Teorijski model računalnih virusa

Ivančić, Antonio

Master's thesis / Diplomski rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:614005

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Antonio Ivančić

TEORIJSKI MODEL RAČUNALNIH VIRUSA

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Mladen Vuković

Zagreb, 2016.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Za Jakova. Za mamu.

Sadržaj

Uvod

Suvremena je računalna mreža nevjerojatno velika. Sa sustavom takve veličine nužna je pojava problema. Jedan od njih svakako je i pojava nove metodologije u zločinu koji se može počiniti. Sprječavanje svake vrste „cyber kriminala" nužnost je s obzirom na stvorenu informacijsku mrežu. Kako to obično biva, za bolje je razumijevanje stvarnosti dobro imati teorijski model. On omogućuje bolje shvaćanje procesa koji se odvijaju, a uloga mu može biti i prediktivna. S tim u vidu, analiza, odnosno teorijska osnova od velike su važnosti.

Temelji su teorije izračunljivosti, teorije složenosti i općenito teorije računarstva udareni danas davne 1936. godine kada je Alan Turing dao definiciju strojeva koje danas njemu u čast nazivamo Turingovim strojevima. Ne treba ni zaboraviti spomenuti kako je u otprilike isto vrijeme nezavisnu definiciju upravo spomenutih strojeva dao i američki matematičar i logičar poljskoga podrijetla, Emil Post. Unatoč tome što danas postoje modeli izračunavanja koji su ekvivalentni modelu kojeg tvore Turingovi strojevi (primjerice, RAM strojevi), skrenimo pažnju na dva aspekta koji su uzrokom da se ovaj model standardno koristi (pažnju ovom modelu posvećujemo s obzirom na to da se glavni dio teorije kojom će se ovaj rad baviti temelji upravo na njemu):

- Jednostavnost
  Kao teorijski model, strojevi su iznimno jednostavni u smislu da je konkretni promatrani stroj uvijek „konačan". Sve što nam je potrebno kako bismo znali gdje će stroj biti u sljedećem koraku jedan je simbol (element abecede dotičnog stroja) i broj, odnosno jedinstvena oznaka stanja u kojem se stroj nalazi. Naime, umjesto skupa $\{q_1, q_2..., q_n\}$ jednostavno promatramo skup $\{1, 2, ..., n\}$. Ovo pojednostavljuje formalne dokaze, odnosno broj slučajeva koje je potrebno promatrati. Usporedimo ovo samo s RAM strojem: sljedeći korak može biti *bilo koja* od dopuštenih operacija, koja ima pristup *bilo kojem* registru.

- Vremenska i prostorna složenost
  Samo su dva modela izračunavanja razvijana praktički paralelno s onim Turingovih strojeva: Churchov λ-račun i Kleenejev model rekurzivnih funkcija (μ-račun). No, oba su ova modela mnogo kompleksnija od modela kojim ćemo se mi koristiti u smislu vremenske i prostorne složenosti. Takoreći, ovi su modeli *preapstraktni* kako bi ih se direktno povezalo s više realnim modelima strojeva. S druge strane, vidjeli smo (vidi [9] za detalje) da se dani termini - vremenska i prostorna složenost - lako dadu izvesti iz modela u kojem je baza Turingov stroj.

Ipak, mnogi su modeli od tih godina stvoreni i aktivno se koriste; jedna je od očitih mana Turingovih strojeva činjenica da njihova konstrukcija uzima mnogo vremena.

U prvom dijelu ovog rada razmatrat ćemo teorijsku osnovu staničnih automata koju je razvio von Neumann, a u kojoj su implicitno sadržani svi potrebni alati za formalizaciju koja nam slijedi u drugom poglavlju. Svrha prvog dijela jest, dakle, dobiti okvir i osjećaj za rezultate koji će se formalno dokazivati u Cohenovoj formalizaciji. Kako dio kojim se prvo poglavlje bavi nije glavna tema ovog rada, bit će predstavljene osnovne definicije i nama najznačajniji rezultati, bez pratećih strogih matematičkih dokaza. Najvažniji dio tog prvog poglavlja svakako je koncept umnažanja, odnosno promjene određenog stanja (*konfiguracije*) kroz vrijeme uz određene uvjete. Drugi dio predstavlja srž ovog rada, a to je formalizacija koncepta računalnog virusa koju nudi Cohen. Najprije se upoznajemo s definicijama i pojmovima koji se koriste, a pomoću kojih predstavljamo osnovne rezultate teorije. Nakon osnovnih teorema i pripadajućih dokaza, dokazujemo glavni teorem ovog rada, a to je neodlučivost virusa. To je vrlo zanimljiv teorijski rezultat kojim je matematički otklonjena mogućnost postojanja savršenog antivirusnog programa. Konačno, rad završavamo kratkim izlaganjem rada Leonarda Adlemana koji se tiče virusa. Leonard Adleman, kako kaže Cohen, i zaslužan je za uvođenje pojma *virusa*. Perspektiva koju on zauzima pri proučavanju ove teorije temelji se na rekurzivnim funkcijama, čime poopćuje rad svog doktoranda Cohena.

Poglavlje 1

Stanični automati

1.1 Uvod

Teorijski model izračunavanja temeljen na Turingovim strojevima srž pronalazi u sljedećem problemu:

Problem. *Neka je f funkcija. Postoji li Turingov stroj M takav da za svaki $x \in \mathcal{D}(f)$ vrijedi $M(x) = f(x)$?*

Aspekt kojim se početni istraživači-teoretičari nisu bavili jest: postoji li neki program za Turingov stroj koji se na neki način, neformalno sada govoreći, može mijenjati?

1.1.1 Rekurzivne funkcije i virusi

Unatoč tome, pojavio se rezultat koji je u uskoj vezi s teorijom virusa (tada se, u tom kontekstu, toga nije bilo svjesno). Rezultat je u literaturi poznat pod nazivom „Teorem o fiksnoj točki". Jasan dokaz može se pronaći u [9], stoga ćemo ga ispustiti.

Teorem 1.1.1. *Za svaku unarnu parcijalno rekurzivnu funkciju F postoji prirodan broj e takav da vrijedi:*

$$\{e\} \simeq \{F(e)\}.$$

Na koji način, s obzirom na kontekst, interpretirati dani rezultat? Ovo u svojoj biti znači da imamo dva različita programa koja obavljaju iste operacije nad ulaznim podatkom. Koristeći se jezikom svojstvenim za programersku struku, govorimo o dva različita *izvorna koda*. Pretpostavimo li da je dana funkcija identiteta (koja je totalna rekurzivna funkcija, a stroj koji je izračunava je prazan stroj u smislu da s

danim ulazom ne radi ništa), rezultat nas upućuje na pojam o trivijalnom umnažanju. Stavimo li se u kontekst izvornog koda, govorimo o promjeni samog izvornog koda. U slučaju da, pak, dana funkcija nije identita, govorimo o jednostavnom rezultatu koji upućuje na moćnu mogućnost programa: stvarnu promjenu izvornog koda bez utjecaja na vlastitu semantičku vrijednost, odnosno bez utjecaja na ukupni rezultat operacija koje izvršava nad ulaznim podatkom. Sada je bitno uočiti kako je teorem dao lijepu osnovu za potencijalno opasno ponašanje programa koje želimo prepoznati temeljem sadržaja koda. Vrlo je zanimljivo za vidjeti kako je teorijski model predvidio osobine programa (mijenjanje vlastitog koda) prvi put tek ostvarenog (kao virusa) godine 1990., imena „1260".

1.2 Stanični automati - matematički model von Neumanna

Početci teorije staničnih automata, čiju ćemo formalno definiciju imati prilike vidjeti nešto kasnije, dani su nam s radovima mađarsko-američkog matematičara Johna von Neumanna. Motivaciju za svoj rad pronašao je u želji za stvaranjem modela koji bi na jednostavan način opisivao evoluciju nekog sustava, u njenom biološkom smislu. Želimo li svrhu odrediti specifičnije, reći ćemo da se htjelo, u terminima modela koji se tek trebao stvoriti, opisati pojam umnažanja (u engleskoj literaturi termin koji se koristi jest *self-reproduction*).

Određivanja modela u konačnici se svelo na određivanje konačnog skupa (jednostavnih) pravila iz kojeg bi, uvjetno rečeno, jednostavan sustav mogao postati kompleksniji. Najbolji matematički model koji opisuje upravo opisanu problematiku predstavljen nam je u obliku staničnih automata. Ovaj model nadilazi na prvi pogled čisto akademsku ulogu i svoju primjenu pronalazi u, očekivano, biologiji, ali i fizici, pa čak i logistici. Poznat je tako Nagel-Schekenbergov model jednodimenzionalnog staničnog automata koji opisuje problem zagušenja prometnica. Više detalja možete vidjeti u [4].

Mnoštvo je primjera ovih strojeva; pravila koja ih određuju uvjetovana su problemima koje pokušavamo opisati. Ipak, zajedničke karakteristike su sljedeće:

- broj ćelija - sustav se sastoji od jedno-, dvo- ili trodimenzionalnih ćelija čiji je broj konačan, ili najviše beskonačno prebrojiv
- homogenost - sve su ćelije međusobno jednake
- konačnost stanja - svaka se ćelija nalazi u jednom od konačno mnogo stanja

- susjedstvo - stanje svake ćelije uvjetovano je njenim stanjem i stanjem ćelija *pored*
- promjena - u vremenskom trenutku t svaka ćelija mijenja svoje stanje na temelju svog stanja i stanja susjedstva

Formalizirajmo sada predočeni koncept dvjema definicijama, i spomenimo kako ćemo pojmove ćelija i stanica koristiti izmjenično, i smatramo ih ekvivalentnima.

Definicija 1.2.1. *(Konačni automat)*
Konačni automat *uređena je petorka* (q_0, Q, F, X, f)*, gdje je je* q_0 *istaknuto stanje iz konačnog* skupa stanja Q*,* $F \subset Q$ *konačan skup kojeg nazivamo* skupom prihvatljivih stanja*,* X *konačan skup kojeg nazivamo* abecedom stroja*, a funkciju* $f : Q \times X \to Q$ funkcijom prijelaza.

