**Računalni model fonološke evolucije jezika**

**Autor: Teo Samaržija**

**Sažetak*:*** *Kako će izgledati naš jezik u budućnosti? U velikoj mjeri, na to je pitanje nemoguće odgovoriti. U rječnik našeg jezika ulazit će, i izlaziti iz njega (Koliko mladih ljudi danas zna što je to* disketa*, a kamoli* fijaker*?), riječi vezane za tehnologiju, čiji je razvoj nemoguće predvidjeti. Također će ulaziti riječi iz jezika koji će služiti internacionalnoj komunikaciji, što ovisi o politici, koju je isto nemoguće dugoročno predvidjeti (Da si nekome u Rimskom Carstvu rekao da će germanski jezik jednom biti internacionalni jezik, i da će skoro svi jezici svijeta imati riječi posuđene iz njega, gledali bi te u nevjerici.). Morfologija i sintaksa slijede neke znanstvene zakonitosti (analitički jezici evoluiraju u aglutinativne, aglutinativni u fuzionalne[[1]](#footnote-2), a fuzionalni u analitičke), ali te su zakonitosti komplicirane za računalno modelirati i vjerojatno pune izuzetaka (Armenski jezik, primjerice, je aglutinativan, razvio se iz fuzionalnog indoeuropskog prajezika, a nema dokaza da jer ikada bio analitičan.). Na morfologiju i sintaksu isto tako vjerojatno donekle utječe politika (Čini se da jezici koje duže vrijeme uče mnogi odrasli ljudi, kao što su engleski i kasni latinski, gube morfologiju i dobivaju kompliciraniju sintaksu. Isto tako, neke sintaksne strukture koje su se u novije vrijeme pojavile u hrvatskom jeziku vjerojatno su utjecaj engleskog jezika.). No, je li moguće predvidjeti kako će se razvijati fonologija jezika? Ovdje sam istražio upravo to, pokušao sam napraviti računalni model fonološke evolucije jezika. Iako u tome nisam bio pretjerano uspješan, moj rad može pomoći drugima, makar da ne ponavljaju moje greške, jer, koliko znam, nitko dosad nije pokušao to napraviti[[2]](#footnote-3). Također sam istražio mogu li računalni modeli biti korisni u etimologiji, te, povezano s time, koliko na entropije ljudskih jezika utječu razni dijelovi gramatike.*

**Abstract*:*** *What will the language we speak now look like in the future? To a large extent, that question is impossible to answer. The vocabulary of our language will gain, but also forget (How many young people today know what a* floppy disk *is, yet alone what a* fiacre *is?), words related to technology, the development of which is impossible to predict. The vocabulary of our language will also receive words from languages that will be used in international communication, which is dependent on politics, and it is also impossible to predict long-term (If you told somebody in Roman Empire that, one day, a Germanic language will be a global language, and that most of the languages all over the world will have loan-words from it, they will look at you oddly.). Morphology and syntax follow some scientific laws (analytic languages evolve into agglutinative ones, agglutinative ones evolve into fusional languages, and fusional languages evolve into analytic ones.), but those laws are difficult to model computationally and probably full of exceptions (Armenian language, for example, is an agglutinative language that evolved from the fusional Indo-European proto-language, but there is no reason to think there was a time when it was an analytic language.). Morphology and syntax are also probably somewhat influenced by politics (It seems as though languages with many adult learners, such as English or Late Latin, tend to have simpler morphology but more complicated syntax. Similarly, some syntactic structures that recently appeared in the Croatian language are probably an influence of the English language.). Nevertheless, is it possible to predict how the phonology of a language will develop? Here, I have researched exactly that, I have tried to make a computer model of the phonological evolution of languages. Although I was not particularly successful at that, I believe my work can come helpful to others, at least not to repeat the mistakes I have made, because, as far as I know, nobody has done anything like that. I have also researched whether computer models can be used in etymology, and, related to that, what effect do different parts of the grammar have on the entropy of human languages.*

**Etymology Game**

*Etymology Game* je flash-card igrica na temu povijesne fonologije koju sam napravio dok sam išao u opću gimnaziju, uz pomoć Daniela Rossa (lingvist koji proučava sintaksu američkih indijanskih jezika i moderator internetskog foruma linguistforum.com) i Borisa Muminovića (programera koji dobro poznaje JavaScript). Dostupna je na mojem blogu[[3]](#footnote-4). Rađena je u Safariju 6.2.8 te koristi radni okvir JQuery i manipulaciju SVG-a iz JavaScripta za grafiku. S obzirom na to da neki internetski preglednici to ne podržavaju, ali svejedno podržavaju dovoljno JavaScripta da se zadaci stvore i ispišu na ekran bez grafičkih dodataka (Internet Explorer 6, NetSurf...), postoji i tekstna verzija, do koje se može doći linkom. Grafička verzije igre ima 2500 redaka.

Etymology Game pokušava generirati riječi kao da se one nalaze u dva srodna jezika. Zatim stvara tri vrste zadataka. U prvom dijelu igre cilj je pogoditi koja je riječ iz prvog jezika srodna kojoj od četiri ponuđene riječi iz drugog jezika. Taj dio igre ima 26 različitih zadataka (koliko slova ima abeceda simuliranog prajezika), te, ako je moguće, igra čini to da svaka od zadanih riječi iz prvog jezika počinje različitim slovom. Ideja je da bude što manja vjerojatnost da se u drugom dijelu pojavi riječ s glasovnom promjenom koja se prije nije pojavila, pa da se tamo mora ići na pogađanje. U drugom dijelu igre dobiva se 8 parova srodnih riječi, a samo je za prvi par rečeno koja riječ pripada kojem od ta dva simulirana jezika, i cilj je pogoditi koja riječ iz ostalih parova pripada kojem jeziku. Ideju za to dobio sam iz zadatka *Tocharian* iz Međunarodne lingvističke olimpijade 2003. U trećem dijelu igre cilj je pogoditi koji simbolički opis glasovnih promjena odgovara kojem od simuliranih jezika.

Kad sam e-mailom pitao nekoliko lingvista što misle o toj mojoj igrici, odgovori su bili različiti. Mate Kapović odgovorio mi je da misli da je problem s tom igricom što vjerojatno igračima daje krivi uvid u to čime se lingvisti zapravo bave, da se lingvisti bave time da iz riječi potomaka pokušavaju rekonstruirati riječi pretke, a ne obrnuto. Dubravka Ivšić je komentirala da je prvi dio, za njen ukus, predugačak, da bi bilo dobro da napravim da se drugi dio uvijek može riješiti s podacima koji su u njemu zadani, te da se riječi njoj često doimaju lažnima, iako ne može objasniti zašto. Nema očitog načina da se osigura da taj drugi dio uvijek bude takav da se može riješiti. Bio sam zadao da prvi par riječi u njemu bude onaj koji je u simulaciji prošao najviše glasovnih promjena, no to, očito, nije dovoljno, i, kao što je primijetila i Dubravka Ivšić, ako igrate igricu nekoliko puta, vjerojatno ćete naletjeti na drugi dio u kojem morate ići na pogađanje. Model fonološke evolucije koji sam programirao u tu igricu sastoji se od pet dijelova:

1. generiranje prajezika
2. primjena glasovnih promjena
3. dio koji pokušava ostvariti fonološku stabilnost
4. fonotaktika jezika potomaka
5. redukcija vokala u jezicima potomcima

**Generiranje prajezika**

Prvi dio je generiranje prajezika, koji koristi restriktivnu fonotaktiku, to jest, on dopušta sve riječi osim onih koje odgovaraju određenom uzorku. Zabranio sam određene glasovne skupine na početku i na kraju riječi koje su mi upadale u oči kao iznimno neizgovorljive. Evo kako to izgleda u JavaScriptu, iz datoteke etymologist.html (sintaksno bojanje napravio VIM 8.2):

578 function syllable() {

579 var ret = vowels.charAt(Math.floor(Math.random() \* vowels.length));

580 **if** (Math.random() < 0.5) {

581 ret += consonants.charAt(

582 Math.floor(Math.random() \* consonants.length)

583 );

584 **if** (Math.random() > 0.75) {

585 var znak = consonants.charAt(

586 Math.floor(Math.random() \* consonants.length)

587 );

588 **while** (

589 znak === "w" ||

590 znak === "y" || //suglasnik+y i suglasnik+w na kraju rijeci tesko se izgovaraju...

591 znak === "j" ||

592 ret.charAt(ret.length - 1) == znak || //...kao i dugi suglasnici na kraju rijeci.

593 (znak === "p" && ret.charAt(ret.length - 1) === "b") ||

594 (znak === "t" && ret.charAt(ret.length - 1) === "d") ||

595 (znak === "b" && ret.charAt(ret.length - 1) === "p") ||

596 (znak === "d" && ret.charAt(ret.length - 1) === "t") ||

597 (znak === "s" && ret.charAt(ret.length - 1) === "z") ||

598 (znak === "z" && ret.charAt(ret.length - 1) === "s")

599 )

600 znak = consonants.charAt(

601 Math.floor(Math.random() \* consonants.length)

602 );

603 ret += znak;

604 }

605 }

606 **if** (Math.random() < 0.75) {

607 ret =

608 consonants.charAt(Math.floor(Math.random() \* consonants.length)) +

609 ret;

610 **if** (Math.random() > 0.75) {

611 var znak = consonants.charAt(

612 Math.floor(Math.random() \* consonants.length)

613 );

614 **while** (

615 znak === "w" ||

616 znak === "y" ||

617 znak === "j" ||

618 znak === "f" ||

619 ((znak === "v" || znak === "h") &&

620 ret.substr(0, 1) !== "l" &&

621 ret.substr(0, 1) !== "r") ||

622 znak === ret.charAt(0) || //Dugi suglasnik ne moze biti na pocetku rijeci.

623 znak === "l" ||

624 znak === "r" ||

625 (znak === "m" &&

626 ret.substr(0, 1) !== "b" &&

627 ret.substr(0, 1) !== "p") || //Malo lakse za izgovoriti, ali svejedno tesko.

628 (znak === "p" && ret.substr(0, 1) === "b") ||

629 (znak === "t" && ret.substr(0, 1) === "d") ||

630 (znak === "b" && ret.substr(0, 1) === "p") ||

631 (znak === "d" && ret.substr(0, 1) === "t") ||

632 (znak === "s" && ret.substr(0, 1) === "z") ||

633 (znak === "z" && ret.substr(0, 1) === "s")

634 )

635 znak = consonants.charAt(

636 Math.floor(Math.random() \* consonants.length)

637 );

638 ret = znak + ret;

639 }

640 }

641 **return** ret;

642 }

643

644 function randomWord() {

645 var ret = syllable();

646 **if** (Math.random() > 0.9) {

647 ret += syllable();

648 **if** (Math.random() > 0.9) ret += syllable();

649 }

650 **if** (ret.length < 12) **return** ret;

651 **return** randomWord();

652 }

653

654 var original = [],

655 brojPrimjenjenihPromjena = [];

656 **for** (var i = 0; i < 1024; i++) {

657 var rijec;

658 **do** {

659 rijec = randomWord();

660 } **while** (original.indexOf(rijec) !== -1);

661 original.push(rijec);

662 brojPrimjenjenihPromjena.push(0);

663 }

Daje li to realistične rezultate? Pretpostavljam da ne previše, i da je jedan od razloga zašto se riječi ne doimaju realistične to što će se u stvarnim jezicima skupine suglasnika na početku ili na kraju riječi ponavljati znatno češće nego što bi to bilo da je slučajno, dok ovdje neće. No, nisam napravio nikakva mjerenja da tu hipotezu provjerim, niti imam preciznu ideju kako bih to provjerio.

**Primjena glasovnih promjena**

Za glasovne promjene, koriste se podaci o tome koji će glasovi najvjerojatnije prijeći u koje. Kako nisam mogao pronaći relevantne podatke, izmislio sam neke na temelju anegdota koje znam iz povijesne lingvistike. Unio sam ih ovako:

670 promjene[0] = [

671 "e",

672 "e",

673 "ei",

674 "ia",

675 "o",

676 "ua",

677 "i",

678 "u",

679 "ar",

680 "er",

681 "ia",

682 "-"

683 ]; //U koje ce se glasove najvjerojatnije promijeniti 'a'.

I tako za svako slovo abecede. To, naravno, nije kako se jezici zapravo ponašaju, jer bi po tome glasovna promjena da *k* prelazi u *c* ispred *a* bila jednako vjerojatna kao promjena da *k* prelazi u *c* ispred *i*, a očito nije tako. Prelazak *k* u *c* ispred *i* dogodila se i u španjolskom i u hrvatskom, a slične promjene u mnogim drugim jezicima, dok se promjena da *k* prelazi u *c* ispred *a*, koliko je meni poznato, nigdje nije dogodila (međutim, u francuskom se dogodila pomalo neobična promjena da *k* ispred *a* prelazi u *š*, i tako je u hrvatski od latinske riječi za koncert, *cantatio*, došla, preko francuskog *chanson*, riječ *šansona*). Isto tako, poznato je Ruki pravilo, koje kaže da *s* ima tendenciju prijeći u *h* nakon glasova *r*, *u*, *k* i *i* (no primijetite da je u grčkom *s* prelazilo u *h*, ali ne po Ruki pravilu, nego na početku riječi, i tako latinski *s* odgovara grčkome *h* na početku riječi: *sex-hex* (oboje znači *šest*), *septem-hepta* (oboje znači *sedam*), *sal-hals* (oboje znači *sol*), *sol-helios* (oboje znači *sunce*), *sus-hys* (oboje znači *svinja*), *super-hyper* (oboje znači *iznad*), *similis-homoios* (oboje znači *slično*), *sollus-holos* (oboje znači *cijelo*), *silva-hyle* (oboje znači *drva*), *somnium-hypnos* (oboje znači *san*), *sanguis-haima* (oboje znači *krv*)...).Pa, ipak, mislim da to neće znatno utjecati na rezultat, jer su korijeni uglavnom jednosložni. Uostalom, čak i da ispravimo to da algoritam smatra da je vjerojatnost promjene *k* u *c* ispred *a* jednaka kao vjerojatnost da se to dogodi ispred *i*, svejedno velik broj glasovnih zakonitosti koje vrijede u jezicima ne bismo mogli simulirati. Vernerov zakon, jedan od najranije formuliranih zakona povijesne fonologije, recimo, kaže da bezvučni suglasnici odmah nakon naglašenog samoglasnika imaju tendenciju postati zvučni. Recimo, hrvatski će nazivi brojeva *sedam* i *osam*, nakon nekoliko stoljeća, biti nešto kao *sedam* i *ozam*, jer se *s* u *sedam* nalazi prije naglašenog samoglasnika, a *s* u *osam* nalazi se odmah nakon njega. Tim se zakonom objašnjavaju nepodudarnosti kao takve da latinska riječ *pa****t****er* (otac) odgovara njemačkom *Va****t****er*, dok latinski *fra****t****er* (brat) odgovara njemačkom *Bru****d****er*: u tim je riječima u indoeuropskom prajeziku, kao i na sanskrtu, bio različit naglasak.Kako ovaj model ne uključuje naglaske, to ne možemo simulirati. Isto tako, nekad kako će se promijeniti neki glas ne ovisi o glasovima odmah nakon njega, nego negdje dalje. Vjerojatno najpoznatiji zakon koji takvu promjenu opisuje zove se Havlikov zakon. On kaže da se poluglasi koji su parni po redu, kada se broji od kraja riječi ili od idućeg samoglasnika, imaju tendenciju prijeći u samoglasnik, dok oni koji su neparni po redu imaju tendenciju nestati. Recimo, riječ *overload* (preopterećenje) na američkom se engleskom izgovara /ˌəʊ.vəˈləʊd/[[4]](#footnote-5). Onaj posljednji poluglas za nekoliko će stoljeća nestati, jer je on prvi od samoglasnika. Međutim, onaj poluglas nakon *v* prijeći će u neki samoglasnik, jer je on drugi po redu od samoglasnika iza njeg. Nestat će, naravno, i onaj poluglas na početku. Dakle, za nekoliko stoljeća, *overload* će se u američkom engleskom izgovarati nešto kao /ˌʊ.vɑːˈlʊd/. Slično, *p* ima tendenciju prijeći u *kw*ako se dalje u istoj riječi nalazi *kw.* Recimo, indoeuropski *\*penkwe* (broj pet) prešao je u latinski *quinque*, dok je indoeuropska riječ za hrast, odakle dolazi i ime slavenskog boga šume *Perun*, prešla je u latinski *quercus*, a isto se dogodilo i u nekim drugim indoeuropskim jezicima. Bi li pokušaj da se simuliraju takve promjene učinilo da rezultati simulacije više odgovaraju stvarnosti? Smatram da ne, zbog pojave u društvenim znanostima koju Bryan Caplan *Law of Diminishing Returns*. On kaže da što više truda ulažeš u nešto vezano za društvene znanosti, omjer dobivene vrijednosti i truda koji si uložio postaje sve manji.

