Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Robert Logožar

Modeliranje dinamičkih sustava *s pomoću* stohastičkih konačnih automata

Magistarski rad

Zagreb, 1999.

Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Robert Logožar

Modeliranje dinamičkih sustava *s pomoću* stohastičkih konačnih automata

Magistarski rad

Zagreb, 1999.

University of Zagreb
Faculty of Electrical Engineering
and Computing

Robert Logozar

Modeling of Dynamical Systems *by* Stochastic Finite Automata

Master Thesis

Zagreb, 1999

Robert Logožar, magistarski rad:

Modeliranje dinamičkih sustava s pomoću stohastičkih konačnih automata

Rad je izrađen: *na Fakultetu Organizacije i informatike u Varaždinu, Sveučilište u Zagrebu, u razdoblju od svibnja 1998. do ožujka 1999.*

Mentor rada: prof. dr. sc. Leo Budin,
Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave (ZEMRIS),
Fakultet elektrotehnike i računarstva,
Sveučilište u Zagrebu.

Rad sadrži:
155 stranica (+ xvii početnih),
30 slika, i
3 tablice.

Rad broj:

Povjerenstvo za ocjenu magistarskog rada:

1. prof. dr. sc. Siniša Srbljić, predsjednik;
2. prof. dr. sc. Leo Budin, (mentor);
3. prof. dr. sc. Nikola Bogunović.

Povjerenstvo za obranu magistarskog rada:

1. prof. dr. sc. Siniša Srbljić, predsjednik;
2. prof. dr. sc. Leo Budin, (mentor);
3. prof. dr. sc. Nikola Bogunović.

Datum obrane: 14. srpnja, 1999.

Modeliranje dinamičkih sustava s pomoću stohastičkih konačnih automata

Robert Logožar

Zadatak magistarskog rada:

Razmotriti teorijske postavke ϵ-strojeva i njihovu povezanost s teorijom automata. Diskutirati pojmove statističke složenosti kao moguće kvantitativne mjere za strukturalnu složenost fizikalnih sustava. Obrazložiti algoritam konstrukcije ϵ-strojeva koji, za razliku od poznatog razreda konačnih automata iz teorije izračunavanja, utjelovljuju i stohastičke elemente. Razviti programsku podršku za izračun i prikaz izloženih teorijskih razmatranja koja bi trebala olakšati eksperimentiranje. Nastojati automatizirati konstrukciju ϵ-strojeva do razine stohastičkih konačnih automata. Omogućiti provjeru ovako dobivenog modela na dobro poznatim nelinearnim dinamičkim sustavima, uz razmatranje divergencije modela na početku puta prema kaotičnom režimu udvostručavanjem perioda.

Sadržaj

Popis slika

Popis tablica

Poglavlje 1

Uvod

Posljednja desetljeća ovog stoljeća obilježena su velikim zanimanjem znanstvene zajednice za proučavanje nelinearnih sustava. Razvoj računarstva i mogućnost numeričke integracije donedavna nerješivih diferencijalnih jednadžbi doveli su do otkrića onog što danas poznajemo pod nazivom *deterministički kaos.* U naizgled jednostavnim sustavima pronađeno je ogromno bogatstvo pojava iza kojih se naziru univerzalni i temeljni prirodni principi. Stoga nije čudno da se interes za nelinearne fenomene proširio na gotovo sva znanstvena područja — od fundamentalnih fizikalnih, preko bioloških i medicinskih, pa sve do tehničkih i ekonomskih. Izučavanje kaotičnih sustava ubrzo je postalo sinonim za interdisciplinarnost moderne znanosti.

U ovom radu opisat ćemo temeljne postavke jednog novijeg pristupa izučavanju dinamičkih sustava, kojemu je cilj izrada računski ustrojenog modela prema idejama i teoriji koju razvija J. P. Crutchfield [1]. Izložit ćemo rezultate u razvoju programske podrške za iznalaženje tzv. *stohastičkih konačnih automata*, na temelju vremenskog niza podataka u kojem je zapisan tijek dinamike sustava. Ovakav pristup ne primjenjuje dostignuća računarskih teorija tek za operacionalizaciju obrade matematički formuliranog problema, već se računarski alati koriste kao temeljni instrumentarij za opis i modeliranje sustava.

Namjera je uvodnog poglavlja da ukratko podastre filozofsku pozadinu ove srazmjerno nove metode i da u kratkom kvalitativnom pregledu iznese glavne postavke ovog rada. Potom ćemo se u idućim poglavljima usredotočiti na formalne i izvedbene detalje.

1.1 Od tradicionalnog do novih pristupa

Tradicionalni klasično-fizikalni pristup izučavanja dinamičkih sustava, temelji se na mehaničkom determinizmu. Analiza sustava se tu svodi na uočavanje vremenski ovisnih veličina, što rezultira postavljanjem diferencijalnih jednadžbi. Time je sustav u potpunosti opisan, a daljnje nalaženje vremenske funkcije stanja kao željenog rješenja svodi se na primjenu matematičkog aparata. Taj je pristup, zbog svoje egzaktnosti i revolucionarnog povijesnog utjecaja na cjelokupni razvoj novovjeke znanosti, uvelike dominirajući u tehnici, a do prije otkrića kvantne teorije i u fizici. Štoviše, ako se ne radi o kvantnom nego o klasičnom sustavu, to je i danas podrazumijevajuća znanstvena metoda egzaktnih znanosti. Prije nego se pozabavimo mogućim alternativama, promotrimo najprije njene osnovne značajke.

1.1.1 Opis s pomoću diferencijalnih jednadžbi

Postavljanje diferencijalnih jednadžbi, ili pak jednadžbi diferencija za nekontinuirane sustave, zadaća je koju znanstvenik treba obaviti na temelju prikupljenih podataka o sustavu i njegovu ustroju. Pri tome se mora uočiti sva epistemološka dubina navedenih pojmova, jer se "ustroj sustava" i "način prikupljanja podataka" ne može izdvojiti izvan konteksta konkretne teorije u okviru koje znanstvenik obavlja svoja istraživanja. Za stručnjake unutar pojedinih područja, pogotovo onih tehničke naravi — u kojima se istraživanja *terrae incognitae* uglavnom prepuštaju temeljnim (prirodnim) znanostima — poznavanje takve teorije dio je pogleda na svijet i prirodu stvari i to se gotovo bespogovorno podrazumijeva kao dio struke. Zadatak se tu najčešće svodi na opis drugačijeg, još neproučenog ili na poseban način složenog primjera, ali u okviru određene teorije te uz uporabu njenog osobitog "jezika". Jezik egzaktnih prirodnih teorija i tehničkih znanosti inherentno podrazumijeva, odnosno propisuje, način promatranja i mjerenja sustava. Proučavanje prirodnih i tehničkih sustava u okviru klasične mehanike ili elektrodinamike upravo propisuje gore navedenu postavku diferencijalnih jednadžbi kao odabrani način opisivanja sustava.

Međutim, usprkos svojoj velikoj uspješnosti i dubokoj ukorijenjenosti u sam temelj znanstvenog postupka, ovakav standardan postupak nije uvijek plodotvoran i, što je važno uočiti, on ne mora biti inherentan promatranom objektu ili barem nekom njegovom (trenutnom) stanju. Upravo su nelinearni sustavi nagnali znanstvenike iz različitih područja da preispitaju svoje uvriježene metodologije, te da počnu stvarati novi, poseban jezik za opis kaotičnih fenomena.

1.1.2 Standardna numerička metodologija

Ono što se na koncu očekuje kao produkt numeričke analize dinamičkog sustava jest — pored nekolicine numeričkih pokazatelja — uglavnom grafički prikaz rezultata u formi faznih i orbitalnih dijagrama te spektralnih analiza. Takav način upoznavanja s dinamičkim sustavima toliko se ustalio da se danas smatra neizostavnim. Upravo je područje determinističkog kaosa poznato po gotovo poetičnom isticanju estetske vrijednosti slikovitih prikaza onog što široj publici i ne mora biti sasvim razumljivo, ali će zasigurno biti lijepo. Zapanjujući odnos cjeline i detalja, složenost koja se razbija na jednostavnu pravilnost — počevši od geometriziranih fraktala, pa do prirodi nalik koloriranih ilustracija ponašanja nelinearnih kompleksnih funkcija. Svi su ti prikazi, baš kao i priroda, lijepi i zanimljivi i iz daleka i izbliza, a dobiveni su kao rezultat numeričke simulacije dinamičkih sustava.

Možda je tu najilustrativnije navesti razmišljanja upravo onih koji bi po svojoj vokaciji mogli biti najkritičniji glede uporabe i vrijednosti numeričkog pristupa — matematičara. Tako mnogi od njih koji se bave nelinearnim sustavima svesrdno podstiču uporabu numeričkih metoda za onaj "prvi susret" s problemom, i to ne samo na pedagoškoj razini i u početnom pristupu, već i kao neizostavno i vrlo korisno istraživačko sredstvo za razvoj intuicije te za stjecanje vizije o tome što bi se trebalo činiti na dubljem razumijevanju problema[2]. Naravno, zbog posebnih svojstava kaotičnih sustava, kao što je npr. osjetljivost na početne uvjete, u matematički rigoroznom razmatranju rezultati numeričkih metoda ne mogu se prihvatiti kao dokaz. Zato poslije digitalnih simulacija i grafičkih ilustracija mora uslijediti egzaktan matematički aparat. Iz upravo izloženog jasno je da će znanosti koje su više izložene problematici i kompleksnosti stvarnog svijeta (fizika), ili pak nužnosti brzog iznalaženja praktičnih rješenja (tehnika), više insistirati na konkretnom, numeričkom pristupu, a manje na onom apstraktnom i rigoroznom. No u oba se područja ne prestaju osluškivati spoznaje koje o nelinearnim fenomenima daje egzaktan matematički pristup, kako bi se što bolje razumjela njihova suština.

1.1.3 Diskretizacija modela

U povijesti razvoja ovog područja ističemo dvije, za naš rad bitne odrednice, koje imaju zajedničku nit u diskretizaciji modela. Prvu predstavlja rad G. D. Birkhoffa, koji je — oslonivši se na Poincaréov kvalitativni pristup — umjesto diferencijalnih jednadžbi uveo razmatranje iterativnih procesa. To se svodi na prevođenje dife-

rencijalnih jednadžbi — koje odgovaraju *glatkom tijeku*, na jednadžbe diferencija — koje odgovaraju diskretnom vremenskom tijeku, što se načelno postiže uporabom tzv. Poincaréovog reza[4]. Nakon diskretizacije vremena razvoj dinamičkih sustava opisuje se *iterativnim preslikavanjima* (engl. *iterative mapping*). Ovakav se pristup uvriježio zbog svoje formalne jednostavnosti i preglednosti, posebice kod jednodimenzionalnih sustava. Stoga ga koristimo i u ovom radu.

Druga je važna apstrakcija diskretizacija varijabli stanja, čime se proučavanje dinamičkih sustava prevađa u *simboličku dinamiku* (vidjeti npr. [3]). U njoj se svako stanje sustava označava zasebnim simbolom, a dinamika sustava izražava se kroz dinamiku tih simbola posredstvom *operatora posmaka* (engl. *shift operator*). Za opis mnogih jednostavnih sustava (preslikavanja) bit će dostatna i elementarna, binarna abeceda, $\mathcal{A} = \{0, 1\}$. Uvođenjem simboličke dinamike izučavanje dinamičkih sustava suštinski se povezuje s teorijom računarstva.

1.2 Prema novim metodama

U ovom radu obrazlažemo jedan relativno noviji pristup izučavanju jednodimenzionalnih (1D) dinamičkih sustava, s namjerom da se on primijeni na nelinearne fenomene. Ono što je zanimljivo sa stanovišta računarskih znanosti jest da se u nekim posve fundamentalnim područjima — od kojih su mnoga tek nedavno prepoznata i začeta — rabe dostignuća teorije automata i formalnih jezika, odnosno teorije izračunavanja. Radi se npr. o područjima *nelinearnog modeliranja i predviđanja*, *fizici informacije*, *teoriji strukture (složenosti), računarskoj ergodičkoj teoriji*[5, 6]. Također, *teorija informacija* od svog nastanka predstavlja svojevrstan izazov termodinamici u pitanjima odnosa materijalnih fizikalnih objekata s jedne i informacije kao nematerijalnog fenomena s druge strane. Postavlja se i pitanje o mogućoj *kvantifikaciji strukture* u prirodnim procesima[7]. Svjedoci smo struje sveopće interdisciplinarnosti u znanosti, uvelike potaknute proučavanjem istih ili sličnih fenomena u svezi s nelinearnim sustavima.

1.2.1 Računarske teorije kao novi pristup

Dostignuća računarskih znanosti u posljednjim su dekadama jednako značajna i na praktičnom i na teorijskom planu, pa je i njihovo uključivanje u istraživanje ostalih područja logična posljedica tih velikih uspjeha. Nakon pola stoljeća sve intenzivnije uporabe računala u standardnom numeričkom modeliranju i simulaciji prirodnih

procesa javlja se ideja da se za te procese razmotri i njihov *računski ustroj*. Ako je rezultat mjerenja nepoznatog sustava niz znakova koje je kodirao neki mjerni instrument, taj se niz može interpretirati kao određeni izričaj u jeziku kojim sustav govori. U analizi gramatike tog jezika uporabit ćemo *teoriju formalnih jezika*. Po nalaženju gramatike možemo zaključiti i na njoj ekvivalentan računski automat, koji tu gramatiku razaznaje. Taj automat kroz svoju procesnu moć te količinu i organizaciju korištene memorije odaje i računski ustroj sustava (pogl. 3).

Iako bi se na prvi pogled ovaj pristup mogao doimati kao krajnje artificijelan, suštinski gledano on nije "neprirodniji" od danas uvriježenih metodologija. Newtonovim su suvremenicima jednako umjetno mogle izgledati neke postavke aksiomatskog pristupa mehanici, kao npr. "djelovanje sile na daljinu". Ono što je u konačnici mjerodavno za prosudbu računskog i svih drugih pristupa modeliranju jest njihova uspješnost u opisivanju i predskazivanju stanja promatranog sustava.

Jasno je da izložene teze otvaraju niz novih pitanja. Usporedo s uvođenjem novog rakursa u znanstvena istraživanja svakako je uputno preispitati i aktualno poimanje o mogućnostima spoznaje. U to spada procjena spoznajne vrijednosti rezultata mjerenja te na njima izgrađenih znanstvenih modela i teorija, na što ćemo se ukratko osvrnuti u odjeljku 4.1. Opširnija epistemološka analiza, kakva je navedena npr. u [8, 9], prelazi okvire i tematiku ovog rada.

1.2.2 Minimalna prezentacija

Nakana prethodne diskusije bila je obrazloženje motivacije za uvođenje novog i drugačijeg diskursa u izučavanju dinamičkih sustava. U njemu se polazi od semantičkog konteksta koji je ovisan i izgrađen od podataka samih. Zbog toga je ovaj pristup lišen dodatnih mogućih zamagljenja zbog loše, npr. prekompleksne, prezentacije sustava u okviru određene teorije.[1] Kao potkrjepu ovom razmišljanju navest ćemo znakovit primjer iz [9]. U Fourierovoj analizi bazni skup funkcija jest potpun i konačan. No prikaz kvadratnog vala sadrži beskonačan skup viših harmonika. Iako se njihova zastupljenost može izračunavati u konačnom programu, veličina modela divergira s

[1] Primijetimo ovdje da je i način mjerenja povezan s teorijom, odnosno da mjerenjima moraju prethoditi premise i izvjesne ekstrapolacije glede modela koji još ne poznajemo. To može biti posebno važno u kvantnoj domeni kad karakter eksperimenta određuje način ponašanja sustava. No može se reći da će — izuzev početne izvedbe mjernog instrumenta — "predrasude" glede budućeg modela u novom pristupu biti minimalne. U odjeljku 4.1 pobliže ćemo obrazložiti problematiku mjernog instrumenta i kanala, a ovdje tek napominjemo da se instrument može i iterativno modificirati u povratnoj sprezi s rezultatima mjerenja.

povećanjem preciznosti opisa. Dakle, usprkos tome što ovakva prezentacija može biti svrsishodna unutar određenog konteksta klasične elektrodinamike (spektralna analiza), ona je potpuno neprikladna glede minimalnosti modela. Rezultati izmjere ovakvog sustava mogu biti opisani jednostavnim deterministčkim automatom, s dva stanja. No za dobivanje točnih podataka o njemu moramo odabrati pravi instrument, koji mora biti ugođen na frekvenciju vala. Odabir pravog instrumenta i (iterativno) podešavanje njegovih parametara predstavlja dodatni, eksperimentalni, skup parametara modela, koji se u našem primjeru svodi na poznavanje osnovne frekvencije sustava. No povrh toga, sva informacija sadržana u podacima idealnog kvadratnog vala sadržana je u svega jednom bitu, pa zaključujemo da će i minimalni računski model u svom rekurentnom dijelu — u kojem je ustanovljena faza — imati svega dva stanja.

Zaključujemo da će uvođenje teorije izračunavanja u najranije faze modeliranja doprinijeti autohtonosti dobivenog modela, odnosno njegovom zasnivanju na inherentnim svojstvima podataka koje primamo od sustava. Također, uz osiguranu iterativnu ugodbu mjernog instrumenta, to u načelu omogućuje nalaženje korektnog i minimalnog takvog modela.

1.3 Izračunska mehanika

Izračunska mehanika (engl. *computational mechanics*) u svojem razmatranju polazi od stanovišta promatrača kojem je cilj zaključiti na računski model nekog sustava iz niza mjerenja tijekom vremena. Kao što je već natuknuto, dobiveni je model iskazan u obličju automata iz *teorije izračunavanja* (engl. *theory of computation*). Pri tom se podrazumijeva uporaba računarskog *zaključivanja* (engl. *inference*), u kojem se gradba modela mora zasnivati isključivo na primljenim podacima, bez inteligentne intervencije i pomoći sa strane u vidu "dodatnih informacija", te naravno, bez poznavanja jednadžbi gibanja sustava.

Dakle, teorijsko-računarski alat ovdje ne služi za rješavanje matematičkim formalizmom zadanih problema — kao što je to u standardnom pristupu, razmatranom u odjeljku 1.1 — već se *ab initio* razmišlja kroz entitete računarskih teorija. Ne započinje se razmatranje tvorbom ili odabirom teorije u okviru koje ćemo postaviti jednadžbe gibanja, već se nastoji da rezultati mjerenja sami grade model sustava. Vjernim odražavanjem uzoraka i struktura pronađenih u primljenim podacima, oni će jednako vjerno oslikavati strukturu promatranog sustava i njegovih stanja.

1.3.1 Izranjanje strukturalne složenosti

Pored glavne zadaće da se rekonstruira računski model koji će biti minimalan glede broja svojih stanja, kao drugi, jednako važan, rezultat modeliranja u okviru izračunske mehanike slijede i statističko-računska svojstva dobivenog modela, a time i promatranog sustava. Kao što smo već naveli, iz dobivenog računskog ustroja proizlazi i računska složenost, odnosno računska sposobnost modela. Primjerice, iz poznatog računskog ustroja doznajemo kakve nizove model može prihvatiti, je li u stanju provjeravati parnost nekih primljenih znakova, ili pak — izravno ili neizravno — izračunati broj π na određeni broj binarnih razlomljenih mjesta.

Odmah postavljamo i pitanje zašto bi takav pristup bio bolji od konciznog matematičkog opisa sustava? Baš zato što matematička formulacija procesa kroz svoju apstrahiranost može skrivati kompleksnost dinamike sustava. Na primjer, diferencijalna jednadžba

$$f'(x) = \rho x\,, \tag{1.1}$$

s potpuno determinističkim rješenjem (eksponencijalni razvoj), "jednostavnim" dodavanjem nelinearnog člana $-(1+\rho)x^2$ prelazi u logističku diferencijalnu jednadžbu:[2]

$$f'(x) = \rho x - (1+\rho)x^2\,, \quad \rho \in [-1, 3]\,. \tag{1.2}$$

Iako jednostavna u svojoj matematičkoj formi, ova nelinearna diferencijalna jednadžba prvog reda skriva u sebi veliko bogatstvo nelinearnih dinamičkih sustava. Njezina je iterativna forma, poznata kao logističko preslikavanje [vidjeti pogl. 2, izraze (2.1, 2.6)], analitički nerješiva. Promjenom njenog kontrolnog parametra (ρ, odnosno r) dobiva se širok raspon sustava — od relativno jednostavnih do ekstremno složenih, od periodičkih do potpuno kvazistohastičkih. Drugim riječima, osim ponašanja sličnog onom za linearne sustave, iz naoko jednostavne jednadžbe gibanja za pojedine iznose njenog parametra izranja neočekivana strukturalna složenost, koja se potom opet razbija u kvazistohastičnosti determinističkog kaosa.

Ove činjenice podstiču na nalaženje novih metoda u istraživanju dinamike nelinearnih sustava. Nužno je da o epistemološkoj utemeljenosti eksperimentalnih rezultata s jedne strane, i o pojavi kvalitativno nove semantike tijekom razvoja sustava s druge strane, vodimo računa na samom početku modeliranja. Time se na razini višoj od puke statističke analize približavamo razumijevanju inherentne "inteligencije sustava", odnosno objašnjenju kvalitativno novih pojava u nelinearnim sustavima.

[2] Veza ρ s parametrom nelinearnosti r u logističkoj funkciji (2.6) jest $\rho = r - 1$.

1.3.2 Nalaženje računskog ustroja sustava

Jedna od posebnosti izučavanja nelinearnih dinamičkih sustava jest u tome što temeljne znanosti nisu razvile metodologije za sagledavanje svih oblika njihove pojavnosti. Naime, standardni fizikalni aparat pripravljen je bilo za:

1. izučavanje potpunog reda, koji se ogleda npr. u čvrstim točkama ili periodičkim orbitama (što općenito odgovara *ravnotežnom stanju* sustava), bilo za
2. potpunu stohastičnost sustava, opisanu termodinamičkim veličinama, kao što su temperatura i entropija.[3]

Dakle, fizika ne raspolaže razvijenim teorijama za opis prirodnih struktura i pojava koje su tipične za dinamičke sustave[10], i koje opisuju pojmovi kao što su *izranjanje strukture*, *pojava uzoraka* i *samoorganizacija*.

Otuda se za izučavanje ovih pojava nameće uporaba koncepata iz teorije izračunavanja, tj. specifično iz teorije *diskretnih strojeva* i *automata*. Izvorno ovi strojevi provjeravaju je li primljeni niz znakova u skladu s gramatikom (strukturom) nekog jezika, odnosno na prihvatljivim nizovima obavljaju određene radnje. Stoga ih možemo iskoristiti da u primljenim simbolički kodiranim mjerenjima iznalaze strukturu procesa. Karakter ovih strojeva je, dakako, diskretan, i u pogledu prezentacije ulaznih veličina i u pogledu modela koji oni predstavljaju. Stoga u ovakvom opisu procesa odstupamo od klasične idealizacije vremensko-prostornog kontinuuma. Naglasimo odmah da je spomenuta idealizacija ionako samo fikcija klasičnog determinizma, koja je u suprotnosti s eksperimentalnom realnošću. Pri mjerenju možemo dobiti tek vremenski i prostorno diskretni niz mjerenja i — ništa više! [4]

Prema tome, diskretni formalni pristup nije tek manje vrijedna aproksimacija, već odraz diskretnog karaktera svakog eksperimenta. Rezultat mjerenja realnog eksperimenta uvijek je predstavljen u obličju niza diskretnih simbola. Digitalni mjerni instrumenti dobar su primjer za izravnu diskretizaciju, koja se kod analognih instrumenata provodi kod očitanja vrijednosti. Po tome se podaci koje primamo kao rezultat mjerenja nekog dinamičkog sustava ne razlikuju od nizova simbola ili riječi formalnih jezika, koji su predmet analize teorije izračunavanja. Mada se radi o računskom opisu sustava, s pomoću discipline koja je donedavno bila rezervirana samo za računarsku i informatičku primjenu, to ne znači da je dobiveni model inferioran.

[3] Novi, sintetički pristup u opisu strukture, J.P. Crutchfield u svojim radovima opisuje naslovom "Beyond a Clock and a Coin Flip". Nelinearni dinamički sustavi u sebi sadrže pravilnosti slične determinizmu sata, ali i stohastičke elemente tipičnog slučajnog procesa kao što je bacanje novčića.

[4] O diskretizaciji mjerenja bit će govora u poglavljima koja slijede (vidjeti npr. 2.3.1 i 4.1).

Radi se upravo o prirodnoj i nužnoj promjeni pristupa, nametnutoj od osobitosti nelinearnog sustava i od našeg zahtjeva da proniknemo u njegovu strukturu.[5]

Dakle, kao što smo već napomenuli u prethodnom odjeljku, iz niza diskretnih simbola primljenih od nekog sustava nastojimo rekonstruirati stroj koji će svojim računskim ustrojem najbolje reproducirati strukturu sadržanu u primljenom nizu i poslužiti kao generalni model za predviđanje budućih stanja sustava. Taj stroj treba biti najbolja aproksimacija promatranog procesa ili, preciznije rečeno, računarskog ustroja tog procesa koji se odražava u mogućnosti odašiljanja (procesiranja) podataka određene kompleksnosti. Pri tom se stupanj aproksimacije veže uz razred stroja (automata) i njegovu veličinu (npr. broj stanja, procesnu moć te veličinu i organizaciju memorije) potrebnu za reprodukciju modela iz podataka određene točnosti (vidjeti također sljedeći pododjeljak).

Struktura procesa bit će iščitana nalaženjem *uzoraka* u primljenim podacima i izražena kroz formu računskih automata i strojeva različitih razreda. Npr. proces može biti opisan jednostavnim konačnim automatom ili, ako se radi o izrazito kompleksnom sustavu, s pomoću univerzalnog Turingovog stroja. Možda će ustroju procesa odgovarati automat sa stogovnom (LIFO) ili repovnom (FIFO) strukturom memorije, ograničene ili proizvoljne veličine. Model opisan takvim strojem omogućuje razumijevanje promatranog procesa, tj. poznavanje njegove semantike, te opis i kvantifikaciju njegove strukture. Njime se u načelu ostvaruje određeni stupanj *sažimanja podataka*, jer je veličina dobrog modela uvijek manja od veličine primljenih podataka. Stoga takav model predstavlja jedan od koraka — koji može biti i konačan — u deduktivnom izvodu teorije promatranog sustava.

1.3.3 Teorija ϵ-strojeva

Modeliranje na temelju vremenskog niza zasniva se na odgovarajuće kodiranim podacima, odaslanim iz mjernog instrumenta koji kvantificira određenu varijablu sustava. Instrument je važan ne samo glede točnosti mjerenja kojeg obavlja već i glede prikladnosti da se njime relevantno izmjeri dinamika promatranog sustava (usporediti s primjerom iznesenim u 1.2.2). Uz točnost instrumenta označenu realnim pozitivnim parametrom ε, računski modeli koje gradimo inherentno nose pogrešku koja se ne može spustiti ispod te vrijednosti. To je razlog da se stvoreni modeli nazovu

[5] Ovdje se može uočiti analogija s revolucionarnom promjenom pristupa koja se zbila pri prelasku iz klasične u kvantnu mehaniku. Novi, kvantni pristup bio je diktiran dubokim teorijsko-eksperimentalnim razlozima, nametnutima zbog posebnog karaktera mikrosvijeta.

ϵ-strojevi[10]. Gledano kvalitativno, parametar ε ima i dodatnu ulogu podsjetnika na upravo izrečenu činjenicu — instrument mada i jako precizan, ali neodgovarajući strukturalnoj suštini sustava, vodi na primitak neadekvatnog vremenskog niza, koji će dati iskrivljenu sliku sustava.

Rekonstrukcija ϵ-strojeva iterativan je proces. Počinje se od samog vremenskog niza kao najniže razine modela, koji predstavlja samoga sebe. Vremenski niz jest fotografski preslikana prošlost sustava, razine preciznosti jednake ε, bez dodatnih aproksimacija. Naravno, bez sažimanja i bez mogućnosti predviđanja to i nije model, već samo početna razina iterativnog procesa gradbe ϵ-strojeva.

Sljedeću, prvu razinu modela predstavlja *raščlambeno stablo* (engl. *parse tree*), koje nastaje raščlambom podnizova konstantne duljine, jednake visini stabla.

Nadalje, na drugoj razini modela, u raščlambenom stablu nalazimo podstabla određene visine koja imaju međusobno različitu strukturu (morfologiju), odnosno različitu topologiju svojih čvorova. Svako od tih podstabala predstavlja pojedino stanje (deterministčkog) konačnog automata. U višoj aproksimaciji ta podstabla možemo razlikovati i po uvjetnim vjerojatnostima njihovih čvorova. Prijelazi između korijena ovih podstabala i vjerojatnosti tih prijelaza definiraju model sustava u formi stohastički nadopunjenog konačnog automata, kojeg nazivamo *stohastički konačni automat* (engl. *stochastic finite automaton*, SFA). On predstavlja osnovni kauzalni model sustava, odnosno temeljni ϵ-stroj. SFA reproducira primljeni vremenski niz do uključivo duljine izlučenih riječi *stohastički točno*, do na korištenu vjerojatnosnu preciznost δ. Njegove će teorijske postavke biti izložene u poglavlju 3, a algoritam nalaženja u poglavlju 4. Ostvarenje programske podrške za nalaženje SFA iz vremenskog niza glavni je predmet ovog rada (pogl. 5).

Stohastički konačni automati jednako su uspješni za prikaz i periodičkog i kaotičnog ponašanja. Drugim riječima, računski ustroji dinamičkih sustava u oba ova režima odgovaraju relativno jednostavnoj razini konačnih automata. Međutim, za neke kritične vrijednosti nelinearnog parametra sustava u područjima na "putu prema kaosu" uočeno je da rast preciznosti SFA-modela (u ovom slučaju rast visine podstabala, odnosno morfova) vodi na divergenciju broja njegovih stanja[13]. To je posljedica povećanja računarske kompleksnosti sustava. Sustave u kojima se to događa možemo — po analogiji s prirodnim fenomenima — opisati kao fazne prijelaze; npr. prijelaz iz uređenog, čvrstog stanja u kaotično, plinovito stanje. Za njihov je opis potreban *inovacijski korak*, u kojem se uvodi ϵ-stroj više razine. On je predstavljen automatom višeg razreda s memorijom proizvoljne veličine, koji daje

konačan model sustava. Njegova se struktura nalazi analizom pravilnosti i simetrija u sve preciznijim SFA, dobivenim s morfovima sve veće visine.

1.3.4 Statistička složenost

Jedan od najintrigantnijih koncepata predložen u radovima J. P. Crutchfielda iz domene je teorije informacija i složenosti sustava, a proizlazi kao rezultat modeliranja s pomoću ϵ-strojeva[1, 9]. Temeljni ϵ-stroj predstavljen je s pomoću stohastičkog konačnog automata, kauzalnog modela koji daje vjerojatnosti prijelaza iz svakog svojeg (kauzalnog) stanja u moguća buduća stanja, po odašiljanju, odnosno primitku određenog simbola. Vjerojatnosti prijelaza u buduća stanja ne ovise o prethodnim (prošlim) stanjima, već samo o trenutnom (sadašnjem), uzročnom stanju. Zato se prijelazi između stanja SFA mogu promatrati kao pozadinski Markovljev proces. Stacionarna distribucija tog procesa jest ona distribucija vjerojatnosti stanja modela koja je invarijantna na djelovanje matrice prijelaza tog procesa. Entropija te stacionarne distribucije nazvana je *statistička složenost* i ona je predložena kao mjera strukturalne složenosti SFA-modela, odnosno sustava kojeg taj model simulira (odjeljak 4.2 i pododjeljak 4.3.6).

Npr. ako je sustav periodičan s periodom n, po nalaženju faze svi su prijelazi između n ponavljajućih stanja deterministički, s vjerojatnošću jedan. Stoga je stacionarna distribucija uniformna, a statistička složenost sustava jednaka je logaritmu od n. S druge strane, sustav u potpuno kaotičnom stanju — kakvog npr. predstavlja nasumično bacanje novčića — emitira Bernoullijev niz nula i jedinica. Pripadni temeljni ϵ-stroj tada je SFA sa samo jednim stanjem, s jednakim izgledima za pojavu nule i jedinice. Zbog jednog stanja entropija pripadne distribucije i složenost modela su nula. I intuitivno je jasno da sustav u potpuno kaotičnom režimu nije strukturalno složen. On je jednostavno (kvazi)stohastičan.

1.4 Objektno orijentirana programska podrška

Već je u prethodnom izlaganju istaknuta primjenljivost fundamentalnih računarskih teorija na prirodne pa i na sve ostale fenomene. Po našem mišljenju i iskustvu još jedno od postignuća računarstva ima predispozicije da posluži kao paradigma šireg istraživačkog pristupa. To je objektno orijentirano programiranje. Upravo je objektni pristup taj koji je u programiranje unio novu stegu glede emuliranja realnosti problema u svim njegovim entitetima, atributima i radnjama — kojima

redom pridružujemo klase, članske varijable i funkcije — te tako i sam postupak izrade programa učinio vrlo sličnim postupku modeliranja. Stoga je ova programska paradigma pravi izbor za opis kako apstraktnih, matematički-formuliranih, tako i realnih fizikalnih sustava, procesa i objekata.

Težište praktičnog dijela ovog rada jest na stvaranju programske osnove za nalaženje i analizu stohastičkih konačnih automata iz binarnog vremenskog niza. Relativno velik trud uložen u dizajniranje i implementiranje našeg programa bio bi daleko manje vrijedan da program nije u potpunosti izrađen prema uzorima objektno orijentiranog programiranja. Kreirane klase predstavljaju čvrste i neovisne gradbene blokove, prikladne za uporabu, ali i pogodne za nadogradnju i eventualno preoblikovanje. Razvoj programa zahtijevao je implementaciju raznolikih i specifičnih struktura podataka te relativno složenih algoritama. Sâm problem je hijerarhijski složen, u smislu da konačni algoritam u sebi objedinjuje reference na većinu razvijenih klasa te na mnoge dodatne usađene klase nužne za ostvarenje potrebnih struktura podataka.

Implementacija korisnički prijateljskog grafičkog sučelja — iako vremenski vrlo zahtjevna i na prvi pogled nepotrebna za ostvarenje znanstvene zadaće ovog rada — pokazala se kao izuzetno korisna. Tako je sučelje za prikaz proizvoljnog dijela glavnog raščlambenog stabla i njegovih podstabala omogućilo pregledno testiranje svih algoritama. Ova dodatna, vizualna dimenzija programa presudna je za uspješnu humanu analizu onog što se zaista događa u procesu modeliranja. To nam je omogućilo da već u samom startu uporabe programa uočimo pojave koje izvorni autor teorije ili nije uočio, ili nije eksplicirao u svojim primjerima (usporediti npr. rezultate iz [1, 9, 10] s našim rezultatima iznesenim u poglavlju 6).

Poglavlje 2

Teorijska polazišta

Zadaća je ovog poglavlja pregled teorijskih osnova te formalizma koji se koristi u idućim poglavljima. Navedeni formalizam bit će karakterističan za cijelo područje *modeliranja iz vremenskog niza*. Definiciju ove discipline možemo sažeti u sljedećoj rečenici:

Definicija 2.1 *Cilj modeliranja iz vremenskog niza jest nalaženje modela sustava isključivo na temelju niza simbola, primljenih kao rezultata izmjere tog sustava s pomoću određenog mjernog instrumenta.*

Pri tom mjerni instrument odražava vrijednosti varijabli stanja i načelno nikoji drugi parametri sustava nisu poznati. Jasno je međutim, da u stvaranju programske podrške za takvo modeliranje, pored odgovarajućeg programskog sučelja za prihvat zadanih simbola, moramo priskrbiti i:

- numeričku simulaciju razvoja dinamičkih sustava, te
- simulaciju odgovarajućeg mjernog kanala.

Simulacija dinamičkog sustava podrazumijeva numeričku iteraciju varijabli stanja. Vrijednosti varijabli stanja se potom posredstvom "mjerenja određenim instrumentom" prevode u simbolički niz koji će predstavljati konačni ulaz za modeliranje. Prema tome, oponašanje mjernog kanala svodi se na uvođenje određene particije faznog prostora (prostora vrijednosti varijabli stanja) koja definira kodiranje, u smislu da se za vrijednosti iz određenog podskupa definiranog particijom, varijabli pridruži određeni simbol.

U našem razmatranju krećemo upravo od simulacije dinamičkih sustava i mjernog kanala, povezujući ove pojmove s iterativnim procesima na faznom prostoru, odnosno sa simboličkom dinamikom. Iako je veza između ovih pojmova intuitivno lako dokučiva, utemeljenje ključnih pretpostavki na matematičkom formalizmu doprinijet će jasnoći i preciznosti daljnjeg izlaganja.

2.1 Iterativni procesi

Bitno pojednostavljenje formalizma diferencijalnih jednadžbi ostvaruje se uvođenjem jednadžbi diferencija, što je u suštini ekvivalentno prelasku na diskretnu vremensku varijablu. U tom slučaju vremenski trenuci postaju elementi iz skupa cijelih nenegativnih brojeva $\mathbb{N}_0^+$, i označavamo ih jednostavno indeksima. Sada se jednadžbe gibanja mogu izraziti u iterativnoj formi, odnosno kao *iterativni procesi*.

Definicija 2.2 *Za iterativni proces, vrijednost nove točke x_{n+1} u trajektoriji računamo na temelju vrijednosti prethodne točke x_n kao*

$$x_{n+1} = F\left(x_n\right) \ , \ n \in \mathbb{N}_0^+ \ . \tag{2.1}$$

Formula je izravno primjenljiva na digitalnim računalima. Indeks n tu predstavlja diskretni vremenski trenutak. Početnu točku ili *sjeme* iteracije označit ćemo s x_0, a n-tu iteraciju s F^n tako da vrijedi

$$x_n = F^n\left(x_0\right) \ . \tag{2.2}$$

F^0 kao nulta iteracija, predstavlja funkciju identiteta $F^0\left(x\right) = x$. Numeričko rješavanje diferencijalnih jednadžbi je u biti ekvivalentno ovom pristupu, uz uobičajene sofistikacije koje nalazimo kod naprednijih algoritama (npr. prilagodba veličine vremenskog intervala prema potrebnoj točnosti). Prikaz dinamičkih sustava preko iterativnih procesa se uvriježio zbog svoje jednostavnosti i preglednosti, pogotovo kod jednodimenzionalnih sustava. Pri tom ne postoji bitan gubitak općenitosti pa ćemo nadalje koristiti ovaj način označavanja jednadžbi gibanja.

2.2 Simbolička dinamika

Kao što je već napomenuto u uvodnom poglavlju, idući korak u apstrakciji modeliranja dinamičkih sustava je diskretizacija i same varijable stanja. Kao logični

nadomjestak praćenja točne trajektorije sustava uvodi se pojam *putopisa* (engl. *itinerary*), koji je u uskoj svezi s *particijom* faznog prostora. U ovisnosti o tome kojem podskupu particije pripada dana točka, pridružujemo joj odgovarajući simbol. Npr. ako točka trajektorije pripada lijevom (desnom) dijelu promatrane domene, u putopis zapisujemo simbol 0 (odnosno 1). Jasno je da, u ovisnosti o karakteru problema, možemo koristiti proizvoljne odrednice za opis podskupova (dijelova) domene.

2.2.1 Nizovni prostor

Definicija 2.3 *Neka je* $\mathcal{A}$ *abeceda koja sadrži* $a = \mathrm{card}(\mathcal{A})$ *simbola. Skup svih beskonačnih nizova sastavljenih od simbola abecede* $\mathcal{A}$ *nazivat ćemo nizovni prostor (engl. sequence space) i označavati sa* Σ_∞ *:*

$$\begin{aligned}\Sigma_\infty &= \{\ (s_0, s_1, s_2\,, \dots\ )\ :\ s_i \in \{0, 1, \dots\,, a-1\}\ \} \\ &= \{0, 1, \dots\,, a-1\} \times \{0, 1, \dots\,, a-1\} \times \ \dots\ \dots \\ &= \{0, 1, \dots\,, a-1\}^\infty\,.\end{aligned} \tag{2.3}$$

2.2.2 Putopis

Definicija 2.4 *Putopis* $\mathbf{s}$ *je element nizovnog prostora,* $\mathbf{s} \in \Sigma_\infty$ *, a često se jednostavno naziva točkom iz tog prostora. Drugim riječima, radi se o beskonačno dugačkom nizu simbola iz* $\mathcal{A}$ *.*

Da ilustriramo ideju, odaberimo elementarnu dvočlanu abecedu $\mathcal{A} = \{0, 1\}$. To je abeceda koju ćemo koristiti u modeliranju, a ujedno je i najčešći izbor u praksi. Također, radi jednostavnosti, ograničimo trenutno razmatranje na 1-D (jednodimenzionalni) sustav i promatrajmo domenu $I \subseteq \mathbb{R}$ tog sustava. Neka je A_1 skup svih onih točaka koje napuštaju interval I nakon jedne iteracije. Tada je $\Lambda = I - A_1$ skup koji se preslikava u sebe sâmog, i njegovo postojanje je preduvjet za pojavu i periodičkih i općenito nedivergentnih orbita. Za funkcije F s "grbom", kao uobičajene kandidate za nelinearnu dinamiku, skup A_1 ako je neprazan, predstavlja upravo područje oko kritične točke (vrha ili dna grbe) koje izlazi iz intervala I. Prema tome A_1 dijeli interval I na dva podintervala, lijevi i desni, koje ćemo označiti kao Λ_0, odnosno Λ_1. Odatle se prirodno nameće definicija putopisa od sjemene točke x_0

kao beskonačnog binarnog niza:

$$\begin{aligned} \mathcal{S} &: \quad \Lambda \to \Sigma_\infty\,, \quad \mathcal{S} : x_0 \mapsto \mathbf{s}\,; \\ \mathbf{s} &= (s_0, s_1, s_2, \dots),\ ili\ kraće:\ \mathbf{s} = s_0 s_1 s_2 \dots\ ;\ \ s_i \in \{0,1\}\,; \\ s_n &= \begin{cases} 0\,, & F^n(x_0) \in \Lambda_0 \\ 1\,, & F^n(x_0) \in \Lambda_1 \end{cases} . \end{aligned} \tag{2.4}$$

Funkcija $\mathcal{S}$ je u bitnom definirana upravo particijom $\mathcal{P}_M$ domene I na disjunktne podskupove Λ_0 i Λ_1,

$$\mathcal{P}_M = \{\Lambda_0,\, \Lambda_1\}\,. \tag{2.5}$$

Particiju $\mathcal{P}_M$ ćemo zvati *mjernom particijom*, i ona u općenitom slučaju ne mora biti potpuna. Tj. $A_1 = I - (\Lambda_0 \cup \Lambda_1)$ je općenito neprazan skup, kao što se može vidjeti u niže izloženom primjeru za kvadratnu porodicu funkcija.

U općenitom slučaju višedimenzionalnih sustava, uvođenjem odgovarajuće *Poincaréove plohe* i *preslikavanja prvog povratka* (engl. termini su i *surface of section*, odnosno *first return map*), problem reduciramo na dvodimenzionalni, odnosno jednodimenzionalni, a daljnjom pogodnom diskretizacijom možemo ga prevesti u simboličku dinamiku kao krajnje apstrahirani dinamički sustav.[1]

Za ilustraciju, primijetimo da je kod logističke funkcije, $F(x) = f_r(x)$,

$$f_r(x) = rx(1-x)\,,\ x \in [0,1]\,,\ r \in [0,4] \tag{2.6}$$

skup $A_1 = \emptyset$. Restrikcija parametra r na interval $[0,4]$ nužna je da se zadrži svojstvo zatvorenosti u grupi definiranoj operacijom iteracije f_r i intervalom $I = [0,1]$. Za te je vrijednosti kodomena funkcije f_r u potpunosti unutar intervala $[0,1]$. Vrijedi, dakle, da je skup $\Lambda = I = [0,1]$. Najprirodnije je kao točku diskriminacije uzeti upravo kritičnu točku $x_c = 1/2$, određenu s $f'(x_c) = 0$. Odatle dobivamo particiju koja je potpuna:

$$\begin{aligned} \mathcal{P}_M &= \{\,[0,1/2)\,,\ [1/2,1]\,\}\,, \\ \Lambda_0 &\cup\, \Lambda_1 = I\,. \end{aligned} \tag{2.7}$$

[1] Simbolička dinamika je dosad uglavnom primjenjivana na jednodimenzionalnim sustavima. Vjerojatno je razlog tome lako uvođenje diskretizacije, s najčešće vrlo ilustrativnom interpretacijom iste.

Za slučaj logističke jednadžbe slijedi funkcija za tvorbu putopisa, $\mathcal{S}(x_0) = s_0 s_1 s_2 \dots$, opisana sa

$$\begin{aligned} \mathcal{S} &: [0,1] \to \Sigma_\infty \,, \\ s_n &= \begin{cases} 0\,, & F^n(x_0) \in [0, 1/2) \\ 1\,, & F^n(x_0) \in [1/2, 1] \end{cases} . \end{aligned} \tag{2.8}$$

Kao sljedeći primjer, promotrimo funkcije iz kvadratne porodice $Q_c(x) = x^2 + c$, uz $x \in I = [-2, 2]$. Ovdje je za općenitu vrijednost parametra c skup A_1 neprazan. Međutim, u praksi numeričkog pristupa je ponovno dovoljno uzeti particiju domene naspram kritične točke, koja je sada $x_c = 0$. Naime trajektorije za sve točke iz skupa A_1 ionako divergiraju u beskonačnost, tj. nisu zanimljive. Lako se može pokazati da je zanimljivo područje parametra c ograničeno na $-2 \leq c \leq 1/4$, gdje vrijedi da je A_1 prazan, tj. svakoj točki iz I ćemo moći pridružiti element iz skupa Σ_∞ te kreirati putopis[3].

2.2.3 Posmačno preslikavanje

Preskačući detalje formalnog pristupa simboličkoj dinamici (vidjeti npr. [2]) spominjemo ovdje *posmačno preslikavanje* (engl. *shift map*). Vrijedi sljedeća definicija:

Definicija 2.5 *Posmačno preslikavanje, ili kratko, funkcija posmaka* $\sigma : \Sigma_\infty \to \Sigma_\infty$, *djeluje na niz simbola tako da uklanja početni simbol, a sve ostale simbole posmiče za jedno mjesto u lijevo:*

$$\sigma(s_0 s_1 s_2 \dots) = s_1 s_2 s_3 \ \dots \ .$$

Otuda slijedi da je k-struko djelovanje posmaka opisano sa

$$\sigma^k(s_0 s_1 s_2 \dots) = s_k s_{k+1} s_{k+2} \dots \,, \ k \in \mathbb{N}_0^+ .$$

Za identitetni posmak na $\mathbf{s} \in \Sigma_\infty$, pišemo $\sigma^0(\mathbf{s}) = \mathbf{s}$. Jasno je da se radi o iterativnom procesu na skupu Σ_∞, u smislu prethodne definicije 2.2. Za vremenski niz sa sjemenom točkom x_0, koja općenito odgovara diskretnom vremenskom trenutku i_0, σ^k posmiče ishodišnu točku putopisa na vremenski trenutak i_k .

Nalaženje periodičkih točaka za funkciju σ je krajnje jednostavno. Ukoliko $\mathbf{s} \in \Sigma_\infty$ predstavlja periodičku točku perioda n, tada je nužno sljedećeg oblika:

$$\mathbf{s} = s_0 s_1 \dots \, s_{n-1} s_0 s_1 \dots \, s_{n-1} \dots \ = \overline{s_0 s_1 \dots \, s_{n-1}} \,,$$

tj. mora vrijediti:

$$\sigma^n(\mathbf{s}) = \mathbf{s}\,. \tag{2.9}$$

Ako se radi o eventualno periodičkoj točki koja nakon k-iteracija prelazi u periodičku točku perioda n, tada vrijedi:

$$\begin{gathered}\mathbf{s} = s_0 s_1 \ldots\ s_{k-1}\,\overline{s_k s_{k+1} \ldots\ s_{k+n-1}}\,, \\ \sigma^n(\sigma^k \mathbf{s}) = \mathbf{s}\,.\end{gathered} \tag{2.10}$$

Za σ postoje dvije čvrste točke, odnosno točke s periodom 1: $\mathbf{s}_{10} = 0000\ldots$ i $\mathbf{s}_{11} = 1111\ldots$. Također, postoje dvije točke perioda 2, $\mathbf{s}_{20} = 010101\ldots$ i $\mathbf{s}_{21} = 101010\ldots$, za koje vrijedi

$$\sigma(\mathbf{s}_{20}) = \mathbf{s}_{21}\,,\ \sigma(\mathbf{s}_{21}) = \mathbf{s}_{20}\,.$$

Točaka s periodom 3 ima ukupno šest, što se pregledno može prikazati u obliku dva period-3 ciklusa:

$$\begin{aligned}\overline{001} &\ \mapsto\ \overline{010} \mapsto \overline{100} \mapsto \overline{001}\,, \\ \overline{011} &\ \mapsto\ \overline{110} \mapsto \overline{101} \mapsto \overline{011}\,.\end{aligned}$$

Za posmačno preslikavanje $\sigma : \Sigma_\infty \to \Sigma_\infty$, važno je istaknuti svojstvo kontinuiranosti. Uz udaljenost d dviju točaka $\mathbf{s},\ \mathbf{t} \in \Sigma_\infty$, definiranu kao

$$d[\mathbf{s}, \mathbf{t}] = \sum_{i=0}^{\infty} \frac{|s_i - t_i|}{2^i}\,, \tag{2.11}$$

definicija kontinuiranosti se svodi na mogućnost pronalaženja po volji bliskih slika funkcije σ. Konkretno, funkcija je kontinuirana u točki $\mathbf{s} \in \Sigma_\infty$ ako za svaki $\epsilon > 0$ postoji $\delta > 0$, takav da iz $d[\mathbf{t}, \mathbf{s}] < \boldsymbol{\delta}$ slijedi $d\,[\sigma(\mathbf{t}), \boldsymbol{\sigma}(\mathbf{s})] < \epsilon$. Za posmak σ se pokazuje da je kontinuiran u svim točkama iz Σ_∞. Ovo je svojstvo nužno za uvođenje korespondencije između posmaka i funkcija na faznom prostoru koje su uglavnom kontinuirane. Primijetimo da je nužan preduvjet kontinuiranosti σ beskonačnost nizova u Σ_∞.

Uočljiva je lakoća nalaženja periodičkih točaka i ciklusa u simboličkoj dinamici, nezamisliva za analizu nelinearnih funkcija zadanih na faznom prostoru. Uvođenje simboličke dinamike s tog je gledišta opravdano, jer će proučavanje dinamičkih sustava biti apstrahirano do najveće moguće jednostavnosti.

2.2.4 Srodnost s dinamikom faznog prostora

Iako različiti u svojoj pojavnosti, prostori Λ i Σ_∞ su kvalitativno ekvivalentni, odnosno *homeomorfni*. Rigorozni pristup čitalac može pronaći u matematičkoj literaturi,[2] a zadatak ovog pododjeljka je tek da uspostavi jasnu vezu s formalizmom simboličke dinamike. Stoga samo ukratko podsjećamo da su dva skupa homeomorfna ako između njih postoji *homeomorfizam*, tj. preslikavanje koje je *jedan naprama jedan*, *na* (dakle bijekcija), te *kontinuirano*, s inverznom funkcijom koja je također kontinuirana. Gore izložena funkcija $\mathcal{S}$ je primjer tog homeomorfizma, gdje se rezultati djelovanja funkcije $F(x)$ na Λ, prevode u rezultat djelovanja posmaka $\sigma(\mathbf{s})$ na Σ_∞, i to vrijedi za većinu "normalnih" funkcija F na faznom prostoru.

