Postavka završnog rada

U okviru ovog završnog rada potrebno je obraditi osnovne koncepte koji se odnose na ćelijske automate. U prvom dijelu rada je potrebno dati matematičke formalne definicije, opisati primjenu i dati pregled stanja u oblasti. U praktičnom dijelu rada je potrebno realizirati simulator za odabranu instancu ćelijskih automata.

Polazna literatura:

1. Wolfram S., 2002, *A New Kind of Science,* Wolfram Media Place
2. Wolfram S., 1984, *Universality and Complexity in cellular Automata*, Physica D: Nonlinear Phenomena10, no. 1–2
3. Schiff J., 2007, *Cellular Automata: A Discrete View of the World*, John Wiley & Sons, Inc., Publication

Sažetak

U radu se obrađuje specijalna klasa konačnih diskretnih modela pod nazivom ‘ćelijski automati’ (eng. cellular automata) koji se baziraju na čisto lokalnoj interakciji ali proizvode uzorke koji se mogu promatrati na globalnijoj skali, te većem nivou apstrakcije. Upravo ova čisto lokalna interakcija omogućava da se pomoću navedenih entiteta omogući modeliranje širokog spektra realnih pojava s obzirom da veliki broj stvari koje se pokušavaju modelirati spada upravo u ovu kategoriju.
Kroz rad se navodi nekoliko klasa, primjera i primjena ćelijskih automata, te se uz pomoć grafičkog simulatora pokušavaju prikazati najbitniji koncepti potrebni za shvatanje načina funkcionisanja te potencijalne primjene ovih modela. Rad služi kao uvod u široku tematiku i primjene ove vrste modela.
Također, ponuđen je i formalni matematički tretman iz raznih aspekta koji uključuju teoriju vjerovatnoće i informacija, te aspekt na koji je stavljen najveći fokus – teorija izračunljivosti s obzirom da se ćelijski automati, tačnije određene instance istih, ekvivalentne Turingovoj mašini što otvara široke primjene ove vrste sistema.

Pregled oznaka i načini enumeracije

Slike su numerisane redom i enumeracije slike nema posebno značenje. Slike u dodacima su numerisane redom sa prefiksom dodatka, na primjer A3 predstavlja treću sliku u dodatku A.

Definicije su numerisane sa x.y.z gdje x.y predstavlja podpoglavlje u kojem se definicija nalazi, dok z predstavlja redni broj definicije u tom podpoglavlju. Numeracija podpoglavlja ide do tri nivoa dubine.

Lista slika sa opisima

Sadržaj

1. Uvod

U uvodnom dijelu pokušat ćemo proći kroz pozadinska razmatranja koja vode do izučavanja klase konačnih diskretnih modela nazvanih ćelijski automati.

Prvo ćemo dati pregled kroz specifičan primjer da se uoče neke od osnovnih karakteristika ovih sistema.

Nakon toga ćemo pregledati historijski koja matematička pitanja su navela da se razmotre neke specijalne klase, te će se nakon toga obratiti pažnja na unificiranje specijalni klasa, te posebno historijski pregled rada Stephena Wolframa za kojeg se može reći da je dao jedan od najvećih doprinosa samom polju.

Bit će dat i pregled potencijalnih primjena te motivacija za izučavanje oblasti ćelijskih automata, te se ovaj dio može smatrati neformalnim uvodom u cjelokupnu tematiku koji može dati osnovnu ideju bilo kome ko želi da se zainteresuje u oblast.

1.1 Motivacija

Nauka stoljećima pokušava da koristeći klasične metode matematike u primijenjenim disciplinama poput fizike objasni i razjasni svijet oko nas, kao i da iz datih početnih uslova nekog sistema da predviđanja za budućnost istog, tj. da predvidi ponašanje bez potrebe da se sam sistem pusti u izvršavanje ili simulaciju. Međutim, i nakon toliko vremena izučavanja, i dalje postoje neka fundamentalna pitanja koja su ostala neodgovorena i čini se kao da ih je nauka zaobilazila te odgovarala samo na pitanja koja su se uklapala u stereotipni način dotadašnjeg razmišljanja da se stvari modeliraju kontinualnim alatima parcijalnih diferencijalnih jednačina. Treba uzeti u razmatranje mogućnost da možda postoji fundamentalno ograničenje ovakvog pristupa te da treba razmotriti neke nove metode koje bi možda dale bolje rezultate [2]. Wolfram, fizičar po struci i jedna od ikonskih figura polja ćelijskih automata, u svojim radovima i knjigama daje primjere novih sistema i njihovih primjena na mjestima gdje tradicionalni naučni pristup ne uspijeva dati zadovoljavajući odgovor (pogledati npr. [4] i [2]).

Ukoliko osmotrimo prirodne konstrukte oko nas, možemo primijetiti da postoji nekoliko osnovnih karakteristika koje možemo uočiti. Za početak, koncept lokalne interakcije široko je rasprostranjen s obzirom da u toku razmatranja nećega uzimamo u obzir uticaje samo onih elemenata koji su vremenski i prostorno dovoljno blizu da bi mogli proizvesti znacajne efekte na ishod ponašanja. Naravno da je moguće da neki dalji objekat ima uticaj. Međutim, kako većina posmatranih pojava zadovoljava svojstvo linearnosti gdje mala promjena u početnim uslovima izaziva i malu promjenu u rezultatima (za razliku od kaoticnih sistema koji se rjedje srecu), to je posmatranje svakodnevnih sistema kroz pretežno lokalnu interakciju poprilično opravdano. Na primjer, prilikom razmtranja hemijske reakcije na molekularnom nivou, nije potrebno uzimati u obzir Jupiterovu

gravitacionu silu, iako je istina da ona vrši uticaj na sistem, ma koliko on malen bio (iz ovih razloga potpuno izoliran sistem je čisto teoretski konstrukt).

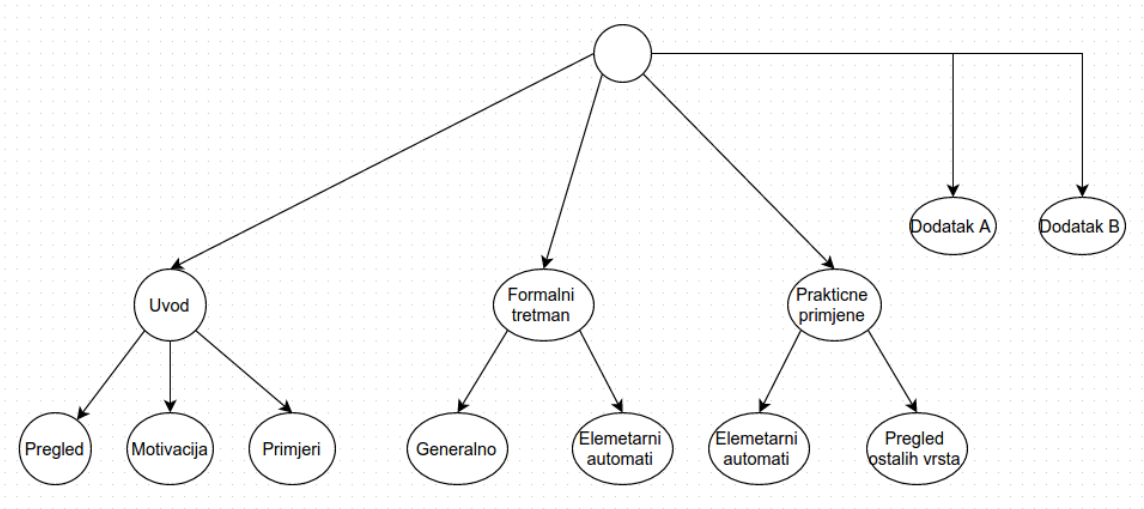
Nadalje, uprkos pretežno čisto lokalnoj interakciji, sistemi ispoljavaju poprilično kompleksna svojstva kao cjeline. Tako da iz čisto lokalnih svojstava nastaju globalna svojstva u kojima učestvuju svi dijelovi. Ovo možemo najbolje shvatiti kroz primjer ljudskog ponašanja kao vrste, gdje ljudi vrše razmjenu informacija i interakciju jedni među drugima lokalno, međutim i ljudsko društvo u cjelini možemo okarakterizirati nekim globalnim osobinama višeg nivoa. Ili npr. sistemi idealizovanih gasova sastavljeni od čestica koje vrše medjusobno mali prostorni uticaj, ali alatima statističke mehanike i sl. možemo okarakterizirati osobine cijelog volumena gasa [5].

Enormna kompleksnost nedvojbeno je dio sistema koji se uzimaju u naučna razmatranja i većine našeg okruženja. Ovo ide toliko daleko da su neke kompleksnosti i dalje ostale nerazjašnjene i uz korištenje najmodernijih tehnika i metoda naučnih disciplina. Svakodnevno se susrećemo kompleksne strukture koje je danas teško precizno modelirati, npr. propagacija toplote, modeli kretanja fluida ili pak nešto što se na prvi pogled čini kao jednostavna stvar poput formiranje oblika snježnih pahulja ili oblici formirani na morskim školjkama [2]. Jedan od osnovnih ciljeva naučnih istraživanja upravo "pripitomljavanje" ovih kompleksnosti i pokušaja objašnjenja istih kroz skup jednostavnijih principa. Postavlja se pitanje da li kompleksnost obavezno zahtijeva i kompleksnost na nižim nivoima i strukturama sistema, ili je nekako moguće, iako kontraituitivno, da poprilično jednostavna pravila mogu da proizvedu ogromnu složenost koja se može uočiti.

Postavlja se pitanje kako spojiti ove naizgled kontradiktorne osobine koje uvidjamo. Kako to da s jednog strana spektra postoji čisto lokalna interakcija, ali nakon što pogledamo sistem sa višeg nivoa apstrakcije uvidjamo da se i sam sistem ponaša kao cjelina koja vrši nešto globalniju interakciju sa sistemima istog nivoa? Također, kako je moguće da ogromnu kompleksnost koju posmatramo pokušavamo svesti na jednostavna pravila koja će je opisati i iz kojih ova kompleksnost može da se izrodi?

Nećemo ići toliko daleko da kažemo, kao što Wolfram tvrdi u svojoj knjizi, da ćelijski automati mogu da posluže da modeliraju cijeli univerzum [2], međutim očigledno je da ćelijski automati i njihovo razumijevanje može dovesti bar na pravi put razjašnjenja nekih od navedenih pitanja. ćelijski automati po svojoj definiciji se baziraju na čisto lokalnoj interakciji, međutim kada se pusti nekoliko koraka simulacije određenih pravila čak i jednodimenzionalnih instanci, uočavaju se globalne osobine. Tako da bi iz tog razumijevanja mogli doći bliže razumijevanju nastajanja globalnog ponašanja iz lokalno ograničenog. S druge strane, pravila ćelijskih automata poprilično su jednostavna, pa su čak i antički narodi mogli doći do istih [2]. Međutim iako su pravila jednostavna, nivo kompleksnosti koji se javlja uopšte nije niskog nivoa. Tako da i s te strane, ćelijski automati i njihovo razumijevanje može da nas pribiliži odgovoru povezniće između nastanka kompleksnosti iz malobrojnog skupa jednostavnih pravila.

1.2 Okvir rada i pregled sadržaja poglavlja



Ovdje ćemo pokušati da prođemo kroz strukturu i način organizacije rada, te ćemo dati kratki pregled poglavlja i njihovih sadržaja.

Na slici iznad dat je šematski prikaz organizacije rada, gdje je strukturom stabla dat način podjele tematskih cjelina, gdje u stablu svaki čvor označava tematsku cjelinu, dok djeca svakog čvora predstavljaju podteme veće tematske cjeline.

Rad je podijeljen na tri poglavlja, te dva dodatka, što je uočljivo i sa šeme. Dodaci služe kao suplementarni materijali čitaocu za praktičan dio rada, dok je ključna tematika oblasti ćelijskih automata obradjena u tri glavna teoretska poglavlja.

Poglavlja su redom *Uvod*, *Formalni tretman* te *Praktične primjene*. Imena poglavlja su poprilično intuitivna, te poglavlja redom predstavljaju neformalni uvod u tematiku, formalni matematički tretman i osobine ćelijskih automata, te prikaz nekih od mogućih praktičnih primjena ove vrste modela.

Poglavlje *Uvod* služi da bi se dobila početna slika vrste sistema koja se izučava, te da se prodje kroz neke od najpoznatijih primjera. Također, u poglavlju uvod razmatra se i motivacija i razlog izučavanja ove specifične teme. Poglavlje služi kao neformalni uvod u tematiku, te ne koristi formalni aparat da bi razmotrilo ćelijske automate i njihove osobine, već se kroz niz grafičkih primjera pokušavaju nagovijestiti osobine ovih modela.

Poglavlje *Uvod* u sebi sadrži podpoglavlja *Pregled*, *Motivacija* i *Primjeri*, koja redom predstavljaju pregled osnovnih primjera i historije ćelijskih automata, razloge za izučavanje ove vrste sistema, te niz grafičkih slikovitih primjera kompleksnih struktura generisanih ćelijskim automatima.

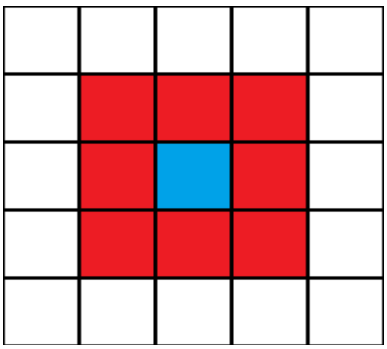
Naredni dio, *Formalni tretman*, izučava neke od osobina navedenih u prvom poglavlju, kao i neke dodatne kroz formalni matematički pristup uz strogo definisanje korištenih pojmova. Daje se pregled osnovnih i najbitnijih rezultata naučnih radova u oblasti. Korišteni radovi su navedeni u referencama, kao i lokalno na nivou teksta gdje se rad referencira ukoliko se koriste rezultati navedeni u njemu. Ovaj dio rada prvo prolazi kroz definisanje preliminarnih matematičkih pojmova potrebnih za razumijevanje daljnjih razmatranja, te generalnih definicija vezanih za ćelijske automate. Nakon toga, s obzirom na najvecu istraženost, posebna pažnja data je klasifikaciji i izucavanju osobina binarnih jednodimenzionalnih ćelijskih automata – takozvanih elementarnih ćelijskih automata. Medjutim, uprkos jednostavnosti pokazuje se da i ovi sistemi mogu da ispoljavaju neke veoma kompleksne osobine.

Na kraju, u dijelu *Praktične primjene*, date su prvo potencijalne praktične primjene elementarnih automata posebno izučavanih u prethodnom poglavlju. Također dat je i neformalni pregled modela koji uključuju automate koji ne spadaju u klasu elementarnih, ali imaju široku primjenu u raznim oblastima nauke.

Kako su kroz rad korišteni rezultati grafičkih simulacija softvera, *Dodatak A* ukratko opisuje arhitekturu i specifičnosti implementacije istog. *Dodatak B* sadrži samo listing koda softvera obradjenom u *Dodatku A*.

1.3 Osnovni prikaz koncepta

Počnimo prvo od pokušaja shvatanja kakve vrste modela predstavljaju ćelijski automati. Zato ćemo prvo krenuti od konkretnog specifičnog primjera kroz koji je moguće shvatiti osnovne odlike ćelijskih automata na koje ćemo se kasnije nadograditi kako budemo gradili formalnu apstrakciju ovog konkretnog primjera.



Slika 1. *Ćelija (plavo) sa susjedstvom od 8 ćelija (crvene).*

Najjednostavniji takav je primjer je beskonačna dvodimenzionalna ravna ploča prekrivena kvadratima koje ćemo nazvati *ćelije*. Kvadrati se medjusobno dodiruju stranicama, te tako svaki kvadrat ima punu vezu preko stranice sa tačno četiri susjedne

ćelije. Za svaku ćeliju kažemo da može biti u dva stanja - on i off. Kako ćemo za prikaz automata pretežno koristiti grafičke interpretacije, ova dva stanja možemo "zakodirati" bojom same ćelije. Tako ćemo uspostaviti konvenciju da crna prestavlja on, dok bijela prestavlja off stanje. Sva trenutna razmatranja bit će formalno definisana kasnije u radu te ovaj dio razmatranja služi samo za intuitivni prikaz osnovnih ideja iza ovakvih vrsta modela.

Na *slici 1* imamo prikazanu dosad opisanu ćeliju sa svojim susjednim ćelijama, te svaka od njih ima svoje definisano stanje.

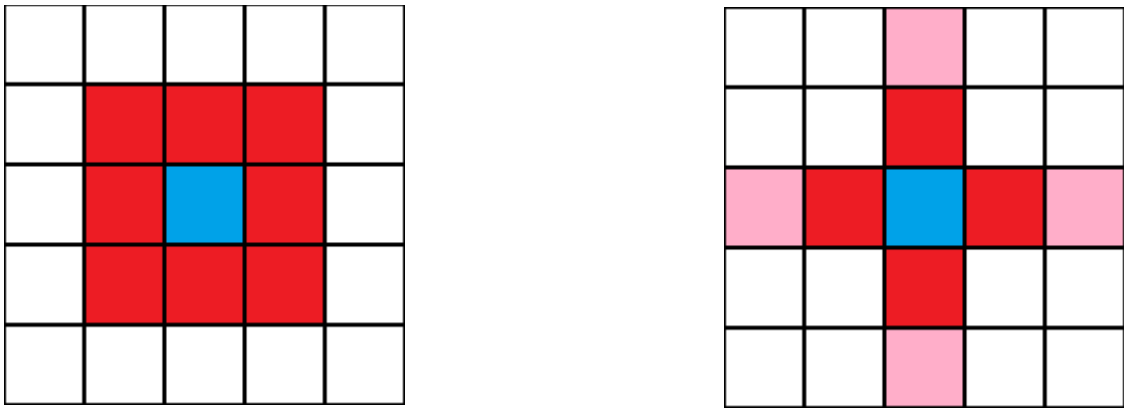
Slika 2. *Dvodimenzionalni grid "Igre života" sa početnom konfiguracijom [preuzeto sa http://dana.loria.fr/doc/game-of-life.html]. Uočljive su pravilne strukture u konfiguraciji.*

Na *slici 2* prikazan je skup ćelija koje zajedno čine 2d *grid*. Primijetimo da je nemoguće simulirati beskonačni grid konačnim metodama izračunavanja, pa se u praksi gotovo uvijek ograničavamo na konačne dimenzije. Ovdje nastaje problem šta raditi sa rubnim ćelijama, te će ta tematika biti detaljnije kasnije obrađena.

Ovakav raspored nazvat ćemo *konfiguracija*. U konfiguraciji, svaka ćelija ima svoje početno stanje, pa je tako za svaku ćeliju definisano da li je ona inicijalno on ili off – crna ili bijela. Stanje je na početku izabrano proizvoljno i može se mijenjati u zavisnosti od potreba, te će različita početna stanja dati nekada i drastično različita ponašanja. Na *slici 2* prikazan je primjer jednog takvog početnog stanja. Skup ovako organizovanih ćelija sa svojim početnim stanjima u konačnom gridu nazivamo *inicijalna konfiguracija grida*.

Naravno, dosadašnja definicija ćelijskih automata ne bi imala nikakvog smisla, s obzirom da imamo samo početnu konfiguraciju i grid. Međutim, ono što čini ćelijske automate pravim modelima koji se mogu koristiti u razne svrhe je takozvana *evolucija ćelijskih*

automata. Nakon što se uspostavi inicijalna konfiguracija grida, ovaj sistem može da se stavi u evoluciju. To znači da će svaka od ćelija da mijenja svoje stanje prema nekim pravilima, te će cijelokupan sistem da se mijenja prema tim pravilima u diskretnim vremenskim intervalima.

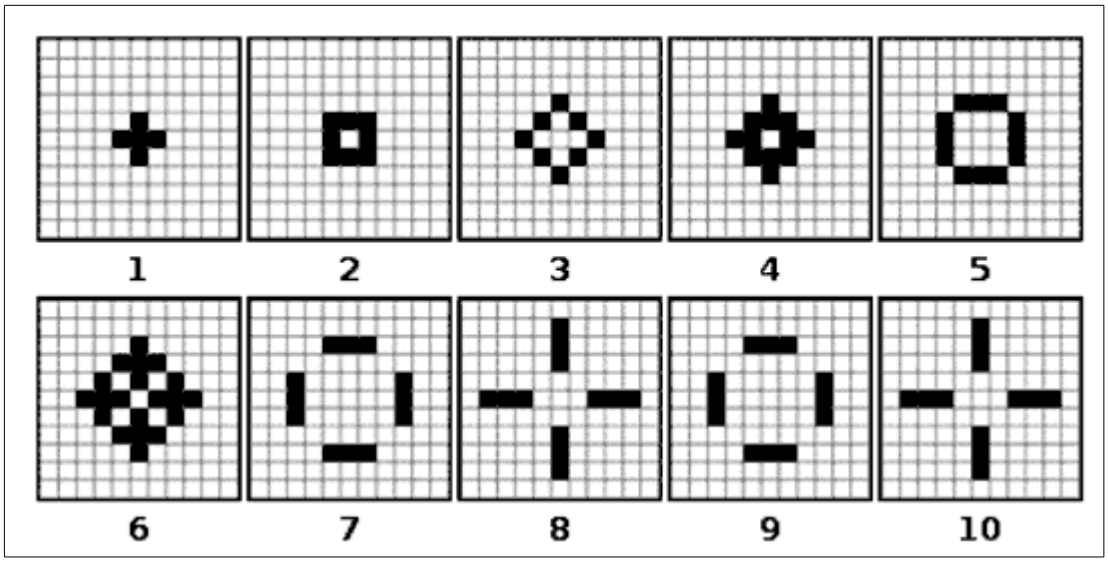


Slika 3. *Tipovi dvodimenzionalnih susjedstva. Moore-ovo (lijevo) i von Neumannovo susjedstvo (desno). Oba imaju primjene u zavisnosti od domena problema.*

Sljedeće stanje svake individualne ćelije u koje prelazi zavisi od njenog trenutnog stanja, kao i stanja okolnih ćelija, te se prema ovim parametrima i formiraju pravila. Susjedne ćelije koje okružuju datu ćeliju a uzimaju se u obzir prilikom rančunanja sljedećeg stanja kolektivno se nazivaju *susjedstvo (eng. neighborhood)*. Evidentno je da izbor susjedstva nije jedinstven. Na *slici 3* prikazano je nekoliko načina izbora susjedstva. Najpoznatija dva ovakva tipa su *Moore-ovo* i *von Neumann-ovo susjedstvo* prikazano na *slici 3*, ali nije isključeno i kreiranje proizvoljnog susjedstva.

Slika 4. *Grafički izbor pravila (prelaza stanja) u zavisnosti od stanja susjedstva jednodimenzionalnog automata sa dva moguća stanja – on/off (crno/bijelo na slici). [generisano sofverom pisanim u sklopu rada]*

Nakon što se izaberu koje ćelije učestvuju u formiranju susjedstva, formira se i skup pravila koji govori o tome kako ćelija evoluira na osnovu svog stanja i stanja svojih susjeda. Na *slici 4* je u naveden i način specifikacije pravila za jednodimenzionalno susjedstvo (jer bi broj mogućnosti za dvodimenzionalno bio ogroman) gdje se za svaku pojedinačnu kombinaciju susjedstva na osnovu trenutnog stanja ćelija bira iduće stanje iste. Tako sa slike možemo uočiti npr. da ukoliko je ćelija okružena sa dvije crne ćelije, prelazi u off stanje.

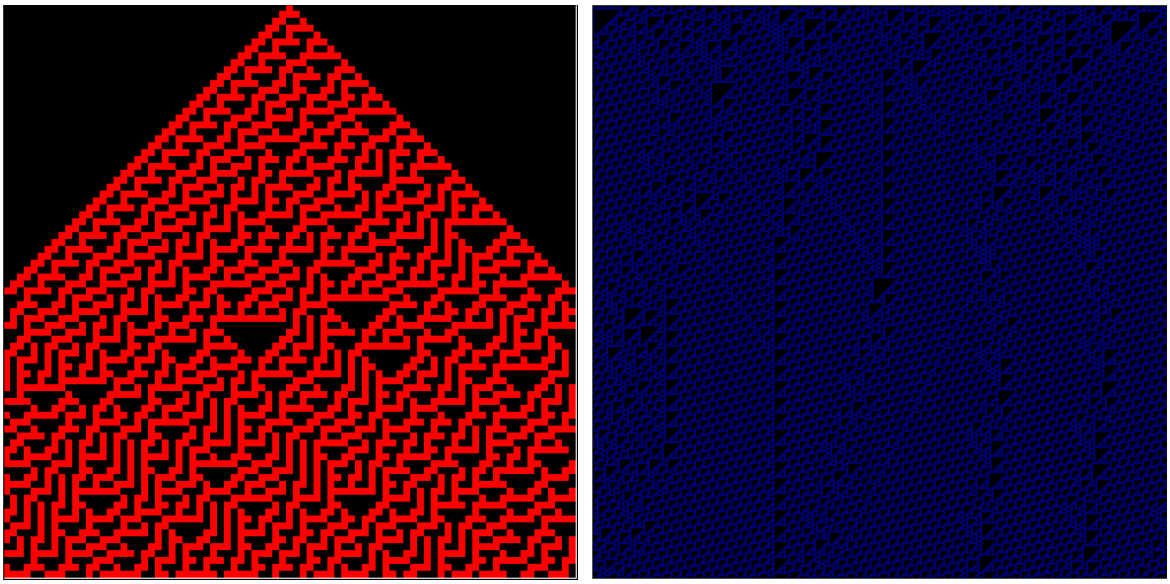


Slika 5. *Deset koraka evolucije specifičnih početnih uslova 10x10 grida "Igre života" [preuzeto sa https://datascienćelab.wordpress.com/2014/01/03/yet-another-game-of-life/]*

Kako su stanja za dosad razmatrane automate binarna, ukoliko n predstavlja broj susjeda koji formiraju susjedstvo, tada je moguće $2^{2^{n+1}}$ mogućih pravila evolucije. Tako za jednostavno Moore-ovo susjedstvo od 8 susjednih ćelija postoji $2^{2^{8+1}} = 2^{2^9} = 1.34 \times 10^{154}$ mogućih pravila evolucije. Ovo opažanje će biti kasnije detaljnije objašnjeno.