Izreći Havlikov zakon na JavaScriptu nije problem (zapravo, moram priznati da je lakše nego na AEC-u, programskom jeziku koji sam napravio[[5]](#footnote-6)), to možemo ovako:

1 // Zaseban JavaScriptski program koji opisuje Havlikov zakon na JavaScriptu.

2 **const** samoglasnici = "aeiou";

3 let rijec = "əuvələud";

4 let u\_koji\_samoglasnik\_pretvaramo\_jake\_poluglase =

5 samoglasnici[Math.floor(Math.random() \* samoglasnici.length)];

6 let indeks\_poluglasa = 1;

7 **for** (let i = rijec.length; i >= 0; i--) {

8 **if** (samoglasnici.indexOf(rijec[i]) !== -1) indeks\_poluglasa = 1;

9 **if** (rijec[i] === "ə") {

10 **if** (indeks\_poluglasa % 2 === 1)

11 // Slabi poluglas

12 rijec = rijec.substr(0, i) + rijec.substr(i + 1);

13 **else** // Jaki poluglas

14 rijec =

15 rijec.substr(0, i) +

16 u\_koji\_samoglasnik\_pretvaramo\_jake\_poluglase +

17 rijec.substr(i + 1);

18 indeks\_poluglasa++;

19 }

20 }

21 console.log(rijec);

Za usporedbu, evo kako bi se Havlikov zakon mogao prevesti na programski jezik AEC:

1 **/\* Ovo je primjer kako bi se Havlikov zakon mogao zapisati na programskom**

2 **\* jeziku AEC, kojeg sam ja dizajnirao i napravio sam compiler za njega koji**

3 **\* ga prevodi u WebAssembly (standardizirani bytecode od JavaScripta).**

4 **\*/**

5

6 **// Uvezimo funkcije iz JavaScripta koje će nam trebati.**

7 **Function** random**()** **Which** **Returns** **Decimal32** **Is** **External;**

8 **Function** printString**(CharacterPointer** string**)**

9 **Which** **Returns** **Nothing** **Is** **External;**

10 **Function** printInteger**(Integer32** integer**)**

11 **Which** **Returns** **Nothing** **Is** **External;**

12

13 **// Napravimo neke deklaracije unaprijed, da kod bude uredniji.**

14 **Function** duljina\_stringa**(CharacterPointer** string**)** **Which**

15 **Returns** **Integer32** **Is** **Declared;**

16 **Function** izbrisi\_znak\_iz\_stringa**(CharacterPointer** mjesto\_u\_stringu**)** **Which**

17 **Returns** **Nothing** **Is** **Declared;**

18 **Function** kopiraj\_string**(CharacterPointer** cilj**,** **CharacterPointer** izvor**)**

19 **Which** **Returns** **Nothing** **Is** **Declared;**

20 **Function** pronadi\_znak\_u\_stringu**(CharacterPointer** string**,** **Character** znak**)**

21 **Which** **Returns** **Integer32** **Is** **Declared;**

22

23 **Integer32** ispisujemo\_li\_poruke\_za\_debugiranje **:=** **0;**

24

25 **// S obzirom na to da moj programski jezik (još) ne podržava Unicode,**

26 **// poluglase ćemo označavati nekim ASCII znakovima.**

27 **Function** je\_li\_poluglas**(Character** znak**)** **Which** **Returns** **Integer32** **Does**

28 **// Moj programski jezik ne razlikuje Boolean (true/false)**

29 **// vrijednosti od 32-bitnih cjelobrojnih vrijednosti, jer ne**

30 **// razlikuje ih ni WebAssembly. To što ih JavaScript razlikuje**

31 **// samo je fasada koju stvaraju compileri za JavaScript, slično**

32 **// kao privatna svojstva u C++-u.**

33 **Return** znak **=** 'I' **or** znak **=** 'U'**;**

34 **EndFunction**

35

36 **// Napišimo sada funkciju koju će pozvati JavaScript kad sve inicijalizira.**

37 **// Moj compiler (još) ne radi programe koji se mogu samostalno pokrenuti u**

38 **// JavaScript engineu koji podržava WebAssembly, nego programi koje moj**

39 **// compiler prevodi u WebAssembly zahtijevaju da JavaScript inicijalizira**

40 **// neke stvari prije no što se oni mogu pokrenuti.**

41 **Function** glavni\_program**()** **Which** **Returns** **Nothing** **Does**

42 **CharacterPointer** samoglasnici **:=** "aeiou"**;**

43 **Character** u\_sto\_se\_pretvaraju\_jaki\_poluglasi **:=**

44 **ValueAt(**

45 samoglasnici **+**

46 random**()** **\*** duljina\_stringa**(**samoglasnici**)**

47 **);**

48 **Character** rijec**[12];**

49 **// U C-u i C++-u nizovi se u većini slučajeva mogu koristiti kao**

50 **// pokazivači, ali u nekim slučajevima, koje određuju neka iznimno**

51 **// komplicirana pravila, ne mogu. To me je živciralo, pa sam odlučio**

52 **// u svom programskom jeziku uvesti pravilo da se niz nikada ne**

53 **// može koristiti kao pokazivač i obrnuto. No, možemo deklarirati**

54 **// niz i pokazivač koji se isto zovu, ovako:**

55 **CharacterPointer** rijec **:=** **AddressOf(**rijec**[0]);**

56 kopiraj\_string**(**rijec**,** "UuvUlUud"**);**

57 **Integer32** indeks\_poluglasa **:=** **1,**

58 i **:=** duljina\_stringa**(**rijec**),**

59 pocetna\_duljina\_stringa **:=** i**;**

60 **While** i **>** **0** **or** i **=** **0** **Loop**

61 **// U svoj programski jezik još nisam ugradio for-petlju.**

62 **If** ispisujemo\_li\_poruke\_za\_debugiranje **Then**

63 printString**(**"U koraku "**);**

64 printInteger**(**pocetna\_duljina\_stringa **-** i **+** **1);**

65 printString**(**", rijec izgleda ovako: "**);**

66 printString**(**rijec**);**

67 printString**(**".**\*n***"**);**

68 **EndIf**

69 **If** **not(**

70 pronadi\_znak\_u\_stringu**(**samoglasnici**,** rijec**[**i**])** **=** **-1**

71 **)** **Then**

72 indeks\_poluglasa **:=** **1;**

73 **ElseIf** je\_li\_poluglas**(**rijec**[**i**])** **Then**

74 **If** **mod(**indeks\_poluglasa**,** **2)** **=** **1** **Then**

75 **// Slabi poluglas**

76 izbrisi\_znak\_iz\_stringa**(**rijec **+** i**);**

77 **Else**

78 **// Jaki poluglas**

79 rijec**[**i**]** **:=** u\_sto\_se\_pretvaraju\_jaki\_poluglasi**;**

80 **EndIf**

81 indeks\_poluglasa **+=** **1;**

82 **EndIf**

83 i **-=** **1;**

84 **EndWhile**

85 **If** ispisujemo\_li\_poruke\_za\_debugiranje **Then**

86 printString**(**"Nakon primjene Havlikovog zakona, rijec izgleda ovako: "**);**

87 **EndIf**

88 printString**(**rijec**);**

89 **If** ispisujemo\_li\_poruke\_za\_debugiranje **Then**

90 printString**(**"**\*n***"**);**

91 **EndIf**

92 **EndFunction**

93

94 **// Napišimo sada pomoćne funkcije koje smo deklarirali unaprijed...**

95 **Function** duljina\_stringa**(CharacterPointer** string**)**

96 **Which** **Returns** **Integer32** **Does**

97 **Return** **ValueAt(**string**)** **=** **0** **?**

98 **0** **:**

99 **1** **+** duljina\_stringa**(**string **+** **1);**

100 **EndFunction**

101

102 **Function** kopiraj\_string**(CharacterPointer** cilj**,** **CharacterPointer** izvor**)**

103 **Which** **Returns** **Nothing** **Does**

104 **While** **ValueAt(**izvor**)** **Loop**

105 **ValueAt(**cilj**)** **:=** **ValueAt(**izvor**);**

106 cilj **+=** **1;**

107 izvor **+=** **1;**

108 **EndWhile**

109 **ValueAt(**cilj**)** **:=** **0;**

110 **EndFunction**

111

112 **Function** pronadi\_znak\_u\_stringu**(CharacterPointer** string**,** **Character** znak**)**

113 **Which** **Returns** **Integer32** **Does**

114 **Integer32** i **:=** **0;**

115 **While** **ValueAt(**string **+** i**)** **Loop**

116 **If** **ValueAt(**string **+** i**)** **=** znak **Then**

117 **Return** i**;**

118 **EndIf**

119 i **+=** **1;**

120 **EndWhile**

121 **Return** **-1;**

122 **EndFunction**

123

124 **Function** izbrisi\_znak\_iz\_stringa**(CharacterPointer** mjesto\_u\_stringu**)**

125 **Which** **Returns** **Nothing** **Does**

126 **While** **ValueAt(**mjesto\_u\_stringu**)** **Loop**

127 **ValueAt(**mjesto\_u\_stringu**)** **:=**

128 **ValueAt(**

129 mjesto\_u\_stringu **+** **1**

130 **);**

131 mjesto\_u\_stringu **+=** **1;**

132 **EndWhile**

133 **EndFunction**

O prednostima i manama pisanja programa u svojim programskim jezicima moglo bi se mnogo pisati. Iz iskustva znamo da je jedan od većih uzroka grešaka u programima krivo razumijevanje programskih jezika. Takvim je greškama iznimno teško ući u trag, jer čak ni *rubber duck debugging* (tehnika uklanjanja grešaka u programima tako da zamislimo da nekome u detalje, na ljudskom jeziku, objašnjavamo kako bi program trebao funkcionirati) onda ne pomaže. U JavaScriptu, recimo, postoje pravila po kojima parser pokušava sam umetnuti znak za kraj naredbe ; (točka-zarez) da bi ispravio greške u sintaksi, ali ponekad ta pravila dovode do toga da ispravni programi postanu besmisleni. Ili, recimo, da naredba Array.sort po defaultu niz brojeva neće razvrstati po veličini, već abecedno. Isto tako, radi kompatibilnosti s arhajskim internetskim preglednicima (da bi se JavaScriptski programi koji su se vrtjeli u njima vrtjeli i u današnjim), vrijede nelogična pravila da typeof null vraćaobject (netko tko zna samo malo JavaScripta očekivao bi da vraća undefined) i typeof document.all vraća undefined (umjesto object, kako bi vjerojatno svatko tko za taj detalj ne zna očekivao).Pisanje programa u jeziku koji smo sami izradili očito eliminira pogreške koje nastaju zbog takvih stvari. Neki koji se bave programiranjem za web kažu da se taj problem može eliminirati tako da se koristi Emscripten, što je compiler koji prevodi C++ u JavaScriptov bytecode. Po meni, to će samo pogoršati stvar. Makar C++ bio bolji jezik nego JavaScript, on svejedno ima neke svoje osobitosti, koje lako mogu postati izvor greške u programu. I te osobitosti C++-a nisu ni približno toliko opće poznate koliko su osobitosti JavaScripta, što čini stvar gorom. Izgubio sam hrpu vremena dok sam shvatio da se C++-ov konstruktor kopije koji compiler ugrađuje u klase ne ponaša onako kao što sam ja bio uvjeren da se ponaša (bio sam uvjeren da je moguće objektu pridružiti objekt čiji se pokazivač nalazi u samom tom objektu, a zapravo mi je pokušaj da to napravim ponekad korumpirao sintaksno stablo u compileru za moj programski jezik), da operator uglatih zagrada na vektorima u pravilu neće javiti nikakvu pogrešku ako pokušamo pomoću njega pristupiti elementu izvan granica (bio sam uvjeren da će to rezultirati ili bacanjem iznimke ili *segmentation fault*om, a zapravo se dogodilo to da mi je program radio ispravno na svim operacijskim sustavima na kojima sam ga isprobao osim na 32-bitnom FreeBSD-u), da shvatim što to compiler za C++ misli kad me upozorava da mi klasa ima virtualne metode bez virtualnog konstruktora (to dovodi do curenja memorije), i tako dalje. Dakle, ja, na temelju svog iskustva, mislim da je problem krivog razumijevanja jezika u kojem pišemo programe ozbiljan problem koji se može izbjeći jedino programiranjem u programskom jeziku koji smo sami napravili. Pravljenje compilera za svoj programski jezik također je dobar način da se upoznate s okruženjem u kojem će se programi koje pišete pokretati, pa ćete bolje razumjeti poruke o pogreškama. Neki bi još tvrdili da je prednost programiranja u svom programskom jeziku to što tada možemo bolje prilagoditi program svom načinu razmišljanja, no nisam siguran što bi to značilo. Jedan od nedostataka je, kao što vidite iz onog primjera gore, to što je teško da ćete napraviti programski jezik bolji od onog koji su radili stručnjaci. Da se napiše Havlikov zakon na mom programskom jeziku, treba višestruko duži kod nego da ga se napiše na JavaScriptu, i veća je vjerojatnost da ćemo napraviti neku sitnu grešku zbog koje program neće raditi (kao što se i meni dogodilo). Napraviti već i ovakav programski jezik bio je naporan posao: compiler za moj programski jezik, pisan u C++-u, ima 5500 redaka. Također, naravno, ako pišemo programe u svom programskom jeziku, teže ćemo koristiti radne okvire (*frameworks*) pisane u drugim programskim jezicima, to jest, programi koje pišemo bit će manje kompatibilni. I, naravno, programi, koji su pisani u programskom jeziku za kojeg nijedan debugger (programi za dijagnozu i uklanjanje pogrešaka u drugim programima) nema podršku, debugiraju se na stari način: sistematskim ispisivanjem među-rezultata i debugiranjem asemblerskog koda.