Uočimo još jedno važno svojstvo funkcije $\mathcal{S}$. Neka je $x \in \Lambda$, tada vrijedi

$$\mathcal{S} \circ F = \sigma \circ \mathcal{S} \,. \tag{2.12}$$

Ovo svojstvo jednostavno proizlazi iz sljedećeg razmatranja. Neka je $\mathcal{S}(x) = s_0 s_1 s_2 \ldots$, tada vrijedi da sjeme x trajektorije mora pripadati skupu Λ_{s_0}, a općenita iduća i-ta iteracija skupu Λ_i, uz $i = s_i$. Analogno će vrijediti za trajektoriju koja započinje u sljedećoj točki $F(x)$, što možemo sumarno zapisati kao:

$$\begin{array}{ccc} F^0(x) \in \Lambda_{s_0} & ; & F^1(x) \in \Lambda_{s_1} \\ F^1(x) \in \Lambda_{s_1} & ; & F^2(x) \in \Lambda_{s_2} \\ \vdots & & \vdots \\ F^n(x) \in \Lambda_{s_n} & ; & F^{n+1}(x) \in \Lambda_{s_{n+1}} \end{array}$$

Druga kolona predstavlja upravo putopis razvijen iz sljedeće točke iteracije $F(x)$, tj. vrijedi da je:

$$\mathcal{S}(F(x)) = s_1 s_2 s_3 \ldots = \sigma(s_0 s_1 s_2 \ldots) = \sigma(\mathcal{S}(x)) \,,$$

čime je gornja tvrdnja dokazana.

Pregledno se koncept sparenosti funkcija, odnosno homeomorfnosti prostora, prikazuje u obličju komutativnog dijagrama:

$$\begin{array}{ccc} \Lambda & \xrightarrow{F} & \Lambda \\ \mathcal{S} \downarrow & & \downarrow \mathcal{S} \\ \Sigma_\infty & \xrightarrow{\sigma} & \Sigma_\infty \end{array} .$$

[2] Npr. za kvadratnu porodicu funkcija dokaz homeomorfnosti je u [2]. Za često navođenu logističku jednadžbu, zbog ograničenosti njenog intervala na [0,1], može se uočiti da će dokaz biti jednostavnije naravi, no njegov prikaz izlazi iz okvira ovog rada.

Pošto je funkcija $\mathcal{S}$ homeomorfizam, ima svojstva bijekcije, a otuda slijedi i postojanje inverzne funkcije $\mathcal{S}^{-1}$. Stoga postoji i "inverzni" komutativni dijagram:

$$\begin{array}{ccc} \Lambda & \underrightarrow{F} & \Lambda \\ \mathcal{S}^{-1}\uparrow & & \uparrow\mathcal{S}^{-1} \\ \Sigma_\infty & \overrightarrow{\sigma} & \Sigma_\infty \end{array} .$$

Tvrdnja inverznog komutativnog dijagrama je

$$\begin{aligned} F\circ\mathcal{S}^{-1} &= \mathcal{S}^{-1}\circ\mathcal{S}\circ F\circ\mathcal{S}^{-1} \\ &= \mathcal{S}^{-1}\circ\sigma\circ\mathcal{S}\circ\mathcal{S}^{-1} \\ &= \mathcal{S}^{-1}\circ\sigma\,. \end{aligned} \tag{2.13}$$

Konačno, promotrimo djelovanje funkcije $\mathcal{S}$ na proizvoljnu n-tu točku F-orbite:

$$\begin{aligned} \mathcal{S}\circ F^n &= \mathcal{S}\circ F\circ F^{n-1} \\ &= \sigma\circ\mathcal{S}\circ F^{n-1} \\ &= \sigma\circ\mathcal{S}\circ F\circ F^{n-2} \\ &= \sigma^2\circ\mathcal{S}\circ F^{n-2} \\ &\;\;\vdots \\ &= \sigma^n\circ\mathcal{S}\,. \end{aligned} \tag{2.14}$$

Ako funkcija koja je homeomorfizam ima i upravo obrazloženo svojstvo iskazano izrazom (2.12), tada se ona još naziva i *konjugacija.*[3] Iz gore iznesenog slijedi da je $\mathcal{S}$ konjugacija, a za odgovarajuće funkcije F i σ kažemo da su međusobno *konjugirane* (srodne).

Zaključujemo da funkcija $\mathcal{S}$ pretvara F-orbite u σ-orbite, a funkcija $\mathcal{S}^{-1}$ σ-orbite u F-orbite. Ovaj zaključak će biti posebno važan u našem daljnjem razmatranju, jer se ispostavlja da analizom putopisa dobivamo ispravne zaključke u svezi ponašanja orbita u faznom prostoru. Ako smo zaključili da je neka točka $\mathbf{s}\in\Sigma_\infty$ periodička s obzirom na posmak σ, tada će i točka $x=\mathcal{S}^{-1}(\mathbf{s})$, $x\in\Lambda$, biti periodička s obzirom na iteraciju F. Isto će vrijediti i za eventualno-periodičke ili čvrste točke, tj. općenito postoji potpuna srodnost između dinamike posmaka σ na skupu Σ_∞ i funkcije F na skupu Λ. U bitnom, te dvije dinamike možemo smatrati podudarnima.

[3] Konjugacija u smislu stapanja, sparivanja, srođivanja.

2.2.5 Kaos u simboličkoj dinamici

U pododjeljku 2.2.3 je već navedena kontinuiranost kao jedno od važnih svojstava funkcije posmaka σ. Također, za primjenu simboličke dinamike u proučavanju nelinearnih dinamičkih sustava presudno je istaknuti da je *posmačno preslikavanje* σ *kaotični dinamički sustav.* Iako do danas ne postoji jedinstveni konsenzus oko definitivnih kriterija koji čine dinamički sustav kaotičnim, možemo se poslužiti skupom kriterija kako slijedi (prema [3]):

Definicija 2.6 *Dinamički sustav* ϕ *je kaotičan ako vrijedi:*

1. Periodičke točke od ϕ su guste.

2. ϕ je tranzitivan.

3. ϕ je osjetljiv na početne uvjete.[4]

U letimičnom pregledu podsjećamo na ovdje korištene definicije. Neki podskup je *gust* unutar svojeg nadskupa ako za svaku točku nadskupa postoji točka iz podskupa koja joj je proizvoljno blizu. Dinamički sustav je *tranzitivan* ako unutar okoline točke x određene s $\epsilon > 0$, postoji sjeme z orbite koja se unutar ϵ približava nekoj proizvoljnoj točki y.

Najčešće navođeno svojstvo kaotičnih sustava je svojstvo *osjetljivosti na početne uvjete,* ili kratko, *osjetljivost.* Dinamički sustav je osjetljiv ako za svaki x i za svaki $\epsilon > 0$ postoji y na udaljenosti unutar ϵ od x, i $k \in \mathbb{N}_0^+$ takav da vrijedi da će se k-te iteracije $\phi^k(x)$ i $\phi^k(y)$ udaljiti međusobno za više od nekog proizvoljnog parametra $\kappa > 0$. Drugim riječima, za sjemena udaljena manje od ϵ zahtijevamo divergenciju orbita za više od neke proizvoljne donje granice.

Za sustav $\sigma : \Sigma_\infty \to \Sigma_\infty$ relativno se lako pokazuje da posjeduje sva nabrojena svojstva te da je, kao što smo već istaknuli, kaotičan. Možemo u kratkom neformalnom pregledu pokazati da to vrijedi.

Periodičke točke su guste, jer za bilo koju točku $\mathbf{s} \in \Sigma_\infty$ zadanu sa svojih prvih $n+1$ simbola kao $\mathbf{s} = s_0 s_1 s_2 \ldots s_n$, tako da je $2^{-n} < \epsilon$, možemo kreirati periodičku točku udaljenu od nje za manje od ϵ tako, da jednostavno uzmemo $\mathbf{t} = s_0 s_1 s_2 \ldots s_n \overline{s_0 s_1 s_2 \ldots s_n}$.

[4] Gornja svojstva nisu u potpunosti nezavisna (iako vezu nije trivijalno pokazati), već iz prva dva slijedi treće kao posljedica.

Za dokaz tranzitivnosti koristi se posebna točka $\hat{\mathbf{s}} \in \Sigma_\infty$, koja ima svojstvo da je njezina orbita gusta u Σ_∞. To je točka:

$$\hat{\mathbf{s}} = \underbrace{0\,1}_{l=1}\ \underbrace{00\ 01\ 10\ 11}_{l=2}\ \underbrace{000\ 001\ \ldots}_{l=3}\ \underbrace{0000\ \ldots\ \ldots}_{l=4\quad\ldots}\ \ldots\,,\ \ldots \tag{2.15}$$

u kojoj smo grupirali sve podnizove duljine 1, 2, itd. Sada se, uz sličnu argumentaciju kao i u prethodnom dokazu, pokazuje da orbita točke $\hat{\mathbf{s}}$ prolazi proizvoljno blizu bilo koje druge točke.

Da dokažemo osjetljivost sustava $\sigma : \Sigma_\infty \to \Sigma_\infty$, odaberimo kao parametar divergencije $\kappa = 1$. Pretpostavimo $\mathbf{s}, \mathbf{t} \in \Sigma_\infty$, $\mathbf{s} \neq \mathbf{t}$, i neka je $d[\mathbf{s}, \mathbf{t}] < \epsilon$, uz $\epsilon < 2^{-n}$. Drugim riječima $\mathbf{s}$ i $\mathbf{t}$ se razlikuju, ali tek u znakovima za neki indeks $k > n$, tako da vrijedi $|s_k - t_k| = 1$. Sada je dovoljno primijeniti k iteracija funkcije σ pa da se orbite koje polaze od $\mathbf{s}$ i $\mathbf{t}$ udalje za više ili jednako κ:

$$\begin{aligned} d\left[\sigma^k(\mathbf{t}),\, \boldsymbol{\sigma}^k(\mathbf{s})\right] &= \frac{|s_k - t_k|}{2^0} + \sum_{i=1}^{\infty} \frac{|s_i - t_i|}{2^i} \\ &\geq \frac{|s_k - t_k|}{2^0} = 1\,. \end{aligned}$$

U slučaju proizvoljnog κ moramo jednostavno napraviti $\ulcorner n\kappa \urcorner$ iteracija, i time izneseni dokaz postaje općenit.

Važnost iznesenih postavki zaslužuje njihov kratak sažetak. Po ustanovljenju konjugacije između (uobičajenih) nelinearnih dinamičkih sustava, definiranih funkcijama iteracije na faznom prostoru, i simboličke dinamike na prostoru beskonačnih nizova, o značajkama prvih može se govoriti proučavanjem puno jednostavnijeg slučaja drugih. Egzaktno gledano, ako želimo proučavati neki dinamički sustav nepoznatih svojstava te ustanovimo da postoji konjugacija $\mathcal{S}$ koja prvi prostor čini srodnim drugom, tada je to dokaz da se u bitnom radi o ekvivalentnim sustavima. Konkretno, to je dokaz da je nepoznati sustav kaotičan, jednako kao što je kaotičan posmak na prostoru svih beskonačnih nizova.

2.3 Primjena simboličke dinamike

Pored zadaće navedene u uvodu ovog poglavlja, prethodni i ovaj odjeljak imaju za dodatni cilj premoštenje jaza koji čitalac može primijetiti čitajući, s jedne strane matematičku literaturu, a s druge strane radove koji nastoje primijeniti teoriju u

okruženju eksperimentalne stvarnosti. Naravno, tek "izlazak u stvarni svijet" i primjena razvijanog matematičkog aparata na modeliranju stvarnih problema, predstavlja ultimativnu provjeru i daje smisao onom što radimo. S druge strane, eksperimentalna ograničenja, a također i ograničenja računarskih izvorišta u stvarnom svijetu, zahtijevaju odricanje od mnogih elegantnih matematičkih idealizacija.

Jedna od idealizacija neostvarivih u mjerenjima je i izmjera vremensko-prostornog kontinuuma. Posmačna funkcija σ iz prijašnjeg poglavlja je kontinuirana kad djeluje na prostoru Σ_∞ beskonačno dugačkih nizova. No u praksi ćemo se očito morati zadovoljiti prijamom nizova konačne duljine, i shodno tome, u definiciji kontinuiranosti odustati od proizvoljno malog parametra $\epsilon > 0$. Umjesto toga, iz procesa mjerenja i modeliranja proizići će donja granica za ϵ kao pokazatelj točnosti našeg postupka.

2.3.1 Vremenski niz

U praktičnom pristupu, kad na osnovi zadane iteracijske jednadžbe želimo generirati vremenski niz koji ćemo nadalje proučavati, gornju formalnu definiciju funkcije $\mathcal{S}$ prilagođujemo činjenici da je za izračun sljedeće točke dovoljna ona prethodna, odnosno da n-ti simbol putopisa dobivamo tako da odgovarajuće (binarno) kodiramo točku $x_n \in I$,

$$\begin{aligned} x_n &= F(x_{n-1}) \,,\ n \geq 1 \,; \\ s_n &= \mathcal{B}(x_n) \,. \end{aligned} \tag{2.16}$$

Ovdje je $\mathcal{B}$ funkcija kodiranja koju ćemo nazivati *mjerno kodiranje*, iz očitog razloga što podrazumijevamo da će to biti zadaća mjernog instrumenta. Mjerno kodiranje je u bitnom određeno particijom $\mathcal{P}_M$ domene I na podskupove Λ_0 i Λ_1,

$$\mathcal{P}_M = \{\Lambda_0,\, \Lambda_1\} \,. \tag{2.17}$$

Već je istaknuto da mjerna particija $\mathcal{P}_M$ u općenitom slučaju ne mora biti potpuna (primjer kvadratne porodice), dok je za logističku funkciju ona potpuna: $\mathcal{P}_M = \{\,[0, 1/2)\,,\ [1/2, 1]\,\}$, tj. $\Lambda_0 \cup \Lambda_1 = I$.

Ako je particija binarna, kao što smo imali u našem slučaju, tada je i funkcija mjernog kodiranja binarna:

$$\mathcal{B}:\ I \to \{0, 1\} \,, \tag{2.18}$$

$$\mathcal{B}(x) = \begin{cases} 0 \,, & x \in \Lambda_0 \\ 1 \,, & x \in \Lambda_1 \end{cases} \,.$$

Veza s funkcijom $\mathcal{S}$ je očita:

$$\begin{aligned} \mathcal{S}(x_0) &= s_0\ s_1\ s_2\ \ldots \\ &= \mathcal{B}\left(F^0(x_0)\right)\ \mathcal{B}\left(F^1(x_0)\right)\ \mathcal{B}\left(F^2(x_0)\right)\ \ldots \\ &= \mathcal{B}\left(F^0(x_0)\right)\ \mathcal{B}\left(F^0(x_1)\right)\ \mathcal{B}\left(F^0(x_2)\right)\ \ldots \\ &= \mathcal{B}(x_0)\,\mathcal{B}(x_1)\,\mathcal{B}(x_2)\ \ldots\ . \end{aligned} \tag{2.19}$$

Da rezimiramo, putopis kreiramo tako da nakon svake iteracije na faznom prostoru dobivenu točku kodiramo (binarno), te novodobiveni znak (bit) dodajemo u putopis. Dobiveni putopis $\mathbf{s}$ predstavlja beskonačni vremenski niz znakova kao simbolički zapis trajektorije sustava.

Gledano u kontekstu gradbe modela, dobivanje vremenskog niza odgovarajuće kodiranih podataka modelar (tvorac modela) očekuje kao rezultat mjerenja mjernog instrumenta. Jasno je da se modelar mora zadovoljiti aproksimacijom putopisa s pomoću njegovog podniza konačne duljine. U protivnom ne bismo bili u stanju dovršiti modeliranje u konačnom vremenu. Ova nužna aproksimacija ne predstavlja bitan gubitak općenitosti za sustave s konačnom složenosti, ili drugim riječima, za sustave u kojima se među simbolima pojavljuju korelacije konačne duljine. Modelar će jednostavno nastojati, barem u početku, uzimati dulje nizove od očekivane korelacije da osigura adekvatnu prezentaciju ponašanja sustava.

Otuda se nameće uvođenje pojma *vremenskog niza* (engl. *time series*), koji je uobičajen u ovom području. U našem razmatranju taj će pojam biti vezan uz ulazne podatke koji su na raspolaganju modelaru, i kao takvi moraju biti konačni. Stoga se nameće sljedeće određenje:

Definicija 2.7 *Vremenski niz je putopis generiran u konačnom vremenu.*

Navedeni tip aproksimacije je uobičajen za eksperimentalni pristup, neovisno o linearnosti, odnosno nelinearnosti promatranih sustava. Konačna prezentacija rezultata mjerenja zahtijeva odustajanje od koncepta egzaktnog putopisa, tj. moramo se zadovoljiti s vremenskim nizom konačne duljine. S problematikom mjerenja, koja predstavljaju izvorište vremenskog niza, pozabavit ćemo se u odjeljku 4.1.

2.3.2 Generirajuća particija

Već smo naglasili da je homeomorfizam $\mathcal{S}$ u stvari definiran nekom mjernom particijom $\mathcal{P}_M$ domene I. Dakle, prilikom prevođenja dinamike faznog prostora u

simboličku dinamiku, od presudne je važnosti nalaženje adekvatne particije domene (faznog prostora), tako da funkcija $\mathcal{S}$ ima svojstva homeomorfizma koji funkcije iteracije na Λ i Σ_∞ čini srodnima. Za funkcije dobrog ponašanja (npr. analitičke), uglavnom možemo primijeniti već spomenuti kriterij da se za diskriminirajuću točku particije uzme kritična točka x_c. Ranije navedeni primjeri, za particiju logističke jednadžbe ($x_c = 1/2$) i kvadratne porodice funkcija ($x_c = 0$, uz napomene u 2.2.2), dobro su poznati. Jasno je da ćemo u općenitom slučaju, a posebice ako se radi o funkcijama koje nisu glatke, za svaku pojedinu funkciju morati pribjeći rigoroznom matematičkom postupku.

Naziv *generirajuća particija* (engl. *generating partition*), koji je čest u literaturi ovog područja, odnosi se upravo na takvu mjernu particiju koja će definirati homeomorfizam s faznog prostora Λ na prostor simboličke dinamike Σ_∞. Uobičajeno se, međutim, generirajuća particija opisuje sljedećom kvalitativno izrečenom definicijom [10]:

Definicija 2.8 *Neka particija domene (faznog prostora) je* ***generirajuća*** *ako stvara takve vremenske nizove da se uz dovoljnu duljinu niza može po volji točno opisati početna točka u faznom prostoru.*

Ova je definicija ekvivalentna zahtjevu da je funkcija $\mathcal{S}^{-1}$ kontinuirana u svakoj točki $\mathbf{s} \in \Sigma_\infty$. Drugim riječima uz zadani $\epsilon > 0$, mora postojati n takav da za nizove $\mathbf{s}$ i $\mathbf{t}$, $\mathbf{s} \neq \mathbf{t}$, koji se podudaraju u svih n prvih znamenaka s_i, $i = 0, 1, \ldots, n-1$, vrijedi da je

$$d[\mathcal{S}^{-1}(\mathbf{s}),\ \mathcal{S}^{-\mathbf{1}}(\mathbf{t})] = d[x,\, y] < \epsilon\,;\quad x,\, y \in \Lambda\,. \tag{2.20}$$

Kao što je već istaknuto, dokaz za kontinuiranost funkcije $\mathcal{S}^{-1}$ provodi se u sklopu dokaza da se radi o homeomorfizmu (za kvadratnu porodicu vidjeti npr. [2]).

U našem je razmatranju bitno uočiti to, da i uz mjerni postupak vrlo grube naravi (koji se kod binarnih particija sastoji u nalaženju $s_j = \mathcal{B}(x_j)$, na način da se tek konstatira je li točka x_j u lijevom ili desnom dijelu 1D domene), uzimanjem dovoljnog broja slijednih simbola s_{j+i}, $i = 1, 2, \ldots, n-1$, za koje vrijedi:

$$s_{j+i} = \mathcal{B}(x_{j+i}) = \mathcal{B}(F^i(x_j)), \tag{2.21}$$

dobivamo po volji točno približenje točki x_j u Λ.

Istu tvrdnju možemo izreći i na sljedeći način. Ako primjenom funkcije iteracije F na točku x_j dobijemo trajektoriju $x_j x_{j+1} \ldots x_{j+n} \ldots$ kojoj odgovara beskonačni

vremenski niz $s_j s_{j+1} \dots s_{j+n-1} s_{j+n} \dots$, tada iščitavanjem konačnog podniza $\mathbf{s}_{jn} = s_j s_{j+1} \dots s_{j+n-1}$ duljine n, dobivamo informaciju o položaju x_j unutar intervala određenog s $\epsilon > 0$. Pri tom je ϵ zavisan od n i naravno, od oblika funkcije $\mathcal{S}^{-1}$, odnosno F. Nadalje, točku x_{j+1} preciziramo pomoću konačnog podniza $\mathbf{s}_{j+1n} = s_{j+1} s_{j+2} \dots s_{j+n}$ s istim redom točnosti ϵ, itd.

Primijetimo da funkcija $\mathcal{S}^{-1}$ nije tako jednostavne naravi kao $\mathcal{S}$, pa ni točnost ϵ nije u općem slučaju lako izraziva. Za logističku funkciju definiranu na jediničnom intervalu $[0, 1]$, s binarnom particijom koja dijeli interval po pola, u prvoj (linearnoj) aproksimaciji možemo uzeti da je $\epsilon \approx 2^{-n}$.

Poglavlje 3

Računarski pristup

Nakon što smo u prethodnom poglavlju opisali način dobivanja vremenskog niza iz dinamičkih sustava, okrećemo se teorijskim postavkama središnje teme ovog rada — ***stohastičkim konačnim automatima***. Isprva ćemo iznijeti kratak pregled nekih temeljnih pojmova iz *teorije izračunavanja* (*diskretnog izračunavanja*), *teorije automata i formalnih jezika*, a potom uspostaviti formalnu vezu s konačnim automatima. U tom je smislu ovo poglavlje teorijska podloga za naše daljnje razmatranje.

U teoriji ϵ-strojeva stohastički konačni automat predstavlja temeljni ϵ-stroj, na drugoj razini računskih modela, koji daje kauzalni model dinamičkog sustava. To će, kao i način tvorbe ovih automata, biti objašnjeno u sljedećem poglavlju.

3.1 Teorija automata i formalnih jezika

Najprije ćemo razmatrati razred konačnih automata iz kojeg, uvođenjem stohastičnosti prijelaza i formalnom prenamjenom za prihvat apriori nepoznatih nizova, slijede stohastički konačni automati. Na samom početku dane su neke temeljne definicije teorije formalnih jezika, automata i izračunavanja. Prikazana je hijerarhija automata, te zatim detaljnije izložen razred konačnih automata i njima odgovarajućih pravilnih izraza. Obrazložena je geneza kojom se iz konačnih automata dolazi do stohastičkih konačnih automata, za koje je dana i formalna definicija.

3.1.1 Formalni jezici

Objekti koje razmatra računarstvo su, kao što je već istaknuto u uvodnom poglavlju, diskretnog značaja. Ti objekti su nizovi znakova koje nazivamo i riječima

$\mathbf{w} = s_0 s_1 \dots s_{l-1}$. Znakovi ili simboli su elementi iz neke abecede $s_i \in \mathcal{A}$, $\mathcal{A} = \{0, 1, \dots, a-1\}$, gdje je a kardinalni broj skupa $\mathcal{A}$.

Sada možemo definirati *formalni jezik* na sljedeći način:

Definicija 3.1 *Formalni jezik* $\mathcal{L}$ *je skup riječi iz neke abecede* $\mathcal{A}$.

$$\mathcal{L} = \{\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_{m-1}\},\ \mathbf{w}_i \in \mathcal{A}^n, \quad i = 0, 1, \dots, m-1, \quad n, m \in \mathbb{N}_0^+ .$$

Jednostavnost definicije je više nego upečatljiva. O konkretnom jeziku ovisi pobliže određenje skupa, odnosno izbor riječi koje će on sadržavati.

Jedan od glavnih problema teorije izračunavanja je određenje računarske složenosti prepoznavanja jezika — gdje se prepoznavanje precizira kao razvrstavanje riječi glede njihove pripadnosti jeziku $\mathcal{L}$. Složenost se izražava kroz jeziku pridruženi tip *diskretnog stroja*, odnosno *automata* sposobnog za vršenje razredbenog postupka.

Pod *gramatikom* podrazumijevamo *teoriju o strukturi jezika* [11].

3.1.2 Diskretno izračunavanje i teorija automata

Jedan od problema teorije automata je već naznačen u prethodnom pododjeljku kao problem nalaženja automata koji raspoznaje određeni jezik. Formalno se pokazuje ekvivalencija između određenih jezika i automata koji ih raspoznaju, kao npr. između konačnih automata i tzv. *pravilnih izraza* (engl. *regular expressions*)[12]. U ovom smislu u teoriji izračunavanja postoji stalni dualizam između formalnih jezika kao skupova, i automata kao funkcija koje operiraju na tim skupovima. Konačnim automatima i pravilnim izrazima posvećen je sljedeći odjeljak.

Automati se međusobno razlikuju po svojem ustroju glede korištenih računarskih izvorišta. To se odnosi na memoriju, tj. njezin kapacitet (konačan ili beskonačan) i strukturu (način pristupa podacima), zatim na logičke operacije raspoložive pri obradi, te na vrijeme dopušteno za dovršenje klasifikacije.

Količina i svojstva raspoloživih izvorišta u automatu određuju njihovu kompleksnost, te kompleksnost jezika kojeg raspoznaju. Na taj način se utemeljuje osnova za hijerarhijsku podjelu automata prema strukturalnoj "snazi" i učinkovitosti njihovih mehanizama raspoznavanja. U nedostatku bolje riječi, sa *strukturalnom snagom* smo označili mjeru sposobnosti automata da raspoznaje složenije jezike. Npr. *stogovni automat* PDA (engl. *Push Down Automaton*) ima sposobnost razaznavanja jezika sa *sadržajno neovisnom* (engl. *context-free*) gramatikom, s funkcionalnošću dovoljnom za implementaciju standardnih računarskih jezika. Jasno je da se tu radi

o složenom ustroju, te da ovaj razred automata karakterizira relativno velika strukturalna snaga — za razliku od gore spomenutih konačnih automata kojima je krajnji doseg prihvaćanje relativno jednostavnih, pravilnih izraza.

3.1.3 Hijerarhija diskretnih strojeva

Ukoliko nam je cilj opisati strukturu nekog procesa, tada je nužno pronaći *elementarne uzorke* te strukture, odnosno u njenoj kvantifikaciji trebamo tražiti minimalne vrijednosti koje još uvijek reproduciraju opaženi ustroj. Ili, u okviru gore izložene računarske interpretacije, moramo pronaći stroj s najnižom razinom složenosti koji daje konačnu prezentaciju procesa. Razina složenosti se pri tom odnosi i na čim jednostavniji *razred* stroja, te na čim jednostavniju njegovu *implementaciju* u okviru danog razreda.

Zbog toga ćemo u rekonstrukciji modela poći od hijerarhijski najjednostavnijih strojeva. U slučaju da početno odabrani stroj ne predstavlja odgovarajući model, u sljedećem, tzv. inovativnom koraku, uvodimo moćniji. Stoga nam je zanimljiva hijerarhijska podjela diskretnih strojeva glede njihovih računarskih mogućnosti (sl. 3.1). Idući od dna prema vrhu nailazimo na sve moćnije strojeve, sposobne za interpretaciju jezika sa sve složenijim ustrojem (gramatikom).

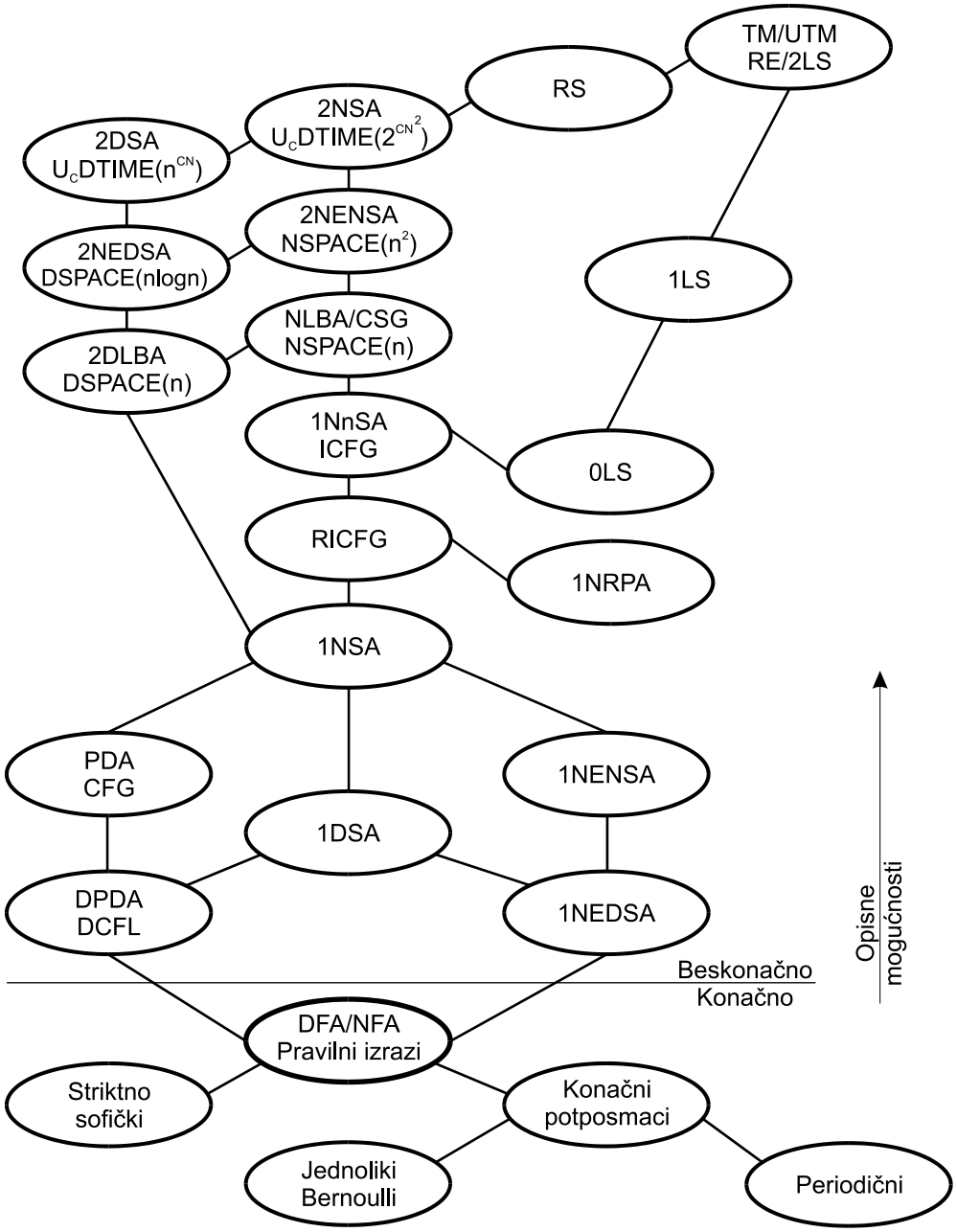
Ukoliko je razred strojeva $\mathcal{M}_i$ ispod razine razreda $\mathcal{M}_j$, te ako je na slici povezan s njime, tada strojevi iz razreda $\mathcal{M}_j$ razaznaju sve jezike kao i $\mathcal{M}_i$, uz mogućnost interpretacije i dodatnih jezika koje razina ispod ne razumije. Navedeni hijerarhijski uređaj ipak nije potpun, jer svi razredi automata nisu izravno usporedivi (ti na slici nisu povezani linijama).

3.2 Konačni automati

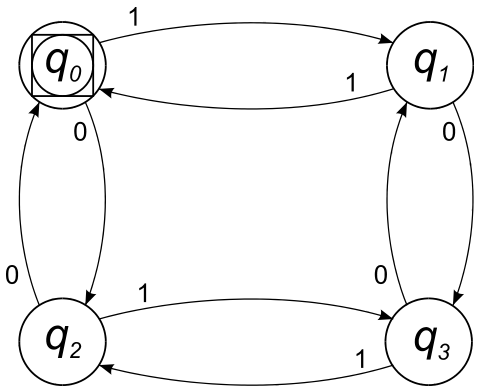
Definicija 3.2 *Konačni (deterministički) automat [engl. finite (deterministic) automaton] (D)FA definiramo kao uređenu petorku* $(Q, \mathcal{A}, q_0, \rho, Q_F)$, *gdje je* Q *konačni skup stanja od FA,* $\mathcal{A}$ *je konačna ulazna abeceda,* q_0 *je početno stanje iz* Q, $Q_F \subseteq Q$ *je skup prihvatljivih konačnih stanja, dok je* ρ *funkcija prijelaza:*

$$\rho : Q \times \mathcal{A} \to Q\,,$$
$$\rho(q, s) = q'\,.$$

Tu je q' *novo stanje, nastalo kao rezultat prijelaza nakon što je u stanju* q *primljen simbol* s.



Slika 3.1: Hijerarhija računarskih strojeva (automata). Preuzeto iz [10]. Idući od dna prema vrhu strojevi su sve moćniji, sposobni za interpretaciju sve složenijih jezika. Stroj više razine razaznaje jezik stroja niže razine s kojim je povezan. Legenda pridjeva (engl.): 1 (2) = 1 (2) way input tape, D (N) = (non)deterministic, I = indexed, RI = restricted I, n = nested, NE = nonerasing, CF = context free, CS = context sensitive, R = recursive, RE = R enumerable, U = universal. Legenda objekata (engl.): G = grammar, A = automata, FA = finite A, PDA = pushdown A, SA = stack A, LBA = linear bounded A, RPA = reading PDA, TM = Turing machine, LS = Lindenmayer system, 0L = CF LS, 1L = CS LS, RS = R set.

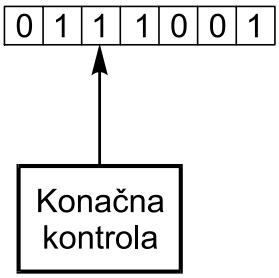


Slika 3.2: Primjer konačnog automata (FA). FA na slici prihvaća sve nizove koji sadrže ukupno paran broj jedinica i paran broj nula. Početno stanje automata označeno je upisanom kružnicom, a konačno upisanim kvadratom. U ovom primjeru konačno stanje se podudara s početnim.

3.2.1 Prijelazni dijagram

Konačni automat prikazujemo kao usmjereni graf u kojem se u ovisnosti o stanju i primljenom simbolu dešava prijelaz ρ. Taj graf se naziva *prijelazni dijagram* (engl. *transition diagram*). Ako niz w primljen kroz početno stanje završava u nekom od konačnih stanja, tada kažemo da je novo stanje automata *prihvatljivo stanje*, odnosno da FA *prihvaća* niz w.

Na slici 3.2 dan je primjer konačnog automata koji prihvaća sve nizove koji sadrže ukupno paran broj jedinica i paran broj nula. Početno stanje automata označeno je upisanom kružnicom, a konačno upisanim kvadratom. U ovom primjeru, konačno stanje se podudara s početnim.



Slika 3.3: Konačna kontrola. Shematski prikaz funkcije konačnih automata.

Konačni automat je stroj na kojem ostvarujemo tzv. *konačnu kontrolu* posredstvom primljenog niza. Općenito će FA primati simbole iz nekog niza, kojeg u teoriji automata zamišljamo kao zapisanog na traci (sl. 3.3). Ako je FA trenutno u stanju

$q \in Q$, pošto s trake očita simbol s prelazi u stanje određeno s $\rho(q, s)$, kao što je već naznačeno gornjom definicijom, te pomiče traku tako da može pročitati sljedeći (desni) simbol. Sve simbole zapisane na traci možemo promatrati kao jedinstveni niz w_T. Ukoliko je niz w_T prihvatljiv, tj. ako je nakon njegovog očitanja FA završio u jednom od konačnih stanja, tada kažemo da je FA prihvatio cijelu traku. Prije nego što je završen prolaz kroz cijelu traku FA može prihvatiti mnogobrojne prefikse od w_T, neovisno o tome da li je cijela traka prihvaćena ili nije.

Pošto nas kod konačne kontrole općenito zanima stanje stroja nakon primitka nekog niza w, u daljnjem razmatranju funkciju ρ zamjenjujemo s općenitijom funkcijom $\widehat{\rho}$:

$$\begin{aligned} \widehat{\rho} &: Q \times \Sigma_w \to Q\,, \\ \widehat{\rho}(q, w) &= q'\,. \end{aligned} \tag{3.1}$$

Ovdje je Σ_w skup svih nizova simbola (riječi) iz $\mathcal{A}$, uključujući i prazan niz koji ne sadrži ni jedan simbol. U skladu s prethodnom definicijom formalnog jezika, Σ_w je jezik koji sadrži u sebi sve moguće jezike definirane abecedom $\mathcal{A}$. Uz oznaku e za prazan niz, sada proizlazi sljedeća rekurzivna veza između funkcija ρ i $\widehat{\rho}$:

$$\begin{aligned} \widehat{\rho} &: Q \times \Sigma_w \to Q\,, \\ \widehat{\rho}(q, e) &= q\,, \\ \widehat{\rho}(q, ws) &= \rho(\,\widehat{\rho}(q, w),\, s\,)\,. \end{aligned} \tag{3.2}$$

Primitkom praznog niza FA ostaje u svom prethodnom stanju, a primitak proizvoljnog niza se svodi na niz prijelaza uzrokovanih svim prefiksima tog niza. Na uobičajeni način je s $w_1 w_2$ označeno spajanje (nadovezivanje) niza w_1 s nizom w_2, gdje će w_1 biti prefiks, a w_2 sufiks novostvorenog spojenog niza.

3.3 Pravilni izrazi

Već je spomenuta ekvivalencija konačnih automata i pravilnih izraza. Iz te ekvivalencije proizlazi da su jezici koje konačni automati prihvaćaju izrazivi s pomoću pravilnih izraza. Ovi izrazi na pregledan način prikazuju strukturu (gramatiku) jezika.

3.3.1 Kratak opis pravilnih izraza

Formalizacija pravilnih izraza izlazi iz okvira ovog rada.[1] Njihov je oblik, međutim, vrlo sugestivan i kod jednostavnijih primjera lako razumljiv. Tako npr. uz binarnu abecedu, pravilni izraz $(0+1)^*$ označava skup svih mogućih nizova sastavljenih od nula i jedinica (uključujući i prazan niz), a izraz $(01)^*$ skup svih nizova u kojima se kao osnovni element pojavljuje podniz 01, tj. skup $\{\, e, 01, 0101, 010101, \dots \}$. Zvjezdica pri tom označava tzv. *Kleenov zatvarač*, koji se za neki skup (riječi) $L \subseteq \Sigma_w$ definira kao:

$$L^* = \bigcup_{i=0}^{\infty} L^i \,. \tag{3.3}$$

Ovdje je $L^0 = e$, $L^i = LL^{i-1}$, a LL^{i-1} označava takav skup nizova u kojem se elementu iz L pripaja element iz L^{i-1}.[2] Primijetimo da vrijedi $(0+1)^* = \{0,1\}^*$, odnosno da u regularnom izrazu znak $+$ ima značenje (proizvoljnog) odabira jednog od elemenata iz skupa dobivenog uniranjem $0 \bigcup 1$, dok navođenje simbola jedan iza drugoga, kao u primjeru $(01)^*$, predstavlja njihovo spajanje u jedinstvenu riječ.

Ilustrativno je spomenuti još nekoliko jednostavnih primjera regularnih izraza. Tako npr. $(0+1)^*00(0+1)^*$ predstavlja skup svih nizova u kojima se pojavljuje najmanje jedan par nula, ispred i iza kojeg slijede bilo kakvi prefiksi, odnosno sufiksi. Izraz $(1+10)^*$ predstavlja skup nizova koji započinju s jedinicom i ne sadrže dvije uzastopne nule, dok $(0+e)(1+10)^*$ predstavlja skup svih nizova koji ne sadrže dvije uzastopne nule. Drugim riječima, jezik definiran predzadnjim pravilnim izrazom sadržan je u jeziku definiranom zadnjim pravilnim izrazom.

Uz ternarnu abecedu $\{0,1,2\}$, izraz $0^*1^*2^*$ označava skup nizova s proizvoljnim brojem nula, iza čega slijedi proizvoljan broj jedinica, iza čega slijedi proizvoljan broj dvojki. U skladu s gornjom definicijom zatvarača u nizu može biti i nula znamenki. Podskup tog skupa koji će sadržavati nizove u kojima je barem po jedna od svake znamenke iz abecede opisujemo izrazom $00^*11^*22^*$, što se kratko označava i kao $0^+1^+2^+$.

[1] Čitalac se upućuje na već spomenutu literaturu [12].

[2] U skladu s našim oznakama, vrijedi da je skup $\Sigma_w = \mathcal{A}^*$, tj. Σ_w je Kleenov zatvarač abecede $\mathcal{A}$.

3.3.2 Primjer pravilnog izraza za FA

Pravilni izraz ekvivalentan FA na slici 3.2 bit će oblika:

$$(\,(00+11)\,+(01+10)\,(00+11)^*\,(01+10)\,)^*\,, \tag{3.4}$$

što se dade zaključiti iz oblika prijelaznog dijagrama. Zagrade oko prvog unutarnjeg izraza nisu nužne i služe da naglase slučaj pojave uzastopnog broja parnih nula ili jedinica. Sljedeće tri unutarnje zagrade čine pravilni izraz za tvorbu sastavljenog niza u kojem broj *uzastopnih* nula i jedinica ne mora biti paran. Prva od njih opisuje slučaj kad se nakon pojave jedne nule i jedne jedinice, ili obrnuto – jedne jedinice i jedne nule, prelazi u desno donje stanje na dijagramu. Srednja opisuje mogućnost da se iz desnog donjeg stanja pročita proizvoljan paran broj nula i jedinica, a posljednja zagrada naznačava da se na koncu trebamo vratiti u konačno stanje u lijevom gornjem uglu. Lako je provjeriti da će gornji pravilni izraz, baš kao i konačni automat sa slike 3.2, prihvatiti svaki niz s parnim brojem nula i jedinica, kao npr. niz w_1:

$$w_1 = 101110000110101101 \;\; (prihvatljiv)\,;$$

a neće prihvatiti niz koji nema to svojstvo, kao npr. w_2:

$$w_2 = 1011100010110 \quad (nije\ prihvatljiv)\,.$$

3.3.3 Veza sa simboličkom dinamikom

Veza konačnih automata sa simboličkom dinamikom izloženom u prethodnom poglavlju očito se nameće. Raspišemo li skup Σ_w svih riječi, uz ograničenje na binarnu abecedu:

$$\Sigma_w \;=\; \{\, \underbrace{e}_{l=0},\, \underbrace{0,\ 1}_{l=1},\, \underbrace{00,\ 01,\ 10,\ 11}_{l=2},\, \underbrace{000,\ 001,\ \ldots}_{l=3},\, \ldots\ldots \,\}\,, \tag{3.5}$$

lako je uočiti da je to skup svih podnizova. Može se primijetiti da je Σ_w jednak skupu svih podnizova sadržanih u gustoj orbiti (2.15) simboličke dinamike.

Odavde slijedi da je čitanje niza duljine l od strane FA analogno l-strukom djelovanju posmačnog preslikavanja σ na točke iz prostora beskonačnih nizova Σ_∞.

Primanjem uzastopnih nizova FA analizira da li su oni prihvatljivi ili nisu, odnosno da li odgovaraju njegovom ustroju. Ako je određeni niz prihvatljiv, tj. ako njegova struktura odgovara pravilnom izrazu ekvivalentnom za dati FA, tada će

svaki takav niz biti ekvivalentan nekom konačnom stanju. Na taj način, FA opisan uređenom petorkom $(Q, \mathcal{A}, q_0, \rho, Q_F)$, odnosno svojim prijelaznim dijagramom, predstavlja kriterij za usporedbu nepoznatih struktura sa svojom strukturom. Nalaženjem odgovarajućeg FA koji će prihvatiti sve nizove emitirane od nepoznatog procesa, našli smo računarski opis te strukture i omogućili njenu kvantifikaciju. Kvantifikacija se može zasnivati npr. na broju stanja u Q, ili u idućoj aproksimaciji, na odgovarajućoj kvantifikaciji mogućih prijelaza i njihovih svojstava, i sl.

3.4 Stohastički konačni automati

Stohastički konačni automati (engl. *Stohastic Finite Automata*) *SFA* predstavljaju prirodnu preinaku razreda konačnih automata, na način da oni oslikavaju i statistička svojstva sustava. Ako je npr. iz nekog stanja sustava moguće izvršiti više prijelaza, pri čemu se svaki od njih dešava kod pojave određenog ulaznog simbola, tada će SFA sadržavati vjerojatnosti za svaki takav prijelaz. Uvođenjem statističkih elemenata SFA postaju temeljni elementi *Statističke mehanike* dinamičkih sustava[1, 9]. Njezina je zadaća da, po analogiji sa statističkom fizikom, objašnjava "makroskopske pojave" kroz sakupljanje podataka o "mikroskopskim stanjima". U slučaju modeliranja to se svodi na sažimanje tog (relativno opsežnog) skupa mikroskopskih ili elementarnih mjerenja u (relativno mali) skup makroskopskih opservabli. Odnosi između opservabli predstavljeni su ustrojem konačnog modela. Na taj način dobivamo kvalitativno novi pristup koji sadrži i statističke elemente, ali i opis strukture izražene jezikom diskretnog izračunavanja.

3.4.1 Geneza iz razreda FA

Polazimo od razreda FA predstavljenog uređenom petorkom $(Q, \mathcal{A}, q_0, \rho, Q_F)$, prema definiciji 3.2. Uvedimo skup R definiran kao skup svih mogućih prijelaza r u usmjerenom grafu FA. r odgovara uređenoj trojci (q, q', s) koja opisuje dozvoljenu promjenu stanja q u q' po primitku simbola s:

$$R = \{\, r \,:\, r = (q, q', s),\ \ q, q' \in Q\,,\ s \in \mathcal{A}\,\}\,, \qquad (3.6)$$
$$ili\ \ r = q \underset{s}{\rightarrow} q'\,.$$

Za svaki mogući prijelaz, SFA daje podatak o njegovoj vjerojatnosti pod uvjetom da je početno stanje bilo q, što ćemo opisati djelovanjem uvjetne vjerojatnosti Pr

na prijelaze iz skupa R. Time se dobiva novi skup $\mathbf{T}$ vjerojatnosti prijelaza koji zbog diskretnosti skupova Q i $\mathcal{A}$ ima značaj tenzora. U tom tenzoru će prva dva indeksa označavati početno, odnosno konačno stanje, a treći će označavati simbol po prijamu kojeg je došlo do prijelaza. Iznesene tvrdnje pišemo kao:

$$\begin{gathered} \Pr(r) = \Pr(\,(q,\, q',\, s)\,) = \Pr(\,q',\, s \mid q\,)\,; \\ \Pr(R) = \mathbf{T}\,, \quad \mathbf{T}^{s}_{qq'} = \Pr(\,q',\, s \mid q\,)\,. \\ \sum_{q' \in Q} \sum_{s \in \mathcal{A}} \Pr(\,q',\, s \mid q) = 1\,. \end{gathered} \tag{3.7}$$

Prijelazni tenzor $\mathbf{T}$ u potpunosti opisuje prijelaznu i vjerojatnosnu strukturu modela.

3.4.2 Nedefiniranost konačnih stanja

Prije nego okončamo formalizaciju SFA potrebno je obratiti pažnju na još jednu njihovu posebnost u odnosu na razred konačnih automata. Ta posebnost proizlazi iz njihove primjene na analizu nepoznatih nizova. Tj. za razliku od FA, kod SFA ne znamo a priori gramatiku jezika kojeg prihvaćamo. Otuda se pojavljuje problematika nalaženja i definiranja konačnih stanja. Ukratko bi se moglo ustvrditi da se konačna stanja mogu pronaći u sljedećoj fazi, u kojoj bismo već rekonstruirane SFA nastojali semantički dublje proanalizirati u kontekstu njihove vlastite strukture. Stoga ćemo skup Q_F jednostavno smatrati identičnim skupu svih stanja: $Q_F = Q$. Drugim riječima, svaki niz koji vodi na neko od postojećih stanja u SFA je prihvatljivi niz. Modelar koji gradi računski model sustava nema načina da unaprijed diskriminira pojedina stanja kao prihvatljiva, a druga kao neprihvatljiva.

Ovo bismo mogli opisati kao nužnost *nepristranosti* u gradbi modela prema pravilima modeliranja iz vremenskog niza.

3.4.3 Formalni opis

Uvažimo li činjenicu o irelevantnosti konačnih stanja za stohastičke konačne automate iz prethodnog pododjeljka, te izraze (3.6) i (3.7), vrijedi sljedeće određenje:

Definicija 3.3 *SFA opisujemo kao uređenu četvorku* $(\,Q,\, \mathcal{A},\, q_0, \mathbf{T})$ *određenu skupom stanja* Q, *abecedom* $\mathcal{A}$, *početnim stanjem* q_0 *i vjerojatnosnim tenzorom prijelaza* $\mathbf{T}$.

Primijetimo da je u tenzoru $\mathbf{T}$ sadržana potpuna informacija o skupu dozvoljenih prijelaza R. Naime, za svaki dozvoljeni prijelaz $r \in R$, pripadna je vjerojatnost $\Pr(r) > 0$.[3] Za nemoguć prijelaz vrijedit će $\mathbf{T}^{s}_{qq'} = 0$, a za deterministički prijelaz, u kojem se prihvaća samo jedan simbol $s_d \in \mathcal{A}$, bit će $\mathbf{T}^{s_d}_{qq'} = 1$.

Uz zanemarivanje stohastičke dimenzije možemo govoriti o aproksimaciji SFA s FA, uz već spomenuto zanemarenje konačnih stanja.

Iz izloženog slijedi da:

i. *stohastički konačni stroj* M_{SFA} iz razreda SFA formalno opisujemo kao $M_{SFA} = (Q, \mathcal{A}, q_0, \mathbf{T})$,

ii. a njegovu *nestohastičku* aproksimaciju s pomoću stroja iz razreda FA, kao $M_{FA} = (Q, \mathcal{A}, q_0, R)$.

Svakom SFA možemo pridružiti odgovarajući pravilni izraz, iako se oni kompliciraju zbog vjerojatnosnih parametara. Zbog toga će pravilni izrazi biti praktični tek u jednostavnijim slučajevima. Npr. za slučaj Bernoullijevog niza nula i jedinica s jednakom vjerojatnosti za pojavu oba simbola, imali bismo pravilan izraz $(0 + 1)$ analogan onom za razred FA, a u slučaju da je vjerojatnost za 0 dva puta veća od vjerojatnosti za 1, mogli bismo to zapisati kao $(0[\frac{2}{3}] + 1[\frac{1}{3}])$.

[3] Ovdje se implicitno podrazumijeva da je SFA "dobro" definiran. Npr. ako smo ga reproducirali iz mjerenja, SFA mora primiti statistički zadovoljavajući broj nizova simbola kako bi ustanovljene vjerojatnosti statistički signifikantno opisivale ustroj sustava (vidjeti također sljedeća dva poglavlja).

Poglavlje 4

Gradba ϵ-strojeva

Zadatak ovog poglavlja jest pregled osnovnih postavki teorije ϵ-strojeva[1, 9, 10, 13]. U uvodnom je poglavlju već dan kratak esejski opis iterativnog procesa gradbe ϵ-strojeva kao začetak moguće nove teorije modeliranja općenitih nelinearnih sustava (1.3.3). J. P. Crutchfield navodi nekoliko naziva područja čiji je zadatak utrti nove putove nelinearnog modeliranja. Radi se npr. o *teoriji inovativnosti* (engl. *theory of innovation*) i *izračunu novopojavnosti* (engl. *calculi of emergence*), u sklopu novog formalnog pristupa induktivnom zaključivanju uz uporabu računarskih koncepata.