Domena se funkcije iz gornje definicije dade proširiti na skup svih riječi skupa X, u oznaci X^*, na način $f(q, w) = f(f(q, w_1), a)$, gdje je $w \in X^*, w_1 \in X^*$ i $a \in X$, pri čemu $w = w_1 a$. Za konkretni ćemo stroj koristiti jednostavniju notaciju (V, v_0, f), dakle riječ je o uređenoj trojci gdje je V konačan skup stanja, v_0 istaknuto stanje iz V, a koje nazivamo *početnim stanjem*, te f analogno kao gore. Uz postojeću notaciju, $Q = V^n$ nazivamo memorijom stroja.

Ovaj koncept dobro nam je poznat na temelju obrađene hijerarhije jezika (Chomsky) i strojeva na kolegiju „Interpretacija programa". Stoga za više detalja i primjere vidi [5].

S obzirom na činjenicu da je promatrani model von Neumannov, definicija automata koja slijedi dana je u kontekstu dvodimenzionalnog prostora.

Definicija 1.2.2. *(Stanični automat)*
Stanični automat*, ili* stanični prostor*, nad prostorom* $\mathbb{N} \times \mathbb{N}$ *definiramo kao uređenu trojku* $((\delta_n), g, (V, v_0, f))$*, gdje je redom:*

1. (δ_n) *konačan niz u* $\mathbb{N} \times \mathbb{N}$*, pri čemu je* $\delta_1 = (0, 0) = 0 \in \mathbb{N} \times \mathbb{N}$*,* $n \in \mathbb{N} \setminus \{0\}$.
2. g*:* $\mathbb{N} \times \mathbb{N} \to 2^{\mathbb{N} \times \mathbb{N}}$ funkcija susjedstva *definirana sa:*

$$g(\alpha) = \{\alpha + \delta_1, \alpha + \delta_2, \ldots, \alpha + \delta_n\},$$

   gdje se zbrajanje vrši po točkama.

3. (V, v_0, f) konačni automat *gdje je* V skup stanja ćelija (stanica)*,* v_0 početno istaknuto stanje*, i funkcija* f *je* funkcija prijelaza *s* V^n *na* V*, pri čemu uvodimo:*

$$f(v_0, \ldots, v_0) = v_0.$$

Za zadani stanični automat i svaki $t \in \mathbb{N}$ definiramo funkciju $h^t : \mathbb{N} \times \mathbb{N} \to V^n$ na sljedeći način:

$$h^t(\alpha) = (v^t(\alpha), v^t(\alpha + \delta_2), \ldots, v^t(\alpha + \delta_n)))),$$

gdje je sa $v^t(\alpha)$ dano stanje ćelije α u trenutku t.

Gornja restrikcija zapravo hoće reći kako se stanje ćelije neće promijeniti ako je ona sama u nultom stanju, i svi njeni susjedi su u nultom stanju. Funkcija nam f na temelju stanja daje novo stanje - prijelaz.

Intuitivno, stroj možemo zamišljati kao jednostavnu ploču, pri čemu je svaka ćelija te ploče određena svojim koordinatama (pozicija) i bojom (stanje), u odnosu na neko proizvoljno odabrano ishodište.

Nakon ovih dviju definicija, prirodno se pitamo na koji način opisati cjelokupni automat u trenutku t, odnosno koje ćelije „čine" automat? Uočimo, zanimaju nas one ćelije koje se ne nalaze u stanju v_0.

Definicija 1.2.3. *(Konfiguracija staničnog automata)*
Konfiguracija *ili* globalno stanje danog staničnog automata *je svaka funkcija $c : \mathbb{N} \times \mathbb{N} \to V$ takva da vrijedi da je skup*

$$supp(c) = \{\alpha \in \mathbb{N} \times \mathbb{N} : c(\alpha) \neq v_0\}$$

konačan.

Reći ćemo da funkcija c ima *konačni nosač s obzirom na* v_0. S obzirom na pojam konfiguracije, uvodimo i pojam podkonfiguracije.

Definicija 1.2.4. *(Podkonfiguracija staničnog automata)*
Funkciju c' nazivamo podkonfiguracijom konfiguracije *c ako vrijedi:*

$$c\big|_{supp(c')} = c'\big|_{supp(c')}.$$

Sada smo u stanju opisati „ponašanje" staničnog automata kroz vrijeme.

Definicija 1.2.5. *(Globalna funkcija prijelaza)*
Neka je $S = ((\delta_n), g, (V, v_0, f))$ stanični automat, i neka je $\mathcal{C}_S = \{c_0, c_1, \ldots\}$ skup svih njegovih konfiguracija, pri čemu je c_t konfiguracija staničnog automata u trenutku t. Globalnu funkciju prijelaza automata *S definiramo kao funkciju $F_S : \mathcal{C}_S \to \mathcal{C}_S$, pri čemu je:*

$$F_S(c_t)(\alpha) = f(h^t(\alpha)), \forall \alpha \in \mathbb{N} \times \mathbb{N}.$$

U narednim ćemo razlaganjima umjesto staničnog automata $S = ((\delta_n), g, (V, v_0, f))$ kratko pisati samo stanični automat S.

Označimo li inicijalnu (zadanu) konfiguraciju nekog stroja s c_0, uz pomoć ove funkcije dobivamo niz konfiguracija kojeg nazivamo *propagacija*. Dakle, riječ je o „promjeni" automata (stanja ćelija) kroz vrijeme s obzirom na početnu konfiguraciju. Stavljamo naglasak na pojam s obzirom na sličnost pojma evolucije koju daje Cohen, a koja se u punoj općenitosti može opisati kao promjena kroz vrijeme. Dakle, za c_0 imamo propagaciju:

$$c_0, c_1, \ldots, c_t, \ldots,$$

pri čemu je

$$c_{t+1} = F_S(c_t), \ \forall t \in \mathbb{N},$$

odnosno

$$c_0, F_S(c_0), F_S^2(c_0), \ldots.$$

Potonja notacija naglašava upravo tu evoluciju u ovisnosti na inicijalno stanje, odnosno konfiguraciju stroja. Sa $F_S^n(x)$ označili smo djelovanje funkcije F_S na x, i to n puta, to jest $\underbrace{F_S(F_S(\ldots(F_S}_{n}(x))\ldots)$. Jasno, ne ponašaju se sve konfiguracije jednako. Čak štoviše, neke imaju posebna svojstva.

Definicija 1.2.6. *(Svojstva konfiguracije)*
Neka je S stanični automat, $c \in \mathcal{C}_S$ neka njegova konfiguracija i F_S pripadajuća globalna funkcija prijelaza stroja S.

1. *Kažemo da je c* pasivna *ako je $F_S(c) = c$.*
2. *Kažemo da je c* potpuno pasivna *ako je svaka podkonfiguracija c' od c pasivna.*
3. *Kažemo da je c* stabilna *ako postoji $t \in \mathbb{N}$ takav da je $F_S^t(c)$ pasivna.*
4. *Neka je c_τ neka konfiguracija. Kažemo da je c_τ* translacija *konfiguracije c ako postoji $\tau \in \mathbb{N} \times \mathbb{N}$ tako da vrijedi $c_\tau(\alpha) = c(\alpha - \tau)$.*

Kako je svaka konfiguracija svoja trivijalna podkonfiguracija, odmah dobivamo tvrdnju sljedeće leme.

Lema 1.2.7. *Neka je $c \in \mathcal{C}_S$ neka konfiguracija staničnog automata S i F_S pripadajuća globalna funkcija prijelaza. Ako je c potpuno pasivna, onda je c i pasivna.*

Vrijedi li obrat? Ne. Naime, ćelije koje čine susjedstvo ćelije α u podkonfiguraciji c' (od c) ne moraju biti iste one koje čine susjedstvo ćelije α u konfiguraciji c (jer neke ćelije iz c ne moraju biti uključene u podkonfiguraciju c' upravo zato jer je

c' podkonfiguracija). Stoga njen prijelaz, s obzirom na funkciju prijelaza, odnosno globalnu funkciju prijelaza, ne mora biti isti, odnosno ne mora biti $F_S(c') = c'$.

Uočimo i da je c translacija od c_τ ako je c_τ translacija od c. Naime:

$$c(\alpha) = c_\tau(\alpha + \tau) = c_\tau(\alpha - (-\tau)).$$

Postoji efektivni algoritam koji određuje je li određena konfiguracija (potpuno) pasivna ili ne. Ipak, ne postoji efektivan algoritam koji će ispitati je li određena konfiguracija stabilna. Čak štoviše, može se pokazati kako se iz halting problema Turingova modela dade izvesti problem stabilnosti konfiguracije von Neumannova modela. Dokaz se tog rezultata može naći u [6].

1.2.1 Umnažanje

Nakon kratkog izlaganja von Neumannovog modela u stanju smo konačno formalno definirati umnažanje.

Kako bismo znali u kojem se stanju nalaze ćelije proizvoljnog staničnog automata, moramo znati na koji je način definirano susjedstvo jer stanje svake ćelije ovisi upravo o njenom susjedstvu. Stoga je za svaki automat ključno definirati funkciju g.