Na *slici 5* data su tri koraka evolucije sistema sa jednim od specifičnih pravila, te je ovo dobro poznata konfiguracija nazvana *"Game of Life"* ili *"Igra života"*.

Vidimo da opisana čisto lokalna interakcija može da kreira poprilično kompleksne oblike, te će se dalje ispitati kolika je ta kompleksnost i na koji način se mogu koristiti ova zapažanja za neka generalnija izračunavanja.



Slike 6. i 7. *Kompleksne strukture generisane jednostavnim pravilima [generisano sofverom pisanim u sklopu rada]*

Na *slikama 6 i 7* prikazana je kompleksnost koja može da proiziđe iz jednostavnih pravila prezentovanih iznad.

1.4 Historija

Da bismo dobili opštu sliku oblasti, te razloga nastanka i razvoja iste, potrebno je istražiti kako je došlo do pitanja i problema koja su dovela do toga da se naučna javnost zainteresira za ovu vrstu sistema.

Kao i sve ostale oblasti nauke i matematike, tako su i ćelijski automati nastali pokušajem odgovaranja na specifičan skup pitanja koji se kasnije proširio u cijelu oblast nakon što se uvidjela njihova moguća generalnost i opštija primjena.

Pioniri ove konkretne oblasti su bili 1940-ih istrazivači u *Los Alamos* Nacionalnoj Laboratorji *Stanislaw Ulam* i *John von Neumann*. Zanimljiva je činjenica da su se bavili ovom oblasti sa strane u vidu hobija. Oni su se, posebno von Neumann, inicijalno zanimali pokušajem da naprave sistem u kojem bi bilo moguće da entitet unutar samog sistema vrši *samoreplikaciju*. Ovo bi značilo da određeni objekat pravi identičnu kopiju samoga sebe, te bi ta kopija također pravila svoju kopiju ad infimum.

Prvobitno je von Neumann želio da kreira robota koji bi vršio samoreplikaciju (što danas nazivamo *kinematskim* – realnim modelom samoreplikacije). Međutim nakon teškoća pri nalaženju ogromnog broja dijelova koje bi robot morao da ima na raspolaganju da bi

izvršio ovaj zadatak, te nakon prijedloga Ulama, odlučio je da iskoristi apstraktni diskretni model za ovaj zadatak. Model koji je koristio može se smatrati prvim primjerom korištenja ćelijskih automata.

Ovo predstavlja početak oblasti diskretnih sistema ćelijskih automata. Rezultat je bio ono što danas nazivamo von Neumann-ov univerzalni konstruktor. On se sastoji od ćelijskih automata koji imaju 29 stanja i potrebno je ~200.000 ćelija da bi se izgradio univerzalni konstruktor. Von Neumann je dao okvirni dizajn i dokaz postojanja, ali nikada nije implementirao ovaj sistem [1]. Tek 1990-ih godina je grupa predvođena italijanskim naučnikom *Pasaventom* uspjela da napravi pravu implementaciju ovoga sistema, iako je ideja konceptualno začeta gotovo 50 godina prije prve implementacije.

Nešto kasnije, 1950-ih, ista dvojica naučnika iz Los Alamos laboratorija iskoristili su ćelijske automate u prvom pokušaju modeliranja realnosti koristeći iste. Kreirali su model koji *predviđa kretanje fluida* na način da smatraju fluid sastavljenim od diskretnih jedinica – ćelijskih automata, čije kretanje zavisi od susjednih jedinica. Na ovaj način moguće je aproksimirati kretanje cjelokupnog fluida modeliranjem samo lokalne interakcije susjednih čestica. Ovim modelom pokazano je da ćelijski automati imaju i širu primjenu van čisto teoretskih razmatranja za koja su ranije korišteni, te ovo predstavlja svojevrsni početak generalne primjene ove vrste modela u nauci.

Trebalo bi napomenuti i da je neke od ranih istraživanja u ovom polju vršio i pionir u oblasti vjestačkog života, norveško-italijanski naučnik *Nils Aall Bariccelli* koji je jedan od prvih prepoznao potencijalnu univerzalnost ćelijskih automata kao modela koji mogu predstavljati realne pojave.

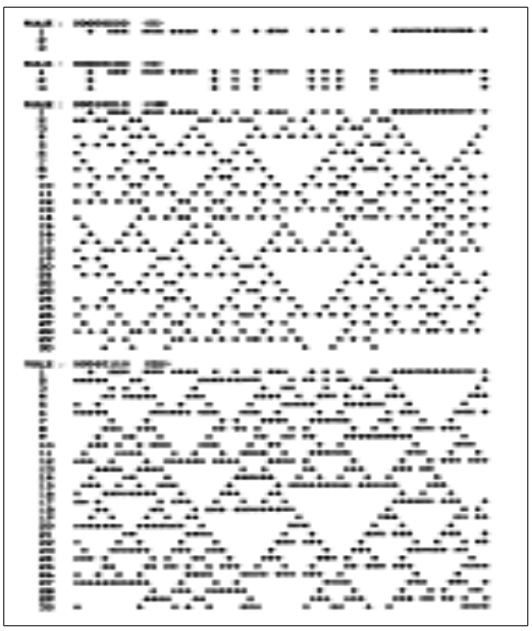
Još neku od ranih primjena ćelijski automati našli su u modeliranju *propagacije talasa u medijima*. Rani ovakav model ćelijskih automata konstruisan je 1940-ih. Međutim kako je taj model koristio kontinualnu funkciju kao signal, ne može se smatrati diskretnim modelom ćelijskih automata, tako da su prvi pravi ovakav model korišten u svrhe modeliranja impulsa kardio sistema u čovječijem tijelu konstrukisali *J. M. Greenberg* i *S. P. Hastings* 1978-e. Ovaj model je i dalje često korišten i referenciran u istraživačkim radovima.

Prvih dvadeset godina od pojavljivanja modela ćelijskih automata, gotovo niko nije izvršavao rigorozno naučno i matematičko ispitivanje osobina ovih sistema. Jedan od pionirskih radova u polju bio je rad američkog matematičara *Gustava A. Hedlunda*, koji je kroz matematičku oblastu *dinamika simbola* (koju je sam i osnovao, eng. *symbolic dynamics*) posmatrao ćelijske automate kao mijenjajuće nizove karaktera uz određena pravila prelaza. Ovime je došao do nekih od najkorisnijih rezultata u ovom polju. Njegov rad iz 1969-e zajedno sa *Curtis-Hedlund-Lyndon teoremom* za koju snosi djelomične zasluge, koja klasificira globalni prostor pravila automata, i dalje predstavlja jednu od osnova za bilo koga ko planira da se upusti u ozbiljnije ispitivanje ove vrste sistema.

Pravu popularizaciju oblast je doživjela pametno konstruisanim primjerom od strane

britanskog matematičara i fizičara *John Conway*-a 1970-e godine. On je svoj specifičan model baziran na dvodimenzionalnim ćelijskim automatima prikladno nazvao "*Igra života"* (eng. "*Game of Life"*). Ovaj model je standardni dvodimenzionalni sistem ćelijskih automata sa dva stanja, međutim uprkos jednostavnim pravilima, nakon što se sistem pusti u rad, počinju da se pojavljuju visoko kompleksne strukture koje ispoljavaju osobine koje bi mogli pripisati i nekim živim bićima, kreću se, jedu jedni druge, razmnožavaju se i slično. Zbog ovoga je model i dobio svoje ime. Upravo zbog ovih osobina gdje se visoka kompleksnost javlja iz poprilično jednostavnih pravila, *"Igra"* a samim tim i oblast ćelijskih automata doživjela je veliku popularizaciju, te većina ljudi za ćelijske automate sazna upravo iz ovog primjera. Iako se ovaj model smatrao pretežno dijelom rekreativne matematike te sredstvom popularizacije ideje ćelijskih automata za širu javnost, nešto kasnije je Berlekamp u saradnji sa još nekoliko matematičara dokazao univerzalnost *"Igre života"*, tj. da sistem može ekvivalentno da se koristi u svrhu univerzalnog izračunavanja kao i bilo koji drugi računar prema Turingovoj tezi.

Možemo napomenuti da je i njemački računarski pionir *Konrad Zuse* 1969-e u svojoj knjizi *Računajući svemir* raspravljao neke od širih filozofskih implikacija sistema ćelijskih automata. On je naveo da je moguće da cijelokupan univerzum zapravo jedan veliki ćelijski automat koji se sinhrono osvježava u vremenskim koracima, te je ova ideja otvorila prostor za potpuno novu oblast nazvanu *digitalna fizika*.



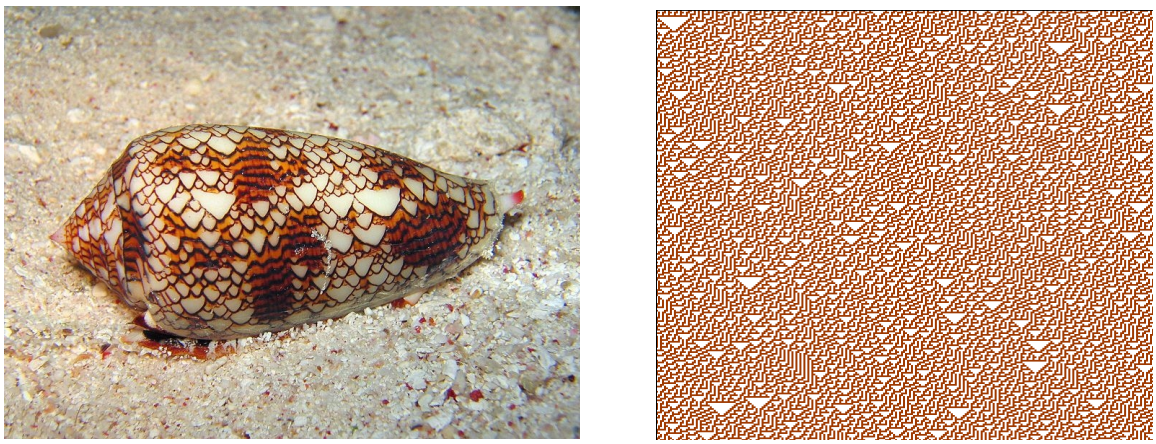
Slika 8. *Neki od prvih simulacija sistema ćelijskih automata od strane Stephena Wolframa. [2]*

Najdetaljnije i rigoroznije ispitivanje osobina ćelijskih automata izvršio je u svojim radovima tokom dvadeset godina istraživanja britanski matematičar i fizičar *Stephen Wolfram*, koji se smatra jednom od najbitnijih figura za ovu oblast. Počeo je sa svojim

istraživanjima 1981-e u pokušajima da razmotri kako se kompleksni uzorci u prirodi formiraju naizgled narušavajući drugi zakon termodinamike. Tada je izvršavao simulacije na ranim računarima, te nakon što je u simulacije unio određene klase ćelijskih automata, bio je zapanjen kolika kompleksnost proizilazi iz jednostavnih pravila koje je postavio (*slika 8*). Ovo naizgled kontraintuitivno ponašanje koje ga je zapanjilo navelo ga je da u idućim decenijama prebaci svoju sferu rada sa fizike na matematiku i kompjutersku nauku. U seriji od preko dvadeset radova Wolfram je izvršio klasifikaciju i opisao osobine pretežno jednodimenzionalnih ćelijskih automata, te predložio mnoge njihove primjene kao alternativu trenutno korištenim modelima poput parcijalnih diferencijalnih jednačina. Također je doprinio dokazivanju univerzalnosti jednog od pravila jednodimenzionalnih ćelijskih automata. Svoje pronalaske, stavove i historiju istraživanja kompajlirao je 2002. u knjizi *Nova vrsta nauke* (eng. *A New Kind of Scienće*), gdje se zalaže za ćelijske automate kao budućnost modeliranja prirodnih pojava te bilo kakve vrste apstraktnih sistema [2]. Dotiče se i filozofskih implikacija ćelijskih automata. Wofram i dalje nastavlja da popularizira ovu temu kroz serije govora, te je poznat kao i kreator *Wolfram Alpha* i *Mathematica* softverskih paketa.

1.5 Primjeri

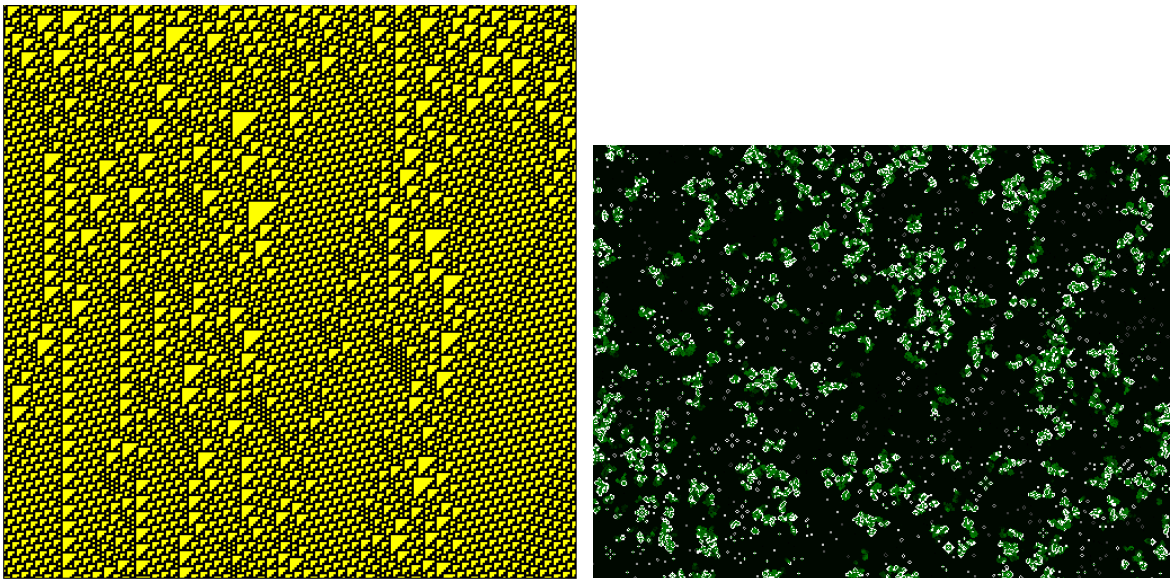
Da bi se formirala kompletnija opšta slika ćelijskih automata kao modela kompleksnosti, prikladno bi bilo dati nekoliko primjera kompleksnosti i vrsta struktura generisanih ćelijskim automatima. Ovo može dati jasniju sliku i nekome ko nije toliko zainteresovan u matematičke detalje samog modela na način da može prikazati raznovrsnost struktura koje ćelijski automati mogu proizvesti.



Slika P1. *Dva primjera kompleksnih struktura. Prirodno generisana struktura oklopa školjke (lijevo) i elementarno pravilo 30 (desno). Uočljiva je sličnost između ova dva naizgled nepovezana sistema.*

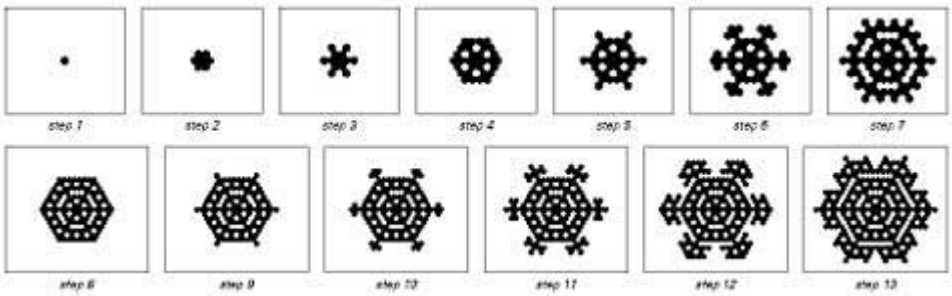
Na *slici P1* možemo uočiti sličnost između strukture morske školjke generisane prirodnim putem i evolucijom elementarnog pravila 30. Smatralo se da kompleksne strukture uzoraka na životinjskim ljušturama moraju nastati kao rezultat nekog

kompleksnog procesa, međutim ako pogledamo strukture generisane jednostavnim pravilom elementarnih ćelijskih automata, možemo vidjeti da i najjednostavnija pravila mogu da proizvedu strukture velike kompleksnosti. Zbog ovoga je oblast ćelijskih automata jedna od zanimljivijih s obzirom da povezuje naizgled nespojive cjeline lokalne jednostavnosti i globalne kompleksnosti [2].



Slika P2. *Struktura generisana pravilom 110 (lijevo) [generisano softverom] i "Igra života" (desno) [preuzeto sa http://www.marekfiser.com/Projects/Conways-Game-of-Life-on-GPU-using-CUDA]*

Na *slici P2* s lijeve strane možemo vidjeti strukture generisane elementarnim pravilom 110 za koje je postulirano i kasnije dokazano da je kompleksno kao i bilo koji sistem univerzalnog izračunavanja. S desne strane možemo vidjeti jednu konfiguraciju Conwayove Igre života simulirane na CUDA grafičkoj kartici sa milionima individualnih ćelija koje izgledaju kao da formiraju grupacije i nešto kao živuće strukture.



Slika P3. *Generisanje kompleksne strukture snježnih pahuljica pomoću ćelijskih automata [preuzeto sa http://radicalart.info/AlgorithmicArt/grid/cellular/2D/]*

Formiranje kristala snježnih pahuljica nije još dovoljno izučena oblast s obzirom da nemamo rješenja pojedinih diferencijalnih jednačina propagacije toplote, međutim pokazuje se da jednostavni dvodimenzionalni model ćelijskih automata može da generiše strukture koje poprilično oslikavaju izglede pravih pahuljica bez potrebe da riješe te jednačine, kao što se može vidjeti na *slici P3*.

2. Formalni matematički tretman

Nakon neformalnog uvoda, potrebno je i rigorozno matematički definisati šta se misli kada se koristi pojam ćelijskih automata. Zbog raznovrstnosti modela koji se mogu smatrati ćelijskim automatima, u literaturi ne postoji opšteprihvaćena generalna definicija, već autori strogo definišu samo konkretan model kojim se bave, npr. jednodimenzinalni binarni ćelijski automati u slučaju Stephena Wolframa (vidjeti npr. [2], [3], [4]).

Na početku bit će data generalna definicija koja pokušava što opštije da pokrije sve diskretne modele koji se mogu smatrati ćelijskim automatima, mada su moguće neke iznimke zbog širine diskretnih modela i mogućnosti proširenja osnovnog modela po raznim parametrima.

Nakon toga će se redom proći kroz specifične instance poput jednodimenzionalnih i dvodimenzionalnih binarnih automata koje su najviše izučavane tokom godina. Obratit će se pažnja i na njihovu formalnu definiciju, tretman i matematičke osobine.

Također, definicije i teoreme, te osobine automata bit će popraćene primjerima i grafičkim simulacijama iz implementacijskog dijela rada koji će sam po sebi biti kasnije pokriven.

2.0 Preliminarne matematičke definicije

U ovoj sekciji potrebno je definisati nekoliko preliminarnih matematičkih pojmova koji će kasnije biti iskorišteni u izučavanju osobina ćelijskih automata. Čitatelj se trenutno ne treba zamarati ovim dijelom, već se po potrebi vratiti na njega ukoliko bude korišten neki od pojmova ovdje definisan.

U informacionoj teoriji, potrebno je na neki način kvantificirati količinu informacija u nekom sistemu, te za to koristimo koncept entropije.

Definicija 2.0.1 Za sistem sa n mogućih dogadjaja sa vjerovatnoćama dešavanja i-tog dogaaja p_i, entropija se definiše kao $= -\sum_{i}^{n} p_i log(\frac{1}{p_i})$.

Entropija će nam biti korisna u razmatranjima ćelijskih automata i njihovih statističkih osobina sa aspekta teorija informacija i kodiranja.

Kako su ćelijski automati zapravo specijalna klasa automata iz teorije izračunljivosti, to je veoma korisno gledati njihove osobine i kroz ovu naučnu oblast, pa će nadalje biti navedene neke od osnovnih definicija potrebne za daljnja razmatranja.

Definicija 2.0.2 Turingova mašina definiše se kao 7-orka $M = (Q, \Gamma, b, \Sigma, \delta, q_0, F)$, gdje su oznake redom:

- Q je skup stanja
- Γ je skup simbola
- b predstavlja specijalni 'blank' simbol
- $\Sigma \subseteq \Gamma \setminus \{b\}$ skup ulaznih simbola
- $\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ funkcija prelaza stanja
- q_0 početno stanje
- F skup prihvaćenih stana

Turingova mašina predstavlja način formalizacije svih funkcija koje nazivamo izračunljivim. Intuitivno, izračunljiva je funkcija koju čovjek može da obavi sa papirom i olovkom bez paženja na vremensko-prostorna ograničenja. Ovo je koristan koncept pri formalnim razmatranjima izračunljivosti i sposobonostima izračunavanja nekog sistema.

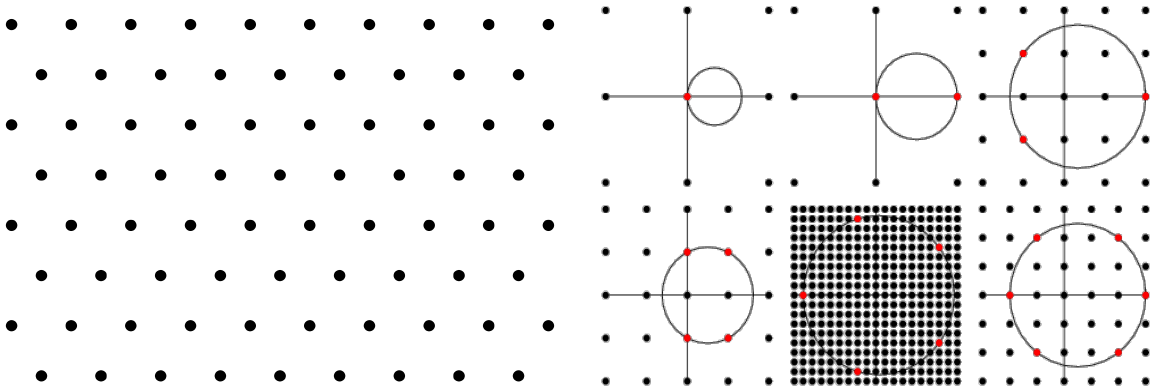
2.1 Generalna definicija ćelijskih automata

Pokušajmo prvo kroz nekoliko zapažanja iz konkretnih primjera uočiti koje su to generalne osobine i koncepti koje bi definicija morala da pokrije, a nakon toga pokušat ćemo to da uklopimo u neki matematički model.

Prvo idu neke preliminarne matematičke definicije kroz primjere koje su neophodne za razumijevanje krajnje definicije ćelijskih automata u generalnom slučaju. Pokušat ćemo kroz te definicije postepno proći kroz osobine apstraktnih pojmova za koje vežemo instance ćelijskih automata.

Sva dešavanja modela moraju da se odvijaju na nekom apstraktnom prostoru, koji treba da zadovoljava određena svojstva. Dakle prvi koncept koji definicija treba da pokrije je prostor odvijanja modela. Prostor odvijanja modela ćelijskih automata je u pravilu diskretan, što možemo da uočimo iz nekoliko dosad razmotrenih primjera s obzirom da se uvijek radilo u pravougaonoj matrici kvadratnih mjesta ili jednodimenzionalnom ekvivalentu istih. Također prostor mora da bude definisan na takav način da se na njemu mogu definisati i ostali bitni koncepti poput susjedstva.

Prvo onda definišimo prostor. Matematički objekat koji može da uhvati sve navedene osobine je latica ili rešetka (eng. lattice). Intuitivno rešetka je skup tačaka sa ravnomjernim i homogenim rasporedom kao što je prikazano npr. na *slici 9*. Naravno tu je i dosad poznati primjer na *slici 1* u prethodnom poglavlju.



Slika 9. *Primjer strukture rešetke [preuzeto sa http://mathworld.wolfram.com/CircleLatticePoints.html]*

Da bismo formalno predstavili šta je to ravnomjerno raspoređenje, rešetku možemo definisati kao skup tačaka kod kojih je udaljenost između bilo koje dvije od njih cjelobrojna linearna kombinacija unaprijed određenih vektora u prostoru gdje su tačke definisane (pretežno se radi sa skupom/prostorom R^n). Ključan detalj je cjelobrojnost jer time zadržavamo osobinu diskretnosti samog skupa, što je esencijalno za ćelijske automate kao diskretne modele izračunavanja.

