

UNIVERZA V LJUBLJANI  
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Iztok Jeras

**Predslike 2D celičnih avtomatov**

MAGISTERSKO DELO  
NA UNIVERZITETNEM ŠTUDIJU

Ljubljana, 2016



UNIVERZA V LJUBLJANI  
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Iztok Jeras

**Pred slike 2D celičnih avtomatov**

MAGISTERSKO DELO  
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Branko Šter

Ljubljana, 2016



# Zahvala

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



# Kazalo

<b>Povzetek</b>	<b>1</b>
<b>Abstract</b>	<b>2</b>
<b>1 Uvod</b>	<b>3</b>
1.1 CA kakor model vesolja . . . . .	3
1.2 Informacijska dinamika . . . . .	3
1.3 Problem predslik 2D CA . . . . .	3
1.4 Algoritmi za iskanje predslik . . . . .	4
<b>2 Definicija 2D CA</b>	<b>6</b>
<b>3 Konstrukcija mreže predslik</b>	<b>8</b>
3.1 De Bruijnov diagram . . . . .	8
3.2 Mreža . . . . .	12
3.3 Robni pogoji . . . . .	13
<b>4 Algoritem za štetje in izpis predslik</b>	<b>15</b>
4.1 Procesiranje v eni dimenziji . . . . .	15
4.2 Procesiranje v drugi dimenziji . . . . .	17
4.3 Izpis predslik . . . . .	17
4.4 Nezmožnost procesiranja z linearno zahtevnostjo . . . . .	18

<b>5</b>	<b>Primerjava z znanimi algoritmi</b>	<b>20</b>
5.1	Iskanje GoE za GoL . . . . .	20
5.2	Iskanje predslik GoL na splošno . . . . .	21
5.3	Polni algoritmi za štetje in izpis predslik . . . . .	22
<b>6</b>	<b>Sklepne ugotovitve</b>	<b>23</b>
6.1	Analiza 2D CA s pomočjo končnih avtomatov . . . . .	23
6.2	Izboljšave podanega algoritma . . . . .	24
<b>A</b>	<b>De Bruijinov diagram za GoL</b>	<b>25</b>
<b>B</b>	<b>Primer delovanja algoritma za CA z okolico quad</b>	<b>26</b>
<b>C</b>	<b>Don Woods opiše svoj algoritem</b>	<b>30</b>
	<b>Seznam slik</b>	<b>32</b>
	<b>Literatura</b>	<b>33</b>



# Seznam uporabljenih kratic in simbolov

<b>1D</b> .....	eno dimenzionalen
<b>2D</b> .....	dvo dimenzionalen
<b>3D</b> .....	tro dimenzionalen
<b>CA</b> .....	celični avtomat
<b>GoL</b> .....	Conway's Game of Life (Conwayeva igra življenja)
<b>GoE</b> .....	Garden of Eden (stanje brez predslik)
<b>trid</b> .....	okolica CA sestavljena iz treh celic
<b>quad</b> .....	okolica CA sestavljena iz štirih celic $M_x = M_y = 2$
$C$ .....	poljubna konstanta
$S$ .....	množica nabora stanj celice
$ S $ .....	število možnih stanj celice
$c$ .....	stanje posamezne celice (celoštevilska vrednost)
$c_{x,y}$ .....	vrednost celice na položaju $[x, y]$ znotraj 2D polja celic
$c^t$ .....	vrednost celice v sedanjosti
$c^{t+1}$ .....	vrednost celice v prihodnosti (eden korak)
$N_x$ .....	velikost 2D polja celic v dimenziji X
$N_y$ .....	velikost 2D polja celic v dimenziji Y
$N$ .....	število celic v 2D polju
$M_x$ .....	velikost 2D okolice celice v dimenziji X
$M_y$ .....	velikost 2D okolice celice v dimenziji Y
$m$ .....	število celic v 2D okolici
$n$ .....	stanje okolice posamezne celice (celoštevilska vrednost)
$n_{x,y}$ .....	vrednost okolice celice na položaju $[x, y]$ znotraj 2D polja celic

$n^{t-1}$	.....	vrednost okolice celice v preteklosti (eden korak)
$n^t$	.....	vrednost okolice celice v sedanjosti
$f$	.....	tranzicijska funkcija, ki definira časovno evolucijo avtomata
$f^{-1}$	.....	obratna tranzicijska funkcija
$o_x$	.....	prekrivanje okolice v dimenziji X
$o_y$	.....	prekrivanje okolice v dimenziji Y
$o_{xy}$	.....	prekrivanje okolice 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. To magistrsko delo poizkuša aplicirati metode razvite za 1D avtomate na 2D problem. Pri-  
kazan je algoritem, ki omogoča štetje in izpis predslik. Razvit je bil s pomočjo grafičnega modela, mreže predslik, ki omogoča uporabo teorije grafov pri izračunih in dokazih.

## Ključne besede:

celični avtomati, predslike, procesna 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. This masters thesis attempts to apply methods developed for 1D automata to the 2D problem. An algorithm is shown, which can count and list preimages. It was developed with the help of a graphical representation, the preimage network, which enables using graph theory for computation and proofs.

## Key words:

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

# Poglavje 1

## Uvod

### 1.1 CA kakor model vesolja

Ker lahko vsak univerzalen sistem modelira vsak drug univerzalen sistem, lahko predpostavimo, da lahko z univerzalnimi 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 [27]), 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 [25]. Oba sta pomembna informacijska pojava v našem vesolju.

### 1.2 Informacijska dinamika

Informacijsko dinamiko CA se najpogosteje opisuje samo kakor reverzibilno ali ireverzibilno, obstaja tudi nekaj člankov, ki opazujejo entropijo sistema. Pogosto je tudi opazovanje dinamike delcev pri GoL in elementarnem pravilu 110. Ne obstaja pa še splošna teorija dinamike informacij v CA. V svojem članku [17] in prispevkih na konferencah [11, 13, 16], sem grafično upodobil predslike trenutnega stanja za 1D problem. Iz upodobitve je videti, da se ponekod izgubi več informacije kakor drugod, kar kaže na možnost izpeljave kvalitativne in kvantitativne teorije dinamike informacij. Na žalost se ta možnost še ni udejanila. Podobno je možno grafično upodobiti predslike 2D CA, ter iz grafov sklepati o izgubi informacij v 2D CA.