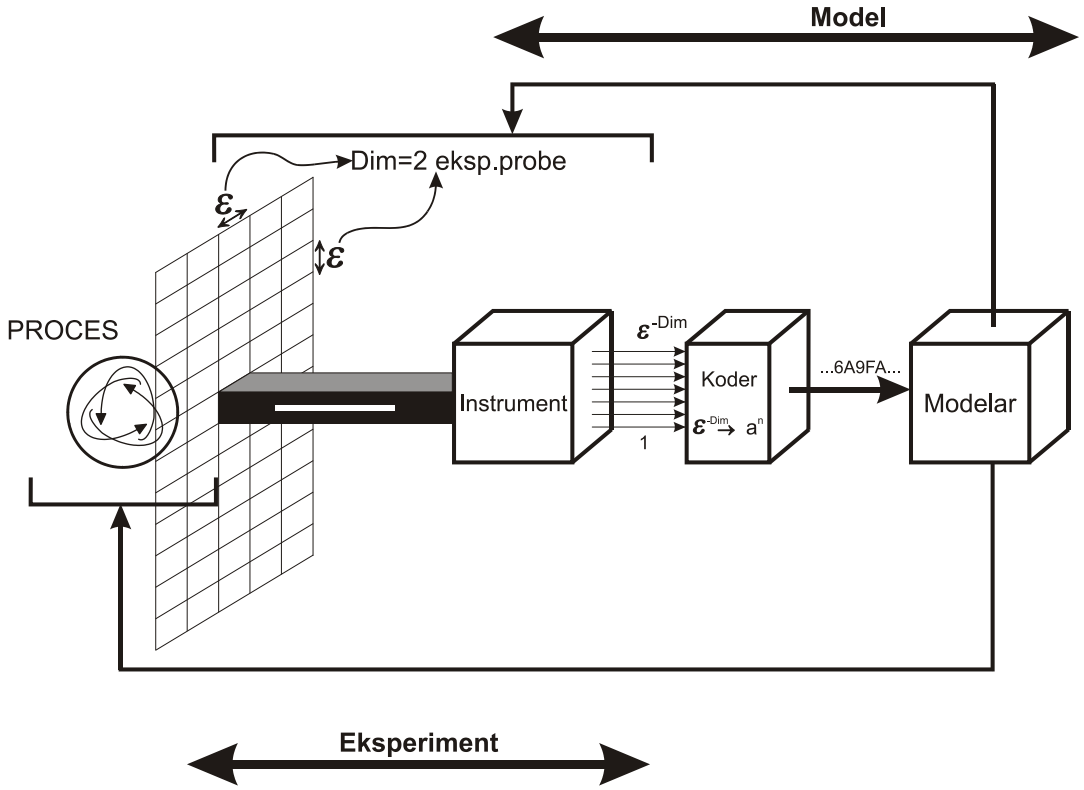
Glavna pitanja ovog područja su sljedeća:

- Kako modelar (agent) iz promatranja procesa (svoje okoline), na temelju svojih (ograničenih) računarskih izvorišta i nekog početnog, elementarnog modela, može iznaći nešto novo?
- Kako će ta novopojavnost biti izražena u terminima starog znanja?
- Kako poopćiti i automatizirati proces inovativnosti i stvaranja modela?

U daljnjem izlaganju nastojimo odgovoriti na ova pitanja i konkretizirati ideje iznesene u ranijim poglavljima. Nakon razmatranja *mjernog kanala*, posvećujemo se pojmu *statističke složenosti* i općenitoj definiciji ϵ-strojeva. Završni odjeljci daju algoritam gradbe ϵ-strojeva te njegovu specijalizaciju za tvorbu SFA.

4.1 Mjerni kanal

Neka nam je zadaća opisati sustav $\mathcal{S}$ uz koji načelno vežemo neki (iterativni) proces P. Na raspolaganju nam je određeni *mjerni instrument* $\mathcal{I}$ i *apstraktni modelar*



Slika 4.1: Shematski prikaz mjernog kanala. Na slici je prikazan primjer 2-Dim eksperimentalne probe koja diskretizira mjerni prostor na ćelije dimenzija $\varepsilon \times \varepsilon$. Postoji ukupno $\varepsilon^{-\,\mathrm{Dim}}$ mogućih vrijednosti očitanja koje se odgovarajuće kodiraju i prenose modelaru (agentu). Instrument je zajedno s procesom sastavni dio eksperimenta, ali i dio modela. Pretpostavlja se mogućnost modificiranja instrumenta u tijeku modeliranja, na osnovi informacija prikupljenih o sustavu.

kao stvaralac modela. Cilj modelara je izgraditi najbolji mogući model promatranog procesa P na osnovi dobivenih mjerenja. U tom sadržajnom okviru govorimo o mjernom kanalu (slika 4.1).

4.1.1 Nužnost eksperimentalne aproksimacije

Primijetimo da se gradba ϵ-strojeva inherentno naslanja na eksperiment kao temeljnu odrednicu modeliranja iz vremenskog niza. Način modeliranja nas tjera da razmišljamo što to u stvari primamo, i u kojoj mjeri ta informacija predstavlja valjanu osnovicu za gradbu vjerodostojnog modela. Već smo u pododjeljku 2.3.1 istaknuli nužnost aproksimacije putopisa s konačnim vremenskim nizom. Bez obzira je li promatrani sustav linearan ili nelinearan, njegova eksperimentalna izmjera je dana

podacima konačne (ograničene) točnosti. Konačna prezentacija rezultata mjerenja zahtijeva odustajanje od koncepta egzaktnog putopisa, tj. moramo se zadovoljiti s vremenskim nizom konačne duljine. Kod linearnih sustava se ova činjenica gotovo previđala, odnosno njezin utjecaj nije bio od presudne važnosti na poimanje znanstvenih mjerenja. Tu su, uz dobro poznati model, mala odstupanja u mjerenju početnih uvjeta rezultirala u odgovarajuće malim odstupanjima od konačnih vrijednosti stanja sustava.[1]

Pod *mjerenjem sustava* podrazumijevamo izmjeru njegovih varijabli stanja pri nekim zadanim parametrima koje podrazumijevamo konstantnima. Čak i da imamo točna mjerenja, model s konačnom prezentacijom koji je stvoren s pomoću konačnih računarskih izvorišta, rezultira s implicitnom nemogućnošću za točno predviđanje stanja sustava. Ta činjenica nas tjera da preispitujemo i naše poimanje eksperimentalnog pristupa, odnosno toga koliko bi precizna mjerenja trebala biti i što uopće možemo očekivati pri zaključivanju iz rezultata mjerenja.[2]

Modeliranje na temelju vremenskog niza bitno se zasniva na instrumentu s kojeg je niz primljen, i to ne samo glede točnosti mjerenja koje obavlja instrument, nego i sveukupne prikladnosti instrumenta da u sklopu tzv. mjernog kanala, kroz mjerenja odražava suštinske odrednice sustava (usporediti s primjerom iznesenim u 1.2.2). Uz točnost instrumenta označenu realnim pozitivnim parametrom ε, računski modeli koje gradimo inherentno nose pogrešku koja se ne može spustiti ispod te vrijednosti. To je jedan od razloga da se stvoreni modeli nazovu ϵ-strojevi[10], gdje je ϵ kumulativni pokazatelj točnosti. Kvalitativno gledajući, ϵ ima i dodatnu ulogu da podsjeća na upravo izrečenu činjenicu – instrument mada i jako precizan ali neodgovarajući strukturalnoj suštini sustava, vodi na neadekvatan vremenski niz koji će dati iskrivljenu sliku sustava.

Najeksplicitnije je ova problematika izražena kroz svojstvo kaotičnih sustava da su osjetljivi na početne uvjete. Otuda je lako razumjeti opravdanost pitanja koliko precizno ima smisla mjeriti sustav koji je npr. u kaotičnom režimu? Ili, ono što je bitno za računarstvo, koliko precizno ima smisla izračunavati trajektoriju takvog sustava? Naravno, sa stanovišta predviđanja, ne moramo odmah posegnuti za čistim statističkim pristupom. Dinamički sustavi pokazuju slojevitu evoluciju na svom prijelazu od determinizma ka stohastičnosti ili, kako se to uobičajeno kaže, *na putu*

[1] U okvirima Laplacéovog determinizma općenito se problem mjerenja nije smatrao inherentnim problemu modeliranja.

[2] Očita je sličnost s eksperimentalnim nedeterminizmom pri mjerenju mikrosustava, gdje je nedeterminizam posljedica kvantnih relacija neodređenosti.

prema kaosu. Na tom se putu pojavljuje bogatstvo struktura, i osim prikaza tih struktura, željeli bismo uvesti metode za kvantifikaciju njihove složenosti. Sigurno je da, odustajući od determinizma trajektorije, moramo nalaziti kvalitativno nove načine za opis ponašanja sustava. To je cilj našeg daljnjeg izlaganja.

4.1.2 Diskretizacija mjernih signala

Označimo s $\mathbf{X}_t$ stanje sustava u trenutku t, sa $\boldsymbol{\xi}_t$ proces koji predstavlja uneseni šum, te s $\mathbf{F}$ jednadžbe gibanja kao vektorske funkcije. Tada jednadžbu iterativnog procesa P pišemo kao

$$\mathbf{X}_{t+\triangle t} = \mathbf{F}\left(\mathbf{X}_t\,,\,\boldsymbol{\xi}_t\,,\,t\right)\,. \tag{4.1}$$

Kao što je prikazano na sl. 4.1, stanje procesa $\mathbf{X}_t$ doznajemo posredstvom instrumenta $\mathcal{I}$ kao projekciju $\mathcal{S}$ stanja $\mathbf{X}_t$ na tzv. mjerni prostor $\mathcal{R}^{\mathrm{Dim}}$ (npr. Euklidski prostor). Tu je dimenzija Dim jednaka broju eksperimentalnih proba instrumenta $\mathcal{I}$. Instrument je neki *pretvornik* (engl. *transducer*) i kao što se nazire sa slike, on ima mogućnost prilagođivanja i ugađanja za dati eksperiment "u hodu".

U ovisnosti o rezoluciji ε instrumenta, govorimo o podjeli mjernog prostora na *skup mjernih signala*, ili jednostavno *skup podjeljaka* koji označavamo s $\Pi_\varepsilon(\mathrm{Dim})$. Vrijedi:

$$\begin{aligned}\Pi_\varepsilon(\mathrm{Dim}) &= \{\,\pi_i : \pi_i \in \mathcal{R}^{\mathrm{Dim}},\ i \in N_\pi\,\},\quad \mathrm{Dim} \in \mathbb{N}\,,\ N_\pi \subset \mathbb{N} \\ N_\pi &= \{0, 1, \ldots\,, n_{\mathcal{I}} - 1\}\,,\ n_{\mathcal{I}} = \ulcorner\, \varepsilon^{-\,\mathrm{Dim}}\, \urcorner\,.\end{aligned} \tag{4.2}$$

Svaka ćelija π_i je klasa ekvivalencije skupa svih stanja $\mathbf{X}_t$ projiciranih na $\mathcal{R}^{\mathrm{Dim}}$ koja su nerazlučiva za dati mjerni instrument. Skup N_π predstavlja skup svih mogućih indeksa i, kao skup svih efektivnih rezultata mjerenja. Funkcija instrumenta $\mathcal{I}$ je nalaženje podjeljka π_i kojem odgovara projekcija $\mathcal{S}\left(\mathbf{X}_t\right)$, što možemo opisati kao

$$\mathcal{I} : \mathcal{R}^{\mathrm{Dim}} \to N\,, \tag{4.3}$$

$$\mathcal{I}\left(\mathcal{S}\left(\mathbf{X}_t\right)\right) = i\,,\ \ uz\ \ \mathcal{S}\left(\mathbf{X}_t\right) \in \pi_i\,. \tag{4.4}$$

Ono što predstavlja rezultirajući izlaz mjernog kanala je odgovarajući kodni prikaz indeksa i ustanovljenog podjeljka π_i u obličju niza znakova (engl. *string, series*). Prirodno je da skup N_π s $n_{\mathcal{I}}$ elemenata interpretiramo jednostavno kao kodnu abecedu $\mathcal{A}$ s $a = \mathrm{card}\left(\mathcal{A}\right) = n_{\mathcal{I}}$ znakova, tako da u daljnjem razmatranju govorimo o abecedi $\mathcal{A}$ kao skupu od a elemenata. Niz rezultata mjerenja se tako formalno

svodi na niz simbola, odnosno vremenski niz $\mathbf{s}$ uveden u 2.3.1:

$$\mathbf{s} = s_0 s_1 \ldots s_i \ldots , \quad s_i \in \mathcal{A} , \quad i = 0, 1, 2, \ldots , l-1 . \tag{4.5}$$

Uobičajeni izbor binarne abecede $\mathcal{A}$ je već diskutiran (vidjeti 2.2), no radi općenitosti možemo zasad nastaviti izlaganje za proizvoljnu abecedu. Također, primijetimo da unutar vremenskog niza $\mathbf{s}$ možemo promatrati podnizove $\mathbf{w}$ kao riječi nekog formalnog jezika, u skladu s razmatranjem u 3.1.1. U praksi će se to svesti na sustavno analiziranje riječi $\mathbf{w}$ izvađenih iz primljenog niza $\mathbf{s}$ prema nekim kriterijima, što će biti ekvivalentno analizi strukture jezika odaslanog od procesa (vidjeti odjeljak 4.4). Na ovaj način smo jasno povezali temelje teorije ϵ-strojeva sa simboličkom dinamikom s jedne i računarskim teorijama s druge strane.

4.1.3 Gruba izmjera

Čitatelj je lako mogao uočiti da prethodni izraz (4.5) odgovara vremenskom nizu formalno uvedenom u pododjeljku 2.3.1. Strogo gledano, to će vrijediti u slučaju kad uz binarnu abecedu uvedemo tzv. *grubi instrument* (engl. *coarse graining, coarse measurement, ...*), što ćemo nazivati *gruba izmjera.* Taj se pojam u potpunosti poklapa s pojmom mjerne particije $\mathcal{P}_M$ (2.17) faznog prostora. Tu ćemo particiju dobiti u slučaju jednodimenzionalnih sustava uz $a = n_{\mathcal{I}} = 2$. Drugim riječima, za rezoluciju ε danog instrumenta mora vrijediti

$$\varepsilon \simeq 1/2 , \quad 1/2 \leq \varepsilon < 1 ,$$

što vodi na samo dvije ćelije $\pi_i , \ i = 0, 1$ i binarnu abecedu $\mathcal{A} = \{0, 1\}$. Sada je niz (4.5) ekvivalentan vremenskom nizu kao konačnom putopisu.

Usprkos gruboj izmjeri, u skladu s razmatranjem generirajuće particije u 2.3.2, uz dovoljno veliki broj k emitiranih simbola dobit ćemo po volji točne rezultate. Uvjet za to je da instrument bude "dobro ugođen", u smislu da podjeljci π_0 i π_1 moraju realno odražavati pripadnost varijabli stanja podskupovima mjerne particije $\mathcal{P}_M$.

4.1.4 Proces kao izvor informacije

Sa stanovišta modelara instrument sa sl. 4.1 je izvor informacija, pa možemo razmatrati i njegov informacijski aspekt. U svezi s procesom P uvodimo mjeru $\mu(\mathbf{X})$ na njegovom konfiguracijskom prostoru, te *omjer entropije* (engl. *entropy rate*) $H_\mu(\mathbf{X})$,

kao mjeru količine informacije koju proces stvara u jedinici vremena, odnosno tijekom vremena τ potrebnog za jednu izmjeru stanja. Kao što je uobičajeno u teoriji informacija, uzimamo $\tau = 1$. Sada instrument $\mathcal{I}$ kojim mjerimo proces P ima značaj informacijskog kanala kroz koji se informacija $H_\mu(\mathbf{X})$ prenosi do modelara. Kapacitet tog kanala predstavlja maksimalnu informaciju I_P koju instrument može prenijeti o procesu:

$$I_P = H(\mathbf{p}) \,. \tag{4.6}$$

Ovdje je $\mathbf{p}$ distribucija vjerojatnosti

$$\mathbf{p} = (p_0, p_1, \ldots, p_{a-1}) \,,\; p_i = \mu(\pi_i) \,,\; \pi_i \in \Pi_\varepsilon(\mathrm{Dim}) \,. \tag{4.7}$$

Dakle, p_i predstavlja vjerojatnost da je pojedini rezultat mjerenja iz mjernog prostora $\mathcal{R}^{\mathrm{Dim}}$ unutar podjeljka π_i. Mjera μ mora biti normirana kako bi bio zadovoljen uvjet normiranosti distribucije vjerojatnosti. Na uobičajeni način, s $H(\mathbf{p})$ smo označili Shannonovu entropiju distribucije vjerojatnosti $\mathbf{p}$:

$$H(\mathbf{p}) = -\sum_{i=0}^{a-1} p_i \,\mathrm{ld}\, p_i \,, \tag{4.8}$$

uz oznaku dualnog logaritma $\log_2 = \mathrm{ld}$.

Da bi instrument prenio svu informaciju o procesu, mora vrijediti Shannonov teorem za kodiranje bez dodatka šuma, koji u našem slučaju implicira:

$$I_P \geq H_\mu(\mathbf{X}) \,. \tag{4.9}$$

Ako gornji izraz ne vrijedi, tada je na instrumentu došlo do gubitka informacije (ekvivokacije) od $H_\mu(\mathbf{X}) - I_P > 0$. Ova izgubljena informacija je nenadoknadiva i predstavlja donju granicu efektivnog šuma unesenog u rekonstruirani model.

Iz gornjeg razmatranja proizlazi da je u slučaju dovoljnog kapaciteta mjernog kanala moguće izvršiti preslikavanje (tj. kodiranje) iz mjernog prostora u skup indeksa podjeljaka (niza simbola) bez unošenja šuma. No, to bi vrijedilo samo uz mogućnost raspolaganja proizvoljno velikim resursima zaključivanja. Uz realnu pretpostavku konačnih računarskih izvorišta pojavljuje se i dodatni pad kvalitete modela, odnosno porast razine šuma.

4.1.5 Važnost odabira instrumenta

Ukoliko je skup podjeljaka instrumenta $\Pi_\varepsilon(\mathrm{Dim})$ generirajuća particija, u skladu s definicijom 2.8 to znači da će beskonačni niz indeksa i odgovarati pojedinom stanju procesa. Za taj slučaj Kolmogorov teorem[14] govori da je $H(\mathbf{p})$ maksimalno ukoliko se radi o deterministčkom procesu za koji je entropija $H_\mu(\mathbf{X}) > 0$. Pod deterministčkim procesom pri tom podrazumijevamo takav proces koji ne posjeduje inherentne stohastičke elemente, ali mu je entropija kao mjera neizvjesnosti stanja općenito veća od nule. Drugim riječima, iako je proces inherentno deterministički, zbog svoje nelinearnosti i osjetljivosti on navodi na mogućnost postojanja neizvjesnosti za vanjskog promatrača.

Slična tvrdnja vrijedi i za kombinirane sustave, tj. one koji su inherentno deterministički ali i pod utjecajem nekog vanjskog izvora šuma, što odgovara tipičnoj situaciji stvarnih mjerenja[15].

Iz toga proizlazi da zahtjev za generirajućom particijom utječe na to kako instrument mora biti konstruiran. Npr. broj eksperimentalnih proba Dim je time strogo određen, a instrument koji neće uhvatiti odgovarajuću dimenzionalnost problema je neadekvatan. Pažljivi čitatelj će uočiti i još jedan problem — kako odrediti generirajuću particiju bez poznavanja jednadžbi gibanja? U 2.3.2 je kriterij za stvaranje generirajuće particije jednodimenzionalnih sustava bila kritična točka, definirana preko jednadžbi iterativnog procesa. Ako nam te jednadžbe nisu a priori poznate, kao što slijedi po definiciji zaključivanja iz vremenskog niza, tada se javlja problem početne izvedbe instrumenta. Očito je da instrument mora biti "pažljivo odabran" i "dobar", odnosno, da bi trebao davati čim više informacija o procesu. No za modeliranje *ab initio* ne postoji način da se ti kriteriji odrede unaprijed.

4.1.6 Epistemološka problematika mjerenja

Gornje obrazloženje možemo izreći i na drugi način. U eksperimentu nam je cilj posredstvom instrumenta izvući čim više informacija iz procesa. Kolmogorov teorem kaže da će za svaki proces s entropijom većom od nule maksimalna spoznatljiva informacija $H(\mathbf{p})$ biti primljena onda, ako je (instrumentalna) particija $\Pi_\varepsilon(\mathrm{Dim})$ generirajuća. Sa stanovišta modeliranja nepoznatih sustava, međutim, zbog nepoznavanja jednadžbi gibanja ne možemo ustanoviti koja bi to particija zaista bila generirajuća. S druge strane, da poznajemo jednadžbe gibanja, modeliranje ne bi bilo *ab initio*. S tim u svezi, Kolmogorov teorem nije od praktične koristi u našem

razmatranju[1], ali omogućuje da se spoznaju operativni kriteriji za pojam *dobrog instrumenta.* To je onaj instrument koji će nam omogućiti prijam podataka dostatnih za početak modeliranja, te koji će kroz povratnu informaciju od modelara moći biti modificiran u smislu dobivanja novog niza podataka bolje aproksimacije, itd. Stoji fundamentalna primjedba da su ovako dobiveni zaključci nerigorozni. Kao protuodgovor pak možemo istaći da je (rigorozno) znanje predstavljeno jednadžbama gibanja samo matematička idealizacija.

I na kraju ovog odjeljka, istaknimo najvažnije zaključke. Instrument kao najvažniji čimbenik mjernog kanala mora biti prilagođen i ugođen promatranom procesu, što načelno zahtijeva iteracijski postupak njegove modifikacije. Modifikacija instrumenta treba ići u smjeru približenja generirajućoj particiji kao idealnoj podjeli mjernog prostora, što će se odražavati u maksimizaciji količine informacija koje modelar prima od instrumenta. Izvor neadekvatnosti modela može biti i instrument premalog kapaciteta, ali i neizbježna ograničenost računarskih izvorišta. Upravo je to ograničenje razlog iz kojeg proizlaze prividna stohastičnost i nepredvidljivost sustava, koje same ne moraju nužno biti pravi odraz promatranog procesa.

4.2 Statistička složenost

Općenito se metrika računarskih mogućnosti izražava u terminima *složenosti* (*kompleksnost*, engl. *complexity*). Npr. automat većih računarskih mogućnosti ima veću složenost. Promatramo li stanje nekog općenitog objekta $\mathbf{x}$ opisanog rječnikom iz nekog jezika $\boldsymbol{\lambda}$, tada se govori o "prezentaciji $\mathbf{x}$-a pomoću $\boldsymbol{\lambda}$". To odgovara skalarnom produktu $\langle \mathbf{x}|\boldsymbol{\lambda}\rangle$ vektora $\langle \mathbf{x}|$ i $|\boldsymbol{\lambda}\rangle$, iz međusobno dualnih vektorskih prostora, tj. prostora koji opisuje stanja objekta s jedne, i mogućih riječi odabranog formalnog jezika s druge strane. Naša je pretpostavka da je prvo (x) izrazivo s drugim $(\boldsymbol{\lambda})$. Prvo može predstavljati niz rezultata mjerenja nekog sustava, a drugo, naš izbor u modeliranju.

Označimo li sada s $M_{\min}\langle \mathbf{x}|\boldsymbol{\lambda}\rangle$ minimalnu takvu prezentaciju s obzirom na rabljeni vokabular unutar jezika $\boldsymbol{\lambda}$, tada je kompleksnost određena mjera (veličina) te minimalne reprezentacije:

$$C(x) = || M_{\min}\langle \mathbf{x}|\boldsymbol{\lambda}\rangle ||. \tag{4.10}$$

Jasno je da ovako definirana složenost ovisi o izboru jezika $\boldsymbol{\lambda}$ iz kojeg odabiremo vokabular, odnosno ovisi o izboru razreda automata kao osnove modeliranja. Npr.

složenost će općenito biti različita za slučaj modeliranja temeljenog na razredu konačnih automata, od onog temeljenog na razredu stogovnih automata.

4.2.1 Deterministička složenost

Iz prethodnog razmatranja se nameće ideja da se odabirom jednog standardnog jezika, odnosno standardnog stroja, čvrsto definira pojam složenosti. Prirodan izbor za to je *univerzalni Turingov stroj* (engl. *Universal Turing Machine*), koji ćemo u daljnjem tekstu označavati kraticom UTM.

U teoriji informacija (vidjeti npr.[16]) uobičajeno se definira *složenost po Kolmogorovu i Chaitinu*, ili kraće *K-C složenost,* na sljedeći način:

Definicija 4.1 *K-C složenost* $K(\mathbf{x})$ *nekog niza* $\mathbf{x}$ *se definira kao*

$$K(\mathbf{x}) = \min_{prog\,:\,UTM(prog)=\mathbf{x}} length(prog) \ ,$$

tj. jednaka je duljini najkraćeg programa koji na univerzalnom Turingovom stroju kao izlaz proizvodi (ekvivalent od) $\mathbf{x}$ *i zaustavlja stroj.*

K-C složenost se često naziva i *algoritamska složenost*, dok je u našem kontekstu označavamo kao *deterministička složenost*. Pridjev "deterministička" se tu odnosi na činjenicu da je rad UTM-a u potpunosti determiniran, pa je takav i ishod svih programa koji se na njemu izvršavaju.

Deterministička složenost ima svojstvo da je relativno mala za nizove koje je algoritamski jednostavno opisati, a za posve slučajne nizove divergira s porastom duljine niza. Da ovo matematički uobličimo, neka je objekt (npr. Markovljev lanac) niz simbola $\mathbf{x} = s_0 s_1 \ldots s_{l-1}$ duljine l, s omjerom entropije H_μ za pojavu pojedinog simbola. Veza između omjera entropije H_μ i entropije niza $\mathbf{x}$ je sljedeća:

$$H_\mu = \lim_{l\to\infty} \frac{1}{l} H(\mathbf{Pr}(\mathbf{x})) = \lim_{l\to\infty} \frac{1}{l} H(\mathbf{Pr}(s_0 s_1 \ldots s_{l-1})) \ . \tag{4.11}$$

Tu smo s $\mathbf{Pr}(\mathbf{x})$ označili distribuciju vjerojatnosti[3] za pojavu nizova duljine l. Omjer entropije predstavlja "brzinu" kojom sustav odašilje informaciju, odnosno prosječnu informaciju po znaku. Za omjer rasta K-C složenosti vrijedi analogni izraz:

$$\frac{K(s_0 s_1 \ldots s_{l-1})}{l} \underset{l\to\infty}{\longrightarrow} H_\mu \ ,$$

[3]Distribucije vjerojatnosti ćemo po analogiji s prikazom vektora označavati masno pisano kao $\mathbf{Pr}$, a iznos funkcije vjerojatnosti obično pisano, tj. kao $\Pr$.

koji je posljedica formalne podudarnosti teorije K-C složenosti s teorijom informacija temeljenoj na Shannonovoj entropiji[16]. Iz prethodnog je izraza jasno da će za entropiju $H_\mu > 0$, K-C složenost divergirati i biti proporcionalna entropiji niza

$$K(s_0 s_1 \ldots s_{l-1}) \underset{l\to\infty}{\longrightarrow} H_\mu \times l = H(\mathbf{x}) . \tag{4.12}$$

Glavna je konceptualna novina K-C složenosti u odnosu na entropiju njena neovisnost o distribuciji vjerojatnosti. To omogućuje da se pojam količine informacije proširi i na determinističke sustave. Također, ovakvim pristupom se dobiva dublji algoritamski uvid u vjerojatnosne mehanizme stohastičkih procesa. S druge strane, ma koliko gornja dva izraza bila fascinantna u smislu povezivanja teorije automata sa statističkim pristupom, za slučaj vjerojatnosno ovisnog sustava proizlazi da K-C kompleksnost ne predstavlja kvantitativno novu mjeru za informaciju sustava u odnosu na entropiju.

Drugim riječima, ako u procesu P postoji neki stohastički izvor informacije, deterministička složenost će divergirati s povećanjem duljine promatranog niza usprkos njegovoj strukturalnoj jednostavnosti. Npr., može se raditi o bacanju novčića. K-C složenost, pored prave "algoritamske" složenosti objekta (koja najčešće ne ovisi bitno o duljini niza), predstavlja i njegovu *mjeru stohastičnosti* (koja je linearno ovisna o duljini niza). Zbog toga će za velike vrijednosti l stohastička komponenta složenosti prevladati nad bitovima koji predstavljaju algoritamsku ovisnost. Problem, naravno, leži u podrazumijevajućem determinističkom konceptu. Prema definiciji K-C složenosti, tražimo najkraći program koji će točno, u determinističkom smislu, reproducirati niz $\mathbf{x}$. Ukoliko je niz potpuno stohastičan tada, zbog odsustva svake pravilnosti, složenost algoritama raste s povećanjem niza.

4.2.2 Koncept statističke složenosti

U prethodnom pododjeljku je obrazložena manjkavost deterministične K-C složenosti za opis strukture stohastički uvjetovanih sustava. Crutchfield u svojim radovima [1, 10] predlaže novi koncept, koji naziva *statistička složenost* (engl. *statistical complexity*). Glavna je razlika u odnosu na determinističku složenost ta, da se od iznosa $K(\mathbf{x})$ oduzimaju bitovi utrošeni na "računarski napor" UTM-a za simulaciju slučajno generiranih bitova u $\mathbf{x}$. Možemo reći da se odustaje od (uzaludne) ideje da se potpuno slučajan uzorak algoritamski točno reproducira. Umjesto toga, slučajan uzorak je dovoljno statistički analizirati i oponašati ga dodatnim izvorom stohastičnosti (vidjeti odjeljak 4.3).

Tri su intuitivno jasna postulata koji određuju granično ponašanje statističke složenosti C_μ te određuju ovu veličinu kao pogodnu za kvantifikaciju strukture:

1. Za *statističku složenost potpuno stohastičkog objekta* $\mathbf{x}_{stoh}$ *vrijedi:*

$$C_\mu\left(\mathbf{x}_{stoh}\right) = 0\,.$$

Takav objekt nema strukturu već je on jednostavno nasumičan. Za razliku od toga K-C složenost je $K\left(\mathbf{x}_{stoh}\right) > 0\,,$ *i divergira uz produljenje niza.*

2. *Za trivijalne periodičke procese, kao npr.* $\mathbf{x}_{fix} = 111\ldots1\,,$ *također vrijedi*:

$$C_\mu(\mathbf{x}_{fix}) = 0\,.$$

Ovaj objekt nema strukturu, odnosno struktura mu je potpuno monotona. To se podudara s K-C složenosti, tj. $K\left(\mathbf{x}_{fix}\right) = 0\,.$

3. *Za sustave sa svojstvima različitim od prijašnja dva granična slučaja, statistička složenost je općenito veća od nule:*

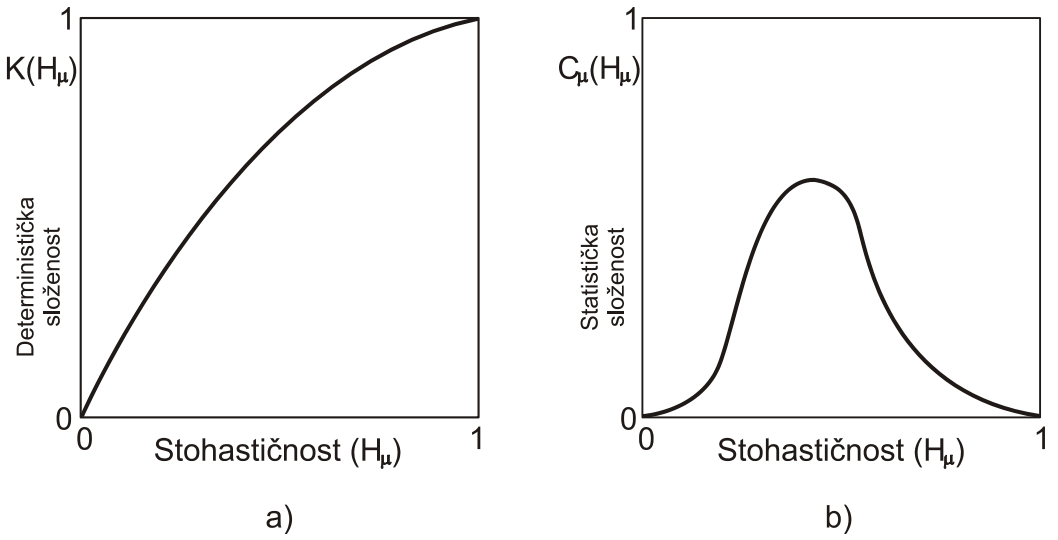
$$C_\mu\left(\mathbf{x}\right) > 0\,,$$

te za neke vrijednosti parametara sustava poprima svoju maksimalnu vrijednost.

Grubi aproksimativni odnos determinističke i statističke složenosti dan je sljedećim izrazom:

$$K\left(s_0 s_1 \ldots\, s_{l-1}\right) \approx C_\mu\left(s_0 s_1 \ldots\, s_{l-1}\right) + \, H_\mu \times l\,. \tag{4.13}$$

Kao što je već naglašeno, oduzimanjem "stohastičkih bitova" iz K-C složenosti, zadržavamo samo strukturalno relevantne bitove koji odgovaraju statističkoj složenosti. Ovaj odnos je prikazan na slikama 4.2 a i b, gdje su veličine $K\left(\mathbf{x}\right)$ i $C_\mu\left(\mathbf{x}\right)$ prikazane u ovisnosti o stupnju stohastičnosti procesa izraženom s omjerom entropije H_μ. Uočljivo je da $K\left(\mathbf{x}\right)$ monotono raste s H_μ, tj. ne predstavlja kvalitativno novu informaciju naspram H_μ. Nasuprot tome, $C_\mu\left(\mathbf{x}\right)$ ima maksimum između točaka nulte i maksimalne stohastičnosti procesa. Za binarnu abecedu, koja se podrazumijeva na slici, $H_{\mu\,\max} = 1\,\mathrm{bit}\,,$ što možemo nazvati *idealna stohastičnost* (engl. *ideal randomness, random oracle*).



Slika 4.2: Usporedba determinističke složenosti $K(X)$ (sl. a) i statističke složenosti $C_\mu(X)$ (sl. b) u ovisnosti o stohastičnosti procesa izraženoj kroz omjer entropije $H_\mu(X)$. Statistički gledano, potpuno stohastičan proces je krajnje jednostavan, pa vrijedi $C_\mu(H_\mu = 1) = 0$. Za razliku od toga, K-C kompleksnost ocjenjuje takav proces kao maksimalno složen, jer će program potreban da UTM reproducira sve duže nizove divergirati. S druge strane, na BTM-u (sl. 4.3) će odgovarajući program biti duljine nula, jer ustroj takvog sustava možemo simulirati izravnim povezivanjem "toplinske kupke" s izlaznom trakom.

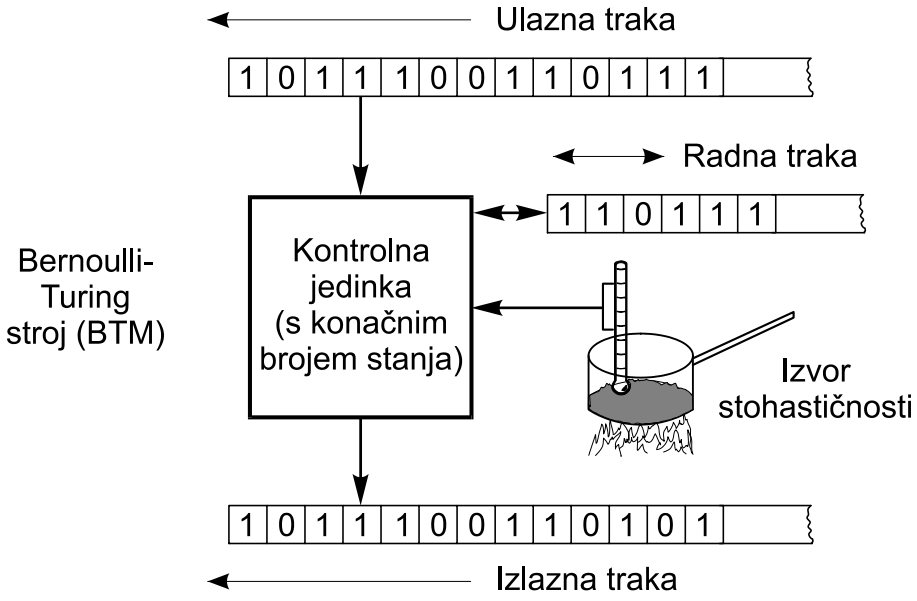
Druga, vrlo sugestivna interpretacija je sljedeća: *statistička složenost je minimalna količina informacije prikupljena u povijesti sustava, koja je dovoljna za optimalno predviđanje simbola (bitova) u objektu* $\mathbf{x}$ *s neizvjesnosti jednakom omjeru entropije* H_μ.

Za potpuno stohastički sustav (granični slučaj 1) koji se sastoji u bacanju novčića, vrijedi da je $H_\mu = 1\,\text{bit}$. Količina informacije koju trebamo izvući iz povijesti da bismo s neizvjesnosti od $1\,\text{bit}$ predviđali buduća stanja je nula, ili drugim riječima, možemo odustati od traženja strukture u prošlosti.

Zaključujemo da je statistička složenost upravo komplementarna K-C složenosti i Shannonovoj entropiji. Također, C_μ je eksplicitno povezana s računarstvom i induktivnim zaključivanjem te odražava računski ustroj objekta.

4.2.3 Metrika statističke složenosti

U prethodnom izlaganju istaknuta su svojstva statističke složenosti u odnosu na deterministički ili algoritamsku složenost. Dodamo li univerzalnom Turingovom stroju zasebni stohastički mehanizam koji će služiti za "pogađanje", odnosno za



Slika 4.3: Prikaz Bernoulli-Turingovog stroja (BTM). BTM nastaje tako da univerzalnom Turingovom stroju (UTM) dodamo izvor stohastičnosti za simuliranje slučajno generiranih bitova. U našem slučaju je to ostvareno sondom uronjenom u kipuću vodu. Na taj način odustajemo od deterministčkog opisa npr. sustava u kaotičnom režimu. UTM bi zahtijevao općenito beskonačni računarski napor za reprodukciju takvih sustava, dok će BTM simulirati kaotično ponašanje iz svojeg vlastitog izvora slučajnih brojeva.

tvorbu Bernoullijevog niza znakova, dobivamo stroj koji ćemo nazvati *Bernoulli-Turingov stroj* (*Bernoulli-Turing Machine*), u daljnjem tekstu BTM (sl. 4.3).

Otuda, po analogiji s deterministčkom složenosti slijedi:

Definicija 4.2 *Statistička složenost* $C_\mu(\mathbf{x})$ *nekog niza* $\mathbf{x}$ *se definira kao*

$$C_\mu(\mathbf{x}) = \min_{prog\,:\,BTM(prog)=\mathbf{x}} length\,(prog)\,,$$

tj. jednaka je duljini najkraćeg programa koji na Bernoulli-Turingovom stroju kao izlaz proizvodi (ekvivalent od) $\mathbf{x}$ *i zaustavlja stroj.*

Uz ranije uvedenu vektorsku notaciju, te uz uvođenje vektora vokabulara $|\mathbf{UTM}\rangle$ i $|\mathbf{BTM}\rangle$ za univerzalni, odnosno Bernoulli-Turingov stroj, izraze za deterministčku i statističku složenost možemo pregledno zapisati kao:

$$K(\mathbf{x}) = ||\, M_{\min}\,\langle \mathbf{x}|\mathbf{UTM}\rangle\,||\,, \tag{4.14}$$

$$C_\mu(\mathbf{x}) = ||\, M_{\min}\,\langle \mathbf{x}|\mathbf{BTM}\rangle\,||\,. \tag{4.15}$$

Gornje definicije statističke složenosti nisu primjerene za praktičnu uporabu, osim u jednostavnim i graničnim slučajevima kad je lako razlučiti između stohastički i algoritamski uvjetovanih sastavnica sustava. Operativni izrazi za dobivanje statističke složenosti proizići će iz konkretnog rekonstruiranog modela i njima odgovarajućih (stohastičkih) matrica, kao što je već nagoviješteno (također vidjeti 4.3.6).

4.3 ϵ-strojevi

Zbirno gledano, da bi se teorija formalnih jezika, automata i izračunavanja uspješno primjenjivala na modeliranje sustava iz stvarnog svijeta, Crutchfield uočava tri nužne odrednice koje moraju biti uključene u njih kao kvalitativno novi aspekti[10, 7]:

i. Stohastičnost (vjerojatnosni aspekti sustava),

ii. Induktivno zaključivanje (uvođenje inventivnog koraka pri modeliranju),

iii. Uvođenje višedimenzionalnosti (opis prostornih fenomena).

Zahtjev (i) smo, kao što je upravo istaknuto, već uvažili — na aksiomatskoj razini uvođenjem BTM-a, a na konkretnoj razini uvođenjem SFA. Zahtjev (iii) predstavlja posebno područje koje se bavi problemima aproksimacije prostornog kontinuuma što se načelno rješava modelima *povezanih ćelija*[4]. Jedan takav pristup je i uvođenje *staničnih automata* (engl. *cellular automata*)[17]. Kod staničnih automata stanice s diskretnim stanjima su raspoređene u prostoru na određeni način (najjednostavniji je raspored u liniju), i između njih je uvedena neka jednostavna interakcija (npr. s najbližim susjedom). Time se reproducira razvoj prostornih struktura.

Mi se još osvrćemo na ispunjenje zahtjeva (ii). U analizi mjernog kanala već smo naglasili nužnost iterativne prilagodbe instrumenta i postepeno poboljšanje modela. Moć induktivnog zaključivanja je presudna u spoznajnom procesu. U nastavku ovog odjeljka obrazložit ćemo prve induktivne korake u izgradnji modela što predstavlja temelj teorije ϵ-strojeva.

4.3.1 Uzročna stanja

U odjeljku 4.1 o mjernom kanalu zaključili smo da se rezultat mjerenja svodi na nalaženje indeksa i podjeljka π_i u danom mjernom prostoru, gdje je $i \in N = \{0, 1, \ldots, n_{\mathcal{I}} - 1\}$, a $n_{\mathcal{I}}$ je vezan za dimenzionalnost i točnost mjerenja: $n_{\mathcal{I}} \geq \varepsilon^{-\mathrm{Dim}}$. Prema našem formalnom pristupu je $n_{\mathcal{I}} = a$, i taj se indeks kodira kao znak:

$$s = s_i \in \{s_0,\, s_1, \ldots, s_{a-1}\} = \mathcal{A}\,, \tag{4.16}$$

iz abecede $\mathcal{A}$. Kodirana mjerenja s su (posredni) pokazatelji stanja ξ promatranog sustava

$$\xi \in \Xi = \{\xi_0, \xi_1, \ldots, \xi_{v-1}\}\,, \tag{4.17}$$

gdje je Ξ skup svih mogućih stanja, uz $v = \mathrm{card}\,(\Xi)$. Cilj modelara je iznaći ta skrivena stanja iz niza mjerenja $\mathbf{s} = s_0 s_1 s_2 \ldots$, i opisati ih "čim vjernijim" modelom. Vjeran model će biti onaj u kojem pojedina stanja automata, uključujući njihove moguće prijelaze i odgovarajuće vjerojatnosti (vidjeti odjeljak 3.4), dobro odgovaraju stvarnim stanjima.[4]

U svezi s time definiramo pojam *uzročnog stanja* (engl. *causal state*)[1]:

Definicija 4.3 *Neka je zadan vremenski niz podataka*

$$\mathbf{s} = \ldots\, s_{t-2}\, s_{t-1}\, s_t\, s_{t+1}\, s_{t+2} \cdots\,,$$

uz $s_i \in \mathcal{A}$, *gdje su pojedini podaci (mjerenja) indeksirani s obzirom na diskretni vremenski trenutak* $t \in \mathbb{Z}$. *Tada* $\mathbf{s}$ *možemo podijeliti na vremenski slijed*[5] *unaprijed*

$$\mathbf{s}_t^{\rightarrow} = s_t\, s_{t+1}\, s_{t+2} \cdots\,, \tag{4.18}$$

i vremenski slijed unatrag

$$\mathbf{s}_t^{\leftarrow} = \ldots\, s_{t-2}\, s_{t-1}\,. \tag{4.19}$$

Za vremenski slijed $\mathbf{s}_t^{\leftarrow}$ *unatrag, kažemo da je uzročno stanje vremenskom slijedu* $\mathbf{s}_t^{\rightarrow}$ *unaprijed.*

Ovako definirani sljedovi unaprijed (unatrag) predstavljaju informaciju o budućnosti (prošlosti) sustava u odnosu na vremenski trenutak t. Primijetimo da u općenitom slučaju promatramo niz mjerenja koja ne moraju imati dobro definirani početak, što odgovara eksperimentalnoj realnosti glede problema nalaženja faze procesa.

Združene vjerojatnosti (ili tzv. 2-dimenzionalne vjerojatnosti) svih mogućih parova sljedova unatrag i unaprijed čine dobro definiranu distribuciju vjerojatnosti na algebri svih mogućih združenih događaja. Uvedimo sada standardne oznake:

[4] Stvarna stanja, naravno, nikad ne možemo egzaktno dokučiti, ali usporedbom predviđanja koja daje naš model s onim što se stvarno dešava možemo izvršiti testiranje i ocijeniti uspješnost ili neuspješnost izgrađenog modela. Strogoća znanstvenih kriterija u tom smislu može biti potpuna.

[5] U daljnjem tekstu rabimo izraze *vremenski slijed* i *vremenski niz* kao sinonime.

- $\Pr(\mathbf{s}_t^{\leftarrow}, \mathbf{s}_t^{\rightarrow})$ za združenu vjerojatnost pojave para događaja:
  (slijed unatrag $= \mathbf{s}_t^{\leftarrow}$, slijed unaprijed $= \mathbf{s}_t^{\rightarrow}$),
- $\Pr(\mathbf{s}_t^{\rightarrow} | \, \mathbf{s}_t^{\leftarrow})$ za uvjetnu vjerojatnost pojave uvjetnog događaja: [6]
  (slijed unaprijed $= \mathbf{s}_t^{\rightarrow}$ | *pod uvjetom:* prethodio je slijed unatrag $= \mathbf{s}_t^{\leftarrow}$).

Vjerojatnost za pojavu niza $\mathbf{s}$ sada možemo izraziti na sljedeći način:

$$\Pr(\mathbf{s}) = \Pr(\mathbf{s}_t^{\leftarrow}, \mathbf{s}_t^{\rightarrow}) = \Pr(\mathbf{s}_t^{\rightarrow} | \, \mathbf{s}_t^{\leftarrow}) \Pr(\mathbf{s}_t^{\leftarrow}) \, . \tag{4.20}$$

Uz određeni niz $\mathbf{s}_t^{\leftarrow} = \overline{\mathbf{s}}$, možemo promatrati distribuciju uvjetnih vjerojatnosti s obzirom na uzročno stanje $\overline{\mathbf{s}}$:

$$\Pr(\mathbf{s}_t^{\rightarrow} | \, \overline{\mathbf{s}}) \, ,$$

kao vjerojatnosnu mjeru zadanu za sve buduće sljedove $\mathbf{s}_t^{\rightarrow}$ koji su mogući nakon $\overline{\mathbf{s}}$.

4.3.2 Ekvivalentna stanja

Prethodno izlaganje pripremilo je tlo za definiciju *ekvivalentnih stanja*:

Definicija 4.4 *Dva uzročna stanja* $\overline{\mathbf{s}} = \mathbf{s}_t^{\leftarrow}$ *i* $\overline{\mathbf{s}}' = \mathbf{s}_{t'}^{\leftarrow}$ *definirana vremenskim trenucima* $t, t' \in \mathbb{Z}$, *su* δ-***ekvivalentna*** *(*δ-***jednaka****) za neki* $\delta \in \mathbb{R}$, $\delta > 0$, *ako su im uvjetne distribucije vjerojatnosti budućih stanja jednake, što pišemo kao*

$$\overline{\mathbf{s}} \underset{\delta}{\sim} \overline{\mathbf{s}}' \iff |\Pr(\mathbf{s}_t^{\rightarrow} | \, \overline{\mathbf{s}}) - \Pr(\mathbf{s}_t^{\rightarrow} | \, \overline{\mathbf{s}}')| \leq \delta \, , \; \forall \, \mathbf{s}_t^{\rightarrow} \in \mathcal{A}^* . \tag{4.21}$$

U skladu s notacijom u odjeljku 3.3 s $\mathcal{A}^*$ smo označili skup svih nizova proizvoljne duljine, od kojih će mogući biti samo oni kojima je gornja uvjetna vjerojatnost veća od nule. U daljnjem izlaganju ćemo nazive *δ-ekvivalencija* i *ekvivalencija* smatrati sinonimima, odnosno prvi ćemo rabiti u slučajevima kad želimo naglasiti postojanje vjerojatnosnog kriterija.

Uočimo prirodnost ove definicije sa stanovišta modelara. Trenuci t i t', te njima pripadajuća stanja $\overline{\mathbf{s}}$ i $\overline{\mathbf{s}}'$ su ekvivalentni, jer je "paleta" budućih stanja jednaka. Odnosno, preciznije, jednake su uvjetne vjerojatnosti tih budućih stanja naspram trenutnih.

[6] Ova je oznaka uobičajena u teoriji vjerojatnosti i naglašava činjenicu da je s obzirom na zadani uvjet dobro definirana distribucija vjerojatnosti svih mogućih posljedičnih događaja. Ista je oznaka preuzeta i u teoriji informacija, iako je neprikladna s obzirom na standardni matrični prikaz matrice prijelaza u kanalu[16, 18].

Drugi način interpretacije ove definicije mogao bi biti i sljedeći: *modelar ili agent nalazi da su dva stanja u općenito različitim trenucima jednaka, ako je njihovo "znanje" o budućnosti s obzirom na prošlost jednako.*

Pri tom, kao što i proizlazi iz gornje definicije, sama stanja mogu imati različiti tijek razvoja u prošlosti, što opisujemo izrazom $\overline{\mathbf{s}} \neq \overline{\mathbf{s}}'$, a da vjerojatnosti za buduća stanja budu jednaka, odnosno da vrijedi ekvivalentnost: $\overline{\mathbf{s}} \sim \overline{\mathbf{s}}'$. Primijetimo da se implicitno podrazumijeva da je skup mogućih budućih stanja neprazan, tj. da postoji vjerojatnost veća od nule za prijelaz u barem neko od budućih stanja, jer u protivnom gubimo kriterij za usporedbu.

Skup Ξ svih stanja sustava promatramo kao skup razreda ekvivalencije uzročnih stanja $\mathbf{s}_l^{\rightarrow} \in \mathcal{A}^*$ s obzirom na relaciju $\sim$, što pišemo kao

$$\Xi = \{\, \xi_i : \overline{\mathbf{s}},\ \overline{\mathbf{s}}' \in \mathcal{A}^*\,,\ \overline{\mathbf{s}} \underset{\delta}{\sim} \overline{\mathbf{s}}'\ \}\,,\ i = 0, 1, \ldots\ v-1\,. \tag{4.22}$$

Iako skup $\mathcal{A}^*$ označava skup svih zamislivih nizova, u praksi se taj skup omeđuje na nizove s duljinom $l \leq l_{\max}$, gdje je $l_{\max}$ parametar modela vezan uz njegovu točnost. Takav skup bismo mogli označiti dodatnim parametrom, tj. kao $\mathcal{A}^*_{l_{\max}}$, no u slučajevima kad to nije potrebno naglašavati ispuštat ćemo oznaku maksimalne duljine niza.

