forall x, y : o_{\leftarrow}^{t-1}(x, y) = o_{\rightarrow}^{t-1}(x, y) = o_{\leftarrow}^{t-1}(x+1, y) \quad (2.37)$$

3. Vsak par okolic v dimenziji Y  $\{n^{t-1}(x, y), n^{t-1}(x, y+1)\}$  se mora ujemati v prekrivanju.

$$\begin{aligned} \forall x, y : o_{\downarrow}^{t-1}(x, y) &= o_{\downarrow}^{t-1}(x, y+1) \\ &= o_{\uparrow}^{t-1}(x, y+0) \end{aligned} \quad (2.38)$$

4. Vsaka četvorka diagonalnih okolic  $\{n^{t-1}(x, y), n^{t-1}(x+1, y), n^{t-1}(x, y+1), n^{t-1}(x+1, y+1)\}$  se mora ujemati v prekrivanju.

$$\begin{aligned} \forall x, y : o_{\times}^{t-1}(x, y) &= o_{\searrow}^{t-1}(x+0, y+1) = o_{\swarrow}^{t-1}(x+1, y+1) \\ &= o_{\nearrow}^{t-1}(x+0, y+0) = o_{\nwarrow}^{t-1}(x+1, y+0) \end{aligned} \quad (2.39)$$

Opazovanje prekrivanja okolic je glavni element algoritmov za iskanje predslik.

## 2.2.7 Primer

Pri 1D elementarnih CA ( $m = 3$ ) je vseh pravil le 256, tako da so bila že vsa opazovana in opisana. Za 2D CA z okolico quad ( $m = 4$ ) je pravil 65536, kar je nekje na meji, kar lahko raziskovalci podrobno pregledajo. Za 2D CA z Moorovo okolico ( $m = 9$ ) je vseh pravil dosti preveč, da bi lahko pregledali posamično.

Podana konstrukcija mreže predslik in algoritem za izračun predslik omogočata uporabo bolj splošne definicije. Za vsako celico je lahko definiran lasten nabor predslik  $n^{t-1} \in S^m$ , ki je na splošno neodvisen od stanja celice. Ta posplošitev je uporabljena za konstrukcijo umetnih mrež predslik, ki poudarjajo konkretne probleme, povezane s kompleksnostjo algoritma.

# Poglavje 3

## Konstrukcija mreže predslik

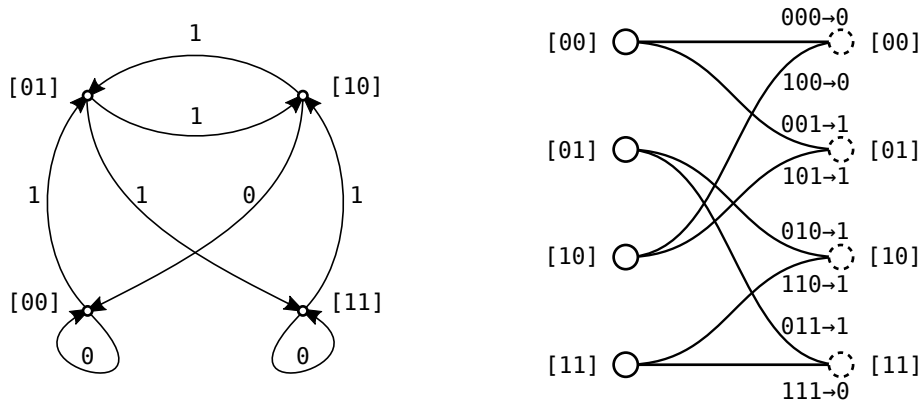
Mreža predslik je grafični konstrukt, ki omogoča upodobitev okolice, prekrivanja okolic in tranzicijske funkcije kakor ločene grafične elemente (vozlišča, poti in ploskve). Odnosi med temi elementi definirajo pravila, na katerih se gradijo algoritmi za iskanje predslik.

### 3.1 De Bruijnov diagram

Osnovni element grafične upodobitve je nekoliko svobodna interpretacija De Bruijnovega diagrama [6]. V osnovi ta obravnava ciklične premike končnih zaporedij simbolov ter njihovo prekrivanje. Harold V. McIntosh [28] in njegova skupina [35] so priredili De Bruijnove grafe za potrebe analize 1D CA (slika 3.1 levo). Vozlišča predstavljajo nabor možnih prekrivanj med okolicami, poti med njimi pa okolice, ki se v vrednosti celic ujemajo s temi prekrivanji. Tranzicijska funkcija pripiše vsaki okolici stanje celice, posledično so poti med vozlišči označene z vrednostmi iz tranzicijske tabele. De Bruijnove Grafe nato razdelimo na podgrafe za vsako stanje celice. Podgraf vsebujejo le tiste poti, katerih pripadajoče okolice se preslikajo v eno od stanj celice. De Bruijnov podgraf predstavlja nabor vseh preteklosti celice z danim stanjem.

### 3.2 Mreža predslik 1D CA

Sam sem razvil [25] modificirano upodobitev De Bruijnovega grafa, kjer so originalna vozlišča podvojena, in gredo poti vedno od originala proti dvojniku (slika 3.1 desno). Taka upodobitev omogoča nizanje grafov. Originalna vozlišča grafa za naslednjo celico se ujemajo s kopijami vozlišč za naslednjo celico (slika 1.4). Medtem ko osnovni De Bruijnov graf opisuje okolico le ene same celice, nizan graf opisuje niz celic. Nizan graf sem poimenoval mreža predslik, saj predstavlja nabor vseh preteklosti celotne konfiguracije.



Slika 3.1: De Bruijnov graf za elementarni 1D CA, pravilo 110. Na levi je McIntosheva upodobitev, na desni pa moja, ki omogoča nizanje. S prekinjeno črto so predstavljene kopije vozlišč.

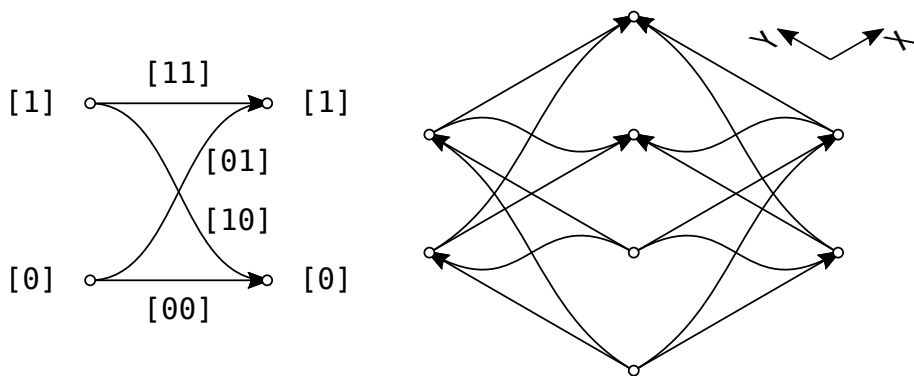
### 3.3 Mreža preslik 2D CA

Za potrebe opisa 2D celičnih avtomatov je bila elementom De Bruijnovega grafa dodana nova dimenzija. Povezave med vozlišči se spremenijo v ploskve, vozlišča pa se spremenijo v robove ploskev. Elementi celičnega avtomata, ki se preslikajo v graf, so:

- nabor vseh možnih **okolic** celice postane nabor vseh **ploskev** (slika 3.3)
- nabor **prekrivanj okolic v smeri 2D dimenzij** (slika 2.7) postane nabor **povezav**
- nabor **prekrivanj okolic v diagonalni smeri** (slika 2.9) postane nabor **vozlišč**