Dakle, prevesti Havlikov zakon na JavaScript nije problem. No, problem je, i pretpostavljam da nije vrijedno truda, izmijeniti moj model tako da se to može ubaciti u njega.

Dakle, funkcija za primjenu glasovne promjene u mom modelu izgleda ovako:

1045 function primjeniPromjenu(nakon, prvi, prije, drugi, rijec) {

1046 **for** (var i = 0; i < rijec.length; i++) {

1047 **if** (

1048 (nakon !== "0" || i === 0) && //Glasovna promjena zahvaca samo glasove na pocetku rijeci.

1049 (prije !== "0" || i === rijec.length - 1) && //Samo na kraju rijeci.

1050 (nakon !== "N" ||

1051 rijec.charAt(i - 1) === "m" ||

1052 rijec.charAt(i - 1) === "n") && //Nakon nazalnog suglasnika.

1053 (prije !== "N" ||

1054 rijec.charAt(i + 1) === "m" ||

1055 rijec.charAt(i + 1) === "n") && //Prije nazalnog suglasnika.

1056 (nakon !== "V" || vowels.search(rijec.charAt(i - 1)) !== -1) && //Nakon samoglasnika

1057 (prije !== "V" || vowels.search(rijec.charAt(i + 1)) !== -1) && //Prije samoglasnika

1058 (prije !== "C" || consonants.search(rijec.charAt(i + 1)) !== -1) && //Prije suglasnika

1059 (nakon !== "C" || consonants.search(rijec.charAt(i - 1)) !== -1) && //Nakon suglasnika

1060 (nakon === rijec.charAt(i - 1) ||

1061 nakon === "Z" ||

1062 nakon === "0" ||

1063 nakon === "C" ||

1064 nakon === "N" ||

1065 nakon === "V") && //Rijetke glasovne promjene koje se dogadaju samo nakon nekog odredenog glasa

1066 (prije === rijec.charAt(i + 1) ||

1067 prije === "Z" ||

1068 prije === "0" ||

1069 prije === "C" ||

1070 prije === "N" ||

1071 prije === "V") && //One koje se dogadaju samo prije tog glasa

1072 rijec.charAt(i) === prvi

1073 ) {

1074 var novi = rijec.slice(0, i);

1075 **if** (drugi !== "0") novi += drugi;

1076 novi += rijec.slice(i + 1);

1077 **if** (drugi === "0") i--;

1078 rijec = novi;

1079 i += drugi.length - 1;

1080 }

1081 }

1082 **return** rijec;

1083 }

Dalje algoritam pretpostavlja da je, recimo, 0.5\*0.25=12.5% glasovnih promjena koje se događaju u jezicima bezuvjetno, kao Grimmov zakon, dakle, da ne ovise niti o onome što dolazi nakon tog glasa niti o onome što dolazi prije tog glasa u riječi. Grimmov zakon, prvi imenovani zakon povijesne fonologije, kaže da, ako se zanemare drugi faktori, aspirirani zvučni suglasnici (*bh* kao u ***Bh****agavad Gita*, *dh* kao u ***dh****arma, gh*) sami po sebi imaju tendenciju prelaziti u neaspirirane zvučne suglasnike (*b*, *d*, *g*), neaspirirani zvučni suglasnici sami po sebi imaju tendenciju prelaziti u neaspirirane bezvučne suglasnike (*p*, *t*, *k*), dok neaspirirani bezvučni suglasnici sami po sebi imaju tendenciju prelaziti u bezvučne frikative (*f*, *th*, *h*).

1043 var uvjeti = "VCN0-VC0";

1085 var BrojSimuliranihPromjena = 11;

1086 **for** (var i = 0; i < BrojSimuliranihPromjena \* 2; i++) {

1087 var prije;

1088 **if** (Math.random() < 0.5) prije = "Z";

1089 //Bezuvjetna glasovna promjena.

1090 **else** prije = uvjeti.charAt(Math.floor(Math.random() \* uvjeti.length));

1091 **if** (prije === "-")

1092 prije = abeceda.charAt(Math.floor(Math.random() \* abeceda.length));

1093 var nakon;

1094 **if** (

1095 (prije != "Z" && Math.random() < 0.75) ||

1096 (prije == "Z" && Math.random() < 0.25)

1097 )

1098 nakon = "Z";

1099 **else** nakon = uvjeti.charAt(Math.floor(Math.random() \* uvjeti.length));

1100 **if** (nakon === "-")

1101 nakon = abeceda.charAt(Math.floor(Math.random() \* abeceda.length));

1102 var prvi = Math.floor(Math.random() \* 26);

1103 **if** (

1104 i >= BrojSimuliranihPromjena &&

1105 fonologija[1].indexOf(

1106 (nakon === "Z" ? "" : nakon) +

1107 "[" +

1108 prvi +

1109 "]" +

1110 (prije === "Z" ? "" : prije) +

1111 ">" +

1112 drugi

1113 ) + 1

1114 ) {

1115 //Ne daj da se iste promjene dogode u oba simulirana jezika

1116 i--;

1117 **continue**;

1118 }

Naredba da se ista promjena ne može dogoditi u oba jezika ovdje je samo da bi igrica bila zanimljivija, naravno da ne odgovara stvarnosti, štoviše, upravo suprotno.

**Fonološka ravnoteža**

Fonološka se ravnoteža u mom modelu simulira tako da se, ukoliko se primijeti da neki glas od onih 26 početnih u jeziku nedostaje, a glas koji smo odabrali za početni glas u glasovnoj promjeni može prijeći u njega (da je tako navedeno u dvodimenzionalnom polju promjene), on prijeđe u njega:

1119 var drugi;

1120 var jesmoLiOdabraliZamijenuZaPrvi = false;

1121 **for** (var j = 0; j < 12; j++) {

1122 var fonem = promjene[prvi][j];

1123 **if** (fonem == "-" || fonem == "0") **continue**;

1124 var nalaziLiSeFonemUJeziku = false;

1125 **if** (i < BrojSimuliranihPromjena) {

1126 **for** (var k = 0; k < language1.length; k++)

1127 **if** (language1[k].indexOf(fonem) + 1) {

1128 nalaziLiSeFonemUJeziku = true;

1129 **break**;

1130 }

1131 } **else** {

1132 **for** (var k = 0; k < language2.length; k++)

1133 **if** (language2[k].indexOf(fonem) + 1) {

1134 nalaziLiSeFonemUJeziku = true;

1135 **break**;

1136 }

1137 }

1138 **if** (nalaziLiSeFonemUJeziku) **continue**;

1139 jesmoLiOdabraliZamijenuZaPrvi = true;

1140 drugi = fonem;

1141 **break**;

1142 }

1143 **if** (!jesmoLiOdabraliZamijenuZaPrvi)

1144 drugi = promjene[prvi][Math.floor(Math.random() \* 12)];

1145 **if** (drugi === "-") {

1146 var jeLiFonologijaURavnotezi = true;

1147 **for** (var k = 0; k <= abeceda.length; k++) {

1148 var fonem = abeceda.charAt(k);

1149 var nalaziLiSeFonemUJeziku = false;

1150 **if** (i < BrojSimuliranihPromjena) {

1151 **for** (var j = 0; j < language1.length; j++)

1152 **if** (language1[j].indexOf(fonem) + 1) {

1153 nalaziLiSeFonemUJeziku = true;

1154 **break**;

1155 }

1156 } **else** {

1157 **for** (var j = 0; j < language2.length; j++)

1158 **if** (language2[j].indexOf(fonem) + 1) {

1159 nalaziLiSeFonemUJeziku = true;

1160 **break**;

1161 }

1162 }

1163 **if** (nalaziLiSeFonemUJeziku) **continue**;

1164 jeLiFonologijaURavnotezi = false;

1165 drugi = fonem;

1166 **break**;

1167 }

1168 **if** (jeLiFonologijaURavnotezi)

1169 drugi = abeceda.charAt(Math.floor(Math.random() \* abeceda.length));

1170 }

Taj algoritam pretpostavlja da je, recimo, jezik koji nema *b*, ali ima *p*, jednako vjerojatan kao jezik koji ima *p*, a nema *b*. To, naravno, nije slučaj. Jezici koji imaju *b*, a nemaju *p* (kao što su većina dijalekata arapskoga jezika, somalski jezik te neki keltski jezici), daleko su češći nego jezici koji imaju *p*, ali ne i *b*. Zapravo, jedna od čestih rasprava u današnjoj lingvistici upravo je o tome koliko je današnja rekonstrukcija indoeuropskog prajezika tipološki neobična, jer u njemu postoji *p*, ali ne postoji *b*. O tome postoje različita mišljenja. Većina lingvista, među njima i Mate Kapović, misle da je pozivanje na lingvističku tipologiju prilikom rekonstruiranja prajezika zapravo kršenje znanstvene metode. Lingvisti Tamaz Gamkrelidze i Vyacheslav Ivanov misle da indoeuropski prajezik zapravo uopće nije imao zvučne praskavce, ni *b*, ni *d*, ni *g*, nego da je na mjestima gdje se oni rekonstruiraju imao glotalni praskavac nakon kojeg su slijedili *p*, *t* ili *k*, što manje odskače od fonologija današnjih jezika. Neki lingvisti, među njima je Paul Hopper, smatraju da postoji nezanemariva vjerojatnost da je Grimov zakon zapravo netočan, da se nije dogodilo da su se svi indoeuropski praskavci promijenili u germanskim, točarijanskim, anatolijskim jezicima i armenskom jeziku po Grimovom zakonu, nego da se dogodilo to da se u većini starih indoeuropskih jezika dogodio niz glasovnih promjena suprotan od onih što ih opisuje Grimov zakon. Dakle, da je indoeuropski prajezik možda zapravo zvučao sličnije gotskome ili armenskome nego sanskrtu, latinskom ili grčkom. Da je indoeuropski prajezik, kao današnji arapski, imao *f,* tamo gdje se obično rekonstruira *p*,i da je imao *b*, tamo gdje se obično rekonstruira *bh*, a da nije imao *p*.Rijetko koji se lingvist s time slaže, jer bi to značilo da su mnoge općeprihvaćene etimologije u germanskim jezicima pogrešne (recimo, da je engleska riječ za konoplju, *hemp*, posuđena iz nekog ugrofinskog jezika u germanski prajezik prije no što su se glasovne promjene koje opisuje Grimov zakon počele događati).Neovisno o tome svemu, takvo pojednostavljenje, pretpostavka da je jezik koji ima *p* a nema *b* jednako vjerojatan kao jezik koji ima *b* a nema *p*,znatno pojednostavljuje programiranje modela, a teško da će utjecati na rezultat.

**Fonotaktika jezika potomaka**

Dio modela za fonotaktiku jezika potomaka ponovno zahtijeva neke podatke koje nije lagano naći, pa sam ih izmislio. To su podaci s kojom skupinom suglasnika riječ može početi ili završiti:

493 var allowed\_beginning\_consonant\_clusters = [

494 "sq",

495 "st",

496 "ts",

497 "sk",

498 "ks",

499 "zd",

500 "dz",

501 "th",

502 "tr",

503 "dr",

570 var allowed\_ending\_consonant\_clusters = [];

571 **for** (var i = 0; i < allowed\_beginning\_consonant\_clusters.length; i++)

572 **if** (allowed\_beginning\_consonant\_clusters[i].substr(1, 1) !== "q")

573 allowed\_ending\_consonant\_clusters.push(

574 allowed\_beginning\_consonant\_clusters[i].substr(1, 1) +

575 allowed\_beginning\_consonant\_clusters[i].substr(0, 1)

576 );