4.3.3 Definicija morfova

Prethodno izlaganje je podastrijelo formalni okvir za definiciju *morfa* (engl. *morph*):

Definicija 4.5 *Skup M_{ξ_i} svih budućih sljedova $\mathbf{s}_i^{\rightarrow}$ unaprijed koji su mogući iz nekog stanja $\xi_i \in \Xi$,*

$$M_{\xi_i} = \{\, \mathbf{s}_i^{\rightarrow} : \Pr(\mathbf{s}_i^{\rightarrow} | \, \mathbf{s}_i^{\leftarrow}) > 0 \,\}\,,$$

naziva se budući morf (ili kraće jednostavno morf), a skup svih sljedova $\mathbf{s}_i^{\leftarrow}$ unazad, koji su doveli do stanja ξ_i naziva se prošli morf.

Iz definicije 4.4 izravno slijedi da je nužan uvjet za ekvivalenciju dva stanja jednakost njihovih morfova, tj.

$$\xi_i \underset{\delta}{\sim} \xi_j \Longrightarrow M_{\xi_i} = M_{\xi_i}\,,\ \ \xi_i, \xi_j \in \Xi\,. \tag{4.23}$$

Sada možemo preobličiti definiciju 4.4 na sljedeći način:

Definicija 4.6 *Stanja su ekvivalentna ako i samo ako*

i. imaju jednake morfove, i dodatno,

ii. ako su im jednake uvjetne vjerojatnosti za sve sljedove unaprijed.

Također, nameće se i sljedeće određenje:

Definicija 4.7 *Za stanja koja imaju jednake morfove, ali nisu nužno ekvivalentna, kažemo da su* ***morfološki ekvivalentna (jednaka)****, što ćemo označavati kao*

$$\xi_i \underset{L}{\sim} \xi_j \,.$$

Parametar L predstavlja karakteristični parametar (razinu aproksimacije) opisa morfa (npr. dubinu podstabla — vidjeti 4.5.5). Morfološka jednakost predstavlja grublju relaciju, i broj razreda ekvivalencije dobivenih primjenom *morfološke jednakosti* bit će manji ili jednak broju razreda dobivenih primjenom *vjerojatnosne jednakosti* stanja. Ukoliko se zadovoljimo ovom grubljom aproksimacijom, stanja sustava (modela) i morfovi postaju sinonimi.

Primijetimo da iz gornje definicije proizlazi da svaki prošli morf definira jedan i samo jedan budući morf. Prema tome, kao što i intuicija nalaže, uz svaki prošli morf mora biti vezano određeno stanje sustava.

Za stanja koja nisu morfološki ekvivalentna kažemo da su *morfološki različita*. Općenito, ako su stanja morfološki različita, tada po definiciji 4.6 izravno slijedi da stanja nisu δ-ekvivalentna, tj. vrijedi

$$\xi_i \underset{L}{\not\sim} \xi_j \implies \xi_i \underset{\delta}{\not\sim} \xi_j \,, \quad \xi_i \,, \xi_j \in \Xi \,. \tag{4.24}$$

Drugim riječima, u tom slučaju nije ni potrebno provjeravati vjerojatnosti prijelaza.

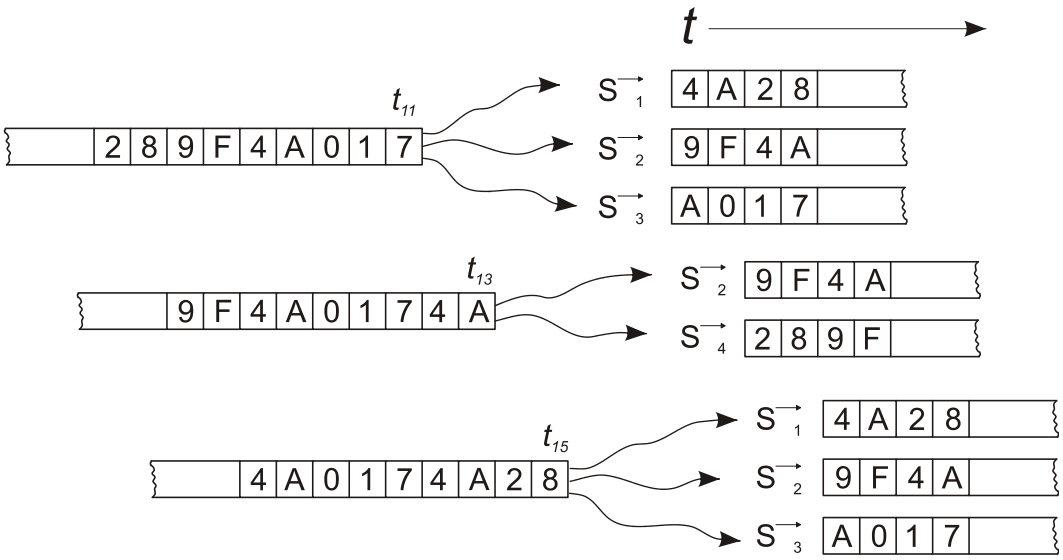
Situacija je zorno prikazana na slici 4.4. Stanja u t_{11} i t_{15} su morfološki ekvivalentna, mada s različitom prošlosti, jer oba stanja imaju jednake izglede za razvoj u budućnosti. To možemo zapisati kao

$$\mathbf{s}_{t_{11}} = \xi_i \,, \;\; \mathbf{s}_{t_{15}} = \xi_j \,, \;\; \xi_i \underset{L}{\sim} \xi_j \,.$$

Stanje u t_{13} opisano sa $\mathbf{s}_{t_{13}} = \xi_k \,, \; k \neq i, j \,,$ je različito — jer nas vodi u različitu budućnost. U tom stanju ne postoji mogućnost za prijelaz u ista stanja kao u slučaju ξ_i i ξ_j. Možemo reći da je stanje ξ_k informacijski minimalno ovisno o ξ_i i ξ_j, jer nudi različitu informaciju o budućnosti.

Nadalje, ukoliko su za ξ_i i ξ_j uvjetne vjerojatnosti prijelaza za sva nova stanja (za sve sljedove unaprijed) jednaka unutar parametra δ, tada su morfološki ekvivalentna stanja ξ_i i ξ_j po definiciji i ekvivalentna stanja:

$$\xi_i \underset{\delta}{\sim} \xi_j \,.$$



Slika 4.4: Prikaz budućih morfova (zakrivljene linije sa strelicama) proizašlih iz općenito različitih uzročnih stanja. Kad su palete budućih mogućnosti i odgovarajuće vjerojatnosti jednake, proces je u istom uzročnom stanju, neovisno o prošlosti (koja je prethodila trenutnom stanju). U trenucima t_{11} i t_{15} uzročna stanja su jednaka, jer su izgledi za budućnost podudarni.

4.3.4 Definicija ϵ-strojeva

Jednom kad se iz uzročnih stanja nekog niza primjenom relacije ekvivalencije ustanove sva ona koja su različita, vremenski razvoj procesa se svodi na preslikavanje iz stanja u stanje, opisano operatorom $\mathcal{T}$

$$\mathcal{T} : \Xi \underset{\overrightarrow{\mathbf{s}_i}}{\longrightarrow} \Xi \qquad (4.25)$$
$$\xi_{t+1} = \mathcal{T}\ (\xi_t\,, \overrightarrow{\mathbf{s}_i})\ .$$

Nalaženje operatora $\mathcal{T}$ se svodi na analizu morfa iz kojeg iščitavanjem uvjetnih vjerojatnosti $\Pr(\overrightarrow{\mathbf{s}_t}|\,\overleftarrow{\mathbf{s}_t})$ dobivamo potpuni uvid u moguće prijelaze i njihove vjerojatnosti. Naravno, uz određenje uvjetnih prijelaznih vjerojatnosti, te uz postupak analogan onom u odjeljku 3.4, možemo djelovanje operatora $\mathcal{T}$ povezati s poopćenom inačicom prijelaznog tenzora $\mathbf{T}$ definiranog izrazom (3.7).

Odavde slijedi definicija ϵ-strojeva:

Definicija 4.8 *Uređeni par* $(\Xi\,, \mathcal{T}\ )_\epsilon$ *nazivamo* ϵ*-stroj.*

Parametar ϵ nas podsjeća da se radi o konstrukciji koja je:

i. zavisna o značajkama mjernog instrumenta, kao što su njegova sposobnost razlučivanja ili točnost (ε), broj eksperimentalnih proba Dim, i sl.;

ii. aproksimacija računarske strukture procesa s pomoću određenog stroja.

Odavde zaključujemo na sljedeća svojstva ϵ-strojeva:

1. ϵ-strojevi daju minimalnu informacijsku ovisnost između morfova, i u tom smislu predstavljaju jezgrovitu (vidjeti točku 2) uzročno-posljedičnu strukturu morfova kao događaja. Strojevi bilježe tijek informacije unutar danog podatkovnog niza. Npr. ako stanje ξ_2 slijedi nakon stanja ξ_1, tada je sa stanovišta modelara ξ_1 uzrok od ξ_2, odnosno ξ_2 je posljedica od ξ_1.

2. Rekonstrukcija ϵ-stroja proizvodi minimalni model — do na grešku predviđanja. Greška predviđanja je u svezi s raspoloživim resursima zaključivanja. Minimalnost, koja proizlazi iz iterativnog postupka gradbe (vidjeti odjeljak 4.4), garantira da ne postoje drugi događaji (morfovi) koji bi mogli razbiti izravnu svezu uzročno posljedičnih stanja ξ_1 i ξ_2 opisanih u točki 1.

3. Ako je ξ_1 uzrok od ξ_2, tada je tok informacija od ξ_1 ka ξ_2, a količina prenesene informacije je vlastita informacija vezana uz prijelaznu vjerojatnost $\Pr(\xi_2|\,\xi_1) = p_{\xi_1\to\xi_2}$, tj. vrijedi $I_{\xi_1\to\xi_2} = -\operatorname{ld} p_{\xi_1\to\xi_2}$.

4.3.5 Omjer entropije H_μ

U odjeljku 4.2 iznijeli smo formalne definicije deterministične i statističke složenosti, te općenito uveli razmatranje statističkih svojstava sustava. Uvažavamo oznake iz spomenutog odjeljka i podrazumijevamo da poznajemo ϵ-stroj $(\Xi, \mathcal{T})_\epsilon$.

Da bismo izračunali omjer entropije H_μ načelno bismo trebali poznavati distribuciju vjerojatnosti $\mathbf{Pr}(\mathbf{x}^\infty)$ za nizove beskonačne duljine $\mathbf{x} = s_0 s_1 \ldots s_i \ldots$. Iz nje bi slijedile distribucije $\mathbf{Pr}(\mathbf{x}^l)$ za nizove duljine l kao marginalne distribucije. Pošto je distribucija $\mathbf{Pr}(\mathbf{x}^\infty)$ nepoznata kod sustava za koje nemamo drugih podataka osim onih dobivenih eksperimentalno, izraz 4.11 nije od koristi. Umjesto toga, promatramo distribuciju uvjetnih vjerojatnosti $\mathbf{Pr}(s_{t+1}|\,\mathbf{x}_t^{\leftarrow})$ za pojavu simbola $s_{t+1} \in \mathcal{A}$ uz uvjet da mu je prethodio slijed $\mathbf{x}_t^{l\leftarrow}$ unatrag duljine l. Vjerojatnost za pojavu niza $\mathbf{s} = \mathbf{x}_t^{l\leftarrow} s_{t+1}$ (niz $\mathbf{x}_t^{l\leftarrow}$ unatrag sa sufiksom s_{t+1}) iznosi

$$\Pr(\mathbf{s}) = \Pr(\mathbf{x}_t^{l\leftarrow},\, s_{t+1}) = \Pr(s_{t+1}|\,\mathbf{x}_t^{l\leftarrow})\,\Pr(\mathbf{x}_t^{l\leftarrow}). \tag{4.26}$$

Odavde se lako pokazuje (vidjeti npr. [18]) da granična vrijednost entropije uvjetne distribucije konvergira prema omjeru entropije H_μ za $l \to \infty$, što pišemo kao

$$\lim_{l\to\infty} H\left(\mathbf{Pr}\left(s_{t+1}|\,\mathbf{x}_t^{l\leftarrow}\right)\right) = \lim_{l\to\infty} \frac{1}{l}\, H\left(\mathbf{Pr}\left(\mathbf{x}^l\right)\right) = H_\mu\,. \tag{4.27}$$

Svaki niz $\mathbf{x}_t^{l\leftarrow}$ unatrag je po definiciji 4.5 vezan uz određeno stanje sustava $\xi \in \Xi$, tako da na temelju našeg modela omjer entropije H_μ računamo kao

$$H_\mu = H\left(\mathbf{Pr}\left(s_{t+1}|\,\xi_t\right)\right) = H\left(\mathbf{Pr}\left(s_i|\,\xi_j\right)\right)\,. \tag{4.28}$$

Ovdje je $s_i \in \mathcal{A}$ idući simbol koji se pojavljuje nakon nekog od stanja ξ_j iz Ξ.

4.3.6 Mjere složenosti kao rezultat modeliranja

Pored omjera entropije kao uobičajenog statističkog pokazatelja u teoriji informacija, ovdje ćemo operacionalizirati izračunavanje mjera strukturalne složenosti koje slijede izravno iz parametara rekonstruiranih ϵ-strojeva. Kao što je već spomenuto, definicija statističke složenosti 4.2 nije od velike praktične koristi pošto ne nudi konkretne metode za njeno određivanje.

Ono što se nameće kao prva aproksimacija je *topološka složenost* (engl. *topological complexity*) C_0 procesa, definirana s

$$C_0 = \mathrm{ld}\,||\,\Xi\,|| = \mathrm{ld}\left(\mathrm{card}\left(\Xi\right)\right) = \mathrm{ld}\,v\,. \tag{4.29}$$

Ovdje smo sa $||\,\Xi\,||$ označili kardinalni broj skupa stanja kao najjednostavniju mjeru na tom skupu. Topološka složenost, dakle, vodi računa samo o broju stanja. Drugim riječima, ona ima značenje maksimalne informacije potrebne da se ustanovi u kojem je stanju proces — što odgovara situaciji u kojoj su sva stanja jednako vjerojatna.

No rekonstrukcija ϵ-stroja nam daje i operator $\mathcal{T}$ iz kojeg proizlazi prijelazni tenzor $\mathbf{T}$. Ukoliko nas zanimaju samo vjerojatnosti prijelaza između pojedinih stanja, neovisno po kojem ulaznom simbolu su se prijelazi desili, tada iz $\mathbf{T}$ izvađamo matricu prijelaza T:

$$T = \sum_{s\in\mathcal{A}} \mathbf{T}^{\,s}_{qq'}\,. \tag{4.30}$$

Sumacija je po svim simbolima abecede $\mathcal{A}$, tako da se dobiva ukupna vjerojatnost za svaki od prijelaza $q \to q'$, $q, q' \in Q$. Matrica T dimenzija $v \times v$ predstavlja projekciju tenzora $\mathbf{T}^{\,s}_{qq'}$ s prostora $\Xi \times \Xi \times \mathcal{A}$ na prostor $\Xi \times \Xi$, i ekvivalentna je matrici prijelaza za Markovljeve procese. Za vremenski invarijantne Markovljeve

procese presudni je statistički pokazatelj *stacionarna* (*asimptotska*) *distribucija vjerojatnosti* $\mathbf{p}_{st}$, definirana uvjetom

$$\mathbf{p}_{st}\, T = \mathbf{p}_{st}\,. \tag{4.31}$$

$\mathbf{p}_{st} = (p_0, p_1, \ldots, p_{v-1})$ figurira ovdje kao lijevi svojstveni vektor matrice T, za svojstvenu vrijednost 1 kao jedinu koja zadržava svojstvo normiranosti distribucije: $\sum_{i=0}^{v-1} p_i = 1$.

Mjera složenosti modela je upravo entropija ove stacionarne distribucije. Ona predstavlja **statističku složenost** danog ϵ-stroja kao modelarovu prosudbu složenosti procesa:

$$C_\mu = H\left(\mathbf{p}_{st}\right)\,. \tag{4.32}$$

Gradba ϵ-strojeva podrazumijeva insistiranje na minimalnosti modela, pa ako je rekonstrukcija uspješna glede tog kriterija, tj. ako veličina modela izražena kao $||\,\Xi\,||$ ne divergira, tada C_μ predstavlja količinu memorije nužnu da modelar predvidi stanje procesa na danoj ϵ-razini točnosti.

U zaključku ovog odjeljka istaknimo da je izrada modela iz vremenskog niza mjerenja dinamički proces, u kojem prateći dvije veličine — entropiju sustava H_μ i statističku složenost C_μ, poboljšavamo svojstva modela. Što je razlika $|H_\mu - C_\mu|$ manja, to je manji omjer krivog predviđanja. Ako se C_μ približi H_μ, izgledi za uspješnost modela se povećavaju. Da se to postigne, modelar mora svoja računarska izvorišta posvetiti gradbi modela. Također, primijetimo da pošto C_μ odražava strukturalnu složenost realnog sustava uz koji je uvijek pridodan i izvor stohastičnosti, vrijedi da je $C_\mu \leq H_\mu$. Drugim riječima, za izrazito stohastičke sustave strukturalna komponenta složenosti može biti relativno mala.

4.4 Algoritam gradbe ϵ-strojeva

Pojmovi uvedeni u prethodnom odjeljku u uskoj su svezi s algoritmom gradbe ϵ-strojeva, koji uključuje postepeno, inkrementalno usavršavanje modela. Razvoj modela je kao i njegova statistika kvazistacionaran te se događa na dvije razine. Kao prvo, uključuje poboljšanje parametara modela primanjem novih podataka u okviru nekog odabranog razreda strojeva, a kao drugo, sadrži i inovativni korak povezan s promjenom (poboljšanjem) primijenjenog razreda. Promjena razreda ne mora nužno značiti i povećanje računarske sofisticiranosti razreda modela. Na slici 3.1 smo već

diskutirali kako ne postoji jedinstveni put prema složenijim strojevima teorije izračunavanja. Uređaj automata nije bio strogo hijerarhijski i nekompatibilni razredi nisu bili izravno sumjerljivi. U teoriji ϵ-strojeva to će biti zamijenjeno hijerarhijskim uređajem na kojem se temelji inovativnost modeliranja.

Inovativnost možemo opisati kao unapređenje i poboljšanje modelarovog (agentovog) poimanja procesa (okoline) kroz uočena uzročna stanja. Primijetimo da pojam inovativnosti nije vezan isključivo na humanog modelara. Stoga u širem kontekstu govorimo o agentu koji u sklopu neke fizikalne, kemijske ili biološke okoline razaznaje procese. Inovativnost je pogotovo nužna u živim sustavima suočenim s neminovnošću borbe za opstanak. Inovativnost se nadalje može povezivati sa širim pojmovima inteligencije i kreativnosti.[7]

4.4.1 Hijerarhijska rekonstrukcija

ϵ-strojevi kao tipično računarske tvorevine (automati) predstavljaju funkcionalne strojeve za razaznavanje formalnih jezika te, shodno tome, bacaju svjetlo na njihovu gramatiku (pogl. 3). Rekonstrukciju ϵ-strojeva možemo, dakle, promatrati i kao rekonstrukciju jezika kojim "govori" proces, odnosno okolina. Rekonstrukcija je hijerarhijska. Na dnu te hijerarhije su najjednostavniji jezici, npr. binarni podaci sami, koji su opisani najjednostavnijim (trivijalnim) automatima sa stanjima korespondentnima samom vremenskom nizu. Idući prema vrhu hijerarhije nailazimo na sve složenije automate i njima odgovarajuće jezike.

Ovaj zahtjev stoji u opreci sa situacijom u teoriji automata gdje ne postoji stroga hijerarhijska uređenost. Mnogobrojnost i raznovrsnost strojeva rezultira njihovom općenitom nekompatibilnošću. Da bi se ostvarila konzistentnost rekonstrukcije ϵ-strojeva, Crutchfield uvodi strogu hijerarhiju[10], tj. za predstavnike razreda odabire automate s međusobno dobro definiranim hijerarhijskim odnosom. Na neki način to možemo opisati kao izbor povoljnih baznih vektora u prostoru induktivnog zaključivanja. Hijerarhija računskih modela i ϵ-strojeva prikazana je na sl. 4.5.

Modeli predstavljeni na slici 4.5 sastoje se od stanja (krugovi ili kvadrati) i prijelaza (strelice s oznakama). Svaki model ima jedinstveno početno stanje označeno

[7] Ono što se nameće kao filozofska implikacija gore rečenog je pridavanje inovativnih osobina, a posljedično i *određene razine kreativnosti i inteligencije*, ne samo živim, već i tipično fizikalnim ili kemijskim sustavima. Kristalne strukture, tvari u dinamici faznih prijelaza, ali i općenita interakcija fizikalnih objekata, primjeri su za to. Kod toga se inovativnost u našem smislu suštinski razlikuje od *slučajnih* evolucijskih procesa koji se javljaju u npr. genetičkim algoritmima. Navedena tema otvara ideje koje prelaze zadaću ovog rada, što će eventualno biti izloženo drugdje.

upisanom kružnicom.

Najnižu, nultu, razinu hijerarhije predstavlja sam podatkovni niz. Nadalje, na razini jedan se iz podatkovnog niza uzimaju svi podnizovi konstantne duljine D, koje u obliku staza uvađamo u raščlambeno stablo visine (dubine) D. Na razini modeliranja broj dva, unutar raščlambenog stabla uočavamo morfove u obliku podstabala određene duljine L ($L \approx D/2$). Morfovi predstavljaju osnovu za raspoznavanje stanja u novom razredu stohastičkih konačnih automata, čiji nam statistički parametri omogućuju nalaženje statističke složenosti (vidjeti prethodni odjeljak). Najviša razina prikazana na slici su *strojevi za proizvodnju nizova* (engl. *string production machines*) s kraticom *PM*. Oni se dobivaju grupiranjem stanja u FA za koja se uočava pravilnost u pridruženim im nizovima. Za te se nizove nalaze proizvodna pravila P koja služe za upravljanje tvorbom nizova u registru stroja. Moguće više razine modela naznačene su točkicama.

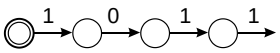
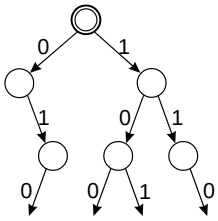
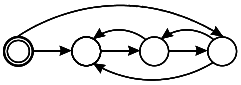
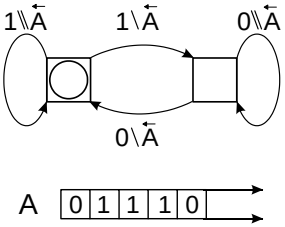
Ono što je specifično za teoriju ϵ-strojeva jest raščlamba vremenskog niza kombinacijom računarskog pristupa u vidu analize formalnih jezika, uz dodatne statističko-informacijske aspekte. Tako se riječi (podnizovi) unutar primljenog vremenskog niza raščlanjuju na način svojstven prevodiocima — s pomoću raščlambenog stabla, ali se istovremeno gradi i statistika procesa, što će detaljnije biti izloženo u sljedećem odjeljku (vidjeti 4.5).

Hijerarhijska rekonstrukcija ϵ-strojeva može se sada opisati u dva koraka:

I. Započinje se na najnižoj razini računarske hijerarhije, gradbom stohastičkih konačnih automata (SFA). Tu su uključene tri razine (prikazane na slici 4.5), od 0 do 2, kako slijedi:

   0. podatkovni niz
   1. stabla
   2. stohastički konačni automati

II. Ako na bilo kojoj od razina obrazloženoj u (I) dolazi do divergencije modela — bez obzira na uporabu sve većih računarskih izvorišta za poboljšanje točnosti modela (npr. povećanjem duljine nizova D) — tada je potrebno iznaći novi, viši razred modela u okviru *inovacijskog koraka*. Kriterij za poticanje inovacijskog koraka je iscrpljivanje računarskih resursa. Pod divergencijom modela podrazumijevamo da s porastom broja primljenih podataka model raste po svojoj veličini (broju stanja) bez granica.

Razina	Razred modela	S T R O J [računski model]	Veličina modela	Relacija ekvivalencije
0	Podatkovni niz		n	Rezultat mjerenja
1	Stablo		$\mathcal{O}'(\|\|A\|\|^D)$	Nezavisnost blokova
2	Stohastički konačni automat	 [Temeljni ϵ-stroj]	$\mathcal{O}'(\|\|T\|\|)$	Uvjetna nezavisnost
3	Stroj za proizvodnju nizova		$\mathcal{O}'(\|\|T\|\| + \|\|P\|\|)$	Konačno-rekurzivna uvjetna nezavisnost
. . .	. . .	. . .	. . .	. . .
. . .	. . .	. . .	. . .	. . .

Slika 4.5: Hijerarhija računskih modela i ϵ-strojeva. Hijerarhija odražava iterativnost postupka modeliranja na temelju vremenskog niza. Na svakoj je razini model predstavljen određenim razredom modela, odnosno automata. Model se sastoji od stanja označenih kružićima ili kvadratićima. Početno stanje označeno je upisanim kružićem. Podaci su početna (nulta) razina iz koje na razini 1 slijede stabla dubine D. Iz stabala se nalaženjem morfova određuje stohastički konačni automat (SFA), koji predstavlja ϵ-stroj temeljne razine. Veličina modela tu je označena kao mjera dobivena iz tenzora T, u kojem je sadržana informacija o svim stanjima i njihovim prijelazima. Iznad SFA je razina strojeva za proizvodnju nizova (PM). (Prema izvorniku u [10].)

4.4.2 Inovacijski korak

Inovacijski korak predstavlja tzv. evolucijsku dinamiku, koja se promiče od manje prema više uspješnim razredima modela na temelju sličnosti i simetrija u organizaciji stanja nižih razreda, te u njihovom odgovarajućem grupiranju (sažimanju).

Promotrimo to detaljnije na sljedećim primjerima. Neka je primljen niz $\mathbf{s}$ duljine m, koji se sastoji od m mjerenja. Ako je izvor periodički, tada je model 0-te razine, tj. podaci sami, ovisan o duljini m. Za $m \to \infty$ ovaj "model" divergira. U stvari, ovu razinu ni ne možemo nazvati svrsishodnim modelom. Ona niti za trivijalni primjer čvrste točke ne daje konačan model, odnosno ne vrši adekvatno sažimanje podataka i ne ostvaruje mogućnost predviđanja kako to očekujemo od modela. S druge, formalne strane, uočimo da je 0-ti model kao jednostavna replika podataka samih, ujedno i njihov najtočniji prikaz.

Pređemo li na višu razinu, te uzmemo model stabla dubine D, gdje je D veće od perioda izvora, tada će raščlambeno stablo predstavljati točan i konačan model procesa iako su podaci beskonačne duljine. Stanja u ovom modelu su sve staze duljine D određene dinamikom zadanog perioda, gdje svaka staza odgovara drugoj fazi unutar ponavljajućeg uzorka u nizu $\mathbf{s}$.

Nadalje, ako je niz $\mathbf{s}$ neperiodičan, razred stabala više neće dati konačan model neovisan o duljini m. Uz entropiju izvora $H_\mu > 0$ veličina modela će biti proporcionalna a^{DH_μ}, gdje je $n_{\mathcal{I}} = a$ broj podjeljaka $n_{\mathcal{I}} = {}^{\ulcorner}\varepsilon^{-\mathrm{Dim}\,\urcorner}$ prema (4.2). Kako m raste, modelar mora uzimati sve veće dubine D stabla, jer je izvor neperiodičan, te bi i veličina modela rasla eksponencijalno.

Ukoliko u gore spomenutom neperiodičkom izvoru postoje tranzijentne pojave, odnosno korelacije koje trnu s vremenom relativno brzo, tada će sljedeća razina (2) stohastičkih konačnih automata dati konačnu reprezentaciju procesa. Broj stanja $n_Q = ||Q|| = \mathrm{card}(Q)$ u SFA predstavlja mjeru utrošene memorije modela, odnosno pokazatelj je količine memorije koju je modelar razaznao u promatranom sustavu.

Ta veličina memorije je izravno proporcionalna maksimalnoj duljini korelacije koja se može očekivati u sustavu. Označimo s t_{corr} *vrijeme korelacije* izraženo brojem diskretnih vremenskih intervala $\triangle t = t_{i+1} - t_i$. Jasno je da se ono može izraziti preko broja pronađenih stanja o čemu govori sljedeći izraz:

$$t_{corr} \lessapprox \frac{\mathrm{ld}\, n_Q}{\mathrm{ld}\, a} \,. \tag{4.33}$$

Npr. uz grubu izmjeru i binarno kodiranje ($a = 2$), maksimalno je vrijeme korelacije $t_{corr\,\max} \approx \mathrm{ld}\, n_Q$, što odgovara redu veličine broja operacija potrebnih za pretraži-

vanje usmjerenog grafa oblika stabla.

4.4.3 Odrednice razvoja modela

Kao bitne odrednice koje nam pomažu u analizi uspješnosti, odnosno neuspješnosti postojeće razine modela, te u postavljanju kriterija i samom izvođenju inovacijskog koraka, navedimo sljedeće:

1. Simetrije – koje odražavaju modelarove pretpostavke o strukturi okoline, iz kojih proizlazi semantički sadržaj pojedinog modelarnog razreda $\mathcal{M}_R$. Taj sadržaj, odnosno značenje, proizlazi iz relacije ekvivalencije $(\sim)$ za danu simetriju.

2. Modeli M, unutar razreda $\mathcal{M}_R$, koji se sastoje od stanja i prijelaza rekonstruiranih kroz mjerenja.

3. Formalni jezici – kao skupine konačno-predstavljivih ponašanja procesa.

4. Rekonstrukcija – kao postupak stvaranja željenog (procijenjenog) modela $M \in \mathcal{M}_R$, što formalno označavamo kao $M = \mathbf{s} \,/\sim$. Tj. rekonstrukcija iz ulaznih nizova $\mathbf{s}$ izlučuje simetrije glede relacije $\sim$.

5. Složenost (topološka) procesa, kao veličina minimalnog modela M unutar razreda $\mathcal{M}_R$ rekonstruiranog iz primljenog niza: $C\,(\,\langle \mathbf{s}|\mathcal{M}\rangle) = ||M_{\min}||\,.$

6. Predvidljivost – utvrđenu u odnosu na razlučiva stanja i izraženu s $H_\mu = H(\,\mathbf{Pr}\,(s\,|\,\xi_i)\,)\,,\ s \in \mathcal{A}\,,\ \xi_i \in \Xi\,.$

Iz ovog izlaganja proizlazi relativnost informacije, entropije i složenosti s obzirom na razinu razreda modela, odnosno na stupanj točnosti modelarove računarske prezentacije. Svojstva prezentacije određuju značenje ovih veličina, te vode na semantiku stanja procesa. Hijerarhija ϵ-strojeva izložena na slici 4.5 je ujedno i induktivna hijerarhija u kojoj s porastom razine pojam stanja dobiva sve veću semantičku težinu. Definiciju ϵ-stroja iz 4.3.4 možemo sada izreći i na sljedeći način:

Definicija 4.9 *ϵ-stroj je upravo:*
(i) minimalni model, na
(ii) najmanje moćnoj računarskoj razini, koji daje
(iii) konačni opis.

4.4.4 Algoritam

Kao zaključak ovog odjeljka navodimo algoritam za gradbu ϵ-strojeva. Njime se nadopunjuje i poopćuje okvirni postupak za SFA iznesen u pododjeljku 4.4.1.

Algoritam 4.1 *Gradba ϵ-stroja:*

1. Na najnižoj razini $lev = 0$ niz podataka je svoj vlastiti model $M_0 = \mathbf{s}$, za koji možemo reći da je degeneriran, trivijalan ili neinformativan. Postavimo razinu na za 1 višu vrijednost: $lev \leftarrow lev + 1$.

2. Potrebno je rekonstruirati model M_{lev} na razini lev prema modelu niže razine M_{lev-1}, uočavanjem i izlučivanjem pravilnosti, tj. razvrstavanjem razreda ekvivalencije u strukturi stanja i prijelaza tog nižeg modela, tako da vrijedi $M_{lev} = M_{lev-1}/\sim$, gdje je $\sim$ relacija ekvivalencije koja definira uzročna stanja razine lev.

   Drugim riječima, traže se pravilnosti u skupinama stanja u M_{lev-1}. Skupine koje sadrže međusobno srodna stanja u M_{lev-1} postaju uzročna stanja u M_{lev}, a prijelazi između skupina stanja u M_{lev-1} postaju prijelazi između stanja u M_{lev}.

3. Provjeravamo sažetost (škrtost) razreda na razini lev, tako da utvrđujemo izgled sve točnijih modela. Kao ocjena veličine aproksimacije služi nam skupni parametar ϵ (koji npr. možemo promatrati kao $\epsilon \sim D^{-1}$, gdje je D duljina nizova koje primamo i ujedno dubina[8] stabla). Za $\epsilon \to 0$ dosižemo granicu najtočnijih mogućih modela.

4. Ako složenost modela divergira

$$|| M_{lev}|| \underset{\epsilon \to 0}{\longrightarrow} \infty ,$$

   tada je potrebno povisiti razinu $lev \leftarrow lev + 1$, i vratiti se na korak 2.

5. Ukoliko je

$$|| M_{lev}|| \underset{\epsilon \to 0}{<} \infty ,$$

   tada je ovaj postupak našao prvu razinu koja je računarski najmanje moćna, a daje konačnu prezentaciju sustava. ϵ-stroj je rekonstruiran. Kraj algoritma.

[8] Termin *dubina* koristimo kao sinonim za visinu stabla. Vidjeti također diskusiju u 4.5.2.

Osvrnimo se još ukratko na mogućnost algoritmizacije inovacijskog koraka pri povratku iz točke 4 u točku 2 gornjeg algoritma. Ključni korak u iznalaženju novog razreda modela je otkrivanje pravih relacija ekvivalencije. U tome će se ogledati domišljatost, odnosno *inteligencija* modelara.

Primijetimo da i u inovacijskom koraku možemo rabiti načelni pristup analogan algoritmu konstrukcije ϵ-stroja, pa možemo govoriti o rekurzivnom pozivu gornjeg algoritma iz njega samog. To opisujemo na sljedeći način:

i. Rezultat koraka 3 je niz modela $M_{lev-1}(\epsilon)$, $M_{lev-1}(\epsilon/2)$, $M_{lev-1}(\epsilon/4)$, ... , sa sve boljim parametrom točnosti ϵ, koje možemo reinterpretirati kao nove ulazne podatke za naš induktivni postupak.

ii. S ulaznim podacima iz (i) postupamo kao s ulaznim podacima iz koraka 1 gornjeg algoritma, tj. ponavljamo korak 2 iz algoritma 4.1.

iii. Ponavljamo korak 3 iz algoritma 4.1.

iv. Ponavljamo korak 4 iz algoritma 4.1. Ako model M_{lev} divergira, tada smo u koraku 2 odabrali krive relacije ekvivalencije, odnosno nismo na tragu pravih simetrija. Treba se vratiti na korak 2 i iznaći prave simetrije procesa.

v. Model M_{lev} ne divergira, što znači da su uočene simetrije valjane, i da je novoodabrana razina modela odgovarajuća.

Problem *dna rekurzije* u ovom algoritmu odražava imanentni epistemološki problem — kako započeti bilo kakvo modeliranje, kako odabrati instrument? O tome je već bilo riječi u uvodu ovog poglavlja.

4.4.5 Univerzalno modeliranje?

Na kraju ovog odjeljka osvrnimo se na upravo izneseni *plan univerzalnog modeliranja,* kao prijedlog ambicioznog projekta u kojem bi se pored teorije izračunavanja trebala koristiti i dostignuća umjetne inteligencije za ostvarenje "inteligentnog" inovacijskog koraka. Prema našim saznanjima, Crutchfield kao autor teorije ϵ-strojeva spominje i konkretizira automatizaciju postupka samo do razine stohastičkih konačnih automata[1] — što je ostvareno i u ovom radu. Daljnje faze rada, kao što je npr. nalaženje strojeva za proizvodnju nizova (razina 3 na sl. 4.5) predstavljaju veliki skok u algoritamskoj kompleksnosti postupka. Eventualno ostvarenje te zamisli možemo smatrati kao ozbiljan računarski izazov za budućnost.

4.5 Od vremenskog niza do SFA

Prethodni odjeljak je iznio opće odrednice modeliranja dinamičkih sustava kroz gradbu ϵ-strojeva. Sada se usredotočujemo na pojašnjenje detalja u temeljnoj etapi njihove gradnje, tj. onoj koja završava razinom stohastičkih konačnih automata [9]. U tom smislu je ovaj odjeljak prethodnica sljedećem poglavlju, u kojem će težište biti na programskoj implementaciji ovdje zacrtanog postupka.

4.5.1 Vremenski niz

U gornjem izlaganju definirali smo rekonstrukciju ϵ-strojeva kao iterativan proces. Počinje se od samog vremenskog niza, dobivenog na temelju rezultata mjerenja nekog mjernog instrumenta (sl. 4.1). To je početna (nulta) razina opisa sustava, na kojoj se zasniva daljnje modeliranje.

Za jednodimenzionalne sustave uz grubu izmjeru, mjerna particija ima dva elementa $(\varepsilon \simeq 1/2)$, i abeceda je binarna. Slijedi da je vremenski niz oblika

$$\mathbf{s} = s_0 s_1 s_2 s_3 \ldots , \quad s_i \in \mathcal{A} = \{0, 1\} , \; i = 0, 1, \ldots , n_{\mathbf{s}} - 1 . \tag{4.34}$$

Vremenski niz je duljine $n_{\mathbf{s}}$, i to predstavlja aproksimaciju putopisa kao orbite u simboličkoj dinamici (vidjeti diskusiju u 2.3.1). Sljedeća aproksimacija odnosi se na maksimalnu duljinu riječi $\mathbf{w}$ koje kao podnizove iščitavamo iz $\mathbf{s}$. Uvažit ćemo da je ta duljina D. Zamišljamo prozor širine D mjesta koji pomičemo u desno, počevši od diskretnog trenutka $t = 0, 1, \ldots$, pa sve do $t = n_{\mathbf{s}} - D$. S riječima $\mathbf{w}^D \in \{0, 1\}^D$ duljine D, pročitanim unutar tog prozora, "hranimo" raščlambeno stablo. Da preciziramo, neka je:

$$\mathbf{s} = s_0 s_1 s_2 \ldots s_{D-1} \, s_D \, s_{D+1} \ldots s_{n_{\mathbf{s}}-D} \, s_{n_{\mathbf{s}}-D+1} \ldots s_{n_{\mathbf{s}}-1} . \tag{4.35}$$

Iz ovog niza možemo pročitati $n_{\mathbf{s}} - D + 1$ riječi duljine D, koje su redom:

$$\begin{aligned} \mathbf{w}_0^D &= s_0 s_1 \ldots s_{D-1} , \\ \mathbf{w}_1^D &= s_1 s_2 \ldots s_D , \\ \mathbf{w}_2^D &= s_2 s_3 \ldots s_{D+1} , \\ &\vdots \;\; \vdots \\ \mathbf{w}_{n_{\mathbf{s}}-D}^D &= s_{n_{\mathbf{s}}-D} \, s_{n_{\mathbf{s}}-D+1} \ldots s_{n_{\mathbf{s}}-1} . \end{aligned} \tag{4.36}$$

Svaka iduća riječ dobivena je iz prethodne djelovanjem posmačnog preslikavanja σ (vidjeti 2.2.3), s time da se na mjesto zadnjeg simbola postavlja prvi sljedeći

simbola iz niza $\mathbf{s}$:

$$\mathbf{w}_{i+1}^D = \sigma(\mathbf{w}_i^D)\, s_{i+D}\,, \quad i = 0, 1, \ldots, n_{\mathbf{s}} - D - 1\,, \tag{4.37}$$

$$\mathbf{w}_i^D = \mathbf{w}^D\left(\sigma^i(\mathbf{s})\right)\,, \quad i = 0, 1, \ldots, n_{\mathbf{s}} - D\,. \tag{4.38}$$

Dakle, ovdje primjenjujemo aproksimaciju simboličke dinamike s točnošću reda $\varepsilon^D = 2^{-D}$, pa svi teorijski zaključci izvedeni u odjeljku 2.2 vrijede do na tu točnost. Npr. svaka riječ aproksimira točku u faznom prostoru do na 2^{-D}, funkcija posmaka je kvazikontinuirana do na tu veličinu itd.

U upravo izloženoj eksplikaciji se oslikava bit diskusija o nužnosti aproksimacije u modeliranju, što se provlači kroz cijelo naše razmatranje (vidjeti 2.3, 4.1, 4.4.2). Aproksimacija se ogleda u postojanju instrumenta s rezolucijom $\varepsilon > 0$, a s druge strane imamo samo konačne računarske resurse. Primijetimo da je, sa stanovišta gradbe modela, presudno ovo drugo. Kad bismo mogli promatrati riječi proizvoljne duljine $D \to \infty$, tada bi se i pogreška simboličkog prikaza stezala prema nuli.

4.5.2 Raščlambeno stablo

Nakon što smo u prethodnom koraku iz vremenskog niza izlučili riječi $\mathbf{w}^D$, one se, po uzoru na prevođenje simboličkog kôda u prevodiocima, analiziraju u raščlambenom stablu. Odmah se može uočiti i specifična razlika našeg stabla. Kao prvo, pošto su riječi građene od binarnih simbola, stablo je binarno. Nadalje, sve riječi kojima hranimo stablo su čvrste duljine D — pa su u njemu svi listovi na istoj visini. U skladu s oznakom D za duljinu riječi, kojoj možemo pridijeliti i značenje "dubine aproksimacije", umjesto termina *visina* stabla, u ovom odjeljku koristimo termin *dubina*. Dubina dakle predstavlja duljinu *najdulje staze* od korijena stabla do lista.

Odavde slijedi definicija *ϵ-stabla:*[9]

Definicija 4.10 ϵ-stablo *je binarno stablo u kojem su svi listovi na istoj dubini* D.

Tvrdnja ekvivalentna gornjoj definiciji je: *idući bilo kojom stazom u ϵ-stablu, možemo ostvariti put duljine* D. Pošto se u našem radu bavimo isključivo ovakvim stablima, prefiks ϵ ćemo ispuštati, i pod nazivom *stablo* podrazumijevati ϵ-stablo.

Primijetimo da, iako su ulazne riječi konstantne duljine, raščlambeno stablo nam daje podatke o svim podnizovima duljine $l \leq D$. Tu leži glavna reprezentativna

[9] Crutchfield ne uvodi poseban naziv za stabla u svojoj teoriji. Računarski gledano, potrebno je formalizirati specifičnost ovih struktura.

snaga ovakvog prikaza. Npr. da smo na uobičajeni statistički način analizirali pojavljivanje binarnih podnizova duljine D u nekom vremenskom nizu, ništa ne bismo doznali o statističkim parametrima nizova kraćih od D.

Razlika našeg stabla u odnosu na raščlambeno stablo nekog prevodioca je očita. Ona proizlazi iz različitih zadaća koje imaju modeliranje i prevođenje. Dok se prevođenje sastoji u prepoznavanju i razvrstavanju primljenih nizova prema unaprijed određenoj strukturi (definiranoj gramatikom jezika), u modeliranju tu strukturu tek trebamo razaznati. Jedna od neminovnih posljedica toga je razmatranje riječi duljine $D = const$.

Modelar nema nikakvih podataka o sustavu osim onih koji proizlaze iz mjerenja. Očito je da se radi o aproksimaciji skupa Σ_w svih mogućih riječi neke abecede (odjeljak 3.2) samo s riječima duljine $l \leq D$. Postavljanje gornje granice na l je eksperimentalna nužnost, što je već diskutirano u više navrata. To izravno utječe i na gornju granicu korelacije koju možemo pratiti u nizu (usporediti s 4.4.2).

Gledano sa stanovišta dekodiranja u teoriji informacija, radi se o problemu *jednoznačne trenutne razrješivosti* nizova (kôdnih riječi) promjenljive duljine. Pri tom treba odmah naglasiti da trik, koji se sastoji u označavanju kraja riječi na neki drugi način (kao npr. kod Morseovog kôda), ne predstavlja rješenje problema. Označavanje kraja riječi na "neki drugi način" je jednostavno ekvivalentno uvođenju novog simbola u abecedu. Značenje tog novog simbola je na početku gradbe modela isto tako nepoznato kao i značenje svih ostalih simbola.[10]

4.5.3 Hranidba stabla

U procesu koji možemo nazvati *hranidba stabla* potrebno je izgraditi strukturu prema definiciji 4.10. Stablo je u početku prazno, u smislu da nije prihvatilo ni jednu riječ. Takvo stablo ne sadrži ni jedan čvor (pa ni korijen). Kad započne proces hranidbe, kreiramo korijen i počinjemo s unašanjem ili *pohranom* riječi. Korijen, kao i svaki čvor stabla, sadrži brojač koji se inkrementira ukoliko je staza koju definira unesena riječ prošla kroz dati čvor. Pošto svaka staza izlazi iz korijena, brojač u korijenu

[10] Da nadopunimo gornju ilustraciju promotrimo slučaj primatelja koji želi odgonetnuti Morseov kôd, uz pretpostavku da on ne poznaje niti definiciju tog kôda, niti širi semantički kontekst nužan za njegovo razumijevanje (kao što je npr. latinska abeceda). Iz mjerenja trajanja primljenih impulsa i vremenskih intervala između njih, primatelj će relativno lako ustanoviti da se poruke sastoje od kratkog i dužeg signala, te od kraće i duže stanke. Međutim, izvesti zaključak da je duža stanka oznaka kraja kôdne riječi, je problem koji je kvalitativno jednako težak i, naravno, zbog toga ekvivalentan dešifriranju cijelog kôda. Prema tome, moramo poći od elementarnijeg modela s čvrstom duljinom niza.

pokazuje broj ukupno pohranjenih riječi.

Da preciziramo ovaj postupak, pretpostavimo da je potrebno pohraniti riječ

$$\mathbf{w}^D = s_0 s_1 s_2 \ \ldots \ s_{D-1} \, .$$

Ona definira ukupno $D+1$ prefiks ako uvažimo prazan niz e, te samu riječ $\mathbf{w}^D$ kao moguće prefikse. Primijetimo da svaki od prefiksa definira točno jedan čvor stabla. Tako prefiks e definira korijen (razina čvora je nula), prefiks s_0 definira lijevi (desni) čvor prve razine za slučaj da je pročitani simbol 0 (1) , prefiks $s_0 s_1$ analogno definira jedan od četiri čvora na razini dva itd., sve do same riječi $\mathbf{w}^D$ koja definira čvor na razini D. Svi čvorovi predstavljeni prefiksima riječi $\mathbf{w}^D$ definiraju *hranidbenu stazu* $\Theta(\mathbf{w}^D)$. Prilikom hranidbe, ako je neki čvor iz $\Theta(\mathbf{w}^D)$ već postojao, tada se brojač u tom čvoru inkrementira, a ako nije postojao tada je čvor potrebno stvoriti i brojač inicijalizirati na 1.

Na slici 4.6 prikazan je proces hranidbe stabla vremenskim nizom

$$\mathbf{s} = 01101011011110110101 \ldots \, . \tag{4.39}$$

Taj je niz generiran iz procesa kojeg možemo okarakterizirati nazivom: "generiraj 1 ako je prethodilo 0", ili drugim riječima: "nema uzastopnih nula". Pravilni izraz koji opisuje taj proces je

$$(0 + e)(1 + 10)^* \, , \tag{4.40}$$

(vidjeti odjeljak 3.3), uz pretpostavku da $+$ označava jednaku vjerojatnost izbora.[11] Na slici je podebljanom linijom označena staza $\Theta(01101)$ koja se dobiva pohranom početne riječi $\mathbf{w}_0^D$ iz gornjeg niza $\mathbf{s}$.

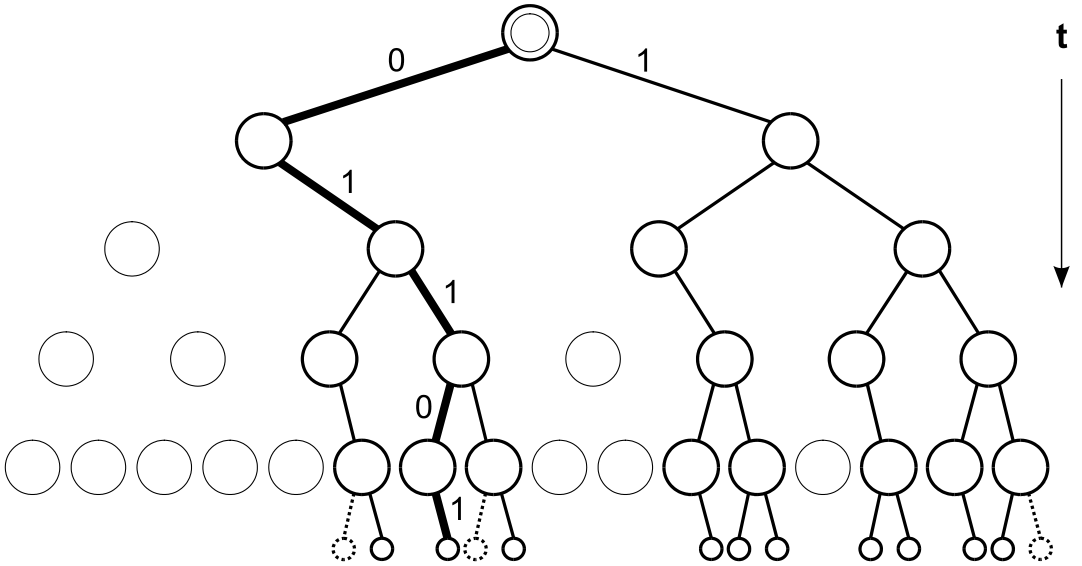
4.5.4 Vjerojatnosni parametri

Ukoliko je niz $\mathbf{s}$ duljine $n_{\mathbf{s}}$, tada je vjerojatnost za pojavu podniza $\mathbf{w}^l$, $l \leq D$, kojem odgovara čvor η na razini l, sljedeća:

$$\Pr\left(\mathbf{w}^l\right) = \Pr\left(\eta\right) = \frac{c_\eta}{n_{\mathbf{s}} - D + 1} \ = \frac{c_\eta}{c_{root}}. \tag{4.41}$$

Tu smo s c_η označili broj odbrojaka u brojaču čvora η, a c_{root} predstavlja ukupan broj pohranjenih podnizova.

[11] Ovaj konkretni niz se dobiva uz podrazumijevajuće parametre u okviru razvijenog programa (vidjeti također poglavlje 6).



Slika 4.6: Hranidba binarnog stabla. Prikazan je izgled binarnog (ϵ)stabla dubine $D = 5$ nakon hranidbe vremenskim nizom $s = 01101011011110110101\ldots$. Podnizovi duljine 5 učitavaju se u raščlambeno stablo. Podebljanom linijom je označena staza koja je stvorena učitavanjem prvog podniza $w_{t_0}^5 = 01101$, duljine 5 u trenutku t_0. Crtkano su označeni čvorovi koje niz s ne sadrži, ali su dozvoljeni prema pravilu našeg sustava.