### 1.3 Problem predslik 2D CA

Najbolje teoretično raziskan 2D CA je GoL (Game of Life ali slovensko igra življenja). Ogromno truda je bilo vloženega v raziskovanje delcev in njihove dinamike. S pomočjo osnovnih gradnikov, je mogoče skonstruirati kompleksnejše sisteme, med katerimi so najzanimivejši turingov stroj [24] in univerzalni konstruktor [6].

Delci so dejansko atraktorji v razvoju CA na končni periodični mreži (thorus). Pri 1D CA se algoritmi za iskanje predslik uporabljajo za določitev atraktorjevega korita [30]. V vsakem CA, ki ni lokalno injektiven se pojavljajo stanja brez predslik imenovana GoE (Garden of Eden ali rajski vrt) [20,21]. Pomensko so GoE nasprotje delcev, saj se nahajajo kar najdlje od atraktorja na robu korita. GoE stanja tudi privlačijo raziskovalce, čeprav v nekoliko manjši meri kakor delci.

Največ raziskav s področja predslik GoL je bilo opravljenih ravno s ciljem iskanja GoE stanj. S stališča algoritma za štetje predslik je GoE stanje tako, ki nima nobene predslike. Algoritem za štetje predslik je možno pretvoriti v manj zahteven algoritem za preverjanje ali je stanje GoE, tako da se operacije nad celimi števili pretvori v logične operacije nad Boolovimi stanji.

## 1.4 Algoritmi za iskanje predslik

Raziskave algoritmov sem se lotil s predpostavko, da je možno opraviti štetje z zahtevnostjo, ki je linearno odvisna od velikosti problema  $O(N)$  ( $N$  je število opazovanih celic). Čeprav je to možno pri 1D problemu, se izkaže, 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 CA polja  $O(C^{N_x})$ , je pa možno, da obstaja algoritem z nižjo kompleksnostjo.

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 Bruijn-ovi diagrami. Paulina Léon in Genaro Martínez [19] poizkušata aplicirati De Bruijn-ove diagrame na 2D CA, doslej je bilo to orodje uporabljeno le na 1D problemih. Točneje, opazujeta dva CA: 'Game of Life' in 'Diffusion rule', s poudarkom na opazovanju stabilnih delcev. Sam uporabljam De Bruijn-ovi diagrame nekoliko drugače, in jih tudi drugače grafično upodabljam. Posamezen De Bruijn-ov diagram postane mreža predslik ene celice. Mreže posameznih celic se nato povezujejo, tako da na koncu opisujejo celotno polje celic.

Doslej sem že razvil napredne algoritme za izračun predslik 1D CA [17]. Skozi zgodovino so taki algoritmi napredovali, tako da je padala njihova procesna zahtevnost in opisna/implementacijska zahtevnost.

1. 'brute force' algoritmi  $O(C^N)$
2. improvizirani algoritmi
3. zasnove matematičnega modela
4. optimalni algoritmi  $O(N)$

Iskanje predslik 2D CA je trenutno nekje med improvizacijo in matematičnim modelom.

Opisan algoritem na koncu primerjam z ostalimi doslej znanimi algoritmi. Čeprav s stališča procesne zahtevnosti ne prinaša zelenega napredka, pa to, da temelji na pregledni

grafični upodobitvi, daje upanje, da bojo razne optimizacije razvidne bodočim raziskovalcem.

Celični avtomat GoL je definiran z Moorovo okolico velikosti  $3 \times 3$ . Tako velika okolica ima z 9 celicami  $2^9 = 512$  možnih stanj, zaradi česar so ilustracije mreže predslik velike in nepregledne. Splošno je znana manjša von Neumannova okolica velikosti  $3 \times 3$  v obliki križa iz 5 celic. Z  $2^5 = 32$  stanji bi bile ilustracije bolj pregledne, ampak obstajajo tudi okolice z manj celicami. Toffoli [26] je leta 2008 spodbudil raziskovalce k iskanju univerzalnosti za CA z okolicama trid (tri celice na heksagonalnem polju) in quad (štiri celice na kvadratnem polju). Powley [23] hitro dokaže obstoj univerzalnega avtomata z okolico trid, tako da ustvari avtomat, ki lahko s pomočjo določenega začetnega stanja simulira poljubni 1D elementarni celični avtomat, na primer pravilo 110. Sam sem obe okolici poznal že prej. Njuna najboljša lastnost je majhen nabor stanj okolice,  $2^4 = 16$  za quad in  $2^3 = 8$  za trid. Za primere je bil uporabljen quad, saj so diagrami na kvadratni mreži bolj enostavni in pregledni, kakor diagrami na heksagonalni mreži.

Algoritem je implementiran kot računalniški program v jeziku C [12]. 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 [15] in orodje za prikaz poljubne mreže predslik [14]. Obe orodji se lahko zažene kar v internetnem brskalniku.

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 2D CA

Predstavljena definicija 2D CA je enostavna in manj formalna v primerjavi z definicijo 1D CA v podobnih prispevkih. Bolj formalna definicija ni potrebna, saj se opisani problemi pri 2D CA, še ne povezujejo z drugimi vejami matematike toliko kakor pri 1D CA.

Osnovni element 2D CA je celica, kakor del polja celic. Vsaka celica ima diskretno vrednost  $c$  iz nabora stanj celice  $S$ . Stanja so običajno kar oštevilčena.

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

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 definira kakor pravokotnik velikosti  $N_x \times N_y$ . Skupno število celic v končnem polju je  $N = N_x \cdot N_y$ .

Prihodnje stanje neke celice  $c_{x,y}$  na položaju  $(x, y)$  je odvisno od trenutnega stanja pripadajoče okolice  $n_{x,y}$  (slika 2.1). 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 okolice.

$$n \in S^m \quad \text{in} \quad S^m = \{0, 1, \dots, |S|^m - 1\} \quad (2.2)$$

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.

Preslikava sedanje okolice  $n_{x,y}^t$  v prihodnjo istoležno celico  $c_{x,y}^{t+1}$  je definiran s tranzicijsko funkcijo  $f$ , ki vsaki vrednosti okolice pripiše vrednost celice.

$$c_{x,y}^{t+1} = f(n_{x,y}^t) \quad (2.3)$$





Slika 2.1: Celica  $c_{x,y}$  in pripadajoča okolica  $n_{x,y}$  z dimenzijama  $M_x = M_y = 3$ .