Definicija 2.1.1 Latica ili rešetka se definiše kao $\Lambda := \left\{ \sum_i \mathrm{a}_i \; v_i \; : \; a_i \in \mathbb{Z} \right\}$ gdje je $v_i \in \{v_1, v_2, v_3, ...\}$ baza vektorskog prostora V nad skupom skupa gdje je rešetka definisana.

Ovo samo znači da izaberemo fiksan skup vektora udaljenosti od tačke i na tačku nadodjemo cjelobrojne umnoške vektora da bismo tako održali regularnost koja je zahtijevana za definiciju ćelijskih automata.

Ćelijski automati su kako prostorno tako i vremenski diskretni sistemi. U primjeru jednodimenzionalnih automata, vrijeme je bilo samo prirodan broj koji je govorio o kojoj se iteraciji automata radi, pa tako moramo moći definisati i diskretan koncept vremena u kojem automati evoluiraju kao dodatnu dimenziju pored prostorne koju smo već obradili.

Upravo iz diskretnosti dimenzije vremena za sisteme ćelijskih automata, javlja se i najjednostavniji način njegove definicije. Dovoljno je samo da se vrijeme može staviti u *jedan-na-jedan (bijektivno)* mapiranje sa skupom prirodnih brojeva, što se i uklapa u traženu diskretnost.

Definicija 2.1.2 Neka je dat skup T takav da postoji funkcija $b : T \to \mathbb{N}$ koja je bijektivna. Tada skup T nazivamo diskretnim vremenskim skupom.

Da bi se izbjegle zabune, zbog postojanja bijektivnosti moguće je u daljnjim razmatranjima umjesto apstraktnog skupa T koristiti skup prirodnih brojeva.

Nakon što smo definisali prostorne i vremenske dimenzije u kojima će ćelijski automati biti smješteni, još jedan osnovni koncept je stanje svakog automata koji se nalazi u prostor-vremenu.

Za jednodimenzionalni i dvodimenzionalni slučaj imali smo samo dva moguća stanja – 1 ili 0, crno ili bijelo u grafičkoj reprezentaciji. Na osnovu ovoga, svaka ćelija mora biti u jednom od konačnog broja stanja, što je i jedino ograničenje na ovaj skup, da mu je kardinalnost konačna, tj. odgovara nekom prirodnom broju. Moguće su i neke generalizacije u određenim primjenama samog skupa stanja gdje npr. imamo ćelijske automate čija stanja odgovaraju realnim brojevima u intervalu $[0, 1]$ što očigledno nije konačan skup stanja.

Definicija 2.1.3 Skup Q nazivamo skupom stanja ukoliko $\mid Q \mid = n$, gdje $n \in \mathbb{N}$.

Potrebno je još razmotriti šta bi bilo povoljno da se koristi u definiciji susjedstva te prelaza (evolucije) svake ćelije unutar automata. Ovi koncepti ništa ništa do prostorne i vremenske transformacije konkretne ćelije. Npr. susjedstvo je samo mapiranje/dodjeljivanje nekoliko ćelija jednoj konretnoj ćeliji u rešetki. Evolucija je samo dodjeljivanje idućeg stanja ćeliji na osnovu prije dodijeljenog susjedstva i prijašnjeg stanja. Tako da matematičke funkcije ili mapiranja savršeno odgovaraju opisu objekta koji bi mogao da se koristi, što će i biti slučaj.

Sada je potrebno dati definiciju ćelijskih automata čija se evolucija odvija na diskretnom prostoru i u diskretnom vremenu sa konačnim brojem stanja uz postojanje susjedstva svake ćelije i tranzicijskog pravila za istu. Vidimo da smo pokrili sve koncepte koje smo ranije vidjeli u specifičnim instancama, tako da je moguće dati generalnu definiciju.

Definicija 2.1.4 Ćelijski automat možemo definisati kao uredjenu šestorku $C := (\Lambda, Q, T, \sigma, \eta, \phi)$, pri čemu je:

- Λ rešetka nad nekim skupom
- Q skup stanja
- T diskretni vremenski skup
- $\sigma : \Lambda \times T \to Q$
- $\eta : \Lambda \to \Lambda^c$
- $\phi : Q^{c+1} \to Q$ $(c \in \mathbb{N} \wedge c \geqslant 1)$

Funkcije σ, η i ϕ nazivaju se funkcije konfiguracije, susjedstva i prelaza respektivno. Prirodni broj c se naziva veličina ili kardinalnost susjedstva. Funkcija σ je rekurzivno definisana pomoću funkcije η kao
$\sigma(\lambda, n) = \phi(\sigma(\eta(\lambda), n-1), \sigma(\lambda, n-1))$
gdje $\lambda \in \Lambda, n \in T$, ali možemo uz prije navedenu diskretnost skupa T smatrati da $n \in \mathbb{N}$.

Potrebno je ukratko pojasniti detaljniju intuiciju iza funkcija σ, η, ϕ u *Definiciji 3.4*.

Funkcija σ predstavlja funkciju trenutne konfiguracije, što znači da pridružuje stanje iz skupa stanja svakoj ćeliji u datoj vremenskoj instanci.

Funkcija η predstavlja funkciju susjedstva, te ona jednostavno pridružuje svakoj ćeliji onoliko ćelija koliko je definisano veličinom susjedstva.

Na kraju, funkcija ϕ predstavlja funkciju prelaza ćelija ćelijskog automata, te ona definiše u koje sljedeće stanje prelazi ćelija ćelijskog automata uzimajući u obzir stanja njenog susjedstva, te njeno trenutno stanje. Definisana je rekurzivno, jer svaka iteracija zavisi samo od prethodne, a veoma je moguće da je pojedina ili čak većinu pravila prelaza nemoguće predstaviti elementarnim funkcijama na koje smo navilkli. Ovo se već dalo zaključiti iz kompleksnosti uzoraka koje ova vrsta sistema generiše.

Može se primijetiti da ćelijski automati posjeduju nekoliko osobina koje su zadovoljene za svaku instancu istih. To su homogenost, paralelizam i lokalnost. Ovo je s obzirom da sve ćelije evoluiraju paralelno po istim pravilima koja su određena samo lokalnim susjedstvom i stanjem ostalih ćelija u njemu [1].

Ovim je završeno razmatranje generalne definicije ćelijskih automata, te će nadalje biti izučavane najinteresantnije i najistraženije instance istih koje ćemo pojedinačno pokušati uklopiti u ovaj model, te detaljno razmotriti specifične matematičke osobine svake od tih

instanci.

2.2 Binarni jednodimenzionalni (elementarni) ćelijski automati

Najosnovniji tip sistema ćelijskih automata predstavljaju binarni jednodimenzionalni ćelijski automati. Ovo znači da svaka individualna ćelija posjeduje dva moguća stanja – binarni, te da svaka ćelija ima tačno dva susjeda, te da su ćelije medjusobno poredane u jednodimenzionalnoj rešetki – jednodimenzionalni.

Međutim, uprkos njihovoj jednostavnosti, upravo ova vrsta predstavlja najizučavaniji tip ćelijskih automata. Ovo možda možemo pripisati činjenici da se cjelokupna nauka vezana za ove vrste sistema i bazira na što jednostavnijim gradivnim elementima koji produkuju kompleksne strukture, pa su tako najjednostavniji od njih vjerovatno i najzanimljivi za istražiti. Ovo nije slučaj za većinu drugih oblasti sa skroz drugim pogledom na kompleksnost.

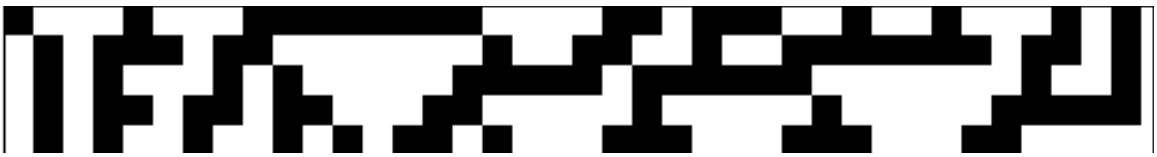
Drugi naziv koji se koristi u literaturi za ovu vrstu ćelijskih automata je elementarni ćelijski automati, tako da će ova dva termina u daljem tekstu biti korištenja naizmjenično.

2.2.1 Definicija i način enumeracije pravila

Pokušajmo sada dati definiciju i prikazati kakvu vrstu sistema predstavljaju binarni jednodimenzionalni ćelijski automati. Najjednostavnije je shvatiti ove sisteme kao niz ćelija poredanih jedna do druge u liniji – zbog čega se i zovu jednodimenzionalni, prilikom čega svaka ćelija može biti u jednom od dva stanja – on ili off, što grafički predstavljamo crno i bijelom bojom respektivno. Generalno, grafički prikaz je često korišten alat u izučavanju ćelijskih automata jer može da da neke indikacije o osobinama izučavanih sistema.

Slika 10. *Primjer proizvoljne početne konfiguracije jednodimenzionalnog ćelijskog automata sa dva stanja [generisano softverom]*

Na *slici 10* dat je prikaz jednog ovakvog niza ćelija u liniji gdje svaka ćelija ima slučajno dodijeljenu vrijednost. Ovakav skup ćelija sa odgovarajućim stanjima naziva se konfiguracija. Svaka ćelija evoluira, tj. mijenja svoje stanje u idućem koraku prema određenom pravilu koje zavisi od stanja te konkretne ćelije, kao i stanja nekih okolnih ćelija. Izmjena stanja svih ćelija vrši se paralelno, tako da cjelokupan niz – sistem evoluira paralelno. Kasnije ćemo vidjeti generalan način klasifikacije ovih pravila evolucije i njihovog definisanja. Upravo je i ovaj paralelizam često korišten argument u zagovaranju praktičnih primjena ćelijskih automata u sistemima paralelnih procesiranja (pogledati npr. [5]).



Slika 11. *Pet koraka evolucije početnih uslova jednodimenzionalnog ćelijskog automata sa dva stanja. Specifično pravilo evolucije primijenjeno je pravilo 30 [generisano softverom]*

Na *slici 11* nadalje prikazano je pet koraka evolucije sistema prema unaprijed izabranom pravilu, gdje svaki red predstavlja sljedeću konfiguraciju sistema. Prvi red nazivamo početna konfiguracija, te svaki sljedeći red nastaje opisanim postupkom paralelne evolucije ćelija zasebno. Evolucija se može nastaviti u nedogled, dok je ovdje postupak izvršen tek pet puta. Primijećujemo da je evolucija sistema izvršena u diskretnim vremenskih koracima, što je još jedna od navedenih ključnih osobina ćelijskih automata.

Sada je potrebno doći do formalne definicije jednodimenzionalnih binarnih ćelijskih automata koja se javlja kao posebna instanca nešto prije obrađene generalne definicije. Također, potrebno je i naći način da se struktuirano navedu pravila izbora načina evolucije ovih sistema.

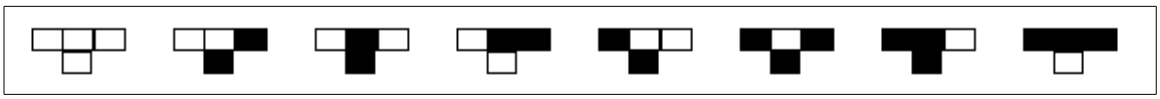
Definicija 2.2.1 Jednodimenzionalni binarni ćelijski automat možemo definisati kao uredjenu trojku (C, σ, ϕ), gdje C predstavlja skup ćelija koje su prebrojive, tj. moguće je izvršiti enumeraciju istih. Radi jednostavnosti, moguće je umjesto C koristiti skup $\mathbb{Z}$. Funkcija σ predstavlja mapiranje $\sigma : \mathbb{N} \times \mathbb{Z} \to \{0, 1\}$ i naziva se pravilo evolucije. Funkcija ϕ predstavlja mapiranje $\phi : \{0, 1\}^n \to \{0, 1\}$, gdje se n naziva veličina susjetstva. Mapiranje σ definisano je rekurzivno kao: $\sigma(t+1, i) = \phi(\sigma(S_{j \in J(i)} \sigma(t, j)))$, gdje $J(i)$ predstavlja susjedstvo konkretne ćelije izabrano prema nekom pravilu, te $\phi(0, i) \in \{0, 1\}$ nazivamo početnom konfiguracijom sistema.

Ovu potpuno formalnu definiciju intuitivno možemo shvatiti na način da se za svaku posebnu ćeliju, koja je označena cijelim brojem koji je jedinstveno identifikuje s obzirom na raspored ćelija, prema određenom pravilu prvo bira susjedstvo. Susjedstvo predstavlja skup ćelija od čijih stanja zavisi i iduće stanje izabrane ćelije. Susjedstvo se bira na identičan način za svaku ćeliju. Nakon toga koristeći pravilo ϕ na osnovu stanja ćelije i stanja njenih susjeda dodjeljujemo joj stanje u idućoj diskretnoj vremenskoj instanci. Stanja ćelija kako je već navedeno pripadaju skupu od dva moguća stanja: $\{0, 1\}$.

Sada je potrebno razmotriti na koji način birati susjedstva, te na osnovu toga definisati moguća pravila koja mogu biti primjenjena na evoluciju sistema. Generalno, moguće je koristiti bilo koje pravilo za izbor susjedstva. Međutim, u praksi izučavanja pokazalo se da je najjednostavnije pravilo koje uzima samo simetrično susjedstvo od $2n$ najbližih ćelija sasvim dovoljno da se pokažu i najkompleksnije osobine.

Prodiskutujmo sada broj mogućih pravila za svako izabrano susjedstvo. Cjelokupno susjedstvo će imati $2\mathrm{n}+1$ ćeliju uključujući i ćeliju u razmatranju. Za svaku od kombinacija stanja, tačnije 2^{2n+1} mogućih s obzirom na 2 moguća stanja, potrebno je definisati sljedeće stanje u koje se prelazi ukoliko se naiđe na tu situaciju. Svako od idućih stanja također pripada skupu $\{0,\ 1\}$ tako da je ukupan broj pravila $2^{2^{2n+1}}$ za ovako izabrano susjedstvo.

Ispostavilo se da i ovdje najjednostavnija pravila daju najzanimljivije rezultate, što smo vidjeli u već dosta primjera u ovoj oblasti, te za najjednostavnije susjedstvo sa $\mathrm{n}=1$ daje poprilično zanimljivu kompleksnost i moguća razmatranja. Najveći broj radova iz oblasti je pisan upravo za ovaj slučaj (pogledati npr. većinu radova Stephena Wolframa uključujući [2], [3], [4] i mnoge druge).



Slika 12. *Grafički izbor pravila elementarnog ćelijskog automata. [generisano softverom]*

Sada, prema gore dobijenoj formuli imamo da je ukupan broj mogućih pravila $2^{2^3}=2^8=256$. Ovo može grafički biti prikazano kao što je dato na *slici 12* gdje imamo sljedeće stanje za svaku kombinaciju susjedstva.

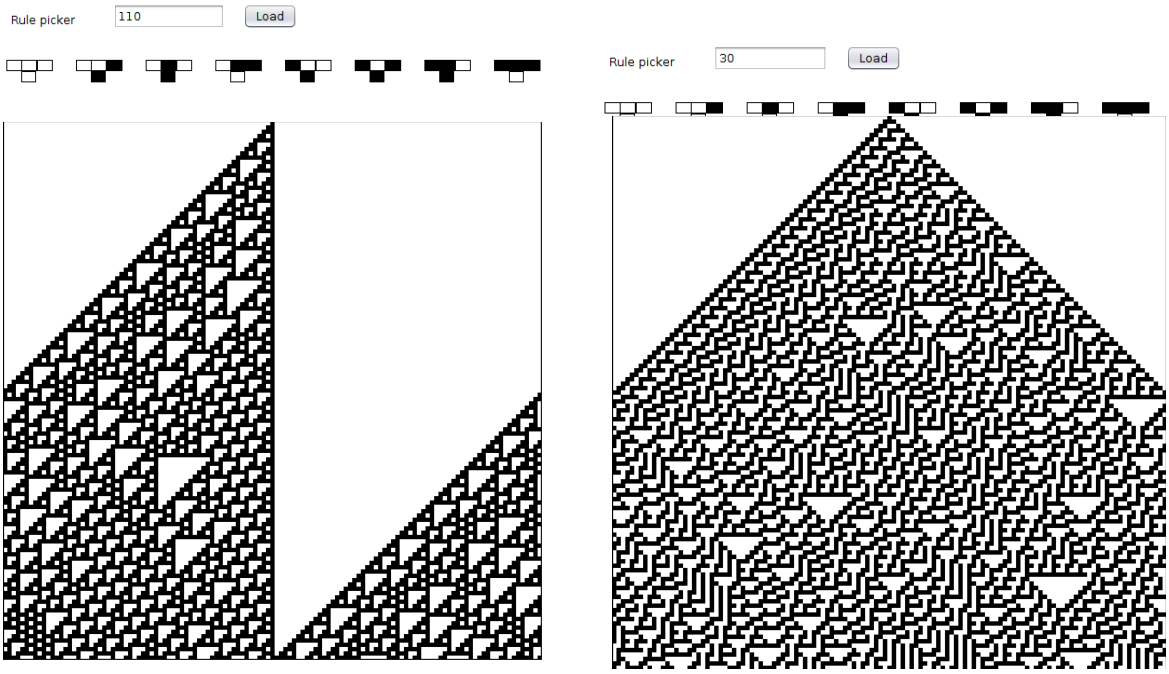
Primijetimo i da proširenjem susjedstva na $\mathrm{n}=2$ broj pravila raste na $2^{2^5}=2^{32}=4294967296$, tako da nešto kompleksniji sistemi imaju eksponencijalno veći broj pravila. I ovo je također jedan od razloga izučavanja najjednostavnijeg slučaja zbog mogućnosti pregleda svih pravila, što bi za slučaj $\mathrm{n}=2$ bilo teze.

U daljem tekstu sva razmatranja odnosit će se pretežno na slučaj susjedstva $\mathrm{n}=1$ binarnih jednodimenzionalnih ćelijskih automata.

Način na koji definišemo koje pravilo se koristi pri evoluciji ovih sistema, a predložio ga je Stephen Wolfram na početku izučavanja ove oblasti, temelji se na enumeraciji pravila na osnovu bita u koje prelaze leksikografski poredane kombinacije susdjedstva. To znači da su susjedstva poredana kao na *slici 12*, dok redoslijed on/off stanja u koji ona prelaze predstavlja binarno zakodiran broj pravila. Tako na primjer pravilo na *slici 12* je predstavljeno brojem $01101110_2=110_{10}$, gdje se nule i jediniće redom uzimaju kao off ili on stanja respektivno.

Ostaje još i problem rubnih ćelija. Iako je u definiciji ćelijskog automata potencijalno beskonačna rešetka, u realnim primjenama koristi se konačan broj ćelija u automatu, pa za jednodimenzionalni binarni slučaj koji ispitujemo postoje ćelije koje se nalaze na krajevima niza i koje nemaju lijeve i desne susjede. Tako da je potrebno na neki način definisati susjedstvo i za njih.

Za ovaj problem koristi se nekoliko mogućih rješenja u zavisnosti od primjene i svrhe u koju se koristi ćelijski automat. Moguće je zamisliti da se kraj i početak niza spajaju u torusnu topologiju, tako da je niz neprekidan i prva i posljednja ćelija su susjedne. Ovo je najčešće korišten model u teoretskim izučavanjima. Također moguće su i alternative, kao npr. imati fiksne vrijednosti rubnih ćelija što je korisno u određenim simulacijama protoka toplote i sličnih fizikalnih pojava. Pokazuje se da izbor ovih rubnih uslova nema veći efekat na kvantitativnu i statističku analizu [6].



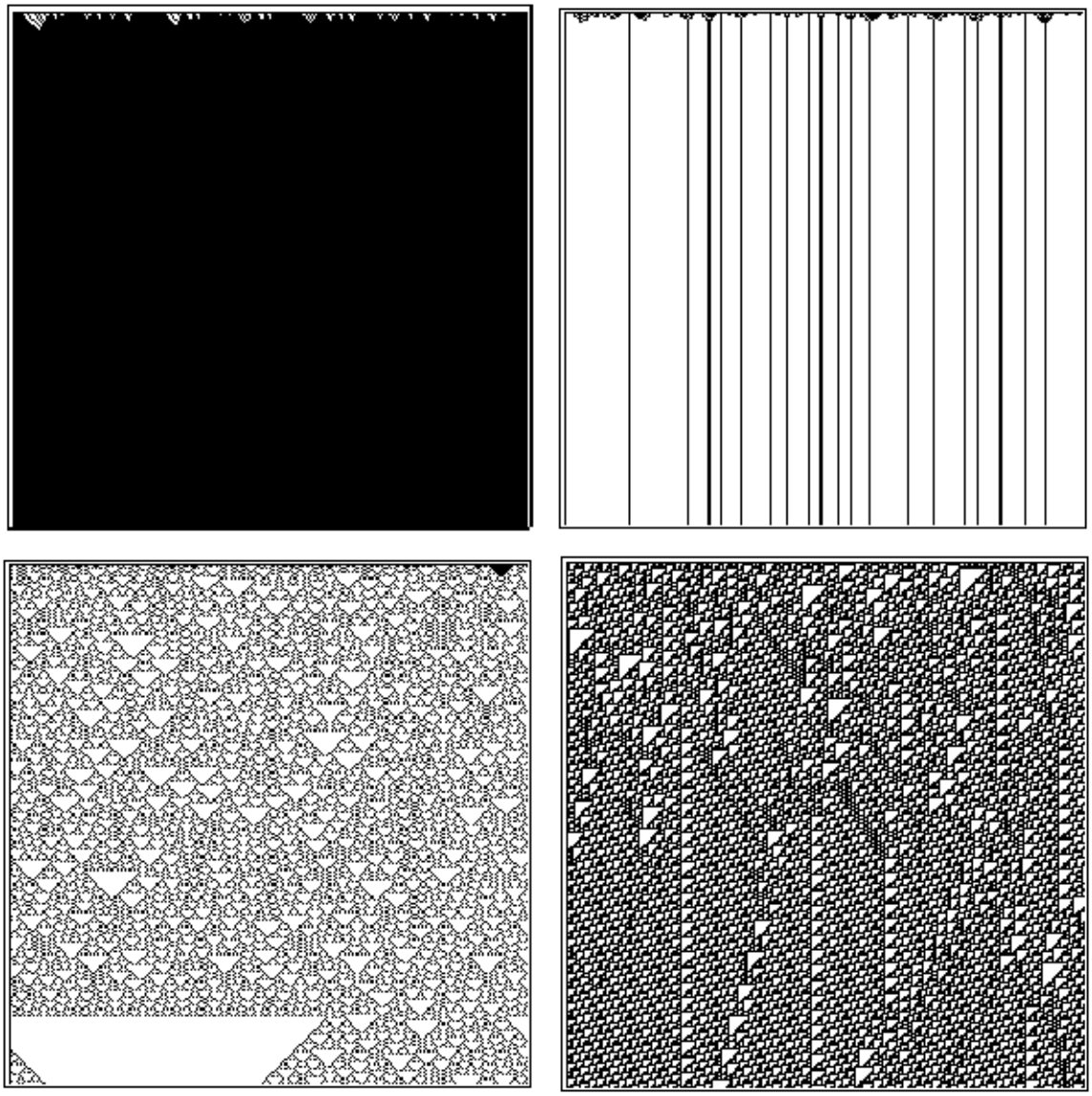
Slika 13. *Prikaz više koraka evolucije pravila 110 i 30. Pravila su grafički prikazana na vrhu slike. Uočljive su kompleksne strukture. [generisano softverom]*

Na *slici 13* prikazana je grafička evolucija od nekoliko stotina koraka pravila 110 i 30 sa torusnom topologijom niza respektivno da bismo počeli uočavati kompleknost koja proizilazi iz nekih od najjednostavniji sistema izračunavanja koji mogu da se smisle. Kasnije ćemo vidjeti da je moguće i formalnim putem pokazati da su ovi sistemi poprilično moćni sa strane mogućnosti izračunavanja generalnih funkcija – tj. simulacije ili ekvivalentnosti sa Turingovom mašinom.

2.2.2 Početna statistička zapažanja i klasifikacija ponašanja pravila ćelijskih automata

Elementarni ćelijski automati predstavljaju poprilično jednostavne sisteme sa ne tako mnogobrojnim skupom pravila za evoluciju istih. Prije smo vidjeli da taj broj iznosi svega 256 što omogućava čak i brute force pretraživanje skupa pravila i ispitivanje

osobina svakog pravila ponaosob. Čak i neka od prvih izučavanja bazirala su se upravo na ovom principu (pogledati [2]). Upravo ova jednostavnost koja ne umanjuje kompleksnost struktura koje se formiraju omogućava detaljno teoretsko izučavanje elementarnih ćelijskih automata za razliku od nekih drugih sistema sa podjednako kompleksnim formacijama.