Primjer 1.2.8. *Zadajmo niz* (δ_n), *odnosno definirajmo funkciju* g.

$$(\delta_n) = (\delta_1, \delta_2, \delta_3, \delta_4, \delta_5) = ((0,0), (0,1), (1,0), (0,-1), (-1,0))$$

. Tada je:

$$g(\alpha) = \{\alpha, \alpha + (0,1), \alpha + (0,-1), \alpha + (1,0), \alpha + (-1,0)\}.$$

Uz oznaku $\alpha_i = \alpha + \delta_i$, *za* $i = 2, ..., 5$, *opisano se susjedstvo ćelije* α *grafički jednostavno prikazuje:*

	α_2	
α_5	α	α_3
	α_4	

Pretpostavimo li da smo (neku) ćeliju α *jednoznačno odredili koordinatama* $(0,0)$, *tada bi iz zadanog niza* (δ_n) *i funkcije* g *uz uvedene oznake imali da je drugi susjed (prvi je susjed sama ćelija, po definiciji) ćelije* α *dan s* $\alpha + \delta_2 = \alpha + (0,1) = (0,1) = \alpha_2$. *Dakle, u kontekstu koordinata pomaknuli smo se jedno mjesto gore - baš kao što se vidi na slici kad se pogleda ćelija* α_2 *u odnosu na* α. *Potpuno analogno rezoniranje vrijedi i za ostale ćelije koje čine susjedstvo od* α, *dakle za ćelije* α_3, α_4 *i* α_5, *a koje su na slici zatamnjene.*

Postoje mnoge druge vrste susjedstava. Jedno od najpoznatijih je svakako Mooreovo susjedstvo koje se koristi u Conwayovoj „Igri života". Uz

$$(\delta_n) = (\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_6, \delta_7, \delta_8, \delta_9) =$$

$$= ((0,0), (-1,1), (0,1)(1,1), (1,0), (1,-1), (0,-1), (-1,-1), (-1,0))$$

te analogne oznake za α_i i pojašnjenja kao gore, grafički dobivamo:

α_2	α_3	α_4
α_9	α	α_5
α_8	α_7	α_6

Za proučavanje koncepta umnažanja (*self-reproduction*) i, općenitije, pojavu neke konfiguracije, potrebno je uopće ustvrditi može li se određena konfiguracija, s obzirom na globalnu funkciju prijelaza i inicijalnu konfiguraciju, dostići. S obzirom na to, imamo sljedeću definiciju.

Definicija 1.2.9. *(Konstrukcija konfiguracije)*
Neka je S stanični automat, $c, c' \in \mathcal{C}_S$ dvije njegove konfiguracije i F_S pripadajuća globalna funkcija prijelaza. Kažemo da konfiguracija c gradi (konstruira) *konfiguraciju c' ako postoji $U \subsetneq \mathbb{N} \times \mathbb{N}$ i trenutak $t \in \mathbb{N}$ tako da je $supp(c) \cap U = \emptyset$ i $c' = F_S^t(c)\big|_U$.*

Sada definiramo umnažanje u von Neumannovom smislu.

Definicija 1.2.10. *(Umnažanje u von Neummanovom smislu)*
Neka je S stanični automat, $c \in \mathcal{C}_S$ neka njegova konfiguracija i F_S pripadajuća globalna funkcija prijelaza. Kažemo da se konfiguracija c umnaža *ako postoji translacija c_τ od c takva da c konstruira c_τ.*

Sama je definicija vrlo jasna i intuitivna. Konfiguracija se umnaža ako se ona u nekom kasnijem trenutku (od onog kojeg promatramo) ponovno pojavi „na ploči" (intuitivno, spomenuli smo na početku, stanični automat zamišljamo kao ploču; konfiguraciju u tom smislu možemo zamišljati kao one ćelije koje su „obojane", tj. koje se ne nalaze u početnom stanju). Objasnimo dodatno o čemu se radi pomoću jednostavnog, gotovo trivijalnog primjera.

Primjer 1.2.11. *Promatramo stanični automat sa samo dva stanja. Dakle, $V = \{0,1\}$. Kao početno stanje biramo stanje 0, odnosno stavljamo $v_0 = 0$. Funkciju prijelaza definramo na sljedeći način:*

$$f(v_1, v_2, v_3, v_4, v_5) = \begin{cases} 1, & za\ v_5 = 1, \\ v_1, & inače. \end{cases}$$

Drugim riječima, ćelija α ostaje u stanju u kojem jest ako joj je prvi susjed lijevo u stanju 0*, a ako je prvi susjed lijevo u stanju* 1*, ona sama prelazi u stanje* 1*. Ilustracije radi, prikažimo jednu trivijalnu konfiguraciju i njeno umnažanje. Pritom ćemo ćeliju označavati stanjem u kojem se nalazi. Ako je ćelija u početnom stanju, ne prikazujemo je osim u slučaju da nije od važnosti za sljedeći korak.*

Pretpostavimo da je inicijalna konfiguracija c_0 takva da je $|supp(c_0)| = 1$, odnosno postoji samo jedna ćelija koja se nalazi u stanju 1*. Na „ploči" je, dakle, u vremenskom trenutku $t = 0$ situacija sljedeća:*

	1	*0*	

No, već u sljedećem trenutku, a temeljem funkcije prijeaza f, situacija je ovakva:

1	*1*	*0*	

Očito, konfiguracija c_0 se umnožila. Konkretno, c_0 konstruira svoju translaciju c_τ, gdje je $\tau = (-1, 0)$. Naime, $c_\tau(\alpha) = c(\alpha - (-1, 0)) = c(\alpha + (1, 0))$, to jest c_τ je konfiguracija c_0 pomaknuta jedno mjesto u desno.

Nakon uvođenja definicije virusa, a poglavito nakon leme 2.2.13, sličnost će između ovih koncepata biti očita. Naime, u toj se lemi dokazuje kako je, s obzirom na definiciju, svaki niz znakova koji se kopira na traci Turingova stroja upravo - virus.

Poglavlje 2

Virusni skup

2.1 Uvod

U prošlom su poglavlju izloženi koncepti koji u sebi sadržavaju sve nužne alate za konstrukciju računalnog virusa, no oni se nisu pojavili sve do sedamdesetih godina prošloga stoljeća. Iako je virusnih programa bilo, njihov broj nije bio značajan prije druge polovice 80-ih godina prošlog stoljeća kada Cohen objavljuje svoju doktorsku disertaciju (1986.). Štoviše, sam pojam *virus* prvi se put pojavljuje upravo u tom radu Fredericka Cohena; do tada se za program tog tipa upotrebljavalo ime „umnažajući program" („self-replicating program").

Kada govorimo o virusima u najosnovnijem smislu, govorimo o nizu simbola u memoriji stroja, i to neovisno o obliku. Može se raditi o glavnoj memoriji, registrima, disku, nekoj vrpci, ... Ono što taj niz simbola čini elementom sada nam još nepoznatog (virusnog) skupa V sposobnost je da u spomenutoj memoriji stvori novi element koji je također unutar skupa V. U skladu s prethodno rečenim, na um najčešće padaju najjednostavniji oblici virusa - singleton virusni skup (skup samo s jednim elementom). Riječ je, dakle, upravo o umnažajućem programu. Ipak, to nije jedina mogućnost. Primjerice, mnogi virusi pojavljuju se kao rezultat evolucije drugog virusa, ili pak kao rezultat evolucije većeg broja drugih. Ovo je važno jer problem uklanjanja ili detekcije virusa čini mnogo težim nego što bi bilo u slučaju da je dopušteno postojanje samo onih virusa koji umnažaju same sebe, pritom se ne mijenjajući. Iz ovih neformalnih razmatranja izvedimo široko prihvaćeno intuitivno poimanje računalnog virusa.

(Intuitivna definicija virusa)
Virus *je konačan niz simbola koji u određenoj okolini (stroju) ima mogućnost mijenjanja drugih nizova simbola, uključujući i samog sebe.*

2.2 Cohenova formalizacija

2.2.1 Pojmovi i definicije

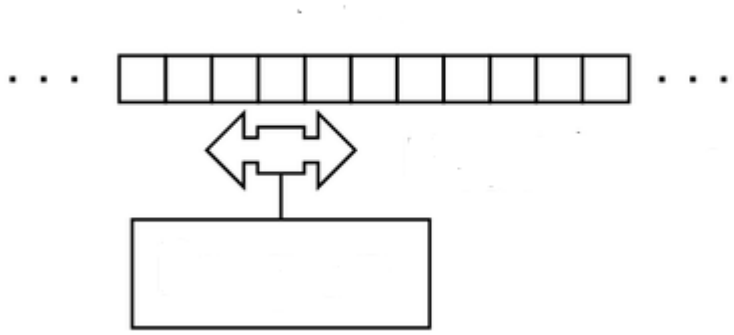
Kao što smo na samom početku naglasili, za formalno razmatranje virusa, odnosno virusnog skupa, nužan nam je model Turingova stroja. Ipak, Cohen ovaj model oblikuje nešto drugačije nego što smo na to navikli. Naime, on posebnu pažnju pridodaje samom koraku Turingova stroja, razlažući ga uz pomoć zasebno definiranih funkcija. Ovo je važno jer predstavlja početnu točku za daljnja razmatranja. Iz toga razloga dajemo drugu verziju definicije pojma Turingova stroja.

Definicija 2.2.1. *(Turingov stroj)*
Turingov stroj M *uređena je petorka* $(S_M, I_M, O_M, N_M, D_M)$*, pri čemu redom imamo:*

- S_M *je konačan* skup stanja *stroja dan sa* $\{s_0, s_1, \ldots, s_n\}$*,*
- I_M *je konačan niz simbola dan sa* $\{i_0, i_1, \ldots, i_n\}$ *kojeg nazivamo* abecedom stroja*,*
- $O_M : S_M \times I_M \to I_M$ *je* funkcija izlaza*,*
- $N_M : S_M \times I_M \to S_M$ funkcija prijelaza*,*
- $D_M : S_M \times I_M \to d$ funkcija pomaka*,*

gdje je $d = \{-1, 0, 1\}$.

Stroj, naravno, zamišljamo klasično. Riječ je o dvostrano neograničenoj traci podijeljenoj na polja. Svakom polju dodijeljen je indeks, pri čemu se „ishodište", odnosno polje s indeksom 0 izabire proizvoljno. Na svakom od polja nalazi se jedan od simbola iz I_M (uključujemo prazan simbol). Nadalje imamo glavu stroja koja se pomiče nad trakom i čita, odnosno upisuje znakove na traku. Konačno, imamo glavni dio stroja koji je dan pomoću gore definiranih funkcija i koji diktira kako stroj radi. Opisani model možemo i lako grafički prikazati:



Kako je spomenuto, naglasak se postavlja na same korake stroja M. Zato opisujemo sljedeće tri takozvane vremenske funkcije. Navodimo ih redom:

1. Funkcija *vrijeme-stanje* $\$_M : \mathbb{N} \to S_M$ koja koraku pridružuje stanje u kojem se stroj nakon koraka nalazi.

2. Funkcija *sadržaj ćelije-vrijeme* $\square_M : \mathbb{N} \times \mathbb{N} \to I_M$ koja koraku i indeksu ćelije pridružuje simbol koji se nalazi na ćeliji s tim indeksom nakon koraka.