O tome zapravo govori zakon sonornosti slogova, no, po meni je to jako gruba aproksimacija, toliko da izmišljeni podaci možda daju bolji rezultat. Zakon sonornosti slogova kaže da sonornost glasova raste do jezgre sloga, a zatim opada do kraja sloga. Najsonorniji glas (glas koji uzrokuje najviše podrhtavanja glasnica) je samoglasnik *a*, nakon njega slijede samoglasnici *e* i *o*, nakon čega slijede samoglasnici i polu-samoglasnici *i*, *j*, *w* i *u*, onda slijede suglasnici *r* i *l* (koji mogu biti samoglasni u mnogim jezicima, recimo, hrvatski *vrt* ili *bicikl*), zatim slijede nazalni suglasnici *m*, *n* i *ng* (*n* kao u *pisa****n****ka*), nakon toga slijede zvučni frikativi *v* i *z*, pa onda bezvučni frikativi *f*, *th* i *s*, pa onda zvučni praskavci *b*, *d* i *g*, te konačno bezvučni praskavci *p*, *t* i *k*. Međutim, očito ima i drugih faktora ovdje. Recimo, čini mi se iz iskustva da mnogo više jezika dopušta da riječ počinje sa *st* nego što dopušta da im riječ počinje sa *ts*. Zato sam napravio takve podatke. Naravno, ne dopušta svaki jezik iste suglasničke skupine na početku riječi. Latinski jezik je dopuštao da mu riječ počinje sa *st*, latinska riječ za zvijezdu bila je *stella* ili *sterola* (umanjenica od *stella*, *zvjezdica*), dok je riječ za stajati bila *stare*. U današnjem talijanskom, gramatika isto dopušta da riječ počinje na *st*, pa su talijanske riječi za *zvijezda* i *stajati* još uvijek *stella* i *stare*. No, španjolska i okcitanska gramatika to ne dopuštaju. Zato je španjolska riječ za zvijezdu *estrella,* a okcitanska je *estela*. I španjolska i okcitanska riječ za *stajati* je *estar*. Latinska riječ za *glup* bila je *stupidus*, odatle dolaze talijanska riječ *stupido* i španjolska riječ *estupido*.Latinska riječ za sloj ili plahtu bila je *stratus*, odatle dolaze talijanska riječ *strato* i španjolska riječ *estrato*.Isto tako, latinska i talijanska gramatika dopuštaju da riječ počinje na *sp*, ali španjolska ne dopušta. Zato su latinska i talijanska riječ za *disati* oboje *spirare*, dok španjolska riječ zaod istog korijena glasi *espirar* (ili, možda, s obzirom na to da španjolska riječ znači *izdahnuti*, to dolazi od latinskog *exspirare*, nisam to baš previše proučavao). Latinska i talijanska riječ za *nadati se* jest *sperare*, dok su španjolska i okcitanska riječ *esperar*. Latinska i talijanska riječ za pjenu jest *spuma*, dok je španjolska *espuma*. Latinska riječ za ogledalo bila je *speculum*, talijanska riječ za ogledalo je *specchio*, a španjolska je *espejo*. Latinski i talijanski dopuštaju da riječ počinje na *sk*, ali španjolski ne dopušta. Od latinske riječi za stepenicu, *scalae* (*pluralia tantum*, uvijek množina, kao hrvatska riječ *ljestve*), dolaze talijanska riječ za stepenicu, *scala*, i španjolska riječ za ljestve, *escala*.Dakle, prije zabranjene skupine suglasnika na početku riječi španjolski i okcitanski jezik dodaju protetski samoglasnik, što je za njihov jezik *e*. Moj model pretpostavlja da jezici isto tako dodaju samoglasnik na kraj riječi ako riječ završava na zabranjenu suglasničku skupinu. On također pretpostavlja da je više od dva suglasnika zaredom uvijek zabranjeno, što je lagano za isprogramirati, a i uvijek daje rezultate koje je moguće izgovoriti, te da *q* (kako sam označavao glas *kw*) ne može biti niti na kraju riječi niti ispred samoglasnika. Ako se nađu tri suglasnika za redom, taj moj model nakon drugog suglasnika umeće epentetski samoglasnik koji se bira posebno za svaki jezik, a umeće taj isti samoglasnik i nakon *q* ako se nakon njega ne nalazi samoglasnik (dakle, ako iza *q* slijedi suglasnik ili kraj riječi). Evo kako taj algoritam izgleda u JavaScriptu, iz datoteke etymologist.html:

1228 var allowed\_beginning\_consonant\_clusters\_in\_first\_language = [];

1229 **for** (var i = 0; i < allowed\_beginning\_consonant\_clusters.length; i++)

1230 **if** (Math.random() < 0.5)

1231 allowed\_beginning\_consonant\_clusters\_in\_first\_language.push(

1232 allowed\_beginning\_consonant\_clusters[i]

1233 );

I tako, copy-pasteano, i za suglasničke skupine na kraju riječi i za drugi jezik. To pretpostavlja da su suglasničke skupine koje će jezik dopuštati neovisne jedni o drugima, što, naravno, nije točno. Postoje brojni jezici koji na početku ili na kraju riječi ne dopuštaju niti jednu suglasničku skupinu (jedna od karakteristika samojedskih jezika je da dopuštaju mnoge suglasničke skupine u sredini riječi, ali nikakve na početku ili kraju riječi), što bi, da je ovaj model točan, bilo iznimno nevjerojatno.

Zatim se gleda nedostaje li možda koji samoglasnik u jeziku nakon glasovnih promjena, da se epentetskim samoglasnikom uspostavi fonološka ravnoteža:

1252 var jesmoLiOdabraliEpentetskiVokal = false;

1253 var epenteza1;

1254 **for** (var i = 0; i < vowels.length; i++) {

1255 var samoglasnik = vowels.charAt(i);

1256 var nalaziLiSeFonemUJeziku = false;

1257 **for** (var j = 0; j < language1.length; j++)

1258 **if** (language1[j].indexOf(samoglasnik) + 1) {

1259 nalaziLiSeFonemUJeziku = true;

1260 **break**;

1261 }

1262 **if** (!nalaziLiSeFonemUJeziku) {

1263 epenteza1 = samoglasnik;

1264 jesmoLiOdabraliEpentetskiVokal = true;

1265 **break**;

1266 }

1267 }

1268 **if** (!jesmoLiOdabraliEpentetskiVokal)

1269 epenteza1 = vowels.charAt(Math.floor(Math.random() \* vowels.length));

I to isto, copy-pasteano, za drugi jezik. Naravno, to opet radi nerealnu pretpostavku da je, recimo, jezik koji nema *a* jednako vjerojatan kao jezik koji nema *e*. To nije točno, mnogi jezici diljem svijeta nemaju *e* i *o*, nego od samoglasnika imaju samo *a*, *i* i *u*. Recimo, većina arapskih dijalekata i australijskih aboridžinskih jezika. Još jedna česta rasprava u današnjoj lingvistici jest upravo o tome smijemo li rekonstruirati indoeuropski prajezik kao jezik u kojemu su jedini samoglasnici *e* i *o*, kako se indoeuropski prajezik obično rekonstruira, a nijedan takav jezik danas ne postoji.Neovisno o tome, mislim da ne trebam izmijeniti algoritam kako ne bi pretpostavio da je jezik bez *a* jednako vjerojatan kao jezik bez *e*, jer,kao što sam rekao, *Law of Diminishing Returns*.Zatim jezgra tog dijela modela:

1291 **for** (var i = 0; i < language1.length; i++) {

1292 **if** (language1[i].charAt(language1[i].length - 1) == "q")

1293 language1[i] += epenteza1;

1294 **if** (

1295 consonants.search(language1[i].charAt(0)) !== -1 &&

1296 consonants.search(language1[i].charAt(1)) !== -1 &&

1297 allowed\_beginning\_consonant\_clusters\_in\_first\_language.indexOf(

1298 language1[i].substr(0, 2)

1299 ) === -1

1300 )

1301 language1[i] = epenteza1 + language1[i];

1302 **if** (

1303 consonants.search(language1[i].charAt(language1[i].length - 1)) !==

1304 -1 &&

1305 consonants.search(language1[i].charAt(language1[i].length - 2)) !==

1306 -1 &&

1307 allowed\_ending\_consonant\_clusters\_in\_first\_language.indexOf(

1308 language1[i].substr(language1[i].length - 2, 2)

1309 ) === -1

1310 )

1311 language1[i] += epenteza1;

1312 **for** (var j = 1; j < language1[i].length - 1; j++)

1313 **if** (

1314 (consonants.search(language1[i].charAt(j - 1)) !== -1 &&

1315 consonants.search(language1[i].charAt(j)) !== -1 &&

1316 consonants.search(language1[i].charAt(j + 1)) !== -1) ||

1317 (consonants.search(language1[i].charAt(j)) !== -1 &&

1318 language1[i].charAt(j - 1) === "q")

1319 )

1320 language1[i] =

1321 language1[i].slice(0, j) + epenteza1 + language1[i].slice(j);

1322 }

I to isto, copy-pasteano, za drugi simulirani jezik. Odgovara li taj algoritam stvarnosti? Očito ne. Kao prvo, brojni jezici dopuštaju suglasničke skupine duže od tri suglasnika, neki čak dopuštaju i na početku riječi. Može li to utjecati na rezultate? Zapravo i može, puno je veća vjerojatnost da će ovakvo pojednostavljenje utjecati na rezultate nego da će ono pojednostavljenje fonološke ravnoteže utjecati. No, opet, model realistične fonotaktike bio bi znatno kompliciraniji za programirati, a teško je reći koliko dobivamo time. Ovdje sam prije svega ciljao da perceptivnu korektnost, a ne na realističnost. Isto tako, pretpostavka da će jezici na zabranjenu skupinu suglasnika na početku riječi reagirati protetskim samoglasnikom očito ne vrijedi uvijek. Recimo, engleski je dakotsko ime rijeke *Msizipi* posudio kao *Mississippi*, dakle, stavio je epentetski samoglasnik između ta dva suglasnika, a ne ispred njih. Nasuprot tome, grčku riječ *mnemonikos* posudio je kao *mnemonic*, ali to se i u britanskom i u američkom engleskom izgovara kao da početnog *m* nema (u britanskom /nɪˈmɒn.ɪk/, a u američkom /nɪˈmɑː.nɪk/[[6]](#footnote-7)). Slično je engleski jezik napravio i s riječju *pneumonia*, kako engleska fonotaktika zabranjuje da riječ počne s *pn*, *pneumonia* se čita kao da tog početnog *p* nema. Slično se i *psychology* izgovara kao da nema početnog *p* (u britanskom /saɪˈkɒl.ə.dʒi/, a u američkom /saɪˈkɑː.lə.dʒi/[[7]](#footnote-8)).To su Vam društvene znanosti, jezici nisu matematika.

**Redukcija vokala**

Kad sam gledao kakve rezultate ispisuje dosad opisani algoritam, primijetio sam da on generira mnoge dugačke riječi s mnogo hijatusa (zijevova, nakupine samoglasnika koje ne razdvaja suglasnika), što mi se doimalo nerealističnim (nekako imam dojam da hijatusi postoje samo u kratkim riječima). Zato sam zadao da se vokali reduciraju, ovako:

1356 **for** (var i = 0; i < language1.length; i++) {

1357 var jaka = false;

1358 **if** (language1[i].length > 8) {

1359 language1[i] = language1[i].replace(/y/g, "i");

1360 language1[i] = language1[i].replace(/w/g, "u");

1361 language1[i] = language1[i].replace(/v/g, "u");

1362 jaka = true;

1363 }

1364 **for** (var j = 0; j < language1[i].length; j++)

1365 **if** (

1366 (vowels.search(language1[i].charAt(j)) !== -1 &&

1367 language1[i].charAt(j - 1) !== "q" &&

1368 (!jaka

1369 ? language1[i].charAt(j) === language1[i].charAt(j - 2)

1370 : vowels.search(language1[i].charAt(j - 2)) !== -1) &&

1371 (!jaka

1372 ? language1[i].charAt(j) === language1[i].charAt(j + 2)

1373 : vowels.search(language1[i].charAt(j + 2)) !== -1)) ||

1374 (jaka &&

1375 vowels.search(language1[i].charAt(j)) !== -1 &&

1376 language1[i].charAt(j - 1) !== "q" &&

1377 (vowels.search(language1[i].charAt(j - 1)) !== -1 ||

1378 vowels.search(language1[i].charAt(j + 1)) !== -1)) ||

1379 (jaka && language1[i].charAt(j) === language1[i].charAt(j - 1))

1380 ) {

1381 language1[i] = language1[i].slice(0, j) + language1[i].slice(j + 1);

1382 j--;

1383 }

1384 **if** (

1385 consonants.search(language1[i].charAt(language1[i].length - 1)) !==

1386 -1 &&

1387 consonants.search(language1[i].charAt(language1[i].length - 2)) !==

1388 -1 &&

1389 allowed\_ending\_consonant\_clusters\_in\_first\_language.indexOf(

1390 language1[i].substr(language1[i].length - 2, 2)

1391 ) === -1

1392 )

1393 language1[i] += epenteza1;

1394 **if** (

1395 consonants.search(language1[i].charAt(0)) !== -1 &&

1396 consonants.search(language1[i].charAt(1)) !== -1 &&

1397 allowed\_beginning\_consonant\_clusters\_in\_first\_language.indexOf(

1398 language1[i].substr(0, 2)

1399 ) === -1

1400 )

1401 language1[i] = epenteza1 + language1[i];

1402 }

I nakon toga to isto, copy-pasteano, za drugi simulirani jezik. Nakon toga:

1450 **for** (var i = 0; i < language1.length; i++) {

1451 **for** (var j = 0; j < vowels.length; j++)

1452 language1[i] = language1[i].replace(

1453 **RegExp**("ii" + vowels.charAt(j), "g"),

1454 "iy" + vowels.charAt(j)

1455 );

1456 **for** (var j = 0; j < vowels.length; j++)

1457 language1[i] = language1[i].replace(

1458 **RegExp**(vowels.charAt(j) + "ii", "g"),

1459 vowels.charAt(j) + "yi"

1460 );

1461 **for** (var j = 0; j < vowels.length; j++)

1462 language1[i] = language1[i].replace(

1463 **RegExp**("uu" + vowels.charAt(j), "g"),

1464 "uw" + vowels.charAt(j)

1465 );

1466 **for** (var j = 0; j < vowels.length; j++)

1467 language1[i] = language1[i].replace(

1468 **RegExp**(vowels.charAt(j) + "uu", "g"),

1469 vowels.charAt(j) + "wu"

1470 );

1471 language1[i] = language1[i].replace(/yy/g, "iy");

1472 language1[i] = language1[i].replace(/ww/g, "uw");

1473 }

I onda slijedi to isto za drugi simulirani jezik. Nemam razloga za vjerovati da se jezici stvarno tako ponašaju, ali model daje daleko uvjerljivije rezultate (meni barem) za dugačke riječi s ovim nego bez toga. Lingvistika je mekana znanost, i u njoj se ovakve stvari toleriraju.