Pod *prijelaznom vjerojatnosti* $\Pr(\eta \to \eta')$ između čvorova η i η', gdje je η' nasljednik od η, podrazumijevamo uvjetnu vjerojatnost za pojavu niza $\mathbf{w}^{l_2}$ nakon što je primljen niz $\mathbf{w}^{l_1}$, $l_1 + l_2 \leq D$:

$$\Pr(\eta \to \eta') = \Pr\left(\mathbf{w}^{l_2} \middle| \mathbf{w}^{l_1}\right) = \begin{cases} \frac{\Pr\left(\mathbf{w}^{l_1}\,\mathbf{w}^{l_2}\right)}{\Pr\left(\mathbf{w}^{l_1}\right)} = \frac{c_{\eta'}}{c_\eta}, & c_\eta > 0 \\ 0\,, & c_\eta = 0 \end{cases} . \tag{4.42}$$

Donji slučaj uz $c_\eta = 0$ znači da čvor η ne postoji, pa ne postoje ni njegovi nasljednici. U slučaju kad je η' neposredni nasljednik, tj. lijevo ili desno dijete čvora η, tada iz gornje formule slijedi vjerojatnost prijelaza iz jednog uzročnog stanja u sljedeće:

$$\Pr(\eta \to \eta') = \Pr\left(s \,|\, \mathbf{w}^{l_1}\right) = \begin{cases} \frac{\Pr\left(\mathbf{w}^{l_1}\,s\right)}{\Pr\left(\mathbf{w}^{l_1}\right)} = \frac{c_{\eta'}}{c_\eta}, & c_\eta > 0 \\ 0\,, & c_\eta = 0 \end{cases} . \tag{4.43}$$

4.5.5 Nalaženje morfova

Nakon opisane hranidbe raščlambenog stabla, prema algoritmu 4.1 nalazimo se na razini $lev = 1$. Radi distinkcije, ovo stablo ćemo u daljnjem tekstu nazivati *glavno*

stablo. Sada u okviru koraka 2 moramo unutar glavnog stabla dubine D iznaći simetrije i pravilnosti, te precizirati relaciju ekvivalencije. Strukture unutar stabla su ponovo stabla. Uzročna stanja u našem modelu će odgovarati pojedinim čvorovima u glavnom stablu, kao korijenima iz kojih niču *podstabla* dubine L. Svi (pod)nizovi definirani ovakvim podstablom predstavljaju skup mogućih budućih nizova koji se mogu emitirati iz danog uzročnog stanja, tj. prema definiciji 4.5 radi se upravo o **budućem morfu**.

Kao problem tu se javlja izbor optimalne dubine L podstabla, za koju mora vrijediti $1 \leq L \leq D$. Gornja granica je jasna, a za $L = 0$ bi stablo degeneriralo u jedan jedini čvor. Kao takvo, ne bi imalo nikakvu strukturu niti vezu prema budućim stanjima, pa bi zbog toga bilo beskorisno. Ukratko ćemo razmotriti ideju optimiziranja L uz zadani D. Prilikom utvrđivanja podstabala kao morfova htjeli bismo da unutar glavnog stabla imamo čim više primjeraka pojedinog morfa, što će doprinijeti statističkoj težini modela. Uz gornje parametre, broj podstabala dubine L u stablu dubine D može maksimalno biti jednak broju čvorova u stablu dubine $D - L$. Maksimalni broj čvorova u stablu dubine d iznosi

$$n\,(d) = 2^{d+1} - 1\,, \tag{4.44}$$

pa je maksimalni broj $n_{ST\,\max}$ mogućih podstabala jednak:

$$n_{ST\,\max} = 2^{D-L+1} - 1\,. \tag{4.45}$$

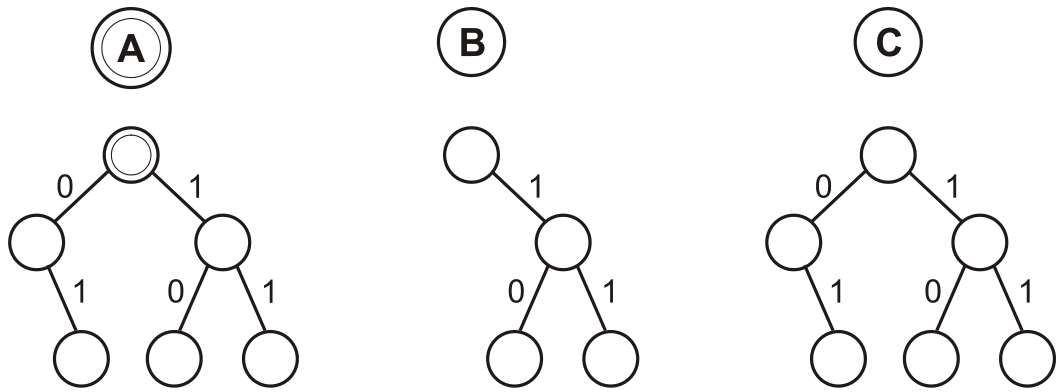
Ovaj kriterij nas, prema tome, navodi na zaključak da treba odabirati $L \ll D$. S druge strane, želimo li dobiti čim raznolikiju strukturu, tj. dozvoliti pojavu čim većeg broja različitih morfova, tada ćemo zahtijevati da L bude čim veće moguće, tj. $L \simeq D$. Naime, za broj različitih morfova dubine L, tj. za ukupan broj različitih (ϵ-)stabala pokazuje se da iznosi

$$m_L = 2^{2^L} - 1\,. \tag{4.46}$$

Broj m_L raste "dvostruko eksponencijalno" s L, pa otuda gornji zahtjev. Kompromisno je rješenje uzimanje

$$L_{opt} \approx \frac{1}{2} D\,. \tag{4.47}$$

Izraz (4.46) je izveden relativno lakim kombinatoričkim računom i potvrđen matematičkom indukcijom, što neće biti izloženo ovdje. Također, točan izvod za L_{opt} prelazi okvire ovog rada [1].



Slika 4.7: Podstabla dubine $L = 2$ pronađena kao jedinstveni morfovi unutar raščlambenog stabla sa sl. 4.6. Svaki morf nosi oznaku s pomoću koje će biti označen kao stanje u SFA (vidjeti sl. 4.8). Korijenski morf je uvijek jedinstven (po definiciji) i predstavlja početno stanje označeno upisanim kružićem. Morfovi A i B morfološki su različiti, odnosno morfološki jedinstveni. Morf C je blizanac morfu A (morfološki je jednak), ali ima različite vjerojatnosne parametre.

Da bismo rekonstruirali ϵ-stroj kao uređeni par $(\Xi, \mathcal{T})_\epsilon$ potrebno je uočiti strukturu svakog mogućeg podstabla u glavnom stablu, počevši od podstabla koje ima korijen zajednički s glavnim stablom. Zatim uočimo u kakvo podstablo prelazi početno podstablo po primitku simbola 0, odnosno po primitku simbola 1. Prema našoj definiciji to odgovara uspoređivanju budućih morfova za svako uzročno stanje.

4.5.6 Morfološki različita stanja

Kao primjer promotrimo sl. 4.6. Uz $D = 5$ uzet ćemo $L = 2$. Iako je s ovakvom dubinom stabla moguće uočiti samo kratke korelacije, to će biti dovoljno u našem slučaju jer je struktura procesa, kao odraz zadanog pravilnog izraza, također jednostavna. Prvo podstablo koje niče iz korijena glavnog stabla prikazano je na slici 4.7 i označeno s A. Pošto se radi o prvom uočenom morfu, on je po definiciji i jedinstven, tj. predstavlja zaseban razred ekvivalencije.

Pogledajmo na slici 4.6 u kakva stanja prelazi naš model iz početnog stanja A po primitku ulaznih simbola. Po primitku 0 prelazi u stanje definirano morfom (podstablom) na sl. 4.7 označenim s B, koje je po svom obliku različito od A. Iz relacije (4.24) je jasno da B predstavlja novi razred ekvivalencije. Po primitku 1 pak iz stanja A prelazimo u stanje koje je morfološki jednako stanju A, jer su pripadni morfovi (podstabla) dubine 2 po svojoj strukturi jednaki. Međutim, iako morfološki jednako stanju A, ovo novo stanje ima morf s drugačijom vjerojatnosnom

strukturom, kao što ćemo vidjeti u daljnjem izlaganju.

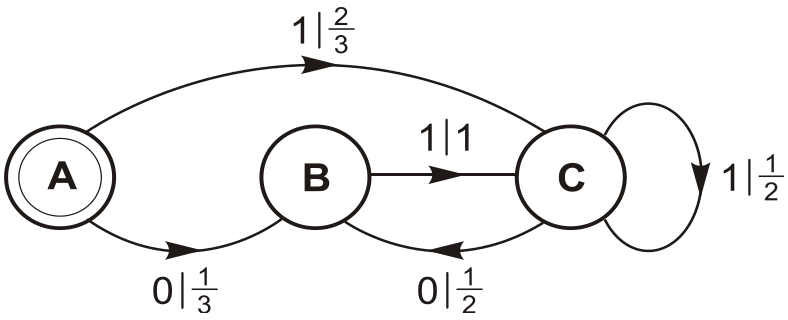
4.5.7 δ-različita stanja

Pošto u ovom test-primjeru poznajemo strukturu procesa iz pridruženog mu pravilnog izraza (4.40), možemo predvidjeti vjerojatnosti prijelaza. Stanje A odgovara situaciji kad modelar nije primio ni jedan simbol iz niza, pa ne može razmišljati kroz kontekst uvjetnih vjerojatnosti. Pošto naš proces u prosjeku emitira dva puta više jedinica od nula, slijedi da vjerojatnost za 0 iznosi $1/3$, a za 1 iznosi $2/3$. Prema tome vjerojatnost za prijelaz iz stanja A u stanje B je $1/3$. Prelazak u stanje B odgovara dakle prijamu nule, a pošto je pravilo našeg procesa da nema uzastopnih nula, morf B ima samo jednu granu iz svojeg korijena. Drugim riječima, nakon stanja B prijelaz je deterministički, jer jedino što se može desiti je pojava jedinice. Privremeno označimo s X stanje u koje prelazimo iz B po pojavi simbola 1. Kad je sustav u tom stanju vjerojatnost za prijam, odnosno odašiljanje, nule i jedinice jest jednaka, što je lako uočljivo iz pravila na kojem se temelji naš sustav. Prema tome, vjerojatnost prijelaza iz stanja X u njegov lijevi podmorf iznosi $1/2$, jednako kao i za desni podmorf. Prema izrazu (4.21) zaključujemo da je:

$$\begin{aligned} |\Pr(0\,|\,X) - \Pr(0\,|\,A)| &= \left|\frac{1}{2} - \frac{1}{3}\right| = \frac{1}{6}\,, \\ |\Pr(1\,|\,X) - \Pr(1\,|\,A)| &= \left|\frac{1}{2} - \frac{2}{3}\right| = \frac{1}{6}\,. \end{aligned} \tag{4.48}$$

Otuda za svaki izbor $\delta < \frac{1}{6}$ proizlazi da je stanje X δ-različito od A, pa ćemo ga nazvati $C = X$ i uvrstiti u skup jedinstvenih stanja. Još je potrebno objasniti što se događa kad u stanju A primimo simbol 1. Nazovimo to stanje Y. Po definiciji našeg sustava, nakon jedinice može uslijediti bilo nula, bilo jedinica, s jednakom vjerojatnosti, a to je upravo odlika stanja C. Prema tome Y nije jedinstveno stanje već je ekvivalentno stanju C. Situacija je prikazana na slici 4.8.

Pažljivi je čitatelj mogao primijetiti da smo u gornjem izlaganju razmatrali samo vjerojatnosti za prijelaze u neposredni podmorf, što odgovara vjerojatnostima za pojavu nizova duljine $l = 1$. U stvari, primjedba je relevantna samo za slučaj usporedbe stanja Y sa C, gdje bi se eventualno moglo desiti δ-odstupanje za neki od nizova duljine 2. Kratko objašnjenje će pokazati da to nije slučaj. Nizovi duljine $l = 2$ koji se mogu pojaviti nakon što smo primitkom simbola 1 prešli u stanje Y su: 01, 10, i 11. Uvjetne vjerojatnosti za njihovo pojavljivanje nakon stanja Y su



Slika 4.8: Stohastički konačni automat za proces "nema uzastopnih nula". U početnom stanju A model odražava svoje neznanje o fazi procesa. Vjerojatnosti prijelaza tu su jednake (bezuvjetnim) vjerojatnostima za pojavu nula odnosno jedinica. Stanje A je tranzijentno. Čim se pojavi nula, model razaznaje fazu i dalje oscilira između stanja B i C. Prijelaz iz B u C je deterministički.

redom: 1/2, 1/4, i 1/4. Tako npr. za slučaj primitka niza 01 vrijedi:

$$\Pr\left(01|Y\right) = \Pr\left(01|1\right) = \Pr(1|0)\Pr\left(0|1\right) = 1\times\frac{1}{2} = \frac{1}{2}\,. \tag{4.49}$$

Potpuno je isto razmatranje za slučaj stanja C, pa se zaključuje da su stanja Y i C ekvivalentna. Također, može se primijetiti da struktura našeg procesa ima maksimalnu korelaciju duljine 1, i u tom su smislu podstabla dubine 2 bila dovoljna za njeno uočavanje. Razmatranje uz podstabla s $L = 1$ dala bi istovjetne rezultate.

Zaključujemo da je ϵ-stroj $(\Xi, \mathcal{T})$ promatranog procesa predstavljen s pomoću SFA na sl. 4.8 korektan. Skup stanja je $\Xi = \{A, B, C\}$, odakle proizlazi topološka složenost

$$C_0 = \mathrm{ld}\,||\,\Xi\,|| = \mathrm{ld}\,3 = 1.5850\,\mathrm{bit}\ .$$

Potpuna struktura operatora $\mathcal{T}$ je uočljiva iz grafičkog prikaza automata, a statistička svojstva prijelaza možemo sažeti u matrici prijelaza T:

$$T = \begin{pmatrix} 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & 0 & 1 \\ 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}\ .$$

Lijevi svojstveni vektor za gornju matricu prijelaza iznosi

$$\mathbf{p}_{st} = \left(0, \frac{1}{3}, \frac{2}{3}\right)\ ,$$

odakle slijedi statistička složenost procesa:

$$C_\mu\left(\Xi, \mathcal{T}\right) = H\left(\mathbf{p}_{st}\right) = 0.918296\,\mathrm{bit}\ .$$

U asimptotskoj vjerojatnosti $\mathbf{p}_{st}$ je $\Pr(A) = 0$, što je naravno posljedica činjenice da je A tranzijentno stanje. Uočljiva je i velika razlika između topološke i statističke složenosti, koja također proistječe iz istog razloga. Nakon što pri prvom prijelazu proces izađe iz tranzijentnog stanja, on boravi u jednom od preostala dva.

4.5.8 Specijalizacija algoritma za SFA

Prethodni je primjer trebao poslužiti kao podrobna ilustracija postupka gradbe računskih modela do razine temeljnog ϵ-stroja, predstavljenog stohastičkim konačnim automatom. Pošli smo od hranidbe raščlambenog stabla nizom generiranim iz relativno jednostavnog procesa, što nam je omogućilo da lako zaključimo na oblik glavnog stabla te da odredimo odgovarajuće prijelazne vjerojatnosti. Otuda smo odredili SFA i izračunali statističke parametre procesa.

U zaključku ovog poglavlja dajemo algoritam ključnog dijela tog postupka, koji se odnosi na nalaženje stohastičkih konačnih automata iz raščlambenog stabla. Kao takav, on predstavlja specijalizaciju općeg algoritma 4.1 za slučaj prelaska iz prve razine modela na drugu, što odgovara nalaženju razreda ekvivalencije podstabala unutar glavnog stabla s obzirom na relaciju ekvivalencije (4.21).

Algoritam se temelji na općenitoj definiciji ekvivalencije morfova za ovaj korak izgradnje modela (def. 4.4). On zaokružuje izložene teorijske postavke te formalizira ideje iznesene u gornjem primjeru.

Algoritam 4.2 *Nalaženje razreda ekvivalencije za morfove prezentirane podstablima dubine L unutar glavnog stabla dubine D.*

1. Postavimo *tekući* čvor η' u korijen nepraznog glavnog stabla. Svaki morf definiran podstablom dubine $L \leq D$ koje započinje u korijenu glavnog stabla je jedinstveni morf i definira razred ekvivalencije. Taj morf ćemo označiti kao $\boldsymbol{\mu}_0$, a njemu pripadno stanje sa ξ_0. Podstablo je jedinstveno određeno svojim korijenom i dubinom. Svaki čvor je jedinstveno određen binarnim nizom. U slučaju korijena glavnog stabla to je prazan niz e, tj. vrijedi $\eta' = e$.

   Tako skup stanja zasada ima jedan član, $\Xi = \{\xi_0\}$. U skupu razreda ekvivalencije $\mathcal{C}$ imamo jedan razred ekvivalencije predstavljen morfom $\boldsymbol{\mu}_0$, odnosno $\mathcal{C} = \{\boldsymbol{\mu}_0\}$.

2. Uzmemo morf iz $\mathcal{C}$ za koji nismo provjerili podstabla. Morf iz $\mathcal{C}$ je neprazno ϵ-podstablo za koje po definiciji vrijedi da mora imati barem jednu stazu duljine L. Krenemo prema jednom od moguća dva djeteta, postavljanjem

$$\eta' \leftarrow \eta' s\,,\ s \in \{0,1\}\,. \tag{4.50}$$

   Ovdje pretpostavljamo da je prikaz čvora η' dan u formi niza, te da je prema tome, novi čvor određen nizom kojem smo dodali bilo 0 (za lijevo dijete), bilo 1 (za desno dijete). Ukoliko je korijen η' novog podstabla na razini dubljoj od $D - L + 1$, prelazimo na korak 3.

   (a) Promotrimo novo podstablo dubine L s korijenom u čvoru η', koje definira morf $\boldsymbol{\mu}'$. Morf $\boldsymbol{\mu}'$ redom uspoređujemo s jedinstvenim morfovima iz $\mathcal{C}$ tako da provjeravamo kriterij jednakosti (4.21). Za slučaj morfova predstavljenih podstablima, to se svodi na usporedbu vjerojatnosti svih nizova $\mathbf{s}$ duljine $1 \leq l \leq L$ koji su mogući u podstablu morfa $\boldsymbol{\mu}'$, u skladu sa sljedećim izrazom:

$$\boldsymbol{\mu}_i \underset{\delta}{\sim} \boldsymbol{\mu}' \iff |\Pr(\mathbf{s}\,|\,\eta) - \Pr(\mathbf{s}\,|\,\eta')| \leq \delta\,,\ \forall\,\mathbf{s} \in \boldsymbol{\mu}'. \tag{4.51}$$

   Vjerojatnosti u gornjoj relaciji računamo prema (4.42). Ako postoji takav morf $\boldsymbol{\mu}_i \in \mathcal{C}$ da vrijedi gornja relacija, tada mu je $\boldsymbol{\mu}'$ δ-ekvivalentan (spada u njegov razred ekvivalencije). Drugim riječima, $\boldsymbol{\mu}'$ nije jedinstveni morf. Prelazimo na korak 2b.

Ako ne postoji morf $\boldsymbol{\mu}_i \in \mathcal{C}$ kojemu je $\boldsymbol{\mu}'$ δ-ekvivalentan, tada je $\boldsymbol{\mu}'$ jedinstveni morf koji uvrštavamo u $\mathcal{C}$ kao novi razred ekvivalencije: $\mathcal{C} \leftarrow \mathcal{C} \cup \boldsymbol{\mu}'$. Pri tom zabilježimo da li je $\boldsymbol{\mu}'$ lijevo ili desno dijete svojeg roditelja, što daje informaciju o simbolu koji je prouzročio prijelaz, te vjerojatnost prijelaza od nadmorfa (roditelja) do promatranog morfa.

(b) Ukoliko postoji i drugo dijete, postavljamo tekući čvor η' na mjesto tog drugog djeteta i ponavljamo postupak iznesen u 2a.

3. Ako smo u koraku 2 pronašli nove morfove, odnosno ako još postoje morfovi za koje nismo provjerili razred ekvivalencije njihovih podmorfova, tada se vraćamo na korak 2. Ako nema neprovjerenih morfova, ili ako je korijen η' novog podstabla na razini dubljoj od $D - L + 1$, tada su u skupu $\mathcal{C}$ sadržani svi jedinstveni podmorfovi koji odgovaraju stanjima SFA, što zajedno s podacima o karakteru i vjerojatnostima prijelaza definira operator $\mathcal{T}$.

   Stvoren je model M_ϵ procesa u obličju ϵ-stroja $(\Xi, \mathcal{T})$ na razini 2, koji odgovara SFA zadanom s $M_{SFA} = (Q, \mathcal{A}, q_0, \mathbf{T})$. Kraj algoritma.

Poglavlje 5

Razvoj programskog okruženja

Izlaganje ovog poglavlja obuhvaća originalni doprinos našeg rada, koji se sastoji u ostvarenju programskog okruženja za modeliranje dinamičkih sustava s pomoću stohastičkih konačnih automata. Prethodna poglavlja su imala za cilj da podrobno objasne formalnu pozadinu teorije ϵ-strojeva. Posebno, odjeljak 4.5 je konkretizirao zadaću koju programsko okruženje treba izvršiti: *iz primljenog vremenskog niza potrebno je zaključiti na model predstavljen razredom stohastičkih konačnih automata.* U prvom odjeljku koji slijedi razmatramo programska načela koja su primijenjena u okvirima dosljedno provedenog objektno orijentiranog pristupa. Nadalje predstavljamo one ostvarene klase koje su najspecifičnije za ovaj rad. Potpuni pregled programske dokumentacije, u vidu svih zaglavnih i implementirajućih datoteka, može se naći na disketi koja je priložena uz ovaj rad.

5.1 Objektno orijentirani pristup

Razvoj programskog okruženja za relativno kompleksan i višeslojan problem kao što je naš, teško je zamisliv bez primjene koncepta objektno orijentiranog pristupa. Pored svih dobrobiti koje on pruža, možda je upravo najznačajniji taj da je programer od samog početka prisiljen razmišljati prema načelu "*planiraj globalno, a programiraj lokalno*". Prividno udaljenje od konkretnog problema na početku programiranja, zbog nužnosti formalizacije klasa te određenja njihove hijerarhije i odnosa, na koncu je višestruko nagrađeno.

5.1.1 Program *Dynamical Systems Automata*

Program koji je razvijen u sklopu ovog rada nazvan je *Dynamical Systems Automata* (hrv. *Automati za dinamičke sustave*), s kraticom *DynSysA*, odnosno DSA. Napisan je u objektno orijentiranom jeziku C++, današnjem *de facto* standardu izradi profesionalnih primjenskih programa. Korišten je programski alat *Visual C++* u sklopu okoline *Developer Studio* tvrtke Microsoft (inačice Dev. Studio 97 i Vis. C++ 5.0).

Objektno orijentirani jezik C++ ubraja se u alate s relativno sporom razvojnom krivuljom, posebice kad se kreira i *grafičko korisničko sučelje* (engl. *Graphical User Interface*, GUI). No budući da je GUI standard današnjih profesionalnih programa, kojim se značajno poboljšava učinkovitost njihove uporabe i preglednost dobivenih rezultata, njegova je izrada postavljena kao dodatan zadatak ovog rada. On je ostvaren s pomoću MFC podrške (od engl. *Microsfot Foundation Classes*), koja nudi obilje mogućnost, ali je često i vrlo zahtjevna za savladavanje.

Upravo zbog relativne složenosti gore opisanog izbora znanstvenici iz tehničkih i prirodnih područja često pribjegavaju tzv. RAD programskim okruženjima (akronim od engl. *Rapid Application Development*). Ona najčešće omogućuju brži pristup samom problemu i brže dobivanje rezultata, uz manji gubitak vremena za oblikovanje programa i podatkovnih struktura te ostvarenje korisničkog sučelja. No tako dobiveni programi po svojim se izvedbenim odlikama ne mogu mjeriti s dobro implementiranim C++ programom. Također, nama je bio cilj ne samo dobivanje rezultata, već i ostvarenje programske osnove uz maksimalno poštivanje načela dobrog programiranja i korištenje objektno utemeljenog pristupa. Samo dobar temelj omogućuje relativno lake i učinkovite preinake u budućnosti i daljnji razvoj programa. A glede toga objektno orijentirano programiranje nema alternative.

5.1.2 Programska načela

Iako cilj ovog rada nije detaljnija analiza primijenjenog objektnog pristupa i načina dizajniranja, spomenut ćemo osnovna načela koja su bila vodilja pri ostvarenju ovog projekta. Od mnogih načela koja se ističu pri određenju termina *dobrog programiranja,* za program čija je izrada limitirana vremenski i izvorišno (na jednog programera), morali smo postaviti prioritete. Npr. Tucker [19] spominje devet načela:

Ispravnost, robusnost (*otpornost*), *bliskost korisniku, promjenljivost* (*prilagodljivost*), *ponovljivost uporabe, međudjelovanje* (*s drugim programima*), *učinkovitost, prenosivost, sigurnost* (*zaštićenost od zloporabe*).

Gornji kriteriji nisu poredani po svojoj važnosti za naš projekt — s izuzetkom prvog. *Ispravnost* (engl. *correctness*) je gotovo bez iznimke kriterij broj jedan u programiranju većine problema, a u razvoju programske podrške za znanstvenu uporabu to je, naravno, neosporni prioritet. Kritični dijelovi programa, kao što su npr. ključni algoritmi za iteraciju dinamičkih sustava, pretraživanje stabla, hranidbu stabla i nalaženje morfova, testirani su na više razina — i na "papiru", i u radu. Dio korisničkog sučelja je bio izrađen upravo radi ostvarenja potpune kontrole nad radom algoritama, te za provjeru sadržaja mnogobrojnih korištenih podatkovnih struktura. Ovo se posebice odnosi na strukturu stabla, o kojoj će biti više riječi u nastavku.

Sljedeći kriterij kojem je posvećena posebna pažnja je *učinkovitost* (engl. *efficiency*). Učinkovitost je tretirana selektivno, uočavanjem kritičnih *prostornih* i *vremenskih* zahtjeva u algoritmima i strukturama. Npr. kod binarnog stabla, kao strukture koja je najspecifičnija za ovaj rad, broj mogućih elemenata raste eksponencijalno s njegovom visinom. Tu se prioritetno moralo voditi računa o prostornoj komponenti sustava. Zato je stablo ostvareno dinamičkim pridjeljivanjem memorijskog prostora (*dynamic storage allocation*).

Kao suprotni primjer vremenski kritične funkcije, možemo navesti hranidbenu funkciju koja će detaljnije biti izložena u pododjeljku 5.3.4. Tu je bilo nužno uvođenje optimizacije prikaza i izračuna orbite glede prostornih i vremenskih zahtjeva, što je i izvršeno pod cijenu kompleksnijih algoritamskih rješenja. U takvim slučajevima, postavljeno načelo učinkovitosti nam nije dozvoljavalo kompromisna rješenja.

Ujedno, kao nadopuna kriteriju učinkovitosti, a u svezi s gornjom diskusijom, nameće se postavljanje i dodatnog kriterija kojeg bismo mogli nazvati *strukturalna podobnost* programa. Taj bi se kriterij vjerojatno mogao svesti na nekoliko gore nabrojanih, ali smo mišljenja da se radi o zahtjevu zasebne vrijednosti. Strukturalna podobnost programa bi se trebala ticati one najranije faze projektiranja, u kojoj se kroz ekspertizu problema zacrtava što i kako raditi. Nama je cilj bio da u začetku dobro pogodimo strukturalnu podobnost našeg programa, u smislu da je on podoban za modeliranje dinamičkih sustava.[1]

Što se tiče preostalih navedenih načela, strogo su poštovana ona koja u stvari proizlaze iz uporabe objektno orijentiranog pristupa. U jeziku kao što je C++ po-

[1] Npr. pada nam na pamet usporedba programa Word i TeX. Prvi je strukturalno podoban za obradu i prikaz teksta u grafičkom okruženju, dok je drugi strukturalno podobniji za slovoslagarski posao. Program SWP (Scientific Work Place) tvrtke TCI, u kojem je pisan ovaj rad, koristi strukturalnu podobnost prvog za grafičko sučelje s korisnikom, a koncept drugog (u stvari koristi sam TeX) za konačno oblikovanje teksta.

stoji mogućnost da se izbjegne objektna paradigma, no to se programeru najčešće brzo osveti. To je bila rana lekcija i autoru ovog projekta, pa se odustalo od bilo kakvog izvrgavanja principa objektnog programiranja (vidjeti npr. [20]). U konačnici programer ostvaruje sve dobrobiti objektnog pristupa, odakle načela promjenljivosti (prilagodljivosti), i ponovljivosti uporabe slijede kao podrazumijevajuća, odnosno, za njih nije potrebno ulagati dodatne napore. Istaknimo ovdje kao presudno, ostvarenje svojstva *promjenljivosti*. Tj. ukoliko neko od preostalih načela i nije ispunjeno u početnoj etapi programiranja, svojstvo relativno lake prilagodljivosti objektno orijentiranih programa garantira da će to biti moguće popraviti u budućnosti.

U ovom kratkom osvrtu na ostvareni program dužni smo istaknuti da preostala načela nisu zasada ostvarena u potpunosti. To su ona bez kojih se moglo doći do konačnih rezultata, odnosno koja su mogla biti ostavljena za sljedeću etapu ovog projekta. Tako je npr. bliskost korisniku moguće dalje unaprijediti mnogobrojnim pogodnostima koje za to pruža odabrani programski alat. Korisno bi bilo ostvariti više grafičkih prikaza, dosljednu serijalizaciju podatka i sl. Daljnje moguće unapređenje je međudjelovanje s drugim programima u okviru Microsoftove *Active X* (*OLE*) tehnologije.

Zbog koncepta *otvorenosti*, pod kojom podrazumijevamo da se korisniku pružaju otvorene mogućnosti da proizvoljno kombinira način korištenja pojedinih dijelova programa, pojavio se relativno velik broj mogućih putova, od kojih svi vrlo vjerojatno nisu ispitani. Otvorenost zahtijeva veću "programsku inteligenciju", u smislu predviđanja i korigiranja stanja nastalih slobodnom uporabom; npr. ako se stablo nakon hranidbe vremenskim nizom generiranim iz jednog sustava želi hraniti i iz nekog drugog sustava, ili ostvarenje očuvanja podataka o dobivenim automatima nakon što je njihovo glavno stablo uništeno, i sl. Stoga je potrebno poraditi na ispitivanju svih mogućih putova i otklanjanju eventualnih pogrešaka. Naravno, radi se o svojstvu robusnosti, čije je poboljšanje važan zadatak koji slijedi. To je najbolje provesti usporedo s konkretnim korištenjem programa. Svojstvo prenosivosti u našem slučaju izlazi iz domene programera i prelazi u domenu međudjelovanja operacijskih sustava, a svojstvo sigurnosti je zasada irelevantno.

5.2 Simulacija dinamičkih sustava

Iako dio programa koji se odnosi na simulaciju dinamičkih sustava ne ulazi u domenu modela (sl. 4.1), on je naravno nužan, da bismo mogli generirati vremenski niz.

Dinamički sustavi koje ćemo implementirati moraju biti pogodni za analizu binarnim instrumentom. Za sada podrazumijevamo da su to jednodimenzionalni sustavi. Neki od njih uvršteni su radi testiranja i ne predstavljaju (nelinearne) sustave zanimljive zbog svojih posebnih svojstava. Nelinearni kaotični sustavi pak su implementirani radi izravnog proučavanja njihove strukture[1, 5]. Naravno, analogno se ostvaruje modeliranje na temelju vremenskog niza iz izvora potpuno nepoznatih svojstava.

Za tipične slučajeve uporabe ovog programa u praksi mora se ostvariti statistički značajan broj odbrojaka i u najdubljim čvorovima stabla. Zato će se za stabla dubine D morati kreirati nizovi duljine reda $d^{-1} \times 2^D$, gdje je d procjena reda točnosti. Uz zahtjev da je točnost reda $d \approx 10^{-3}$, izlazi da je potrebno ostvariti nizove duljine 2^{D+10}. Primijetimo da se već za $D = 20$ to penje gotovo do granice opsega 32-bitne cjelobrojne aritmetike. Pomirenje zahtjeva za vremenskom i prostornom učinkovitosti je tu ostvareno uz primjenu *kružnog spremnika* (engl. *circular buffer*) ugrađenog unutar odgovarajućih klasa (klase sustava i klase instrumenta). Za temelj kružnog spremnika je odabran standardni C-poredak (engl. *array*) kao elementarna i vremenski najučinkovitija struktura, čijim se elementima u petlji pristupa inkrementacijom kazaljki, a ne preko općeg indeksiranog elementa.

5.2.1 Temeljna klasa dinamičkih sustava - CDynSys1D

Temeljna klasa dinamičkih sustava apstraktna je klasa *CDynSys1D* (od engl. *Class Dynamical Systems, 1-Dimensional*). U programu dosljedno provodimo široko prihvaćenu *mađarsku notaciju*, tako da većina rabljenih imena (uz malo mašte) sugerira svoje značenje. Potpun pregled sučelja i implementacije funkcija klase moguć je u datoteci *DynSysA_ CFiles* na priloženoj disketi. Najzanimljivije je tu promotriti ostvarenje temeljne funkcije za tzv. *grupnu iteraciju,* koja je izdvojena u ispisu 5.1.

Ispis 5.1 *Implementacija funkcije grupne iteracije u temeljnoj klasi dinamičkih sustava CDynSys1D.*

```
inline void CDynSys1D::InitIteration(double x0)
{
    _ASSERT(m_nPoints <= iMax);
    *Getpx() = x0;
    SetnBIterDone(0);
}

inline void CDynSys1D::PrepBatchIterRepeat(UINT istrl)
{
```

```
    double* p1 = Getpx();
    double* p2 = p1 + (GetnPoints() - istrl + 1);
    for(; p2 < (Getpx() + GetnPoints()); )
    {
        *p1 = *p2;
        p1++; p2++;
    }
}

void CDynSys1D::DoBatchItersInclBI(bool bFrmLastPnt, UINT nBIt,
                                   UINT istrl)
{
    UINT nBIter = nBIt;
    if ( (m_nBIterDone == 0) || !bFrmLastPnt )
    {
        SetnBIterDone(0);
        FirstBatchIteration();
        nBIter = nBIt - 1;
    }

    for(UINT i = 0; i < nBIter; i++)
        RepeatBatchIteration(istrl);
    if(m_bInclBinInstr)
        ((CABinInstrument*) m_pBinInstr)->BinMeasurments();
}
```

Završna razina funkcionalnosti ostvarena je u funkciji `DoBatchItersInclBI`, koja se poziva kad je potrebno izračunati novu skupinu točaka orbite. Riječ *batch* upućuje da se radi o grupnoj iteraciji, tj. odjednom se izračunaju sve točke koje stanu u kružni spremnik. Za kružni spremnik je odabrana veličina od 8K elemenata dvostruke preciznosti. Argumenti funkcije su broj iteracija `nBIt` koje želimo izvršiti, i duljina `istrl` podnizova, koja odgovara dubini stabla D. Ukoliko se radi o prvoj iteraciji, tada se koristi funkcija `FirstBatchIteration`. Ona, počevši od zadane vrijednosti sjemena, cijeli spremnik ispunja točkama orbite. Svaka se sljedeća grupna iteracija mora pripremiti s pomoću funkcije `PrepBatchIterRepeat` koja je sadržana unutar skupne funkcije `RepeatBatchIteration`. Priprema se sastoji u tome da se posljednjih `istrl - 1` simbola iz kružnog spremnika prebaci na njegov početak. Sljedeća točka koja će se računati je točka s indeksom `istrl - 1`, na temelju njoj prethodne točke. Sama iteracija se vrši pomoću virtualne funkcije `BatchIteration` pozvane iz `RepeatBatchIteration`, koja će biti konkretizirana u klasi svakog sustava.

Nakon što je cijeli kružni spremnik napunjen, točke se "mjere" s pomoću tzv. *binarnog instrumenta (BI)*. Njegova je zadaća da niz točaka orbite prevede u niz

binarnih simbola. Diskriminacija se vrši s obzirom na kritičnu točku sustava, kao što je diskutirano u odjeljku 4.1.

5.2.2 Izvedene klase dinamičkih sustava

Iz klase `CDynSys1D` izvedene su apstraktne klase `CDynSys1D_0Par` i `CDynSys1D_1Par` koje oklapaju funkcionalnost za sustave s nula, odnosno jednim parametrom. Prvi su npr. jednostavni sustavi zadani pravilnim izrazima, a drugi su standardni (nelinearni) dinamički sustavi. Iz tih klasa se izvode konkretne klase. U ispisu 5.2 dan je početak objave konkretne klase `CDS1D1P_LogisticMap`, nakon čega slijedi implementacija virtualne funkcije `BatchIter`.

Ispis 5.2 *Implementacija funkcije iteracije za logističku jednadžbu.*

```
class CDS1D1P_LogisticMap :  public CDynSys1D_1Par
{
    ...
}

void CDS1D1P_LogisticMap::BatchIteration(UINT istrl)
{
    _ASSERT( istrl > 1);
    double* p = Getpx() + istrl - 1;
    double x;
    for (; p < (Getpx() + GetnPoints()); p++)
    {
        x = *(p - 1);
        *p = Getc1() * x * (1.0 - x);
    }

    IncrnBIterDone();
}
```

Pribavljačka funkcija `Getpx` daje vrijednost kazaljke poretka u kojem je kružni spremnik, a `GetnPoints` broj točaka u njemu. Nakon što je grupna iteracija završena, pomoću funkcije `IncrnBIterDone` poveća se članska varijabla koja sadrži broj izvršenih iteracija.

Kao dodatni primjer, navest ćemo dinamički sustav zadan pravilom "nema uzastopnih nula", što se opisuje pravilnim izrazom (4.40). Za približenje konkretnoj simulaciji ovog sustava, možemo gornji sustav opisati preciznije na sljedeći način:

Prvi simbol se generira na slučajan način. Ukoliko je prethodni simbol bio nula, sljedeći mora biti jedan. Ukoliko je prethodni simbol bio jedan, sljedeći se generira na slučajan način.

Implementacija ovog algoritma jednostavna je i ostvarena je unutar klase `CDS1D0P_1IfPrevIs0`.

Istovjetni sustav možemo simulirati i na drugi način ako izravno interpretiramo pravilni izraz $(0+e)(1+10)^*$. To je ostvareno u okviru klase `CDS1D0P_NoConsec0s`, i predočeno u ispisu 5.3.

Ispis 5.3 *Implementacija algoritma: "Generiraj nula ili ništa, a potom niz u kojem se pojavljuje ili 1, ili 10".*

```
void CDS1D0P_NoConsec0s::BatchIteration(UINT istrl)
{
    _ASSERT( istrl > 1);
    double* p = Getpx() + istrl - 1;
    double dNorm = RAND_MAX + d10_m10;
    double d2Norm = 2.0 * RAND_MAX + d10_m10;
    double dTmp;
    // srand( (unsigned int) (*(p1-1) * RAND_MAX) );
    // Seed is already initialized in function InitIterate().

    for (; p < (Getpx() + GetnPoints() - 1); p++)
    {
        *p = 0.5 + double(rand())/d2Norm ;
        dTmp = double(rand())/dNorm ;
        if(dTmp < 0.5)
        *(++p) = dTmp ;
    }

    if(p == (Getpx() + GetnPoints() - 1))
        *p = 0.5 + double(rand())/d2Norm ;

    IncrnBIterDone();
}
```

Primijetimo da je zadržana dosljednost u pojmu orbite. Tj. iako nas zanima samo niz nula i jedinica, orbita je zabilježena kao niz točaka iz intervala $[0,1)$,[2] koje se tek kasnije binarno kodiraju. Za potrebe testiranja i prezentacije programa, zadovoljili smo se s postojećom standardnom funkcijom za generiranje 16-bitnih binarnih brojeva.

Funkcionalnost binarnog instrumenta je uklopljena u zasebnu klasu `CABinInstrument`. Kao što proizlazi iz diskusije u 4.1.3, zadaća ove klase je vrlo

[2] Gornja granica intervala je otvorena zbog toga što smo maksimalnom slučajnom broju dodali mali inkrement, predstavljen globalnom konstantom `d10_m10` $= 10^{-10}$. Ovako su brojevi ljepše raspoređeni u intervalu $[0,1)$.

jednostavna. Ona se svodi na kodiranje svake točke orbite u nulu ili jedinicu, već prema tome da li je točka manja, odnosno veća od kritične točke.

Za ostvarenje izbora dinamičkog sustava služi zasebna klasa `CDSCommInterface`. Ona djeluje kao sučelje prema standardnoj klasi dokumenta programa, odnosno prema klasi dijaloškog prozora u kojoj se vrši odabir.

Usprkos činjenici da je obuhvaćena široka paleta dinamičkih sustava, zadanih na različite načine, s različitim svojstvima, domenama i sl., s ovakvom organizacijom hijerarhija klasa je cjelokupna zajednička funkcionalnost sadržana u temeljnoj klasi. Konkretne klase su vrlo jednostavne i pregledne. Dodavanje novog dinamičkog sustava svodi se na specifikaciju konkretne klase, u kojoj je u bitnom potrebno ostvariti samo funkciju grupne iteracije `BatchIteration`.

5.3 Klase stabala

Izlaganje u prijašnjim poglavljima dalo je naslutiti da će okosnica programske implementacije ovog rada biti struktura binarnog stabla. Podsjetimo se, radi se o specifičnom ϵ-stablu definiranom precizno u pododjeljku 4.5.2. Tri su klase stabala, čija je hijerarhija opisana kroz skicu njihovih objava u ispisu 5.4 (vidjeti također dodatak A).

Ispis 5.4 *Hijerarhija nasljeđivanja klasa stabala.*

```
class CABinTree :  public CObject
{    ...    }

class CMainABinTree :  public CABinTree
{    ...    }

class CSubABinTree :  public CABinTree
{    ...    }
```

Klasa `CABinTree` (od engl. *Automata Binary Tree*) temeljna je apstraktna klasa iz koje podklase `CMainABinTree` (klasa glavnog stabala) i `CSubABinTree` (klasa podstabla) crpe svoju osnovnu funkcionalnost.

Ove klase u sebi ugrađuju dvije pomoćne klase: `CBinTreeNode` i `CTreeStatistics`. Za potpuni uvid u strukturu ovih klasa potrebno je posegnuti za dokumentacijom priloženom na disketi. Ukratko, svaki čvor se sastoji od brojača koji broji koliko je puta podniz određen danim čvorom pronađen u vremenskom nizu. Nadalje, čvor

sadrži i podatak o svojoj apsolutnoj razini s obzirom na korijen glavnog stabla, te kazaljke na svog roditelja, i lijevo i desno dijete. Klasa `CTreeStatistics` sadrži podatke o stablu kao cjelini, tj. broj njegovih lijevih i desnih čvorova, broj lijevih i desnih listova, te sumu odbrojaka u zasebno lijevim i desnim listovima. Ovi podaci će se pokazati kao iznimno korisni pri provjeri konzistencije ϵ-stabla, te pri usporedbi podstabala.

Programski kôd za klase stabala je relativno opsežan, pa je jasno da se ovdje ne možemo posvetiti mnogobrojnim interesantnim detaljima. U kratkim crtama ćemo se osvrnuti samo na ono najvažnije.

5.3.1 Temeljna klasa stabla – CABinTree

Već smo naveli da je klasa koja utemeljuje funkcionalnost ϵ-stabla klasa `CABinTree`. Jedan od originalnih doprinosa ovog rada je upravo razvoj nerekurzivnih algoritama za prolazak (engl *traversal*) kroz ovakva binarna stabla. Razvijeni su algoritmi za *prijeredni* (engl. *preorder*), *poslijeredni* (engl. *postorder*) i *međuredni* (engl. *inorder*) nerekurzivni prolazak kroz stablo. Pri tom se naziv odnosi na red posjete roditeljskog čvora u odnosu na djecu. Tako prijeredni prolazak posjećuje najprije roditelja, a potom djecu; poslijeredni najprije djecu a potom roditelja; a međuredni posjećuje najprije jedno (lijevo) dijete, pa roditelja, i potom drugo (desno) dijete.

Za rekurzivne strukture načelno se predlaže uporaba rekurzivnih algoritama. Pošto je stablo rekurzivna struktura, rekurzivni algoritmi su zaista najjednostavniji (za rekurzivne algoritme primijenjene na binarnom stablu vidjeti npr. [21]). Međutim, u slučaju stabala velike visine rekurzivni bi algoritam znatno opteretio memorijski stog i usporio rad. Zbog toga je umjesto jednostavnosti rekurzivnog pretraživanja ostvaren zamjenski nerekurzivni algoritam. Na taj način je složenost algoritma iz prostorno-vremenske domene prebačena u algoritamsku domenu[3]. Ovo predstavlja još jedan primjer, u skladu s razmatranjem u 5.1.2, da su kritični programski zahtjevi rješavani bez kompromisa. Sučelje osnovne klase stabla prikazano je u dodatku B.

5.3.2 Nerekurzivni algoritmi prolaska

Ideje za nerekurzivna pretraživanja općenitih (ne binarnih i "ne ϵ") stabala se mogu naći u npr. [19]. Zbog specifičnosti naših stabala algoritmi prolaska morali su biti

[3]Detaljnija analiza usporedbe rekurzivnog i nerekurzivnog algoritma prelazi okvire ovog rada, i bit će eventualno dana drugdje.

skrojeni "po mjeri", te uvažavati funkcionalne zahtjeve koje će postaviti izvedene klase. Npr. klasa podstabla se ostvaruje tako da se unutar glavnog stabla odredi čvor koji predstavlja njen korijen, i to novo podstablo će u potpunosti biti definirano tim korijenom i svojom dubinom. Drugim riječima, za čvorove podstabla nismo utrošili dodatnu memoriju. Također, jedna od specifičnosti naših stabala je i kriterij za određivanje listova:

Definicija 5.1 *Čvor je list u nekom (pod)stablu ako i samo ako je njegova relativna razina u odnosu na korijen stabla jednaka visini stabla.*[4]

Sada smo pripravni da kao primjer ostvarenja algoritma prijerednog prolaska kroz stablo, promotrimo sljedeću funkciju:

Ispis 5.5 *Funkcije InitPreOrd i TravrsPreOrd, za inicijalizaciju i prijeredni prolazak kroz binarno stablo.*

```
CBinTreeNode* CABinTree::InitPreOrd()
{
    // Consistency check:  the tree must be nonempty!
    ASSERT(m_pRoot != NULL);
    ASSERT((m_pRoot->GetnCount() != 0) || (m_pRoot->GetpLChild() ==
                    NULL) || (m_pRoot->GetpRChild() == NULL) );
    m_bDone = false;
    return m_pRoot; // Pre order traversal starts from the root.
}

CBinTreeNode* CABinTree::TravrsPreOrd(CBinTreeNode* pNdC)
{
    CBinTreeNode* pNd;
    if (Level(pNdC) < m_iHeight) // Not at the leaves' level.
    {                            // Move to the leftmost child:
        if( (pNd = pNdC->GetpLChild()) != NULL)
            pNdC = pNd; // To LChild.
        else
            pNdC = pNdC->GetpRChild(); // To RChild.
    }
    else // The current node (pNdC) is a leaf.
    {
        ASSERT(IsLeaf(pNdC)); // Consistency check!
        while (GetpRSibling(pNdC) == NULL) // No right sibling.
        {
            if (Level(pNdC) > 0)
```

[4] U teorijsko-računarskoj domeni umjesto termina *visina* (engl. *height*) stabla češće se koristi naziv *dubina* (engl. *depth*) stabla. U ovom radu koristimo oba naziva, s time da u programerskom dijelu (ovom poglavlju), programskom kodu i DSA programu rabimo termin *visina*.

```
                pNdC = pNdC->GetpParent(); // To Parent.
            else // At the root!!!
            {
                m_bDone = true; // Done. No more right siblings.
                return pNdC;
            }
        }
        pNdC = pNdC->GetpParent()->GetpRChild();
        // Move to the first right child available.
    }
    return pNdC;
}
```

Funkcija `InitPreOrd` služi za započinjanje prijerednog prolaska. Ona postavlja logičku člansku varijablu `m_bDone` na vrijednost `false`, određujući time da prolazak nije završen i potom vraća vrijednost kazaljke na korijen stabla.

Očekujemo da se funkcija `TravrsPreOrd` rabi nakon što je prethodno (na početku) provedena inicijalizacija. Tada ćemo njenom uzastopnom primjenom dobiti korektan prolazak kroz stablo, tj. čvor će biti posjećen jednom i samo jednom, i to će vrijediti za svaki čvor u stablu. Daljnji tijek algoritma najbolje je uočljiv iz gornjeg ispisa i priloženih komentara. Iako je algoritam znatno kompleksniji od rekurzivne inačice, njegova je uporaba jednostavnija. Naime, rezultat funkcije je jednostavno kazaljka na sljedeći čvor. To nije slučaj kod primjene rekurzije, gdje se funkcija koja se na čvoru vrši mora ugraditi unutar rekurzivnih poziva[21].

Zainteresirani čitatelj može u dodatku B pronaći funkcije `TravrsInOrd`, `TravrsPostOrd` i njima odgovarajuće funkcije za inicijalizaciju prolazaka kroz stabla. Svaki od ovih prolazaka ima svoje odlike koje nam mogu poslužiti u danim okolnostima. Uobičajen je prijeredni prolazak. On se, primjerice, koristi kod nalaženja morfova. S druge strane, kod uklanjanja (engl. *delete*) svih čvorova glavnog stabla (funkcija `EmptyMainTree`), najsvrsishodnije je rabiti poslijeredni prolazak.

5.3.3 Funkcija CompareTo

Jedna od najkompleksnijih funkcija ove klase je funkcija `CompareTo`. Njezina je zadaća usporedba promatranog stabla s drugim stablom. Funkcija vraća vrijednost tipa `float`, s dvojakom namjenom. Ukoliko su stabla morfološki različita tada je vrijednost funkcije 2.0, što interpretiramo kao zastavicu. Ukoliko su stabla morfološki jednaka tada funkcija vraća maksimalnu apsolutnu razliku vjerojatnosti uočenu između dva čvora. Također, u tom slučaju varijabla `sMaxDiffNd` navodničkog tipa (engl. *ref-*

erence type) `CString&` predstavlja niz kojim je opisan čvor promatranog stabla, za koji se dešava najveće vjerojatnosno odstupanje.

U ispisu 5.6 prikazana je ova funkcija. Ona predstavlja dobru ilustraciju programske problematike našeg rada.

Ispis 5.6 *Implementacija algoritma usporedbe stabala: funkcija CompareTo.*

```
float CABinTree::CompareTo(CABinTree* pTr, CString& sMaxDiffNd)
{
    float fDiffMax = 0.f;
    float fFlg;
    CBinTreeNode* pNdC;   // Pointer to the current node of "this" tree.
    CBinTreeNode* pNdCT; // Pointer to the current node of *pTr tree.

    ASSERT( !IsEmpty() && !pTr->IsEmpty() );
    ASSERT(m_iHeight == pTr->m_iHeight);

    if( !(m_TreeStat == pTr->m_TreeStat) ) // Different tree
        return fFlg = 2.f;                 // statistics.

    pNdC = InitPreOrd();         // Initialize preorder traversal
    pNdCT = pTr->InitPreOrd(); // for both trees.

    // Compare roots:
    if(HasChildren(pNdC) == pTr->HasChildren(pNdCT))
    {
        pNdC = TravrsPreOrd(pNdC);
        pNdCT = pTr->TravrsPreOrd(pNdCT);
        while( !GetbDone() ) // Check all the nodes below the root:
        {
            if(  HasChildren(pNdC) == pTr->HasChildren(pNdCT) &&
                (IsLeftChild(pNdC)  && pTr->IsLeftChild(pNdCT) ||
                 IsRightChild(pNdC) && pTr->IsRightChild(pNdCT)) )
            // The nodes are topolog. equal, comp. their probabilities:
            {
                fFlg = float(fabs(NodeProblty(pNdC) -
                    pTr->NodeProblty(pNdCT)));
                if ( fFlg > fDiffMax )
                {
                    fDiffMax = fFlg;
                    sMaxDiffNd = NodeFullString(pNdC);
                }
            }
            else
            {
                return fFlg = 2.f; // Morphologically different trees.
            }
            pNdC = TravrsPreOrd(pNdC);
```

```
            pNdCT = pTr->TravrsPreOrd(pNdCT);
        } // while
        fFlg = fDiffMax;
    }
    else // The difference at the root:
    {
        fFlg= 2.f; // Morphologically different trees.
    }
    return fFlg;
}
```

Na samom početku funkcija uspoređuje morfološki relevantne elemente (broj lijevih i desnih čvorova, te broj lijevih i desnih listova) ranije spomenute statistike stabala. Ukoliko postoji takva razlika, tada su stabla nužno morfološki različita i funkcija vraća zastavicu 2.0 . Naravno, obrat ne vrijedi. Stabla mogu biti morfološki različita i uz isti broj lijevih i desnih čvorova i listova, ali drugačije raspoređenih. To će biti provjeravano u idućim koracima algoritma.