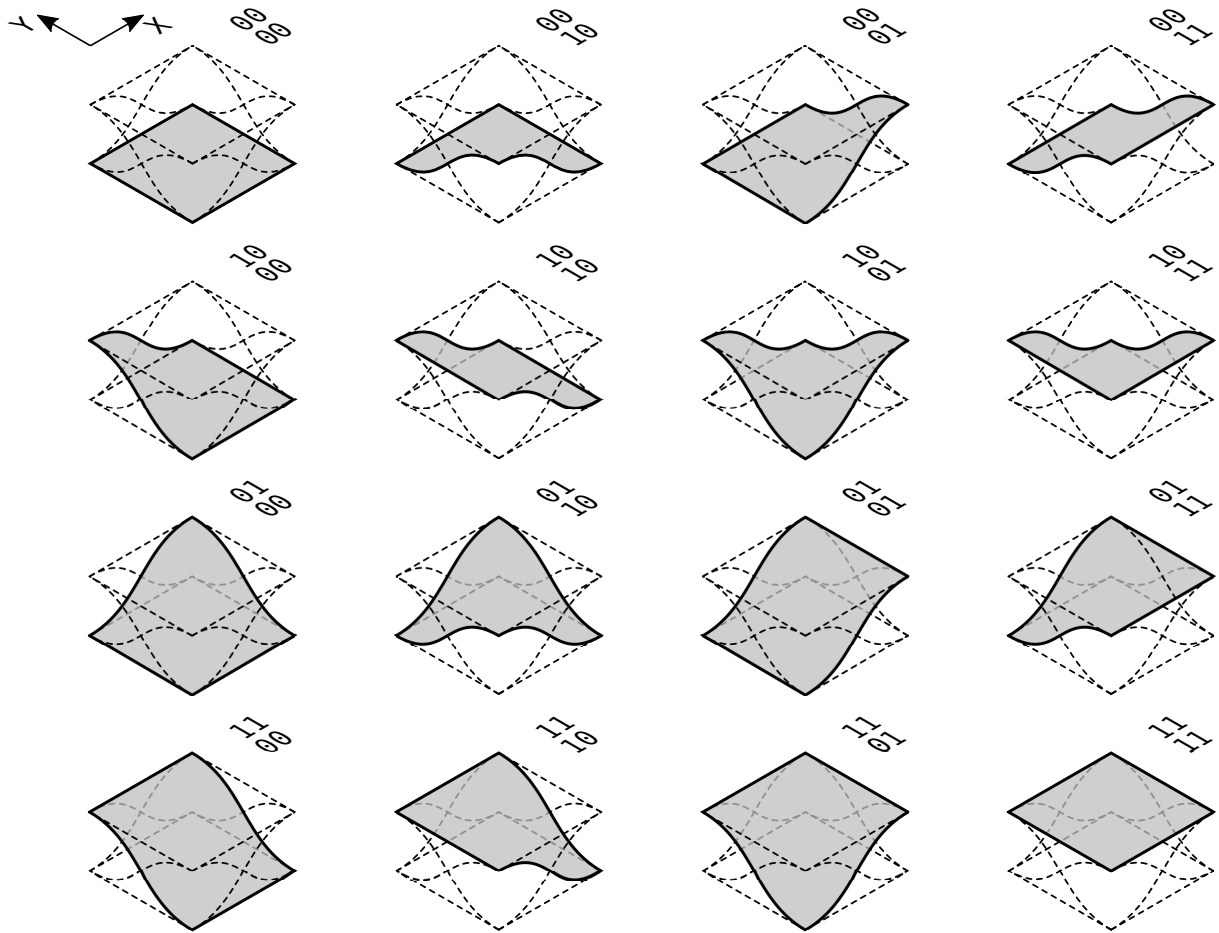
Graf preslik je postavljen na kvadrat, ki predstavlja okolico  $n$ . Nad vsakim vogalom je nabor vozlišč, ki predstavljajo nabor vogalnih prekrivanj  $o_{x-y-}$ ,  $o_{x+y-}$ ,  $o_{x-y+}$  in  $o_{x+y+}$ . Vozlišči sosednjih vogalov sta povezani, če sta del iste konfiguracije celic, ki predstavlja povezavo. Na primer, pri okolici quad povezava za prekrivanje okolic [01] povezuje vozlišči [0] in [1] (slika 3.2 levo).

Nastali graf (slika 3.2 desno) ima poleg vozlišč in povezav med njimi tudi ploskve. Ploskve bi v teoriji grafov opisali kakor zanke v grafu, z dodano omejitvijo, da mora vsako vozlišče ali povezava v zanki pripadati drugemu prekrivanju okolic.



Slika 3.2: Mreža ene celice za binarni CA z okolico quad  $M_x = M_y = 2$ .





Slika 3.3: Nabor ploskev za vseh 16 možnih vrednosti okolic za binarni CA z okolico quad  $M_x = M_y = 2$ .

### 3.4 Mreža

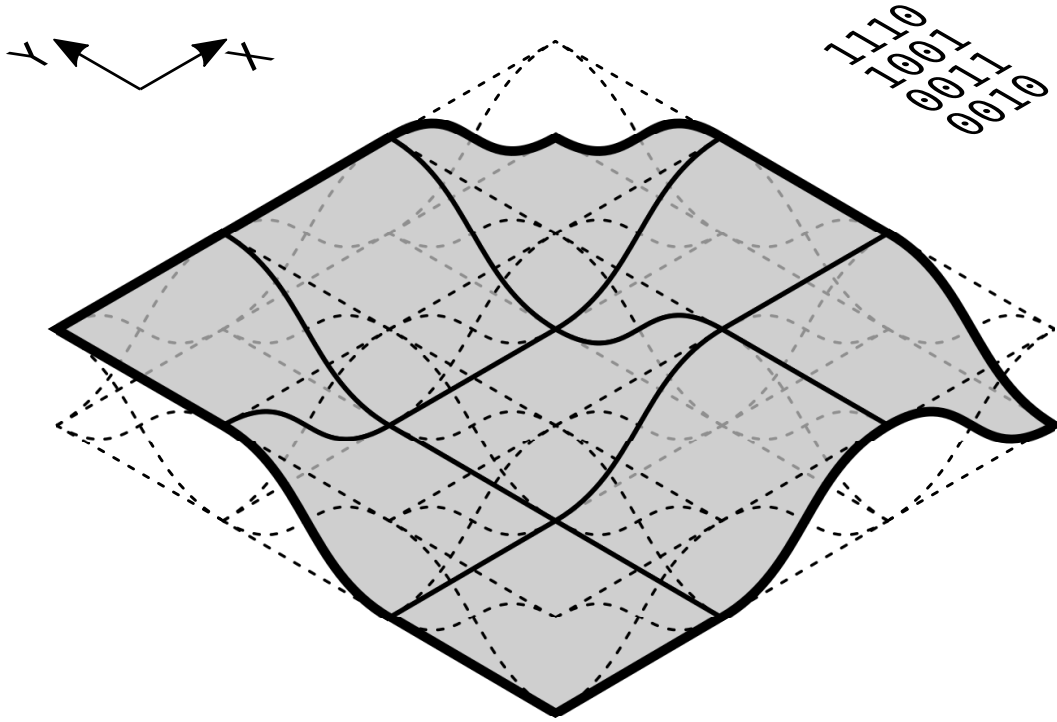
Diagrame, ki opisujejo preteklost posamezne celice, je možno sestaviti v mrežo, ki opisuje polje več celic. Nabor predslik celotnega polja je ekvivalenten naboru vseh *zveznih* ploskev, ki prekrivajo celotno polje (zahteva po prekrivanju okolic) in ki jih je možno sestaviti iz naborov ploskev diagramov posameznih celic (zahteva po ujemanju s tranzicijsko funkcijo).

Preslikava iz zvezne ploskve v mreži predslik v konfiguracijo polja celic predslike je enolična. Najlažje je razumeti preslikavo za okolico velikosti  $3 \times 3$ . Od vsakega odseka ploskve za posamezno celico vzamemo centralno celico, na koncu pa dodamo še za eno celico širok rob okoli celotne ploskve.

Podani primeri uporabljajo binarni CA z okolico quad. Za ta CA je velikost diagonalnega prekrivanja okolic ena sama celica (slika 2.9); posledično ima nabor vozlišč le dve vrednosti, ki neposredno predstavljajo vrednosti celic v predsliki (slika 3.4). Nabor okolic/ploskev pa obsega 16 kombinacij (slika 3.3).

Pravilnost razširitve diagrama ene celice v mrežo lahko dokažemo z indukcijo. Dokazati želimo, da je neka konfiguracija celic predslika dane sedanosti, če in samo če je ta ekvivalentna zvezni ploskvi v mreži predslik.

**Prvi element:** Iz definicije velja, da je za eno samo celico v mreži nabor ploskev enak naboru vseh predslik. **Naslednji element:** Obstoječi mreži predslik dodamo novo celico. Ploskev iz nabora dodane celice se zvezno veže s ploskvijo iz obstoječega nabora zveznih ploskev, ko se z njo ujema v robu (vozliščema in povezavi med njima). Robovi ploskev se ujemajo natanko v primeru, ko se ujemajo prekrivanja okolic. Temu je tako, ker so indeksi vozlišč in povezav ekvivalentni vrednosti prekrivanj okolic.



Slika 3.4: Mreža velikosti  $N_x = 3$  in  $N_y = 3$  za binarni CA z okolico quad. Poudarjena je ena zvezna ploskev in njen zvezen rob. Konfiguracija pripadajoče predslike je izpisana.

### 3.5 Robni pogoji

Pri 1D CA so robni pogoji definirani na dveh koncih, ki omejujejo končno število celic na neskončni premici. Če je 1D CA definiran kakor poltrak, je robni pogoj samo eden. Robni pogoj definira, katere okolice (vozlišča pri mreži predslik za 1D CA) in s kakšnimi utežmi so na voljo ob robu. Lahko si jih predstavljamo tudi kakor vpliv neskončnega poltraka celic, ki sega izven roba opazovane konfiguracije.

Pri 2D CA je robni pogoj definiran kakor utež sklenjene poti okoli ploskve opazovane konfiguracije celic. V mreži predslik za 2D CA povezave med vozlišči definirajo rob ploskve (slika 3.4). Vsaka zvezna ploskev v mreži ima zvezen rob. Robni pogoj določa, kako je ta ploskev obravnavana, oziroma, ali je sploh obravnavana. Ker uporabna vrednost splošnega robnega pogoja še ni znana, in bi splošnost izrazito povečala zahtevnost algoritma za iskanje predslik, so tukaj vsi robovi obravnavani enako, z utežjo ena. Temu bomo rekli odprti rob, ker ta ne definira nobenih omejitev, katere zvezne ploskve v mreži predslik so dovoljene in katere ne.