3. Funkcija *vrijeme-ćelija* $P_M : \mathbb{N} \to \mathbb{N}$ koja koraku pridružuje indeks ćelije ispred glave stroja nakon koraka.

Kada govorimo o koraku, ne promatramo skup d iz definicije samog Turingova stroja, već se nalazimo u kontekstu rednog broja koraka stroja. Od sada ćemo o koraku govoriti kao o vremenskom trenutku t, što opravdava ime vremenskih funkcija.

Uz pomoć ovih triju funkcija sada možemo uvesti termin povijesti Turingova stroja.

Definicija 2.2.2. *(Povijest Turingovog stroja)*
Povijest Turingova stroja*, u oznaci H_M, definira se kao uređena trojka*

$$H_M = (\$_M, \square_M, P_M).$$

Povijest stroja u trenutku t nazivamo konfiguracijom*, i definiramo je po točkama:*

$$H_M(t) = (\$_M, \square_M, P_M)(t) = (\$_M(t), \square_M(t, i), P_M(t)), i \in \mathbb{N}.$$

Stroj na početku svog rada nije napravio nijedan korak, dakle nalazimo se u vremenskom trenutku $t = 0$, pa je konfiguracija dana s $H_M(0)$. Sve konfiguracije poslije mogu se lako dobiti iz inicijalne konfiguracije uz korištenje funkcija O, N, D iz definicije stroja. Naime, $\forall t \in \mathbb{N}$ vrijedi sljedeće:

- $\$_M(t+1) = N_M(\$_M(t), \square_M(t, P_M(t)))$

  Lijeva strane jednakosti odgovara stanju u kojem se stroj nalazi nakon vremenskog trenutka $(t+1)$, odnosno nakon $(t+1)$. koraka.
  S desne strane imamo funkciju prijelaza N_M; ona kao izlaz daje (neko) stanje stroja (dakle, po pitanju kodomene se podudaramo). Kao prvi argument dajemo joj stanje stroja u kojem se stroj nalazi nakon vremenskog trenutka t, a to je upravo $\$(t)$. Kao drugi argument potreban nam je simbol zapisan u ćeliji nad kojom se nalazi glava stroja. Nakon trenutka t, sadržaj je ćelije upravo $\square_M(t, P_M(t))$.

- $\square_M(t+1, P_M(t)) = O_M(\$_M(t), \square_M(t, P_M(t)))$

  Slično kao gore, iz definicije vremenskih funkcija vidimo da lijeva strana odgovara sadržaju ćelije indeksa $P_M(t)$ nakon trenutka $(t+1)$.
  Očito, na desnoj se strani nalazi „dobra" funkcija - funkcija O_M. Ona će kao izlaz dati novi sadržaj ćelije nad kojom se glava nalazi. Nužno nam je stanje u kojem se stroj nalazi nakon trenutka t, a to je upravo $\$_M(t)$. Nadalje, treba nam simbol nad ćelijom indeksa $P_M(t)$ nakon trenutka t, a upravo to odgovara drugom argumentu funkcije. Sada samim djelovanjem funkcije O_M dobivamo sadržaj ćelije indeksa $P_M(t)$ nakon trenutka $t+1$, što odgovara lijevoj strani.

- $(\forall j \neq P_M(t))\, \square_M\,(t+1, j) = \square_M(t, j)$

  Ova jednakost je jednostavna, i hoće reći kako u se nekom vremenskom trenutku sadržaj ćelija nad kojima se glava ne nalazi ne može mijenjati.

- $P_M(t+1) = P_M(t) + D(\$_M, \square_M(t, P_M(t)))$
  Očito. Desna strana uzima vremenski trenutak t i pomiče glavu na način određen funkcijom D, što odgovara lijevoj strani jednakosti.

Kao što znamo, Turingov stroj ne mora stati. No, *znamo* da je stroj stao ako se njegova konfiguracija s vremenom ne mijenja.

Definicija 2.2.3. *(Kraj rada Turingovog stroja)*
Kažemo da je Turingov stroj M stao u trenutku t ako za svaki $t' > t$ vrijedi:

$$\$_M(t) = \$_M(t'),$$

$$(\forall i \in \mathbb{N})\ \ \square_M\,(t, i) = \square_M(t', i)$$

i

$$P_M(t) = P_M(t').$$

U narednim ćemo se izlaganjima često koristiti strukturama čije definicije slijede.

Definicija 2.2.4. *(Program za Turingov stroj)*
Program v za Turingov stroj $M = (S_M, I_M, O_M, N_M, D_M)$ *konačan je niz simbola iz* I_M. *Skup svih programa za stroj* M *označavamo s* TP_M.

Ako, primjerice, imamo $I_M = \{a, b, c\}$, kao programe možemo navesti a, bc, $aabbcc$ i tako dalje.

Označimo li klasu svih Turingovih strojeva s $\mathcal{M}$, za Turingov stroj $M = (S_M, I_M, O_M, N_M, D_M)$ kratko pišemo $M \in \mathcal{M}$.

Definicija 2.2.5. *(Skup programa za Turingov stroj)*
Neka je $M \in \mathcal{M}$. *Neprazan skup* V *nazivamo* skupom programa za Turingov stroj M, *i pišemo* $V \in TS_M$, *ako vrijedi:*

$$(\forall v \in V)\ v \in TP_M.$$

Uočimo, u samom modelu Turingovih strojeva ne postoji nikakva fundamentalna razlika između programa i podatka. Riječ je, u jednom i u drugom slučaju, konačnom nizu simbola (elemanata abecede stroja). Isto tako vidimo, s obzirom na gornje dvije definicije, da je zapravo $TP_M = \cup_{k=0}^{\infty} I_M^k$, odnosno $TS_M = \mathcal{P}(TP_M)$.

Sada nam slijedi središnji dio ovoga rada, ujedno i dio koji predstavlja srž rada kojim je Frederick Cohen započeo razvoj ove teorije. U intuitivnom poimanju virusa koje smo dali, opisalo smo virus kao program koji može mijenjati druge programe, i to na način da mogu sadržavati njegovu kopiju, ili njegovu modificiranu verziju. U strogoj matematičkoj definiciji održavamo ovu razinu općenitosti. Evoluciju programa u smislu njegove promjene nagovijestio je Teorem o fiksnoj točki. Dotični smo iskaz i pripadajuće objašnjenje važnosti teorema dali na početku rada. Perspektivu koju Cohen uzima predstavlja virus kao element nekog skupa, i to ne nužno kao skupa s jednim elementom. Dakle, on nas upoznaje s konceptom takozvanog virusnog skupa, koji sadrži virus i sve njegove modifikacije, to jest varijante.

U opisu virusa uočavamo i pojam „okoline". Riječ je jednostavno o nekom stroju. Taj je podatak od velike važnosti. Naime, program v možda je virus kad ga se kao ulazni podatak da stroju M, ali nije nužno da je program v virus u nekom drugom stroju M'. Prebacimo li se na trenutak u okvir realnih računala, vidimo da neki program u nekom programskom jeziku može biti virus za određeni operacijski sustav, ali njegovo ponašanje neće biti virusno u drugom okolini, odnosno drugom operacijskom sustav. U definiciji ćemo zato, uz sam skup, morati u obzir uzimati i neki konkretni stroj M, uz koji neki program ima smisla nazivati virusom. Taj će M predstavljati okolinu.

Formalna je definicija virusnog skupa, odnosno virusa kao elementa tog skupa, kompleksna, stoga ćemo je prvo navesti u njenom rigoroznom obliku, a nakon toga ponuditi objašnjenje za ono što predstavlja. Koristimo se svim do sad uvedenim objektima.

Definicija 2.2.6. *(Virusni skup)*
Neka je $M \in \mathcal{M}$*, i neka je* $V \in TS_M$*. Uređeni par* (M, V) *nazivamo* virusnim skupom *ako vrijedi:*

$$
\begin{aligned}
&[\forall v \in V\ [\forall H_M\ [\forall t, j \in \mathbb{N} && (2.1)\\
&\quad [\\
&\quad\ 1.\ P_M(t) = j, && (2.2)\\
&\quad\ 2.\ \$_M(t) = \$_M(0), && (2.3)\\
&\quad\ 3.\ (\square(t,j), \ldots, \square(t, j + |v| - 1)) = v && (2.4)\\
&\quad]\\
&\implies [\exists v' \in V\ [\exists t' > t\ [\exists j' && (2.5)\\
&\quad [\\
&\quad\ 1.\ [[(j' + |v'|) \leq j]\ \textit{ili}\ [(j + |v| \leq j')]], && (2.6)\\
&\quad\ 2.\ (\square(t', j'), \ldots, \square(t', j' + |v| - 1)) = v, && (2.7)\\
&\quad\ 3.\ [\exists t''\ \textit{td.}\ [t < t'' < t']\ i\ [P_M(t'') \in \{j', \ldots, j' + |v'| - 1\}] && (2.8)\\
&\quad]\\
&\quad]]]\\
&\ ]]].
\end{aligned}
$$

U tom slučaju svaki $v \in V$ *nazivamo* virusom s obzirom na M*. Skup svih virusnih skupova za stroj* M *označavamo s* $\mathcal{V}$*.*

Dodajmo sada gore izrečenoj definiciji njeno intuitivno poimanje redom po retcima definicije. Dakle:
Uređeni par (M, V) nazivamo virusnim skupim ako:

- (2.1): Ako za svaki program $v \in V$ i za svaku povijest (odnosno konfiguraciju) H_M, svaki vremenski trenutak t i ćeliju j imamo:
  - (2.2): u trenutku t glava stroja nalazi se ispred ćelije j (govorimo o ćeliji indeksa j),
  - (2.3): stroj M u trenutku t nalazi se u inicijalnom stanju,

  - (2.4): program v nalazi se na vrpci stroja, s tim da je njegov krajnji lijevi znak na ćeliji j,

- (2.5): onda postoji program $v' \in V$, trenutak $t' > t$ i ćelija j' tako da vrijedi:
  - (2.6), (2.7): program v' nalazi se negdje na traci, ali ispred ili iza programa v,
  - (2.8): postoji vremenski trenutak t'' između početnog t i završnog t' unutar kojeg je program v' zapisan na traku

Neprecizno, ali najjednostavnije rečeno, program $v \in V$ je virus s obzirom na stroj M ako u nekom trenutku stroj M na traku zapiše $v' \in V$. Kako smo već napomenuli, riječ ne mora biti o identičnoj kopiji, odnosno ne mora biti $v = v'$. Isto tako, uočimo kako nema govora o nekoj štetnoj proceduri. Riječ je samo o kopiranju, odnosno evoluciji. To je osnovna karakteristika virusa od koje polazi Cohen.