Nadalje se začinje prijeredni prolazak kroz oba stabla. Uspoređuju se korijeni glede njihove nasljedne strukture, tj. da li posjeduju samo lijevo, samo desno ili oba djeteta. Ako korijeni nemaju istu strukturu nasljednika, tada su i podstabla morfološki različita, te ponovo izlazimo iz funkcije. Ukoliko su korijeni jednaki, vrši se daljnji korak prijerednog prolaska kroz oba stabla. Za odgovarajuće čvorove oba stabla ispituje se njihov karakter, tj. položaj u strukturi stabla. Najprije se, slično kao za korijene, ustvrđuje da li im je struktura nasljednika jednaka, a potom da li su oni u jednakom odnosu naspram svojih roditelja. Ako to ne vrijedi, stabla su morfološki različita i izlazi se iz funkcije sa zastavicom 2.0 . Ako nije uočena morfološka različitost nastavlja se s vjerojatnosnom usporedbom. Najveća uočena vjerojatnosna razlika i nizovni prikaz čvora kod kojeg je ona uočena, bilježi se u lokalnim varijablama. Postupak se zatim nastavlja za sljedeći par čvorova dobiven prijerednim prolaskom kroz stablo.

5.3.4 Klasa glavnog stabla

Već je istaknuto da je klasa `CABinTree` apstraktna klasa, koja se konkretizira tek u svojim nasljednicama: `CMainABinTree` (engl. *Main Automata Binary Tree*) i `CSubABinTree` (engl. *Sub Automata Binary Tree*). Glavno stablo je pri tom jedino stablo u okviru kojeg se kreiraju čvorovi. To se odvija tijekom procesa hranidbe, o kojem je već bilo riječi. Najprije ćemo obrazložiti funkciju hranidbe stabla visine `m_iHeight` s binarnim podnizom iste duljine, što je prikazano u ispisu 5.7.

Ispis 5.7 *Hranidba stabla binarnim podnizom, funkcija FeedWithABinString.*

```
void CMainABinTree::FeedWithABinString(BYTE* p)
// Feeds a string of binary values into the CMainABinTree.
{
    CBinTreeNode* pNewNode;
    CBinTreeNode* pChild;
    CBinTreeNode* pNdC = GetpRoot();
// Construct new or increment the old nodes below the root:
    for (UINT j = 0; j < (GetiHeight() - 1); j++)
    {
        ASSERT( (*p == 0x00) || (*p == 0x01) );
        // Internal consistency check!
        if (*p == 0x00)
        {
            pChild = pNdC->GetpLChild();
            if ( pChild == NULL) // Make a new left child:
            {
                pNewNode = new CBinTreeNode(j + 1);
                ASSERT(pNdC->GetnAbsLev() == j);
                // Internal consistency check!
                m_TreeStat.IncrLNodes();
                pNewNode->SetpParent(pNdC);
                pNdC->SetpLChild(pNewNode);
            }
            else // Increment the old left child:
                pChild->IncrnCount();

            pNdC = pNdC->GetpLChild(); // Move to the left child.
        }
        else // (*p == 0x01)
        {
            pChild = pNdC->GetpRChild();
            if (pChild == NULL) // Make a new right child.
            {
                pNewNode = new CBinTreeNode(j + 1);
                ASSERT(pNdC->GetnAbsLev() == j);
                // Internal consistency check!
                m_TreeStat.IncrRNodes();
                pNewNode->SetpParent(pNdC);
                pNdC->SetpRChild(pNewNode);
            }
            else // Increment the old right child:
                pChild->IncrnCount();

            pNdC = pNdC->GetpRChild(); // Move to the right child.
        }
        p++;
    } // The end of the for loop.
```

```
    // Leaves' level - construct new or increment old leaves:
    //   repeat the same procedure for the leaves' level and addition-
    //   ally increment the m_TreeStat SumLLvsCnts and SumRLvsCnts.
    //  [For that part of the code see DynSysA_CFiles.]
    ...                    ...                    ...

    ...                    ...                    ...

    //
    // A successful feed. To note it, increment the counter (element)
    // in the root:
    GetpRoot()->IncrnCount();
}
```

Jedini argument funkcije `FeedWithABinString` je kazaljka na tekuće mjesto početka binarnog podniza. Algoritam je relativno jednostavan. Pročita se prvi znak podniza te ako je on 0 provjerava se lijevo dijete. Ako ono ne postoji, kreira se. Novonastalo dijete se odgovarajuće poveže s roditeljem, a njegov brojač inicijalizira na 1. Ako lijevo dijete postoji, njegov brojač se inkrementira. Zatim se kazaljka tekućeg čvora prebaci na to lijevo dijete i postupak se nastavlja. U slučaju da je znak bio 1, postupa se na isti način s desnim djetetom. U oba slučaja se odgovarajuće obnavlja statistika stabla. Zbog toga što se statistika listova provodi na poseban način, razina listova se obravnava u izdvojenom dijelu programskog koda, koji u gornjem ispisu nije prikazan. Ova je funkcija vremenski kritična i stalno provjeravanje da li se radi o razini listova je izbjegnuto povećanjem programskog koda. Ispušteni dio funkcije je u bitnom istovjetan, osim u detaljima obnove statistike listova.

Isti je stil programiranja rabljen i kod funkcije `FeedWithBinStrings`, čiji je zadatak da omogući hranidbu stabla s proizvoljnim brojem podnizova. Ona se poziva izravno iz izbornika, odnosno na pritisak dugmeta u dijaloškom prozoru. Funkcija treba omogućiti pohranu i jednog jedinog podniza, što je korisno kod testiranja i prezentacije programa, pa do $\approx 10^9$ podnizova, kao gornje granice sadašnje implementacije brojača s 32-bitnim nepredznačenim cjelobrojnim tipom. Tu je, naravno, primijenjeno načelo prilagodbe algoritma prvenstveno za ove druge slučajeve, koji su kritični glede velikog broja ponavljanja. Organizacija našeg kružnog spremnika za pohranu rezultata grupne obrade, dolazi ovdje do punog izražaja. Sama funkcija je relativno opsežna i kompleksna glede svoje konkretne realizacije, ali zato što ne posjeduje veću algoritamsku posebnost, nećemo je detaljnije razmatrati.

Opisane "rigorozne" mjere za optimizaciju vremenski kritičnih dijelova DSA programa primjer su naših stalnih nastojanja da se zadana programska načela beskom-

promisno poštuju. Uza sve učinjeno, za hranidbu raščlambenog stabla visine 16 za sustav logističkog preslikavanja s $r = 4$ (kaotični režim, koji generira puno stablo), uz hranidbeni faktor = 1024 (vidjeti 6.1.2), u konačnoj (engl. *release*) inačici programa treba sveukupno oko 1.5 min.[5] Tu je uključeno i vrijeme potrebno da se izračuna 67.1×10^6 točaka orbite i odredi isto toliko bitova binarnog vremenskog niza (pohranjenih u kružnim spremnicima). Potom se iz tog niza izlučuje otprilike isto toliko riječi duljine 16 i parsira kroz raščlambeno stablo. Za povećanje točnosti vjerojatnosti čvorova za jedno decimalno mjesto valja hranidbeni faktor povećati za red veličine, tj. učitati barem 10 puta više riječi. To zahtijeva proporcionalno, dakle oko 10 puta, veće vrijeme (izmjereno $\approx$ 15 min). Za viša glavna stabla pak računarski zahtjevi rastu kao i broj listova stabla — eksponencijalno s njegovom visinom. Očito je da bi svako memorijski pojednostavljeno rješenje, kao npr. izračun orbite točku po točku, zbog potrebe stalnog pozivanja funkcije iteracije znatno pogoršalo vremenske konstante, a time i učinkovitost ove kritične faze modeliranja.

Ova diskusija ujedno pokazuje da je *interpreterska izvedba* iteracijske funkcije — koja bi unutar neke propisane forme korisniku omogućila unos proizvoljne jednadžbe sustava — za naš projekt neprikladna. Bitno produljeno vrijeme takvog rješenja predstavljalo bi problem kod iteracija 10^6, 10^7, pa i 10^9 točaka, kakve su javljaju u našem modeliranju. Stoga je naše rješenje — u kojem se za svaki sustav kreira njegova (pod)klasa i unutar nje definira iteracijska funkcija — itekako opravdano većom brzinom izvođenja iteracija. Naravno, takva je specifikacija iteracijskih funkcija i potpuno slobodna, tj. lišena sintaktičkih ograničenja interpreterskog rješenja.

5.4 Klasa morfova

Algoritam nalaženja ϵ-strojeva kulminira u prepoznavanju jedinstvenih morfova kao stanja budućeg modela. U našem slučaju će morfovi odgovarati podstablima jedinstvenih morfoloških i vjerojatnosnih osobina, pronađenim unutar glavnog stabla, što je detaljno diskutirano u odjeljku 4.5.

Morfovi su opisani klasom `CATrMorphs` (*Automata Tree Morphs*) koja u sebi ugrađuje pomoćnu klasu `CMrphTrans`. Funkcije u kojima se ogleda većina svojstava ove klase su one za nalaženje pojedinačnih podmorfova (`FindRootMorph`, `FindChildMorphs`) te za nalaženje svih morfova (`FindAllMorphs`). Ove funkcije slijede algoritam 4.2 iz prethodnog poglavlja.

[5] Na PC računalu s procesorom Pentium II na 350 MHz i operacijskim sustavom Windows NT.

5.4.1 Funkcija FindRootMorph

Prema gore spomenutom algoritmu, najprije je potrebno iznaći korijenski morf. To je zadaća funkcije `FindRootMorph` koja je prikazana u ispisu 5.8.

Ispis 5.8 *Funkcija za ekstrakciju korijenskog morfa: FindRootMorph.*

```
bool CATrMorphs::FindRootMorph()
{
    if(m_pMainTree->IsEmpty())
        return false;

    CSubABinTree* pSTr = (m_pTrvrsTree->GetpRoot());
    CSubABinTree* pSTr = NewSubTrMrphCand(m_pTrvrsTree->GetpRoot());
        // New subtree as a possible new morph candidate.
    ASSERT(m_MorphsArr.GetSize() == 0);
    ASSERT(m_MrphTrnsArr.GetSize() == 0);

    m_MorphsArr.Add(pSTr); // The subtree emanating from the root
          // is always a unique morph, since it is the first one.
         // Add this subtree as the first morph, with index 0, to the
        // m_MorphsArr array of morphs.
    CMrphTrans* pTrn = new CMrphTrans("", 2.f);
    m_MrphTrnsArr.Add(pTrn); // Add new CMrphTrns object to
                            // m_MrphTrnsArr.
    return true;
}
```

Ova funkcija vraća vrijednost `false` ako je glavno stablo prazno. Inače, za neprazno stablo, s pomoću funkcije `NewSubTrMrphCand` stvara se novo podstablo konstantne visine definirane pri kreaciji objekta iz klase `CATrMorphs`. S pomoću makronaredbi `ASSERT`, koje operiraju u ispravljačkoj (engl. *debug*) inačici programa, želimo se uvjeriti da je ova funkcija primijenjena prije nego što je pronađen bilo koji drugi morf, što je preduvjet za konzistenciju algoritma. Kazaljka `pSTr` na korijensko podstablo se zatim umeće u *poredak predložaka* (engl. *template array*) `m_MorphsArr` tipiziranih kazaljki, koji je ugrađen u klasu. Uporaba ovih naprednijih inačica poredaka dodanih u jezik C++ je motivirana njihovom većom fleksibilnošću i sigurnom provjerom tipova. Stoga je ona opravdana uvijek kad se ne radi o vremenski kritičnim dijelovima programa.

Nadalje, za svaki novi morf (podstablo) se stvara novi objekt klase `CMrphTrans`, koji je zasada inicijaliziran na početne vrijednosti. U ovom objektu će se zabilježiti podaci o tome koji je lijevi i desni podmorf datog morfa, te koje su odgovarajuće vjerojatnosti prijelaza.

5.4.2 Funkcija FindChildMorphs

Uočavanje svih morfološki i δ-različitih podmorfova svakog pojedinog morfa je zadatak funkcije `FindChildMorphs` prikazane u ispisu 5.9. Ona je jedna od hijerarhijski najsloženijih funkcija ovog projekta, u smislu da poziva mnogobrojne funkcije različitog značaja, koje su i same srazmjerno visoke razine apstrakcije. Tipičan primjer za to je poziv funkcije `CompareTo` diskutirane u prethodnom odjeljku.

Argumenti funkcije `FindChildMorphs` su `fDelta` koji ima ulogu parametra δ (vidjeti izraze (4.21), (4.51) i pododjeljak 4.5.7), te indeks morfa čije podmorfove tražimo. Prije nego se posvetimo analizi ove funkcije, primijetimo da se ona u bitnom sastoji od dva dijela, od kojih prvi vodi računa o (mogućem) lijevom, a drugi o (mogućem) desnom podmorfu. Ova dva dijela funkcije su, razumljivo, simetrična glede zamjene *lijevo* $\rightarrow$ *desno* i obratno.

Ispis 5.9 *Funkcija FindChildMorphs za nalaženje podmorfova.*

```
UINT CATrMorphs::FindChildMorphs(float fDelta, int im)
{
    CSubABinTree* pSTr;
    UINT nMorphs = GetnMorphs();

    float fFlg;      // Local variable for the CompareTo output flag.
    float fMinDiff; // Minimal delta-difference that is found.
    int iMinDiff;  // Index of the morph for which fMinDiff is found.

    CString sMaxDiffNd; // The string of the node with max. prob. diff.
    CMrphTrans* pTrn;
    UINT        iNewMorphs = 0;

    // The root node of the im-th subtree in morphs array:
    CBinTreeNode* pNdT = m_MorphsArr.GetAt(im)->GetpRoot();

    if (m_pTrvrsTree->IsLeaf(pNdT))
        return iNewMorphs;

    // Make a child tree:
    hasChldrnType chldType = m_pTrvrsTree->HasChildren(pNdT);

    if( (chldType == leftOnly) || (chldType == both) )
    {
        fFlg = fMinDiff = 2.f;
        iMinDiff = -1;
        sMaxDiffNd = "";
        pNdT = pNdT->GetpLChild();
        pSTr = NewSubTrMrphCand(pNdT); // Caution! Delete this sub-
```

```
    // tree if it will not be added to the m_TreeMrphsArr.
    // Compare the subtree to all the morphs in the array:
    for (UINT i = 0; (fFlg > fDelta) && (i < nMorphs) ; i++)
    {
        fFlg = pSTr->CompareTo(m_MorphsArr.GetAt(i), sMaxDiffNd);
        if(fFlg < fMinDiff)
        {
            fMinDiff = fFlg; iMinDiff = i;
        }
    }

    if(fFlg <= fDelta)  // The subtree is equal to the
    {                   // morph with the index i - 1.
        ASSERT(fFlg == fMinDiff);
        pTrn = m_MrphTrnsArr.GetAt(im);
        if(pTrn->m_iLSubMrph == -1)
        {
            pTrn->m_iLSubMrph = i - 1;
            pTrn->m_fLSubProb = pSTr->SubTreeProblty();
        }
        // This branch should not be visited anymore!
        delete pSTr; // Delete the subtree created with "new".
    }
    else // The subtree is a new morph!
    {
        pTrn = m_MrphTrnsArr.GetAt(im);
        pTrn->m_iLSubMrph = m_MorphsArr.Add(pSTr);
        pTrn->m_fLSubProb = pSTr->SubTreeProblty();
        pTrn = new CMrphTrans(pSTr->GetsRoot(), fMinDiff);
        pTrn->m_iMinDiff = iMinDiff;
        pTrn->m_sMaxDiffNd = sMaxDiffNd;
        m_MrphTrnsArr.Add(pTrn);
        iNewMorphs++; nMorphs++;
    }

    if(chldType == both)
        pNdT = pNdT->GetpParent();
}

if( (chldType == rightOnly) || (chldType == both) )
{
    fFlg = fMinDiff = 2.f;
    iMinDiff = -1;
    sMaxDiffNd = "";
    pNdT = pNdT->GetpRChild();
    pSTr = NewSubTrMrphCand(pNdT); // Caution!  Delete this sub-
    // tree if it will not be added to the m_TreeMrphsArr.
    // Compare the subtree to all the morphs in the array:
```

```
        for (UINT i = 0; (fFlg > fDelta) && (i < nMorphs); i++)
        {
            fFlg = pSTr->CompareTo(m_MorphsArr.GetAt(i), sMaxDiffNd);
            if(fFlg < fMinDiff)
            {
                fMinDiff = fFlg; iMinDiff = i;
            }
        }

        if(fFlg <= fDelta)  // The subtree is equal to the
        {                   // morph with the index i - 1.
            ASSERT(fFlg == fMinDiff);
            pTrn = m_MrphTrnsArr.GetAt(im);
            if(pTrn->m_iRSubMrph == -1)
            {
                pTrn->m_iRSubMrph = i - 1;
                pTrn->m_fRSubProb = pSTr->SubTreeProblty();
            }
            // This branch should not be visited anymore!
            delete pSTr; // Delete the subtree created with "new".
        }
        else // The subtree is a new morph!
        {
            pTrn = m_MrphTrnsArr.GetAt(im);
            pTrn->m_iRSubMrph = m_MorphsArr.Add(pSTr);
            pTrn->m_fRSubProb = pSTr->SubTreeProblty();
            pTrn = new CMrphTrans(pSTr->GetsRoot(), fMinDiff);
            pTrn->m_iMinDiff = iMinDiff;
            pTrn->m_sMaxDiffNd = sMaxDiffNd;
            m_MrphTrnsArr.Add(pTrn);
            iNewMorphs++; // nMorphs++; // (Not needed anymore!).
        }
    }

    return iNewMorphs;
}
```

Na početku funkcije lokalna se varijabla `iNewMorphs`, za broj nađenih morfova, inicijalizira na nulu. U poretku `m_MorphsArr` se nađe kazaljka na podstablo s indeksom `im`. Kazaljka na korijen tog podstabla se pohrani u lokalnu varijablu `pNdT`, koja ima značenje kazaljke na tekući čvor.

Za razumijevanje sljedećeg koraka potrebno je objasniti koncept *prolaznog stabla* (*Traversal Tree*) pokazanog članskom varijablom `m_pTrvrsTree`. To je podstablo unutar glavnog stabla, čija je visina, izražena rječnikom prošlog poglavlja jednaka $D - L$ (D je visina glavnog stabla, a L je visina morfa). Podmorfove možemo tražiti samo za ona stabla čiji su korijeni čvorovi tog prolaznog stabla iznad razine

listova. Drugim riječima, za podstabla kojima su korijeni na razini listova prolaznog stabla, podmorfovi imaju korijene na razini $D-L+1$, pa oni ne mogu biti legalna ϵ-podstabla visine L. Dakle, ukoliko je morf list prolaznog stabla, njegovi podmorfovi nisu dobro definirani, pa se funkcija vraća s `iNewMorf = 0` pronađenih podmorfova.

Ukoliko je promatrani morf iznad razine listova u prolaznom stablu, provjerava se struktura nasljednika njegovog korijena, iz čega se doznaje da li on ima samo lijevi, samo desni ili oba podmorfa. Promotrimo prvu granu algoritma, za slučaj da morf ima lijevi ili oba podmorfa. Na početku traženja eventualno sličnog morfa postavljamo `fMinDiff` na vrijednost 2.0, koja prema diskusiji funkcije `CompareTo` (ispis 5.6) znači morfološku različitost. Također, odgovarajući indeks `iMinDiff` najsličnijeg morfa postavljamo na -1, što označava da ne postoji morfološki jednako podstablo, a niz `sMaxDiffNd` za specifikaciju čvora u kojem se pojavilo maksimalno vjerojatnosno odstupanje, postavljamo da bude prazan niz.

Pošto je pri provjeri utvrđeno da postoji lijevo dijete, pomaknemo tekući čvor na lijevo dijete, i s pomoću funkcije `NewSubTrMrphCand` tvorimo novo podstablo na koje pokazuje kazaljka `pSTr`. Ono je potencijalni kandidat za jedinstveni morf. U petlji koja slijedi, stablo pokazano s `pSTr` uspoređujemo sa svim dosad iznađenim jedinstvenim podstablima. Ujedno se bilježi za koje od već postojećih stabala je funkcija `CompareTo` vratila minimalnu zastavicu `fFlg`. Ako nikoje postojeće podstablo nije morfološki jednako našem kandidatu, tada će `fMinDiff` ostati na 2.0 , i to je znak da je kandidat morfološki jedinstven. Ako je za neko od podstabala funkcija `CompareTo` ustanovila da je morfološki jednako našem kandidatu, tada će `fMinDiff` postati manje od 1.0, odražavajući vjerojatnosnu razliku. Ukoliko je uočena vjerojatnosna razlika manja od zadane δ vrijednosti, ili ako više nema podstabala za usporedbu, izlazimo iz petlje.

Po izlasku iz petlje provjerimo da li je izlaz bio uvjetovan time što je nađen δ-ekvivalentan morf. Ako jest, u objekt tipa `CMrphTrans`, pospremljen u poretku predložaka `m_MrphTrnsArr` kao element s indeksom `im`, zabilježimo da lijevi podmorf odgovara morfu (podstablu) uvrštenom u `m_MorphsArr` pod indeksom `i - 1`. Također zabilježimo vjerojatnost tog prijelaza. Pošto se kandidat nije pokazao kao jedinstven, uklonimo ga operatorom `delete` koji poziva destruktor za klasu podstabla.

Ukoliko smo iz petlje izašli bez da je pronađen δ-ekvivalentan morf, tada je naš kandidat δ-jedinstven, te ga uvrštavamo u `m_MorphsArr` kao novi morf. Ponovimo radnje za određenje prijelaza i prijelazne vjerojatnosti, analogno kao u prijašnjem

paragrafu, te zabilježimo indeks morfa kojem je kandidat najbliži (ako je `fMinDiff` ostalo na 2.0, tada takav kandidat ne postoji, odnosno `iMinDiff` je -1). Ovaj će podatak biti od presudne važnosti za analizu veličine δ-parametra, što će biti prikazano u sljedećem poglavlju. Također, ukoliko je `fMinDiff` manje od 2.0, tada zabilježimo u kojem se čvoru pojavila maksimalna vjerojatnosna razlika. Konačno, inkrementiramo brojač `iNewMorphs` novonađenih morfova, te u slučaju da su postojala oba podmorfa, pripremimo tekući čvor `pNdT` za provjeru desnog podmorfa. Nalaženje desnog podmorfa je analogno nalaženju lijevog. Funkcija vraća broj pronađenih morfova.

5.4.3 Funkcija FindAllMorphs

Zahvaljujući svrsishodnosti prethodno izložene funkcije `FindChildMorphs`, funkcija `FindAllMorphs` je srazmjerno jednostavna. Ona je prikazana u ispisu 5.10:

Ispis 5.10 *Nalaženje svih δ-jedinstvenih morfova u glavnom stablu.*

```
void CATrMorphs::FindAllMorphs()
{
    if (FindRootMorph())
    {
        int sum = 1;
        int i = 0;
        do
        {
            int find = FindChildMorphs(i++);
            ASSERT( (sum += find) == GetnMorphs() );
        }
        while(i < GetnMorphs());
    }

    m_bSearchDone = true;
    ConserveSubTrStatistics();
}
```

Postupak traženja svih morfova inicijalizira se pozivom funkcije `FindRootMorph` i stavljanjem internog brojača `i` na 0. Zatim se poziva funkcija `FindChildMorphs(0)`, sa zadatkom da se pronađu podmorfovi korijenskog morfa. Lokalne varijable `sum` i `find` uvedene su samo radi ostvarenja neovisnosti provjere u makro-naredbi `ASSERT`. Postupak se ponavlja sve dok ima morfova kojima nismo provjerili podmorfove. Primijetimo da će pribavljačka funkcija `GetnMorphs` dati uvijek tekuće stanje broja morfova, koje se općenito mijenja unutar `FindChildMorphs`.

Mada se na prvi pogled može učiniti da je ovaj algoritam sklon divergiranju, to nije tako. Istina je da svaki od pronađenih morfova može imati do dva jedinstvena podmorfa, a novopronađeni morfovi jednako toliko itd. No maksimalni broj morfova određen je brojem čvorova prolaznog stabla, za koji vrijedi da ne prelazi gornju granicu od $2^{D-L+1} - 1$. Štoviše, ovaj se algoritam pokazao kao izrazito efikasan.

Daljnja analiza ovog algoritma, kao i složena problematika odabira δ-parametra prelazi okvire ovog rada. O parametru δ će još biti riječi tijekom prezentacije rezultata, a u ovom poglavlju dajemo još letimičan pogled na ostale implementirane klase.

5.5 Osvrt na ostale klase

Klase predstavljene u ovom poglavlju predstavljaju tek mali dio onoga što je ukupno moralo biti implementirano da bi se ostvarila sadašnja funkcionalnost programa. Ukratko, napisano je 14100 linija koda, za što je utrošeno preko 1000 programerskih sati. Ukupna programska dokumentacija, koja se sastoji od ispisa svih zaglavnih i implementirajućih C++ datoteka, sadrži preko dvije stotine stranica s prosječno oko 70 redova po stranici.

Usprkos tome, ostvarena funkcionalnost je ipak samo temeljne razine, i kao što je diskutirano u odjeljku 5.1, mnoga od programskih načela zaslužuju dodatnu pažnju. Sve to upućuje na znatnu složenost postavljenog zadatka.

Velika funkcionalnost ostvarena je u prezentaciji strukture stabla, što se može smatrati i jednom od posebnih odlika ovog rada. Prva ideja vodilja je bila da se mora ostvariti pregledan alat za učinkovito testiranje svih originalnih algoritama u svezi sa stablima, i preko njih definiranih morfova. Tijekom razvoja su ovom korisničkom sučelju dodavana sve nova i nova svojstva, te dodatne mogućnosti. Glavna klasa za to je klasa `CTreeFeedAndShowDlg` koja je nasljednica standardne MFC[6] klase `CDialog`. Ona omogućuje promatranje čvorova stabla, odnosno stanja brojača i odgovarajućih vjerojatnosti (bilo vjerojatnosti prijelaza, bilo vjerojatnosti čvora s obzirom na korijen itd.). Uveden je koncept *pokaznog stabla* (*Show Tree*) visine 4, s pomoću kojeg korisnik može promatrati pojedine dijelove proizvoljno visokog glavnog stabla. To je omogućilo i lako testiranje algoritama te ostvarenje onog što je kod stabala velike visine najteže — dobra vizualizacija.

Iako opsežna, s oko 2400 linija koda, ova klasa još uvijek ne uključuje sve pogod-

[6] Microsoft Foundation Class library.

nosti koje bi od grafičkog sučelja mogao očekivati današnji korisnik. Za temeljnu svrsishodnost programa to i nije nužno, odnosno ostaje kao mogućnost u daljnjem razvoju.

Funkcionalnost klase `CTreeFeedAndShowDlg` koristi i iz nje izvedena klasa `CMorphsFindAndShowDlg`, koja služi kao sučelje pri nalaženju morfova.[7] Dobar je primjer korištenja objektno orijentirane paradigme činjenica da je ova klasa napisana sa svega oko 500 linija, iako postoji dosta razlika i preinaka koje su morale biti ostvarene. Npr. nužno je bilo uvesti pojam morfova kao podstabala neovisno o pojmu pokaznog stabla, te integrirati pojam prolaznog stabla unutar glavnog.

Mada je u ostvarenju sučelja bilo izrazito zanimljivih problema s računarskog gledišta, zato što oni ne predstavljaju okosnicu algoritma za nalaženje ϵ-strojeva, ovdje ih nećemo obrazlagati.

[7] Za ovu klasu i pripadno sučelje treba uzeti u obzir da neki postojeći grafički elementi koji nisu bili presudni za konačne rezultate projekta, još nisu u punoj funkcionalnosti.

Poglavlje 6

Primjena i rezultati

Kao što se može uočiti iz strukture dosadašnjeg izlaganja, težište ovog rada bilo je pomaknuto prema formalizaciji teorije ϵ-strojeva i ostvarenju odgovarajuće programske podrške kao alata za novi pristup modeliranju dinamičkih sustava. U ovom poglavlju dat ćemo pregled nekih jednostavnijih sustava kao ilustraciju primjene našeg programa. Uporaba jednostavnih sustava, za koje se ponašanje može analitički predvidjeti i zatim uspoređivati s onim što daje program, je presudna u fazi provjere. Jednako je to važno i za prezentaciju programa, odnosno za upoznavanje korisnika s onim što se može očekivati od modela. Prelaskom na složenije primjere i, na koncu, na analizu nelinearnih dinamičkih sustava, otvaramo pravo područje primjene. Na kraju poglavlja izloženi su preliminarni rezultati u istraživanju nelinearnog dinamičkog sustava predstavljenog logističkom jednadžbom.

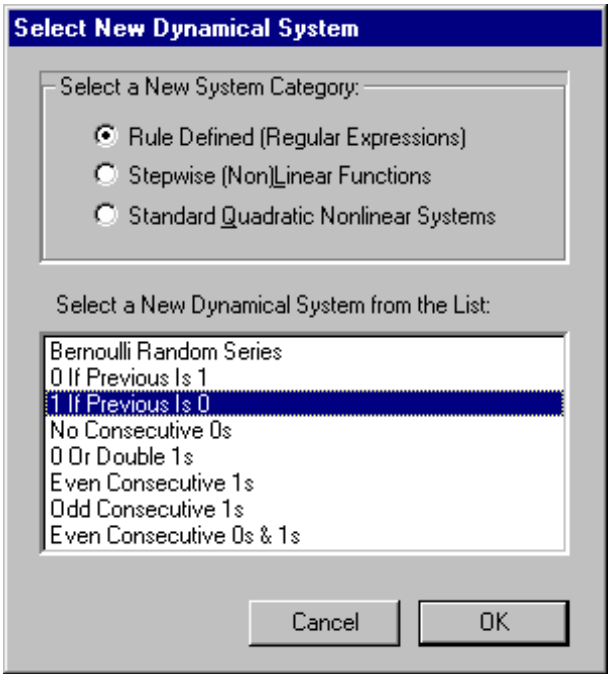
6.1 Primjer uporabe programa

Upoznavanje s programom ćemo provesti kroz analizu jednostavnog dinamičkog sustava "nema uzastopnih nula", s kojim smo se već dobro upoznali na kraju poglavlja 4, u pododjeljku 4.5.3, te u prošlom poglavlju u 5.2.2.

6.1.1 Izbor i iteracija sustava

Izbor sustava se vrši odabirom *Select New* u izborniku *System.* Otvara se prozor dijaloga u kojem korisnik može odabrati tri načelno različite skupine sustava (slika 6.1)

Kao što je diskutirano u prethodnom poglavlju, sustav "generiraj 1 ako je pre-



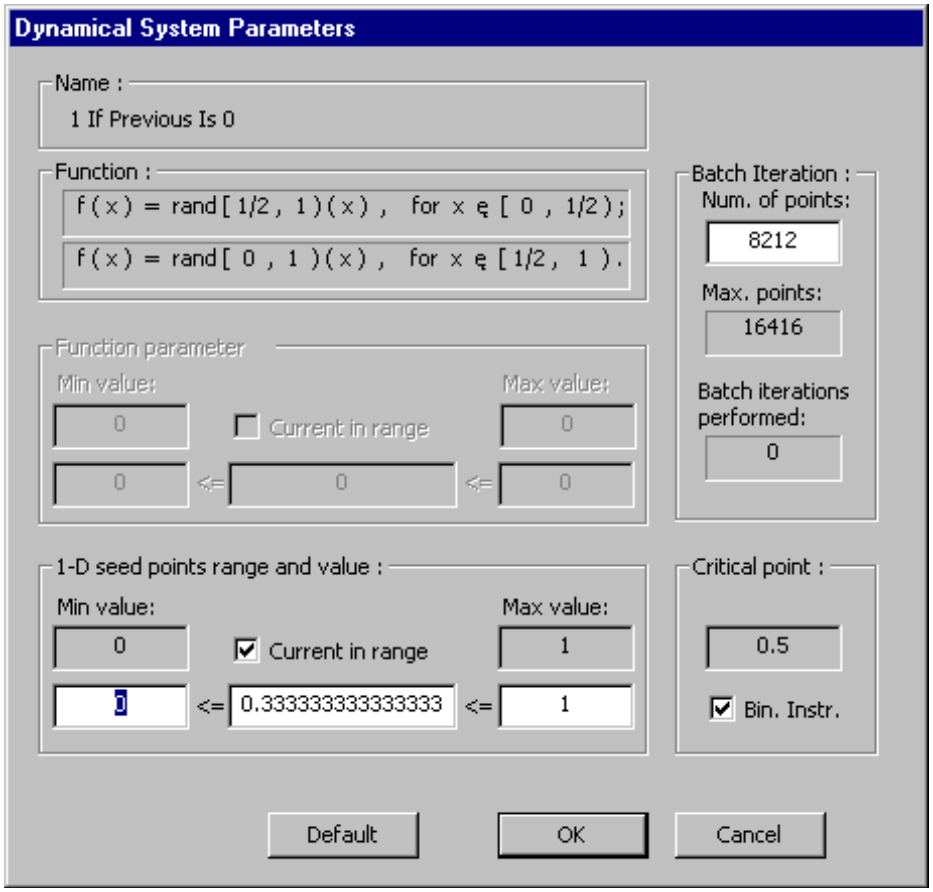
Slika 6.1: Dijaloški prozor za odabir sustava. Dinamički sustav se može izabrati iz tri raznorodne skupine. Na slici su prikazani sustavi zadani "pravilom", odnosno u terminima teorije formalnih jezika, pravilnim izrazom. Na slici je odabran sustav: "generiraj 1 ako je prethodilo 0". Sustav "nema uzastopnih nula" je ekvivalentan, uz nešto drugačiju programsku izvedbu.

thodilo 0", odabran na slici i onaj neposredno ispod ("nema uzastopnih nula"), su ekvivalentni sustavi dobiveni različitom implementacijom iteracijske funkcije.

Nakon što je sustav odabran, otvara se prozor za podešavanje njegovih parametara (slika 6.2). Na slici su prikazani podrazumijevajući parametri. Ovaj dijaloški prozor služi kao korisničko sučelje prema članskim varijablama klase `CDynSys1D` i njenih podklasa. Ulazne vrijednosti se prilagođuju posebice svakom tipu sustava, prema njegovoj prirodi.

Po određenju parametara korisnik može provjeriti tijek iteracije izborom stavke *Batch Iteration* u izborniku *System.* Kao što je već naglašeno u prethodnom poglavlju, iteracije se vrše za sve točke unutar kružnog spremnika odjednom, kako bi se ubrzalo nalaženje dugačkih orbita. Izgled dijaloškog prozora dan je na slici 6.3. Prikazane vrijednosti odgovaraju stanju nakon izvršenih 10 000 grupnih iteracija, odnosno nakon što je izračunato oko 82×10^6 točaka orbite. Vrijednosti točaka se mogu provjeriti unutar samog dijaloga, ili ispisati na predočniku.

Lista početka orbite našeg sustava vidljiva je na slici 6.4. To je upravo staza iz



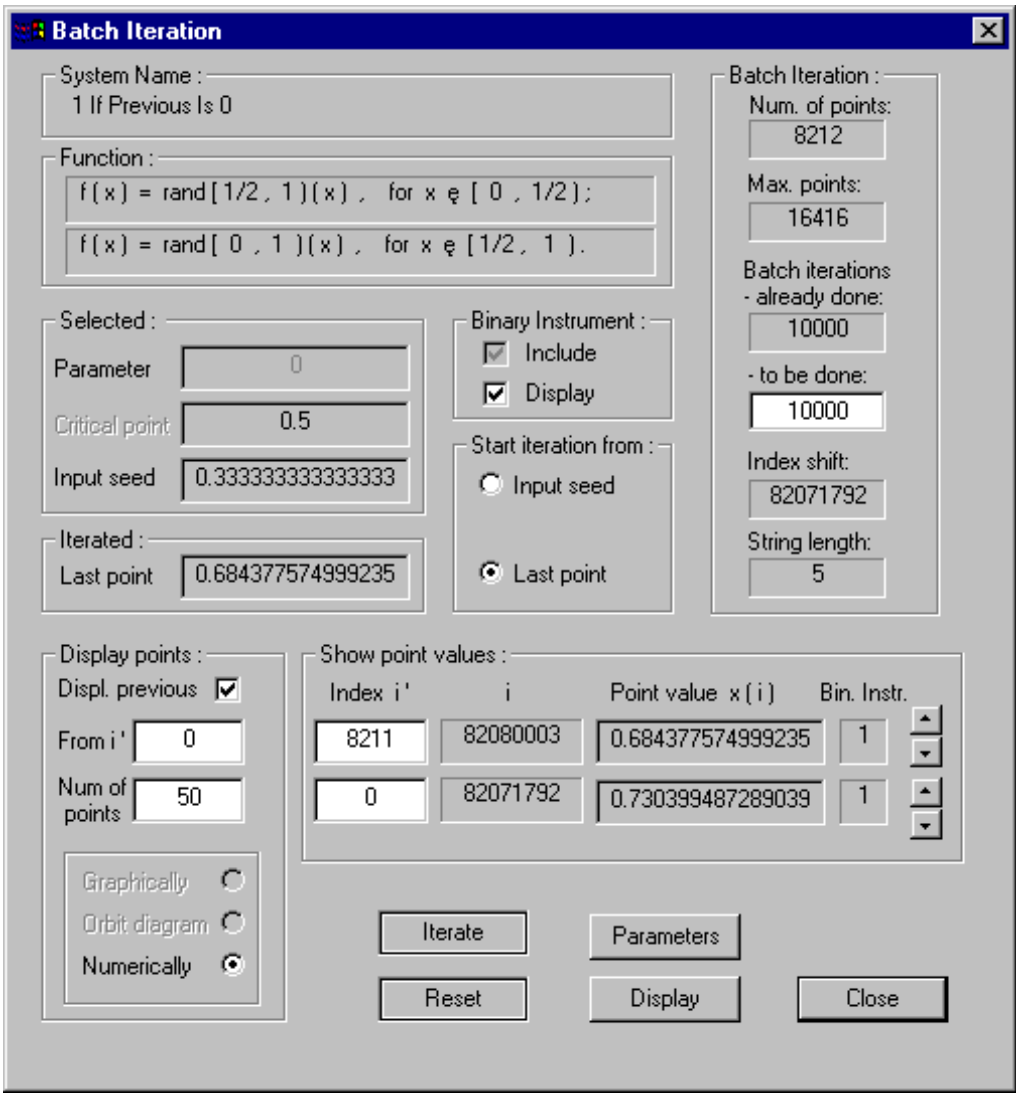
Slika 6.2: Prikaz i izbor parametara dinamičkog sustava. Ulazne vrijednosti se provjeravaju posebice za svaki sustav, prema njegovoj prirodi. Sustavi zadani pravilom, kao što je sustav sa slike, nemaju parametar funkcije.

koje je dobiven vremenski niz (4.39) i slika 4.6.

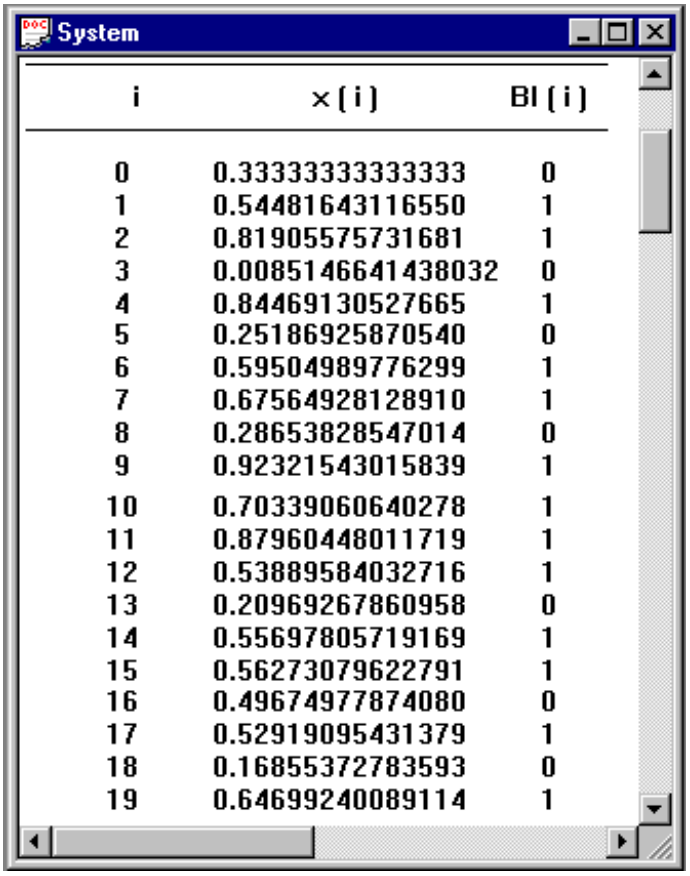
6.1.2 Glavno raščlambeno stablo

Nakon što je definiran dinamički sustav koji će generirati vremenski niz, pripravni smo za gradbu *glavnog raščlambenog stabla* (*Main Parse Tree*). Odabirom stavke *New* u izborniku *Tree,* otvara se dijaloški prozor koji omogućava izbor parametara glavnog stabla. Prozor je prikazan na slici 6.5.

Visina glavnog stabla se može mijenjati u rasponu od 1 do 20. Gornja granica je definirana izborom 32-bitnog nepredznačenog cjelobrojnog tipa za brojače u čvorovima stabla, uz statistički relevantan hranidbeni faktor. *Hranidbeni faktor* (engl. *feed factor*) definira željeni prosječni iznos odbrojaka pojedinog lista stabla po završetku



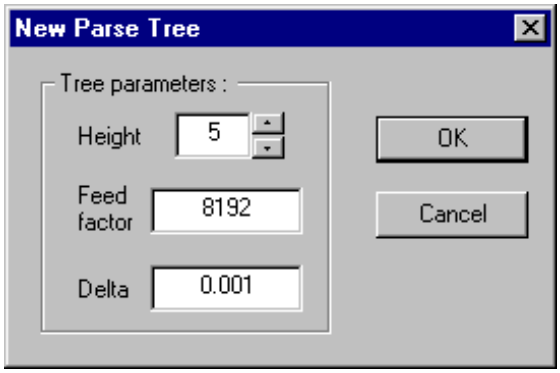
Slika 6.3: Dijaloški prozor za grupnu iteraciju (batch iteration). Vidljivi su odabrani parametri sustava, te broj dosada izvršenih grupnih iteracija. U grupnoj iteraciji izračuna se orbita za sve točke iz kružnog spremnika. Korisnik može odabrati proizvoljan broj željenih grupnih iteracija (do na granicu 32-bitnog nepredznačenog cjelobrojnog tipa), te na licu mjesta provjeriti iznose pojedinih točaka iz zadnje grupne iteracije.



Slika 6.4: Lista točaka orbite dinamičkog sustava "1 ako je prethodni bio 0", što je ekvivalentno opisu "nema slijednih nula". Prikazane su vrijednosti točaka u intervalu $[0, 1)$ kao rezultat simulacije s pomoću generatora slučajnih brojeva, te odgovarajuće vrijednosti nakon grube izmjere na binarnom instrumentu.

hranidbe, uz pretpostavku da stablo ima maksimalni broj listova. Delta određuje (za sada okvirni) prag u relaciji δ-ekvivalencije podstabala (vidjeti izraz 4.21). Vrijednost δ je također uključena u procjenu potrebne duljine niza. Odabirom visine 5 slijedimo primjer stabla sa slike 4.6.

Sada smo spremni da izvršimo hranidbu glavnog stabla. Ukoliko želimo promatrati proces hranidbe i istraživati strukturu stabla, odabiremo stavku *Feed And Show* u izborniku *Tree*. Time se otvara dijaloški prozor ostvaren s pomoću klase `CTreeFeedAndShowDlg` opisane u odjeljku 5.5, i pripadnog mu dijaloškog izvorišta (*Dialog Resource*) identificiranog s `IDD_M_TR_FEED_AND_SHOW`. Namjena je ovog prozora da na jednom mjestu ponudi sve informacije o glavnom stablu. Prozor je zbog svoje veličine prikazan u dva dijela, na slikama 6.6 i 6.7. Tu je dano stanje stabla nakon pohrane 15 podnizova duljine 5, što je ostvareno upisom brojke 15 u polje za broj

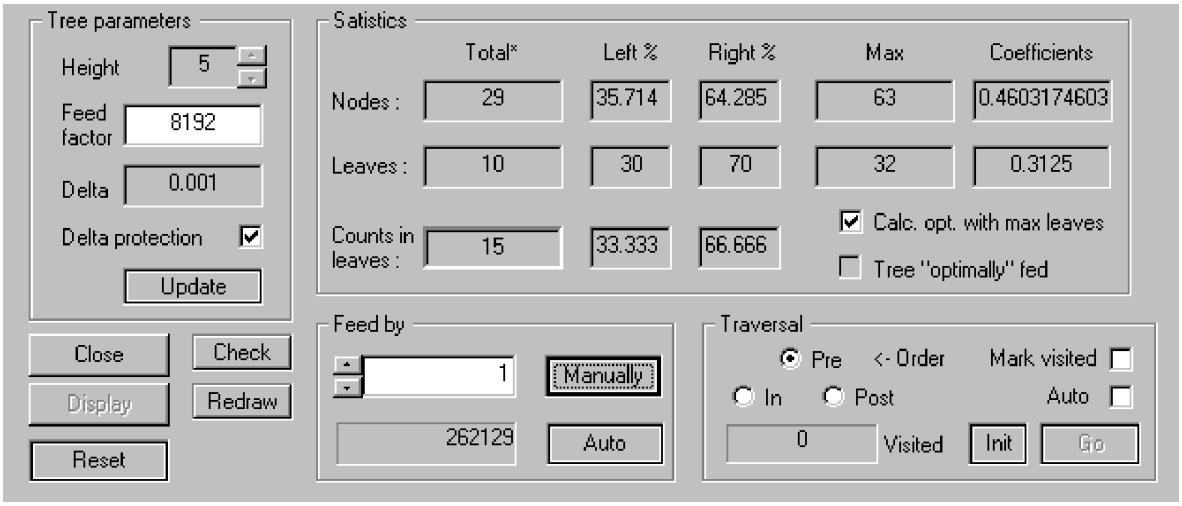


Slika 6.5: Izbor parametara glavnog raščlambenog stabla (Parse Tree). Visina glavnog stabla se može mijenjati u rasponu od 1 do 20. Hranidbeni faktor (Feed Factor) definira željeni prosječni iznos odbrojaka pojedinog lista stabla po završetku hranidbe, uz pretpostavku da stablo ima maksimalni broj listova. Delta određuje okvirnu granicu za relaciju δ-ekvivalencije podstabala.

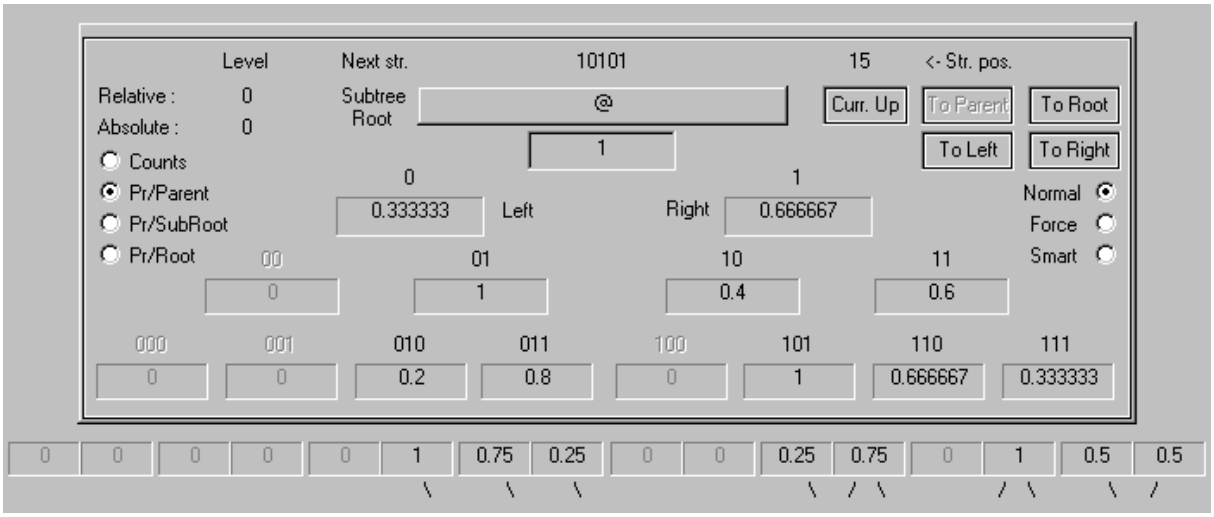
nizova pri "ručnoj" hranidbi, te pritiskom na dugme *Manually*. U gornjem dijelu prozora sadržana je sva statistika glavnog stabla koja se temelji na informacijama sadržanim u objektu klase `CTreeStatistics`. U našem je slučaju nakon pročitanih 15 podnizova duljine 5, stvoreno 29 čvorova od 63 maksimalno mogućih, tj. *koeficijent popunjenosti stabla* iznosi 0.460 . Broj desnih čvorova je nešto manje nego dva puta veći od broja lijevih čvorova, a odstupanje od omjera $2:1$, je naravno, posljedica male statističke težine broja učitanih podnizova. Listova ima ukupno 10, od maksimalno mogućih 32. Ukupna suma odbrojaka u listovima mora biti jednaka broju učitanih podnizova. Funkcionalna skupina označena s *Traversal* logički spada u opis strukture stabla i bit će objašnjena niže.

Donji dio dijaloškog prozora prikazuje samo stablo, tj. stanje njegovih čvorova (sl. 6.7). Pokazno stablo (već diskutirano u 5.5) daje potpuni prikaz za čvorove do visine 4, a struktura nasljeđivanja je vidljiva i za čvorove razine 5 . Tako imamo gotovo cjelovit uvid u glavno stablo kreirano u našem primjeru. Na slici je *korijen pokaznog podstabla* (*Subtree root*) upravo u korijenu glavnog stabla. Niz koji odgovara korijenu stabla je prazan niz $e = ""$, koji je označen znakom @ . Svi ostali čvorovi pokaznog stabla označeni su svojim (pod)nizovima relativno u odnosu na korijen pokaznog podstabla, i na taj su način jedinstveno definirani.

Iznad korijena pokaznog podstabla zapisan je podniz koji će biti učitan u stablo pri idućoj pohrani, a desno od njega je oznaka pozicije tog niza. Primijetimo da nakon čitanja 15 podnizova duljine 5, u nizu $\mathbf{s} = 01101011011110110101\ldots$ duljine 20 , preostaje podniz 10101, kao što je i označeno na slici.



Slika 6.6: Gornji dio prozora glavnog raščlambenog stabla. Tu je prikazana statistika stabla načinjena na osnovi informacija iz objekta klase CTreeStatistics, nakon što je pročitano 15 podnizova duljine 5, iz vremenskog niza $s = 01101011011110110101\ldots$. Uveden je pojam koeficijenta popunjenosti za čvorove i listove posebice (listovi su uključeni u čvorove). Statistika daje broj lijevih i desnih čvorova te lijevih i desnih listova, što pruža globalnu informaciju o stablu.