Za potrebe iskanja predslik je zanimiva obratna funkcija  $f^{-1}$ , ki ob podanem stanju trenutne celice  $c_{x,y}^t$  vrne množico okolice, ki se preslikajo v to vrednost.

$$f^{-1}(c^t) = \{n^{t-1} \in S^m \mid f(n^{t-1}) = c^t\} \quad (2.4)$$

Dodaten pogoj za predslike polja celic je, da se morajo okolice sosednjih celic ujemati povsod, kjer se prekrivajo.

Tranzicijsko funkcijo je možno definirati s pravilom. Pravilo  $r$  je celo število v  $S$ -iškem številskem sestavu, kjer so cifre zaporedje vrednosti celic za vsako od  $|S|^m$  možnih okolice. Vseh pravil je na voljo  $|S|^{|S|^m}$ .

$$r = \sum_{n=0}^{n=|S|^m-1} |S|^n \cdot f(n) \quad (2.5)$$

$$r \in \{0, 1, \dots, |S|^{|S|^m} - 1\} \quad (2.6)$$

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 neodvisen od stanja celice. Ta posplošitev je uporabljena za konstrukcijo umetnih mrež predslik, ki poudarjajo konkretne probleme povezane s kompleksnostjo algoritma.

Za dano polje celic velikosti  $N_x \times N_y$  in z odprtimi robovi, je možno izračunati prihodnje stanje polja velikosti  $(N_x - (M_x - 1)) \times (N_y - (M_y - 1))$ . V primeru, če so robovi polja ciklično zaprti (polje v obliki thorusa), je pa polje prihodnjega stanja enako veliko kakor polje sedanjega. Podobno velja za računanje predslik, za sedanje polje velikosti  $N_x \times N_y$  je za odprte robove velikost polja predslik  $(N_x + (M_x - 1)) \times (N_y + (M_y - 1))$ . Za ciklične robove pa sta velikosti enaki.

## Poglavje 3

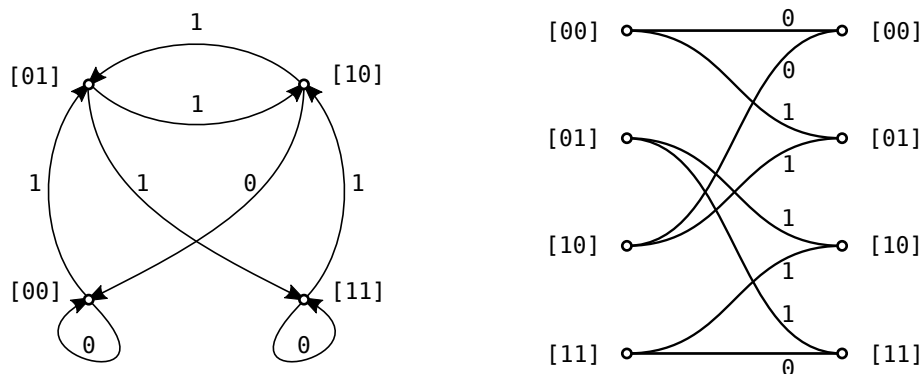
# Konstrukcija mreže predslik

Mreža predslik je grafični konstrukt, ki omogoča upodobitev posameznih pojmov iz definicije CA, kakor ločene grafične elemente. Odnosi med temi elementi definirajo pravila na katerih se gradijo algoritmi za iskanje predslik.

### 3.1 De Bruijinov diagram

Osnovni element grafične upodobitve je De Bruijinov diagram. V osnovi ta obravnava ciklične premike končnih zaporedij simbolov, ter njihovo prekrivanje. Vozlišča v diagramu so vsa možna končna zaporedja, povezave med njimi pa definirajo, kako se ta zaporedja prekrivajo med seboj.

Pri 1D CA se problem neposredno preslika na De Bruijinov graf. McIntosh in njegova skupina uporabljajo za analizo te De Bruijinove grafe neposredno. Sam sem pa razvil modificiran graf, kjer so vozlišča podvojena, in gredo poti vedno od originala proti dvojniku (slika 3.1). To omogoča veriženje grafov, in razširitev osnovnega De Bruijinovega grafa. Medtem ko osnovni De Bruijinov graf opisuje okolico ene celice, verižen graf opisuje verigo celic.



Slika 3.1: De Bruijinov graf za elementarni 1D CA, pravilo 110.

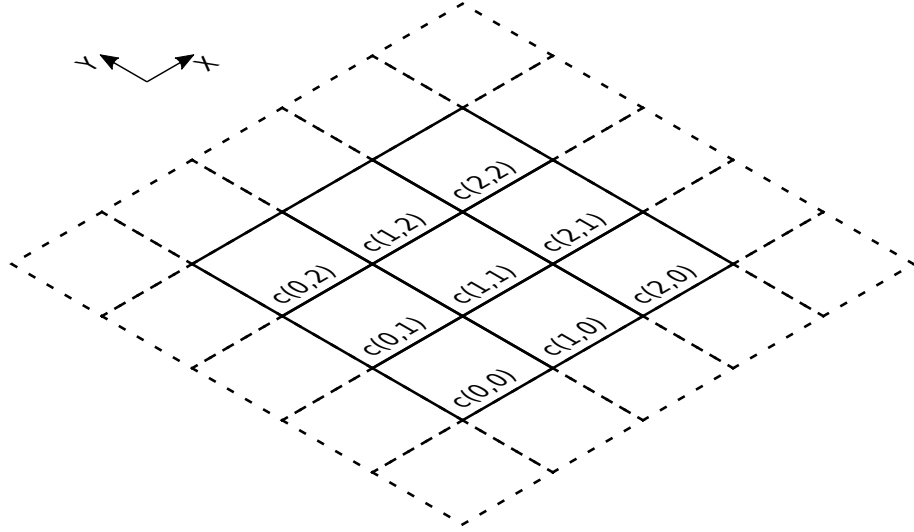
Za potrebe opisa 2D celičnih avtomatov, je bila elementom De Bruijinovega grafa dodana nova dimenzija. Povezave med vozliči se spremenijo v ploskve, in vozlišča se spremenijo v robove ploskev. Elementi celičnega avtomata, ki se preslikajo v graf so:

- nabor vseh možnih **okolic** celice postane nabor vseh **ploskev** (slika 3.9)
- nabor **prekrivanj okolic v smeri 2D dimenzij** (slika 3.5) postane nabor **povezav**
- nabor **prekrivanj okolic v diagonalni smeri** (slika 3.7) postane nabor **vozlišč**

Elemente celičnega avtomata, kakor okolice in prekrivanja okolic je potrebno indeksirati, tako da se lahko vsakemu elementu pripiše unikatna števena vrednost. Vsaki okolici je pripisana zaporedna vrednost, ki je konstruirana kakor  $m$  mestno število v  $S$ -iškem številskem sistemu (za podane primere dvojiški). Cifre si sledijo od spodaj levo do zgoraj desno znotraj okolice (sliki 3.2 in 3.3).

$$n = \sum_{\substack{x=0 \\ y=0}}^{x=M_x-1 \\ y=M_y-1} |S|^{yM_x+x} \cdot c_{x,y} \quad (3.1)$$

$$n \in \{0, 1, \dots, |S|^{M_x M_y} - 1\} \quad (3.2)$$



Slika 3.2: Indeksiranje okolice celice z dimenzijama  $M_x = M_y = 3$ .

Celice se v smeri dimenzije X prekrivajo za ploskev velikosti  $(M_x - 1) \times M_y$  (sliki 3.4 in 3.5). To ob indeksiranju da nabor:

$$o_{x-} = \sum_{\substack{x=0 \\ y=0}}^{x=M_x-2 \\ y=M_y-1} |S|^{y(M_x-1)+x} \cdot c_{x,y} \quad (3.3)$$

$$o_{x+} = \sum_{\substack{x=1 \\ y=0}}^{x=M_x-1 \\ y=M_y-1} |S|^{y(M_x-1)+x} \cdot c_{x,y} \quad (3.4)$$



Slika 3.3: Indeksiranje okolice celice z dimenzijama  $M_x = M_y = 2$ .

$$o_x \in \{0, 1, \dots, |S|^{(M_x-1)M_y} - 1\} \quad (3.5)$$

Celice se v smeri dimenzije Y prekrivajo za ploskev velikosti  $M_x \times (M_y - 1)$  (sliki 3.4 in 3.5). To ob indeksiranju da nabor:

$$o_{y^-} = \sum_{\substack{x=0 \\ y=0}}^{x=M_x-1 \\ y=M_y-2} |S|^{yM_x+x} \cdot c_{x,y} \quad (3.6)$$

$$o_{y^+} = \sum_{\substack{x=0 \\ y=1}}^{x=M_x-1 \\ y=M_y-1} |S|^{yM_x+x} \cdot c_{x,y} \quad (3.7)$$

$$o_y \in \{0, 1, \dots, |S|^{M_x(M_y-1)} - 1\} \quad (3.8)$$



Slika 3.4: 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.

Celice se v diagonalni smeri prekrivajo za ploskev velikosti  $(M_x - 1) \times (M_y - 1)$  (sliki 3.6 in 3.7). To ob indeksiranju da nabor:

$$o_{x-y^-} = \sum_{\substack{x=0 \\ y=0}}^{x=M_x-2 \\ y=M_y-2} |S|^{y(M_x-1)+x} \cdot c_{x,y} \quad (3.9)$$



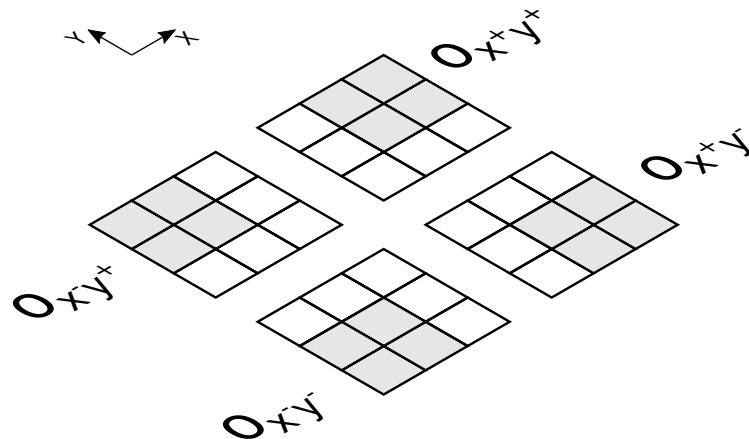
Slika 3.5: Prekrivanje okolic sosednjih celic v smeri dimenzij X in Y, za velikost okolice  $M_x = M_y = 2$ . Okolice se prekrivajo v 2 celicah od 4.

$$o_{x^-y^+} = \sum_{\substack{x=0 \\ y=1}}^{x=M_x-2 \\ y=M_y-1} |S|^{y(M_x-1)+x} \cdot c_{x,y} \quad (3.10)$$

$$o_{x^+y^-} = \sum_{\substack{x=1 \\ y=0}}^{x=M_x-1 \\ y=M_y-2} |S|^{y(M_x-1)+x} \cdot c_{x,y} \quad (3.11)$$

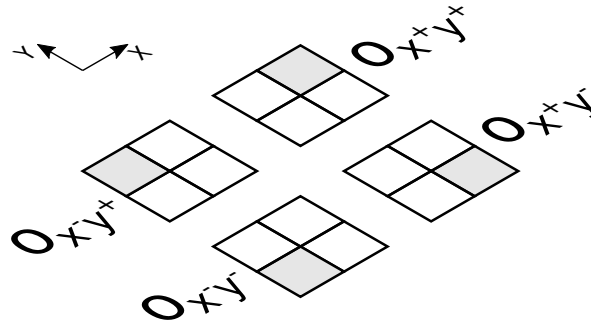
$$o_{x^+y^+} = \sum_{\substack{x=1 \\ y=1}}^{x=M_x-1 \\ y=M_y-1} |S|^{y(M_x-1)+x} \cdot c_{x,y} \quad (3.12)$$

$$o_{xy} \in \{0, 1, \dots, |S|^{(M_x-1)(M_y-1)} - 1\} \quad (3.13)$$

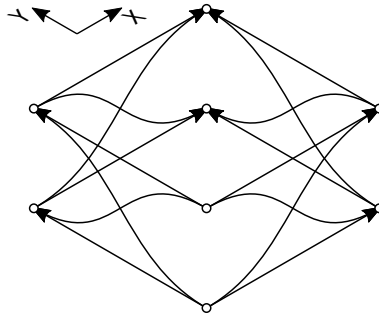


Slika 3.6: Prekrivanje okolic sosednjih celic v diagonalni smeri, za velikost okolice  $M_x = M_y = 3$ . Okolice se prekrivajo v 4 celicah od 9.