Kratko ćemo reći da je uređeni par (M, V) virusni skup ako je $(M, V) \in \mathcal{V}$, a v virus ako $(M, V) \in \mathcal{V}$ i $v \in V$. Kao skraćeni zapis za uvjete 2.1-2.8 iz definicije virusnog skupa koristit ćemo sljedeću oznaku:

$$[v \overset{M}{\Longrightarrow} V].$$

Dakle, definicija se uz upravo uvedenu oznaku svodi na:

$$[[(M, V) \in \mathcal{V}] \iff [[M \in \mathcal{M}] \text{ i } [V \in TS_M] \text{ i } [v \overset{M}{\Longrightarrow} V]]].$$

Sada konačno imamo alate potrebne za formalnu definiciju pojma evolucije koji smo do sada upotrebljavali isključivo kolokvijalno.

Definicija 2.2.7. *(Evolucija virusa)*
Kažemo da virus v evoluira u virus v' *s obzirom na* M *ako vrijedi:*

$$[[(M, V) \in \mathcal{V}] \; i \; [[v \in V] \; i \; [v' \in V] \; i \; [v \overset{M}{\Longrightarrow} \{v'\}]]].$$

Kažemo da virus v' evoluira iz virusa v *s obzirom na* M *ako virus* v *evoluira u* v' *s obzirom na* M*.*
Kažemo da je virus v' evolucija virusa v *ako vrijedi:*

$$[(M, V) \in \mathcal{V} \; [\exists i \in \mathbb{N} \; [\exists V' \in V^i$$

$$[v, v' \in V] \; \& \; [\forall v_k \in V' \; [v_k \overset{M}{\Longrightarrow} v_{k+1}]] \; \& \; [\exists l, m \in \mathbb{N}, \; l < m, \; v_l = v, \; v_m = v']]]].$$

Posljednji uvjet u definiciji možemo „prevesti" ovako: virus v' je evolucija virusa v ako virus v' nastaje direktno od v (to imamo u slučaju $l = m-1$, dakle je v' nastao odmah na traci temeljem rada stroja s ulazom v), ili se u nekom trenutku pojavio na traci (tijekom evolucije virusa v).

Nakon izlaganja definicija, prirodno se zapitati koja svojstva imaju virusi, odnosno virusni skupovi.

2.2.2 Osnovni rezultati

Kako to obično biva sa skupovima, pitanje kojim ćemo se prvo baviti jest pitanje zatvorenosti na konačne unije. Za očekivati je da je unija dvaju (a onda i bilo koja konačna unija) virusnih skupova opet virusni skup. Ako je v iz prvog skupa koji čini uniju, onda na njegovu evoluciju neće utjecati virusi iz drugih skupova iz unije. Dakle, za v će ponovno vrijediti sva svojstva iz definicije. Situacija je, naravno, potpuna simetrična, odnosno analogna za sve ostalo skupove koji čine promatranu uniju. Naravno, uniju skupova ima smisla promatrati u slučaju istih okolina.

Teorem 2.2.8. *(Konačna unija virusnih skupova)*
Neka je $M \in \mathcal{M}$, i neka je $U^* \subset \mathcal{P}(I_M^*)$. Tada vrijedi:

$$[[\forall V \in U^*, (M,V) \in \mathcal{V}] \implies [(M, \cup U^*) \in \mathcal{V}]].$$

Dokaz. Označimo $\cup U^* = U$. Sada se dokaz teorema svodi na dokazivanje da je upravo U virusni skup s obzirom na M. To znači da treba pokazati da vrijede uvjeti definicije, to jest da vrijedi:

$$[[U \in TS_M] \text{ i } [M \in \mathcal{M}] \text{ i } [[\forall v \in U]\ v \overset{M}{\implies} U]].$$

Očito, samo je posljednji uvjet uistinu potrebno provjeriti. Prepostavili smo da je $(M,V) \in \mathcal{V}$, za svaki skup $V \in U^*$. To ponovno po definiciji jednostavno znači:

$$[\forall V \in U^*, \forall v \in V\ [v \overset{M}{\implies} V]]. \tag{2.9}$$

No, sad iz $V \in U^* \implies V \subset U$ i (2.9) odmah imamo:

$$[\forall v \in U\ [v \overset{M}{\implies} U]],$$

što je i trebalo pokazati. □

Ovaj teorem oruđe je kojim možemo dokazati tvrdnju o „najvećem virusnom skupu" s obzirom na stroj M, pri čemu je taj skup upravo unija svih virusnih skupova

s obzirom na taj stroj. Naravno, o ovome ima smisla govoriti ako postoji barem jedan virusni skup za stroj. Napomena je bitna jer postoji stroj za kojeg virus ne postoji. Dokaz je tvrdnje trivijalan - potrebno je uzeti stroj koji ne radi ništa ni za jedan ulaz - ali je tvrdnja važna, stoga je izražavamo zasebno.

Propozicija 2.2.9. *Postoji stroj $M \in \mathcal{M}$ za koji ne postoji $V \in TS_M$ takav da je $(M, V) \in \mathcal{V}$.*

Iskažimo sada rezultat o „najvećem virusnom skupu."

Teorem 2.2.10. *(Najveći virusni skup)*
Neka je $M \in \mathcal{M}$ za koji postoji barem jedan $V_0 \in TS_M$ tako da je $(M, V_0) \in \mathcal{V}$. Tada postoji $U \subset \mathcal{P}(I_M^*)$ tako da vrijedi:

1. $(M, U) \in \mathcal{V}$,
2. $[\forall V \in U\ [(M, V) \in \mathcal{V}] \implies [\forall v \in V\ [v \in U]]]$.

Skup U u tom slučaju nazivamo najvećim virusnim skupom s obzirom na stroj *M, i pišemo $U \in LVM(M)$.*

Dokaz. S obzirom na to da smo tvrdnju izrazili po slučajevima, tako ćemo provoditi i dokaz. Prvu ćemo točku izvesti koristeći se teoremom o uniji virusnih skupova, a drugu ćemo dobiti tako što ćemo pretpostaviti suprotno i doći do kontradikcije.

Kako smo već najavili, definiramo skup U sa $U = \cup\{V : (M, V) \in \mathcal{V}\}$. Prva tvrdnja slijedi direktno iz prethodnog teorema jer ako je V takav da je $(M, V) \in \mathcal{V}$, onda je i $(M, \cup\{V : (M, V) \in \mathcal{V}\}) \in \mathcal{V}$, odnosno imamo $(M, U) \in \mathcal{V}$.

Drugu tvrdju dokazujemo koristeći se kontradikcijom, dakle pretpostavljamo suprotno. Tada slijedi postojanje skupa $V \in U$ takvog da je $(M, V) \in \mathcal{V}$, pri čemu postoji $v \in V$ za koji je $v \notin U$. No, iz definicije unije imamo da $v \in V$ povlači $v \in U$. Dakle, dobivamo $v \in U$ i $v \notin U$, što je kontradikcija. □

Kako imamo najveći virusni skup, okrećemo se i pitanju „najmanjeg virusnog skupa". Taj skup je virusni za neki stroj M, ali izbacivanjem i jednog njegovog elementa to više ne bi vrijedilo. Definirajmo ovo formalno.

Definicija 2.2.11. *Kažemo da je (M, V)* najmanji virusni skup s obzirom na *$M \in \mathcal{M}$, u oznaci $(M, V) \in SVS(M)$, ako vrijedi:*

$$(M, V) \in \mathcal{V},$$

i

$$[\nexists V' \subset V\ [(M, V') \in \mathcal{V}]].$$

Koliko „malen" može biti element iz $SVS(M)$? Pokazujemo kako postoji stroj M čiji virusni skup ima smo jedan element.

Teorem 2.2.12. *Postoji Turingov stroj M, skup $V \in TS_M$, tako da vrijedi:*

1. $(M, V) \in SVS(V)$, *i*
2. $|V| = 1$.

Dokaz. Dokaz se provodi konstruktivno. Gradimo tablicu prijelaza za stroj M, pri čemu je $S_M = \{s_0, s_1\}$ i $I_M = \{0, 1\}$.

$S_M \times I_M$	N_M	O_M	D_M
$(s_0, 0)$	s_0	0	0
$(s_0, 1)$	s_1	1	$+1$
$(s_1, 0)$	s_0	1	0
$(s_1, 1)$	s_1	1	$+1$

Sada stavimo $V = \{1\}$. Druga je točka iz tvrdnje očito zadovoljena. Sada po definiciji najmanjeg virusnog skupa za V moramo pokazati da je $(M, V) \in V$ i da je $(M, \{\}) \notin \mathcal{V}$. Ovo potonje vrijedi jer $\{\}$ ne promatramo kao skup programa (naglasili smo kako $V \in TS_M$ mora biti neprazan).

Iz dane tablice očito je da je ulaz „1" virus. Naime, čim stroj pročita jedinicu, on će je u sljedećem vremenskom trenutku zapisati na prvo mjesto nadesno. Zato je $j' = j + 1, t' = t + 2, t'' = t + 1$ (uz oznake iz definicije virusa). Dakle, vrijedi: $1 \overset{M}{\Longrightarrow} \{1\}$. □

Iz gornjeg teorema vidimo da se virus doslovno kopira, odnosno umnaža. Vrijedi i svojevrsni obrat. Kad god se neki program duplicira s obzirom na neki stroj, onda je on virus s obzirom na taj stroj. Ovo je vrlo jednostavan rezultat na koji ćemo se često pozivati.