**Validacija algoritma**

Postavlja se pitanje kako na donekle objektivan način provjeriti koliko su rezultati koje daje taj moj model povijesne fonologije realistični. S čime to iz stvarnog svijeta trebamo usporediti njegove rezultate? Kako osigurati da na rezultate što manje utječu promjene u vokabularu jezika (Promjene u fonologiji nisu razlog zašto je u većini današnjih romanskih jezika uobičajena riječ za zvijezdu nešto što dolazi od *stella*, a u klasičnom je latinskom riječ *sidus* bila daleko češća, niti zašto u većini današnjih romanskih jezika riječ koja dolazi od latinskog *domus* znači *katedrala*, dok im riječ za *dom* uglavnom dolazi od rijetke latinske riječi *casa*, niti zašto u većini romanskih jezika riječ za konja dolazi od rijetke latinske riječi *caballus*, a ne od uobičajene latinske riječi *equus*, niti zašto u većini romanskih jezika riječ za svinju dolazi od rijetke latinske riječi *porcus*, umjesto od uobičajene latinske riječi *sus*, niti zašto u većini romanskih jezika riječ za jabuku potječe od rijetke latinske riječi *pomum*, umjesto da dolazi od uobičajene latinske riječi *malum*.), promjene u morfologiji (U većini današnjih romanskih jezika riječ za zub glasi sličnije talijanskom *dente* nego latinskom *dens*, ali ne zato što je *s* prešlo u *t*, nego zato što se nastavak *s* za nominativ jednine u trećoj deklinaciji izgubio u kasnom latinskom. I nije očito koja su pravila za vrijedila. U slučaju riječi za zub korijen kosih padeža, *dent*,postao je novi nominativ, ali nije uvijek bilo tako: riječ za vrijeme, *tempus*, reinterpretirana je kao imenica muškog roda druge deklinacije, umjesto da je zadržala korijen kosih padeža *tempor*, a riječ za srce, *cor*, u kosim je padežima izgubila *d* koje je postojalo u klasičnom latinskom), te pristranost etimologija (jer, suočimo se s time, etimologija nije egzaktna znanost). Ipak, mislim da su nazivi brojeva od 1 do 10 dobar primjer za testiranje tog mog modela povijesne fonologije, jer se korijeni naziva brojeva od 1 do 10 gotovo nikad ne mijenjaju s vremenom, a i obično se koriste bez gramatičkih dodataka. Zato sam sastavio JSON bazu podataka od 74 naziva brojeva od 1 do 10 u različitim prajezicima, te odgovarajućih naziva brojeva u jezicima koji su od njih nastali, da bih pomoću nje provjerio koliko ta simulacija odgovara stvarnosti. Ta baza podataka može se skinuti preko linka koji se nalazi na dnu *Etymology Game*a. Koristio sam JSON umjesto SQL-a zato što, kao prvo, JSON znam mnogo bolje nego SQL, a, kao drugo, JSON mogu doslovno staviti neizmijenjen u program pisan u JavaScriptu, jer JSON jest podskup JavaScripta. Da sam koristio SQL, morao bih ga čitati iz datoteke, za što ne postoje naredbe u standardnom JavaScriptu, i morao bih koristiti neki radni okvir za interpretiranje SQL-a, koji gotovo sigurno ne bi bio pisan tako da bude kompatibilan s raznim JavaScript engineima (vjerojatno bi bio kompatibilan samo s NodeJS-om, dok se ovako taj moj program za testiranje tog modela zasigurno može vrtjeti i u NodeJS-u, i u QuickJS-u i u ChakraCoreu, a gotovo sam siguran i da bi se vrtio i u SpiderMonekeyu, tako da možda ne morate instalirati ništa novo na svoje računalo da biste taj moj program pokrenuli). Potprogram koji validira taj moj model izgleda ovako (iz datoteke etymology\_game\_validator.js):

2066 **for** (let prarijec **of** baza\_podataka) {

2067 prarijec.broj\_pogodenih = 0;

2068 prarijec.simulirani\_potomci = [];

2069 }

2070 let jesmo\_li\_naletjeli\_na\_gresku = false;

2071 **const** koliko\_cemo\_puta\_izvrsiti\_simulaciju = 500;

2072 **for** (let i = 0; i < koliko\_cemo\_puta\_izvrsiti\_simulaciju; i++) {

2073 simulacija();

2074 **for** (let i = 0; i < baza\_podataka.length; i++) {

2075 **if** (baza\_podataka[i].root != original[i]) {

2076 console.log(

2077 "Neki dio simulacije je promijenio dio originala, prekidamo!"

2078 );

2079 jesmo\_li\_naletjeli\_na\_gresku = true;

2080 **break**;

2081 }

2082 **if** (!baza\_podataka[i].simulirani\_potomci.includes(language1[i])) {

2083 baza\_podataka[i].simulirani\_potomci.push(language1[i]);

2084 **if** (baza\_podataka[i].descendants.includes(language1[i]))

2085 baza\_podataka[i].broj\_pogodenih++;

2086 }

2087 **if** (!baza\_podataka[i].simulirani\_potomci.includes(language2[i])) {

2088 baza\_podataka[i].simulirani\_potomci.push(language2[i]);

2089 **if** (baza\_podataka[i].descendants.includes(language2[i]))

2090 baza\_podataka[i].broj\_pogodenih++;

2091 }

2092 }

2093 **if** (jesmo\_li\_naletjeli\_na\_gresku) **break**;

2094 }

Kôd prikazan gore zadaje modelu da generira 1000 riječi potomaka (500 u oba jezika) za svaki korijen iz baze podataka, te broji koliko se puta dogodilo da se riječ identična onoj u simulaciji nalazi u bazi podataka kao njezin potomak u realnosti. Nakon koda koji je prikazan gore slijedi ispisivanje HTML koda stranice s tablicom s rezultatima simulacije na standardni izlaz, pomoću multi-line stringova i naredbe console.log (koja *de facto*, iako ne i *de iure*,pripada standardnom u JavaScriptu). Rezultati te validacije, što ju je program etymology\_game\_validator.js ispisao, dostupni su na mom blogu, preko linka koji se nalazi na dnu stranice *Etymology Game*. Kad sam ih vidio, doimali su se pomalo razočaravajućima: moj je model pogodio pravog potomka u svega 0.56% slučajeva. Naravno, taj rezultat zapravo je beznačajan ako ga nemamo s čime usporediti. Dobar način da dobijemo dojam koliko je to dobar rezultat bio bi da ga usporedimo s rezultatom modela koji nasumično izmjenjuje riječi, u kojeg nisu programirani nikakvi zakoni lingvistike. To sam i napravio, ovako (iz datoteke random\_algorithm\_validator.js):

1462 function nasumicno\_izmijeni\_rijec(pocetna\_rijec) {

1463 let nova\_rijec = pocetna\_rijec;

1464 **const** vjerojatnost\_izmjene\_pojedinog\_slova = 1 / (pocetna\_rijec.length \* 2);

1465 **for** (let i = 0; i < pocetna\_rijec.length + 1; i++) {

1466 **if** (Math.random() < vjerojatnost\_izmjene\_pojedinog\_slova) {

1467 **const** nasumicna\_vrijednost = Math.random();

1468 **if** (nasumicna\_vrijednost < 1 / 3)

1469 //Nasumicno umetanje slova.

1470 nova\_rijec =

1471 nova\_rijec.substr(0, i) +

1472 abeceda[Math.floor(Math.random() \* abeceda.length)] +

1473 nova\_rijec.substr(i);

1474 **else** **if** (nasumicna\_vrijednost < 2 / 3)

1475 //Nasumicno brisanje slova.

1476 nova\_rijec = nova\_rijec.substr(0, i) + nova\_rijec.substr(i + 1);

1477 //Nasumicna zamijena slova drugim slovom iz abecede.

1478 **else**

1479 nova\_rijec =

1480 nova\_rijec.substr(0, i) +

1481 abeceda[Math.floor(Math.random() \* abeceda.length)] +

1482 nova\_rijec.substr(i + 1);

1483 }

1484 }

1485 **return** nova\_rijec;

1486 }

1487 function simulacija() {

1488 language1 = [];

1489 language2 = [];

1490 **for** (**const** prarijec **of** original) {

1491 language1.push(prarijec);

1492 language2.push(prarijec);

1493 }

1494 **for** (let i = 0; i < Math.min(language1.length, language2.length); i++) {

1495 language1[i] = nasumicno\_izmijeni\_rijec(language1[i]);

1496 language2[i] = nasumicno\_izmijeni\_rijec(language2[i]);

1497 }

1498 }

Taj algoritam daje točno rješenje, kako sam izmjerio, u 0.48% slučajeva. Dakle, moj model je jedva (0.56-0.48)/0.48=16.7% bolji od modela koji daje posve nasumične rezultate. Pa što se ovdje zapravo događa?

Pokušat ću na to odgovoriti *eyeball*iranjem podataka. Prvo što upada u oči da nasumičan algoritam nerijetko proizvodi neizgovorljive riječi, kao što su *krzkd* ili *ghghad*, što je, kako mi se sada čini, i jedini razlog zašto je moj model bolji od nasumičnog algoritma. Ali isto tako, nekada su riječi koje moj model smatra nemogućima upravo točna rješenja. Recimo, nasumičan će algoritam relativno lako od berberske riječi za broj 10, *meraw*, proizvesti tamazitsku riječ *mraw* (Za nasumičan algoritam vjerojatnost za to je ((2\*5-1)/(2\*5))^4\*1/(2\*5)\*1/3=2.2%, naime, budući da korijen *meraw* ima 5 slova, vjerojatnost da će ostala četiri ostati neizmijenjena iznosi ((2\*5-1)/(2\*5))^4, vjerojatnost da će upravo drugo slovo biti izmijenjeno iznosi 1/(2\*5), a vjerojatnost da će ono biti izbrisano, a ne da će se na njegovo mjesto umetnuti neko drugo slovo ili da će biti zamijenjeno nekim drugim slovom, iznosi 1/3.), dok će moj model smatrati da je vjerojatnost za to 0%, jer mu nije programirano da riječ može početi na *mr*. Ima još slučajeva gdje moj model radi *worse than chance* predviđanja. Recimo, nasumičan će algoritam relativno lagano od indeuropskog korijena *decm* izvesti latinsku riječ za broj 10, *decem* (za nasumičan algoritam vjerojatnost za to je ((2\*4-1)/(2\*4))^3\*1/(2\*4)\*1/3\*1/26=0.1%, dok moj model tvrdi da je vjerojatnost za prijelazak *decm* u *decem* 0.006%, dakle, da bi se to trebalo dogoditi samo u jednom od oko 15000 jezika, što očito manje odgovara stvarnosti), dok moj model smatra riječ kao što je *decme* daleko vjerojatnijom (smatra da je vjerojatnost da *decem* prijeđe u *decme* 2.6%). I nije latinski jezik jedini indoeuropski jezik koji je između *c* i *m* u riječi za broj deset umetnuo epentetski samoglasnik. Gotska riječ za broj deset bila je *taihun*, ovdje *t* dolazi od indoeuropskog *d*, diftong *ai* od indoeuropskog *e*, *h* dolazi od indoeuropskog *c* (glasa kao *c* u engleskom *cube*, često se označava i kao *kj*), *n* dolazi od indoeuropskog *m*, dok je *u* epentetski samoglasnik dodan radi lakšeg izgovora.Slično tomu, armenska riječ za broj deset jest *tasu*, od staroarmenskog *tasun*. Isto vrijedi i za latinsku riječ za broj 7, *septem*, od indoeuropskog *septm*: moj model je pristran tome da smatra takvo umetanje epentetskog samoglasnika između suglasnika koji se ne smiju pojaviti zajedno na kraju riječi mnogo manje vjerojatnim nego dodavanje paragognog samoglasnika na kraj riječi, iako činjenice govore drugačije. Zato je, primjerice, moj model kao potomke indoeuropskog korijena *septm* pogodio točarijansku riječ za broj sedam, *spat*, i grčku riječ za broj sedam, *hepta*, dok je nasumični algoritam pogodio latinsku riječ *septem* i ladinsku riječ *set*.

Za većinu riječi u prajezicima nisu niti nasumični model niti moj model pogodili nijednog potomka. Za neke riječi je to bilo posve očekivano, recimo, sinotibetanska prariječ za broj osam bila je *triat*, a u gotovo svim današnjim sinotibetanskim jezicima ona počinje ili sa *b* ili sa *p*. Do objašnjenja zašto je točno Robert Shafer u knjizi *Introduction to Sino-Tibetan* rekonstruirao sinotibetansku prariječ za broj osam kao *triat* (i zašto je to danas, kako se čini po onome što piše na internetu, općeprihvaćeno) nisam uspio doći. Uvelike mi u tome odmaže i činjenica da je velika većina literature dostupne besplatno na internetu o sinotibetanskoj lingvistici, naravno, ili na kineskom ili na ruskom. To i nije toliko važno, jer slične stvari imamo i u indoeuropskoj lingvistici. Recimo, indoeuropska riječ za broj osam rekonstruira se kao *h3ekjtow*, gdje *h3* označava mekano *h* (indoeuropski prajezik razlikovao je tvrdo h, koje se piše *h1*, srednje h, *h2*, i meko h, *h3*), iako u većini indoeuropskih jezika naziv za broj osam počinje sa samoglasnikom *o*. Ipak, imamo vrlo dobar razlog za vjerovati da je indoeuropska prariječ za broj osam uistinu počinjala s mekim *h*. Ferdinand de Saussure predvidio je postojanje indoeuropskog jezika gdje riječ ne može počinjati ili završiti samoglasnikom, gdje kratko *a* u drugim indoeuropskim jezicima odgovora skupini glasova *h2e* u njemu, dugo *a* odgovara skupini *eh2* u njemu, a dugo *o* odgovara *eh3*, desetljećima prije no što je hetitski jezik, koji je to predviđanje potvrdio, otkriven. No, u kasnom indoeuropskom, nakon što su se anatolijski jezici (hetitski i njemu blisko srodni jezici) odvojili od njega, ta su slova *h* izgubljena, a fonotaktika se znatno promijenila. Model koji bi predviđao veliku vjerojatnost nestajanja svih glasova sličnih *h* i prelaska *he* u *ho*, ili model koji bi predviđao veliku vjerojatnost prelaska *tr* u *pr*, davao bi bolje rezultate za indoeuropske ili sinotibetanske jezike, ali znatno lošije za jezike općenito.

Prepravljanje modela na temelju ovih rezultata i onda ga opet testirati na tim istim podacima bilo bi ekstremno kršenje znanstvene metode. Ako bismo ga prepravili da, recimo, dopušta *mr* na početku riječi i da umetanje epenteze unutar zabranjene suglasničke skupine na kraju riječi smatra vjerojatnijim od dodavanja paragoge na kraj riječi, vjerojatno bi dao bolji rezultat za ove podatke, ali zaključiti iz toga da bi davao bolji rezultat za bilo koje dobre podatke bio bi udžbenički primjer logičke pogreške *Texas Sharpshooter Fallacy*. Ako ćemo raditi izmjene na ovom modelu, trebali bi nam novi podaci za testiranje, a ne znam koji bi podaci više bili toliko dobri.

**Primjena računalnih modela u etimologiji**

Već godinama radim na svojem alternativnom tumačenju naziva mjesta (toponima) u Hrvatskoj.