Nastali graf (slika 3.8) 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 3.7: Prekrivanje okolic sosednjih celic v diagonalni smeri, za velikost okolice  $M_x = M_y = 2$ . Okolice se prekrivajo v eni celici od 4.



Slika 3.8: Mreža ene celice za binarni CA z okolico quad  $M_x = M_y = 2$ .

## 3.2 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 in, ki jih je možno sestaviti iz naborov ploskev diagramov posameznih celic.

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 se vzame centralno celico, na koncu pa se doda še za eno celico širok rob okoli celotne ploskve.

Podani primeri uporabljajo binarni CA s quad okolico. Za ta CA je velikost diagonalnega prekrivanja okolic ena sama celica (slika 3.7), posledično ima nabor vozlišč le dve vrednosti, ki neposredno predstavljajo vrednosti celic v predsliki (slika 3.10). Nabor okolic/ploskev pa obsega 16 kombinacij (slika 3.9).

Razširitev diagrama ene celice v mrežo lahko dokažemo z indukcijo. Dokazati želimo, da je neka konfiguracija celic predslika dane sedanjosti *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 natanko v primeru, ko se z njo ujema v robu (vozliščema in povezavi med njima). Temu je tako, ker so indeksi vozlišč in povezav ekvivalentni vrednosti prekrivanj okolic.



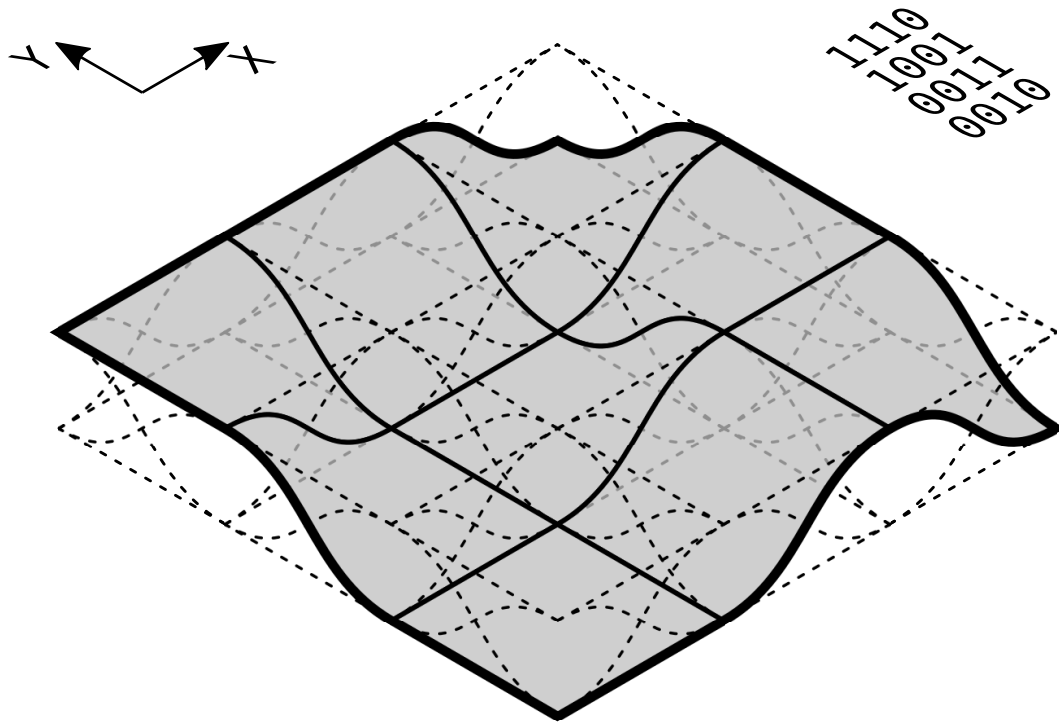
Slika 3.9: Nabor ploskev za vseh 16 možnih vrednosti okolic za binarni CA z okolico quad  $M_x = M_y = 2$ .

### 3.3 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 3.10). Na splošno ima vsaka zvezna ploskev v mreži svoj rob, robni pogoj določa, kako je ta ploskev 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. Temu bomo rekli odprti rob, ker ta ne definira nobenih omejitev, katere zvezne ploskve v mreži predslik so dovoljene in katere ne.

Obstaja še eden enostaven robni pogoj, ki je definiran za ciklično sklenjena končna CA polja. 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



Slika 3.10: 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.

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 števecv 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 polinomsko kompleksnostjo 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 je najprej poiskan nabor predslik za prvo vrstico. Nabor predslik je izračunan kakor uteži za robove na nasprotni strani od začetne. seznam robov in njihove uteži so uporabljeni kakor vhodni robni pogoji za naslednjo vrstico. Robne uteži izračunane za zadnjo vrstico predstavljajo število vseh predslik.