Slika 6.7: Donji dio prozora glavnog raščlambenog stabla predstavlja strukturu stabla i podatke o čvorovima. Stanje na slici je posljedica hranidbe s 15 podnizova duljine 5, iz vremenskog niza $s = 01101011011110110101\ldots$, počevši od nulte pozicije. Podniz na poziciji 15, koji će biti pročitan pri sljedećoj hranidbi je naznačen iznad korijena pokaznog podstabla. Čvorovi definirani s 01010 , 01110 i 11111 nisu (još) popunjeni, iako pravilo našeg sustava to dozvoljava (usporediti sa slikom 6.9).

Na lijevoj strani stabla možemo radio dugmetom odabrati željeni način prikaza stanja čvorova: bilo preko stanja brojača u njima, ili preko tri tipa uvjetnih vjerojatnosti. Ukoliko želimo doznati što se događa na nekoj od nižih razina stabla možemo se spustiti u neku od njegovih grana. Za to odabiremo povoljnu strategiju spuštanja; npr. *Force* ako želimo da se odabrani čvor na slici povuče maksimalno prema gore. Kroz stablo možemo putovati pritiskom na dugmad koja označava smjer slijednog kretanja (od čvora do čvora), ili izravno pritiskom na neki neprazan čvor.

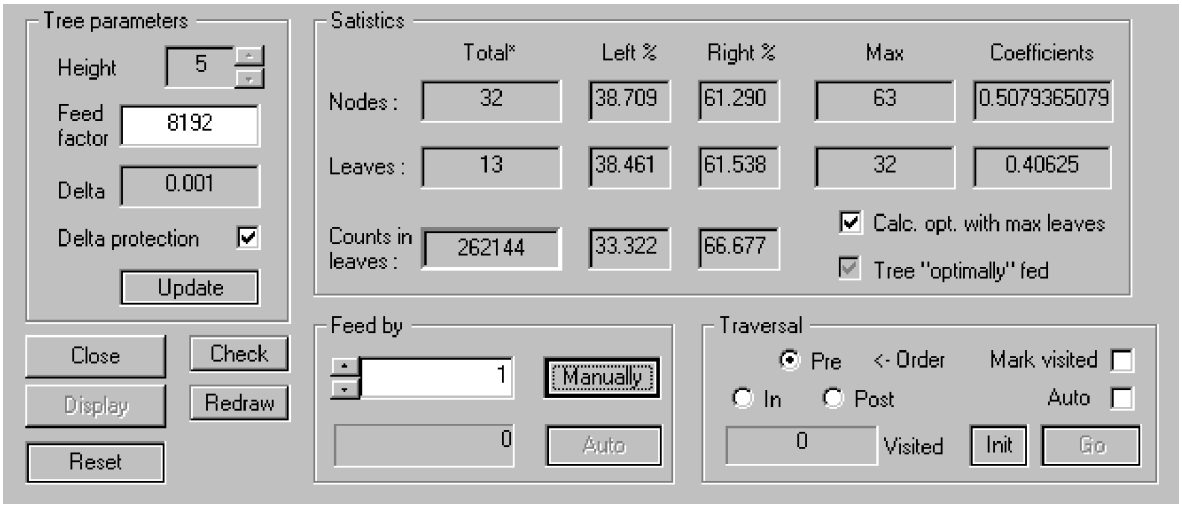
Također, možemo sustavno proći kroz čvorove stabla po prijerednom, međurednom ili poslijerednom prolasku, i to čvor po čvor ili *automatski* kroz cijelo stablo — nakon čega dobivamo informaciju o broju nađenih čvorova. To se ostvaruje unutar kontrolne skupine *Traversal* koja je dio gornje polovice dijaloškog prozora (sl. 6.6).

Unutar glavnog stabla jasno je uočljiva struktura podstabala, iako je učitan mali broj podnizova. Primijetimo da je najdonji desni čvor (na visini 5) prazan jer do njega ne vodi kosa crtica koja bi naznačivala da postoji (desno) dijete. Zaista, u gornjem nizu $\mathbf{s}$ ne postoji podniz 11111. Postoji samo jedan podniz od četiri uzastopne jedinice. Pošto je, prema pravilu našeg sustava, dozvoljena pojava proizvoljnog broja uzastopnih jedinica, to je samo posljedica kratkoće učitanog niza (usporediti sa slikom 6.9). Isto vrijedi i za čvorove 01010 i 01110.

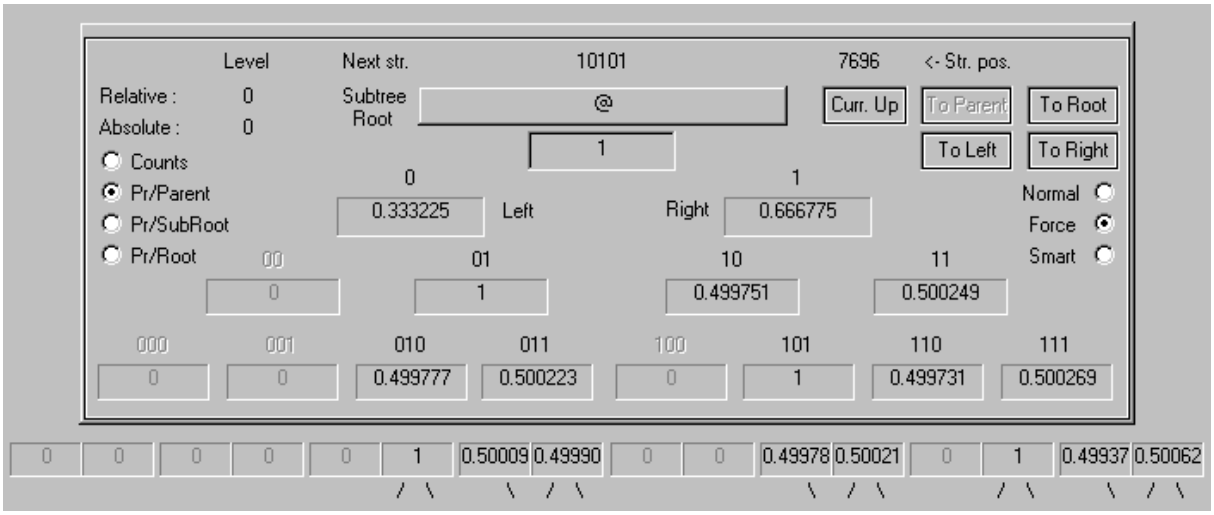
Na sljedećem paru slika 6.8 i 6.9 prikazani su rezultati analize glavnog stabla nakon dodatne, obilne hranidbe s $262\,129$ podnizova. Dovoljno je kliknuti na dugme *Auto* i program će izvršiti učitavanje toliko podnizova da će listovi u prosjeku imati iznos odbrojaka jednak redu veličine faktora hranjenja, odnosno da će biti "nekoliko" puta veći od recipročne vrijednosti parametra δ.

Koeficijent popunjenosti može poslužiti kao ogledalo strukture podstabala unutar glavnog stabla. Zanimljivo je primijetiti da je njegovo kvalitativno ponašanje analogno deterministočkoj složenosti. Ukoliko imamo jednostavnu determinističku strukturu — kao što je npr. periodička orbita, bit će popunjen izrazito mali broj čvorova i taj će koeficijent biti blizak nuli. U protivnom, ako je sustav potpuno stohastičan, ergodička teorija nalaže da će svi čvorovi biti popunjeni, uz podjednaku raspodjelu vjerojatnosti na svakoj razini stabla. To odgovara koeficijentu popunjenosti 1. U oba navedena slučaja je strukturalna složenost sustava mala. Prema zaključcima iz pododjeljka 4.2.3, ona će biti najveća za vrijednosti koeficijenta između navedenih krajnjih vrijednosti. Detaljnija usporedba koeficijenta popunjenosti i determinističke složenosti traži dublju analizu koja ovdje neće biti izlagana.

Nadalje, statistika nam nudi omjer broja odbrojaka u lijevim i desnim listovima



Slika 6.8: Statistika glavnog stabla nakon što je učitano $262\,144$ podnizova duljine 5. Koeficijent popunjenosti čvorova je oko 0.5, što upućuje na postojanje morfološke strukture podstabala. Broj odbrojaka u lijevim i desnim listovima se odnosi kao $0.33322 : 0.66677$, što predstavlja relativno odstupanje od 10^{-4} u odnosu na teorijske vrijednosti.



Slika 6.9: Izgled strukture glavnog stabla nakon hranidbe statistički značajnim brojem od $262\,144$ podnizova. Podudarnost s predviđanjima potpuna je i glede strukture morfova i glede vjerojatnosti njihovih prijelaza (usporediti s izlaganjem 4.5, te slikama 4.6 i 4.8).

$0.33322 : 0.66677$, što predstavlja relativno odstupanje od 10^{-4} u odnosu na teorijske vrijednosti. U stvari, u ovom je slučaju naš model dobar test za primijenjeni generator slučajnih brojeva. Broj odbrojaka u lijevim (desnim) listovima je izravno proporcionalan vjerojatnostima za pojavu nula (jedinica). Za razliku od omjera broja odbrojaka, omjer broja lijevih i desnih čvorova, koji je gotovo jednak omjeru lijevih i desnih listova, ne reproducira odnos $1 : 2$.

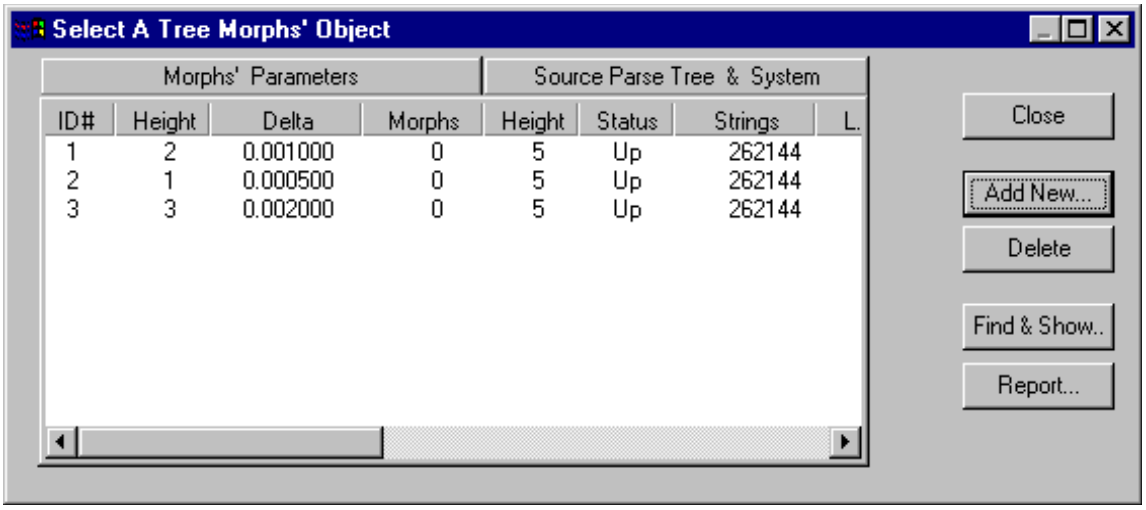
Struktura glavnog stabla uočljiva je na slici 6.9. Sada ne samo da smo se približili teorijskim predviđanjima iz poglavlja 4, već su i vjerojatnosti prijelaza u potpunom suglasju s teorijskim predviđanjima (vidjeti odjeljak 4.5, te sl. 4.6 i 4.8). Uočavanjem podstabala visine 2, počevši od korijena glavnog stabla pa naniže, može se primijetiti struktura koju smo već podrobno opisali. Na slici 6.9 izravno je vidljiva i struktura morfova i vrijednosti njihovih prijelaznih vjerojatnosti. To će biti potvrđeno u posljednjem koraku primjene našeg programa koji slijedi.

6.1.3 Nalaženje jedinstvenih morfova

Hranidba stabla može biti izvršena u upravo opisanom dijaloškom prozoru ili jednostavno odabirom stavke *Feed Auto* unutar izbornika *Tree.* Nakon toga moguće je pristupiti nalaženju morfova.[1] U izborniku *Morphs* odaberimo najprije stavku *Select.* Unutar dijaloškog prozora kreiramo objekte klase `CTrMorphs`. Svaki objekt će predstavljati *skupinu morfova* koji će biti pronađeni u postupku njihovog traženja. Svaka skupina je okarakterizirana parametrima navedenim u zaglavlju izvješća, kao što se može vidjeti na sl. 6.10.

S desne strane se nalaze opći podaci o glavnom stablu u kojem je izvršeno traženje morfova, kao što su njegova visina, stanje, broj učitanih podnizova te statistika stabla. Stanje izvornog stabla se odnosi na to je li ono još "živo", u kojem slučaju je označeno s *Up*, ili je "posječeno" (tj. uništeno zamjenom s novim stablom), što je označeno s *Cut.* Ustrajnost ovih objekata je načelno dulja od vijeka glavnog stabla. Krajnje desno je zapisano ime dinamičkog sustava iz kojeg je izvršena (zadnja) hranidba stabla. S lijeve strane su podaci koji se tiču samih morfova. Identifikacijski broj je jednostavno broj objekta u listi. Zatim slijede visina morfova i parametar δ kao ključne odrednice. Ukoliko za dati objekt traženje još nije izvršeno, to se ogleda u broju nađenih morfova, koji je u tom slučaju nula. Ukoliko je traženje za dati objekt izvršeno, tada je broj nađenih morfova veći ili jednak 1. U našem primjeru,

[1] Kad spominjemo nalaženje morfova, uvijek, u skladu s definicijom 4.3.3, mislimo na *jedinstvene morfove.* Riječ "jedinstvene" ponekad ispuštamo radi jednostavnosti.



Slika 6.10: Dijaloški okvir za izbor objekta skupine morfova (iz klase CTrMorphs), odnosno budućeg SFA-modela. Bitne su odrednice modela visina morfova i parametar δ. S desne strane nalaze se podaci o izvornom stablu svake skupine. Budući da postupak nalaženja morfova još nije pokrenut, njihov je broj nula i SFA-modeli su nedefinirani.

otvorena su tri objekta, različitih visina i različitih izbora parametara δ.

Postupak traženja morfova se pokreće bilo iz ovog dijaloga, pritiskom na dugme *Find & Show*, ili smo to mogli učiniti izravno iz izbornika *Morphs* odabirom stavke *Find Morphs*. To rezultira otvaranjem dijaloškog prozora vrlo sličnog onom za glavno stablo (sl. 6.11 i 6.12). Naime, proces nalaženja morfova je najilustrativnije pratiti upravo promatranjem glavnog stabla. Također, pronađeni morfovi su i sami (pod)stabla, pa je u tom smislu klasa ovog dijaloškog prozora izvedena kao nasljednica klase prozora glavnog stabla. Razlika je u tome što su ovdje u gornjem lijevom uglu podaci o visini morfova, a također i statistika se odnosi na morfove.[2]

Provjerimo, dakle, da li smo zadovoljni s odabranim objektom morfova, odnosno s njegovim parametrima. Ako nismo, pritiskom na dugme *Select* pozivamo dijalog za izbor, u kojem možemo odabrati drugu ili kreirati novu skupinu morfova. Odaberimo skupinu s visinom 2 i $\delta = 0.001$. Primijetimo da je odmah po određenju visine morfova moguće provjeriti koliki je njihov maksimalni broj za dano glavno stablo. Taj broj ne može biti veći nego što je broj čvorova u prolaznom stablu opisanom u 5.4.2 (stablo s visinom $D - L$ koje kreće iz korijena glavnog stabla), i određuje se brojanjem čvorova u prolaznom stablu. Za naše glavno stablo maksimalni broj morfova visine 2 iznosi 11.

[2]Neke predviđene radnje u ovom prozoru još nisu funkcionalne.

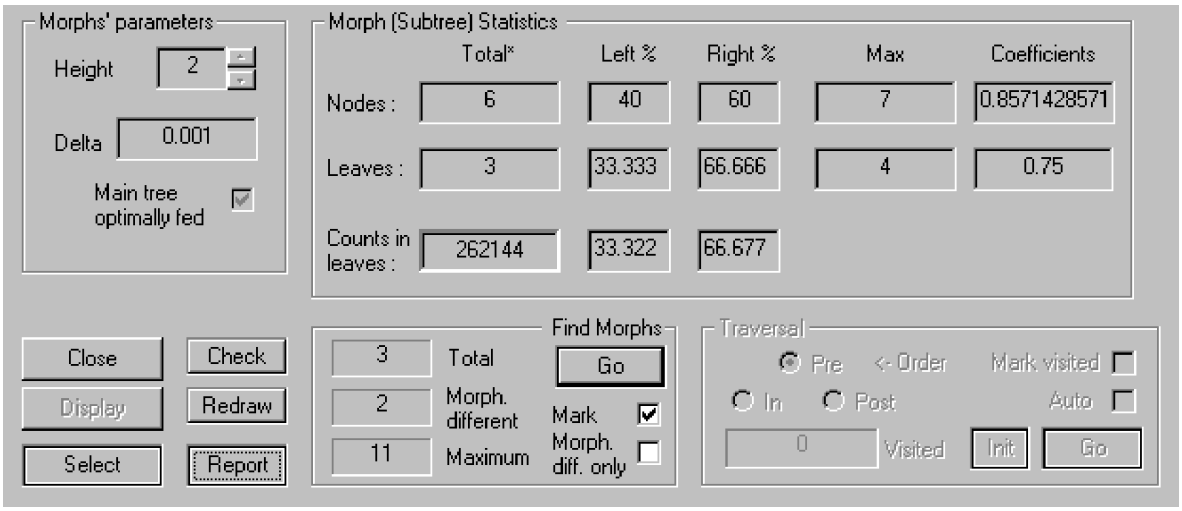
Aktiviranjem dugmeta *Go* pokreće se postupak nalaženja morfova, što je ostvareno funkcijom `FindAllMorphs` opisanom u 5.4.3. Odmah po završetku, mijenja se broj nađenih morfova, prikazan lijevo od dugmeta *Go*. U našem slučaju, pronađena su ukupno 3 morfa, od kojih su dva morfološki različita (sl. 6.11).

Ukoliko je odabrana opcija *Mark* (neposredno ispod dugmeta *Go* vidljivog na slici 6.12), tada će svi korijeni pronađenih morfova biti obilježeni kao ispupčeni. Tako na slici odmah uočavamo tri ispupčena čvora — koji definiraju tri podstabla visine 2, odnosno tri jedinstvena morfa.

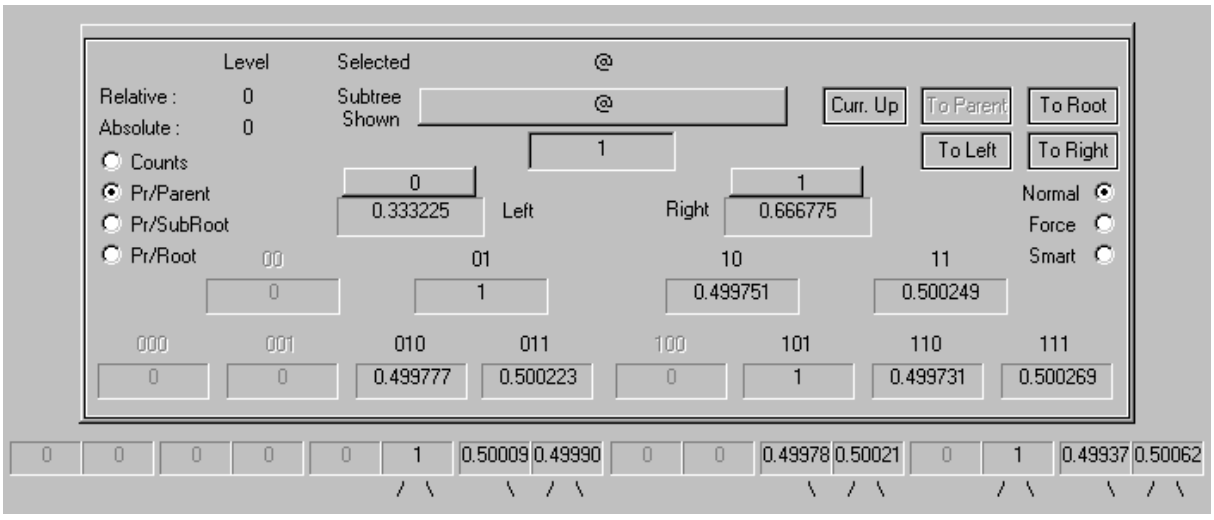
Želimo li dobiti pregledno izvješće o pronađenim morfovima i njihovim prijelazima, to postižemo pritiskom na kontrolno dugme *Report*. Pojavljuje se prozor s nazivom *Stochastic Automaton* (*Morph Transitions*) koji daje popis svih pronađenih stanja i svih prijelaza između stanja (slika 6.13). Primijetimo da je ovim izvješćem potpuno reproduciran ϵ-stroj $(\Xi, \mathcal{T})_\epsilon$ na razini 2, koji odgovara SFA zadanom s $M_{SFA} = (Q, \mathcal{A}, q_0, \mathbf{T})$ (usporediti s točkom 3. algoritma 4.2).

Pored reprodukcije ϵ-stroja, ovo izvješće podastire niz podataka koji nam pomažu u analizi dobivenih rezultata i sugeriraju kako izvršiti daljnja modeliranja. Rezultati izvješća se zasnivaju na funkciji `FindChildMorphs` opisanoj u ispisu 5.9, te na statistici morfa kao podstabla.

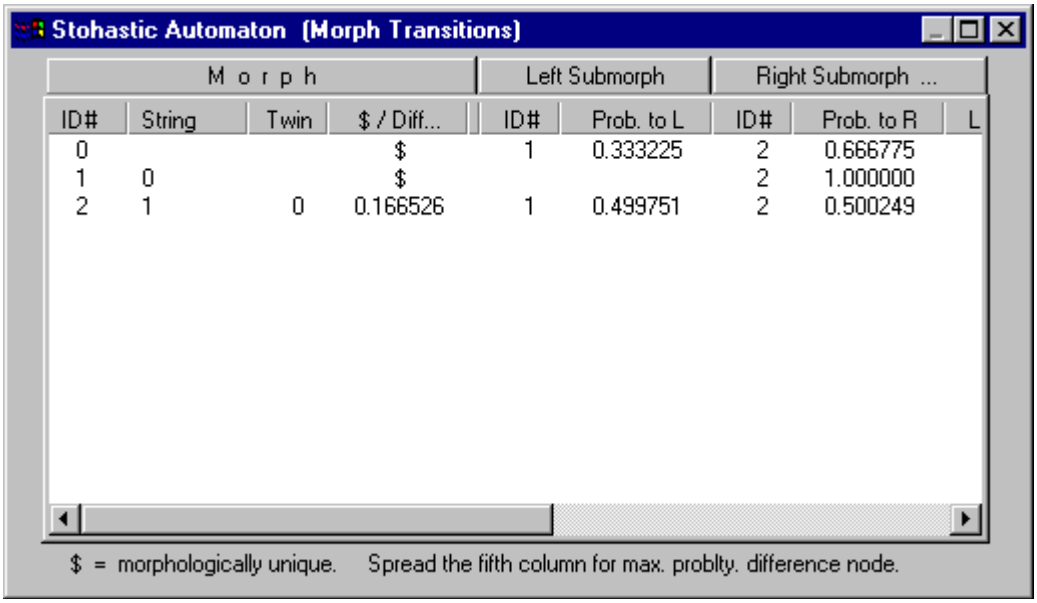
Svaki jedinstveni morf ima pridružen identifikacijski broj. Uz zadanu visinu, morf je jednoznačno definiran sa svojim korijenom kao čvorom u glavnom stablu. Taj je korijen u izvješću prikazan binarnim nizom u drugoj koloni. Iz prijašnjeg izlaganja proizlazi da prvi morf uvijek proistječe iz korijena glavnog stabla, tj. njegov korijen je opisan praznim nizom. Svako morfološki jedinstveno stanje je označeno znakom *$*. Za stanje koje nije morfološki jedinstveno, zapisan je identifikacijski broj onog stanja (podstabla) koje mu je morfološki blizanac i koje je u odnosu na sve ostale eventualne blizance vjerojatnosno najbliže promatranom stanju. U sljedećoj koloni je maksimalno vjerojatnosno odstupanje pronađeno za dva odgovarajuća čvora ovih blizanaca. Ovaj podatak će biti od presudne važnosti u razvijanju strategije modeliranja, kada ćemo promatrati u kojoj mjeri možemo minimizirati model (smanjiti broj njegovih stanja) uz povećanje parametra δ. Proširenjem pete kolone, dobit ćemo i dodatni podatak o čvoru u kojem se to maksimalno odstupanje pojavilo.



Slika 6.11: Gornji dio dijaloga za nalaženje jedinstvenih morfova. Odabrana je morfna skupina s morfovima visine 2 i $\delta = 0.001$. Stanje na slici je nakon pritiska na dugme Go. Vidljiva je statistika korijenskog morfa. Pronađena su 3 morfa od kojih su 2 morfološki različita. Maksimalni broj morfova je 11, što se može provjeriti na slici 6.12 donjeg dijela prozora.



Slika 6.12: Donji dio dijaloga za nalaženje jedinstvenih morfova. Uz odabranu opciju *Mark* svi korijeni pronađenih morfova bit će obilježeni kao ispupčeni. Na slici su uočljiva tri ispupčena čvora — koji definiraju tri podstabla visine 2, kao tri jedinstvena morfa.



Slika 6.13: Izvješće o skupini morfova kao rezultat modeliranja SFA. Svaki jedinstveni morf je označen svojim identifikacijskim brojem i nizom koji odgovara njegovom korijenu u odnosu na korijen glavnog stabla. Ukoliko je morf oblikovno (morfološki) jednak nekom prethodnom morfu, indeks tog ranijeg morfa je u koloni Twin. SFA slijedi iz strukture i vjerojatnosti prijelaza (na desno). Lijevi podmorf događa se po simbolu 0, a desni po 1.

Daljnje kolone daju podatke o prijelazima i njihovim vjerojatnostima, tj. u potpunosti reproduciraju tenzor $\mathbf{T}$. Pomakom vidljivog prozora izvješća u desno, otvaramo dodatne podatke koji se odnose na statistiku svakog morfa kao podstabla. Ustrajnost morfološke skupine kao objekta je učinjena neovisnom od postojanja glavnog stabla, kao što je već nagoviješteno u 4.5.5. Na ovaj smo način o svakom jedinstvenom morfu sačuvali sve one informacije koje možemo, bez da pohranjujemo njegovu strukturu stabla. Tako smo sakupili niz relevantnih podataka, ne samo o reproduciranom SFA nego i o njegovim stanjima kao podstablima.

6.1.4 Statistički pokazatelji

Ponovimo li gornji postupak za preostale dvije morfološke skupine, tj. za onu s visinom 1 i $\delta = 0.0005$, te za treću s visinom 3 i $\delta = 0.002$, dobivamo jedan te isti stohastički automat. Drugim riječima, za ovaj je sustav dovoljno promatrati korelacije duljine 1. Ovaj automat, do na malo odstupanje u vjerojatnostima, u potpunosti reproducira analitički model razvijen u odjeljku 4.5 (sl. 4.8).

Potpunosti radi navedimo da je matrica prijelaza dobivena na osnovi modeliranja

sljedeća:

$$T = \begin{pmatrix} 0 & 0.333225 & 0.666775 \\ 0 & 0 & 1 \\ 0 & 0.499751 & 0.500249 \end{pmatrix} . \tag{6.1}$$

Njen lijevi svojstveni vektor jest

$$\mathbf{p}_{st} = (0\,,\, 0.333219\,,\, 0.666781)\,, \tag{6.2}$$

a statistička složenost procesa

$$C_\mu\,(\Xi, \mathcal{T}\,) = H\,(\mathbf{p}_{st}) = 0.918181\,\mathrm{bit}\,. \tag{6.3}$$

Dobili smo potpuno suglasje s rezultatima predviđenim u 6.1.3.

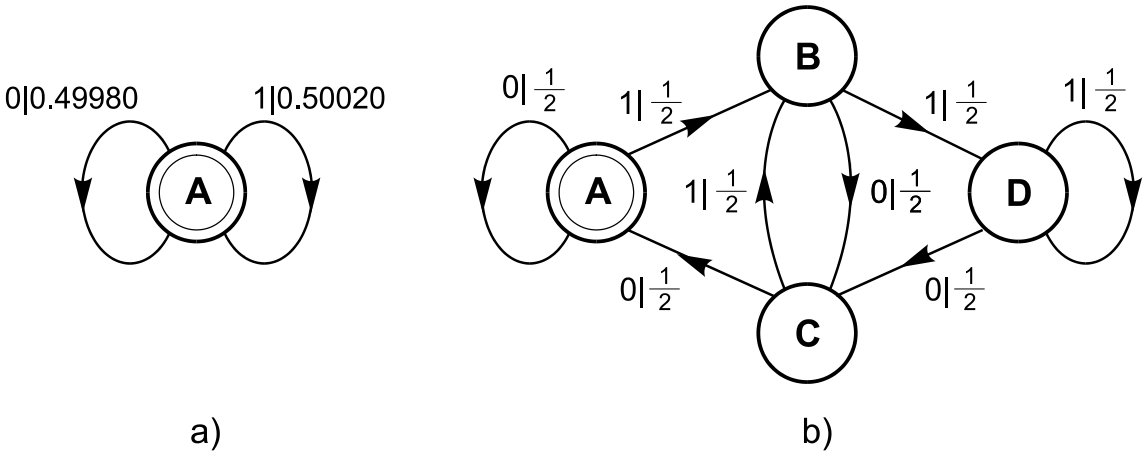
Na koncu ovog odjeljka, istaknimo da funkcija posmaka (2.2) na vremenskom nizu dobivenom iz ovog sustava predstavlja tzv. *potposmak konačnog tipa.* Potposmak je zato jer funkcija operira na podskupu od skupa Σ_∞ svih mogućih vremenskih nizova. Konačnost se jednostavno odnosi na činjenicu da je ovaj sustav moguće opisati konačnim pravilnim izrazom. Naravno, s pomoću SFA ne bismo mogli opisati potposmake beskonačnog tipa. Zanimljivo je primijetiti da funkcija potposmaka na vremenskom nizu opisanom pravilom "nema uzastopnih nula", predstavlja kaotični sustav koji je konjugiran kvadratnoj porodici nelinearnih sustava[2].

6.2 Neki ilustrativni primjeri

Nakon što smo se upoznali s korisničkim aspektom programa *Dynamical Systems Automata*, možemo se posvetiti modeliranju nekolicine ilustrativnih primjera iz skupine sustava koju smo u prethodnom odjeljku nazvali "dinamički sustavi zadani pravilom". Kao što je već istaknuto, njihova relativna jednostavnost omogućuje korisniku uspoređivanje dobivenog modela s definicijskim pravilom, slično kao što smo to napravili na primjeru sustava "generiraj 1 ako je prethodilo 0". Otuda proizlazi njihova uloga u demonstraciji rada programa. Također, ovi sustavi su predstavljali temelj za intenzivno testiranje svih dijelova programa.

6.2.1 Bernoullijev niz

Bernoullijev niz nula i jedinica predstavlja niz generiran iz izvora u kojem je vjerojatnost emitiranja oba simbola jednaka. To je primjer potpuno stohastičkog sustava. Možemo ga opisati jednostavnim pravilnim izrazom $(0+1)^*$. Uz odabir visine 10 za glavno stablo, te 5 za morfove, izvršena je hranidba s 8 388 608 nizova. Dobivena je, prema očekivanju, trivijalna struktura koja odgovara jednom jedinom stanju (sl. 6.14a). Upravo kao što to predviđa ergodička teorija, uz dovoljno dugački vremenski niz svi će podnizovi određene duljine (u našem slučaju duljine 10) biti jednako vjerojatni.[3]



Slika 6.14: SFA za Bernoullijev niz. Oba modela korektno opisuju sustav koji emitira Bernoullijev niz jedinica i nula, opisan pravilnim izrazom $(0+1)^*$. Na slici a) je minimalan model, reproduciran našim programom uz $D = 10$, $L = 5$, $\delta = 2.5 \times 10^{-4}$. Primijetimo da je odstupanje od idealnih vrijednosti tek u četvrtoj decimali. Vrijednost δ mora biti veća od tog odstupanja, u protivnom će za podmorfove biti ustanovljena δ-nejednakost. Na slici b) prikazan je model koji također točno reproducira Bernoullijev niz, ali koji nije minimalan. Ovakav model precjenjuje kompleksnost sustava. To se događa npr. kad parametar δ previše smanjimo, odnosno kad "pretjeramo" u točnosti modela s obzirom na raspoložive podatke.

Bitno je uočiti da smo dobili upravo minimalan model, za koji su topološka i statistička kompleksnost nula:

$$C_0 = C_\mu\left(\Xi, \mathcal{T}\right) = H\left(\mathbf{1}\right) = 0\,. \tag{6.4}$$

[3] Ova činjenica, tako dobro poznata u teoriji informacija, za Bernoullijev je niz posljedica elementarnog vjerojatnosnog računa. Međutim, vrlo je ilustrativno vidjeti da jedan praktični model generira "nasumični niz" kao npr. 0110111010, s istom vjerojatnosti kao i strogo pravilni niz 1111111111. To se izravno uočava u našim dijaloškim prozorima za prikaz strukture stabla.

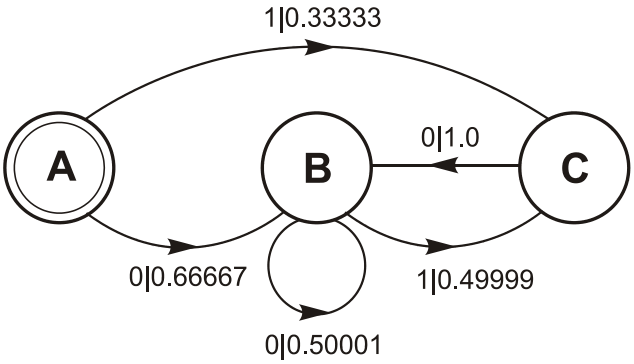
S druge strane, isti se proces može opisati i neminimalnim modelom, kao što je to onaj prikazan na sl. 6.14b. To se zaista i događa ako pri modeliranju previše smanjimo parametar δ. Npr. naš algoritam će davati minimalni model i uz $\delta = 2.5 \times 10^{-4}$, međutim uz $\delta = 2.0 \times 10^{-4}$ model "eksplodira" pronalazeći maksimalnih 63 morfa. Naravno, svi su oni morfološki jednaki, ali δ-različiti. Za podstabla neposredno ispod korijena, vjerojatnosne razlike su jedva iznad zadanog praga (npr. 2.13×10^{-4}), a kako se algoritam spušta i provjerava podmorfove sve nižih i nižih podstabala, vjerojatnosne razlike rastu (npr. čak do 14.67×10^{-4}). To je prirodna posljedica manje statističke težine prikupljenih odbrojaka u nižim čvorovima. Također, ne smijemo zaboraviti da pri ovoj simulaciji nismo imali pravi, već samo kvazislučajan izvor. Provedena diskusija otvara ideje za smjer u kojem bi se trebala poboljšati inteligencija našeg algoritma. To međutim, zahtijeva dublju analizu i bit će ostavljeno za buduće radove.

6.2.2 Nema uzastopnih jedinica

Sustav "nema uzastopnih jedinica" je, naravno, "zrcalno" simetričan dobro proučenom sustavu "nema uzastopnih nula". Dajemo ga ovdje radi potpunosti. Pri modeliranju smo uzeli više glavno stablo (visine 10) nego u odjeljku 6.1, i dobili rezultate koji se s teorijskim predviđanjima slažu do na petu dekadsku znamenku. Automat je prikazan na slici 6.15 i svojom je funkcionalnošću potpuno analogan automatu sa slike 4.8. Struktura i vjerojatnosti prijelaza su međusobno simetrične. Uz zamjenu simbola 0 i 1 te stanja B i C, jedan automat prelazi u drugi.

Primijetimo nadalje, da se toj strukturi može pridijeliti semantička težina [1, 9]. Tako će značenje stanja C u ovom sustavu biti: *sljedeće stanje je sa sigurnošću* B, tj. *može se desiti samo simbol* 0. U sustavu "nema uzastopnih nula" analogno značenje ima stanje B, i ono glasi: *sljedeće stanje je sa sigurnošću* C, tj. *može se desiti samo simbol* 1. Jednako bismo mogli obrazložiti semantiku preostala dva stanja. Semantika, dakle, proizlazi iz *međusobnog odnosa* pojedinih stanja sustava.

Ponovimo činjenicu da je početno stanje A tranzijentno — nakon što sustav izađe iz tog stanja, više se nikad ne vraća u njega. Statistički pokazatelji identični su onima iz pododjeljka 6.1.4.



Slika 6.15: SFA za proces "nema uzastopnih jedinica". Model je rezultat primjene DSA programa. Dobiven je izravnim iščitavanjem izvješća analognog onom prikazanom na sl. 6.13. Točnost je povećana uzimanjem višeg stabla ($D = 10$, $L = 5$), koje je zahtijevalo odgovarajuće povećanu hranidbu. Odstupanja od teorijskih rezultata tek su u petoj znamenci. Ona su odraz statističkih anomalija korištenog generatora slučajnih brojeva.

6.2.3 Paran broj uzastopnih jedinica

Ovaj je sustav programski implementiran na dva rezultatski ekvivalentna načina, što je opisano nazivima "nula ili par jedinica", te "paran broj uzastopnih jedinica" (vidljivo na slici 6.1). Pravilan izraz koji opisuje ovaj sustav je $(0+11)^*$. Parametri modela su odabrani da budu jednaki kao u prethodnom pododjeljku ($D = 10, L = 5,$ $\delta = 0.001$). Dobiveni stohastički automat je prikazan na slici 6.16. Vjerojatnosti na slici su radi jednostavnosti zamijenjene razlomačkim prikazom. Odstupanja od teorijskih predviđanja su, jednako kao i u prethodnom poglavlju, tek u petoj decimali i ne prelaze 1×10^{-5}, što je uočljivo iz matrice prijelaza:

$$T = \begin{pmatrix} 0 & 0.66666 & 0.33334 & 0 \\ 0.74998 & 0 & 0.25002 & 0 \\ 0 & 0 & 0.49999 & 0.50001 \\ 0 & 0 & 1 & 0 \end{pmatrix} . \tag{6.5}$$

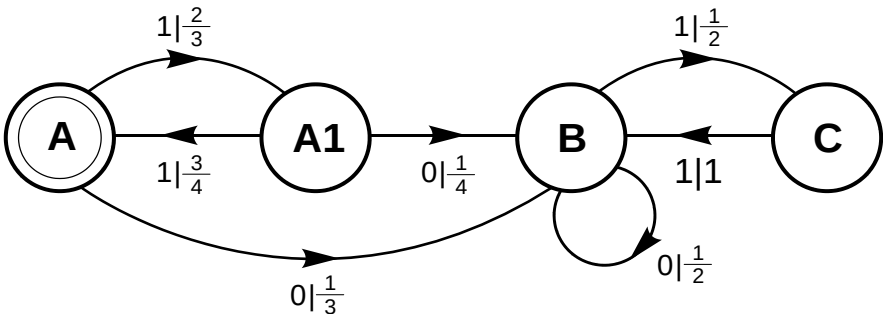
Njen lijevi svojstveni vektor jest

$$\mathbf{p}_{st} = (0\,,\, 0\,,\, 0.66667\,,\, 0.33333)\,, \tag{6.6}$$

a statistička složenost procesa

$$C_\mu\,(\Xi, \mathcal{T}\,) = H\,(\mathbf{p}_{st}) = 0.91830\,\mathrm{bit}\,. \tag{6.7}$$

Topološka složenost je točno $2\,\mathrm{bit}\,,$ a statistička je ekvivalentna kao za sustav "nema uzastopnih nula (jedinica)". To proizlazi iz činjenice da su od četiri postojeća,



Slika 6.16: SFA za proces "paran broj uzastopnih jedinica". Na slici su označene idealizirane vjerojatnosti prijelaza iz rezultata modeliranja sažetih u matrici T danoj izrazom (6.5). Tako dugo dok model ne uhvati fazu, tj. dok ne primi nulu, on oscilira između dva tranzijentna stanja A i A1. Nakon primitka nule, sustav se zadržava u stacionarnim stanjima B i C.

čak dva stanja, A i $A1$, tranzijentna. Otuda zaključujemo da se u stacionarnom slučaju (nakon izlaska iz tranzijentnih stanja), struktura ova dva procesa podudara. Primijetimo da je za sustav "paran broj uzastopnih jedinica" tip tranzijentnosti drugačiji. On ne izlazi iz tranzijentnog stanja nakon prvog koraka, kao što je to bio slučaj za ranije opisane sustave. Tako dugo dok se emitiraju same jedinice, ovaj sustav oscilira između stanja A i $A1$. Semantički ovo odgovara situaciji u kojoj je nemoguće odrediti fazu procesa. Uz pretpostavku našeg modeliranja da ne postoje nikakve dodatne informacije o stanju sustava osim onog što se dade zaključiti iz primljenog vremenskog niza, tako dugo dok se ne primi simbol 0, ne možemo biti sigurni je li prethodno emitirani broj jedinica paran ili nije.

Gornje se vrijednosti mogu u potpunosti vjerojatnosno reproducirati na temelju pravila sustava i činjenice da je broj primljenih jedinica u prosjeku dva puta veći od broja primljenih nula.

U tablici 6.1 prikazani su statistički pokazatelji za nekoliko dinamičkih sustava zadanih pravilima. Prva četiri smo detaljno opisali u prethodnom dijelu ovog poglavlja. Zanimljivo je primijetiti da je sustav "neparan broj uzastopnih jedinica" znatno kompleksniji od sustava "paran broj uzastopnih jedinica". Ima čak tri tranzijentna i tri stacionarna stanja.[4] Jednaki broj stacionarnih stanja ima i sustav "paran broj uzastopnih jedinica i (paran broj) uzastopnih nula", uz postojanje čak pet tranzijentnih stanja.

[4]Ova činjenica se ogleda i u povećanoj kompleksnosti funkcije kojom je ostvaren algoritam iteracije za ovaj sustav.

Dinamički sustav	Broj Stanja			Složenost	
	Ukupno	Morf. razl.	Stac.	C_0 / bit	C_μ / bit
Bernoullijev niz	1	1	1	0	0
Nema uzastopnih 0	3	2	2	1.58496	0.91818
Nema uzastopnih 1	3	2	2	1.58496	0.91818
Paran broj uzast. 1	4	3	2	2.0	0.91830
Neparan broj uzast. 1	6	4	3	2.58496	1.52209
Paran broj uzast. 0 i 1	8	6	3	2.0	1.50000

Tablica 6.1: Pregled statističkih pokazatelja za dinamičke sustave zadane pravilom. Zajednička je značajka ovih sustava postojanje tranzijentnih stanja. Tako posljednja dva sustava posjeduju 3, odnosno 5 tranzijentnih stanja. Posljednje dvije kolone označavaju topološku i statističku složenost.

6.3 Prema nelinearnim sustavima

U ovom odjeljku ukratko ćemo iznijeti rezultate dobivene u preliminarnom istraživanju nelinearnih sustava. Promotrit ćemo dobro znanu logističku funkciju 2.6, i obrazložiti stohastičke konačne automate dobivene za nekoliko karakterističnih vrijednosti parametra r.

6.3.1 Period dva, period tri ...

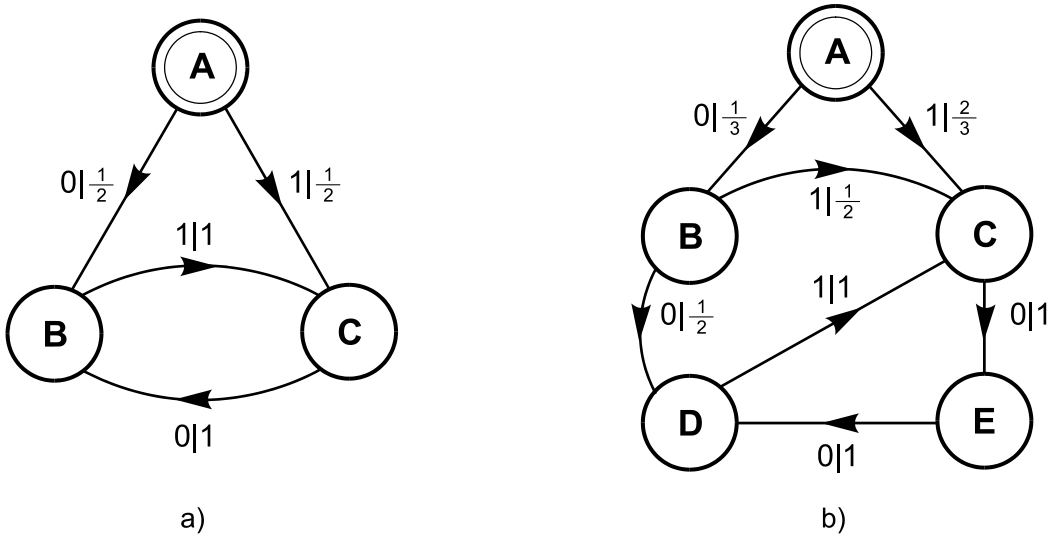
Odabirom parametra $r = 3.25$ za logističku funkciju $f_r(x) = rx(1-x)$ ulazimo u područje *perioda dva* orbitalnog dijagrama (vidjeti npr. [2, 4]). U našem je programu ilustrativno promotriti brzinu konvergencije orbite od sjemene točke $x = 0.5$ do točke perioda 2. Već nakon 22 iteracije točke su stabilne do na točnost dvostruke preciznosti (14 decimala), i orbita oscilira između dvije svoje vrijednosti:

$$0.49526516824547 \;\; i \;\; 0.81242713944683\,,$$

koje će binarni instrument kodirati kao niz nula i jedinica. Uz ispuštanje nekoliko početnih točaka, u glavnom stablu pojavit će se samo dvije staze:

$$\ldots 010101\ldots \;\; i \;\; \ldots 101010\ldots\,.$$

Pronađena su tri jedinstvena morfa, i odgovarajući SFA je prikazan na slici 6.17a. Po primitku prvog simbola sustav izlazi iz tranzijentnog stanja A i počinje deterministički oscilirati između stanja B i C. Lako je uočiti da će stacionarna distribucija vjerojatnosti biti $\mathbf{p}_{st} = \left(0\,,\, \frac{1}{2},\, \frac{1}{2}\right)$, te da je odgovarajuća statistička složenost $1\,\text{bit}$.



Slika 6.17: Logistički sustav u orbitama perioda 2 i 3. Sl. a) prikazuje slučaj perioda 2 ($r = 3.25$), a sl. b) perioda 3 ($r = 3.83$). U oba slučaja je hranidba stabla započela nakon ispuštanja početnih iteracija, dok sustav ne pređe u konačnu periodičku orbitu. Broj stacionarnih stanja je jednak periodu sustava. Svi prijelazi između stacionarnih stanja su deterministički.

Za parametar $r = 3.83$ ulazimo u područje prozora perioda 3. Dobiveni SFA je prikazan na sl. 6.17b. Uočljivo je da su stanja A i B tranzijentna stanja. Jedino ona imaju nedeterminističku dinamiku prijelaza. Kroz njih prolazi sustav u početku, dok ne uhvati fazu. Vidljivo je da će se to desiti najbrže ako je primljen simbol 1. Tada sustav odmah prelazi u stacionarno stanje C i nastavlja deterministički oscilirati s periodom 3 između stanja C, E i D, što odgovara prijamu niza

$$\ldots 100100100 \ldots .$$

Analogno slijedi diskusija u slučaju da je sustav nakon stanja A prešao u prolazno međustanje B, nakon čega može uslijediti bilo stanje D ili C, što odgovara prijamu niza

$$\ldots 001001001 \ldots , \text{ odnosno } \ldots 010010010 \ldots .$$

Nadalje, za $r = 3.5$ sustav prelazi u orbitu perioda 4, za koju će naš model reproducirati SFA s tri tranzijentna i četiri stacionarna stanja. Općenito ćemo za orbitu perioda p dobiti model s $(p-1)$ tranzijentnih i p stacionarnih stanja.

Primijetimo da pronalaskom perioda tri možemo na osnovi teorema Sarkovskog ustvrditi da za taj sustav (r) postoje periodičke točke i svih ostalih perioda, odnosno

Vjer. param. δ	Broj Stanja			Složenost	
	Ukupno	Morf. razl.	Tranz.	C_0 / bit	C_μ / bit
0.025	21	4		4.39232	
0.050	9	4	5	3.16993	1.95623
0.100	6	4	0	2.58496	2.52618
0.135	4	4	0	2.0	1.76901

Tablica 6.2: Modeliranje logističkog sustava za Misiurewiczev parametar. Broj morfološki različitih stanja iznosi 4 i neovisan je o vjerojatnosnom parametru. To odražava statističku invarijantnost sustava za ovu vrijednost parametra r.

da takav sustav ispunjava preduvjet za kaotično ponašanje. No sve te točke ne moraju biti i stabilne, pa ih — kao u ovom slučaju — ne moramo uočiti.

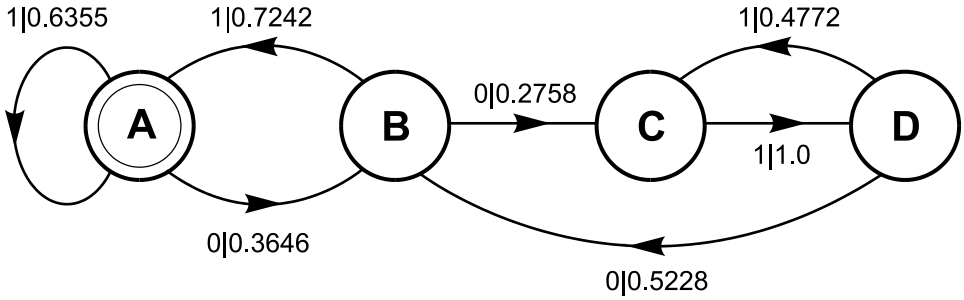
6.3.2 Izranjanje strukturalne kompleksnosti

Nakon primjera jednostavnog stohastičkog i pravilnog deterministčkog ponašanja u prethodna dva pododjeljka, pogledajmo ponašanje sustava za *Misiurewiczev parametar* $r_M \approx 3.9277370017867516\ldots$. Taj je parametar korijen jednadžbe $f^4_{r_M}(x_c) = f^5_{r_M}(x_c)$, gdje je x_c naša uobičajena oznaka za kritičnu točku ($x_c = 0.5$). Pokazuje se da uz vrijednost parametra r_M logistička jednadžba ima posebno dobra statistička svojstva[1].

Izvršeno je modeliranje SFA uz $D = 12$, $L = 6$. Rezultati za različite vrijednosti vjerojatnosnog parametra δ prikazani su u tablici 6.2. Uočljivo je da je broj morfološki različitih stanja konstantan i neovisan o δ. Drugim riječima, bez obzira koliko duboko posezali u glavno stablo pri traženju morfova, ne nailazimo na novu strukturu podstabala. Najgrublji model, uz $\delta = 0.135$, prikazan je na slici 6.18, i on je ekvivalentan rezultatima koje navodi Crutchfield[1]. Pripadna matrica prijelaznih vjerojatnosti jest

$$T = \begin{pmatrix} 0.6355 & 0.3646 & 0 & 0 \\ 0.7242 & 0 & 0.2758 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0.5228 & 0.4772 & 0 \end{pmatrix} . \tag{6.8}$$

Gornji rezultati su ponovljeni za različite duljine podnizova, tj. uz različitu visine stabla $D = 6, 8, 10, 14, 16$. Model sustava pokazuje stabilnost, pa u smislu koraka 3 i 4 iz algoritma 4.4 zaključujemo da je u ovom slučaju dobiveni SFA relevantan model promatranog nelinearnog dinamičkog sustava.