Obstaja še en enostaven robni pogoj, ki je definiran za ciklično sklenjena končna polja CA. Ta tip robnega pogoja tukaj ne bo obravnavan, ker še dodatno poveča kompleksnost algoritmov. Potrebno bi bilo namreč opraviti celoten postopek iskanja predslik za vsak začetni rob posebej. Zvezna ploskev je predslika v cikličnem polju le, če se njen začetni in končni rob ujemata. Pojma začetnega in končnega roba sta opisana v poglavju o algoritmu.

## Poglavje 4

# Algoritem za štetje in izpis predslik

Pri 1D CA ima algoritem za štetje predslik linearno  $O(N)$  procesno in pomnilniško zahtevnost. Posplošeno to pomeni, da se vsaka celica pojavi v izračunu samo enkrat. Dejansko ima vsak algoritem tudi logaritmčno komponento, saj število bitov, potrebnih za zapis števec, raste logaritmčno s številom celic. Algoritem za izpis predslik ima neizogibno eksponentno kompleksnost  $O(C^N)$ , saj maksimalno in povprečno število predslik raste eksponentno v odvisnosti od števila celic.

Za 2D CA se je izkazalo, da obstajajo problemi, ki niso rešljivi z linearno kompleksnostjo. Prikazani algoritem ima maksimalno eksponentno kompleksnost v odvisnosti od posamezne dimenzije  $O(S^{N_x} S^{N_y})$ . Za sedaj je še odprta možnost za obstoj algoritma s polinomske kompleksnostjo, ki je nekje med linearno in eksponentno.

Opisani algoritem za štetje predslik razdeli polje celic na vrstice v dimenziji X. Delitev po stolpcih v dimenziji Y bi bila ekvivalentna, tako da je to arbitrarna odločitev. S stališča procesne zahtevnosti je najbolje izbrati krajšo dimenzijo. Na podlagi zunanjega robnega pogoja algoritem najprej poišče nabor predslik za prvo vrstico. Nabor predslik izračuna kakor uteži za robove na nasprotni strani od začetne. Seznam robov in njihove uteži uporabi kakor vhodni robni pogoji za naslednjo vrstico. Robne uteži, izračunane za zadnjo vrstico, predstavljajo število vseh predslik.

Algoritem za izpis predslik je nadaljevalje algoritma za štetje. Začne z znanim številom predslik, ki ga je dalo štetje, in izpisuje predslike po vrsticah v obratni smeri, kakor je potekalo štetje (od zadnje do prve vrstice). Pridobljen seznam predslik je sortiran. Smer sortiranja je odvisna od smeri procesiranja. Smer procesiranja je možno po potrebi spremeniti.

### 4.1 Procesiranje v eni dimenziji

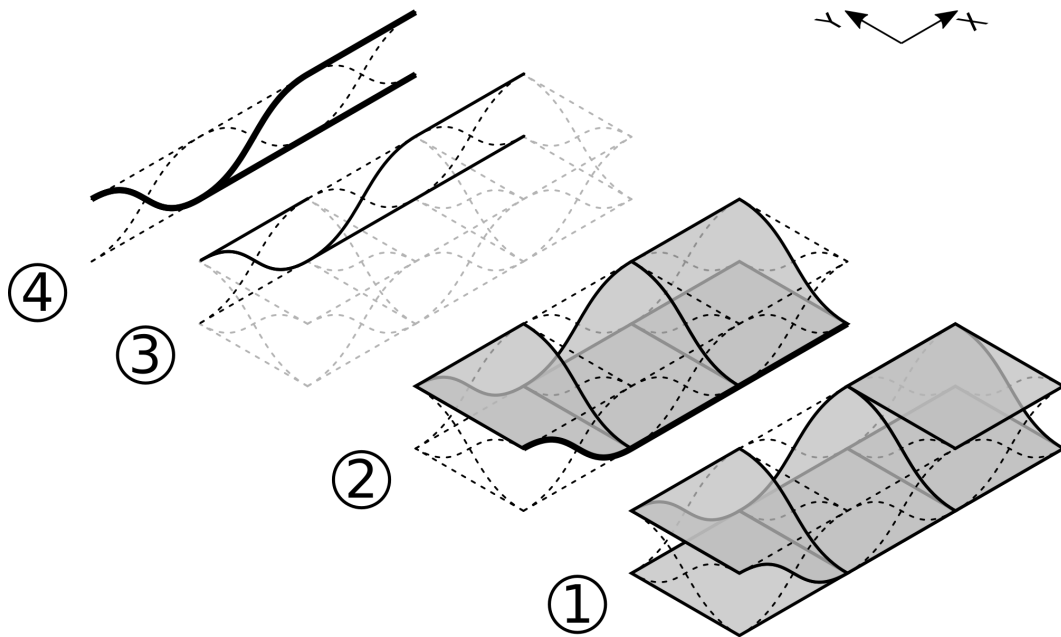
Procesiranje se začne z 1D nizom celic (vrstico). Vsaki celici v nizu pripada lastna mreža predslik, eno dimenzionalnemu nizu celic posledično pripada povezan niz mrež predslik (slika 4.1 mreža 1). Vsak segment mreže v nizu ima svoj nabor veljavnih okolic, ki je

definiran s tranzicijsko funkcijo (ali pa je posplošeno poljuben nabor). Na začetku se ploskve okolic sosednjih celic še ne povezujejo v zvezno ploskev čez celoten niz. Izločiti je potrebno vse ploskve okolic, ki se ne povezujejo z okolicami svojih sosednjih celic.

Ker se vsaka vrstica povezuje s predhodno in naslednjo vrstico, je potrebno upoštevati tudi zveznost ploskev na prehodu med vrsticami. Ta povezava med vrsticami je prerez opazovane ploskve na meji med vrsticama. Vsako ploskev se obravnava posebej in prerez se izrazi kakor robni pogoj. Začetni robni pogoj za vsako vrstico je nabor poti med vozlišči na začetku vrstice. Vsaka pot je obravnava posebej (slika 4.1, mreža **2**, odebeljena pot). Začetni robni pogoj je ekvivalenten nizu prekrivanja okolic  $o_{y-}$ .

Začetni robni pogoj se aplicira tako, da se izloči iz obravnave vse ploskve, ki nimajo skupnega roba z robno potjo (slika 4.1, prehod iz mreže **1** v mrežo **2**). Po tem koraku, so vse ploskve definirane v dveh od štirih vozlišč. Do nezveznosti lahko prihaja le še na robu, nasprotnem začetnemu robu.

Problem se reducira iz 3D mreže in procesiranja ploskev v 2D mrežo, kjer se procesirajo poti med vozlišči (slika 4.1, prehod iz mreže **2** v mrežo **3**). To je problem ekvivalenten iskanju predslik (zvenih poti v grafu) za 1D CA, za kar se uporabi algoritem, opisan v [25]. Rezultat so poti na prerezu med ploskvami (slika 4.1, mreža **4**, končni robni pogoj je odebeljen). Končni robni pogoj je ekvivalenten nizu prekrivanja okolic  $o_{y+}$ .



Slika 4.1: Mreža vrstice velikosti  $N_x = 3$  za binarni CA z okolico quad. Nabor okolic/ploskev za posamezno celico je arbitraren. Izbran je tako, da poudari korake algoritma.

Po procesiranju nabora vseh začetnih robnih pogojev nastane nabor vseh končnih robnih pogojev. Ta nabor se v naslednjem koraku uporabi kakor začetni robni pogoj za naslednjo vrstico.

Nabor vseh možnih poti na meji vrstice raste eksponentno z dolžino vrstice. Število vseh poti se izračuna iz števila celic, ki se prekrivajo med okolicama dveh vrstic. Za deljenje po

vrsticah v dimenziji  $X$  je ta nabor velikosti  $|S|^{(M_y-1)(M_x-1+N_x)}$ . Posledično kompleksnost algoritma raste eksponentno z dolžino vrstice  $O(C^{N_x})$ .

## 4.2 Procesiranje v drugi dimenziji

V drugi dimenziji se procesira zaporedje vrstic od prve do zadnje (slika 4.2). Vsaka vrstica se začne in konča z robnim pogojem. Ti robni pogoji so prerezi ploskev v celotni mreži predslik. Vsako vrstico je potrebno v drugi dimenziji procesirati le enkrat, kar pomeni, da je računska zahtevnost linearno odvisna od števila vrstic  $O(N_y)$ .

Vsakemu robnemu pogoju je pripisana utež. Začetni robovi za prvo vrstico imajo pripisano utež  $w = 1$ . Na splošno se lahko več začetnih robov preslika v isti končni rob ene vrstice. Utež, pripisana končnemu robu dane vrstice, je vsota uteži vseh začetnih robnih pogojev, ki se preslikajo vanj. Utež tako predstavlja število vseh predslik za dano in prejšnje vrstice, ki se končajo z opazovanim robom. Robovi, s katerimi se ne konča nobena od predslik, dobijo utež  $w = 0$ .