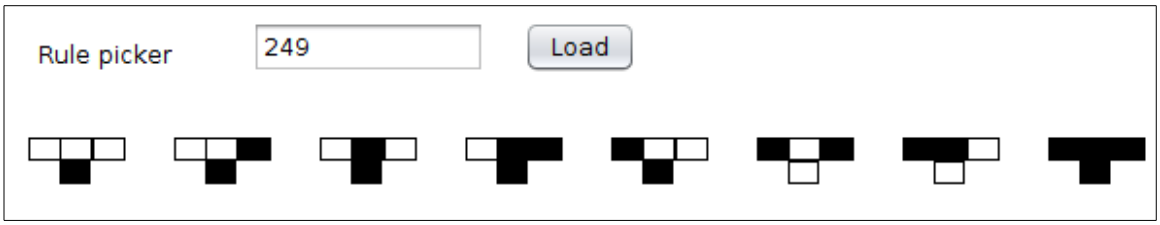
Slika 14. *Evolucija pravila 249, 164, 146, 110 respektivno koja se nalaze u postuliranim klasama I, II, III i IV. [generisano softverom]*

Ukoliko se izvrse simulacije sa slučajnim početnim uslovima svih mogućih pravila elementarnih ćelijskih automata (pogledati [2] za listu grafičkih navedenih grafičkih simulacija), moguće je uočiti određene uzorke u ponašanju istih. Tako naizgled čisto empirijskim ispitivanjem moguće je uočiti grube osobine nekih od pravila, što se kasnije

može pokušati pretočiti u formalnu klasifikaciju.

Klasifikacija sama po sebi je bitna jer predstavlja prvi korak u izučavanju neke vrste sistema. Ukoliko sve instance sistema posmatramo kao odvojene bez nekog načina na koji bismo svrstali svaku od njih u određenu klasu ponašanja, zapravo se radi o ad-hoc izučavanjima specijalnih slučajeva. Sama klasifikacija predstavlja viši nivo za posmatranje sistema koji se nalaze u njoj, te kroz klasifikaciju možemo da uočimo i neke ključne osobine višeg reda, što nas pomjera od početne tačke gdje smo samo definisali sistem i način njegovog funkcionisanja/evolucije.

Ukoliko pogledamo *sliku 14*, prikazana je vremenska evolucija četiri karakteristična pravila koja ujedno predstavljaju i osnovne uočljive klase ponašanja elementarnih automata koje ćemo sada navesti i nešto kasnije i obraditi [1]. Formalna obrada tematike ćelijskih automata predstavlja nešto teži zadatak s obzirom da se radi o nelinearnim sistemima za koje se većinom ne može naći eksplicitan matematički model predviđanja budućih stanja u zavisnosti od sadašnjeg. S obzirom na ovo u dosta slučajeva moguće je izvršiti jedino statistička i slična globalna ispitivanja bez dolaska na konkretan model predviđanja (pogledati npr. [6], [9] kao i većinu radova iz oblasti ćelijskih automata i dinamike kompleksnih nelinearnih sistema).



Slika 15. *Grafički prikaz prelaza stanja pravila 249. [generisano softverom]*

Prva (empirijski!) uočljiva klasa – Klasa I automata predstavljena je pravilima koja konvergiraju u homogene strukture bez obzira na početno stanje sistema. Također, sve informacije kao i slučajnost (eng. randomness) u početnim uslovima gube se u narednim koracima. Tako da ova klasa automata ne predstavlja nikakvu korist sa strane izračunavanja s obzirom da bez obzira na ulaz koji bismo zakodirali kao početno stanje automata, uvijek ćemo dobiti isti izlaz koji ne sadrži nikakve korisne informacije. Primjer ovakvog pravila je pravilo 249 koje bez obzira na strukturu početnih uslova konvergira u niz ćelija u off stanju (crna boja na grafičkom prikazu). Ovo možemo pripisati činjenici da većina kombinacija susjedstva ovog pravila vrši prelaz u off stanje, što je evidentno sa *slike 15*.

Iduću klasu – Klasu II predstavljaju pravila koja ulaze u cikluse ponavljajućih struktura. Na *slici 14* prikazano je među ostalima i pravilo sa enumeracijom 164 koje konvergira u ponavljajuće strukture bez obzira na početno stanje. Razlika ove klase pravila sa Klasom I je to što strukture nisu uvijek iste i nisu obavezno homogene, međutim postoje ciklusi ponavljanja istih. Treba primijetiti da bilo koja konačna konfiguracija automata također

mora da ispolji ciklično ponašanje s obzirom na ukupan broj mogućih stanja 2^N ukoliko N predstavlja broj ćelija u početnoj konfiguraciji. Iako će se ciklus u najgorem mogućem slučaju javljati nakon upravo 2^N ponavljanja nakon što su prodjena sva moguća stanja, što bi značilo da i za poprilično male konfiguracije imamo ogromne cikluse, empirijski rezultati pokazuju da je taj broj manji. Pretežno je ograničen je sa $2^{N/2}$ dok je u dosta slučajeva čak potrebno ne više od N iteracija da bi se dovršio ciklus [6].

Unutar Klase III nalazimo pravila poput 146 i 30 koja ispoljavaju naizgled nepredvidivo slučajno ponašanje sa ponavljajućim uzorcima pretežno ispoljenim u vidu trougaonih struktura. Ova klasa je poprilično osjetljiva na početne uslove i predstavlja dobar alat za izučavanje slučajnosti i nasumičnosti (eng. randomness) [1]. Kasnije ćemo vidjeti nešto rigoroznije razmatranje kao i implikacije za praktičnu primjenu ovog fenomena (pogledati npr. [7]).

Klasa IV predstavlja najkompleksniju klasu ponašanja ćelijskih automata, ali isto tako i najslabije definisanu. Neformalno, u ovoj klasi automata gotovo svi početni uslovi proizvode kompleksne strukture koje vrše kompleksnu međusobnu interakciju. Postulirano je da ova klasa omogućava skladištenje i prenos informacija, što su osnove za univerzalnu izračunljivost. Pravilo 110 iz ove klase čak je i pokazano kao univerzalno izračunljivo, tj. Turing ekvivalentno (za više informacija i dokaz pogledati [8]). I ovo će biti kasnije nešto detaljnije razmotreno.

Sva gore zapažanja su empirijska i neformalna, ali lahko uočljiva samo pogledom na rezultate simulacije. Bilo je nekoliko pokušaja formalne klasifikacije elementarnih automata (i ostalih), od kojih nijedan nije bio potpuno uspješan u smislu da daje predviđanja ponašanja pravila u zavisnosti od neke njegove osobine, međutim svi oni su dali neke rigorozne rezultate koji mogu biti od koristi. Također kroz proces pokušaja pronalaska formalnog okvira za klasifikaciju došlo se i do mnogo zaključaka o ovoj vrsti sistema koji prije nisu bili poznati.

Obradićemo jedan takav pokušaj s obzirom da je dao poprilično dobre rezultate, te otvorio neka nova pitanja i hipoteze ne samo u oblasti izučavanja ćelijskih automata, već dinamike kompleksnih sistema generalno. Tako ćemo i ovdje navesti neke od tih rezultata kroz pokušaj formalne klasifikacije. Klasifikator koji navodimo naziva se Langtonov parametar [9]. Prema riječima Langtona koji je i kreator ovog načina klasifikacije, *"ovaj parametar je agregatna statistika koje je povezana sa, ali ne i sasvim pouzdana u predvidjanju kompleksnosti ponašanja"* [1].

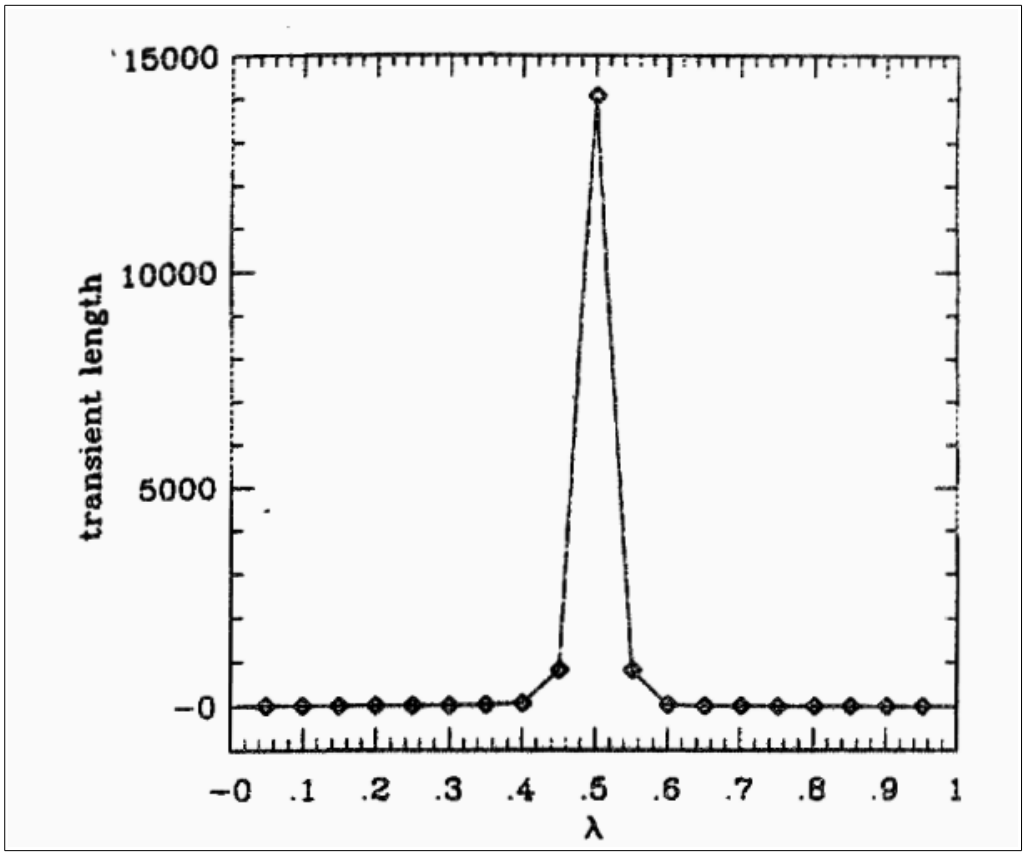
Definicija 2.2.2 Za jednodimenzionalni ćelijski automat sa k stanja i veličinom susjedstva n, Langtonov parametar λ definiše se kao $\lambda = \frac{k^n - \#_q}{k^n}$, gdje je $\#_q$ broj ćelija u funkciji prelaza koje su u unaprijed izabranom specijalnom (eng. quiescent) stanju.

Intuitivno, *definicija 3.6* samo definiše Langtonov parametar kao udio stanja koja nisu off (koje je izabrano kao specijalno za nas slučaj elementarnih automata) u funkciji prelaza

za elementarne automate. Tako na primjer za pravilo 110 čija je funkcija prelaza data na *slici 12* Langtonov parametar iznosi $\lambda = \frac{2^3-3}{2^3} = \frac{8-3}{8} = \frac{5}{8} = 0.625$ s obzirom da su 3 stanja off u prelazu.

Zapažanja koja ćemo navesti Langton u svom radu struktuirano obrađuje za automate sa $\mathrm{n} = 4$ i $\mathrm{k} = 5$ [9] koji predstavljaju znatno kompleksnije sisteme u odnosu na obrađivane elementarne ćelijske automate, međutim dobijeni rezultati govore o generalnim osobinama jednostavnih sistema koji proizvode kompleksne uzorke, te su itekako primjenljivi na razmatranu tematiku.

Ukoliko eksprimentalno diskretno u razmacima od po 0.01 variramo parametar λ te za svaku tu vrijednost konstruišemo niz slučajnih pravila koja zadovoljavaju tu vrijednost parametra, te pustimo simulaciju za iste nad slučajno generisanim početnim uslovima, dobijaju se poprilično zanimljivi rezultati koji govore da postoji određena statistička struktura u ponašanju ovih sistema u zavisnosti od parametra.



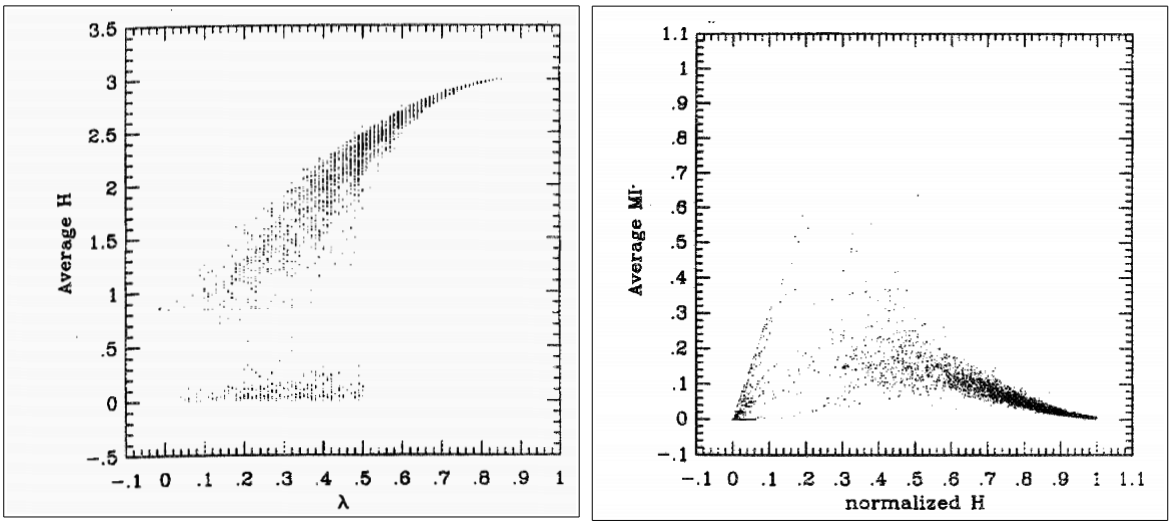
Slika 16. *Prosječno vrijeme stabilizacije automata ponašanja u zavisnosti od Langtonovog parametra. [9]*

Langton postulira da unutar ćelijskih automata postoje klase ponašanja analogne agregatnim stanjima fizičke materije, gdje postoje i prelazne tačke agregatnih stanja koje su od posebne važnosti. Ovo zapažanje se bazira na činjenici da je prelaz stanja u

dinamičkim sistemima (pa tako i agregatnim stanjima materije) ima direktno korelaciju sa vremenom stabilizacije sistema, što predstavlja vrijeme koje je potrebno sistemu da uđe u svoj "prirodni režim rada". Naime, što je sistem bliži prelazu stanja, to je i duže vrijeme njegove stabilizacije.

Ako izuzmemo statistiku iz prethodno opisanog eksperimenta, te ucrtamo na grafik prosječno vrijeme stabilizacije, dobijamo grafik kao na *slici 16.* Vidimo da oko kritične vrijednosti $\lambda = 0.5$ koju ćemo nazvati λ_c vrijeme stabilizacije naglo raste, što bi mogao biti indikator postojanja stanja/režima rada ćelijskih automata, kao i postojanja prelaza stanja za kritičnu vrijednost $\lambda = \lambda_c$.

Poželjno bi bilo sada ispitati nivo kompleksnosti sistema u zavisnosti od λ te vidjeti da li postoji potencijalna poveznica između režima rada automata, te količine informacija koju sistem prenosi, s obzirom da smo vidjeli da za određene sisteme sve početne informacije bivaju izgubljene, dok za druge izgledaju kao potpuno nasumično generisane. Distribucija nivoa prenosa informacija u odnosu na parametar mogla bi dati indikacije i o zavisnosti ove osobine od stanja u kojem se sistem nalazi ukoliko prihvatimo postojanje stanja i njihovih prelaza. Nešto kasnije ovo bi se moglo iskoristiti i za izučavanje izračunljivog potencijala sistema, jer znamo da sistem ukoliko želimo da ga iskoristimo za univerzalni tip izračunavanja mora da ima mogućnost skladištenja i propagacije informacija. Bitno je napomenuti da su ovo sve potencijalne korelacije, ali da ne moraju imati uzročno-posljedičnu vezu. Neki su čak i opovrgnuli ovdje iznijeta zapažanja kao netačna u kasnijim radovima, vidjeti npr [11].



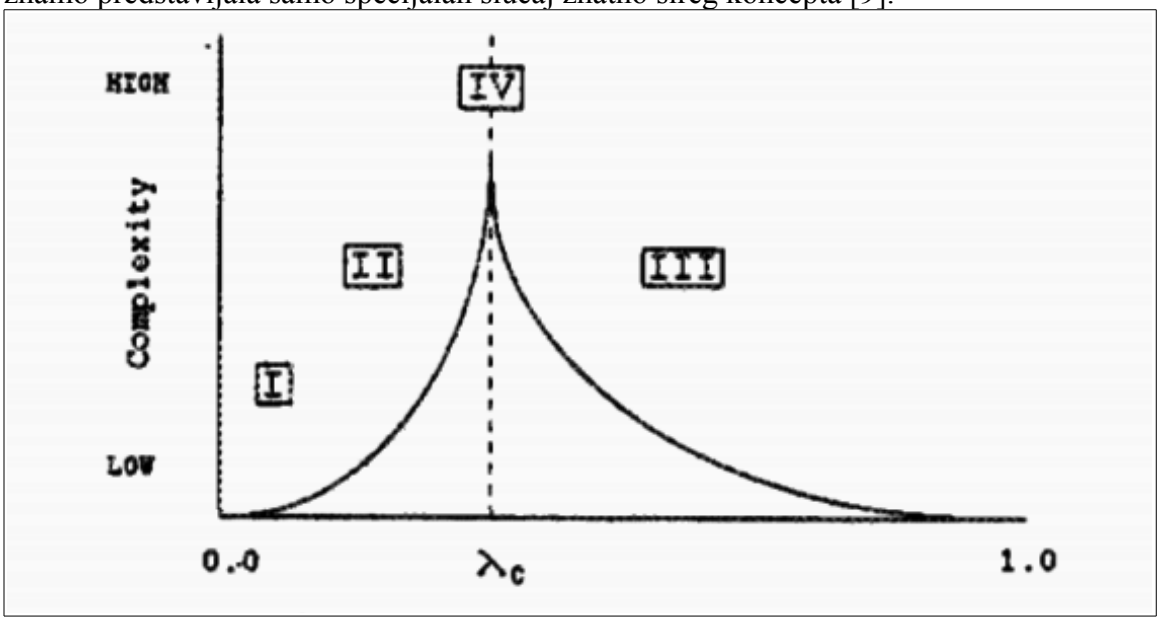
Slika 17. *Prosječna entropija svake ćelije u zavisnosti od parametra (lijevo). Prosječna količina međusobnih informacija ćelija za određenu entropiju (desno). [9]*

Na *slici 17* prikazana je na lijevoj strani prosječna entropija (za definiciju pogledati preliminarne definicije u poglavlju 2.0 ili nešto stručniju literaturu, npr. [10]) svake ćelije u zavisnosti od parametra. Svaka tačka na grafu predstavlja određeno pravilo koje

zadovoljava parametar. Možemo uočiti da sa porastom parametra sve se više smanjuje razlika između minimalne i maksimalne entropije za datu vrijednost parametra. Ključno je i da je za vrijednost parametra 0.5 za koju je postulirano da predstavlja prelaz stanja, entropija varira u pojasu od približno 2.4 - 2.6.

Na desnoj strani na *slici 17* nadalje je prikazan graf koji povezuje prosječnu zajedničku dijeljenu količinu informacija (eng. mututal information) za određenu entropiju. Može se uočiti da postoji maksimalna vrijednost zajedničke količine povezanih informacija koje nose ćelije, te se ta vrijednost upravo poklapa sa vrijednošću entropije u pojasu od približno 2.4 - 2.6, što odgovara upravo postuliranoj vrijednosti parametra $\lambda = 0.5 = \lambda_c$.

Iz ovih vrijednosti moguće je zaključiti (ne sa potpunom sigurnošću) da u kritičnoj vrijednosti parametra gdje se dešava prelaz stanja ujedno se dostiže i maksimum zajedničkih informacija ćelija, što predstavlja osnovu za univerzalnu izračunljivost. Langton ovo naziva "izračunavanje na rubu haosa", te predstavlja koncept da postoji tanka linija između neodređenosti, uređenosti i haosa na kojoj postoje plodni uslovi za univerzalnu izračunljivost. Ovo bi imalo i velike filozofske implikacije na cijele oblasti kompjuterske nauke i dinamike kompleksnih sistema s obzirom da bi kompjutacija kakvu znamo predstavljala samo specijalan slučaj znatno šireg koncepta [9].



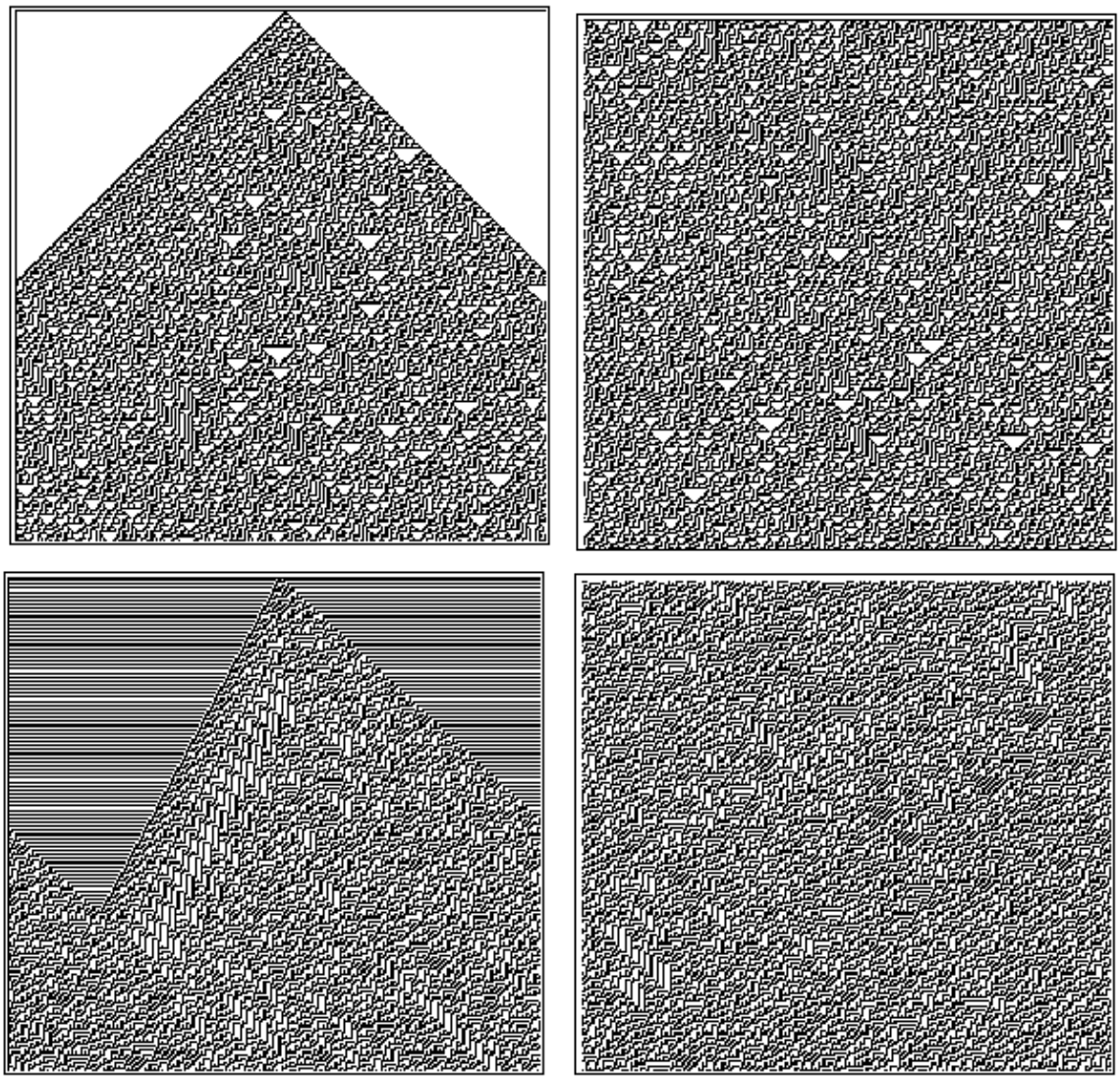
Slika 18. *Wolframove intuitivne klase ponašanja ćelijskih automata u zavisnosti od Langtonovog parametra. [9]*

Ovo ima implikacije i na klasifikaciju ponašanja automata do koje smo željeli doći, jer bi prema tome automati Wolframove klase IV spadali upravo na rub prelaza, tj. u okolini kritične vrijednosti $\lambda = \lambda_c$, dok bi klase I i II bile u vrijednostima manjim od kritične s obzirom na nisku količinu informacije koje prenose, a klasa III bi bila locirana na vrijednostima većima od kritične s obzirom na haotično ponašanje iste. Ovaj raspored klasa prikazan je na *slici 18*.

Kako smo i rekli klasifikacija u većini slučajeva može biti prvi korak u daljnjem izučavanju nekog sistema, pa tako i ova navedena klasifikacija, bilo da se radi o intuitivnoj ili pokušajima formalne, daje naznake o zanimljivim osobinama elementarnih ćelijskih automata koje bi trebalo detaljnije ispitati. Tako klasifikacija predviđa automate koji imaju potpuno nepredvidivo ponašanje (eng. random), kao i automate "na rubu haosa" koji imaju moć univerzalnog izračunavanja. Upravo ove klase koje odgovaraju Wolframovim klasama III i IV kao najzanimljivije i klase sa najvećim potencijalnim primjenama biće izučene u sljedećim razmatranjima.