Jedna od etimologija koju smatram vjerojatnom je da je *Issa*, antički naziv za grad Vis, značilo *izvor ljekovite vode* na ilirskome, da se taj naziv odnosio na izvor koji se nalazio zapadno od termi i od kojeg su se terme napajale[[8]](#footnote-9), i da su te terme napravljene upravo zato što su Iliri vjerovali da je taj izvor ljekovit. Isti se element pojavljuje u antičkom nazivu za Daruvar, *Balissa*, te u antičkom nazivu za Varaždinske Toplice, *Iasa*. Također mislim da znam etimologiju te riječi, da dolazi od indoeuropskog korijena *yes*, koji znači *vreti* (kao što hrvatska riječ *vrelo* dolazi od riječi *vreti*). Naravno, problem etimologije naziva *Issa* kompliciran je. Pokušaji objašnjenja tog naziva potječu još od antike. Teopomp je u 4. stoljeću prije Krista tvrdio da je ime Issa povezano s ilirskim osobnim imenom Jonije, što je očito pogrešno[[9]](#footnote-10). Strabon je u 1. stoljeću prije Krista, u 5. poglavlju 8. svitka Geografije, spekulirao da je ime *Issa* povezano s imenom *Ant****issa*** na grčkom otoku Lezbosu. To je objašnjenje prihvatio Petar Skok, no Dubravka Ivšić misli da je *takvo povezivanje* *pučkoetimološko, ako ne i mitološko* (ne znam zašto onda prihvaća baš slično Strabonovo objašnjenje da su imena *Pharos*, antičko ime za Hvar, i *Paros*, ime grčkog otoka, povezani, a odbacuje daleko smislenije objašnjenje Apolonija s Rodosa da je *Pharos* ilirska riječ za borovu šumu, koje prihvaća Anton Mayer)[[10]](#footnote-11). I Dubravka Ivšić i Branimir Gabričević misle da su te pogrešne etimologije iz antike dokazi da se ime *Issa* ni u to vrijeme nije moglo objasniti, da ono datira još od vremena prije ilirskog jezika. Ja mislim da to nije uvjerljiv dokaz, jer zašto pretpostaviti da su Strabon i Teopomp znali ilirski? Valentin Putanec je napisao, u *Izolekse za pojam otok* objavljenima 1995. u Folia Onomastica Croatica, da misli da je *issa* bila pelazgijska riječ za otok, od istog indoeuropskog korijena kao i latinska riječ *insula*, irska riječ *inis* i grčka riječ *nesos*. Putanec, naime, smatra da je pelazgijski jezik bio indoeuropski. Po meni je ta etimologija malo vjerojatna, jer, kao prvo, ne vidim razloga za vjerovati da je pelazgijski jezik bio indoeuropski. Štoviše, Stjepan Bizantski napisao je da je *Huttenia* bio pelazgijski naziv za Tetrapolis, a znamo da je *tetra* grčka riječ za četiri i da je *polis* grčka riječ za grad, a da je *huth* etrurska riječ za četiri i da je *tam* etrurska riječ za građevinu, što bi po meni jako sugeriralo da je pelazgijski jezik bio srodan etrurskom, a ne da je bio indoeuropski jezik. Kao drugo, glavna struja lingvistike smatra da su riječi *insula*, *inis* i *nesos* nepovezane. Glavna struja lingvistike smatra da *insula* dolazi od *in* (u) i *salum* (more), da je *nesos* povezano s grčkim *nein* (plivati), a da *inis* znači *ono što stoji (u vodi)*. Još jedan problem je ima li smisla pretpostaviti da je *issa* u *Balissa* riječ sa značenjem, a ne sufiks, budući da je u keltskim jezicima *-issa* bio čest sufiks. Ne vidim razloga da vjerujemo da se isti sufiks pojavio i u keltskom i u ilirskom, kada on nije mogao postojati u indoeuropskom prajeziku, koji nije imao udvostručene suglasnike. Nadam se da se slažete sa mnom da je objašnjenje da je *issa* značilo *izvor ljekovite vode* najrazumnije.

Druga stvar koju smatram vjerojatnom jest da je u ilirskom jeziku postojala riječ *karr~kurr* sa značenjem *teći*, od koje dolaze imena rijeka *Krka*, *Korana*, *Krapina*, *Krbavica*, *Karašica* (ime dviju rijeka relativno blizu jedna drugoj, jedna se ulijeva u Dravu, a druga u Dunav) te možda *Kravaršćica* (ime izgleda kao da je nazvano po gradiću Kravarsko, koji bi lako mogao biti nazvan po životinji kravi, no moguće je i obratno). Kolika je vjerojatnost da se, dakle, u šest naziva rijeka u Hrvatskoj na početku pojavi isti par suglasnika, ako ta imena nisu povezana? Na prvi pogled se čini da je vjerojatnost zanemarivo mala, no je li uistinu tako? Sjetimo se paradoksa rođendana (vjerojatnost da u grupi od samo 23 osobe dvije od njih imaju isti rođendan iznosi čak 50%). Događa li se ovdje paradoks rođendana? Kako to izračunati? Nije bilo očito kako do tog odgovora doći analitički, ali relativno je lagano doći numerički, pomoću ovog programa pisanog u C-u (on pretpostavlja da u Hrvatskoj ima 100 imena rijeka koje treba uzeti u obzir, te da prosječan jezik ima 20\*20=400 parova suglasnika koji su podjednako vjerojatni):

1 #include <stdio.h>

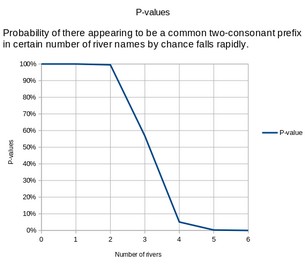
2 #include <stdlib.h>

3 #include <time.h>

4

5 **int** main() {

6 srand(time(0));

  
Slika 1: Paradoks rođendana naglo nestaje kako se broj ljudi koji dijele isti rođendan povećava

7 **int** number\_of\_repeats;

8 scanf("%d",&number\_of\_repeats);

9 **const** **int** number\_of\_rivers=100;

10 **float** sum=0;

11 **int** counter[400];

12 **for** (**int** i=0; i<10000; i++)

13 {

14 **for** (**int** j=0; j<400; j++)

15 counter[j]=0;

16 **for** (**int** j=0; j<number\_of\_rivers; j++)

17 counter[rand()%400]++;

18 **int** max=0;

19 **for** (**int** j=0; j<400; j++)

20 max=(counter[j]>max)?(counter[j]):(max);

21 **if** (max>=number\_of\_repeats) sum++;

22 }

23 printf("%f\n",sum/100);

24 }

Iako paradoks rođendana postoji za n=2 (vjerojatnost da dvije rijeke nepovezanih imena imaju ista prva dva suglasnika iznosi veoma blizu 100%, iako se nekome može učiniti da je znatno manja), on veoma brzo nestaje kako se *n* povećava. Za n=3, ta vjerojatnost već iznosi 56.8%. Za n=4, ona iznosi 5.1%. Za n=5, ona iznosi 0.26%. Za n=6, ona iznosi 0.01% (što je 1/10000). Dokazuje li to da su nazivi tih rijeka uistinu povezani? Ovaj način dokazivanja povezanosti imena veoma je neobičan u lingvistici. Etimologije vlastitih imena teško je dokazati. Obično se za to poziva na komparativnu lingvistiku i na povijesne izvore. Etimologija da je Brač bila ilirska riječ za jelena općeprihvaćena je zato što ju je u 6. stoljeću predložio Stjepan Bizantski, zato što je Strabon u 1. stoljeću prije krista, u 6. svitku u 3. poglavlju Geografije, predložio da ime talijanskog gradića Brindisi dolazi od mesapske (neki su antički povjesničari tvrdili da je mesapski jezik bio srodan ilirskome) riječi za jelena jer zaljev u kojem se on nalazi oblikom podsjeća na glavu jelena, te zato što je današnja latvijska riječ za jelena *briedis*. Ja bih ipak ostavio mogućnost da je ime *Brač* zapravo povezano s *Barbania*, antičkim imenom za Bojanu, kako ga je zapisao Tit Livije u Historia Romana u 44. svitku u 31. poglavlju, vjerojatno povezanim s grčkom riječju za blato, *borboros*. Naime, Stjepan Bizantski napisao je da Bračem teče potok zvan *Brettia*. Nisam proučavao geologiju da potvrdim da je moguće da je to tada bilo tako. Ako jest, to je vjerojatnija etimologija nego da je potok nazvan po životinji koja za Brač nije tipična, štoviše, pitanje je da li je u to vrijeme na Braču uopće bilo jelena.Postavlja se pitanje koliko su pretpostavke koje program naveden gore radi realistične. Smijemo li pretpostaviti da prosječan jezik ima 20 suglasnika koji se ponavljaju približno podjednako često? Ako nasumično odabireš suglasnike iz nekog teksta, kolika je vjerojatnost da ćeš dva puta za redom odabrati isti? Drugim riječima, koliku entropiju imaju suglasnici u ljudskim jezicima? I što uzrokuje da oni budu neravnomjerno raspoređeni? Je li to fonologija, pa da to trebamo uzeti u obzir pri proučavanju toponima? Ili su to uglavnom morfologija i sintaksa, koji na toponime posuđene iz nepoznatog jezika nemaju utjecaja?

**Utjecaj raznih dijelova gramatike na entropiju jezika**

Pitanje „*Ako nasumično odabireš suglasnike iz nekog teksta, kolika je vjerojatnost da ćeš dva puta za redom odabrati isti?”* može se odgovoriti mjerenjem. Dugo sam mislio da je ono ekvivalentno pitanju „*Koliku entropiju imaju suglasnici u ljudskim jezicima.*”, no mjerenja su mi poslije pokazala drugačije. Dakle, primjer programa koji odgovara na prvo pitanje jest sljedeći program u C-u:

1 #include <stdio.h>

2 #include <stdlib.h>

3 #include <ctype.h>

4 #include <time.h>

5 #include <string.h>

6

7 **int** main() {

8 srand(time(0));

9 **FILE** \*input=fopen("text.txt","r");

10 **if** (!input) {

11 fprintf(stderr,"Can't open 'text.txt'!\n"),

12 exit(1);

13 }

14 fseek(input,0,SEEK\_END);

15 **int** length=ftell(input),sum1=0,sum2=0;

16 /\*Variable "length" now contains the total number

17 of characters in "text.txt".\*/

18 **int** alphabet[26]={0};

19 **const** **char** \*vowels="aeiou";

20 **for** (**int** i=0; i<1000000; i++) {

21 **int** tmp=rand()%length;

22 fseek(input,tmp,SEEK\_SET);

23 **char** first=fgetc(input);

24 tmp=rand()%length;

25 fseek(input,tmp,SEEK\_SET);

26 **char** second=fgetc(input);

27 **if** (isalpha(first) &&

28 !strchr(vowels,tolower(first))

29 && isalpha(second) &&

30 !strchr(vowels,tolower(second))) {

31 /\*If both randomly chosen characters

32 from "text.txt" happen to be consonants...\*/

33 sum2++;

34 **if** (first==second) {

35 sum1++;

36 alphabet[toupper(first)-'A']++;

37 }

38 }

39 }

40 printf("1/%f\n",(**float**)sum2/sum1);

41 **int** max=0;

42 **for** (**int** i=0; i<26; i++)

43 **if** (alphabet[i]>alphabet[max]) max=i;

44 printf("%c %f%%\n",max+'A',100.\*alphabet[max]/sum1);

45 }

Kad sam preuzeo neki tekst na engleskom jeziku i spremio ga kao datoteku text.txt, očekivao sam da će rezultat biti nešto malo manji od 21, jer engleski jezik ima 21 suglasnik. Međutim, kad sam to zapravo izmjerio, dobio sam rezultat približno jednak 11, te da je najčešći suglasnik 't'. Pretpostavio sam onda da je velika većina te neravnomjerne raspodjele suglasnika u jeziku zbog sintakse, jer engleska gramatika često zahtijeva da se u rečenice ubacuju riječi na 't' koje ne znače ništa: *the, it, that, than*... Međutim, kada sam to napravio s rječnikom od LibreOfficeove provjere pravopisa za engleski jezik, rezultat se popeo svega na 13, najčešći suglasnik je tada bio 'r' (pretpostavljam da je to zbog engleskih prefiksa i sufiksa *re*- i -*er*). I isto vrijedi i za druge jezike, za hrvatski jezik rezultati su 13 za tekst i 14 za rječnik provjere pravopisa, dok su za njemački jezik rezultati 12 i 15. Pretpostavio sam (bez previše opravdavanja) da morfologija oduzima od jezika jednako entropije koliko i fonotaktika, te, kad sam preuredio simulaciju da to pretpostavlja, dobio sam da je ona vjerojatnost s toponimima 3/10000.

Poslije sam došao na ideju da bih mogao procijeniti koliki utjecaj fonotaktika ima na entropiju jezika tako da napišem program koji će mjeriti entropiju parova suglasnika koji se nalaze jedan do drugog (bio između njih samoglasnik ili ne bio) u rječniku za provjeru pravopisa. To sam napravio ovim programom za NodeJS:

1 "use strict"; // To znači, otprilike, "Ponašaj se po standardu i ne pokušavaj

2 // biti kompatibilan sa starijim verzijama NodeJS-a.". Nekad zna

3 // riješiti naizgled neobjašnjive probleme u programiranju.

4 let suglasnici =

5 "bcčćdđfghjklmnpqrsštvwxyzž"; // NodeJS podržava ne-ASCII (hrvatske...)

6 // znakove u stringovima.

7 suglasnici += suglasnici.toUpperCase();

8 **const** datotecniSustav = require("fs");

9 **const** tekst =

10 datotecniSustav.readFileSync("main.dic", {encoding : 'utf-8', flag : 'r'});

11 let mapa = **new** Map();

12 **for** (**const** prvi **of** suglasnici)

13 **for** (**const** drugi **of** suglasnici)

14 mapa.set((prvi + drugi).toLowerCase(), 0);

15 let prethodni, sadasnji, brojac = 0;

16 **for** (**const** znak **of** tekst) {

17 **if** (suglasnici.indexOf(znak) !== -1) {

18 prethodni = sadasnji;

19 sadasnji = znak.toLowerCase();

20 **if** (prethodni !== **undefined**) {

21 brojac++;

22 mapa.set(prethodni + sadasnji, mapa.get(prethodni + sadasnji) + 1);

23 }

24 }

25 }

26 console.log("Apsolutne frekvencije pojedinih parova:");

27 **for** (**const** parSuglasnika **of** mapa.keys()) {

28 console.log(parSuglasnika + " " + mapa.get(parSuglasnika));

29 }

30 let entropija = 0;

31 **for** (**const** apsolutnaFrekvencija **of** mapa.values())

32 **if** (apsolutnaFrekvencija)

33 entropija -= apsolutnaFrekvencija / brojac \*

34 Math.log2(apsolutnaFrekvencija / brojac);

35 console.log("Entropija parova suglasnika: " + entropija);

36 console.log("Vjerojatnost prosječnog para suglasnika: " + (1 / 2 \*\* entropija));

37 console.log("Inverzna vjerojatnost prosječnog para suglasnika: " +

38 2 \*\* entropija);

Kad sam to napravio s rječnikom za hrvatski jezik, očekivao sam da će ta entropija parova suglasnika biti nešto manje od 2\*log2(14) bita po simbolu, jer je entropija pojedinih suglasnika u rječniku za provjeru pravopisa za hrvatski jezik, mislio sam, bila log2(14) bita po simbolu. Međutim, rezultat je bio posve neočekivan, dobio sam rezultat približno jednak log2(229), što je više od 2\*log2(14). Kao da fonotaktika nekim čudom povećava entropiju jezika, a ne da je smanjuje. Da stvar bude gora, pravi rezultat mora biti veći od log2(229), jer, naravno, riječi u rječniku sadrže česte prefikse i sufikse. Kad sam izmijenio simulaciju da pretpostavlja da parovi suglasnika imaju entropiju log2(229), dobio sam da je vjerojatnost da se slučajno pojavi onaj uzorak u nazivima rijeka u Hrvatskoj oko 1/500, no nisam siguran da takav izračun išta znači.