Za tem, ko so procesirane vse vrstice, je znan končni rob nabora vseh zveznih ploskev na celotni mreži predslik. Vsota uteži vseh končnih robov daje število predslik celotne mreže.

Z uporabo Boolove algebre namesto operacij množenja in seštevanja je možno poenostaviti iskanje predslik samo na njihov obstoj za opazovani robni pogoj. Uteži postanejo binarne vrednosti  $w \in \{0, 1\}$ .

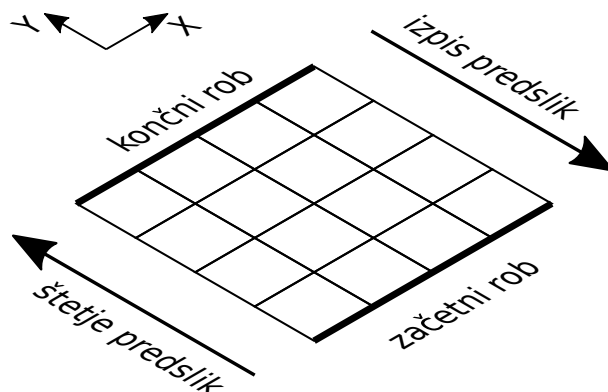
## 4.3 Izpis predslik

Ni nujno, da se vsak vrstični začetni pogoj preslika v nabor končnih pogojev. Možno je, da nobena pot na končnem robu ne zadošča začetnemu pogoju in mreži predslik. Torej po prejšnjih korakih še ni točno določeno, katere ploskve se združujejo v zvezno celoto in katere ne. Možne so ploskve, ki prekrivajo polje samo do neke vrstice in ne naprej.

Procesiranje po drugi dimenziji poteka v obratni smeri kakor pri štetju; začne pri zadnji vrstici in konča pri prvi vrstici (slika 4.2). Z vsakim korakom se izloča še preostale slepe poti iz mreže predslik. Za vsako vrstico je potrebno ponovno rešiti 1D problem, le da sta začetni in končni rob zamenjana. Začetni rob je zaporedje prekrivanja okolice  $o_{y+}$  in končni rob zaporedje prekrivanja okolice  $o_{y-}$ .

Hkrati je možno še izpisovati predslike. Že na začetku procesiranja v obratni smeri je znano število vseh predslik, kar omogoča rezervacijo pomnilnika in inicializacijo seznama predslik. Končne robne poti in njihove uteži se uporabijo za popis tekoče vrstice celic v predslikah. Indeks robne poti (določeno zaporedje prekrivajočih okolice) da vrednost celic v vrstici predslike. Uteži, izračunane v smeri štetja, pa dajo število predslik, ki jih je potrebno popisati z dano vrednostjo za vrstico celic.

Z vsakim korakom v obratni smeri se popiše nova vrstica za vsako od prešteti pred-slik. Ko pride algoritem spet do začetne vrstice, so vse pred slike popisane. Hkrati so iz mreže pred slik izločene vse slepe poti; ostanejo le še ploskve, ki tvorijo zvezne ploskve na celotnem polju celic.



Slika 4.2: Potek smeri procesiranja pri algoritmu za štetje in izpis pred slik.

Algoritem tukaj ni podrobno opisan, je pa preprosta razširitev algoritma uporabljenega za 1D CA, opisanega v [25]. Za podrobnosti je na voljo izvorna koda [21].

## 4.4 Nezmožnost procesiranja z linearno zahtevnostjo

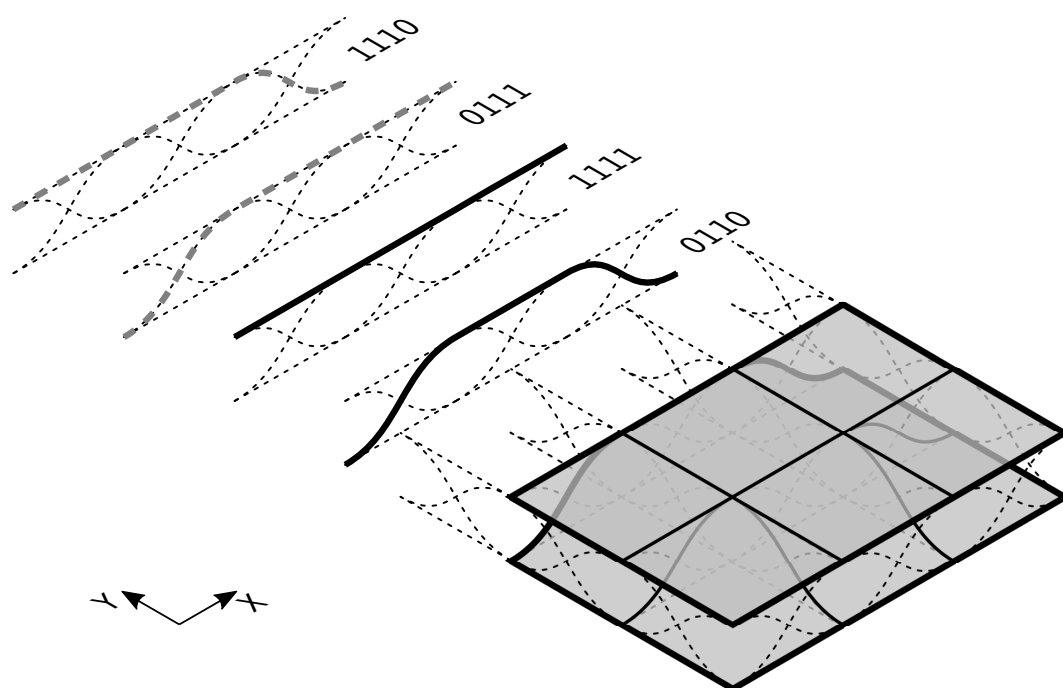
Podan je primer, ki kaže, zakaj procesiranje z linearno zahtevnostjo ni mogoče.

Cilj raziskovalnega dela za to raziskovalno nalogo je bil poiskati učinkovit algoritem za iskanje pred slik 2D CA. Na podlagi izkušenj z 1D CA sem optimistično pričakoval, da bo možno problem rešiti v linearnem času.

Algoritem v linearnem času bi vsako celico obravnaval le enkrat; na splošno bi bilo število obravnav majhna konstanta (4-krat, če se izvaja procesiranje skozi mrežo pred slik v 4 smereh/prehodih). Tak algoritem predpostavlja, da je možno celoten nabor začetnih robnih pogojev upoštevati hkrati v enem samem prehodu. V koraku **2** pri procesiranju vrstice (slika 4.1) bi se izločilo le ploskve, ki ne ustrezajo nobenemu od začetnih robnih pogojev. Po nekaj poizkusih sem ugotovil, da tak algoritem ne deluje pravilno. Vsaj nekatere poti iz nabora je potrebno upoštevati ločeno.

V podanem primeru (slika 4.3) se prvi dve vrstici zaključita z naborom dveh končnih poti **0110** in **1111**. Če bi ti dve poti uporabili hkrati, kakor začetni robni pogoj za naslednjo vrstico, bi dobili enak rezultat, kakor če bi bili del robnega pogoja tudi poti **0111** in **1110**. To pa zato, ker algoritem, ki poti ne obravnava ločeno, ne more vedeti, kje se je pot začela, da bi jo lahko tudi pravilno zaključil. Posledično linearen algoritem lahko vzame začetek ene od poti in ga na vozlišču, skupnem obema potema, nadaljuje po drugi poti. Iz mreže je razvidno, da ti dve poti nista preseka neke zvezne ploskve. Skratka, hkratno obravnavanje celotnega nabora robnih pogojev ni možno, kar pomeni, da je število obravnav neke celice odvisno od števila poti in posledično od eksponenta števila celic v vrstici.

Možno je, da obstajajo drugačne optimizacije algoritma, ki lahko zmanjšajo procesno kompleksnost.



Slika 4.3: Ovira za procesiranje z linearno zahtevnostjo. Veljavni robni pogoj je odebeljen, neveljavni pa črtkan.



# Poglavje 5

## Primerjava z znanimi algoritmi

Obstoječi algoritmi za predslike 2D CA se osredotočajo izključno na GoL. Največ algoritmov je namenjenih iskanju stanj GoE, skratka, preverjajo le obstoj predslik in jih ne štejejo ali izpisujejo. Tukaj opisani algoritem je zamišljen bolj univerzalno za poljuben 2D CA. Vendar univerzalnost v praksi ni ravno pomembna, saj izven raziskovalcev GoL ni dosti zanimanja za teoretične probleme, kakor je iskanje predslik.

### 5.1 Iskanje stanj GoE za GoL

Conway je leta 1970 predstavil CA po imenu GoL. Že leta 1971 sta Roger Banks in Steve Ward predstavila prvo stanje GoE velikosti  $9 \times 33$  celic. Odkritje je bilo objavljeno v [38]. Kasneje je Don Woods [38, 39] s pomočjo računalnika dokazal, da je stanje res GoE.