2.2.3. Nasumičnost (eng. randomness) elementarnih ćelijskih automata

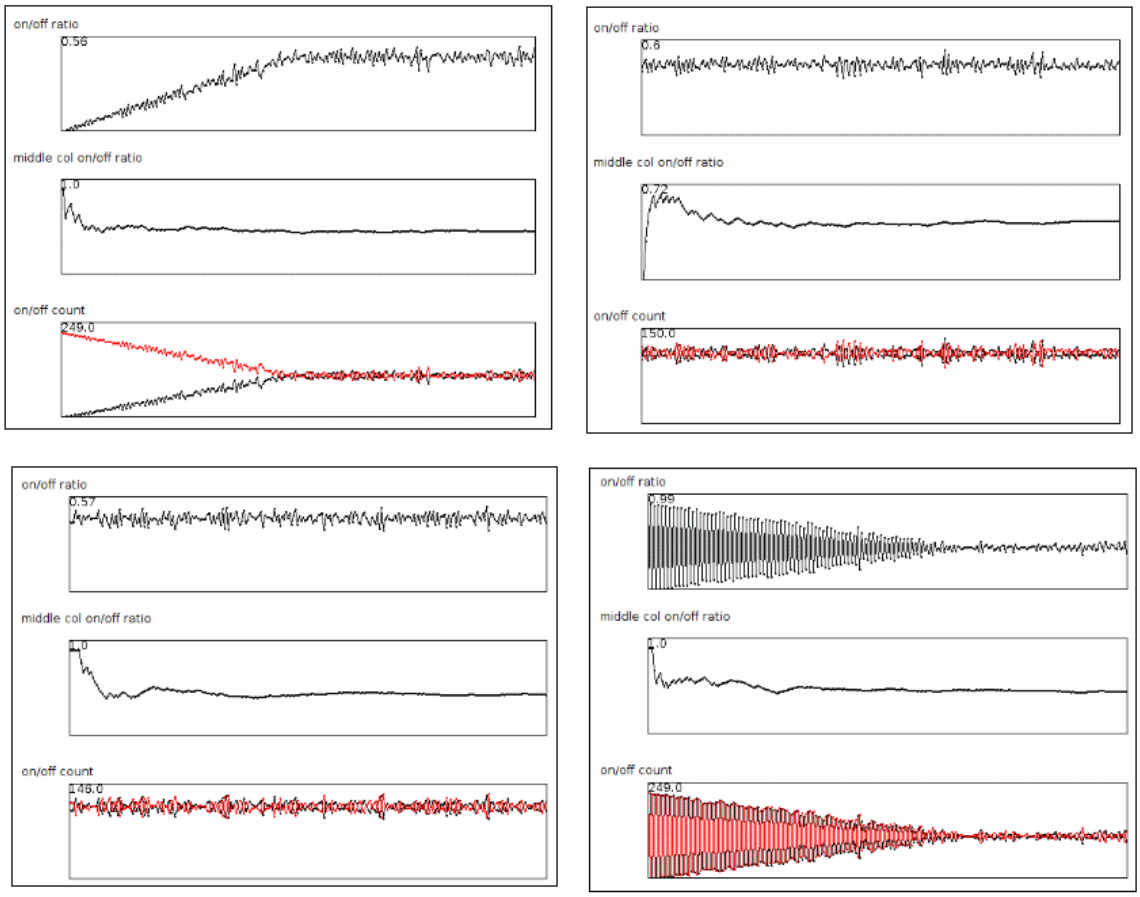
Izučavanje slučajnosti i slučajnih pojava ključno je u raznim dijelovima kompjuterske i ostale nauke s obzirom da neke od ključnih primjena današnjice poput Monte-Carlo algoritamskih metoda i kriptografije su upravo bazirane na ovom konceptu. Odmah se postavlja pitanje da li bi se uočena slučajnost u klasi III elementarnih ćelijskih automata mogla iskoristiti u ove svrhe, s obzirom da bi jednostavnost sistema koji generišu ovu slučajnost mogla biti presudna u izboru istih nad drugim metodama. Iz ovog razloga, korisno je izučiti nešto detaljnije i formalnije količinu slučajnosti koju mogu da proizvedu navedeni sistemi.



Slika 19.a *Naizgled slučajne strukture generisane pravilima 30 (prvi red) i 45 (drugi red). [generisano softverom]*

Ukoliko osmotrimo strukture generisane elementarnim pravilima 30 i 45 (i njihovim simetričnim ekvivalentima) kao što je prikazano na *slici 19*, može se uočiti da nema nekog predvidivog uzorka u ovim strukturama što bi mogao biti dobar indikator njihove potencijalne slučajne prirode [12].

Mnogi su testovi razvijeni da bi se testiralo koliko je zapravo neki sistem statistički slučajan. Treba napomenuti da se koncept slučajnosti razlikuje od statisticke slučajnosti s obzirom da se statistička slučajnost javlja u sistemima koji su u svojoj prirodi deterministički, kao što je primjer i sa trenutno izučavanim elementarnim automatima, međutim ne postoji globalni uzorak ponašanja koji oni zadovoljavaju.



Slika 19.b *Statistika generisana za simulacije prikazane na slici 19. [generisano softverom]*

Najprimitivnije formalno razmatranje koje bi moglo dati informacije o količini slučajnosti sistema je statistika odnosa on i off stanja ćelija u redu, s obzirom da ukoliko je sistem zaista slučajan taj bi odnos trebao da bude blizu $1/2 = 0.5$ s obzirom da sva stanja moraju biti podjednako moguća u potpuno slučajnom sistemu.

Razmotrimo *sliku 20* koja daje po tri statistike za evoluciju pravila 30 i 45 sa uređenim i slučajnim početnim uslovima (pogledati *sliku 19*) respektivno. Prva statistika za svaku konfiguraciju predstavlja odnos on ćelija u sistemu sa ukupnim brojem ćelija (nazvano on/off ratio) koji predstavlja procenat ili udio tih ćelija u svakom od redova. Apscisa predstavlja protok vremena automata, dok je na ordinati unijet ovaj odnos. Druga statistika predstavlja isti udio ali primijenjen ne na pojedinačne redove, već na srednju kolonu jer je predloženo da se upravo ovo koristi kao pseudoslučajni generator u kriptografskim primjenama [12][13]. Treća statistika je poprilično redundatna ako nas zanima samo odnos, te predstavlja stvarni broj on i off ćelija u svakom redu u svakom vremenskom koraku.

Vidimo da se za izvršeni broj koraka u svakom od četiri slučaja simulacije posmatrani odnos kako u svakom redu pojedinačno, tako i u specijalno posmatranoj srednjoj koloni

bliži predvidjenoj vrijednosti 0.5 koja bi bila postignuta za pseudoslučajni generator, tako da ovo može dati jake indikacije o slučajnosti navedenih pravila.

Wolfram je u [12] detaljnije i jačim matematičkim aparatom – informacionom teorijom i naprednom statistikom a ne samo indikacijama iz simulacija formalno obradio slučajnost ovih sistema. Došao je do zaključaka da je pravilo 30 dovoljno slučajno za većinu primjena uz dovoljno veliku početnu konfiguraciju, dok je pravilo 45 znatno manje statistički slučajno, ali opet nepredvidivo u velikoj mjeri. Nešto kasnije u [14] pokazano je da za određene vrijednosti veličine inicijalne konfiguracije kriptoanaliza može da obrne proces pseudoslučajne generacije.

Zaključujemo da je jedna od osobina pojedinih klasa i pravila elementarnih ćelijska automata tako struktuirana da ispoljava statističku slučajnost, što će kasnije biti izučeno s aspekta primjena ove vrste sistema s iskorištavanjem ove specifične osobine.

2.2.4 Univerzalna izračunljivost elementarnih ćelijskih automata

Nakon predstavljanja bilo kojeg sistema koji ispoljava bar neki nivo kompleksnog ponašanja, uvijek je zanimljivo upitati se kolika je zapravo ta kompleksnost te je pokušati na neki način kvantificirati. Ovo znači ispitati da li je sistem sposoban simulirati bilo koji drugi sistem koji se smatra izračunljivim u matematičkom smislu.

Unutar teorije izračunljivosti, kao osnovni model izračunljivog sistema uzima se bilo koji algoritam ili procedura za koji se može konstruisati Turingova mašina koja ga simulira. Ovo direktno slijedi iz rezultata poznatog kao Church-Turingova teza koji govori da je pitanje izračunljivosti ekvivalentno pitanju da li je moguće to ponašanje simulirati na nekoj konstruisanoj Turingovoj mašini. Ovo razmatranje direktno uvodi Turingove mašine kao osnovni model izračunljivih sistema sa kojim se ostali sistemi trebaju porediti ukoliko se želi pokazati njihova računarska moć. Moguće je koristiti i neke druge modele za izučavanje ovog polja poput Alan Churchovog $\lambda - kalkulusa$.

Pitanje univerzalnosti unutar ovih okvira se svodi na mogućnost simulacije Turingovih mašina, tj. sistem se naziva univerzalno kompjutacion ili Turing ekvivalentan ukoliko je u mogućnosti simulirati svaku proizvoljnu Turingovu mašinu, što ima smisla ukoliko se razmotri zašto se Turingova mašina smatra za osnovni model preko kojeg se definiše izračunljivost.

Moguće je sada postaviti ovo pitanje i za elementarne ćelijske automate. Očigledno je i trivijalno pokazati da neka pravila nisu Turing ekvivalentna. Razmotrimo na primjer pravila iz Wolframovih klasa I i II. Ona gotovo pri svakoj inicijalnoj konfiguraciji dovode do homogenih ili ponavljajućih krajnjih stanja sistema, te kao takvi evidentno nisu u mogućnosti mapirati proizvoljan ulaz na proizvoljan izlaz. Pravila iz klase III kako je već pokazano pokazuju poprilično slučajno ponašanje, te kao takva također nisu dobar

kandidat za Turing ekvivalentne sisteme s obzirom da nam za izračunljivost predvidivost igra ključnu ulogu. Ostaje nam klasa IV za koju je i postulirano da predstavlja klasu u kojoj se nalaze pravila dovoljno kompleksna s jedne strane, ali dovoljno strukturirana s druge da bi se mogla iskoristiti u svrhu univerzalne izračunljivosti. Ova pravila također spadaju u tanku liniju koju Langton [9] naziva "rub haosa" na kojoj bi mogle da se dešavaju pojave sposobne za univerzalnu izračunljivost o kojoj smo raspravljali pri problemu klasifikacije elementarnih ćelijskih automata.

Matthew Cook je 2004. godine dao dokaz o univerzalnosti jednog od pravila elementarnih ćelijskih automata, i to pravila 110 za koje je Stephen Wolfram 1985. i postulirao da predstavlja Turing ekvivalentno, tj. univerzalno kompjutaciono pravilo [15].

Da bismo razumjeli okvirno u čemu lezi ključ dokazivanja Turingove kompletnosti sistema elementarnog ćelijskog automata sa tranzicionima pravilom 110, potrebno je prvo navesti neke uvodne elemente.

Turingova kompletnost ima tzv. osobinu tranzitivnosti, tj. ukoliko je neki sistem A takav da je Turing kompletan, a neki drugi sistem B može da ga simulira, tada je i sistem B Turing kompletan. Ovo može biti korisno ukoliko je teško dokazati direktnu mogućnost simulacije Turingove mašine, ali je jednostavnije dokazati mogućnost simulacije nekog drugog sistema za koji je poznato da je Turing kompletan. Ovo je metod koji i Cook koristi u [15], te je izabran ciklični tag sistem za koji je poznato da je Turing kompletan.

Tag sistem generalno predstavlja sistem izračunavanja koji se bazira na iteriranoj modifikaciji početnog stringa. Najbolje je prvo dati primjer i kroz njega pokušati shvatiti koncept, nakon čega će biti data i formalna definicija.

Tag sistem za svaki simbol u alfabetu specifičnog sistema (alfabeti se naravno mogu razlikovati) daje string kojem se taj simbol pridružuje . U svakoj iteraciji, iz string se uklanja simbol sa početka stringa i na osnovu pravila i odgovarajućeg stringa za koje je simbol vezan, originalni string se nadopunjava.

Neka je dat početni string baa unutar alfabeta $\{a, b, c, H\}$, gdje H predstavlja terminalni simbol na koji ukoliko se naiđe obustavlja daljnje izvršenje. Skup pravila je definisan kao $a \to ccbaH, b \to cca, c \to cc$. Tada 6 iteracija ovog sistema izgleda:

Iteracije:
1 baa
2 acca
3 caccbaH
4 ccbaHcc
5 baHcccc
6 Hccccca (halt).

Sistem staje u šestom koraku jer nailazi na halt simbol koji mu to govori.

Nešto modificirana verzija tag sistema koja je originalno korištenja u [15] je ciklični tag sistem, koji je u suštini ekvivalentan sa tag sistemom te postoji poprilično jednostavna pretvorba iz jednog u drugi, s razlikom da ciklični tag sistem ne mora da provjerava kojem simbolu odgovara koji string, već ima skup pravila koja ciklično vrti – zato se i naziva ciklični. Naime, za ciklični tag sistem ne veze se pravilo za svaki simbol, već su pravila u fiksiranom skupu i vrte se ukrug, te se početni znak uvijek mijenja trenutnim pravilom ukoliko znak nije takav da nalaze nemijenjanje, što je objašnjeno u sljedećem primjeru.

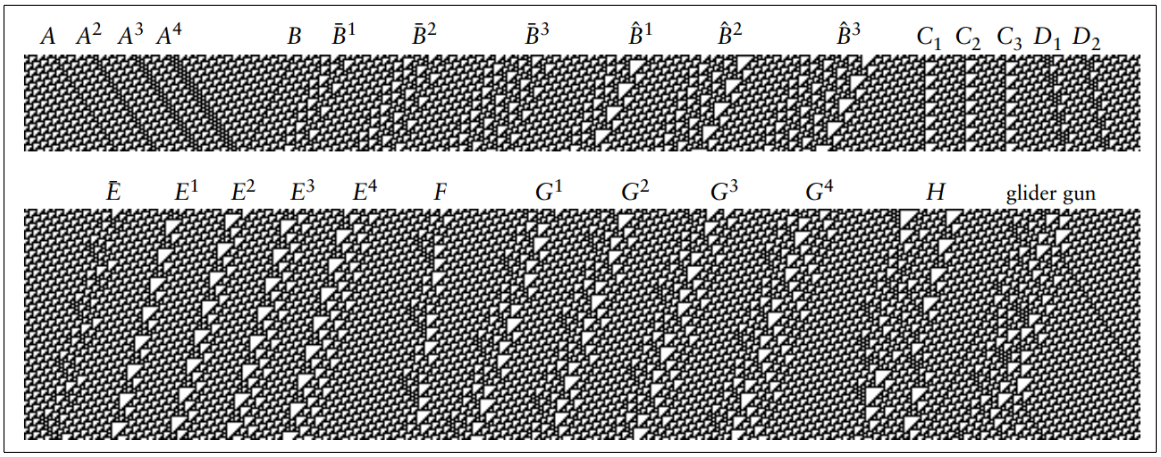
Neka je na primjer dat ciklični tag sistem sa skupom pravila (produkcija) $(010, 000, 1111)$. Alfabet cikličnog tag sistema sastoji se od simbola $\{0, 1\}$, gdje se trenutno aktualno pravilo iz skupa produkcija primjenjuje na string ukoliko je skroz lijevi simbol 1, dok se u suprotnom simbol 0 samo uklanja sa početka stringa. Ukoliko je početni string bio 11001, tada niz iteracija na osnovu ciklično primjenjenih produkcija izgleda:

Iteracije:
trenutno pravilo: 010 trenutni string: 11001
trenutno pravilo: 000 trenutni string: 1001010
trenutno pravilo: 1111 trenutni string: 001010000
trenutno pravilo: 010 trenutni string: 01010000
trenutno pravilo: 000 trenutni string: 1010000
trenutno pravilo: 1111 trenutni string: 010000000
trenutno pravilo: 010 trenutni string: 10000000

Vidimo da se pravila primjenjuju redom u cikličnom redoslijedu, dok se string mijenja samo ukoliko početni simbol nije 0.

Sada ako znamo za osobinu tranzitivnosti Turingove kompletnosti, te činjenicu da je ciklični tag sistem Turing kompletan, tada ukoliko bismo pokazali da je pravilo 110 u mogućnosti na neki način emulirati ciklični tag sistem, tada bi to bilo dovoljno da pokaže da je i samo pravilo 110 univerzalno kompjutaciono.

Na ovom principu je i Mathew Cook dokazao univerzalnost, a ovdje će samo biti izložene osnovne ideje koje se kriju iza dokaza, jer je sam dokaz preobiman te se čitalac za više detalja može referirati na [15].



Slika 20. *Neki od glideri pravila 110. [15]*

Osnovno zapažanje je da pravilo 110, kao i neka druga pravila za koja se postulira univerzalnost, predstavljaju posebnu vrstu sistema u kojima se javljaju "čestice", tačnije podskupovi konfiguracije koje propagiraju vremenom i prostorom u određenim periodima koji se nazivaju glideri. Da bi se bolje shvatilo na šta se misli možemo pogledati *sliku 20* na kojoj su dati osnovni tipovi glidera u pravilu 110.

Pogledajmo na primjer takozvani glider A (Cook je u dokazu dao nomenklaturu gliderima koje koristi što se također može vidjeti na *slici 20*) koji zapravo predstavlja niz konfiguracija $(111, 110, 100)$ koji se javlja periodično, ali sa razlicitim početnim položajem u svakom novom periodu, tako da možemo reci da se glider kreće. Upravo zavisnost vremena i novog početnog polozaja definiše brzinu glidera. Glideri sa nenultom brzinom nazivaju se dinamični. Postoje i stacionarni glideri koji imaju brzinu $\mathrm{v}_g = 0$. Također mogu da glideri vrše međusobnu interakciju, takozvane kolizije, pri čemu se često dešava da kolizija dva glidera proizvodi novi glider sa novim osobinama.

Upravo se na ovom skupu činjenica temelji dokaz o univerzalnost. Tačnije, stacionarni glideri se koriste za čuvanje informacija u sistemu, dok se dinamički glideri koriste da bi kolizijama mogli da modifikuju stacionarne glidera u kojima su informacije sačuvane. Glideri se biraju tako da modifikacije nastale kolizijama odgovaraju produkcijama cikličnog tag sistema, što dokazuje univerzalnost pravila 110.

Praktična primjena i kodiranje realnim problema za izvršavanje pravilom 110 može se naći u [16], gdje se kombinuje pretvaranje Turingove mašine u ciklični tag sistem, nakon čega se on kodira za izvršenje ćelijskim automatom. Iznenađujući je rezultat da ovo računanje i pretvorba zahtijeva asimptotski polinomsko vrijeme, iako se smatralo da je izvršenje barem reda eksponencijalne funkcije. Nije još dovoljno istraženo koliki je potencijal izračunavanja na ovaj način s obzirom na mogućnost fizičke implementacije ćelijskog automata za visoko paralelno izračunavanje, s obzirom da je jedna od osobina ćelijskih automata sinhroni (paralelni) prelaz stanja.

3. Praktične primjene

U dosadašnjem dijelu rada vidjeli smo intuitivno i formalno prikazane osnovne osobine ćelijskih automata generalno, kao i pojedinih specifičnih klasa koje su zanimljive za razmatranje. Sada ćemo pokušati pogledati na koji bi se način mogle iskoristiti te osobine u neke praktične primjene, što bi izvelo oblast iz čisto teoretske kompjuterske nauke u realnu inženjersku aplikaciju.

Biće prvo razmotrene generalne oblasti primjene koje bi bile podržane od strane osobina ćelijskih automata, te će se nakon toga navesti i implementacije ili primjeri primjena nekih klasa specifično. U specifičnim klasama biće obrađeni prvo elementarni ćelijski automati i njihove primjene kao osnovni model izučavan u radu, dok će bit dat i pregled primjena ostalih kategorija ćelijskih automata. U kategoriji ostalih biće izdvojeni samo najzanimljiviji primjeri s obzirom da je gotovo nemoguće proći kroz sve modele zbog širine oblasti izučavanja.

3.1 Generalne oblasti primjene ćelijskih automata

Pogledajmo prvo neke generalne osobine ćelijskih automata koje bi mogle biti iskorištene za primjenu u određenim oblastima.

Pregledajmo prvo ključne osobine koje predstavljaju potencijalno plodno tlo za realne primjene ćelijskih automata. Za početak, ćelijski automati, bar ono što se klasično smatra njima, baziraju se na paralelnosti i homogenosti, to jest sinhronoj promjeni po identičnim pravilima stanja svih komponenti vođeno globalnim diskretnim satom. Nešto slično vidimo i kod modernih računara gdje se elektronska logika ponaša na sličan način. Nadalje, ćelijski automati bazirani su na principu lokalnosti, što znači da pojedina ćelija ne zna ništa o cjelokupnom stanju sistema, već samo stanja svojih najbližih susjeda, što omogućava skladištenje jako male količine informacije i jednostavne komponente. Kao što je viđeno, ćelijski automati iako jednostavni predstavljaju univerzalan izračunljiv sistem, tako da jednostavnost ne znači nužno i beskorisnost. Također, kako spadaju u klasu nelinearnih dinamičkih sistema, pojedine instance imaju haotično i nasumično ponašanje, osobina koja također može biti od koristi u pojedinim oblastima.

Iz ovih osobina i njihovih kombinacija možemo zaključiti oblasti u kojima su te osobine korisne, te tu pokušati primijeniti ćelijske automate.

Iz osobina paralelnosti i univerzalnosti izračunavanja, može se pretpostaviti da bi bilo moguće iskoristiti ćelijske automate u svrhu paralelnog računanja koje sve više postaje paradigma na koju se pokušava prebaciti moderno računarstvo. I dan danas vidimo višejezgrene procesore koji se baziraju na ovom principu, dok bi ćelijski automati ovaj koncept odveli u novi ekstrem gdje bi svaka ćelija predstavljala zaseban jednostavni procesor u visokoparalelnom računanju. Također, moguće je i implementirati ćelijskim

automatima neke visoko paralelizabilne zadatke poput obrade digitalnih slika gdje bi se za svaki piksel pojedinačno uspostavljao ćelijski automat koji bi ga mijenjao u zavisnosti od susjednih piksela. Istraživanja u ovoj oblasti već su dala neke rezultate.

Ako pogledamo realne pojave iz raznih oblasti nauke poput sociologije ili fizike niskog nivoa, vidimo da je jedan od osnovnih koncepta koji se pojavljuju lokalnost i lokalna interakcija koja daje globalno prepoznatljivo ponašanje. Upravo ovo čini ćelijske automate pogodnim za razne vrste simulacija, od kojih će neke biti obrađene i u specifičnim implementacijama primjena.

Ako dodamo mogućnost izmjene, heterogenosti i napredovanja pravila prirodnom selekcijom, tada bi ćelijski automati mogli biti iskorišteni za primjene na polju vještačke inteligencije. Na samom začetku oblasti sam Wolfram je izučavao potencijal ćelijskih automata u primjenama simulacija neuronskih mreža koje su veoma koristan i napredan koncept vještačke inteligencije danas široko primijenjen u raznim oblastima.

Također nepredvidivost nekih određenih instanci ćelijskih automata čini ih korisnim u primjenema gdje je potrebno da je teško moguće pogoditi početno stanje nekog sistema na osnovu neke od njegovih kasnijih iteracija. Očita oblast primjene ovog koncepta je kriptografija, gdje je potrebno naći način da se iz početne konfiguracije (takozvani seed) generiše niz naizgled nasumičnih vrijednosti iz čije se statistike ne može predvidjeti sljedeći član tog niza. Ovo je korisno za primjene šifriranja informacija pri njihovom slanju kroz nesiguran komunikacioni kanal.