Algoritem za izpis predslik je nadaljevaje algoritma za štetje. Starta 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

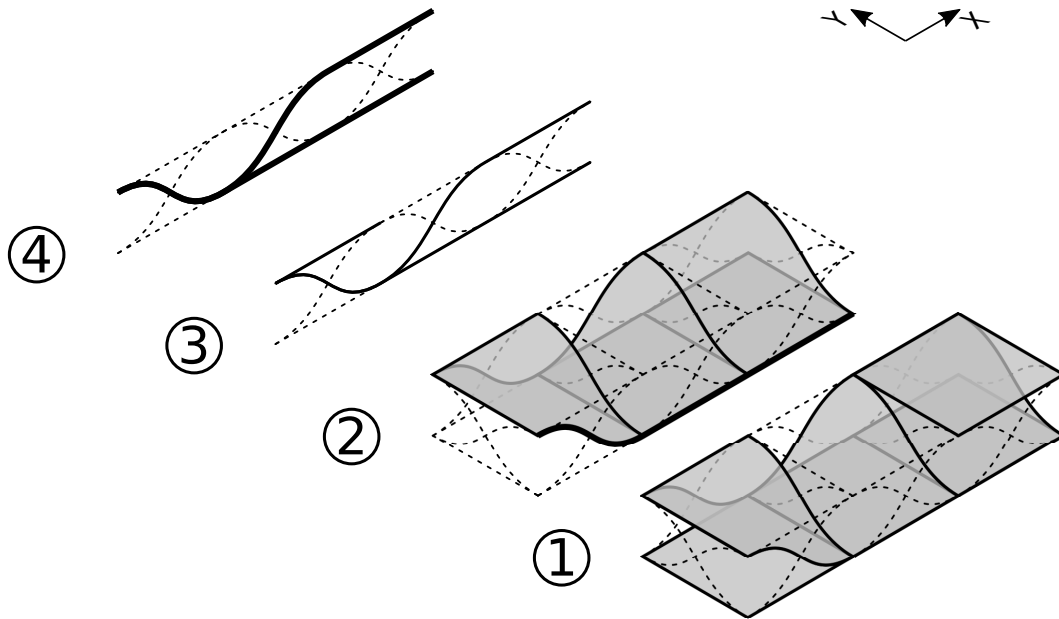
Procesiranje se začne z eno dimenzionalnim nizom celic (vrstico). Vsaki celici v nizu pripada lastna mreža predslik, eno dimenzionalnemu nizu celic posledično pripada povezan niz mrež predslik (slika 4.1 mreža 1). Vsak segment mreže v nizu ima svoj nabor veljavnih

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. Vsaka ploskev je obravnavana posebej in prerez je izražen kakor robni pogoj. Začetni robni pogoj za vsako vrstico je nabor poti med vozlišči na začetku vrstice. Vsaka pot je obravnavana 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**). Za tem korakom, so vse ploskve definirane v dveh od štirih vozlišč, do nezveznosti lahko prihaja le še na robu nasprotnem začetnemu robu.

Problem se reducira iz tri dimenzionalne mreže in procesiranja ploskev v dvo dimenzionalno 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 [17]. 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 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  $|S|^{(M_y-1)(M_x-1+N_x)}$ . 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 procesna 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  $w \in 0, 1$  namesto operacij množenja in seštevanja, je možno poenostaviti iskanje predslik samo na njihov obstoj za opazovani robni pogoj.

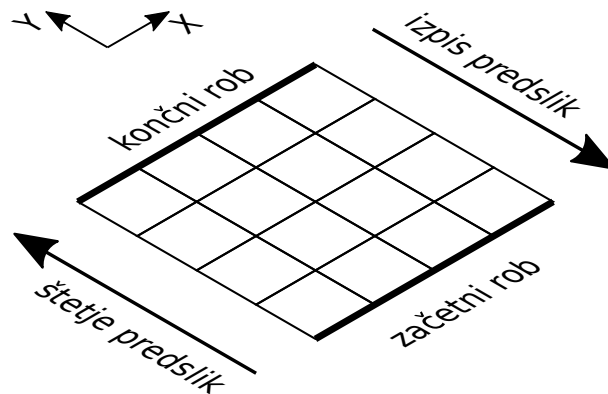
## 4.3 Izpis predslik

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 okolic  $o_{y+}$  in končni rob je zaporedje prekrivanja okolic  $o_{y-}$ .

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 poti (določeno zaporedje prekrivajočih okolic) da vrednost celic v predsliki, uteži izračunane v smeri štetja pa dajo število predslik, ki jih je potrebno popisati z dano vrednostjo celic.

Z vsakim korakom v obratni smeri se popiše nova vrstica za vsako od prešteti predslik. Ko pride algoritem, spet do začetne vrstice, so vse predslike popisane. Hkrati so iz



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

mreže predslik izločene vse slepe poti, ostanejo le še ploskve, ki tvorijo zvezne ploskve na celotnem polju celic.

Algoritem tukaj ni podrobno opisan, je pa preprosta razširitev algoritma uporabljenega za 1D CA opisanega v [17]. Za podrobnosti je na voljo izvorna koda [12].

## 4.4 Nezmožnost procesiranja z linearno zahtevnostjo

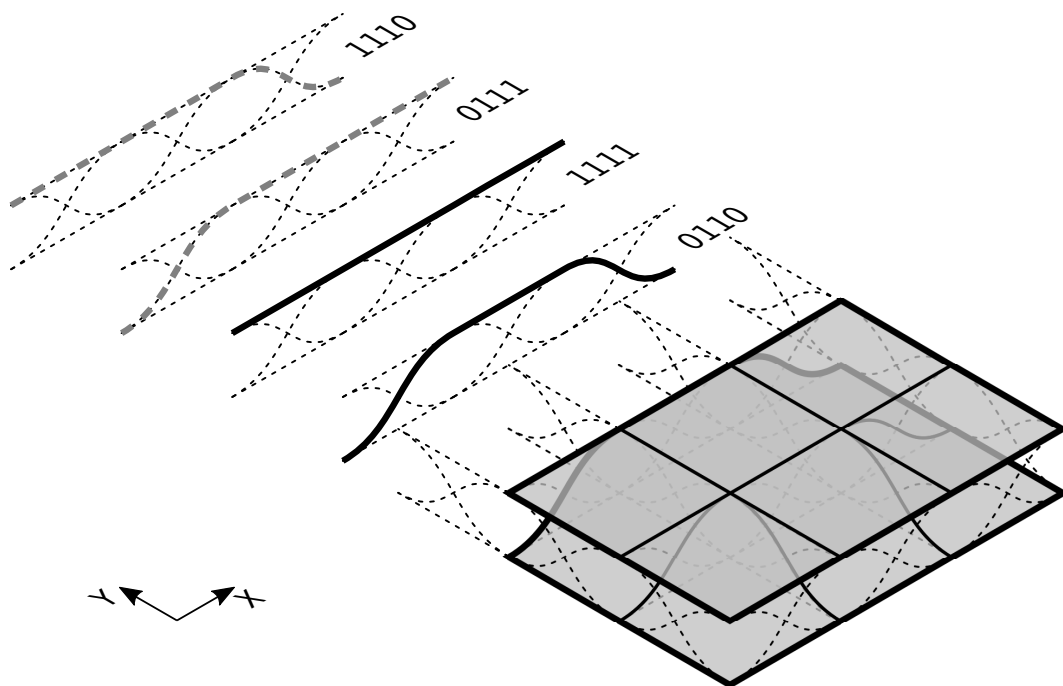
Podan je primer, ki kaže zakaj procesiranje z linearno zahtevnostjo ni mogoče.