Woodsov algoritem, tako kakor moj, deluje na vrsticah. Začne z eno vogalno celico, katere nabor predslik je kar nabor okolic, ki se preslikajo v stanje celice. V naslednjem koraku opazuje sosednjo celico v vrstici. Za vse kombinacije okolic dodane celice in obstoječi nabor predslik preveri, če se ujemajo, kjer se okolice prekrivajo. Če se ujemajo, doda sestavljeno predsliko v nabor, sicer predslika izpade iz nabora. Ko pride algoritem do konca vrstice, nadaljuje v naslednji vrstici, le da je sedaj prekrivanje med naborom predslik in okolico dodane celice drugačno (več sosednjih celic namesto ene). Tako algoritem nadaljuje do konca celotne konfiguracije. Če je končni nabor predslik prazen, je opazovana konfiguracija GoE. Nabor predslik in s tem poraba pomnilnika in procesnega časa ostajajo relativno majhni v primerjavi z naborom vseh stanj. Temu je tako, ker imajo v praksi tudi deli konfiguracije GoE malo predslik. Algoritem bi bil za konfiguracije z dosti predslikami veliko bolj pomnilniško in časovno potraten.

Nicolay Beluchenko je poiskal večje število stanj GoE velikosti  $11 \times 11$ . Sicer ne poznam vseh njegovih pristopov, ampak vsaj v enem primeru je uporabljal podoben algoritem kakor Woods. Začel je z osrednjo celico in dodajal nove celice v spirali okoli osrednje. Za vrednost dodane celice je uporabil tisto, ki je dala manjši nabor predslik. Algoritem zaključil, ko z novo dodano celico ni več možno tvoriti predslik.

Leta 1974 je Duparc [16, 17] razvil drugačen algoritem, ter z njim poiskal stanja GoE velikosti  $6 \times 122$  in  $6 \times 117$ . Algoritem uporablja teorijo končnih avtomatov in regularnih jezikov, ki je v osnovi namenjena 1D sistemom. Duparc je celice iz vrstice 2D polja združil v simbole regularnega jezika, zaporedje več vrstic pa predstavlja besedo. Duparcov algoritem ne obravnava vseh stanj vrstice enakovredno, temveč se osredotoča na vrstice, ki zmanjšujejo nabor predslik. Obravnavanje vseh stanj vrstic bi namreč preseglo pomnilniške sposobnosti takratne in sedanje strojne opreme že za kratke vrstice. Dokaže le, da za vrstice dolžine 1 ne obstaja stolpec, ki bi bil stanje GoE. Originalni članek je v francoščini, tako da lahko algoritem opišem le na podlagi tega, kako ga opisujejo drugi.

V zadnjih letih podoben algoritem uporablja Steven Eker (CSL, SRI International, California, USA). Dokazal je, da za pravokotne konfiguracije višine 2 in 3 ne obstajajo stanja GoE. Dokaz za konfiguracije višine 4 z danim algoritmom še vedno presega sposobnosti sodobne strojne opreme. Število stanj avtomata, ki opisuje jezik GoE, namreč raste eksponentno z višino. Elker je poiskal stanja GoE velikosti  $9 \times 11$ ,  $8 \times 12$  in  $5 \times 83$ . Na žalost raziskovalcu delodajalec prepoveduje objavo rezultatov in podrobnosti, dokler ni opravljen interni pregled. Tako imamo za sedaj na voljo le zgoraj navedene podatke.

Drugačen algoritem uporablja Marijn Heule [18]. S sodelavci je zapisal vsa prekrivanja za določeno stanje v obliki binarnih enačb. Te je nato predal orodju za reševanje SAT problemov. Če rešitev ne obstaja, pomeni, da je stanje GoE. Uporabljena je dodatna predpostavka, da bo rešitev zrcalno simetrična v obeh dimenzijah, kar občutno zmanjša velikost problema. Algoritem je še dodatno optimiziran za iskanje stanj GoE in omogoča pregled večjega števila podobnih stanj. Ob spremembi vrednosti ene same celice omogoča uporabo delnih rezultatov prejšnjega izračuna. SAT metode ne poznam dobro, tako da ne morem primerjati procesne zahtevnosti.

## 5.2 Iskanje predslik GoL na splošno

Erlan [14] je predstavil igro, pri kateri se za dano stanje GoL išče predslike. To dejansko ni implementacija algoritma za iskanje predslik, je pa vzpodbudil druge k iskanju takega algoritma.

Spletna stran `kaggle.com` je pripravila tekmovanje [1], kjer so morali kandidati reševati problem iskanja predslik. Podali so nabor stanj, za katere je treba poiskati predslike, in nabor učnih sekvenc. Pričakovali so rešitev, ki temelji na strojnem učenju ali optimizaciji, torej fokus ni bil na eksaktni rešitvi.

Pri iskanju implementacij algoritma, sem našel Atabot [12], Celični kronometer [13] in program baziran na SAT metodi [31]. Cilj teh algoritmov sicer je iskanje predslik, ampak iskanje ni sistematično in cilj ni nabor vseh možnih predslik. Algoritmi se tudi poslužujejo določene hevrstike pri iskanju.

### 5.3 Polni algoritmi za štetje in izpis predslik

Opisani algoritmi za iskanje stanj GoE, na primer Woods, sicer omogočajo iskanje vseh predslik, ampak to ni njihov glavni namen. Optimizirani so le za stanja GoE, kjer je predslik malo tudi za samo del konfiguracije GoE. Predvsem niso optimizirani za splošen primer, kjer je lahko predslik veliko, posledično so potratni s porabo pomnilnika. Morda bi bilo možno optimizirati porabo pomnilnika, tako da bi algoritem hranil le tekoče razlike med predslikami, namesto da jih hrani v celoti. Nekaj podobnega počne moj algoritem, ko šteje le robove, in ne hrani celotnih predslik.

Našel sem le eno aplikacijo, ki je namenjena štetju in izpisu vseh predslik. Neil Bickford [11] prav tako kakor ostali procesira konfiguracijo po vrsticah, znotraj vrstice pa lahko uporablja različne algoritme (Woods, Duparc). Bickford opiše tudi algoritem, ki namesto procesiranja po vrsticah združuje predslike kvadratnih podsklopov konfiguracij. Na žalost poda primerjavo med različnimi algoritmi le kakor procesni čas za posamezno implementacijo, ne pa tudi bolj splošne ocene maksimalne in povprečne procesne/pomnilniške zahtevnosti.

Našel sem še aplikacijo, napisano v jeziku APL [19]. Dokumentacija te aplikacije ni dovolj jasna, da bi bilo očitno, kaj točno počne. Iz primera pa sklepam, da je sposobna poiskati vse predslike.

# Poglavje 6

## Sklepne ugotovitve

Probleme, povezane z iskanjem predslik, lahko delimo v nivoje glede na zahtevnost:

1. določitev, ali obstajajo predslike za dano trenutno stanje sistema
2. štetje predslik za dano trenutno stanje sistema
3. naštevanje konfiguracij predslik
4. vprašanje reverzibilnosti sistema
5. jezik vseh stanj GoE

V magistrskem delu opišem algoritem, ki rešuje prve tri probleme. Preostala problema sta si sorodna. Problem reverzibilnosti je na splošno dokazano nerešljiv [26]. Problem jezika stanj GoE je za 1D preprosto rešljiv, za 2D pa je verjetno podobno kakor vprašanje reverzibilnosti nerešljiv.

### 6.1 Analiza 2D CA s pomočjo končnih avtomatov

Problem jezika stanj brez predslik bi potreboval teorijo 2D formalnega jezika, ki še ne obstaja. Kljub temu je možno zastaviti probleme, ki se jih da preslikati v končni avtomat. Na primer, če obravnavamo vrstice kakor simbole jezika, lahko sestavimo končni avtomat. Simboli (stanja vrstic) definirajo prehode med stanji končnega avtomata. Stanja avtomata pa so vsi možni nabori robov vrstice. Pomembna je le prisotnost roba, ne pa tudi utež. Poln nabor vsebuje vse možne robove, prazen nabor pa ne vsebuje nobenega roba. Vsako zaporedje vrstic (simbolov), ki v avtomatu pelje iz polnega v prazno stanje, je konfiguracija GoE. Zanke v takem avtomatu pa so sorodne delcem.

Opisani pristop postane nepraktičen že za kratke vrstice. Število vseh možnih robov raste eksponentno z dolžino vrstice  $N_x$ . To število,  $|S|^{(M_y-1)(M_x-1+N_x)}$ , je enako številu vseh načinov, na katere se lahko okolici dveh vrstic prekrivata. Stanja avtomata pa so vsi možni nabori robov. Nabor lahko opišemo kakor binarni niz, kjer vsak bit pomeni prisotnost ali odsotnost enega od robov. Vseh stanj končnega avtomata je tako  $2^{|S|^{(M_y-1)(M_x-1+N_x)}}$ . Število stanj avtomata postane neobvladljivo že za kratke vrstice.