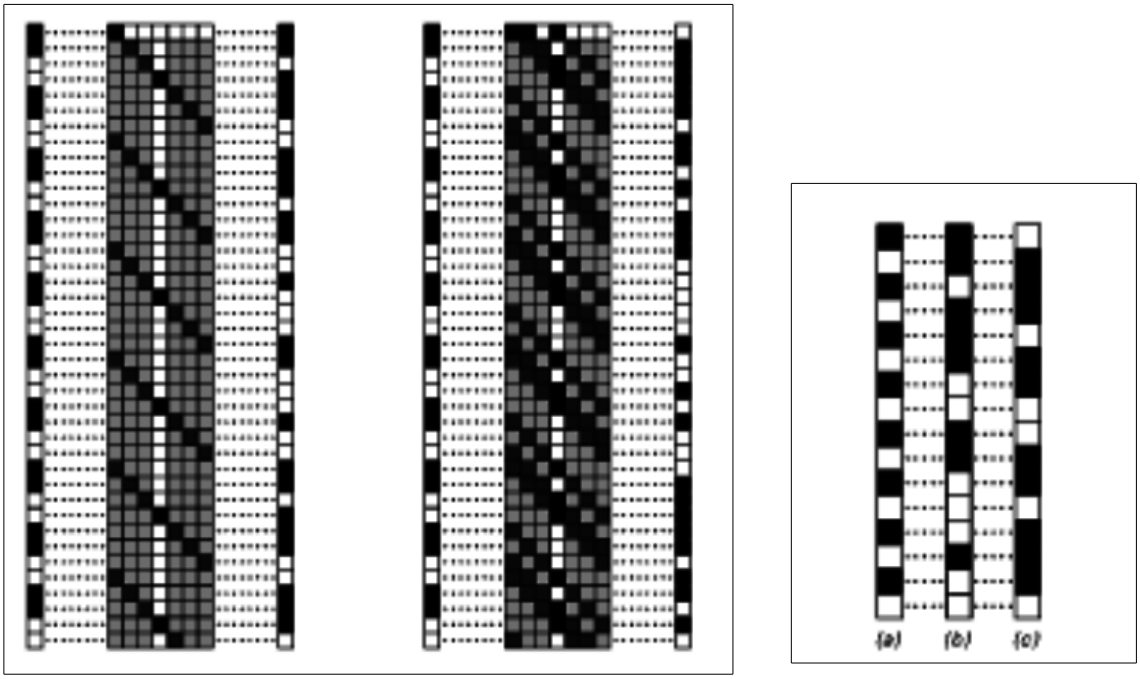
3.2 Primjene elementarnih ćelijskih automata

Čak i najjednostanvniji od sistema ćelijskih automata, elementarni automati sa susjedstvom veličine 3 i 2 moguća stanja imaju potencijalne praktične primjene, kao što ćemo vidjeti u ovoj sekciji. Većinu ovih primjena izučavao je i predložio Stephen Wolfram, međutim dosta njih je kasnije i verifikovano i detaljnije istraženo od strane drugih.

Primjene ove vrste automata direktno slijede iz osobina navedenih u formalnom tretmanu ove vrste sistema u radu. Naime, iz naizgledne (čak i formalno dobro ispitane) nepredvidivosti ponašanja pojedinih elementarnih pravila, slijedi njihova potencijalna primjena u oblastima gdje je ova osobina ključna – kriptografija i sl.

3.2.1. Primjena elementarnog pravila 30 u kriptografiji

Wolfram je 1986. u [13] predložio kriptografski sistem temeljen na elementarnom pravilu 30 ćelijskih automata koji će biti opisan ovdje.



Slika 21.a *Primjer generacije enkripcijske šeme iz početnog seeda (lijevo). Primjer primjene enkripcijske šeme na određen skup bita (desno). [2]*

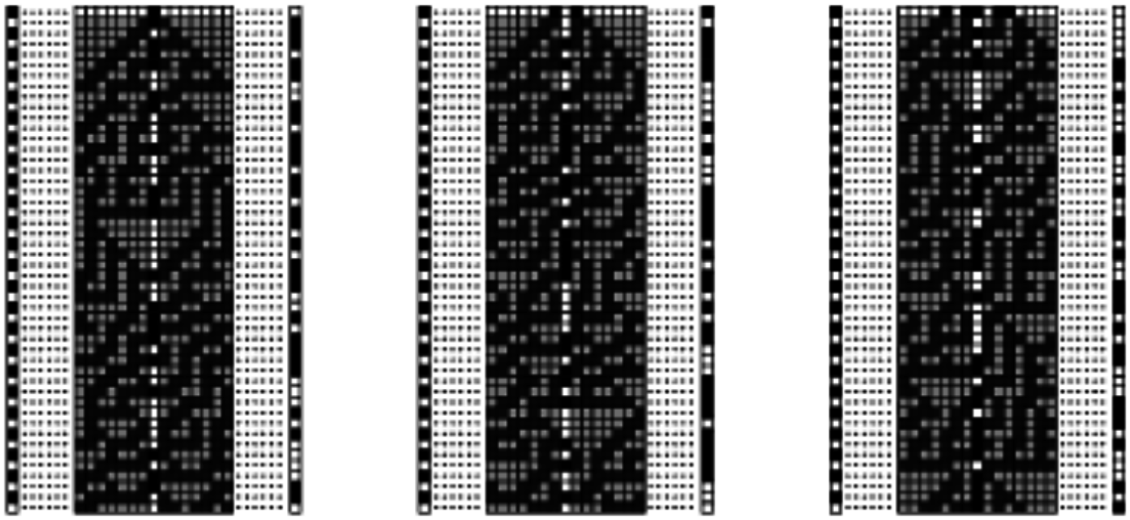
Generalno, sistemi enkripcije mogu se bazirati na primjeni enkripcijske maske na bite koji se žele kodirati. Na primjer, ako pogledamo sliku 21 na desnoj strani, možemo vidjeti da je dat niz bita (a) koji želimo kodirati. Niz bita (b) u srednjoj koloni naziva se enkripcijska šema. Niz bita (c) koji je zapravo kodirani niz (a) nastaje tako što primijenimo logičku operaciju XOR na svaki bit iz originalnog niza sa odgovarajućim bitom iz enkripcijske šeme.

Ovaj sistem enkripcije može davati dobre rezultate ukoliko je enkripcijska šema dovoljno dobra da se ne može zaključiti metodama kriptoanalize. Kriptoanaliza je proces obrnut od enkripcije gdje se pokušava bez znanja ključa dobiti originalna poruka posjedovanjem samo kodirane.

Osnovni problem koji možemo uočiti je da enkripcijska šema mora biti duga koliko i cijela poruka, što nije uvijek moguće, pogotovo iz razloga što većinom dužina poruke nije unaprijed poznata. Zato se pristupa problemu tako što se uzme neki početni binarni string poznat objema stranama. Nakon toga se nizom koraka na taj binarni string primjenjuje serija transformacija, te se pri svakom koraku uzima vrijednost nekog unaprijed izabranog bita. Na *slici 21.a* na lijevoj strani dat je primjer generacije enkripcijske šeme na ovaj način. Naime, početni binarni string (koji nazivamo seed) 1000000 u slučaju a, ili 1101000 u slučaju b). Transformacija koja se ovdje primjenjuje je binarno barel šiftanje udesno, gdje se pri svakom koraku uzorkuje vrijednost srednjeg bita. Proces transformacije se ponavlja onoliko puta koliko je dužina poruke u bitima, jer ćemo ove

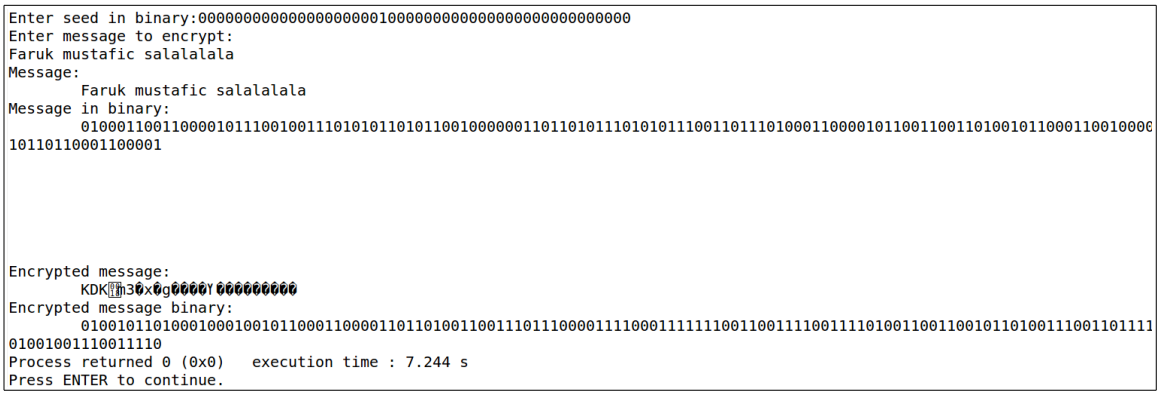
uzorkovane bite koristiti kao enkripcijsku šemu. Sada se na poruku primjenjuje isti postupak kao što je opisano da bi se dobila enkriptovana poruka.

Sada smo riješili problem generisanje enkripcijske šeme, međutim novi problem koji nastaje je jednostavnost inverznog određivanja šeme kriptoanalitičkim metodama. Naime, poznatao je da većina poruka sadrži neku formu pozdrava na početku, ili neke riječi koje se često ponavljaju (npr. 'the' u engleskom jeziku). Koristeći ovu činjenicu jednostavno je otkriti pojedine bite enkripcijske šeme, te ukoliko je postupak generisanja jednostavan, kao što je slučaj u prijašnjem primjeru, tada je moguće rekonstruisati cjelokupnu šemu i samim time otkriti enkriptovanu poruku [2].



Slika 21.b *Primjer generacije enkripcijske šeme iz tri početna seeda sa malim razlikama koristeći elementarno pravilo 30. [2]*

Wolfram u [2] i [13] navodi da je rješenje ovog problema korištenje elementarnog pravila 30 kao transformaciju koju uzastopno primijenjujemo na seed. Postulira da količina izračunavanja potrebna da bi se enkripcija napala prije opisanom metodom eksponencijalno raste sa početnim brojem bita u seedu, tako da i za relativno male veličine seeda problema postaje gotovo nerješiv u realnom vremenu. Također, zbog osobina ćelijskih automata poput lokalnosti i paralelnosti, hardverska implementacija ovog sistema mogla bi iskoristit ovu paralelnost da bi se postigli što bolji rezultati vremena potrebnog za enkripciju [13]. Na *slici 21.b* možemo vidjeti primjer generacije enkripcijske šeme korištenjem elementarnog pravila 30, gdje možemo uočiti da za tri različita seeda koja su poprilično bliska dobijamo potpuno drugačije enkripcijske šeme.



Slika 22. *Primjer enkripcije uz predloženo generisanje sheme korištenjem elementarnog pravila 30. [softver napisan u sklopu rada]*

Na *slici 22* prikazan je praktičan primjer konzolne aplikacije korištenja opisanog postupka enkripcije, gdje vidimo da je početna poruka nakon enkripcije neprepoznatljiva, te da se i na nivou bita poruka poprilično razlikuje od početne.

Najveći problem predložene primjene je njegova simetričnost, tj. potreba da obje strane koje učestvuju u enkripciji poznaju isti seed iz kojeg je generisana enkripcijska šema, što je problem ukoliko te dvije osobe nisu u mogućnosti doći u fizički kontakt radi razmjene seeda. Iz tog razloga se danas pretežno primjenjuju asimetrične enkripcijske šeme poput RSA i sličnih postupaka.

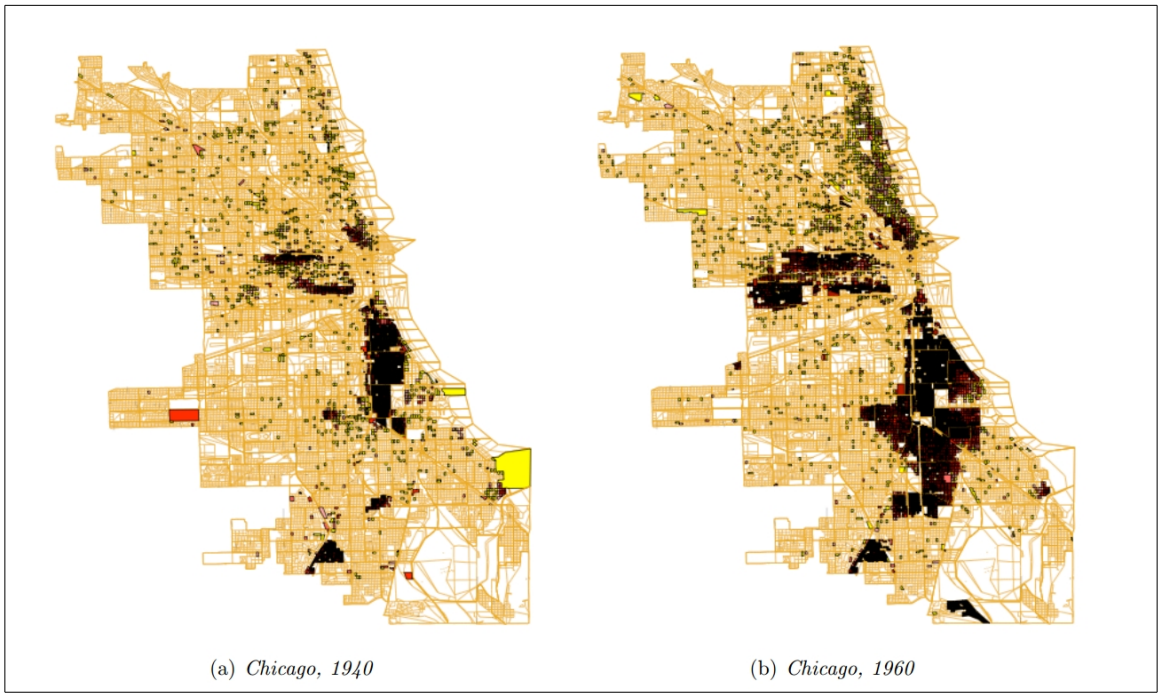
3.3 Pregled primjena ostalih tipova ćelijskih automata

Pored elementarnih automata koji su primarna vrsta koja se izučava u radu, ovdje ćemo dati osnovni pregled primjena i drugih vrsta ćelijskih automata. Iako se pokazalo da su elementarni ćelijski automati, bar pojedina pravila, univerzalno izračunljivi, tj. mogu simulirati svaki drugi izračunljiv sistem, u praksi se pokazuje da ukoliko sistem ima više od jedne dimenzije, jako je teško naći kodiranje koje će taj sistem prevesti u sistem elementarnih automata. Zato posebno u polju simulacija generalno, te simulacija fizikalnih pojava specifično, dvodimenzionalni i ćelijski automati visih dimenzija igraju ključnu ulogu s obzirom da se dosta posmatranih sistema bazira na lokalnoj interakciji – što je jedna od ključnih osobina ćelijskih automata.

Ovdje će biti navedeni primjeri primjena na simulacije izabranih sociooloških i fizikalnih modela.

3.3.1 Schellingov model segregacije

Prvi primjer simulacije uz korištenje dvodimenzionalnih automata je iz polja sociooloških nauka, gdje se izučava razlog iza nastajanja segregacionog naseljavanja. Tačnije, izučava se zašto se u velikim gradovima pojavljuje tendencija razdvajanja susjedstva na rasnoj osnovi, gdje su susjedstva poprilično rasno homogena. Primjer za grad Chicago možemo vidjeti na *slici 23*.



Slika 23. *Primjer rasne segregacije susjedstva u Chicagu. Oblasti u smeđoj i crnoj boji predstavljaju dijelove gdje je crna populacija iznad 75%. [17]*

Schelling je u [18] pokušao da prikaze da je na osnovu čisto lokalnih preferenci o susjedstvu moguće da nastane situacija prikazana na *slici 23* i prisutna u mnogim drugim gradovima sa nehomogenom rasnom populacijom. Model se bazira na tome da će određena osoba promijeniti svoje mjesto stanovanja ukoliko susjedstvo ne zadovoljava da je u njemu prisutno barem t osoba iste populacije, gdje t predstavlja takozvani treshold sistema.

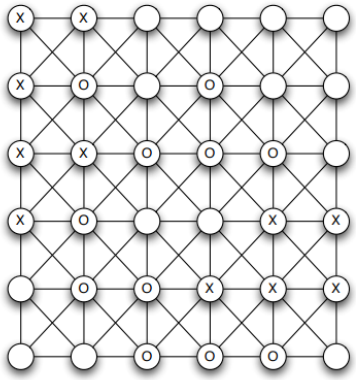
Očito je na prvi pogled da je ovakav model pogodan za implementaciju ćelijskim automatima. Kako i sam model ima dvije dimenzije, koje predstavljaju lokaciju, tako su i dvodimenzionalni ćelijski automati najprikladniji za ovo.

Ostaje još problem preseljenja osobe u modelu, to jeste gdje premjestiti osobu nakon što odluči da nije zadovoljna svojim susjedstvom. Razni modeli na različite načine rješavaju ovaj problem, a jedan od najčešćih pristupa je preseliti osobu na novu slučajno izabranu lokaciju gdje će biti zadovoljna novim susjedstvom. Iz razloga mogućnosti selidbe, u

modelu mora postojati i vrsta ćelija koja predstavlja neokupirana područja, dok su osnovna dva tipa crvene i plave ćelije koje predstavljaju grupacije po kojima se vrši segregacija – crna i bijela populacija u primjeru sa *slike 23*.

X	X				
X	O		O		
X	X	O	O	O	
X	O			X	X
	O	O	X	X	X
		O	O	O	

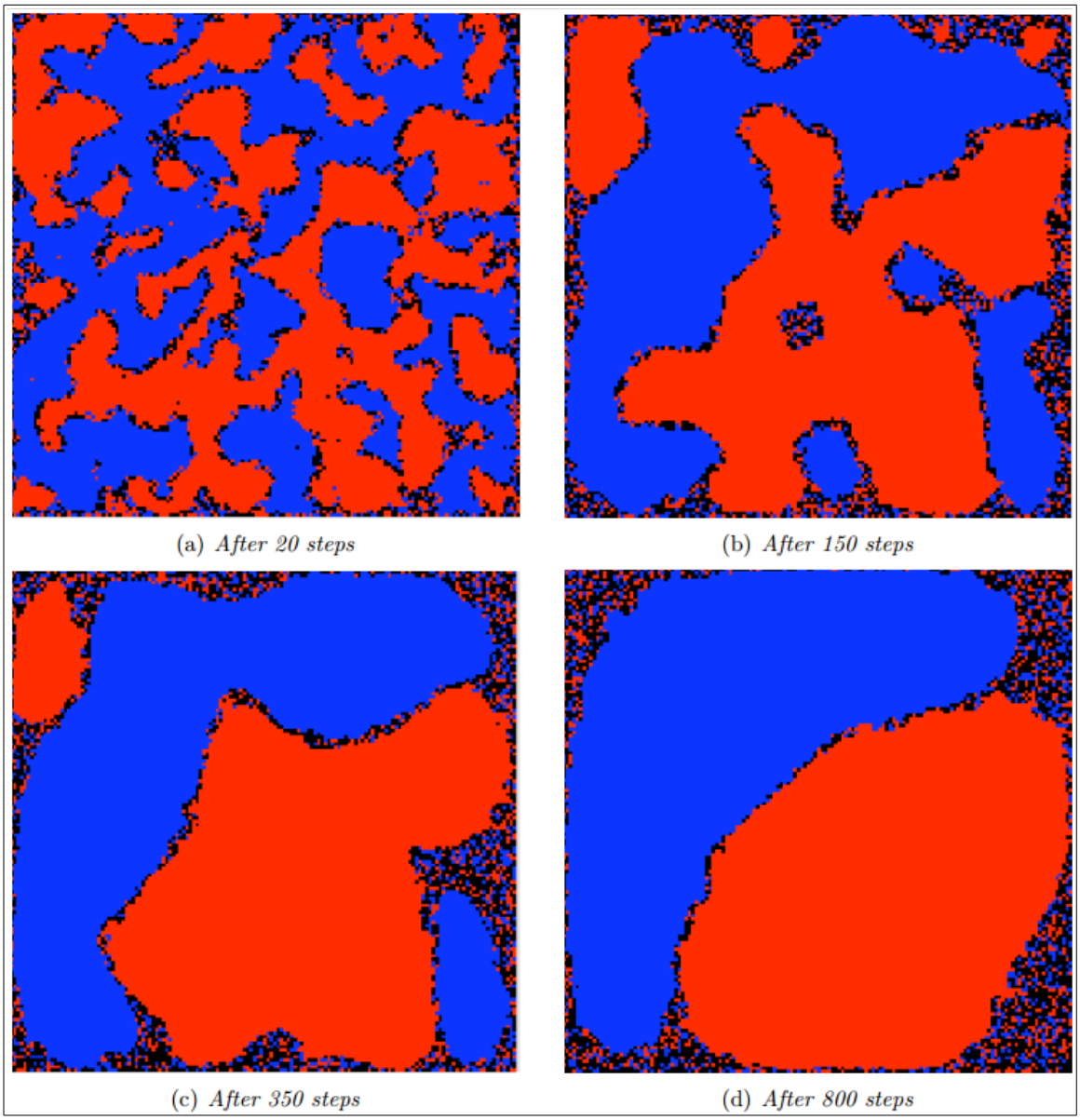
(a) *Agents occupying cells on a grid.*



(b) *Neighbor relations as a graph.*

Slika 24. *Matematički model za izučavanje segregacije (lijevo). Grafovski model susjedstva (desno). [17]*

Na *slici 24* možemo vidjeti ovaj model, gdje ćelije mogu imati vrijednost O ili X koje predstavljaju dvije grupacije, te postoje neokupirane ćelije koje predstavljaju nenaseljene dijelove. Na primjer ćelija u skroz gornjem lijevom uglu modela nije zadovoljna svojim susjedstvom za treshold vrijednost $\mathrm{t} = 3$ s obzirom da ima 2 susjeda istog tipa, te će morati da se preseli u sljedećoj iteraciji. Na desnoj strani *slike 24* dat je model susjedstva predstavljen grafom, gdje se susjedstvom ćelije smatra svaka ćelija koja ima zajedničku ivicu sa njom. Također, i ovaj model se može proširiti slično kao i za elementarne automate tako da predstavlja torusnu topologiju, međutim s obzirom na oblast primjene to ovdje nije poželjno jer gradovi sami po sebi nisu u torusnoj topologiji.

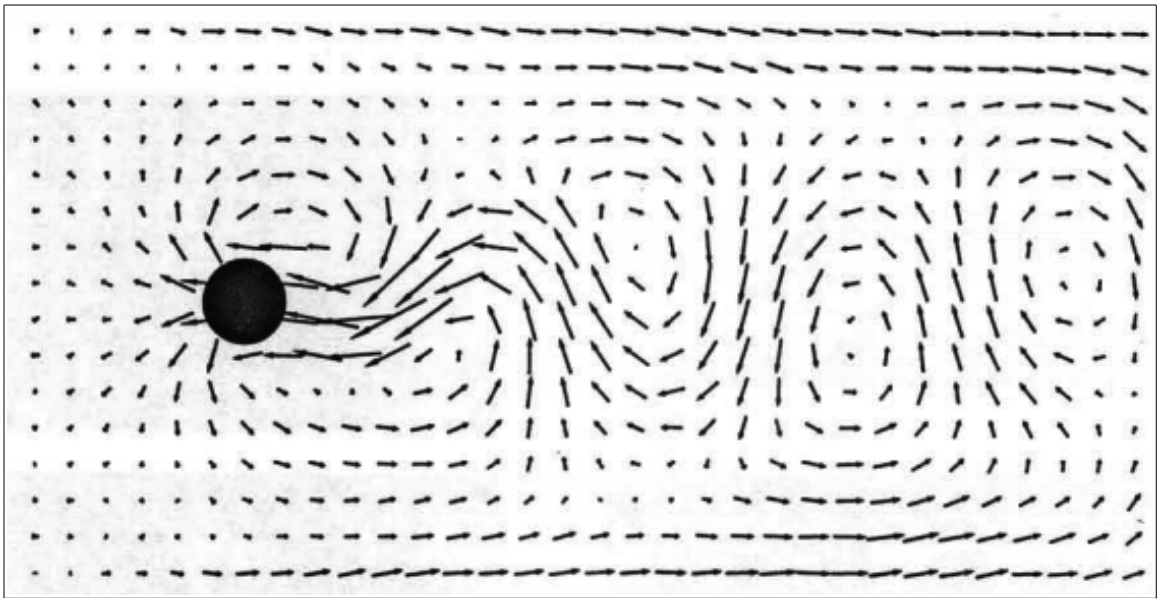


Slika 25. *Rezultat simulacije slučajnih početnih uslova prema modelu, te razni koraci unutar simulacije. [17]*

Ukoliko pogledamo rezultat simulacije uz korištenje ovog modela na *slici 25*, vidimo da iz slučajnih početnih uslova nakon određenog broja iteracija počinju da se stvaraju grupacije te se ukupno populacija sve više dijeli na osnovu boje koja kodira pripadnost određenoj grupi. Ovim je pokazano da čisto lokalne preference mogu dovesti do globalne segregacije, iako nikome od lokalnih učesnika to nije bio krajnji cilj.

3.3.2 Simulacija toka fluida

Predvidjanje i matematički modela toka fluida jedan je od najstarijih i naizučavanijih problema elementarne fizike. Većina današnjih modela bazira se na statističkim zapažanjima i Navier-Stokesovima nelinearnim diferencijalnim jednačinama. Međutim, i dan danas neki od modela ne uspijevaju da sa zadovoljavajućom preciznošću predvide rezultate koji će nastati iz početnih uslova.



Slika 26. *Rezultat simulacije Wolframovog modela toka fluida ćelijskim automatima. [5]*

Wolfram je u [5] predložio korištenje ćelijskih automata u ove svrhe, te opisao rezultate njegovih simulacija na tadašnjem state-of-the-art multiprocesorskom sistemu. Na *slici 26* možemo vidjeti rezultat ove simulacije, te je uočljivo da se javljaju strukture koje primijetimo i eksprimentalnim metodama. Npr. vrtlozi sa desne strane crnog objekta koji predstavlja nepomično tijelo.

A. Realizacija simulatora

U ovom poglavlju biće ukratko prezentiran sadržaj, arhitektura i prikaz načina korištenja simulatora napisanog kao praktičan dio ovog rada. Biće pregledani ciljevi softvera, te razlog za pisanje, kao i neke specifičnosti poput arhitekture i nekih izbora koji su pravljeni pri specifičnoj implementaciji logike sistema. Biće prikazan i način korištenja sistema i najbitnije funkcionalnosti kroz grafički interfejs.