Lema 2.2.13. *Za svaki Turingov stroj M i program $v \in TP_M$ vrijedi:*

$$[[v \overset{M}{\Longrightarrow} \{v\}] \implies [(M, \{v\}) \in \mathcal{V}]].$$

Dokaz. Po definiciji opet imamo:

$$[[(M, \{v\}) \in \mathcal{V}] \iff [\{v\} \in TS_M \text{ i } v \overset{M}{\Longrightarrow} \{v\}]].$$

Tvrdnja slijedi iz pretpostavke da je $v \overset{M}{\Longrightarrow} \{v\}$, i činjenice $v \in TP_M \implies \{v\} \in TS_M$. □

Tvrdnju prethodnog teorema možemo poopćiti.

Teorem 2.2.14. *Za svaki broj $k \in \mathbb{N}$ postoji stroj $M \in \mathcal{M}$ i skup $V \in TS_M$ tako da vrijedi:*

1. $(M, V) \in SVS(M)$,
2. $|V| = k$.

Dokaz. Za proizvoljni, ali fiksirani prirodni broj k konstruirat ćemo stroj M s traženim svojstvom. Definiramo $S_M = \{s_0, s_1, \ldots, s_k\}$, $I_M = \{0, 1, \ldots, k\}$, pri čemu $\forall x \in I \setminus \{0\}$ imamo:

$S_M \times I_M$	N_M	O_M	D_M
$(s_0, 0)$	s_0	0	0
(s_0, x)	s_x	x	$+1$
$(s_x, *)$	s_x	$x + 1 \pmod k$	0

Prvi redak tablice govori kada stroj staje. Drugim retkom tablice vidimo da ako stroj u stanju s_0 pročita znak x, pomiče se u desno i prelazi upravo u stanje s_x pri čemu ništa na zapisuje. Tek u sljedećem trenutku će zapisati, što nam govori posljednji redak tablice kojim poopćujemo rad stroja: kada smo u stanju s_x, jednostavno zapisujemo „sljedeći" znak (s obzirom na x) i ne mičemo se.

Definiramo $V = \{1, 2, \ldots, k\}$. Odmah imamo $|V| = k$. Preostaje dokazati točku 1. Po definiciji moramo pokazati da je $(M, V) \in \mathcal{V}$, i da ne postoji skup $V' \subset V$ za koji je $(M, V') \in \mathcal{V}$. Redom pokazujemo da su oba uvjeta zadovoljena.

- $(M, V) \in \mathcal{V}$
  Iz tablice prijelaza stroja, slično kao u teoremu 2.2.12, a kako smo upravo pojasnili, vidimo da će stroj za svaki program v iz V zapisati $v + 1$, također iz V, i to odmah na mjesto desno. Zapravo imamo:

$$1 \overset{M}{\Longrightarrow} \{2\}$$

$$2 \overset{M}{\Longrightarrow} \{3\}$$

$$\ldots$$

$$k \overset{M}{\Longrightarrow} \{1\},$$

  pri čemu uz oznake iz definicije virusa imamo točno $t'' = t + 1, t' = t + 2, j' = j + 1$. Dakle, $[\forall v \in V\ [v \overset{M}{\Longrightarrow} V]]$, pa je prvi uvjet ispunjen.

- $[\nexists V' \subset V\ [(M, V') \in \mathcal{V}]]$
  Pretpostavimo suprotno, odnosno da takav V' postoji. Kako je $V' \subsetneq V$, postoji niz znakova $v_1, v_2, \ldots, v_l$ takvih da je $v_j \in V \setminus V'$ za $j = 1, 2, \ldots, l$. Označimo sa v neki element tog niza za kojeg postoji $v' \in V'$ takav da je $v' + 1 = v$. Uočimo, takav mora postojati jer u V' postoji barem jedan element. U protivnom je V' prazan, a onda ne može biti $(M, V') \in \mathcal{V}$ po definiciji. Čak štoviše, ne može biti ni $V' \in TS_M$, opet po definiciji. Nadalje, iz dokaza prvog uvjeta, odnosno iz definirane tablice prijelaza za stroj M, vidimo da i evoluira isključivo u $i + 1 \pmod k$. Iz ovoga slijedi da ne može postojati nijedan drugi $v'' \neq v \in V$ za koji je $[v' \overset{M}{\Longrightarrow} \{v''\}]$. Dakle, za v' svakako ne vrijedi $[v' \overset{M}{\Longrightarrow} V' \subset V]$. Ovo je u kontradikciji s pretpostavkom da je $(M, V') \in \mathcal{V}$, pa je time ispunjen i drugi uvjet.

□

Ovim smo teoremom pokazali egzistenciju virusa s konačno mnogo evolucija. Dakle, „mogućnosti" tog virusa u smislu evolucije su ograničene. Može li više od toga? Odgovor na to pitanje daje sljedeći teorem.

Teorem 2.2.15. *Postoji $M \in \mathcal{M}$ i $V \in TS_M$ tako da vrijedi:*

1. $(M, V) \in \mathcal{V}$
2. $|V| = |\mathbb{N}| = \aleph_0$.

Dokaz. Rad ćemo stroja ponuditi opisno, i to primjerom. Naime, tablica koja bi u potpunosti opisala njegov rad sadrži 15 redaka i 13 stanja. Ovim pristupom pojednostavljujemo dokaz, odnosno naglasak stavljamo na razumijevanje rada samog stroja, bez da detaljno proučavamo tehničku stranu tablice prijelaza. Za detaljno predočenu tablicu pogledati [1].

Definiramo $I_M = \{L, R, 0, X\}$. Jednostavno rečeno, stroj će za ulaze oblika $L0^kR$ na traku zapisati $L0^{(k+1)}R$. Zato ćemo definirati $V = \{L0^kR : k \in \mathbb{N}\}$. Očita je bijekcija koju možemo ostvariti između V i $\mathbb{N}$ pa drugu tvrdnju već imamo.

U svrhu primjera kojim demonstriramo rad stroja promotrimo program

$$L00R.$$

Glava stroja na početku se nalazi nad znakom L. Počinje svoj rad tako da pročita taj znak (stroj stoji ako na prvom mjestu nije taj znak), nakon čega sve znakove 0 mijenja u X, sve dok ne dođe do znaka R. Zato ćemo u naredna dva koraka dobiti

$$LXXR,$$

i glava se nalazi nad R. Sada se glava pomiče još jedno mjesto udesno i započinje pisati novi niz znakova, pri čemu prvo zapisuje znak L. Znakove X koristit ćemo za brojanje znakova 0 koje u novi niz trebamo zapisati. Sada, dakle, na traci imamo

$$LXXRL.$$

Sada se glava pomiče ulijevo do prvog znaka X iza znaka R (jer R označava kraj prvotnog niza). Kada pročita znak X, promijeni ga u 0, a onda se glava vraća zapisati 0 u novi niz. Dakle, imamo

$$LX0RL0.$$

Ponovno, glava se vraća do prvog znaka X nakon R, i ponovi postupak. Zato konačno imamo $L00RL00$. Kada glava ponovno krene lijevo, neće pročitati nijedan znak X nakon R, i doći će do L koji označava početak početnog niza. Tada znamo da smo stigli do kraja, stoga se glava vraća udesno do kraja novog niza, zapisuje dodatni znak 0, i kraj niza označava dodatnim znakom R. Dakle, na kraju rada stroja na traci imamo zapisan sljedeći niz:

$$L00RL000R.$$

Ovime smo ujedno i pokazali svojstvo 1. Naime, svaki $v \in V$ je oblika $v = L0^kR$, za neki $k \in \mathbb{N}$, i za njega, pokazali smo primjerom, vrijedi:

$$[v \overset{M}{\Longrightarrow} \{v' = L0^{(k+1)}R\} \in V].$$

□

Objasnimo vrlo važnu posljedicu ovog teorema, a potom je formalno iskažimo. Za isti stroj iz teorema postoji (prebrojivo) beskonačno mnogo programa koji nisu virus s obzirom na taj stroj. Ovo je iznimno važno. Naime, to znači da neki stroj s konačno mnogo stanja ne može ustvrditi je li neki uređeni par (M', V') virusni skup na način da pobroji viruse v (kojih, vidimo iz teorema, može biti beskonačno mnogo), ili na način da pobroji programe koji za taj stroj M' nisu virus (jer i njih, vidjet ćemo sada iz korolara, može biti beskonačno mnogo).

Korolar 2.2.16. *Za stroj M iz teorema 2.2.15 postoji $W \in TS_M$ tako da vrijedi:*

1. $[\forall w \in W\ [\nexists W' \subset W\ [w \overset{M}{\Longrightarrow} W']]]$
2. $|W| = \aleph_0$

Dokaz. S obzirom na način rada stroja M, jednostavno definiramo

$$W = \{X, XX, ..., X^k, ...\}.$$

Stroj M ni za jedan element $w \in W$ neće napraviti nijedan korak, odnosno:

$$[\nexists t' > t \ [P_M(t') \neq P_M(t)]], \ \forall t \in \mathbb{N}.$$

Zato slijedi tvrdnja. □

Već smo spomenuli kako postoji stroj za kojeg ne postoji virus (stroj jednostavno ne radi ništa ni za jedan jedini ulaz). Sada ćemo pak pokazati kako za svaki konačan niz $v \in TP_M$ možemo konstruirati stroj za koji je taj program virus. Štoviše, za taj će stroj, označimo li ga s M, vrijediti $LVS(M) = SVS(M)$. Ovo zapravo znači da nijedan drugi program v' ne može biti virus za taj stroj. Kada bi takav v' postojao, onda bi i unija tih dvaju programa ($\{v\} \cup \{v'\}$) zajedno s M činili virusni skup, pa ne bismo imali danu jednakost. Pokažimo ovo sada formalno.