Odlučio sam provjeriti pretpostavku da je pitanje „*Ako nasumično odabireš suglasnike iz nekog teksta, kolika je vjerojatnost da ćeš dva puta za redom odabrati isti?”* ekvivalentno pitanju „*Koliku entropiju imaju suglasnici u ljudskim jezicima.*”. Ako je tako, program u JavaScriptu koji ću sada navesti trebao bi, za istu datoteku text.txt, dati isti rezultat kao onaj već opisani program u C-u:

1 /\* Ne znam zašto ovaj program u JavaScriptu tvrdi da je entropija tekstova

2 \* znatno veća nego što to tvrdi onaj program pisan u C-u. \*/

3 function jeLiSlovo(znak) { **return** znak.toUpperCase() !== znak.toLowerCase(); }

4

5 **const** datotecni\_sustav = require("fs");

6 **const** tekst = datotecni\_sustav.readFileSync("text.txt", {encoding : "utf-8"});

7 let mapa = **new** Map();

8 **const** samoglasnici = "aeiou";

9 **for** (**const** znak **of** tekst)

10 **if** (jeLiSlovo(znak) && samoglasnici.indexOf(znak.toLowerCase()) === -1)

11 mapa.set(znak.toLowerCase(), (mapa.get(znak.toLowerCase()) | 0) + 1);

12 let zbroj = 0;

13 **for** (**const** apsolutna\_frekvencija **of** mapa.values())

14 zbroj += apsolutna\_frekvencija;

15 let entropija = 0;

16 **for** (**const** apsolutna\_frekvencija **of** mapa.values())

17 entropija -= Math.log2(apsolutna\_frekvencija / zbroj) \*

18 (apsolutna\_frekvencija / zbroj);

19 console.log(2 \*\* entropija);

I... ispostavilo da ne daju isti rezultat. Ovaj program što sam ga sada izlistao tvrda da je entropija teksta (ne rječnika) na engleskom jeziku log2(14), dok je onaj program tvrdio da je ona log2(11). Da bih pronašao uzrok tome, napisao sam sljedeći program u Octavi (trebao bi se vrtjeti i u MatLabu, ali nisam uspio pokrenuti MatLab na svom računalu da to provjerim):

1 suglasnici = 'bcdfghjklmnpqrstvwxyz';

2 testni\_stringovi=cell(100 –length(suglasnici) **+** 1, 1);

3 **for** koliko\_cemo\_staviti\_b\_ova = 100 **-** length(suglasnici) **+** 1 : **-**1 : 1

4 **for** i = 1 : koliko\_cemo\_staviti\_b\_ova

5 testni\_stringovi{koliko\_cemo\_staviti\_b\_ova} = [

6 testni\_stringovi{koliko\_cemo\_staviti\_b\_ova} 'b'

7 ];

8 **end**

9 **for** i = 1 : 100 **-** koliko\_cemo\_staviti\_b\_ova

10 testni\_stringovi{koliko\_cemo\_staviti\_b\_ova} = [

11 testni\_stringovi{koliko\_cemo\_staviti\_b\_ova}

12 suglasnici(

13 int32(

14 **floor**(

15 (i **-** 1) **/**

16 (100 **-** koliko\_cemo\_staviti\_b\_ova) **\***

17 (length(suglasnici) **-** 1)

18 ) **+** strfind(suglasnici, 'c')

19 )

20 )

21 ];

22 **end**

23 **end**

24 function ret = shannonova\_entropija(str, suglasnici)

25 apsolutne\_frekvencije = [];

26 **for** i = 1 : length(suglasnici)

27 apsolutne\_frekvencije = [apsolutne\_frekvencije 0];

28 **end**

29 **for** i = 1 : length(str)

30 znak = str(i);

31 apsolutne\_frekvencije(

32 strfind(suglasnici, znak)

33 ) = apsolutne\_frekvencije(

34 strfind(suglasnici, znak)) **+** 1;

35 **end**

36 relativne\_frekvencije = apsolutne\_frekvencije **/** length(str);

37 ret = 0;

38 **for** relativna\_frekvencija = relativne\_frekvencije

39 **if** relativna\_frekvencija **>** 0

40 ret = ret **-** log2(relativna\_frekvencija) **\*** relativna\_frekvencija;

41 **end**

42 **end**

43 **end**

44 function ret = samarzijina\_entropija(str, suglasnici)

45 broj\_pokusaja = 10000;

46 broj\_pogodaka = 0;

47 **for** i = 1 : broj\_pokusaja

48 prvi = int32(**floor**(**rand**() **\*** length(str) **+** 1));

49 drugi = int32(**floor**(**rand**() **\*** length(str) **+** 1));

50 **if** str(prvi) **==** str(drugi) **&&** prvi **~**= drugi

51 broj\_pogodaka = broj\_pogodaka **+** 1;

52 **end**

53 **end**

54 omjer\_pogodaka = broj\_pogodaka **/** broj\_pokusaja;

55 ret = **-**log2(omjer\_pogodaka);

56 **end**

57 samarzijine\_entropije = [];

58 shannonove\_entropije = [];

59 **for** i = 1 : length(testni\_stringovi)

60 str = testni\_stringovi{i};

61 samarzijine\_entropije = [samarzijine\_entropije

62 samarzijina\_entropija(str, suglasnici)];

63 shannonove\_entropije = [shannonove\_entropije

64 shannonova\_entropija(str, suglasnici)];

65 **end**

66 plot(shannonove\_entropije, samarzijine\_entropije);

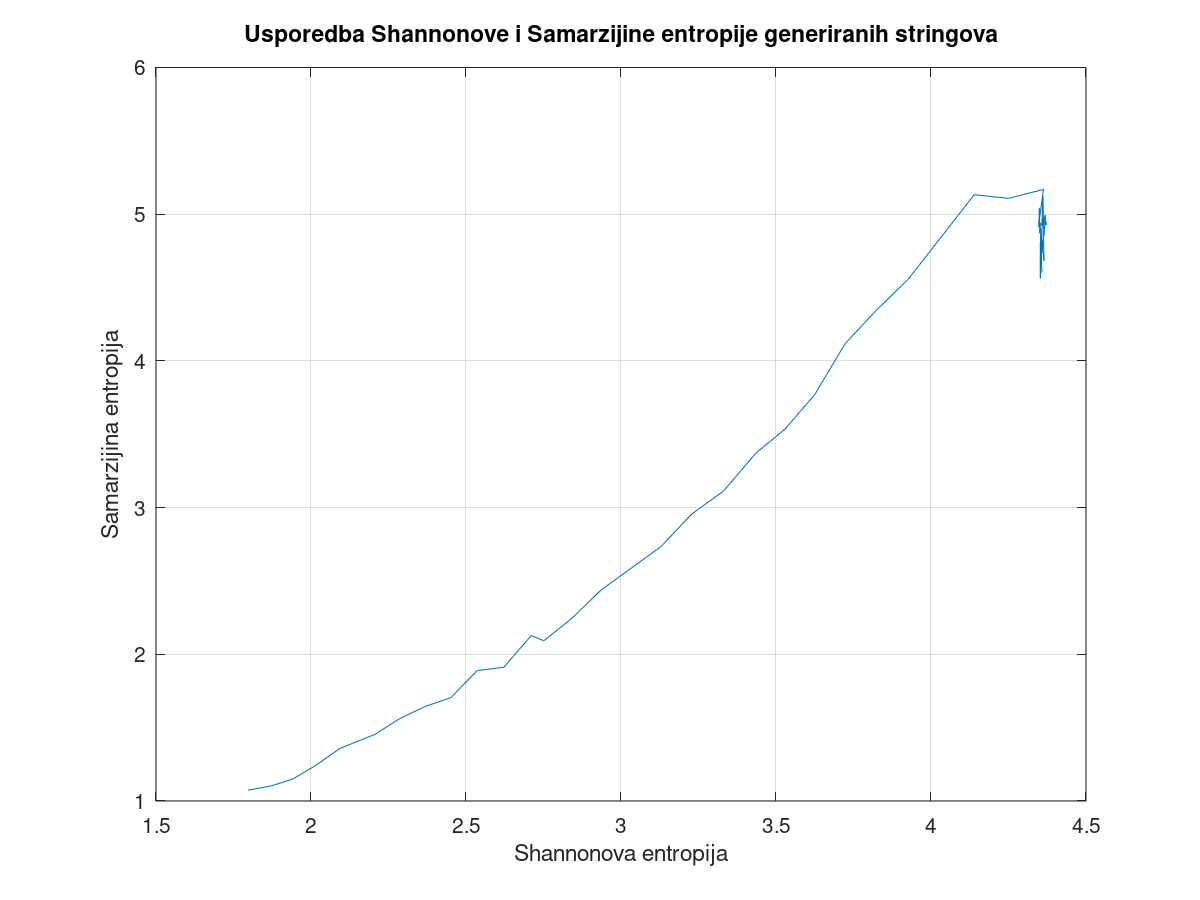
67 xlabel('Shannonova entropija');

68 ylabel('Samarzijina entropija');

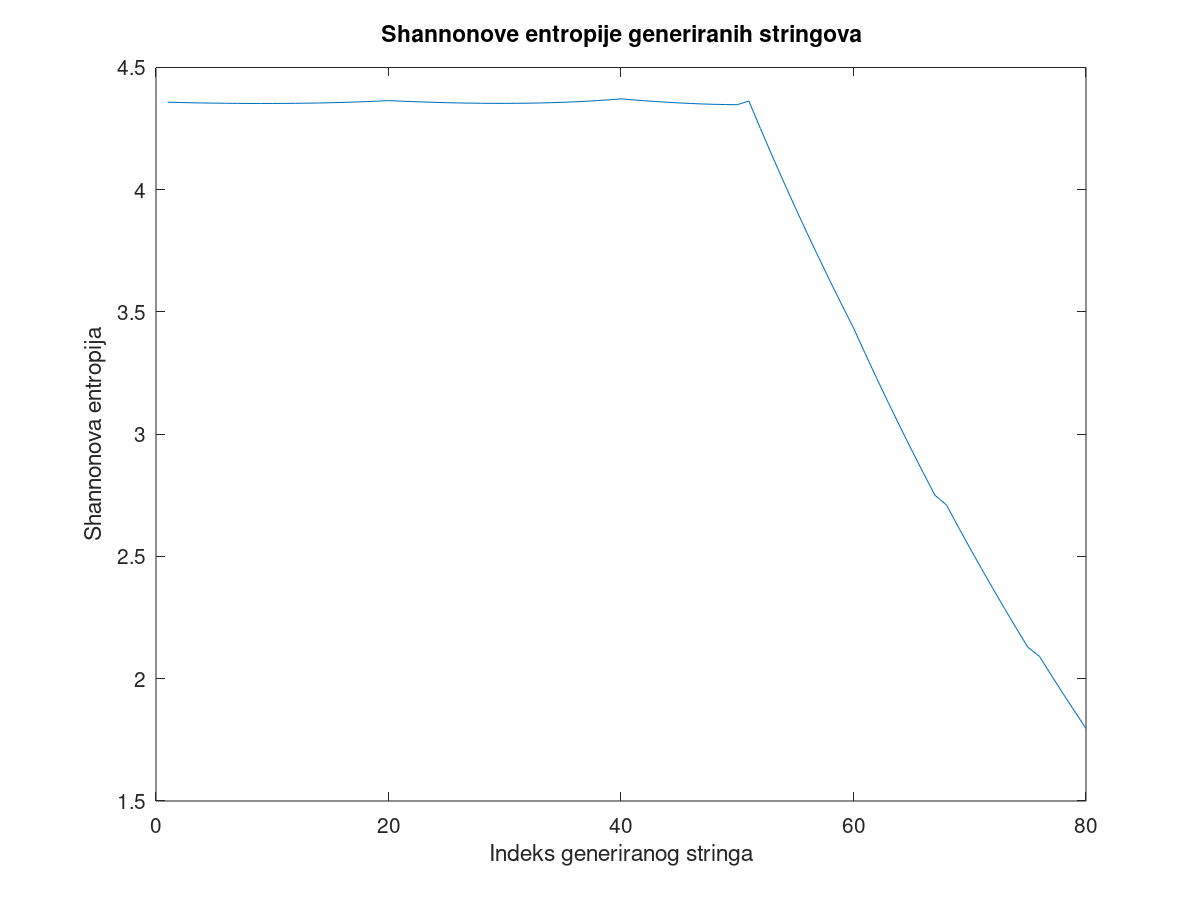
69 title('Usporedba Shannonove i Samarzijine entropije generiranih stringova');

Dakle, taj program generira stringove koji se sastoje od određenog broja slova *b*, od 1 do 80, te podjednako raspodjeljenih preostalih suglasnika engleske abecede. Nakon toga računa entropiju prvo po mom algoritmu, a zatim po Shannonovom algoritmu, te rezultate sprema u polja.