A.1 Namjena i razlog za sistem

Pri izučavanju oblasti ćelijskih automata, a posebno elementarnih koji su najvećim dijelom obrađivana instanca ovih sistema u literaturi i radovima, poseban je naglasak na jednostavnost sistema koji proizvodi kompleksne krajnje rezultate, kako je navedeno u nekoliko navrata u ovom radu. Tako da prilikom implementacije softvera za ovu oblast, nije naglasak na visoko optimizovanom izračunavanju, već je većina potreba javlja se prilikom simuliranja sistema iz datog početnog stanja prema određenim pravilima evolucije, jer su sistemi u razmatranju već sami po sebi poprilično jednostavni i samim time optimizovani jer se baziraju na jednostavnom pregledu (eng. lookup) predefinisanih pravila. Iz ovog razloga pisani softver većinom predstavlja grafičke simulacije sistema elementarnih ćelijskih automata.

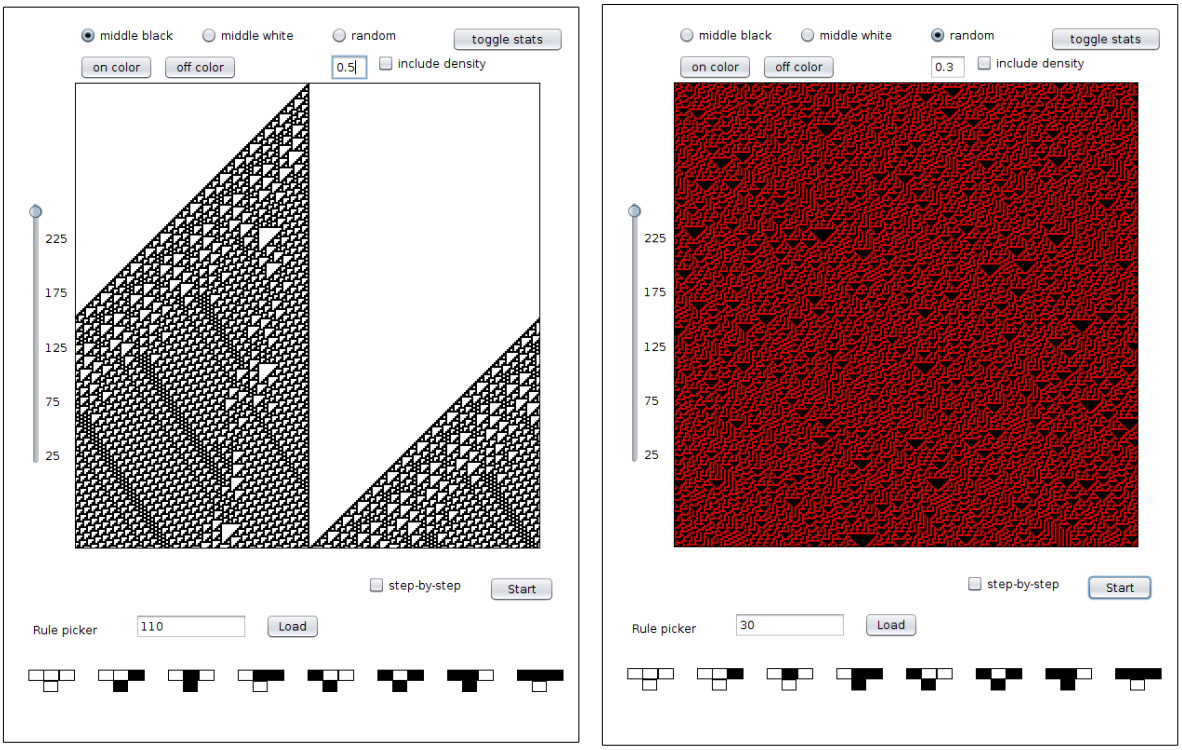
Generalna metodologija izučavanja bilo koje vrste sistema nalaze da se prvo izuče specifične instance da bi se dobila neka osnovna ideja o generalnim osobinama koje se prostiru kroz sve njih, te da se kasnije postulati dobijeni ovim postupkom pokušaju dokazati. Ovo smo mogli vidjeti na primjerima klasifikacije automata, ili uočavanju kompleksnosti glidera pravila 110 pri dokazivanju njegove univerzalnosti. Da bi se omogucilo izučavanje specifičnih instanci prije zaključka o generalizaciji, pisani softver trebao bi da omogući izvršavanje simulacija sa raznim parametrima da bi se mogle uočiti pravilnosti u sistemu kroz te specifične instance. Zato bi trebao da korisniku omogući veliku slobodu izbora, što bi za slučaj simulacije elementarnih ćelijskih automata značilo mogućnost izbora početnog stanja, pravila evolucije grafički i numerički, te broj koraka simulacije koje je potrebno izvršiti.

Nadalje, kako smo vidjeli ćelijski automati teško se obrađuju konrektnim formulama i modelima s obzirom na njihovu nepredvidivu prirodu koja proizilazi iz njihove nelinearnosti, pa je tako statistika jedan od najmoćnijih alata koji može da pomogne. Tako da bi softver koji bi vršio simulaciju trebao da sadrži i neki vid statistike o izvršenoj simulaciji da bi se mogli donijeti neki zaključci.

Također, ovako opisan softver imao bi i edukacionu vrijednost gdje bi se mogle na praktičnim primjerima kroz grafičko okruženje pokazati osobine ćelijskih automata koje bi možda zaintrigirale studente za daljnja izučavanja ovakvih sistema.

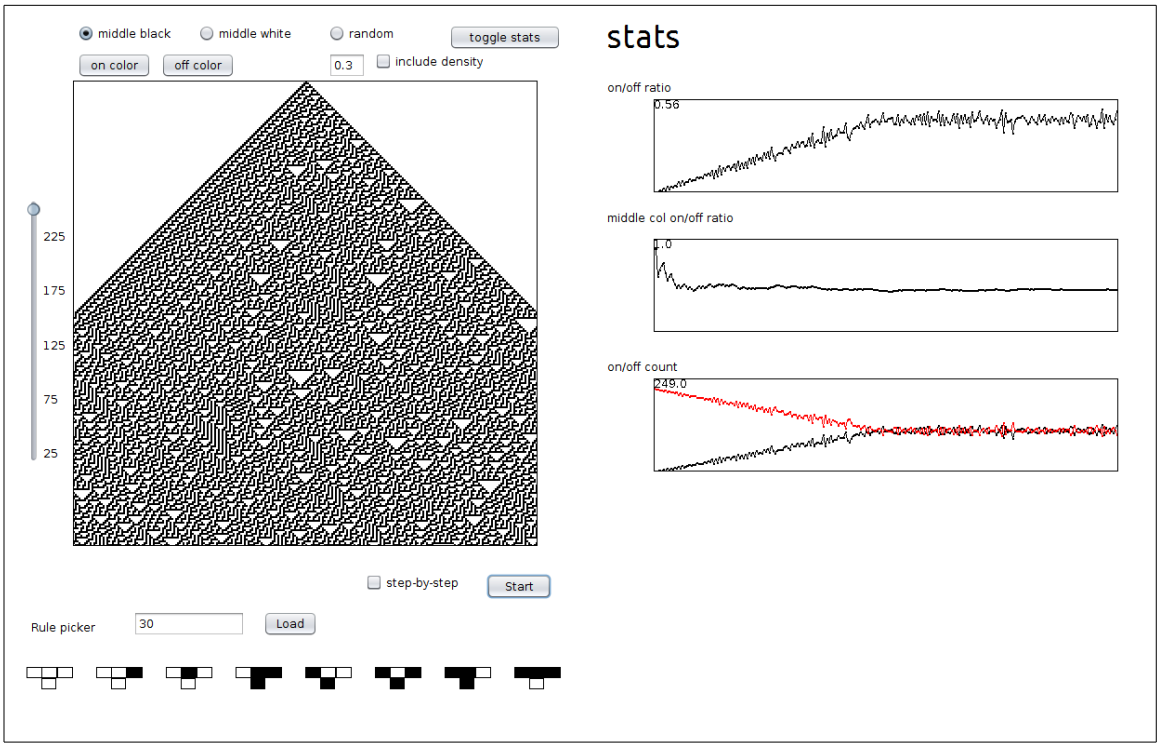
A.2 Prikaz softvera i ključne funkcionalnosti

Pokušat ćemo kroz grafički interfejs softvera proći kroz njegove osnovne elemente i slučajeve upotrebe.



Slika A1. *Osnovni grafički interfejs softvera pisanog u sklopu rada.*

Ako pogledamo *sliku A1*, vidimo osnovni prozor grafičkog interfejsa softvera, te je moguće uočiti elemente prozora za prikaz simulacije, prostora za izbor pravila na numerički način čistim ukucavanjem njegovog Wolfram koda, ili izborom prelaza stanja za svaku od mogućih kombinacija susjedstva grafički. Također u gornjem dijelu prozora moguće je izabrati način formiranja početne konfiguracije gdje je moguće izabrati srednju on ili off ćeliju, te nasumičnu početnu konfiguraciju koja zadovoljava određenu distribuciju. Također postoje i kontrole koje omogućavaju mijenjanje boja on i off stanja radi boljeg grafičkog prikaza u nekim slučajevima. Vidimo da korisnik ima veliku slobodu izbora parametara simulacije. Na lijevoj strani *slike A1* prikazan je softver sa defaultnim početnim parametrima, dok na desnoj strani možemo vidjeti kako izgleda simulacija sa izmijenjenim parametrima boje pravila i početne konfiguracije.

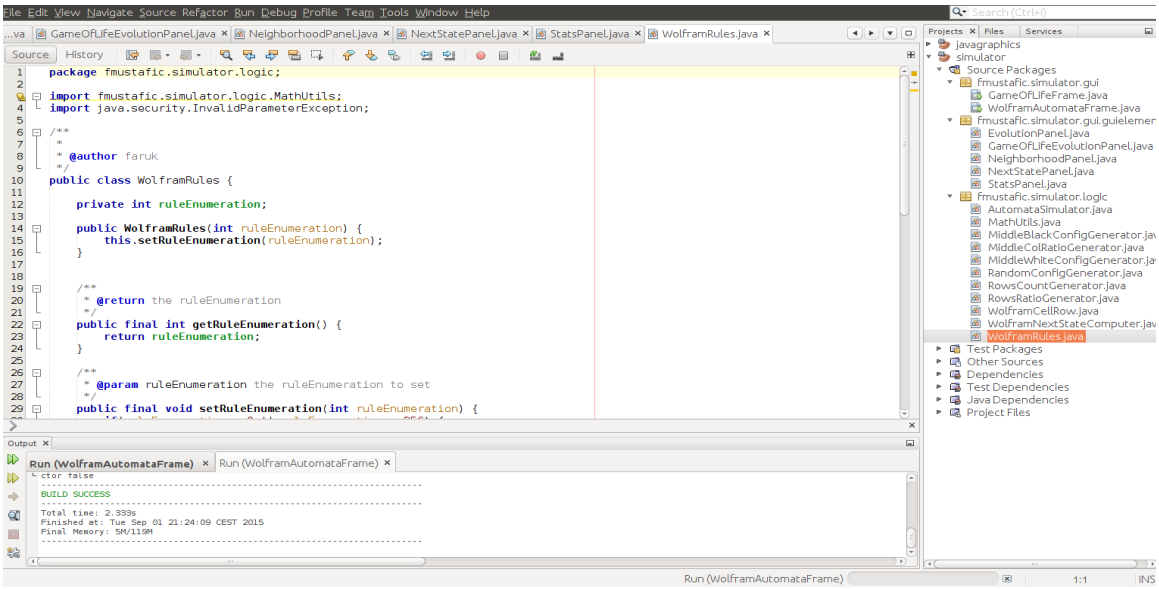


Slika A2. *Prošireni grafički interfejs softvera pisanog u sklopu rada sa prikazanim dijelom za generisanje statistike.*

Na *slici A2* prikazan je grafički interfejs softvera sa proširenim prozorom za statistiku koji se otvara/zatvara na odgovarajuće toggle dugme. Na istoj slici također možemo vidjeti i primjer praktičnog korištenja softvera, gdje statistika za Wolframovo pravilo 30 daje naznake da je srednja kolona nasumična s obzirom da broj on i off ćelija u njoj konvergira u odnos 0.5 kako sistem dalje evoluira, što je raspravljeno detaljnije u poglavlju 2 gdje se formalno tretira nasumičnost elementarnih ćelijskih automata.

A.3 Tehnološke implementacione odluke

Sam softver pisan je u Java programskom jeziku, Netbeans ID okruženju, te koristeći standardne swing GUI elemente koji su dio programskog jezika.



Slika A3. *Prikaz razvojnog okruženja u kojem je vršen razvoj softvera.*

Izbor ove kombinacije tehnologija opravdava se sa nekoliko faktora. Programski jezik Java izabran je iz nekoliko razloga. Kao prvo, smatrano je da open-source kao koncept omogućava ubrzan razvoj softvera, te tako Java kao programski jezik koji je poznat kao jedan od ključnih u ovoj oblasti omogućava dijeljenje koda sa ostatkom open-source zajednice u svrhu njegovog razvijanja. Nadalje, Java kao jezik i tehnologija uz Java virtualnu mašinu omogućava cross-platform development, što je također bio jedan od ciljeva, tako da softver može da se pokrene na mnoštvu operativnih sistema koji podržavaju Java platformu. Java također kao jezik visokog nivoa ima jednu od najrazvijenijih grafičkih biblioteka, što je omogucilo olakšan razvoj grafičkog interfejsa programa. Također, s obzirom na osobine paralelnosti ćelijskih automata kao sistema, uzeta je u obzir i Javina podrška paralelizaciji zadataka koja bi mogla biti korisna u ovoj oblasti za ubrzanje i optimizaciju izračunavanja. Nedostatak Jave je nemogućnost eksplicitne manipulacije memorijom, međutim u slučaju ovog softvera to je više pozitivna nego negativna osobina s obzirom da programer svakako nema potrebe za eksplicitnom alokacijom, dok je garbage collection osobina koja također olakšava i ubrzava proces razvoja s obzirom da se ne mora paziti na detalje niskog nivoa.

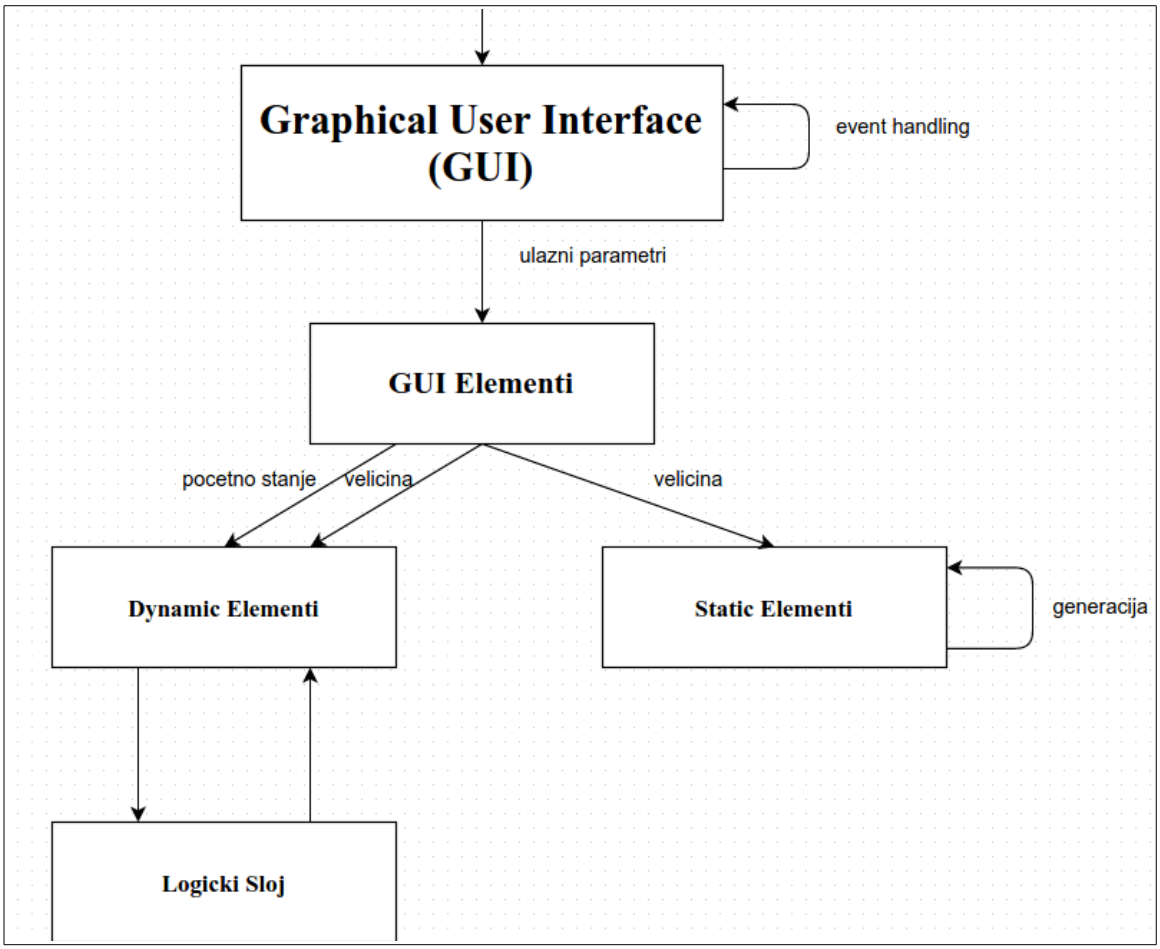
Netbeans razvojno okruženje prikazano na *slici A3* izabrano je iz razloga jednostavnosti integrisanog GUI dizajnera, te veće stabilnosti na platformi Linux od najvećeg konkurenta Eclipse IDE-a.

Možemo napomenuti da je razvoj vršen na Linux operativnom sistemu, ali to je suštinski nebitno s obzirom na portabilnost i cross-platform osobine Java programskog jezika.

A.4 Arhitektura softvera

A.4.1 Pregled generalne arhitekture

Potrebno je izvršiti i pregled arhitekture softvera da bi se detaljije shvatile Specifičnosti istog, s obzirom da je uvijek dobro imati neku vrstu pregleda visokog nivoa. Kroz ovaj pregled, u ovom slučaju generalna arhitektura sistem, daje mogućnost stvaranja šire slike sistema na višem nivou, gdje se pojedini dijelovi posebno obrađuju i detaljnije popunjavaju implementacionim Specifičnostima.



Slika A4. *Pregled arhitekture sistema najvišeg nivoa.*

Na slici A4 prikazana je pomenuta arhitektura. Vidimo da na najvišem nivou imamo grafički korisnički interfejs (eng. GUI – Graphical User Intrefaće), koji je jedini sloj koji direktno dolazi u kontakt s korisnikom. Tačnije, kako je viđeno u pregledu feature softvera, korisnik daje ulazne parametre programu kroz upravo ovaj interfejs. Pozitivna

osobina grafičkih interfejsa je to što se lako može kontrolisati korisnički ulaz kroz grafičke elemente, pa je tako na primjer velicina automata koji se simulira kontrolisana sliderom koji ima definisane minimume i maksimume, što uklanja potrebu za eksplicitnom validacijom ulaza.

Na ovom sloju grafičkog interfejsa, sva logika se svodi na procesiranje takozvanih eventa – dogadjaja na grafičkom interfejsu, te poziva odgovarajucih nižih slojeva u zavisnosti od akcije zahtijevane dogadjajem. Ovo je prikazano na dijagramu sa desne strane, gdje ćemo uspostaviti konvenciju da strelice označavaju prenos informacija, dok strelice koje se vraćaju u objekat definišu vrstu logičkog procesiranja samog sloja/elementa. Dogadjaji su generisani od strane korisnika njegovom interakcijom sa grafičkim elementima.

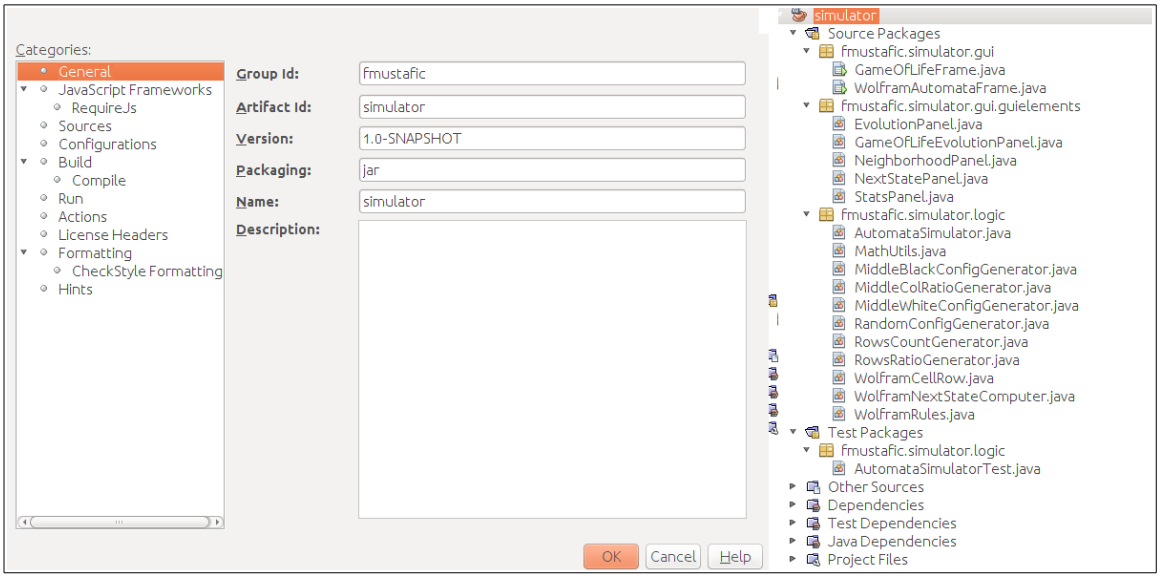
Grafički interfejs sastoji se od standardnih grafičkih elemenata poput dugmadi i slidera, međutim javlja se potreba i za specifičnim custom elementima kao što je u našem slučaju panel za prikaz evolucije ćelijskih automata. Upravo to je idući sloj arhitekture na slici prikazan kao GUI Elementi blok. Na ovom sloju postoje specifične implementacije grafičkih elemenata koji nisu dio standardnog toolkita nekog jezika, te se potreba za njima javlja pretežno iz domene problema koji se razmatra. Na ovom sloju možemo uočiti potrebu za dvije vrste elemenata – statični i dinamični (također prikazano na slici A4).

Statički elementi su takvi da sam njihov izgled i raspored elemenata unutar njih ne zavisi od ulaznih parametara, te je jedina varijabla koja je bitna za njih njihova veličina. Logika unutar ovih elemenata svodi se na jednostavno procesiranje veličine i zavisnosti pozicije dijelova elementa u zavisnosti od veličine.

Dinamični elementi su nešto kompleksniji, te logika za njihovo generisanje nije toliko jednostavna i zavisi pretežno od domena problema koji se razmatra – ćelijski automati u nasem slučaju. Tako da je kao prvo potrebno dati odgovarajuće informacije o početnim uslovima od kojih zavisi izgled ovih elemenata, te je potrebno omoguciti još neki sloj koji bi služio za procesiranje i generisanje cjelokupnog izgleda, s obzirom da bi bila loša praksa “natrpati” svo ovo procesiranje u grafičke elemente. Iz ovog razloga za dinamične elemente imamo još dodatni sloj arhitekture, koji predstavlja logiku generisanja strukture istih, te odvaja grafički od logičkog dijela aplikacije što omogućava razna implementaciona razmatranja za logički sloj Specifično.

A.4.2 Specifična implementacija arhitekture

Obrađena arhitektura implementirana je Specifično prema implementacionim odlukama u prošlom poglavlju, te će ovdje biti dat kratak pregled ove specifične implementacije za slučaj simulacije elementarnih ćelijskih automata sa raznim parametrima.



Slika A5. *Specifična implementacija arhitekture.*

Na slici A5 prikazana je ova Specifična implementacija. Na lijevoj strani prikazane su osvnovne informacije o projektu unutar NetBeans okruženja, dok su sa desne strane dati source-code file-ovi koji predstavljaju implementaciju.

Java paketi predstavljaju način grupisanja logički povezanih file-ova, te ako pogledamo desnu stranu slike A5, vidimo da postoje tri paketa od kojih svaki ugrubo odgovara sloju pomenute arhitekture.

U GUI sloju koji odgovara fmustafic.simulator.gui paketu u projektu nalaze se Frame klase koje predstavljaju ugradjene Java containere za prikaz ostalih elemenata. Također na njima se nalaze osnovni GUI elementi poput buttona, te logika u ovim file-ovima predstavlja većinski event handling.

U sloju grafičkih elemenata, potrebno je bilo custom definisati nekoliko istih. Vidimo da paket fmustafic.simulator.guielements sadrži elemente potrebne za prikaz evolucije automata, grafički izbor pravila evolucije te prikaz statistike koji se nalaze u source code fileovima EvolutionPanel, NeighborhoodPanel i StatsPanel respektivno. Svaka od ovih klasa predstavlja Panel tip koji također predstavlja vid containerskog elementa u Javi.