Najkrajša beseda GoE znotraj regularnega jezika stanj GoE je tista, ki po najkrajši poti in brez zank pripelje iz polnega v prazen nabor. Pri 1D CA se je izkazalo, da je dolžina najkrajše besede GoE povezana s kompleksnostjo prostorsko-časovnih vzorcev, ki se pojavljajo pri danem pravilu. Podobno so verjetno tudi pri 2D CA konfiguracije GoE večje pri pravilih, ki omogočajo kompleksno dinamiko delcev. Verjetno bi bilo možno to izkoristiti za iskanje zanimivih pravil v okolici quad.

## 6.2 Izboljšave podanega algoritma

Implementacijo algoritma je možno izboljšati, tako da zahtevnost ne raste tako hitro z velikostjo problema. Za sedaj vidim samo rešitev, ki se osredotoča na povprečno zahtevnost, namesto na maksimalno zahtevnost. Na primer, sedaj algoritem rezervira pomnilnik za uteži vseh možnih robov, dovolj pa bi bilo, če bi bil rezerviran pomnilnik samo za robove z utežjo večjo od nič. Sploh pri iskanju stanj GoE je robov malo v primerjavi z vsemi možnimi. Tak pristop bi dal dosti manjšo porabo pomnilnika in verjetno tudi znižal čas procesiranja.

Morda bi bilo možno vnaprej določiti zgornjo mejo števila ne-ničelnih robov. To bi omogočalo zgodnjo oceno, če je izračun za dan problem izvedljiv. Na primer, če bi v vrstičnem procesiranju obravnavali vse začetne robove hkrati, namesto vsakega posebej, bi dobili prevelik nabor končnih robov. Tak nabor bi vedno vseboval vse pravilne končne robove, in še nekaj napačnih. Ampak skupno število bi bilo lahko občutno manjše od polnega nabora. To število bi lahko uporabili za oceno porabe pomnilnika in časa. Na ta način bi bilo možno reševati probleme, ki so na meji tega, kar zmore sodobna strojna oprema.

Samo implementacijo algoritma je tudi možno optimizirati. Univerzalnost algoritma, predvsem univerzalnost velikosti okolice, prispeva dosti režijskih stroškov. Če bi bil algoritem specializiran samo za quad ali GoL, bi se lahko izognil precej operacijam. Podobno velja za nabor stanj celice. Za binarne CA je možno optimizirati operacije, ki pretvarjajo 2D nize celic v cela števila in obratno. Binarne podatke je možno kompaktno pakirati v pomnilnik. Mogoče je tudi uporabiti logične operacije nad celo besedo celic, namesto procesiranja vsake celice posebej.

Procesiranje v algoritmu poteka v gnezdenih zankah in v znanem vrstnem redu. Le manjši del dostopov do pomnilnika je naključen. Zaporednost podatkov že sama po sebi omogoča dobro optimizacijo dostopa do predpomnilnika, tako da na tem področju ni dosti očitnih optimizacij.

## 6.3 Dokazovanje

Magistrska naloga je bolj skopa z dokazi. Za 1D CA sem uporabljal neposredno dokaze iz teorije grafov, pri 2D CA pa mreža predslik ni običajen graf, tako da dokazovanje s teorijo grafov ni tako preprosto. Verjetno pa je možno pretvoriti dani graf v dualno obliko s transformacijo zvezda-trikot. Če obstaja dualna oblika grafa, kjer ni ploskev, temveč

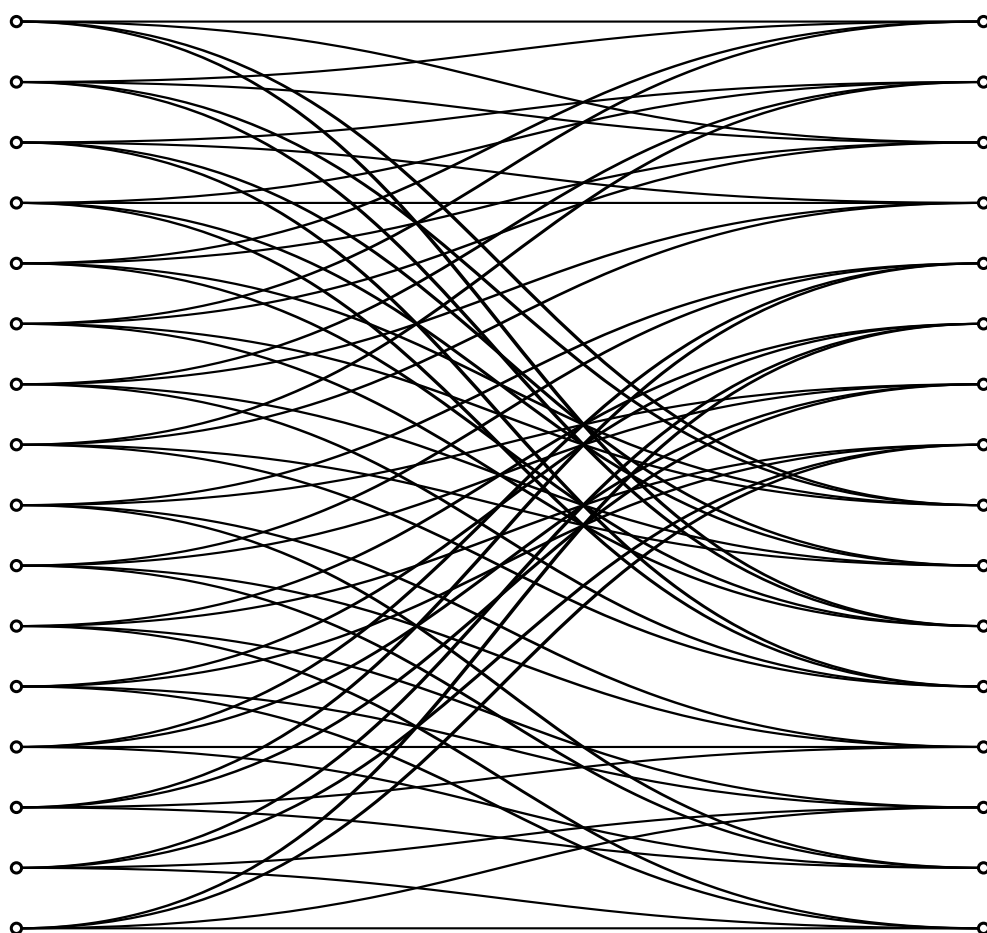
samo vozlišča in povezave, bi jo lahko uporabili za izpeljavo in dokazovanje dejanske računske kompleksnosti problema.

Nekajkrat sem preveril rezultate, ki jih daje implementacija algoritma. Za dano vhodno stanje sem vsako predsliko s tranzicijsko funkcijo pretvoril nazaj v sedanost in rezultat primerjal z vhodnim stanjem. Ker je moj algoritem vračal za dva velikostna razreda več predslik, kakor referenčna aplikacija RetroGUI [11], sem tudi preveril, če so vse našteje predslike enolične. Za nekaj manjših primerov, sem tudi poiskal predslike z brute force metodo. Na ta način nisem našel nobene napake v algoritmu.

## Dodatek A

### De Bruijnov diagram za GoL

Slika A.1 prikazuje del De Bruijnovega diagrama za GoL. Graf s 16 vozlišči in 64 povezavami je očitno preveč kompleksen, da bi bil primeren za razlago algoritma. Število vseh možnih okolice (ploskev v diagramu) pa je celo 512.



Slika A.1: Mreža ene celice za binarni CA z Moorovo okolico  $M_x = M_y = 3$ .

## Dodatek B

# Primer delovanja algoritma za CA z okolico quad

V naslednjem primeru je prikazano štetje in iskanje predslik za binarni CA (celica ima lahko 2 stanji) z okolico velikosti  $2 \times 2$  in pravilom številka 0x725A.

Majhna konfiguracija velikosti  $2 \times 2$  je shranjena v datoteki `test_2x2.cas`. Uporabil bi večjo konfiguracijo, a na žalost ne poznam pravila, ki bi za večjo konfiguracijo dal majhno število predslik.