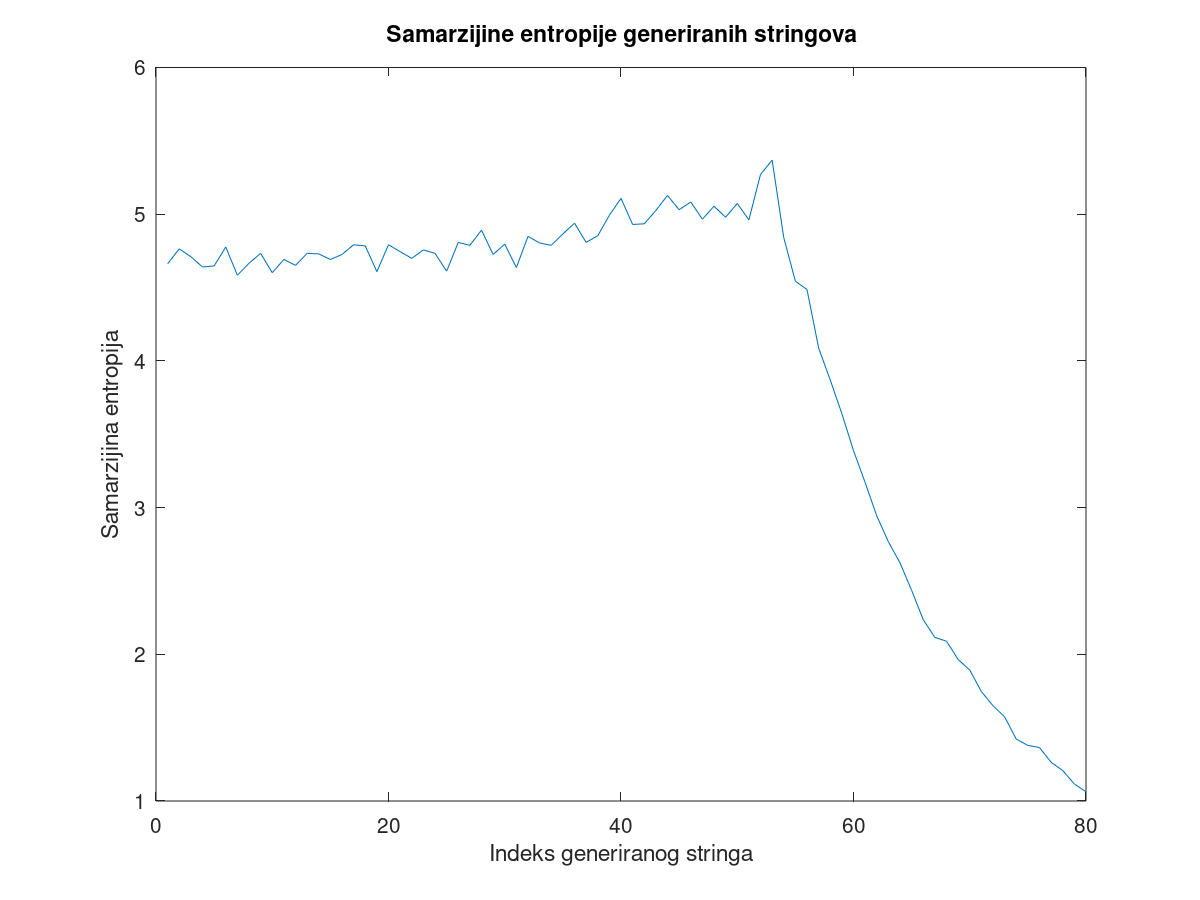
Program je, nakon što se oko dvije minute vrtio na mom računalu na Ubuntu Linuxu, ispisao ovakav rezultat:

  
Slika 2: Usporedba rezultata koje daje moj algoritam računanja entropije i rezultata koje daje Shannonov algoritam

Vidimo da postoji velika korelacija između rezultata koje daju moje algoritam i koje daje Shannonov algoritam. Međutim, kada je Shannonova entropija mala, moj algoritam daje još manji rezultat, i to za više od trećine. Recimo, Shannonova entropija generiranog stringa s najmanjom entropijom iznosi, kako se može očitati s grafikona, približno 1.8 bita po simbolu, dok moj algoritam tvrdi da je ona svega nešto više od 1 bit po simbolu. Kad je entropija oko 3.5 bita po simbolu (dakle, kao da ima 2^3.5=11.31 ravnomjerno raspodijeljenih simbola), moj algoritam i Shannonov algoritam daju gotovo isti rezultat. Rezultat koji daje moj algoritam raste približno linearno sa Shannonovom entropijom, sve dok Shannonova entropija ne bude nešto malo više od 4 bita po simbolu. Tada moj algoritam tvrdi da je entropija nešto više od 5, što je besmislen rezultat. Najveća entropija koju može imati bilo koja varijacija 21 znaka (koliko suglasnika ima u engleskoj abecedi) iznosi log2(21)=4.39 bita po simbolu. Ne znam kako moj algoritam može dati konzistentno davati takve rezultate. Na krajnjem desnom kraju krivulje, kada je Shannonova entropija najveća, moj algoritam tvrdi da entropija opada, isto ne znam zašto. Možemo vidjeti koje su entropije generiranih stringova, po oba algoritma:

  
Slika 3: Shannonove entropije generiranih stringova

Kao što možemo vidjeti na grafu, Shannonova entropija je približno konstantna (vjerojatno upravo već spomenutih 4.39 bita po simbolu) za sve stringove koji imaju slova *b* manje od nešto malo više od 50, gdje postoji mali luk prema gore, a onda približno linearno (možda malo sporije od linearnog, pogotovo pri desnom kraju grafikona) opadanje do stringa s 80 slova *b*, kada je entropija nešto više 1.75 bita po simbolu. Graf s rezultatima koje daje moj algoritam za svaki generirani string izgleda ovako:

  
Slika 4: Entropije koje daje moj algoritam za generirane stringove

Taj šum gdje su entropije približno konstantne očekivan je budući da je moj algoritam mekan. Shannonov algoritam nema neodređenosti u svojoj operaciji, pa je to čvrst algoritam. Onaj luk prema gore prije pada kod stringova koji imaju 50-ak slova *b* ovdje je znatno naglašeniji nego kod Shannonovog algoritma, ne znam zašto.

Kao sam uspio instalirati MatLab na svoje računalo, jedna od prvih stvari koje sam provjerio jest funkcionira li taj mog program u MatLabu, a ne samo u Octavi. I nije funkcionirao, MatLab je javljao hrpu sintaksnih pogrešaka. Evo kako sam taj program preuredio da funkcionira u MatLabu:

1 % Ovo je MatLabski program koji uspoređuje rezultate koje daje moj algoritam

2 % procjenjivanja entropije s rezultatima koje daje Shannonov algoritam.

3 suglasnici = 'bcdfghjklmnpqrstvwxyz';

4 testni\_stringovi=cell(100 **-** length(suglasnici) **+** 1, 1);

5 **for** koliko\_cemo\_staviti\_b\_ova = 100 **-** length(suglasnici) **+** 1 : **-**1 : 1

6 **for** i = 1 : koliko\_cemo\_staviti\_b\_ova

7 testni\_stringovi{koliko\_cemo\_staviti\_b\_ova} = [

8 testni\_stringovi{koliko\_cemo\_staviti\_b\_ova} 'b'

9 ];

10 **end**

11 **for** i = 1 : 100 **-** koliko\_cemo\_staviti\_b\_ova

12 testni\_stringovi{koliko\_cemo\_staviti\_b\_ova} = [

13 testni\_stringovi{koliko\_cemo\_staviti\_b\_ova} suglasnici(int32(**floor**((i **-** 1) **/** (100 **-** koliko\_cemo\_staviti\_b\_ova) **\*** (length(suglasnici) **-** 1) **+** strfind(suglasnici, 'c'))))

14 ];

15 **end**

16 **end**

17 samarzijine\_entropije = [];

18 shannonove\_entropije = [];

19 **for** i = 1 : length(testni\_stringovi)

20 str = testni\_stringovi{i};

21 samarzijine\_entropije = [samarzijine\_entropije samarzijina\_entropija(str)];

22 shannonove\_entropije = [shannonove\_entropije shannonova\_entropija(str, suglasnici)];

23 **end**

24 sgtitle('Usporedba Shannonove i Samarzijine entropije generiranih stringova');

25 subplot(1,2,1);

26 plot(shannonove\_entropije, samarzijine\_entropije);

27 xlabel('Shannonova entropija');

28 ylabel('Samarzijina entropija');

29 subplot(1,2,2);

30 plot(shannonove\_entropije);

31 hold on;

32 plot(samarzijine\_entropije);

33 xlabel('Broj b-ova u stringu');

34 ylabel('Entropija (bit/simbol)');

35 legend('Shannonova entropija', 'Samarzijina entropija');

36 function ret = shannonova\_entropija(str, suglasnici)

37 apsolutne\_frekvencije = [];

38 **for** i = 1 : length(suglasnici)

39 apsolutne\_frekvencije = [apsolutne\_frekvencije 0];

40 **end**

41 **for** i = 1 : length(str)

42 znak = str(i);

43 apsolutne\_frekvencije(strfind(suglasnici, znak)) = apsolutne\_frekvencije(strfind(suglasnici, znak)) **+** 1;

44 **end**

45 relativne\_frekvencije = apsolutne\_frekvencije **/** length(str);

46 ret = 0;

47 **for** relativna\_frekvencija = relativne\_frekvencije

48 **if** relativna\_frekvencija **>** 0

49 ret = ret **-** log2(relativna\_frekvencija) **\*** relativna\_frekvencija;

50 **end**

51 **end**

52 **end**

53 function ret = samarzijina\_entropija(str)

54 broj\_pokusaja = 10000;

55 broj\_pogodaka = 0;

56 **for** i = 1 : broj\_pokusaja

57 prvi = int32(**floor**(**rand**() **\*** length(str) **+** 1));

58 drugi = int32(**floor**(**rand**() **\*** length(str) **+** 1));

59 **if** str(prvi) **==** str(drugi)

60 broj\_pogodaka = broj\_pogodaka **+** 1;

61 **end**

62 **end**

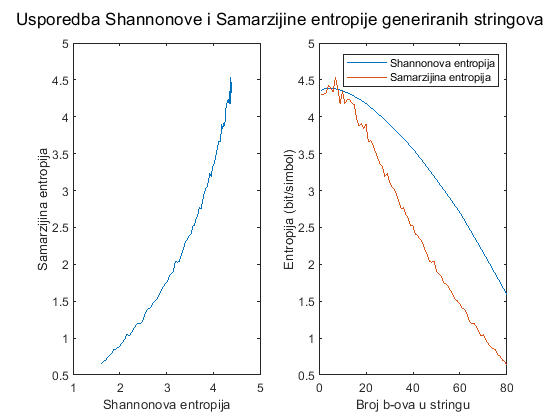
63 omjer\_pogodaka = broj\_pogodaka **/** broj\_pokusaja;

64 ret = **-**log2(omjer\_pogodaka);

65 **end**

On je ispisao ovakav rezultat:

Kako je moguće da taj MatLabov program i ekvivalentan Octavin program daju toliko različite rezultate? I to konzistentno entropija u Octavi počinje opadati tek nakon negdje 50 b-ova u stringu, dok u MatLabu počinje opadati prije 20, koliko god ja puta pokrenem taj program. Nije riječ o slučajnosti, ima nešto dublje. Kad sam to još malo istražio, ispostavilo se da je način na koji je Octave mogao parsirati taj moj program što se vrti u Octavi, a MatLab ga nije mogao, bio to što je Octave na kraj 11. reda automatski umetao točka-zarez da program bude sintaktički ispravan, a da me nije upozorio na grešku. Slično kao što i JavaScript enginei umeću točka-zarez, a da o tome ne upozore, i time uzrokuju greške u programima kojima je teško ući u trag. Tako je Octaveski program pogrešno generirao testne stringove. Sada više ovisnost rezultata koje daje moj algoritam i rezultata koje daje Shannonov algoritam ne izgleda baš linearna, više kao paraboloidna ili nešto slično (vidimo da brzina rasta moje entropije raste sa Shannonovom entropijom, da je prva derivacija krivulje njihove ovisnosti rastuća). Ne znam kako to objasniti.

Slika 5: Izlaz iz MatLabskog programa

**Zaključak**

Svaka znanost počne kao mekana znanost, i mnogi je pokušaju otvrdnuti. Medicina je većinu ljudske povijesti bila mekana znanost i gora nego beskorisna, i Ignaz Semmelweis bio je u pravu kad je tražio da se medicina manje oslanja na osobno iskustvo i tada općeprihvaćene činjenice, a više na statistiku i eksperimentalnu znanost. Međutim, mnogi ljudi koji pokušaju otvrdnuti neku znanost čine je još mekšom. Anatoly Fomenko je, recimo, ruski matematičar koji tvrdi da je moguće statistički dokazati da je čitava antička povijest lažirana od strane nekakve goleme urote u 13. stoljeću. Jesam li ja za lingvistiku ono što je Ignaz Semmelweis za medicinu, ili sam ono što je Anatoly Fomenko za povijest? Procijenite sami.

1. Analitički jezik je onaj koji koristi malo ili nimalo morfologije da označi ulogu riječi u rečenici. Aglutinativni jezik na riječi naslaguje morfeme od kojih svaki ima određeno značenje. Fuzionalni jezik ima morfeme koji označuju više stvari odjednom. Primjer fuzionalnog jezika jest latinski jezik. U njemu dativ jednine od riječi *čovjek* glasi *homin-i*, a dativ množine glasi *homin-ibus*. Tu morfem *-i* znači i *dativ* i *jednina*, i ne može se rastaviti na manje morfeme, kao što se ne može ni *-ibus*. Nominativ jednine jest *homo*, a nominativ plurala je *homines*. Ablativ jednine je *homin-e*, a ablativ množine je isti kao i dativ množine, *homin-ibus*.Primjer aglutinativnog jezika jest mađarski jezik. U njemu dativ jednine od riječi za *čovjek* glasi *ember-nek*, a dativ plurala jest *ember*-*ek*-*nek*. U njemu se, dakle, nastavak za dativ plurala sastoji od nastavka za plural i nastavka za dativ. I isto, u načelu, vrijedi za sve padeže. Nominativ od imenice koja znači *čovjek* jest *ember*, a nominativ plurala je *ember-ek*. Ablativ singulara jest *ember-től,* dok je ablativ plurala *ember-ek-től*. Primjer analitičkog jezika je engleski jezik, koji imenice i pridjeve ne deklinira po padežima, nego umjesto padeža koristi prijedloge. Engleski za *čovjeku* je *to a/the human*, a *ljudima* je *to (the) humans*. [↑](#footnote-ref-2)
2. Koliko sam našao na internetu, Cadel Watson napravio je nešto slično, ali, koliko sam vidio, nije napravio nikakvu validaciju: https://github.com/kdelwat/Onset [↑](#footnote-ref-3)
3. https://flatassembler.github.io/etymologist.html [↑](#footnote-ref-4)
4. Podatak preuzet s Cambridgevog rječnika: https://dictionary.cambridge.org/dictionary/english/overload [↑](#footnote-ref-5)
5. O programskom jeziku koji sam napravio možete čitati na mom blogu: https://flatassembler.github.io/AEC\_specification.html [↑](#footnote-ref-6)
6. https://dictionary.cambridge.org/dictionary/english/mnemonic?q=mnemonics [↑](#footnote-ref-7)
7. https://dictionary.cambridge.org/dictionary/english/psychology [↑](#footnote-ref-8)
8. O termama na Visu mnogo je pisao Branimir Gabričević. Diskusiju o tome odakle su se napajale te terme možete čitati ovdje na stranici 28: https://core.ac.uk/download/pdf/33286142.pdf [↑](#footnote-ref-9)
9. Vidite taj tekst Branimira Gabričevića, stranica 7. [↑](#footnote-ref-10)
10. 203. stranica Ivšićinog doktorata. [↑](#footnote-ref-11)