Na kraju paket fmustafic.simulator.logic koji odgovara najnižem sloju arhitekture sadrži

klase odgovorne za logičko procesiranje evolucije ćelijskih automata, te generisanje slučajnih početnih uslova za elementarni ćelijski automat.

A.4.3. Implementacione odluke po slojevima

Razmotrimo još i specifične odluke u konkretnoj implementaciji koje su se morale donijeti po slojevima arhitekture.

Na najvišem sloju arhitekture, odabrana je javax.swing biblioteka za konstrukciju i implementaciju grafičkih elemenata.

Dalje, na sloju ispod za generisanje specijalnih grafičkih elemenata iz problem domena, korišten je Java Graphics objekat koji omogućava pixelwise manipulaciju elementa, te postavljanje svakog piksela elementa na željeno stanje pojedinačno. Na prva dva sloja nije bilo previše razmatranja s obzirom da postoji poprilično ustaljena procedura za kreiranje grafičkog interfejsa.

Na sloju logike, s obzirom na visok nivo paralelizma u razmatranim sistemima, bilo je razmotrena i opcija implementacije preko Java podrške paralelizmu, te ispod možemo vidjet snippet koda koji prikazuje slučaj upotrebe iste.

```
...
WolframNextStateComputer right =
    new WolframNextStateComputer(
        this.currentState,
        this.from + (this.to - this.from) / 2,
        this.to,
        this.rule
    );

left.fork();
Boolean[] rightResult = right.compute();
Boolean[] leftResult = left.join();

Boolean[] nextState = new Boolean[leftResult.length + rightResult.length];

System.arraycopy(leftResult, 0, nextState, 0, leftResult.length);

for(int i = leftResult.length; i < nextState.length; i++) {
    nextState[i] = rightResult[i - leftResult.length];
}

return nextState;
...
```

Reference

[1] Schiff J., 2007, *Cellular Automata: A Discrete View of the World*, John Wiley & Sons, Inc., Publication

[2] Wolfram S., 2002, *A New Kind of Science,* Wolfram Media Place

[3] Wolfram S., 1983, *C*ellular Automata, Los Alamos Science 9 2–21.

[4] Wolfram S., 1984, *Universality and Complexity in cellular Automata*, Physica D: Nonlinear Phenomena10, no. 1–2

[5] Wolfram S., 1988, *Cellular Automaton Supercomputing,* High-Speed Computing: Scientific Applications and Algorithm Design University of Illinois Press, pp, 40–48

[6] Wolfram S., 1983, *Statistical Mechanics of cellular Automata*, Reviews of Modern Physics 55, no. 3, pp. 601–644

[7] Wolfram S., 1985, *Cryptography with cellular Automata,* Advances in Cryptology: CRYPTO '85 Proceedings

[8] Cook M., 2004, *Universality in Elementary Cellular Automata,* Complex Systems 15, pp. 1-40

[9] Langton C., 1990, *Computation at the edge of chaos*

[10] Gray M. R., *Entropy and Information Theory First Edition*

[11] Mitchell , James P. ,Crutchfield and Hraber, 1994, *Dynamics, Computation, and the "Edge of Chaos": A Re-Examination*

[12] Wolfram S., 1986, *Random Sequence Generation by Cellular Automata*, Advances in Applied Mathematics 7, no. 2, pp.123–169.

[13] Wolfram S., 1986, *Cryptography with Cellular Automata,* Advances in Cryptology: CRYPTO '85 Proceedings

[14] Meier W. and Staffelbach O., 1991, *Analysis of Pseudo Random Sequences Generated by Cellular Automata*, Advances in Cryptology — EUROCRYPT '91 Volume 547 of the series Lecture Notes in Computer Science, pp. 186-199

[15] Cook M., 2004, *Universality in Elementary Cellular Automata,* Complex Systems 15, pp. 1-40

[16] Cook M, 2009, *A Concrete View of Rule 110 Computation*, arxiv.org

[17] Easley D. and Kleinberg J., 2010, *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*, Cambridge University Press

[18] Schelling T., 1971, *Dynamic Models of Segregation of Pittsburgh*, Harvard University Press

B. Listing koda

U ovom dijelu biće naveden listing koda realizovanog simulatora.

EvolutionPanel.java

```
package fmustafic.simulator.gui.guielements;

import fmustafic.simulator.logic.AutomataSimulator;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import javax.swing.BorderFactory;
import javax.swing.JPanel;

/**
 *
 * @author faruk
 */
public class EvolutionPanel extends JPanel {

    public EvolutionPanel(
            Boolean[] initialConfig, int stepsRequired, int size, int rule,
            Color onColor, Color offColor
    ) {
        this.setPreferredSize(new Dimension(size, size));
        this.setSize(new Dimension(size, size));

        this.states = new Boolean[stepsRequired][];

        this.states[0] = initialConfig;
        this.stepsRequired = stepsRequired;
        this.rule = rule;

        this.setBorder(BorderFactory.createLineBorder(Color.BLACK));

        this.onColor = onColor;
        this.offColor = offColor;
    }

    @Override
    public void paintComponent(Graphics g) {
        int unitLength = this.getWidth() / this.states[0].length;
```

```
        AutomataSimulator simulator =
              new AutomataSimulator(this.states[0], this.stepsRequired, this.rule);
        Boolean[][] evolutionSteps = simulator.getSteps();

        for(int i = 0; i < evolutionSteps.length; i++) {
           for(int j = 0; j < evolutionSteps[i].length; j++) {
              g.setColor(evolutionSteps[i][j] ? onColor : offColor);
              g.fillRect(j * unitLength, i * unitLength, unitLength, unitLength);
           }
        }
     }

     public Boolean[][] getStates() {
        AutomataSimulator simulator =
           new AutomataSimulator(this.states[0], this.stepsRequired, this.rule);
        Boolean[][] statesCopy = simulator.getSteps();

        return statesCopy;
     }

     private Boolean[][] states;
     private int stepsRequired;
     private int rule;

     private Color onColor;
     private Color offColor;

}
```

GameOfLifeEvolutionPanel.java

```
package fmustafic.simulator.gui.guielements;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import javax.swing.BorderFactory;
import javax.swing.JPanel;

public class GameOfLifeEvolutionPanel extends JPanel {
   private final Boolean[][] states;

   public GameOfLifeEvolutionPanel(Boolean[][] states, int width, int height) {
```

```
        this.setPreferredSize(new Dimension(width, height));
        this.setSize(new Dimension(width, height));
        this.setBorder(BorderFactory.createLineBorder(Color.BLACK));

        // no check...
        this.states = new Boolean[states.length][];
        for(int i = 0; i < states.length; i++) {
            this.states[i] = states[i].clone();
        }
    }

    @Override
    public void paintComponent(Graphics g) {
        // assuming equal lengths in states
        int unitWidth = this.getWidth()/ this.states[0].length;
        int unitHeight = this.getHeight() / this.states.length;

        System.out.println("height: " + this.getHeight());
        System.out.println("height length: " + this.states.length);
        System.out.println("width: " + this.getWidth());
        System.out.println("width length: " + this.states[0].length);

        System.out.println("uw, uh: " + unitWidth + " " + unitHeight);

        for(int i = 0; i < this.states.length; i++) {
            for(int j = 0; j < this.states[i].length; j++) {
                g.setColor(states[i][j] ? Color.BLACK : Color.WHITE);
                g.fillRect(j * unitWidth, i * unitHeight, unitWidth, unitHeight);

            }
        }
    }

}
```

NeighborhoodPanel.java

```
package fmustafic.simulator.gui.guielements;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
```

```
import javax.swing.BorderFactory;
import javax.swing.JPanel;

/**
 *
 * @author faruk
 */
public class NeighborhoodPanel extends JPanel {

    private Boolean left, middle, right;

    public NeighborhoodPanel(String neighborhood, int width, int height) {

    }

    public NeighborhoodPanel(
        Boolean left, Boolean middle, Boolean right, int width, int height
    ) {
        init(left, middle, right, width, height);
    }

    private void init(Boolean left, Boolean middle, Boolean right, int width, int height) {
        this.setPreferredSize(new Dimension(width, height));

        this.left = left;
        this.middle = middle;
        this.right = right;

        this.setBorder(BorderFactory.createLineBorder(Color.BLACK));
    }

    @Override
    public void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
        g2.setStroke(new BasicStroke(1));

        g2.setColor(left ? Color.BLACK : Color.WHITE);
        g2.fillRect(0, 0, this.getWidth() / 3, this.getHeight());

        // slash
        g2.setColor(Color.BLACK);
        g2.drawRect(0, 0, this.getWidth() / 3, this.getHeight());

        g2.setColor(middle ? Color.BLACK : Color.WHITE);
        g2.fillRect(this.getWidth() / 3, 0, this.getWidth() / 3, this.getHeight());
```

```
        // slash
        g2.setColor(Color.BLACK);
        g2.drawRect(this.getWidth() / 3, 0, this.getWidth() / 3, this.getHeight());

        g2.setColor(right ? Color.BLACK : Color.WHITE);
        g2.fillRect(2 * this.getWidth() / 3, 0, this.getWidth() / 3, this.getHeight());

        // slash
        g2.setColor(Color.BLACK);
        g2.drawRect(2 * this.getWidth() / 3, 0, this.getWidth() / 3, this.getHeight());
    }

}
```

NextStatePanel.java

```
package fmustafic.simulator.gui.guielements;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import javax.swing.BorderFactory;
import javax.swing.JPanel;

/**
 *
 * @author faruk
 */
public class NextStatePanel extends JPanel implements MouseListener {

    Boolean currentState;

    public NextStatePanel(int width, int height, Boolean initialState) {
        this.currentState = initialState;

        this.setPreferredSize(new Dimension(width, height));
        this.setBorder(BorderFactory.createLineBorder(Color.BLACK));

        this.addMouseListener(this);
    }
```

```
    public Boolean getCurrentState() {
        return this.currentState;
    }

    @Override
    public void paintComponent(Graphics g) {
        g.setColor(currentState ? Color.BLACK : Color.WHITE);
        g.fillRect(0, 0, this.getWidth(), this.getHeight());
    }

    @Override
    public void mouseClicked(MouseEvent e) {
        this.currentState = !this.currentState;
        this.repaint();
    }

    @Override
    public void mousePressed(MouseEvent e) {
    }

    @Override
    public void mouseReleased(MouseEvent e) {
    }

    @Override
    public void mouseEntered(MouseEvent e) {
    }

    @Override
    public void mouseExited(MouseEvent e) {
    }

}
```

StatsPanel.java

```
package fmustafic.simulator.gui.guielements;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import javax.swing.BorderFactory;
import javax.swing.JPanel;
```

```
public class StatsPanel extends JPanel {

    public StatsPanel(double[][] dataSet, int width, int height, Color defaultColor) {
        this.dataSet = dataSet;

        this.defaultColors = new Color[2];
        this.defaultColors[0] = defaultColor;
        this.defaultColors[1] = Color.RED;

        this.setSize(new Dimension(
                (int)(Math.ceil(width / dataSet[0].length) * dataSet[0].length),
                height)
        );
        this.setPreferredSize(new Dimension(
                (int)(Math.ceil(width / dataSet[0].length) * dataSet[0].length),
                height)
        );

        this.setBorder(BorderFactory.createLineBorder(Color.BLACK));

        this.setBackground(Color.BLACK);
    }

    @Override
    public void paintComponent(Graphics g) {
        // first one determines the others
        int unitLength = this.getWidth() / this.dataSet[0].length;

        double maxValue = 0;
        for (double[] dataSet1 : this.dataSet) {
            for (int j = 0; j < dataSet1.length; j++) {
                if (dataSet1[j] > maxValue) {
                    maxValue = dataSet1[j];
                }
            }
        }

        paintLabels(g, maxValue);

        int i = 0;
        for (double[] dataSet1 : this.dataSet) {
            int prevX = -1;
            int prevY = -1;
            for (int j = 0; j < dataSet1.length; j++) {
```

```
            int currX = j * unitLength;
            int currY = this.getHeight() - (int) (0.9 * (dataSet1[j] / maxValue) *
this.getHeight());

            g.setColor(this.defaultColors[i]);

            g.drawRect(
               currX,
               currY,
               1,
               1
            );

            if(prevX != -1 && prevY != -1) {
               g.drawLine(prevX, prevY, currX, currY);
            }

            prevX = currX;
            prevY = currY;
         }
         ++i;
      }
   }

   private void paintLabels(Graphics g, double maxValue) {
      int style = Font.PLAIN;
      g.setFont(new Font("Dialog", style, 12));
      g.setColor(this.defaultColors[0]);

      g.drawString(Double.toString( (int)(maxValue * 100) / 100.0 ), 0, 10);
   }

   private double[][] dataSet;
   private static final int X_OFFSET = 50;
   private static final int Y_OFFSET = 50;
   private Color[] defaultColors;

}
```

AutomataSimulator.java

```
package fmustafic.simulator.logic;

/**
```

```
 *
 * @author faruk
 */
public class AutomataSimulator {

    public AutomataSimulator(Boolean[] state, int stepsRequired, int rule) {
        this.initialState = state;
        this.stepsRequired = stepsRequired;
        this.rule = rule;
    }

    public Boolean[][] getSteps() {
        Boolean[][] stepsResult = new Boolean[this.stepsRequired][initialState.length];
        stepsResult[0] = this.initialState;

        WolframCellRow ruleLogic =
                new WolframCellRow(this.initialState.length, this.rule);

        /* parallele test
        ForkJoinPool fjPool = new ForkJoinPool();
        */

        for(int i = 1; i < stepsResult.length; i++) {
            ruleLogic.setRowState(stepsResult[i - 1]);
            ruleLogic.evolveAll();
            stepsResult[i] = ruleLogic.getRowState();

            /* parallele test
            Boolean[] prevStep = stepsResult[i - 1];

            stepsResult[i] =
                fjPool.invoke(
                    new WolframNextStateComputer(prevStep, 0, prevStep.length, this.rule)
                );
            */
        }

        return stepsResult.clone();
    }

    private Boolean[] initialState;
    private int stepsRequired;
    private int rule;

}
```

MathUtils.java

```
package fmustafic.simulator.logic;

import java.security.InvalidParameterException;

/**
 *
 * @author faruk
 */
public class MathUtils {
   public static Boolean isPowerOf4(int n) {
      if(n <= 0) {
         return false;
      }

      while(n >= 4) {
         n /= 4;
      }

      return n == 1;
   }

   public static Boolean bitAt(int n, int bitPosition) {
      for(int i = 0; i < bitPosition; i++) {
         n = n >> 1;
      }

      return (n % 2 == 1);
   }

   public static int parseBinary(Boolean[] binaryArray) {
      int integerResult = 0;
      int arraySize = binaryArray.length;
      for(int i = arraySize - 1; i >= 0; i--) {
         integerResult +=
               (binaryArray[i]) ? Math.pow(2.0, arraySize - 1 - i) : 0;
      }

      return integerResult;
   }

   public static int parseBinary(String binaryString) {
      int integerResult = 0;
```

```
        int stringLength = binaryString.length();
        for(int i = stringLength - 1; i >= 0; i--) {
            char currentChar = binaryString.charAt(i);
            if(currentChar != '0' && currentChar != '1') {
                throw new InvalidParameterException("Must be 0s and 1s only.");
            }
            integerResult +=
                    (currentChar == '1') ?
                        Math.pow(2.0, stringLength - 1 - i) :
                        0;
        }

        return integerResult;
    }
}
```

MiddleBlackConfigGenerator.java

```
package fmustafic.simulator.logic;

import java.util.Random;

/**
 *
 * @author faruk
 */
public class MiddleBlackConfigGenerator {

    public MiddleBlackConfigGenerator(int n) {
        this.n = n;
    }

    public Boolean[] getConfig() {
        Boolean[] config = new Boolean[n];
        Random randomGen = new Random();

        for(int i = 0; i < n; i++) {
            config[i] = (i == n / 2);
        }

        return config.clone();
    }

    private int n;
```

```
}
```

MiddleColRatioGenerator.java

```
package fmustafic.simulator.logic;

/**
 *
 * @author faruk
 */
public class MiddleColRatioGenerator {

    public MiddleColRatioGenerator(Boolean[][] states) {
        this.states = states;
    }

    public double[][] getDataSet() {
        double[][] dataSet = new double[1][];
        dataSet[0] = new double[this.states.length];

        int onCount = 0;
        for(int i = 0; i < this.states.length; i++) {
            if(this.states[i][this.states[i].length / 2]) {
                ++onCount;
            }

            dataSet[0][i] = (double)(onCount) / (i + 1);
        }

        return dataSet;
    }

    private Boolean[][] states;

}
```

MiddleWhiteConfigGenerator.java

```
package fmustafic.simulator.logic;

import java.util.Random;
```

```
/**
 *
 * @author faruk
 */
public class MiddleWhiteConfigGenerator {

    public MiddleWhiteConfigGenerator(int n) {
        this.n = n;
    }

    public Boolean[] getConfig() {
        Boolean[] config = new Boolean[n];
        Random randomGen = new Random();

        for(int i = 0; i < n; i++) {
            config[i] = !(i == n / 2);
        }

        return config.clone();
    }

    private int n;

}
```

RandomConfigGenerator.java

```
package fmustafic.simulator.logic;

import java.util.Random;

/**
 *
 * @author faruk
 */
public class RandomConfigGenerator {

    public RandomConfigGenerator(int n) {
        this.n = n;
    }

    public Boolean[] getConfig() {
        Boolean[] config = new Boolean[n];
```

```
        Random randomGen = new Random();

        for(int i = 0; i < n; i++) {
            config[i] = randomGen.nextBoolean();
        }

        return config.clone();
    }

    public Boolean[] getConfigWithDensity(double density) {
        Random randomGen = new Random();

        int requiredOnCount = (int)(density * n);

        Boolean[] config = new Boolean[n];
        for(int i = 0; i < config.length; i++) {
            config[i] = false;
        }

        int soFarCount = 0;
        while(soFarCount != requiredOnCount) {
            int cellChoice = randomGen.nextInt(config.length);
            if(!config[cellChoice]) {
                config[cellChoice] = true;
                ++soFarCount;
            }
        }

        return config.clone();
    }

    private int n;

}
```

RowsCountGenerator.java

```
package fmustafic.simulator.logic;
public class RowsCountGenerator {

    public RowsCountGenerator(Boolean[][] states) {
        this.states = states;
    }
```

```
    public double[][] getDataSet() {
        double[][] dataSet = new double[2][];
        dataSet[0] = new double[this.states.length];
        dataSet[1] = new double[this.states.length];

        for(int i = 0; i < this.states.length; i++) {
            int onCount = 0;
            int offCount = 0;

            for(int j = 0; j < this.states[i].length; j++) {
                if(this.states[i][j]) {
                    ++onCount;
                } else {
                    ++offCount;
                }
            }

            dataSet[0][i] = onCount;
            dataSet[1][i] = offCount;
        }

        return dataSet;
    }

    private Boolean[][] states;

}
```

RowsRatioGenerator.java

```
package fmustafic.simulator.logic;

/**
 *
 * @author faruk
 */
public class RowsRatioGenerator {

    public RowsRatioGenerator(Boolean[][] states) {
        this.states = states;
    }

    public double[][] getDataSet() {
        double[][] dataSet = new double[1][];
        dataSet[0] = new double[this.states.length];
```

```
        for(int i = 0; i < this.states.length; i++) {
            System.out.println("ctor " + (this.states[i] == null));
            int onCount = 0;
            for(int j = 0; j < this.states[i].length; j++) {
                if(this.states[i][j]) {
                    ++onCount;
                }
            }

            dataSet[0][i] = (double)(onCount) / this.states[i].length;
        }

        return dataSet;
    }

    private Boolean[][] states;

}
```

WolframCellRow.java

```
package fmustafic.simulator.logic;

import java.security.InvalidParameterException;

/**
 *
 * @author faruk
 */
public class WolframCellRow {

    private WolframRules rules;

    private Boolean[] rowState;
    private Boolean[] nextRowState; // temp

    public WolframCellRow(int size, int ruleEnumeration) {
        this.rowState = new Boolean[size];
        this.nextRowState = new Boolean[size];
        this.rules = new WolframRules(ruleEnumeration);
    }

    public Boolean[] getRowState() {
```

```
    return this.rowState.clone();
}

public void setRowState(Boolean[] state) {
    if (state.length != this.rowState.length) {
        throw new InvalidParameterException(
                "State length != " + this.rowState.length
        );
    }

    this.rowState = state;
}

public void setRowState(String state) {
    if (state.length() != this.rowState.length) {
        throw new InvalidParameterException(
            "State length != " + this.rowState.length
        );
    }

    for (int i = 0; i < state.length(); i++) {
        char currentChar = state.charAt(i);
        if (currentChar != '0' && currentChar != '1') {
            throw new InvalidParameterException("State must be only 0s and 1s");
        }

        this.rowState[i] = (currentChar == '1');
    }
}

private int cntr = 0;

public void evolveAll() {
    for (int i = 0; i < this.rowState.length; i++) {
        evolveSpecific(i);
    }

    this.rowState = this.nextRowState.clone();
}

public void evolveSpecific(int n) {
    if (n < 0 || n >= this.rowState.length) {
        throw new InvalidParameterException(
                "n < 0 || n >= this.rowState.length."
        );
```

```
        }

        int leftIndex
                = (n - 1 < 0) ? (rowState.length - 1) : (n - 1);
        int rightIndex
                = (n + 1 >= rowState.length) ? 0 : (n + 1);

        this.nextRowState[n] = this.rules.getNextState(
           this.rowState[leftIndex],
           this.rowState[n],
           this.rowState[rightIndex]
        );
    }

}
```

WolframNextStateComputer.java

```
package fmustafic.simulator.logic;

import java.util.Arrays;
import java.util.concurrent.RecursiveTask;

/**
 *
 * @author faruk
 */
public class WolframNextStateComputer extends RecursiveTask<Boolean[]> {
    private final Boolean[] currentState;
    private final int rule;
    private final int from;
    private final int to;
    private static final int SIZE_TRESHOLD = 250;

    public WolframNextStateComputer(
        Boolean[] currentState,
        int from,
        int to,
        int rule
    ) {
        this.currentState = currentState;
        this.from = from;
        this.to = to;
        this.rule = rule;
```

```
}

@Override
protected Boolean[] compute() {
   System.out.println("from: " + this.from);
   System.out.println("to: " + this.to);
   if(this.to - this.from <= SIZE_TRESHOLD) {
      Boolean[] nextState = new Boolean[this.currentState.length];

      WolframRules rules = new WolframRules(this.rule);

      for(int i = from; i < to; i++) {
         int leftIndex
            = (i - 1 < 0) ? (this.currentState.length - 1) : (i - 1);
         int rightIndex
               = (i + 1 >= this.currentState.length) ? 0 : (i + 1);

         nextState[i] = rules.getNextState(
            this.currentState[leftIndex],
            this.currentState[i],
            this.currentState[rightIndex]
         );
      }

      return nextState;
   } else {
      int stateSize = this.currentState.length;
      WolframNextStateComputer left =
         new WolframNextStateComputer(
            this.currentState,
            this.from,
            this.from + (this.to - this.from) / 2,
            this.rule
      );

      WolframNextStateComputer right =
         new WolframNextStateComputer(
            this.currentState,
            this.from + (this.to - this.from) / 2,
            this.to,
            this.rule
      );

      left.fork();
      Boolean[] rightResult = right.compute();
```

```
            Boolean[] leftResult = left.join();

            Boolean[] nextState = new Boolean[leftResult.length + rightResult.length];

            System.arraycopy(leftResult, 0, nextState, 0, leftResult.length);

            for(int i = leftResult.length; i < nextState.length; i++) {
               nextState[i] = rightResult[i - leftResult.length];
            }

            return nextState;
         }
      }

}
```

WolframRules.java

```
package fmustafic.simulator.logic;

import fmustafic.simulator.logic.MathUtils;
import java.security.InvalidParameterException;

/**
 *
 * @author faruk
 */
public class WolframRules {

   private int ruleEnumeration;

   public WolframRules(int ruleEnumeration) {
      this.setRuleEnumeration(ruleEnumeration);
   }


   /**
    * @return the ruleEnumeration
    */
   public final int getRuleEnumeration() {
      return ruleEnumeration;
   }

   /**
```

```
     * @param ruleEnumeration the ruleEnumeration to set
     */
    public final void setRuleEnumeration(int ruleEnumeration) {
        if(ruleEnumeration < 0 || ruleEnumeration > 256) {
            throw new InvalidParameterException("Ruleset not in bounds.");
        }

        this.ruleEnumeration = ruleEnumeration;
    }

    public final Boolean getNextState(
            Boolean left,
            Boolean element,
            Boolean right
    ) {
        return MathUtils.bitAt(
            this.getRuleEnumeration(),
            MathUtils.parseBinary(new Boolean[] { left, element, right })
        );
    }

    public final Boolean getNextState(String neighborhood) {
        if(neighborhood.length() != 3) {
            throw new InvalidParameterException("Neighborhood != 3 cells.");
        }

        return MathUtils.bitAt(
            this.getRuleEnumeration(),
            MathUtils.parseBinary(neighborhood)
        );
    }


}
```