Slika 6.18: SFA logističkog sustava uz Misiurewiczev parametar, $r_M \approx 3.927737\ldots$. Parametri modela su $D = 12$, $L = 6$, i $\delta = 0.135$. Primijetimo da je parametar δ otpušten na relativno grubu vrijednost. Sva stanja morfološki su jedinstvena i stacionarna. Pripadna matrica prijelaza dana je u (6.8).

Kao sljedeći interesantan slučaj, promotrimo što se dešava u području neposredno iza perioda 4, tj. uz vrijednost parametra $r_c \approx 3.5699456718695445$. Radi se o vrijednosti za koju počinje *put prema kaosu udvostručavanjem perioda* (vidjeti npr. [4]). Za ovaj je parametar provedena analiza konvergencije modela stohastičkih konačnih automata u skladu s algoritmom 4.1 [10]. Promatra se ovisnost veličine modela izražena brojem njegovih stanja $||\,\Xi\,||$, u ovisnosti o duljini podnizova D. Rezultati su sažeti u tablici 6.3. Kolona naslovljena s $\max$. označava koliki je maksimalni broj različitih morfova za dano prolazno podstablo.

Za razliku od slučaja Misiurewiczevog parametra, za ovaj sustav SFA-model divergira. Broj pronađenih stanja raste proporcionalno s dubinom D, štoviše, pokazuje trend porasta brži od rasta D. Za prve tri vrijednosti od D broj morfološki različitih stanja je za jedan manji od ukupnog broja stanja, a za sve ostale vrijednosti su sva iznađena stanja ujedno i morfološki jedinstvena. Dakle, broj nađenih stanja uopće ne ovisi o odabranom vjerojatnosnom parametru δ, već se zaista radi o bogatstvu strukture koju ne možemo opisati s pomoću razreda stohastičkih konačnih automata. Ukoliko bismo promatrali ovisnost statističke složenosti C_μ o entropiji $H_\mu(D)\,/\,D$ po jednom simbolu podniza, tada bismo za ovu vrijednost parametra r dobili maksimalnu vrijednost koja odgovara vrhu krivulje na slici 4.2b (vidjeti također [10]).

Rješenje je ovdje uvođenje inovacijskog koraka, te uzimanje višeg razreda automata kao osnovice modela. Crutchfield je pokazao[10, 22] da će za ovaj slučaj konačni model biti postignut uz uporabu strojeva za proizvodnju nizova. Naša programska podrška predstavlja osnovicu iz koje je moguće izvršiti iskorak prema toj višoj razini modela.

D	$\|\|\,\Xi\,\|\|$	max.	D	$\|\|\,\Xi\,\|\|$	max.
3	3	6	12	15	35
4	5	6	13	15	46
5	7	11	14	15	46
6	7	11	15	15	58
7	7	17	16	23	58
8	11	17	17	25	72
9	13	25	18	25	72
10	13	25	19	27	88
11	15	35	20	27	88

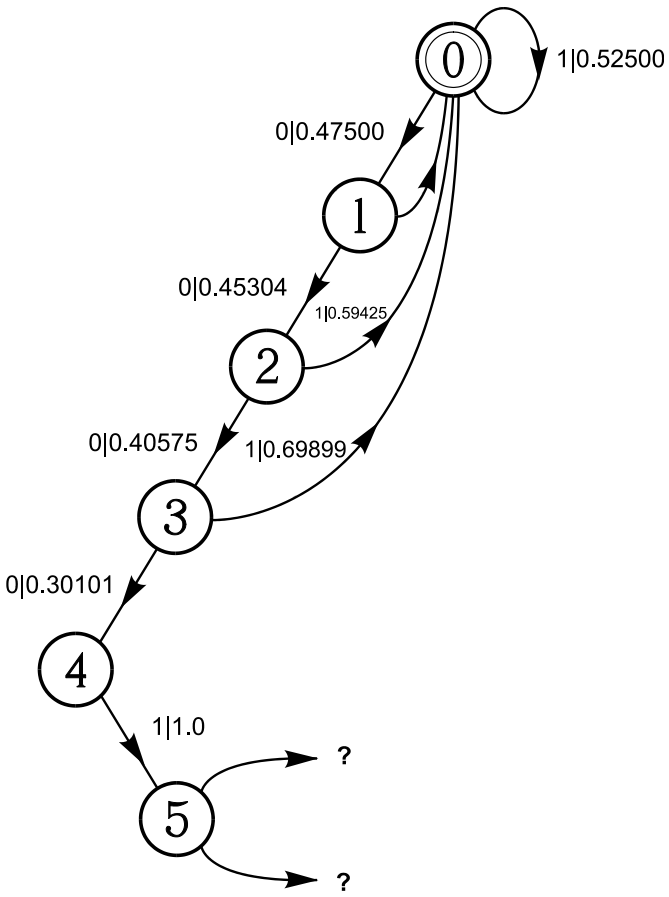
Tablica 6.3: Ovisnost broja nađenih stanja $||\,\Xi\,||$ o duljini (dubini) D podnizova (glavnog stabla) za logistički sustav uz $r_c \approx 3.5699456718695445$. U koloni označenoj s max. dan je maksimalni broj podstabala izražen brojem čvorova prolaznog stabla. Broj stanja raste proporcionalno (čak i nešto brže) od D, tj. divergira. Za $D = 3$ i $D = 4$ broj morfološki jedinstvenih stanja je za jedan manji od ukupnog broja stanja. Za sve ostale vrijednosti D sva pronađena stanja su i morfološki jedinstvena! Bogatstvo struktura je veliko i stalno narastajuće. Model SFA je nedostatan za ovaj kritični parametar, te iziskuje uvođenje inovacijskog koraka.

6.3.3 Od strukture do kaosa

Logistička jednadžba pokazuje kaotično ponašanje za vrijednosti parametra $r \lesssim 4$. Pogled na strukturu tog područja dan je za logistički sustav s parametrom $r = 3.997$. Primijetimo da smo nedaleko od područja Misiurewiczevog parametra. Međutim, ovdje imamo ponašanje koje nije statistički invarijantno. Dobiveni model je prikazan na sl. 6.19.

Sustav je na putu prema kaosu. No nakon primitka više uzastopnih nula, udaljujemo se od stohastičkog ponašanja i nailazimo na naznake zaostale strukture. Tako je prijelaz iz stanja 4 u stanje 5 na sl. 6.19 deterministički, i nakon toga izranjaju stanja koja odgovaraju bogatijoj strukturi. Ukoliko model prima uglavnom jedinice, s malim brojem nula, on je blizak stohastički pogonjenom sustavu. Uz $D = 7$, $L = 3$, $\delta = 0.035$ modeliranje otkriva svega jedno stanje koje je nalik *zakošenom* (engl. *biased*) Bernoullijevom nizu s vjerojatnostima prijelaza od 0.475 (0.525) za pojavu nule (jedinice).

Uz odabir parametra r još bliže broju 4, dolazimo u stanja sustava koja imaju sve manje i manje strukture. Tako je izvršeno modeliranje logističkog sustava s parametrom $r = 3.9999$, uz model s $D = 12$, $L = 6$, $\delta = 0.01$. On nalazi samo jedno stanje, tj. automat je gotovo identičan onom prikazanom na slici 6.14. Međutim



Slika 6.19: SFA-model logističkog sustava s $r = 3.997$. Parametri modela bili su $D = 10$, $L = 5$, $\delta = 0.055$. Po emitiranju četiri uzastopne nule sustav deterministički emitira jedinicu. Time se nagovješćuju zaostaci strukture u prevladavajućem kaosu. Daljnji su prijelazi nedefinirani, pa je model neregularan. Očito je da su za izučavanje ovog sustava potrebne riječi većih duljina ($D > 10$). Sustav pokazuje relativno veliku duljinu korelacije glede niza uzastopno emitiranih nula. Po emitiranju jedinice korelacija je nulte duljine.

provjerom glavnog stabla, duboko dolje na čvoru označenom nizom od 7 nula, mogli bismo pronaći ostatke strukture. Nju bismo uhvatili tek uz povećanje D. Uz našu aproksimaciju, model odgovara stohastičkom sustavu.

Uzimanjem parametra $r = 3.999999$ ostaci strukture još više uzmiču pred nadolazećim, "bezstrukturnim" kaosom. Zaostaci strukture tu se naziru tek nakon primitka 10 uzastopnih nula. Model SFA uz $D = 12$, $L = 6$, dat će samo jedno stanje i uz vrlo nisku vrijednost parametra $\delta = 0.001$. Ovaj sustav je po svojim odlikama vrlo blizak sustavu koji smo dobili primjenom generatora slučajnih brojeva. Vrata determinističkog kaosa su otvorena, a struktura je na izmaku.

Poglavlje 7

Zaključak

Posebnosti nelinearnih dinamičkih sustava i pojava kaotičnosti u njihovom ponašanju navodi nas na uvođenje novih pristupa u njihovom izučavanju. Jedan od takvih pristupa je i teorija *ϵ-strojeva* opisana u radovima J. P. Crutchfielda. Modeliranje dinamičkih sustava se tu vrši s pomoću računarskih alata uvedenih u teorijama izračunavanja, automata i formalnih jezika. Neposredni zadatak je da se na temelju vremenskog niza podataka odaslanih od sustava iznađe njegov model izražen u obličju računarskih strojeva (automata). Model je okarakteriziran parametrom aproksimacije ϵ, u kojem su skupljeni utjecaji konačne razlučivosti instrumenta, točnosti računskog modela glede korištenih računarskih izvorišta (ukupne duljine primljenog niza i duljine izlučenih riječi) te opisne snage njegovog razreda.

U poglavlju 2, *Teorijska polazišta*, dan je kratak prikaz simboličke dinamike i njene veze sa standardnim izučavanjem dinamičkih sustava u faznom prostoru. Simbolička dinamika predstavlja formalnu podlogu za teoriju ϵ-strojeva. Obrazloženi su temeljni pojmovi, kao što je svojstvo konjugacije između funkcije iteracije dinamičkih nelinearnih sustava i funkcije posmaka na skupu svih vremenskih nizova. Za posmak, koji je lako analizirati, pokazano je da tvori kaotičan sustav.

Vremenski niz primljen od sustava, možemo promatrati kao formalni jezik s pridruženom mu gramatikom kao teorijskim opisom njegove strukture. U pogl. 3, *Računarski pristup*, opisan je razred konačnih automata (FA), kao razred koji je prikladan za predstavljanje pravilnih struktura. Pravilne strukture sustava ogledaju se u vremenskom nizu kao proizvodi *pravilnih izraza.* Postoji ekvivalencija između FA i pravilnih izraza. Uvođenjem stohastičnosti kao nužne odrednice u opisu realnih sustava, razred konačnih automata prelazi u razred *stohastičkih konačnih automata* (*SFA*). SFA je formalno definiran svojim skupom stanja, početnim stanjem, abece-

dom ulaznih simbola, te strukturom i vjerojatnostima mogućih prijelaza, što se prikazuje odgovarajućim dijagramom prijelaza.

Detaljniji prikaz SFA, kao temeljnog koncepta u našem modeliranju, dan je u pogl. 4 *Gradba ϵ-strojeva.* Prikupljanje podataka u vidu eksperimenta je prvi, i neizbježni izvor aproksimacije u modeliranju. Teorije koje ne uvažavaju tu fundamentalnu postavku predstavljaju (samo) idealizaciju stvarnosti. Instrument kao ključni dio *mjernog kanala* vrši diskretizaciju mjernog prostora, čime se opis sustava iz vremensko-prostornog kontinuuma mora prebaciti u diskretnu domenu. To nas navodi na zaključak da su računarski alati, kao diskretne tvorevine, i suštinski sukladni eksperimentalnoj stvarnosti. Simbolička dinamika, odnosno njena konačna aproksimacija u vidu posmaka konačnog tipa, točno korespondira s diskretnim rezultatima mjerenja u vidu niza simbola. Uz *generirajuću particiju* dovoljno je pri mjerenju (odnosno kodiranju) koristiti *grubu podjelu.* Time se rezultati mjerenja i formalno poklapaju s pojmom orbite u simboličkoj dinamici. Činjenica da nam računarska izvorišta omogućuju analizu (pod)nizova samo konačne duljine, znači da ćemo u realnoj primjeni imati postavljenu neku gornju granicu točnosti.

Vjerojatnosni parametri sadržani u SFA nam omogućuju da uvedemo *statističku složenost* kao kvantitativnu mjeru za strukturu sustava. Radi se o veličini koja ima bitnu kvalitativnu novinu, u smislu da se potpuno stohastičkim sustavima pridjeljuje nulta složenost. To proizlazi iz definicije statističke složenosti kao minimalnog programa potrebnog za reprodukciju (ekvivalenta) nekog sustava na *Bernoulli-Turingovom stroju* (BTM). Ovaj stroj ima poseban izvor stohastičnosti za simuliranje slučajnih procesa. Na taj se način *deterministički* koncept Kolmogorov-Chaitinove složenosti (definirane na temelju univerzalnog Turingovog stroja), zamjenjuje sa *statističkim* konceptom.

Algoritam gradbe ϵ-strojeva iterativan je proces, temeljen na uočavanju struktura u podacima predočenim vremenskim nizom. Strukture se utjelovljuju u pojmu morfova, kao konceptu koji opisuje mogućnost razvoja budućih stanja na temelju trenutnog. Podastrijeta je općenita definicija morfova kao razreda ekvivalencije s obzirom na moguće buduće prijelaze. Svi koraci gradbe detaljno su opisani u 4.4. U sljedećem odjeljku, 4.5, navodi se konkretizacija gornjeg algoritma za tvorbu modela do uključivo razine SFA, koji predstavljaju temeljne kauzalne modele, odnosno temeljne ϵ-strojeve. Polazi se od definicije ϵ-stabla te obrazlaže raščlamba vremenskog niza u procesu *hranidbe stabla.* U dobivenom se raščlambenom stablu pronalaze *jedinstvena podstabala*, ili *morfovi*, koja predstavljaju stanja SFA-modela. Prijelazi

između njih i njihove vjerojatnosti u potpunosti definiraju SFA (alg. 4.2). Taj je algoritam implementiran u okviru razvijene programske podrške.

Zaključno s poglavljem 4 završen je teorijski dio ovog rada. Njemu je posvećen relativno velik obim pošto teorija ϵ-strojeva predstavlja novi pristup. Skromni doprinos ovog dijela izlaganja sastoji se u nastojanju autora da pojača formalnu vezu ove teorije sa simboličkom dinamikom, te da se naznači geneza SFA iz FA. Struktura glavnog stabla je definirana kao posebni slučaj binarnog stabla.

U poglavlju 5, *Razvoj programskog okruženja*, opisani su praktični i izvorni rezultati rada. Razvijen je program *Dynamical Systems Automata* (DSA), koji iz binarnog vremenskog niza nalazi model sustava do opisane razine SFA. Program je napisan u jeziku C++ uz dosljednu primjenu objektno orijentirane programske paradigme. Strogo su poštovana programska načela koja doprinose korektnosti i učinkovitosti dobivenog programa te omogućuju njegovu laku naknadnu nadogradnju.

Glavna struktura stabla ostvarena je kao dinamička struktura. Time se na prostorno učinkovit način pohranjuju stabla najrazličitijih struktura, od rijetko do potpuno popunjenih. Izvorni računarski doprinos ovog rada nerekurzivni su algoritmi za *prijeredni, poslijeredni* i *međuredni* prolazak (engl. *traversal*) kroz binarna ϵ-stabla, koje je lako poopćiti za djelovanje na općenitim binarnim stablima. Ostvareno je pregledno grafičko sučelje za prikaz glavnog stabla, koje omogućuje njegov potpun pregled, analizu svih čvorova i njihovih prijelaznih vjerojatnosti.

Postupak nalaženja morfova kao jedinstvenih podstabala unutar glavnog stabla, posjeduje relativno veliku hijerarhijsku složenost. Algoritam opisan u 5.4.2 koristi koncepte *podstabla* i *glavnog stabla* utjelovljene u zasebnim klasama, koje su obje nasljednice temeljne klase binarnog stabla.

Poglavlje 6 *Primjena i rezultati* započinje prikazom uporabe programa. To je izvršeno na jednostavnom primjeru *sustava zadanog pravilom*, analitički riješenom ranije. Korisnik u svakom trenutku ima pregled nad podatkovnim strukturama u kojima se čuvaju podaci o sustavu i izvršenim iteracijama, o strukturi glavnog stabla, o rezultatu modeliranja izraženom u vidu skupine pronađenih jedinstvenih morfova te o strukturi i vjerojatnosti njihovih međusobnih prijelaza. Na nizu sličnih primjera izvršeno je testiranje programa. Sustavi zadani pravilom korespondiraju s razredom SFA. Elemente modela ovih jednostavnih sustava pronalazit ćemo općenito i u nelinearnim dinamičkim sustavima. Za nekoliko najzanimljivijih sustava prikazani su odgovarajući automati. Izneseni su i njihovi statistički pokazatelji, u vidu topološke i statističke složenosti. Iako načelno jednostavni, već i ovi modeli ukazuju na oprav-

danost i svrsishodnost ovog novog pristupa modeliranju dinamičkih sustava i izučavanju njihove strukture.

U završnom odjeljku opisani su prvi rezultati primjene programske podrške na nelinearnom sustavu opisanom logističkom jednadžbom. Logistička jednadžba, kao široko poznati primjer, poslužila je da se ilustrira kvalitativna novina koju pruža ostvareni model. Za područja periodičke orbite radi se o (trivijalnim) modelima s deterministčkim prijelazima između stacionarnih stanja. Idemo li dalje, "na putu prema kaosu" otkrivamo bogatstvo struktura koje naši modeli oslikavaju na kvalitativno nov način. Za Misiurewiczev parametar sustav ima stalan broj od četiri morfološki jedinstvena stanja, dok su varijacije u vjerojatnostima prijelaza razlog za uočavanje većeg broja δ-različitih stanja. Otpuštanjem parametra vjerojatnosne razlike δ na relativno nisku točnost, otkrivamo temeljni strukturalni model prikazan upravo s ta četiri stanja. S druge strane, na samom početku puta prema kaosu udvostručavanjem perioda, ponašanje sustava potpuno je oprečno. Broj morfološki različitih stanja, pa prema tome i broj ukupnih stanja neovisno o izboru parametra δ, raste s porastom točnosti modela bez granica. Struktura, izražena kroz statističku složenost, raste približno kao logaritam broja stanja te, prema tome, također divergira. Izgleda da je sustav nepresušni izvor novih pojavnosti i da je neuhvatljiv za računski model. Međutim, uz uvođenje inovacijskog koraka i prelaskom na višu razinu automata (strojeva za proizvodnju nizova) pokazuje se da je moguće postići konačan model.

Kako se približavamo potpuno kaotičnim područjima, modeli temeljeni na SFA oduzimaju puku stohastičnost iz takvih procesa, nakon čega preostaje relativno jednostavna struktura. Zaostatke strukture možemo pronaći tek uz produljenje opažanih korelacija, što će zahtijevati povećani računarski napor. Ili se — uz zadanu ϵ-aproksimaciju — možemo zadovoljiti opisom sustava kao kvazistohastičkog, jednakog onom za režim potpunog kaosa, kojeg simulira SFA s jednim stanjem.

Zahvala. *Zahvaljujem svojem mentoru, prof. dr. Leu Budinu, na usmjerenju ovog magistarskog rada prema području nelinearnih dinamičkih sustava, na ukazanom povjerenju pri izboru teme i strpljenju glede mog kroničnog kašnjenja. J. P. Crutchfieldu zahvaljujem na njegovom "inovativnom koraku" glede uvođenja računarskih teorija u istraživanje fundamentalnih problema te na korisnim savjetima tijekom izrade rada. Posebne zahvale idu mojim dragim prijateljima, B. Lukmanu za izrađene slike i P. Mavaru za pregled teksta.*

Dodatak A

Popis klasa i struktura nasljeđivanja

```
// Dynamical system classes
    class CDynSys1D : public CObject
        class CDynSys1D_0Par : public CDynSys1D
            class CDS1D0P_0IfPrevIs1 : public CDynSys1D_0Par
            class CDS1D0P_0orDouble1s : public CDynSys1D_0Par
            class CDS1D0P_1IfPrevIs0 : public CDynSys1D_0Par
            class CDS1D0P_BernoulliSer : public CDynSys1D_0Par
            class CDS1D0P_DoublingFunc : public CDynSys1D_0Par
            class CDS1D0P_EvenNumOfConsec1s : public CDynSys1D_0Par
            class CDS1D0P_EvenNumOfConsec0sAnd1s : public CDynSys1D_0Par
            class CDS1D0P_NoConsec0s : public CDynSys1D_0Par
            class CDS1D0P_OddNumOfConsec1s : public CDynSys1D_0Par
        class CDynSys1D_1Par : public CDynSys1D
            class CDS1D1P_AbsoluteValue : public CDynSys1D_1Par
            class CDS1D1P_LogisticMap : public CDynSys1D_1Par
            class CDS1D1P_QuadraticMap : public CDynSys1D_1Par

// Tree classes
    class CABinTree :  public CObject
        class CMainABinTree :  public CABinTree
        class CSubABinTree :  public CABinTree
// Tree helper classes
    class CBinTreeNode :  public CObject
    class CTreeStatistics :  public CObject

// Morph classes
    class CATrMorphs :  public CObject
// Morph helper classes
    class CMrphTrans :  public CObject
```

```
// Dialog classes
    class CDsSelectSystemDlg :  public CDialog
    class CDsParametersDlg :  public CDialog
    class CDsBatchIterDlg :  public CDialog
    class CTreeNewDlg :  public CDialog
    class CTreeSettingsDlg :  public CDialog

    class CTreeFeedAndShowDlg :  public CDialog
        class CMorphsFindAndShowDlg :  public CTreeFeedAndShowDlg

    class CMorphsNewDlg :  public CDialog
    class CMorphsSelectDlg :  public CDialog
    class CMorphTransDlg :  public CDialog

// Frame classes
    class CMainFrame :  public CMDIFrameWnd
    class CChildFrame :  public CMDIChildWnd
    class CDSATreeChildFrame :  public CChildFrame
    class CDynSysChildFrame :  public CChildFrame

// View classes
    class CDSATreeView :  public CScrollView
    class CDynSysAView :  public CScrollView

// Document class
    class CDynSysADoc :  public CDocument

// Appllication class
    class CDynSysAApp :  public CWinApp
```

Dodatak B

Osnovna klasa stabla

Ispis B.1 *Sučelje osnovne klase stabla CABinTree iz zaglavne datoteke ABinTree.h*

```
////////////////////////////////////////////////////////////////////
// ABinTree.h:  interface for the CABinTree class.
////////////////////////////////////////////////////////////////////

#include "BinTreeNode.h"
#include "TreeStatistics.h"

////////////////////////////////////////////////////////////////////
//GLOBAL TYPE DEFINITIONS:
//
typedef enum {mainTree, subTree} treeType ;
typedef enum {horizontal, columns, graphical} displayType;
// typedef enum {f0, f1, f2, f3, f4, f5, f6} cmpFlgType ;

const UINT uIntAritBitLen = 32;
const UINT nFeedFactor = 8192; // Default feed factor.
const UINT iDfltTreeHeight = 5;
const UINT iDfltSubTrHeight = 2;
const UINT iTreeHeightMax = 20;
const UINT UINTMax = 4294967295;
const UINT nMinToShwPrgrsBar = 262144;
const UINT nVarFactor = 16;
const UINT nBItrInOutrLp = 1024;
const float fDfltDelta = 0.001f; // Default delta value.

bool AssertValidBinString(CString sBinStr);
float DivUINTToFloat(UINT u1, UINT u2);

int CreatePrgrsBar(CProgressCtrl& PrgrsBar, CWnd* pPrntWnd,
int iRngLo, int iRngHi, int iPos = 0, int iStep = 0);

//
////////////////////////////////////////////////////////////////////
```

```
class CABinTree :  public CObject
{
// Data members:
protected:
treeType m_treeType; // The type of the tree: mainTree or subTree.
UINT m_iHeight; // For binary trees, this is the longest path from the
        // root to a leaf.  For ABinTrees, once chosen, this is a con-
        // stant value, and all the paths from the root to any leaf have
        // the same path length.  In other words, all leaves are on the
        // same level, which is equivalent to the tree height.
//////
CTreeStatistics m_TreeStat; // The tree statistics is contained in
//////                // this embedded object of class CTreeStatistics.

bool m_bDone; // True if the traversal is done.

private:
CBinTreeNode* m_pRoot; // Pointer to the tree root.
bool m_bIsOptFed; // True if the tree is optimally fed.
float m_fDelta;  // Probabilities are delta-equal if they differ
                // within the value of m_fDelta.

UINT m_nFeedFactor; // Factor showing the number of feeds per
        // every predicted leaf (see function CalcIsOptFed()).
        // The member m_nFeedFactor should be greater than the
        // reciprocal value of m_fDelta.

bool m_bTakeMaxLvsNum; // For explanation, see functions
                       // SetnStrToFeedAuto and CalcIsOptFed.
UINT m_nStrToFeedMan;

// Member Functions

// Constructor/destructor and initialization:
public:
CABinTree(); // Empty constructor (for serialization only).

CABinTree(UINT height);
CABinTree(CBinTreeNode* pNdR, UINT iHeight);
CABinTree(CBinTreeNode* pNdR, UINT iHeight, int iTravrsType);

virtual CABinTree* NewSubTree(CBinTreeNode* pNdR, UINT hSub) = 0;
virtual CABinTree* NewSubTree(CBinTreeNode* pNdR, UINT hSub,
                   int iTravrsType) = 0;

virtual CABinTree* NewCopyAsMainTree
                   (CBinTreeNode* pNd, UINT height) = 0;
        // Copies the subtree specified by its root node pointer pNd
        // and argument iHeight as a separate, independent main tree.
        // Not implemented at this class level.
```

```
virtual ~CABinTree();
void ReInitialize();

inline void Reset() { ReInitialize(); m_TreeStat.InitToZero();}

// Accessors:
treeType GetTreeType() const { return m_treeType;}
UINT GetiHeight() const { return m_iHeight;}

CTreeStatistics& GetTreeStatistics() { return m_TreeStat;}
float GetfDelta() const { return m_fDelta;}
CBinTreeNode* GetpRoot() const { return m_pRoot;}
bool GetbDone() const { return m_bDone;}
bool GetbTakeMaxLvsNum() const { return m_bTakeMaxLvsNum;}
UINT GetnMaxLeaves() const;
bool GetbIsOptFed() const { return m_bIsOptFed;}
UINT GetnFeedFactor() const { return m_nFeedFactor;}
UINT GetnStrToFeedMan() const { return m_nStrToFeedMan;}

void GetParamToTreeNewDlg(CDialog* pDl) const;
void GetParamToTrFdAndShwDlg(CDialog* pDl);
void GetNodeCntsToTrFdAndShwDlg(CDialog* pDl);

virtual CABinTree* GetpMainTree() { return NULL;}
            // Not implemented at this class level.

// Retrieves the value of the root node counter, which corresponds
// to the number of strings fed into the tree:
inline UINT GetnStringsFed(CBinTreeNode* pNd) const
{ return pNd->GetnCount();}

bool IsEmpty() const { return (m_pRoot == NULL);} // No nodes!
bool IsFed() const { return (m_pRoot->GetnCount() > 0);}
inline bool IsOptFed()
{ CalcIsOptFedAndRetnStrToFeedAuto(); return m_bIsOptFed;}

// Validation:
void AssertValid() const;
// Serialization:
virtual void Serialize(CArchive& ar);

// Modifiers:
protected:
void SetTreeStat(CTreeStatistics& numN) { m_TreeStat = numN;}

public:
void SetfDelta(float fDelta) { m_fDelta = fDelta;}
void SetpRoot(CBinTreeNode* root) { m_pRoot = root;}
```

```
inline void SetnFeedFactor(UINT nFdFct)
{ m_nFeedFactor = nFdFct;}

inline void SetnFeedFactor(UINT nFdFct, float fDelta)
{ m_nFeedFactor = ChooseFeedFactor(nFdFct, fDelta);}

void SetnStrToFeedMan(UINT nStr) { m_nStrToFeedMan = nStr;}
void SetnStrToFeedAuto(bool bTakeMaxLvsNum = true);
// void SetsCurrent(CString sCurrent) { m_sCurrent = sCurrent;}

protected:
void SetiHeight(UINT iTrHeight) { m_iHeight = iTrHeight;}

inline void SetbTakeMaxLvsNum(bool bMLN = true)
{ m_bTakeMaxLvsNum = bMLN;}

void SetbIsOptFed (bool fed = true) { m_bIsOptFed = fed;}

public:
void SetParamFromTreeNewDlg(CDialog* pDl);
void SetParamFromTrFdAndShwDlg(CDialog* pDl);

float CheckfDelta(float fDelta);
UINT ChooseFeedFactor(UINT nFdFct, float fDelta);

//////////////////////////////////////////////////////////////////////

inline UINT Level(CBinTreeNode* pNd) const
{ return (pNd->GetnAbsLev() - m_pRoot->GetnAbsLev()); }
  // Calculates the (relative) level of the node within "this" tree.
  // If Level(pNd) == m_iHeight, the node pointed by pNd is a
  // leaf (see also IsLeaf function).
  // The relative level is crucial for the notion of subtree,
  // which will be developed in the CSubABinTree class.
  // The root of a subtree can be any node of some tree, on
  // an arbitrary level of that tree.

inline bool IsLeaf(CBinTreeNode* pNd) const
{ ASSERT( (pNd != NULL) && !IsEmpty() );
    if ((pNd->GetnAbsLev() - m_pRoot->GetnAbsLev()) == m_iHeight)
        return true;
    else return false; }
  // Returns true if the current node is a leaf, otherwise false.
  // The tree must be nonempty, and pNd != NULL.

bool IsLeftChild(CBinTreeNode* pNd) const;
bool IsRightChild(CBinTreeNode* pNd) const;
hasChldrnType HasChildren(CBinTreeNode* pNd) const;
```

```
CBinTreeNode* GetpRSibling(CBinTreeNode* pNd) const; // Returns a
              // pointer to the right sibling of the node *pNd.

CBinTreeNode* InitPreOrd(); // Returns a pointer to the node that
                           // initiates the preorder traversal.
CBinTreeNode* TravrsPreOrd(CBinTreeNode* pNdC);
   // The function argument is the pointer pNdC to the current
   // node. The function returns a pointer to the next node in a
   // preorder traversal.

CBinTreeNode* InitInOrd(); // Returns a pointer to the node that
                          // initiates the inorder traversal.
CBinTreeNode* TravrsInOrd(CBinTreeNode* pNdC);
  // Returns a pointer to the next node in an inorder traversal.

CBinTreeNode* InitPostOrd(); // Returns the pointer to the node that
                            // initiates the postorder traversal.
CBinTreeNode* TravrsPostOrd(CBinTreeNode* pNdC);
  // Returns a pointer to the next node in a postorder traversal.

void CalcTreeStatPreOrd();
  // Counts the numbers of the left and right nodes and the left and
  // right leaves of the tree, by completely traversing the tree in
  // a preorder traversal.
  // Changes the m_TreeStat data member accordingly. Note that the
  // total number of nodes in the tree is:
  //
  //   TreeStat = nLNodes + NRNodes + 1 ; ( 1 is for the root.)
  //

void CalcTreeStatInOrd();    // As above, with inorder traversal.
void CalcTreeStatPostOrd(); // As above, with postorder traversal.

CTreeStatistics& CalcTreeStatPreOrd(CBinTreeNode* pNd, UINT hSub);
  // The function calculates the tree statistics of a subtree of
  // height hSub that stems from the tree node pointed by pNd.
  // The (sub)tree statistics includes the data mentioned above:
  // the numbers of left and right nodes and left and right leaves.
  //   The node *pNd, which is the root of the subtree below it,
  // is not counted.  As a root, it is neither left nor right child
  // and neither left nor right leaf of that subtree.

CTreeStatistics& CalcTreeStatistics(CBinTreeNode* pNd, UINT hSub,
                                    int iTravrsType);
  // This function is used for testing of the traversal functions.
  // Performs the same job by using different traversal functions,
  // depending on the iTravrsType:  0 = pre, 1 = in, 2 = post(order).

CTreeStatistics& CalcTreeStatistics(int iTravrsType)
{ return CalcTreeStatistics(m_pRoot, m_iHeight, iTravrsType);}
```

```
inline float NodeProblty(CBinTreeNode* pNd, CBinTreeNode* pNd0)
{ return float(pNd->GetnCount())/float(pNd0->GetnCount());}
  // Probability of a node pointed by pNd, conditioned to the tree
  // node pointed by pNd0. The node *pNd must be a descendant of
  // *pNd0 -- this is a precondition for the application of this
  // function. The probability is simply the ratio of counts in
  // *pNd and *pNd0 nodes.

inline float NodeProblty(CBinTreeNode* pNd)
{return float(pNd->GetnCount())/float(m_pRoot->GetnCount());}
  // Probability of a node pointed by pNd, conditioned to the root of
  // the tree. (It is equal to the probability of appearance of the
  // substring represented by that node in the received binary series.

inline float NodeProbltyWithAChck(CBinTreeNode* pNd,
                                  CBinTreeNode* pNd0);
  // Use this function if not sure whether the pNd0 corresponds to
  // an ancestor of node pointed by pNd.  If so, the probability is
  // returned, otherwise .0f is returned.
  //   This function is always safe, even if one or both nodes have
  // zero counts.  In that case .0f is returned. Note that the count
  // for an existing node must be >= 1, so that the return of zero
  // (.0f) probability for such nodes means inconsistency!

inline float SubTreeProblty()
{ ASSERT(m_pRoot->GetpParent() != NULL);
    return float(m_pRoot->GetnCount())
    float(m_pRoot->GetpParent()->GetnCount()); }
  // Probability of a subtree conditioned to the subtree parent.

CBinTreeNode* StringToNode(CBinTreeNode* pNd, CString str);
  // Returns a pointer to the node that is defined by a string str,
  // starting from the node pointed by pNd.  If such a string does
  // not exist in the tree, the function returns NULL.

CBinTreeNode* StringToNode(CString str)
{ return StringToNode(m_pRoot, str);}

float StringProbability(CBinTreeNode* pNd, CString str);
  // Probability of a given string str, starting from the node
  // pointed by pNd.
  //   The function uses the above function, StringToNode, to find
  // the node defined by pNd and the string str.  Function returns .0f
  // as a sign of inconsistency if the number of counts in the *pNd
  // node is 0, or if str is not in the tree.
  // (In the case that a node exists but has zero counts, the ASSERT
  //  macro in the function warns about the inconsistency of the
  //  CABinTree object.)
```

```
CString NodeFullString(CBinTreeNode* pNd) const;
  // Returns a string that corresponds to the node pointed by pNd.
  // The pointer m_pCurrent is not affected.

CString NodePartString(CBinTreeNode* pNd0, CBinTreeNode* pNd) const;
  // Returns the binary string starting at *pNd0 and ending in *pNd.
CBinTreeNode* GetAncestor(CBinTreeNode* pNd0,
                          CBinTreeNode* pNd, int iDeltaLev);
  // Finds the ancestor of the node *pNd, which (the ancestor)
  // is on the level iDeltaLev greater than the node *pNd0.

inline CBinTreeNode* GetAncestor(CBinTreeNode* pNd, UINT iUp)
{ return GetAncestor(m_pRoot, pNd, iUp);}

UINT GetnStrToFeedAuto(bool bTakeMaxLvsNum = true) const;
  // The function calculates and returns the number of strings that
  // have to be fed into the tree when the automatic (Auto) feed
  // option is selected.
  //   If the argument bMaxLvsTree is true, the maximum possible
  // number of leaves is taken for the supposed number of leaves,
  // which equals to 2^height.
  //   Otherwise, if bMaxLvsTree is false, the mean value of the
  // currently accumulated number of leaves and the maximally
  // possible number of leaves is taken as the presumed number of
  // leaves.
  //   In both cases, the obtained number of leaves is then multi-
  // plied by the member m_nFeedFactor.

UINT CalcIsOptFedAndRetnStrToFeedAuto();
  // Calculates if the tree is optimally fed, and sets the
  // m_bIsOptFed accordingly. The function includes a call to
  // the function SetnStrToFeedAuto and returns its return value.

float CompareTo(CABinTree* pTr, CString& sMaxDiffNd);
  // Compares the tree to another tree, pointed by pTr, for their
  // morphological equality and for delta-equality of the proba-
  // bilities of their nodes.

bool operator == (CABinTree& pTr);
  // The operator uses the CompareTo function and returns true
  // if ComapreTo returns the flag value that is smaller or equal
  // to m_fDelta

void Display(displayType displayHow) {} // Display the tree.

};

//////////////////////////////////////////////////////////////////
```

Ispis B.2 *Implementacija nerekurzivnog poslijerednog prolaska kroz binarno ϵ-stablo. Funkcija* `InitPostOrd` *inicijalizira prolazak nalaženjem najljevijeg lista. Funkcija* `TravrsPostOrd` *nalazi čvor koji se posjećuje nakon čvora definiranog kazaljkom* `pNdC`*. Čvor može biti posjećen tek ako su posjećena njegova djeca, najprije lijevo (ako postoji) pa desno (ako postoji). Ako je prolazak gotov, funkcija postavlja člansku varijablu* `m_bDone` *na istinitu vrijednost.*

```
CBinTreeNode* CABinTree::InitPostOrd()
{
    CBinTreeNode* pNdC;

    // Consistency check:  the tree must be nonempty!
    ASSERT(m_pRoot != NULL);
    ASSERT( (m_pRoot->GetnCount() != 0) || (m_pRoot->GetpLChild() ==
            NULL) || (m_pRoot->GetpRChild() == NULL) );

    CBinTreeNode* pNd;
    m_bDone = false;
    pNdC = m_pRoot;

    UINT tmp = Level(pNdC);
    while (Level(pNdC) < m_iHeight) // Step to the leftmost leaf:
    {
        if ( (pNd = pNdC->GetpLChild()) != NULL)
            pNdC = pNd; // To LChild.
        else
            pNdC = pNdC->GetpRChild(); // To RChild.
    }       // pNdC is pointing to the leftmost leaf.

    return pNdC;
}

CBinTreeNode* CABinTree::TravrsPostOrd(CBinTreeNode* pNdC)
{
    CBinTreeNode* pNd;

    if (pNdC == m_pRoot) // If a postorder traversal reaches the
    {                    // root node, then it's done!
        m_bDone = true;
    }
    else if ( (pNd = GetpRSibling(pNdC)) != NULL)
     // The current node is a left child, and furthermore, the
    // node's right sibling exists.
    {
        pNdC = pNd;  // Move to the right sibling, and look for
                     // its leftmost leaf, if any.
        while (Level(pNdC) < m_iHeight)
        {
            if ( (pNd = pNdC->GetpLChild()) != NULL)
```

```
                pNdC = pNd; // To LChild.
            else
                pNdC = pNdC->GetpRChild(); // To RChild.
        }
    }
    else // The current node is a right child, or the right sibling
        // does not exist.
    {
        ASSERT( IsRightChild(pNdC) || (GetpRSibling(pNdC) == NULL) );
        if (Level(pNdC) > 0)
            pNdC = pNdC->GetpParent(); // To Parent.
        else     // At the root!!!
            m_bDone = true; // Done. No more right siblings.
    }

    return pNdC;
}
```

Ispis B.3 *Implementacija nerekurzivnog međurednog prolaska kroz binarno ϵ-stablo. Funkcija* `InitInOrd` *inicijalizira prolazak nalaženjem najdaljeg uzastopnog lijevog nasljednika korijena. Funkcija* `TravrsInOrd` *nalazi čvor koji se posjećuje nakon čvora definiranog kazaljkom* `pNdC`*. Čvor može biti posjećen tek ako su posjećeni svi čvorovi podstabla koje izvire iz njegovog lijevog djeteta. Ako je prolazak gotov, funkcija postavlja člansku varijablu* `m_bDone` *na istinitu vrijednost.*

```
CBinTreeNode* CABinTree::InitInOrd()
{
    CBinTreeNode* pNdC;
    // Consistency check:  the tree must be nonempty!
    ASSERT(m_pRoot != NULL);
    ASSERT( (m_pRoot->GetnCount() != 0) ||
                (m_pRoot->GetpLChild() == NULL) ||
                (m_pRoot->GetpRChild() == NULL) );

    CBinTreeNode* pNd;
    m_bDone = false;
    pNdC = m_pRoot;

    while ( ((pNd = pNdC->GetpLChild()) != NULL) &&
            (Level(pNdC) < m_iHeight) ) // Step to the furthest
        pNdC = pNd;                     // consecutive left child.

    return pNdC;
}
```

```
CBinTreeNode* CABinTree::TravrsInOrd(CBinTreeNode* pNdC)
{
    CBinTreeNode* pNd;

    if ( (Level(pNdC) < m_iHeight) &&
        ((pNd = pNdC->GetpRChild()) != NULL) )
    // There is an unvisited subtree to the right:
    {
        pNdC = pNd; // To the right child.
        while ( ((pNd = pNdC->GetpLChild()) != NULL) &&
                (Level(pNdC) < m_iHeight) ) // While possible, move
            pNdC = pNd;                     // to the left child.
    }
    else if (IsLeftChild(pNdC)) // This is a left child without the right
                                // child of its own. Move to parent:
        pNdC = pNdC->GetpParent();

    else // This is a right child without the right child of its own.
    {
        ASSERT(IsRightChild(pNdC)); // Consistency check!
        do
        {
            pNdC = pNdC->GetpParent();
        }
        while (IsRightChild(pNdC));

        if (Level(pNdC) > 0)
            pNdC = pNdC->GetpParent();
        else
            m_bDone = true;
    }

    return pNdC;
}
```

Bibliografija

[1] J. P. Crutchfield,[1] "Knowledge and Meaning," in *Modeling Complex Phenomena* (L. Lam and V. Naroditsky, editors), pp. 66–101. Springer, Berlin, 1992.

[2] R. L. Devaney, *A First Course in Chaotic Dynamical Systems*. Addison-Wesley, Reading, Massachusetts, 1992.

[3] R. L. Devaney, *An Introduction to Chaotic Dynamical Systems*, Second Edition. Addison-Wesley, Redwood City, California, 1989.

[4] F. C. Moon, *Chaotic and Fractal Dynamics*. John Wiley & Sons, New York, 1992.

[5] W. H. Zurek, editor, *Complexity, Entropy and the Physics of Information*, vol. VIII of *Santa Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley, Reading, Massachusetts, 1990.

[6] M. Casdagli and S. Eubank, editors, *Nonlinear Modeling and Forecasting,* vol. XII of *Santa Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley, Reading, Massachusetts, 1992.

[7] J. P. Crutchfield, "Observing Complexity and the Complexity of Observation," in *Series in Synergetics* (H. Atmanspacher, ed.), pp. 235–272. Springer, Berlin, 1993.

[8] J. P. Crutchfield, "Is Anything Ever New? Considering Emergence," in *Integrative Themes*, vol. XIX of *Santa Fe Institute Studies in the Sciences of Complexity* (G. Cowan, D. Pines, and D. Melzner, eds.), pp. 479–497. Addison-Wesley, Reading, Massachusetts, 1994.

[9] J. P. Crutchfiled, "Semantics and Thermodynamics," in [6], pp. 317–359 (1992).

[10] J. P. Crutchfield, "The Calculi of Emergence: Computation, Dynamics, and Induction," *Physica D*, vol. 75(1–3), a special issue on the *Proceedings of the Oji International Seminar: Complex Systems — from Complex Dynamics to Artificial Reality*, Numazu, Japan, pp. 11–54, 1994.

[1] Noviji radovi J. P. Crutchfielda dostupni su u Post Script formatu na Internet adresi: http://www.santafe.edu/projects/CompMech/papers/CompMechCommun.html (1999.).

[11] N. Chomsky, "Three Models for the Description of Language," *IRE Transactions on Information Theory*, vol. 2(3), pp. 113–124, 1956.

[12] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

[13] J. P. Crutchfield and Karl Young, "Computation at the Onset of Chaos," in [5], pp. 223–269 (1990).

[14] A. N. Kolmogorov, "A New Metric Invariant of Transient Dynamical Systems and Automorphisms in Lebesgue Spaces" (original in Russian), *Dokl. Akad. Nauk. SSSR*, vol. 119, pp. 861–864, 1958.

[15] J. P. Crutchfield and N. H. Packard, "Symbolic Dynamics of Noisy Chaos," *Physica D,* vol. 7(1–3), pp. 201–223, 1983.

[16] Thomas M. Cover, *Elements of Information Theory*. John Wiley & Sons, New York 1991.

[17] S. Wolfram, "Computation Theory of Cellular Automata," *Communications in Mathematical Physics,* vol. 96(1), pp. 15–57, 1984.

[18] Ž. Pauše, *Uvod u teoriju informacije*, II. dopunjeno izdanje. Školska knjiga, Zagreb, 1989.

[19] A. B. Tucker at al., *Fundamentals of Computing II, C++ Edition*. McGraw-Hill, New York 1995.

[20] C. S. Horstmann, *Mastering Object-Oriented Design in C++*. John Wiley & Sons, New York 1995.

[21] N. Wirth, *Algorithms & Data Structures*. Prentice/Hall International, London, 1986.

[22] J. P. Crutchfield, "Critical Computation, Phase Transitions, and Hierarchical Learning," in *Towards the Harnessing of Chaos* (M. Yamaguti ed.), pp. 29-46. Elsevier Science, Amsterdam, 1994.

Modeliranje dinamičkih sustava s pomoću stohastičkih konačnih automata

Magistarski rad

Robert Logožar
Sveučilište u Zagrebu
Fakultet Organizacije i informatike Varaždin
Pavlinska 2, HR-42000 Varaždin, Croatia
rlogozar@foi.hr

Sažetak

Predstavljen je novi pristup u izučavanju jednodimenzionalnih dinamičkih sustava, temeljen na teoriji ϵ-strojeva J. P. Crutchfielda. Zadatak je iznaći model procesa zaključivanjem isključivo na temelju vremenskog niza primljenog od sustava. Zaokružena je formalna veza između simboličke dinamike i raščlambe binarnog vremenskog niza. Razrađena je geneza stohastičkih konačnih automata (SFA) iz razreda konačnih automata (FA). Obrazložena je statistička složenost kao kvalitativno nova mjera za strukturnu složenost realnih sustava. Opisan je općeniti algoritam hijerarhijske rekonstrukcije ϵ-strojeva i dana njegova specijalizacija za nalaženje temeljnog ϵ-stroja — kauzalnog modela kojeg predstavlja SFA. Prethodnu, prvu razinu modela predstavlja raščlambeno stablo — binarno stablo sa svim listovima na istoj razini, nastalo raščlambom svih mogućih riječi fiksne duljine koje se dadu izlučiti iz vremenskog niza. Konstruirani su izvorni algoritmi za prijeredni, poslijeredni i međuredni nerekurzivni prolazak kroz stabla. Algoritam za usporedbu (pod)stabala provjerava njihovu morfološku i vjerojatnosnu podudarnost. On je okosnica algoritma za pronalaženje jedinstvenih podstabala zadane visine, koje nazivamo morfovi. Pronađeni morfovi definiraju stanja, a uvjetne vjerojatnosti prijelaza između njih stohastička svojstva SFA. Prema načelima objektno orijentiranog programiranja kreiran je DSA program s grafičkim sučeljem, za simulaciju dinamičkih sustava i nalaženje njihovih SFA. Program je testiran na nizu sustava zadanih pravilnim izrazima, za koje su prikazani njihovi SFA i izračunati statistički parametri. Preliminarno su istraženi i nelinearni sustavi logističkog preslikavanja za različite iznose kontrolnog parametra, uključujući i Misiurewiczev te kritični parametar, za početak puta prema kaosu udvostručenjem perioda. Dobiveni rezultati u skladu su s onima koje je iznio autor teorije. Pregled proizvoljnog dijela glavnog stabla i rasporeda morfova u njemu omogućuju učinkovitu provjeru i analizu dobivenih SFA. Također, dodatne prikupljene informacije o SFA-modelu olakšavaju razmatranje optimizacije njegove veličine, tj. mogućeg smanjenja broja njegovih stanja s povećanjem vjerojatnosno-diskriminirajućeg parametra δ.

Modeling of Dynamical Systems by Stochastic Finite Automata

Master Thesis

Robert Logožar
University of Zagreb, Croatia
Faculty of Organization and Informatics Varaždin
Pavlinska 2, HR-42000 Varaždin, Croatia
rlogozar@foi.hr

Abstract

A new approach to the study of dynamical systems, based on J. P. Crutchfield's theory of ϵ-machines, is presented. The goal is to find the model of the process by inferring solely from a time series that is received from the system. A formal connection between symbolic dynamic and the parsing of the time series is made. Stochastic finite automata (SFA) are derived from the class of finite automata (FA). The statistical complexity is described as a qualitatively new measure of structural complexity or real systems. A general algorithm for the hierarchical reconstruction of ϵ-machines is elaborated and its specialization for finding of the basic ϵ-machine — a causal model presented by an SFA — is given. The first level of the model is presented by the parse tree. It is a binary tree with all its leaves on the same level, which is created by parsing all the possible words of a fixed length that can be extracted from the time series. The original algorithms for preorder, postorder and inorder tree traversals are devised. The algorithm for comparison of (sub)trees checks their morphological and probabilistic structure. It is an essential part of the algorithm for finding the unique subtrees of a given height, called morphs. The morphs define the states, and the conditional probabilities of the transitions between them define the stochastic properties of the SFA. Following the principles of object-oriented programming, we have developed a DSA program with a graphical user interface, for simulation of dynamical systems and finding of their SFAs. The program was tested on a number of rule-based systems. The resulting SFA-models and the corresponding statistical parameters are given. A preliminary investigation of nonlinear systems governed by logistic map is made for several values of the control parameter, including Misiurewicz and the critical parameter, at the first period-doubling onset of chaos. The obtained results are concordant with those of the author of the theory. The graphical presentations of the main tree and morphs' positions enable efficient verification and analysis of the found SFAs. Also, the additional information gathered about the SFA-models helps in the investigation of its size optimization, i.e. in the possible decrease of the number of its states with the increase of the probability-discriminating parameter δ.

Životopis autora

Robert Logožar rođen je u Varaždinu 1962. godine. Maturirao je na Varaždinskoj gimnaziji kao najbolji maturant svoje generacije. Sudjelovao je i osvajao prva mjesta na više državnih natjecanja iz astronomije i fizike. Diplomirao je 1987. godine na Prirodoslovno matematičkom fakultetu Sveučilišta u Zagrebu s prosjekom ocjena od 4.75 i s odličnom ocjenom diplomskog rada. Objavio je rad iz područja atomske fizike u prestižnom fizikalnom časopisu Physical Review A. Radio je kraće vrijeme na Institutu za fiziku i Opservatoriju Hvar na Geodetskom fakultetu Sveučilišta u Zagrebu. 1988. godine pokreće privatan posao, a od 1990. radi kao srednjoškolski profesor fizike i računarstva u Varaždinu. 1994. godine upisuje poslijediplomski studij računarstva na Fakultetu elektrotehnike i računarstva. 1995. primljen je u svojstvu asistenta na Fakultetu Organizacije i informatike, gdje i sada radi na održavanju vježbi iz kolegija Arhitektura računala i Teorija informacija. Za oba je kolegija napisao interne skripte te povećao opseg i razinu problemskih zadataka.

(*Osobna stranica na Internetu:* http://www.foi.hr/~rlogozar/robertl.html.)

Popis radova

1. R. Logozar, R. Beuc, M. Movre, "van der Waals and resonance interaction in quasimolecular system Eu-Sr," *Physical Review A*, vol. 38(8), pp. 3969–3983, 1988.

2. R. Logozar, "Quasistatic absorption coefficient of two-component gases," *Hvar Observatory Bulletin*, vol. 10(1), pp. 23-33, 1986.