Uporabljen je program [21] z verzijo v1.

```
$ ./ca2d_preimages 2 2 2 0x725a 2 2 test_2x2.cas
```

CA parameters:

```
sts   : 2
ngb   : siz={2,2}, a=4, n=16
ovl-y: siz={1,2}, a=2, n=4
ovl-x: siz={2,1}, a=2, n=4
rem-y: siz={1,2}, a=2, n=4
rem-x: siz={2,1}, a=2, n=4
ver   : siz={1,1}, a=1, n=2
shf-y: siz={1,1}, a=1, n=2
shf-x: siz={1,1}, a=1, n=2
```

RULE = 0x725a

```
tab [0] = [[0,0],[0,0]] = 0
tab [1] = [[1,0],[0,0]] = 1
tab [2] = [[0,1],[0,0]] = 0
tab [3] = [[1,1],[0,0]] = 1
tab [4] = [[0,0],[1,0]] = 1
tab [5] = [[1,0],[1,0]] = 0
tab [6] = [[0,1],[1,0]] = 1
tab [7] = [[1,1],[1,0]] = 0
```



```

tab [8] = [[0,0],[0,1]] = 0
tab [9] = [[1,0],[0,1]] = 1
tab [A] = [[0,1],[0,1]] = 0
tab [B] = [[1,1],[0,1]] = 0
tab [C] = [[0,0],[1,1]] = 1
tab [D] = [[1,0],[1,1]] = 1
tab [E] = [[0,1],[1,1]] = 1
tab [F] = [[1,1],[1,1]] = 0

```

CA configuration siz [y,x] = [2,2]:

```

1 0
0 1

```

NETWORK: edge weights from forward/backward direction,  
summed preimages

```

net [dy=0][y=0] = [1 1 1 1 1 1 1 1 ]
net [dy=0][y=1] = [2 2 0 2 3 3 0 1 ]
net [dy=0][y=2] = [0 3 5 0 0 2 7 2 ]   cnt [0] = 19
net [dy=1][y=0] = [0 4 4 0 0 4 5 2 ]   cnt [1] = 19
net [dy=1][y=1] = [2 0 3 4 2 0 1 1 ]
net [dy=1][y=2] = [1 1 1 1 1 1 1 1 ]

```

preimage i=0: CA configuration siz [y,x] = [3,3]:

```

1 0 0
0 0 0
0 1 0

```

preimage i=1: CA configuration siz [y,x] = [3,3]:

```

1 0 0
0 0 0
0 1 1

```

preimage i=2: CA configuration siz [y,x] = [3,3]:

```

1 0 0
0 0 1
0 1 0

```

preimage i=3: CA configuration siz [y,x] = [3,3]:

```

1 0 0
0 0 1
0 1 1

```

preimage i=4: CA configuration siz [y,x] = [3,3]:

```

0 1 0
1 1 0
1 0 0

```

preimage i=5: CA configuration siz [y,x] = [3,3]:

```

0 1 0
1 1 0
1 0 1

```

preimage i=6: CA configuration siz [y,x] = [3,3]:

```

0 1 0
1 1 0
0 1 1

```

preimage i=7: CA configuration siz [y,x] = [3,3]:

```

0 1 0
1 1 0
1 1 1

```

preimage i=8: CA configuration siz [y,x] = [3,3]:

```

1 0 1
0 0 0
0 1 0

```

preimage i=9: CA configuration siz [y,x] = [3,3]:

```

1 0 1
0 0 0
0 1 1

```

preimage i=10: CA configuration siz [y,x] = [3,3]:

```

1 0 1
0 0 1
0 1 0

```

preimage i=11: CA configuration siz [y,x] = [3,3]:

```

1 0 1
0 0 1
0 1 1

```

preimage i=12: CA configuration siz [y,x] = [3,3]:

```

0 1 1
1 1 0
1 0 0

```

preimage i=13: CA configuration siz [y,x] = [3,3]:

```

0 1 1
1 1 0
1 0 1

```

preimage i=14: CA configuration siz [y,x] = [3,3]:

```

0 1 1
1 1 0
0 1 1

```

```
preimage i=15:  CA configuration siz [y,x] = [3,3]:  
0 1 1  
1 1 0  
1 1 1
```

```
preimage i=16:  CA configuration siz [y,x] = [3,3]:  
0 1 1  
1 1 1  
1 0 0
```

```
preimage i=17:  CA configuration siz [y,x] = [3,3]:  
1 1 1  
0 0 1  
0 1 0
```

```
preimage i=18:  CA configuration siz [y,x] = [3,3]:  
1 1 1  
0 0 1  
0 1 1
```

# Dodatek C

## Opis algoritma Dona Woodsa

Prilagam pisno izmenjavo z avtorjem <sup>1</sup> prvega algoritma za iskanje predslik GoL.

```
Don Woods <don@icynic.com>          Thu, Jul 28, 2016 at 11:45 PM
To: iztok.jeras@gmail.com
> Dear Mr. Woods,
>
> I am developing an algorithm for computing preimages of 2D CA. It is based
> on the latest research for computing preimages of 1D CA.
> I am using the 2x2 quad neighborhood in the examples instead of the 3x3
> neighborhood used in Life.
> https://github.com/jeras/fri-magisterij (work in progress and Slovene
> language)
> https://github.com/jeras/preimages-2D (not working yet)
>
> As part of my masters thesis I have to compare my algorithm to existing
> ones.
> You are known as the first person to prove a pattern to be a Garden of Eden
> orphan in the Game of Life.
> Could you send me some references to this proof? I was looking for an
> article to reference, but a generic description (preferably written by you)
> or source code would be as good.
>
> Regards,
> Iztok Jeras
>
```

Oof, it's been quite a while; let me see what I can remember.

The "proof" was programmatic, i.e. I wrote a program to compute predecessors to a given position, and the program claimed there were no predecessors to the particular position that someone had theorised was

---

<sup>1</sup>Don Woods /url<http://www.icynic.com/don/>

a Garden of Eden. I did not have a formal proof of correctness for the program. (I was a teenager when I wrote it!) I don't think I still have the source code anywhere, I'm afraid. (I might have a paper printout of it somewhere but don't have time to search right now.)

The program worked by considering each cell in turn and looking at all possible 3x3 predecessors that would produce the given state. It then looked at the next cell to the right, looking for 3x3 predecessors that were consistent with the 6 overlapping cells from the other, etc. (In practice, the 3x3 neighborhoods were indexed so that I could quickly find which of the 8 possible righthand extensions worked.) When it had a  $3 \times (N+2)$  predecessor for the top row of  $N$ , it continued at the left of the next row, then across that row, etc. Obviously in those later rows there was only one new predecessor cell being introduced for each new current cell after the leftmost, so the combinatoric explosion was not excessive, particularly since the GoE pattern by design did not have many potential predecessors even for subsections. I also deliberately oriented the target pattern with the short dimension going across, so that I more quickly reached the point of extending the predecessor one cell at a time in later rows.

My program normally included a 1-wide border around the area being backtracked, to ensure there were no artifacts introduced, but for the GoE proof I restricted it to the non-empty rectangular boundary, which I think was 9 by 33? Thus, as I think was mentioned at the time, what I actually demonstrated was a stronger result: there was no predecessor that led to a state that included the GoE as a 9x33 subsection.

I hope that helps. Let me know if you have further questions.

-- Don Woods

Don Woods <don@icynic.com> Tue, Aug 16, 2016 at 6:42 PM  
 To: iztok.jeras@gmail.com  
 > I found the time to read your response just now.  
 >  
 > Your description of the algorithm confirms what I imagined based on third  
 > party descriptions.  
 > Your details regarding combinatorial explosion which affects memory  
 > consumption and processing time were extra helpful.  
 >  
 > May I cite this email in my master thesis, since there are no other direct  
 > sources I would know of?  
 >  
 > Regards,  
 > Iztok Jeras

Permission granted. Good luck with your thesis!

-- Don Woods

# Slike

1.1	Gosperjeva pištola. . . . .	6
1.2	Trk delcev v pravilu 110. . . . .	6
1.3	Langtonova zanka. . . . .	7
1.4	Mreža predslik 1D CA. . . . .	7
1.5	Atraktorjevo korito. . . . .	8
2.1	Okolica 1D CA. . . . .	12
2.2	Prekrivanje okolic 1D CA. . . . .	12
2.3	Indeksiranje okolice $3 \times 3$ . . . . .	15
2.4	Indeksiranje okolice $2 \times 2$ . . . . .	16
2.5	Velikost okolice. . . . .	16
2.6	Prekrivaje okolic $3 \times 3$ v smeri dimenzij X in Y. . . . .	17
2.7	Prekrivaje okolic $2 \times 2$ v smeri dimenzij X in Y. . . . .	17
2.8	Prekrivanje okolic $3 \times 3$ - diagonalno. . . . .	18
2.9	Prekrivanje okolic $2 \times 2$ - diagonalno. . . . .	18
3.1	De Bruijnov graf, pravilo 110. . . . .	22
3.2	Mreža ene celice. . . . .	22
3.3	Nabor ploskev. . . . .	23
3.4	Mreža polja celic. . . . .	24
4.1	Algoritem procesiranja vrstice. . . . .	27
4.2	Algoritem za izpis predslik. . . . .	29
4.3	Ovira za procesiranje z linearno zahtevnostjo. . . . .	30
A.1	Mreža ene celice za GoL. . . . .	37