Cilj raziskovalnega dela za to raziskovalno nalogo je bil poiskati učinkovit algoritem za iskanje predslik 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 ali na splošno bi bilo število obravnav majhna konstanta (4 krat, če se izvaja procesiranje skozi mrežo predslik v 4 smereh/prehodih). Tak algoritem predpostavlja, da je možno celoten nabor začetnih robnih pogojev upoštevati hkrati v enem samem prehodu. V koraku **2** na pri procesiranju vrstice (sliki 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 te 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 vozliču skupnem obema potema nadaljuje po drugi poti. Iz mreže je razvidno, da te 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 GoE stanj, skratka preverjajo le obstoj predslík 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 predslík.

### 5.1 Iskanje GoE za GoL

Conway je leta 1970 predstavil GoL. Že leta 1971 sta Roger Banks in Steve Ward predstavila prvo GoE stanje velikosti  $9 \times 33$  celic. Odkritje je bilo objavljeno v [28]. Kasneje je Don Woods [28, 29] s pomočjo računalnika dokazal, da je stanje res GoE.

Woodsov algoritem, tako kakor moj, deluje na vrsticah. Starta z eno vogalno celico, katere nabor predslík je kar nabor okolic, ki se preslikajo v stanje celice. V naslednjem koraku se opazuje sosednjo celico v vrstici. Za vse kombinacije okolic dodane celice in obstoječi nabor predslík se preveri, če se ujemajo, kjer se okolice prekrivajo. Če se ujemajo, se doda sestavljeno predslíko v nabor, sicer predslíka izpade iz nabora. Ko pride algoritem do konca vrstice, nadaljuje v naslednji vrstici, le da je sedaj prekrivanje med naborom predslík in okolico dodane celice drugačno (več sosednjih celic namesto ene). Tako algoritem nadaljuje do konca celotne konfiguracije. Če je končni nabor predslík prazen, je opazovana konfiguracija GoE. Nabor predslík 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 GoE konfiguracije malo predslík. Algoritem bi bil za konfiguracije z dosti predslíki veliko bolj pomnilniško in časovno potraten.

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

Leta 1974 je Duparc [7, 8] razvil drugačen algoritem, ter z njim poiskal GoE stanja 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 GoE stanje. 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 GoE stanja, 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 GoE stanja 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 [9]. S sodelavci je zapisal vsa prekrivanja za določeno stanje kakor binarne enačbe. 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 GoE stanj in omogoča pregled večjega števila podobnih stanj. Ob spremembi vrednosti ene same celice omogoča uporabo delnih rezultatov prejšnjega izračuna. SAT metode ne poznam dobro, teko da ne morem primerjati procesne zahtevnosti.

## 5.2 Iskanje predslik GoL na splošno

Erlan [5] 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](https://www.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 optimiziranju, torej fokus ni bil na eksaktni rešitvi.

Pri iskanju implementacij algoritma, sam našel Atabot [3], Celični kronometer [4] in SAT baziran program [22]. 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 GoE stanj, na primer Woods, sicer omogočajo iskanje vseh predslik, ampak to ni njihov glavni namen. Optimizirani le za GoE stanja, kjer je predslik malo tudi za del GoE konfiguracije. 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.

Našel sem le eno aplikacijo, ki je namenjena štetju in izpisu vseh predslik. Bickford [2] 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 [10]. 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 gleda na zahtevnost:

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

V magistrskem delu opišem algoritem, ki rešuje prve tri probleme. Preostala problema sta si sorodna. Problem reverzibilnosti je na splošno dokazano nerešljiv [18]. Problem jezika GoE stanj 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 GoE konfiguracija. Zanke v takem avtomatu pa so sorodne delcem.

Opisan 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 vsem načinom, 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 GoE beseda znotraj regularnega jezika GoE stanj, 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 GoE besede povezana s kompleksnostjo prostorsko-časovnih vzorcev, ki se pojavljajo pri danem pravilu. Podobno so verjetno tudi pri 2D CA GoE konfiguracije 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 GoE stanj, 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 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.

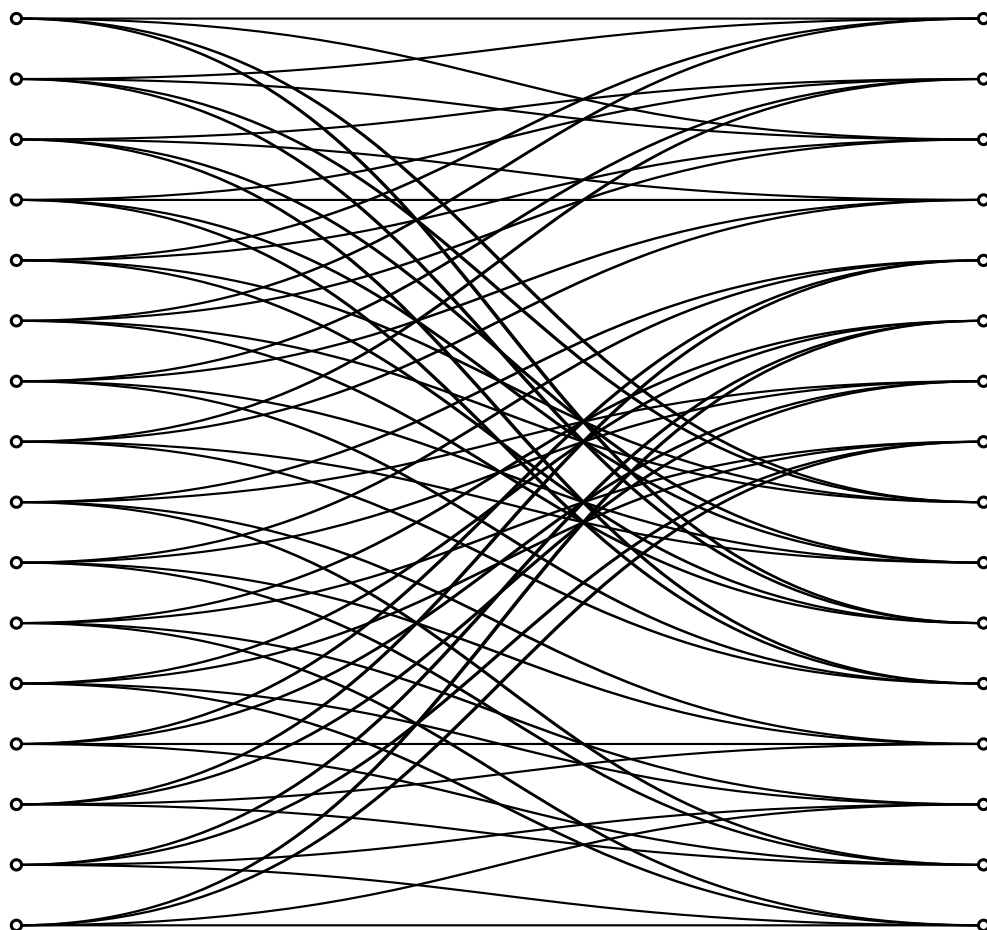
Samo implementacijo algoritma je tudi možno optimizirati. Univerzalnost algoritma, predvsem univerzalnost velikosti okolice prispeva dosti režijskih stroškov. Če bi bil algoritem specializiran samo za quad ali GoL, bi se lahko izognil precej operacijam. Podobno velja za nabor stanj. Za binarne CA je 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.