Propozicija 2.2.17. *Neka je v neki konačan niz znakova. Tada postoji $M \in \mathcal{M}$ tako da vrijedi:*

$$(M, \{v\}) \in \mathcal{V}.$$

Dokaz. Neka je v proizvoljan, ali fiksiran, odnosno $v = v_0 v_1 \dots v_k$, pri čemu I_M definiramo tako da je $v_i \in I_M, \ i = 1, 2, \dots, k$. Definiramo i $S_M = \{s_0, s_1, \dots, s_k\}$. Opišimo riječima jednostavan rad ovog stroja. U vremenskom trenutku $t = 0$, stroj se nalazi u stanju s_0 i za dani ulaz $v' = v'_0 \dots v'_k$ provjerava je li $v_0 = v'_0$. Ako nije, stroj staje - ostaje u istom stanju i ništa ne piše. Ako jest, prelazi u sljedeće stanje, čita sljedeći znak i obavlja analognu provjeru. Dakle, u trenutku $t = i$ stroj M nalazi se u stanju s_i te provjerava je li $v_i = v'_i$. Ako u konačnici vrijedi $v = v'$, onda on na kraj niza v jednostavno redom upisuje znakove $v_0, \dots, v_k$, time zapravo kopirajući v. Sada iz leme 2.2.13 slijedi tvrdnja. □

Za konkretni smo program (niz znakova), dakle, našli stroj u kojem je taj program virus. Sada ćemo konstruirati stroj za kojeg je svaki program virus.

Propozicija 2.2.18. *Postoji $M \in \mathcal{M}$ tako da za svaki $v \in TP_M$ postoji $V \in TS_M$ tako da vrijedi:*

$$[[v \in V] \ i \ [(M, V) \in \mathcal{V}]].$$

Dokaz. Definiramo $I_M = \{X\}, S_M = \{s_0\}$. Tvrdnja direktno slijedi promotrimo li stroj koji za svaki program na kraj jednostavno zapiše X. Tada $\forall v \in TP_M$ naprosto vrijedi $[v \overset{M}{\Longrightarrow} \{X\}]$, uz prethodno definiran skup $V = \{X, v\}$. □

Čak štoviše, s obzirom na rad stroja i definirani I_M, vrijedi $V \in LVS(M)$.

2.2.3 Izračunljivost i virusi

Sada, nakon izlaganja osnovnih rezultata koji se tiču virusnih skupova, znamo koja svojstva očekivati od virusa. No, ono što nas najviše zanima jest kako prepoznati koji je program virus. Želimo ih naučiti „prepoznavati". Kako smo vidjeli iz teorema 2.2.15 i pripadajućeg korolara 2.2.16, ne postoji Turingov stroj koji bi odlučio je li prozvoljni uređeni par (M, V) virusni skup jednostavno pobrojavajući viruse s obzirom na stroj, odnosno programe koji nisu virus s obzirom na stroj (njihov je broj prebrojivo beskonačan). Možemo li taj problem riješiti na neki drugi način? U svrhu „prepoznavanja" virusa specificiramo dva aspekta problema, od kojih je prvi odlučivost samog virusa.

2.2.3.1 Neodlučivost virusnog skupa

Ovo je ono osnovno što nas zanima. Pitamo se postoji li Turingov stroj D koji će za stroj M i skup $V \in TS_M$ u konačno mnogo vremena dati odgovor na pitanje je li (M, V) virusni skup. Ako znamo da je (M, V) virusni skup, efektivno znamo koji program predstavlja virus za okolinu koju promatramo. Potvrdan bi odgovor na postavljeno pitanje značio kako praktički ne može postojati virus koji bi za neko računalo predstavljalo opasnost - jednostavno bismo znali kako dotični program jest virus (pod pretpostavkom da se nađe efektivni algoritam za taj problem). No, sljedeći teorem, fundamentalni rezultat ove teorije, a koji je zapravo posljedica vrlo dobro nam poznatog[1] halting problema, daje nam nažalost negativan odgovor o postojanju takvog odlučitelja D. Definitivna odluka o tome je li v virus za M matematički je nemoguća. Slijedi formalan oblik ove tvrdnje koja se još naziva i problem detekcije virusa, a nakon toga predstavljamo njen dokaz.

Teorem 2.2.19. *(Neodlučivost virusnog skupa)*
Ne postoji Turingov stroj D i $s_i \in S_D$ tako da za proizvoljni Turingov stroj M i skup $V \in TS_M$ stroj D staje u vremenskom trenutku t i nalazi se u stanju s_i ako i samo ako je uređeni par (M, V) virusni skup. Tvrdnju formalno možemo zapisati ovako:

$$[\nexists D \in \mathcal{M}\ [\nexists s_i \in S_D\ [\forall M \in \mathcal{M}\ [\forall V \in TS_M$$

$$[[D \text{ staje u trenutku } t] \text{ i } [\$_D(t) = s_i \iff (M, V) \in \mathcal{V}]]]]]].$$

Dokaz. Dokaz se provodi svođenjem ovog problema na halting problem za kojeg znamo da nije odlučiv. Uzimamo proizvoljni stroj M'. Tada gradimo stroj M koji za ulaz v radi sljedeće:

[1] Navedeni smo problem spomenuli na kolegiju „Interpretacija programa", a detaljno obradili na kolegijima „Izračunljivost" i „Složenost algoritama".

1. na temelju ulaznog programa v na traku zapisuje podniz v' od v koji predstavlja program za M'
2. simulira rad stroja M' s ulazom v'
3. kopira v ako stroj M' staje s ulazom v'

Primjećujemo da se u iskazu teorema spominje proizvoljni stroj M, a zapravo ga konstruiramo uz pomoć M'. No, tvrdnju ćemo teorema samim krajem dokaza u potpunosti opravdati i uspostaviti vezu između ta dva stroja. Nadalje, zapisivanje se programa v' odvija na traci stroja M, i to desno od programa M. U svakom slučaju, vidimo da se v kopira ako i samo ako M' staje s ulazom v'. Kako problem o tome staje li Turingov stroj za neki ulaz nije odlučiv, i jer je svaki program koji se kopira virus (lema 2.2.13), slijedi tvrdnja teorema.

S obzirom na to da će M simulirati rad stroja M', stroj M' moramo malo pažljivije oblikovati. Prvo, moramo osigurati kako M' u nekom trenutku neće promijeniti naš program v (koji moramo kopirati u slučaju da M' s ulazom v' stane) ili sam neće zapisati virus s obzirom na M. Stoga ćemo bez smanjenja općenitosti ograničiti abecedu stroja M' u smislu da neće sadržavati znakove L, R. Sve pojave ovih znakova konzistentno možemo zamijeniti drugima na način da se rad stroja M' ne mijenja. Dodatno, u abecedu ćemo stroja M' (odnosno M) „umjetno" uvesti znak R' koji će imati ulogu separatora programa v i v'. Svi će upravo spomenuti znakovi biti ključni prilikom rada stroja M i definiranja skupa V. Drugo, kad god M' pročita znak R', ne mijenjamo stanje, ostavljamo znak koji smo pročitali, i pomičemo se jedno mjesto desno. Na ovaj način osiguravamo da stroj M' svojim radom ne može utjecati na program v. Dakle, želimo da za M' (odnosno da za M prilikom simulacije stroja M') vrijedi i:

$$[\forall s_i \in S_{M'}, \forall t, j \in \mathbb{N}$$
$$[[[\$_{M'}(t) = s_i] \text{ i } [\square(t, j) = R'] \text{ i } [P_{M'}(t) = j + 1]]$$
$$\Longrightarrow$$
$$[[\$_{M'}(t + 1) = s_i] \text{ i } [\square(t + 1, j) = R'] \text{ i } [P_{M'}(t + 1) = j + 2]]].$$

Treće, kada stroj M' obavi sve korake, odnosno završi s radom, želimo da M prelazi u stanje koje ćemo označiti sa s_x. Simbol x predstavlja neki prirodni broj koji ovisi o samom broju koraka koje načini stroj M prilikom simulacije stroja M' (vidi tablicu prijelaza niže).

Definirajmo sada skup $V = \{v_0\}$, pri čemu je $v_0 = Lv'R$, i v' neki konačan niz znakova uz ograničenja navedena gore.

Primjerom koji slijedi, a kojem je svrha jasnije prikazati rad stroja M, pokazat ćemo da da vrijedi

$$[(M,V) \in \mathcal{V} \iff M' \text{ staje s ulazom } v']. \tag{2.10}$$

U primjeru pretpostavljamo da je stroj M' zadan, i upravo na temelju njega opisujemo rad stroja M. Neka je $v \in V$. U svrhu primjera stavimo $v' = 222$. Stroj M počinje s radom tek ako je kao prvi znak pročitao L. Nakon toga na kraj tog ulaznog programa zapisuje znak R', a potom se vraća do prvog lijevog znaka L te sve znakove između L i R (isključivo) kopira iza R'. Dakle, nakon tih nizova koraka, na traci imamo zapisan sljedeći niz:

$$L222RR'222.$$

Sada stroj pomiče glavu do prvog znaka nakon znaka R', odnosno na početak novostvorenog niza v'. Sada se simulira rad stroj M' s tim v' kao ulazom. Nakon što (i ako) stroj M' završi s radom, glava se stroja pomiče lijevo do prvog znaka L, nakon čega kopira niz znakova između L i R (uključivo) iza znaka R. Dakle, sada na traci svakako imamo zapisano (na traci se možebitno nalaze i neki drugi simboli s obzirom na simulaciju stroja M', no to nam za ono što pokazujemo nije bitno):

$$L222RL222R,$$

odnosno kopirali smo niz v. Jasno, ako stroj M' ne staje, zapravo ne staje ni stroj M. Ovime završavamo primjer kojim smo demonstrirali rad stroja i nastavljamo s dokazom.

Prikažimo sada formalnu tablicu prijelaza. Napomenimo kako ćemo se pritom koristiti „makroima". Riječ je o naredbama koje zamjenjuju određeni konačni niz naredbi i na taj način pojednostavljuju opis rada stroja. Koristimo sljedeće makroe[2]:

- $L(\text{"}a\text{"})$, odnosno $R(\text{"}a\text{"})$ - pomiči glavu stroja lijevo, odnosno desno, dok ne dođeš do znaka a, i prijeđi u sljedeće stanje,
- $CPY1(\text{"}a\text{"},\text{"}b\text{"},\text{"}c\text{"})$, odnosno $CPY2(\text{"}a\text{"},\text{"}b\text{"},\text{"}c\text{"})$ - kopiraj sve znakove između a i b, njih uključujući (odnosno isključujući u slučaju $CPY2$), iza znaka c, i prijeđi u sljedeće stanje.

Sada konačno imamo:

[2]Turingovi strojevi koji obavljaju naredbe koje smo predstavili makroima mogu se naći u [1].