# Literatura

- [1] Conway's Reverse Game of Life. <https://www.kaggle.com/c/conway-s-reverse-game-of-life>, Marec 2013.
- [2] Wikipedia: Backtracking. <https://en.wikipedia.org/wiki/Backtracking>, Avgust 2016.
- [3] Wikipedia: Boolean satisfiability problem. [https://en.wikipedia.org/wiki/Boolean\\_satisfiability\\_problem](https://en.wikipedia.org/wiki/Boolean_satisfiability_problem), Avgust 2016.
- [4] Wikipedia: Brute-force search. [https://en.wikipedia.org/wiki/Brute-force\\_search](https://en.wikipedia.org/wiki/Brute-force_search), Avgust 2016.
- [5] Wikipedia: Conway's Game of Life. [https://en.wikipedia.org/wiki/Conway%27s\\_Game\\_of\\_Life](https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life), Avgust 2016.
- [6] Wikipedia: De Bruijn graph. [https://en.wikipedia.org/wiki/De\\_Bruijn\\_graph](https://en.wikipedia.org/wiki/De_Bruijn_graph), Avgust 2016.
- [7] Wikipedia: Depth-first search. [https://en.wikipedia.org/wiki/Depth-first\\_search](https://en.wikipedia.org/wiki/Depth-first_search), Avgust 2016.
- [8] Wikipedia: Langton's loops. [https://en.wikipedia.org/wiki/Langton%27s\\_loops](https://en.wikipedia.org/wiki/Langton%27s_loops), Avgust 2016.
- [9] Wikipedia: Rule 110. [https://en.wikipedia.org/wiki/Rule\\_110](https://en.wikipedia.org/wiki/Rule_110), Avgust 2016.
- [10] Wikipedia: Von Neumann universal constructor. [https://en.wikipedia.org/wiki/Von\\_Neumann\\_universal\\_constructor](https://en.wikipedia.org/wiki/Von_Neumann_universal_constructor), Avgust 2016.
- [11] Neil Bickford. Reversing the Game of Life for Fun and Profit. <https://nbickford.wordpress.com/2012/04/15/reversing-the-game-of-life-for-fun-and-profit/>, April 2012.
- [12] Peter Borah. Atabot. <https://github.com/PeterBorah/atabot>, 2013.
- [13] Bryan Duxbury. Cellular Chronometer. [https://github.com/bryanduxbury/cellular\\_chronometer/tree/master/rb](https://github.com/bryanduxbury/cellular_chronometer/tree/master/rb), September 2013. Članek: <https://bryanduxbury.com/2013/09/02/cellular-chronometer-part-1-reversing-the-game-of-life/>.
- [14] Yossi Elran. Retrolife and The Pawns Neighbors. *The College Mathematics Journal*, 43(2):147–151, 2012. Dostopno na <http://www.jstor.org/stable/10.4169/college.math.j.43.2.147>.

- [15] Dave Greene. New Technology from the Replicator Project. <http://b3s23life.blogspot.si/2013/11/new-technology-from-replicator-project.html>, November 2013. Dostop: 2-avg-2016.
- [16] Jean Hardouin-Duparc. À la recherche du paradis perdu. *Publ. Math. Univ. Bordeaux Année*, 4:51–89, 1972-1973.
- [17] Jean Hardouin-Duparc. Paradis terrestre dans l'automate cellulaire de Conway. *Rev. Française Automat. Informat. Recherche Operationnelle Ser. Rouge*, 8:64–71, 1974. Dostopno na [http://archive.numdam.org/ARCHIVE/ITA/ITA\\_1974\\_\\_8\\_3/ITA\\_1974\\_\\_8\\_3\\_63\\_0/ITA\\_1974\\_\\_8\\_3\\_63\\_0.pdf](http://archive.numdam.org/ARCHIVE/ITA/ITA_1974__8_3/ITA_1974__8_3_63_0/ITA_1974__8_3_63_0.pdf).
- [18] Christiaan Hartman, Marijn J. H. Heule, Kees Kwekkeboom, in Alain Noels. Symmetry in Gardens of Eden. *Electronic Journal of Combinatorics*, 20, 2013.
- [19] ionreq. calculate previous generations for Conway's Game of Life. <https://github.com/ionreq/GoLreverse>, 2013. Video prispevek: <https://www.youtube.com/watch?v=Z0uMnaarI5k>.
- [20] Iztok Jeras. Solving cellular automata problems with SAGE/Python. Objavljeno v Andrew Adamatzky, Ramón Alonso-Sanz, Anna T. Lawniczak, Genaro Juárez Martínez, Kenichi Morita, in Thomas Worsch, editors, *Automata 2008: Theory and Applications of Cellular Automata, Bristol, UK, June 12-14, 2008*, strani 417–424. Luniver Press, Frome, UK, 2008. Dostopno na <http://uncomp.uwe.ac.uk/free-books/automata2008reducedsize.pdf>.
- [21] Iztok Jeras. 2D cellular automata preimages count&list algorithm. <https://github.com/jeras/preimages-2D>, 2016.
- [22] Iztok Jeras. Cellular Automata SAGE Toolkit. <https://github.com/jeras/cellular-automata-sage-toolkit>, 2016.
- [23] Iztok Jeras. WebGL QUAD CA simulator. <https://github.com/jeras/webgl-quad-ca>, 2016.
- [24] Iztok Jeras in Andrej Dobnikar. Cellular Automata Preimages: Count and List Algorithm. Objavljeno v Vassil N. Alexandrov, G. Dick van Albada, Peter M. A. Sloot, in Jack Dongarra, editors, *Computational Science - ICCS 2006, 6th International Conference, Reading, UK, May 28-31, 2006, Proceedings, Part III*, del 3993 of *Lecture Notes in Computer Science*, strani 345–352. Springer, 2006. Dostopno na [http://dx.doi.org/10.1007/11758532\\_47](http://dx.doi.org/10.1007/11758532_47).
- [25] Iztok Jeras in Andrej Dobnikar. Algorithms for computing preimages of cellular automata configurations. *Physica D Nonlinear Phenomena*, 233:95–111, September 2007.
- [26] Jarkko Kari. Reversibility of 2D cellular automata is undecidable. *Physica D: Nonlinear Phenomena*, 45:379–385, 1990.
- [27] Paulina A. Léon in Genaro J. Martínez. Describing Complex Dynamics in Life-Like Rules with de Bruijn Diagrams on Complex and Chaotic Cellular Automata. *Journal of Cellular Automata*, 11(1):91–112, 2016.



- [28] Harold V. McIntosh. Linear Cellular Automata via de Bruijn Diagrams. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.41.8763&rep=rep1&type=pdf>, August 1991.
- [29] E. F. Moore. Machine models of self-reproduction. Objavljeno v *Mathematical Problems in Biological Sciences (Proceedings of Symposia in Applied Mathematics)*, del 14, strani 17–33. American Mathematical Society, 1962.
- [30] John Myhill. The converse of Moore's Garden-of-Eden theorem. Objavljeno v *Proceedings of the American Mathematical Society*, del 14, strani 685–686. American Mathematical Society, 1963.
- [31] Florian Pigorsch. A SAT-based forward/backwards solver for Conway's "Game of Life". <https://github.com/flopp/gol-sat>, Oktober 2015.
- [32] Edward Powley. QUAD Prize Submission: Simulating Elementary CAs with Trid CAs. *Journal of Cellular Automata*, 5:415–417, 2010. Dostopno na [http://uncomp.uwe.ac.uk/automata2008/files/quadprize\\_powley.pdf](http://uncomp.uwe.ac.uk/automata2008/files/quadprize_powley.pdf).
- [33] Paul Rendell. A Turing Machine In Conway's Game Life. <https://www.ics.uci.edu/~welling/teaching/271fall09/Turing-Machine-Life.pdf>, Avgust 2001. Dostop: 2-avg-2016.
- [34] Chris Salzberg in Hiroki Sayama. Complex genetic evolution of artificial self-replicators in cellular automata. *Complexity*, 10:33–39, 2004. Dostopno na <http://www3.interscience.wiley.com/journal/109860047/abstract>.
- [35] José Manuel Gómez Soto. Computation of Explicit Preimages in One-Dimensional Cellular Automata Applying the De Bruijn Diagram. *Journal of Cellular Automata*, 3:219–230, 2008.
- [36] Tommaso Toffoli. Background for the Quad Prize. <http://uncomp.uwe.ac.uk/automata2008/files/quad.pdf>, Februar 2008.
- [37] Erik P. Verlinde. On the origin of gravity and the laws of Newton. 2010.
- [38] Robert Wainwright. Lifeline newsletter - volume 3. [http://www.conwaylife.com/w/index.php?title=Lifeline\\_Volume\\_3](http://www.conwaylife.com/w/index.php?title=Lifeline_Volume_3), September 1971.
- [39] Robert Wainwright. Lifeline newsletter - volume 4. [http://www.conwaylife.com/w/index.php?title=Lifeline\\_Volume\\_4](http://www.conwaylife.com/w/index.php?title=Lifeline_Volume_4), December 1971.
- [40] Christopher Wellons. WebGL Game of Life. <https://github.com/jeras/webgl-quad-ca>, 2014.
- [41] Andrew Wuensche. Discrete Dynamics Lab. <https://www.ddlab.com>, 2016.
- [42] Andrew Wuensche in Mike Lesser. *The Global Dynamics of Cellular Automata*. Santa Fe Institute, 1992. Dostopno na <http://uncomp.uwe.ac.uk/wuensche/gdca.html>.