## Dodatek A

### De Bruijinov diagram za GoL

Slika A.1 prikazuje del De Bruijinovega diagrama za GoL. Graf s 16 vozlišči in 64 povezavami je očitno preveč kompleksen, da bi bil primeren za razlago algoritma. Število vseh možnih okolici (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.

Konfiguracija je majhna velikosti  $2 \times 2$  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 [12] 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

## Don Woods opiše svoj algoritem

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

2.1	Velikost okolice. . . . .	7
3.1	De Bruijinov graf, pravilo 110. . . . .	8
3.2	Indeksiranje okolice $3 \times 3$ . . . . .	9
3.3	Indeksiranje okolice $2 \times 2$ . . . . .	10
3.4	Prekrivaje okolic $3 \times 3$ v smeri dimenzij X in Y. . . . .	10
3.5	Prekrivaje okolic $2 \times 2$ v smeri dimenzij X in Y. . . . .	11
3.6	Prekrivanje okolic $3 \times 3$ - diagonalno. . . . .	11
3.7	Prekrivanje okolic $2 \times 2$ - diagonalno. . . . .	12
3.8	Mreža ene celice. . . . .	12
3.9	Nabor ploskev. . . . .	13
3.10	Mreža polja celic. . . . .	14
4.1	Algoritem procesiranja vrstice. . . . .	16
4.2	Algoritem za izpis predslik. . . . .	18
4.3	Ovira za procesiranje z linearno zahtevnostjo. . . . .	19
A.1	Mreža ene celice za GoL. . . . .	25

# Literatura

- [1] Conway's Reverse Game of Life. <https://www.kaggle.com/c/conway-s-reverse-game-of-life>, 3 2013.
- [2] 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/>, 4 2012.
- [3] Peter Borah. Atabot. <https://github.com/PeterBorah/atabot>, 2013.
- [4] Bryan Duxbury. Cellular Chronometer. [https://github.com/bryanduxbury/cellular\\_chronometer/tree/master/rb](https://github.com/bryanduxbury/cellular_chronometer/tree/master/rb), 9 2013. Članek: <https://bryanduxbury.com/2013/09/02/cellular-chronometer-part-1-reversing-the-game-of-life/>.
- [5] 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>.
- [6] Dave Greene. New Technology from the Replicator Project. <http://b3s23life.blogspot.si/2013/11/new-technology-from-replicator-project.html>, 11 2013. Dostop: 2-avg-2016.
- [7] Jean Hardouin-Duparc. À la recherche du paradis perdu. *Publ. Math. Univ. Bordeaux Année*, 4:51–89, 1972-1973.
- [8] 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).
- [9] Christiaan Hartman, Marijn J. H. Heule, Kees Kwekkeboom, in Alain Noels. Symmetry in Gardens of Eden. *Electronic Journal of Combinatorics*, 20, 2013.
- [10] 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>.
- [11] 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>.
- [12] Iztok Jeras. 2D cellular automata preimages count&list algorithm. <https://github.com/jeras/preimages-2D>, 2016.
  - [13] Iztok Jeras. Cellular Automata SAGE Toolkit. <https://github.com/jeras/cellular-automata-sage-toolkit>, 2016.
  - [14] Iztok Jeras. Displaying 2D CA (quad) preimage network. <https://github.com/jeras/three.js>, 2016.
  - [15] Iztok Jeras. WebGL QUAD CA simulator. <https://github.com/jeras/webgl-quad-ca>, 2016.
  - [16] 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).
  - [17] Iztok Jeras in Andrej Dobnikar. Algorithms for computing preimages of cellular automata configurations. *Physica D Nonlinear Phenomena*, 233:95–111, September 2007.
  - [18] Jarkko Kari. Reversibility of 2D cellular automata is undecidable. *Physica D: Nonlinear Phenomena*, 45:379–385, 1990.
  - [19] 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.
  - [20] 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.
  - [21] 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.
  - [22] Florian Pigorsch. A SAT-based forward/backwards solver for Conway's "Game of Life". <https://github.com/flopp/gol-sat>, 10 2015.
  - [23] 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).
  - [24] Paul Rendell. A Turing Machine In Conway's Game Life. <https://www.ics.uci.edu/~welling/teaching/271fall09/Turing-Machine-Life.pdf>, 8 2001. Dostop: 2-avg-2016.

- [25] 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>.
- [26] Tommaso Toffoli. Background for the Quad Prize. <http://uncomp.uwe.ac.uk/automata2008/files/quad.pdf>, 2 2008.
- [27] Erik P. Verlinde. On the origin of gravity and the laws of Newton. 2010.
- [28] 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), 9 1971.
- [29] 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), 12 1971.
- [30] 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>.