$S_M \times I_M$	N_M	O_M	D_M
(s_0, L)	s_1	L	0
$(s_0, * \setminus \{L\})$	s_0	$*$	0
s_1		$R(\text{"}R\text{"})$	
s_2	s_3	$*$	$+1$
$(s_3, *)$	s_4	R'	0
s_4		$L(\text{"}L\text{"})$	
s_5		$CPY2(\text{"}L\text{"}, \text{"}R\text{"}, \text{"}R'\text{"})$	
s_6		$L(\text{"}R'\text{"})$	
$(s_7, *)$	s_8	$*$	$+1$
s_8		*simuliraj M'*	
s_x		$L(\text{"}L\text{"})$	
s_{x+1}		$CPY1(\text{"}L\text{"}, \text{"}R\text{"}, \text{"}R\text{"})$	

Sada dokazujemo nekoliko pomoćnih tvrdnji.

Prvo, s obzirom na to da se M odmah zaustavlja ako kao prvi simbol ne pročita L, i jer stroj M' ne može zapisati L na traku niti se L može nalaziti u nizu v', znamo da M' ne može zapisati virus na traku. Naime, u suprotnom (dakle, kada nekih od upravo spomenutih ograničenja ne bi bilo) bi se moglo dogoditi da M' na temelju v' na traku zapiše v i ne stane, što bi bilo u suprotnosti s (2.10).

Nadalje, stroj M' pomiče glavu desno ukoliko se dogodi da pročita znak R' koji predstavlja kraj programa v. Znamo da M' ne može utjecati na v.

Konačno, ako stroj M' nikad ne stane, stanje s_{x+1} nije dohvatljivo; to pak znači da M ne kopira v, odnosno v nije virus. Obratno potpuno analogno. Ako se v ne kopira, znači da stanje s_{x+1} nije dostignuto, a to znači da M' ne staje s ulazom v'. To jest:

$$[[M' \text{ staje s ulazom } v'] \iff [\exists t \in \mathbb{N}\ [\$_M(t) = s_{x+1}]]],$$

odnosno

$$[[M' \text{ staje s ulazom } v'] \implies [v \text{ se kopira s obzirom na } M]].$$

Dakle:

$$[[\exists t \in \mathbb{N}\ [\$_M(t) = s_{x+1}]] \implies [\forall v \in V\ [v \overset{M}{\Longrightarrow} \{v\}]]].$$

I u ovom dokazu koristimo koristimo lemu 2.2.13 pa imamo:

$$[[v \overset{M}{\Longrightarrow} \{v\}] \implies [(M, \{v\}) \in \mathcal{V}]].$$

S druge strane:

$$[[M' \text{ ne staje s ulazom } v'] \implies [\nexists t \in N\ [\$_M(t) = s_{x+1}]] \implies [v \text{ se ne kopira}]].$$

Slijedi:

$$[[M' \text{ staje s ulazom } v'] \iff [(M, V) \in \mathcal{V}]].$$

Halting problem nije odlučiv pa ne znamo hoće li M' stati s v'. Dakle, iz neodlučivosti halting problema imamo:

$$[\nexists D \in \mathcal{M}\ [\nexists s_i \in S_D\ [\forall M' \in \mathcal{M}\ [\forall v' \in TP_{M'}$$

$$[[D \text{ staje u trenutku } t] \text{ i } [\$_D(t) = s_i \iff M' \text{staje s ulazom } v']]]]]],$$

a uz upravo pokazanu ekvivalencije konačno dobivamo:

$$[\nexists D \in \mathcal{M}\ [\nexists s_i \in S_D\ [\forall M \in \mathcal{M}\ [\forall V \in TS_M$$

$$[[D \text{ staje u trenutku t}] \text{ i } [\$_D(t) = s_i \iff (M, V) \in \mathcal{V}]]]]]],$$

što je ono što smo htjeli pokazati.

□

Uočimo da je prethodni dokaz mogao biti nešto jednostavniji u smislu ograničenja koje smo postavili na strojeve da smo se koristili višetračnim Turingovim strojem. U tom bi slučaju stroj M imao dvije trake, i naprosto bi v' zapisao na drugu traku, i simulaciju stroja M' provodio nad tom trakom i programom v'. Dakle, pojednostavili bismo oblikovanje stroja M. O broju traka stroja M' ne bismo morali brinuti. Mogli bismo samo pretpostaviti da je stroj jednotračan s obzirom na to se svaki višetračni Turingov stroj može svesti na ekvivalentan jednotračni uz kvadratno povećanje složenosti. Taj rezultat može se vidjeti u [8].

Drugi aspekt problema „prepoznavanja" virusa jest pitanje prepoznavanja njegove evolucije.

2.2.3.2 Neodlučivost evolucije virusa

Nakon pitanja možemo li odlučiti je li v virus, i negativnog odgovora, prirodno se okrećemo sljedećem aspektu istog problema: možemo li barem za proizvoljan program v' ustvrditi da je on evolucija nekog virusa v? Odgovor je ponovno negativan, a dokaz se tvrdnje temelji na dokazu prethodnog teorema.

Teorem 2.2.20. *(Neodlučivost evolucije virusa)*
Ne postoji Turingov stroj D i $s_i \in S_D$ tako da za proizvoljni Turingov stroj M i programe v i v' za M stroj D staje u trenutku t i nalazi se u stanji s_i ako i samo ako je v' evolucija od v s obzirom na M. Tvrdnju formalno možemo zapisati ovako:

$$[\nexists D \in \mathcal{M}\ [\nexists s_i \in S_D\ [\forall M \in \mathcal{M}\ [\forall v, v' \in TP_M$$

$$[[D \textit{ staje u trenutku } t] \textit{ i } [\$_D(t) = s_i \iff v \overset{M}{\Longrightarrow} \{v'\}]]]]]].$$

Dokaz. Dokaz je prethodnog teorema bio nešto formalno kompleksniji, ali zahvaljujući njemu, ovaj dokaz je znatno lakši. Tvrdnja izlazi iz jednostavnije verzije stroja M iz teorema kojeg smo upravo dokazali. Za ulaz v stroj M kopira v. Iz jednostavne, ali često korištene leme 2.2.13 imamo da je $(M, \{v\}) \in \mathcal{V}$. Uočimo, ovo je nužan uvjet. Govorimo o evoluciji od v, stoga v mora biti virus. Nakon toga se na temelju v, baš kao u dokazu teorema o neodlučivosti virusa, na traku zapisuje v'. Sada treba ispitati je li v' virus (jer ispitujemo je li v' evolucija od v), a taj problem nije odlučiv, kako smo maloprije pokazali. Dakle, ne može postojati odlučitelj D iz iskaza tvrdnje. □

Ovim dvama najvažnijim rezultatima završavamo pregled Cohenove formalizacije virusa. Nešto drugačiji pristup istom problemu u kratkim crtama predstavljamo u posljednjem dijelu ovog rada.

2.3 Adlemanova formalizacija

Dvije godine nakon objave Cohenovih radova (1986.) svoj nastavak njegove teorije objavljuje i Leonard Adleman. Njegov je pristup vrlo moćan i zanimljiv s obzirom na to da u sebi sadrži sve moguće tipove infektivnih računalnih programa. Adlemanove ćemo ideje i rezultate u kratkim crtama predstaviti u ovom poglavlju, ispuštajući stroge matematičke dokaze s obzirom na to da njegova formalizacija nije glavna tema ovog rada.

Kako bi definirao virus, Adleman proučava sljedeća svojstva (akcije) koja smatra bitnima u smislu definicionih uvjeta koje će postaviti:

1. *Infekcija* - program izvršava svoj inicijalni zadatak nakon čega infektivno djeluje na neki drugi program.

2. *Nanošenje štete* - program izvršava svoj inicijalni zadatak, nakon čega izvršava dodatnu akciju (akcija ne mora nužno biti infekcija). Ova „dodatna" akcija ovisi isključivo o virusu, ne o programu koji je zaražen.

3. *Imitacija* - program izvršava samo svoj inicijalni zadatak. Ova se situacija može promatrati kao specijalni slučaj infekcije, pri čemu je broj programa koji se mogu zaraziti jednak nuli.

U narednom ćemo se izlaganju koristiti oznakama čiji opisi slijede. Prvo, sa S ćemo označiti skup svih konačnih nizova prirodnih brojeva. Sa $e : S \times S \to \mathbb{N}$ označavamo izračunljivu injektivnu funkciju čiji je inverz također injektivan i izračunljiv. Pojam izračunljivosti koristimo intuitivno: postoji algoritam koji određuje vrijednost $e(s,t)$, za $s, t \in S$, te (za inverz) postoji algoritam koji određuje postoje li $s', t' \in S$ takvi da je $e(s', t') = n$, za neki prirodni broj n, i ako postoje, algoritam ih efektivno određuje.

O argumentima ove funkcije možemo razmišljati kao o *kodu* programa i podacima koje taj program dobiva kao ulaz[3]. Vrijednost funkcije e za bilo koja dva niza s i t označavamo s $<s,t>$ i nazivamo *proširenim indeksom funkcije* g ako je niz s zapravo kod nekog programa koji izračunava funkciju g i t neki argument od g, odnosno ulazni podatak za taj program. Za funkciju $f : I \subseteq N \to \mathbb{N}$ sa $f(s,t)$ označavamo $f(<s,t>)$. Ako vrijednost $f(s,t)$ nije definirana, to označavamo s $f(s,t)\uparrow$, a inače $f(s,t)\downarrow$.

Kako bismo formalno definirali karakteristične tri akcije kojima Adleman određuje virus i time pojam virusa zapravo definirali, potrebne su nam sljedeće dvije definicije.

Definicija 2.3.1. *(Slaba ekvivalencija funkcija)*
Neka su f, g, h parcijalne funkcije na skupu prirodnih brojeva, tj. neka je $f, g, h : I \subsetneq \mathbb{N} \to \mathbb{N}$. Kažemo da su funkcije f i g slabo ekvivalentne s obzirom na h *ili* $h-$ekvivalentne u slabom smislu *ako za sve $(s,t) \in S \times S$ vrijedi:*

$$f(s,t)\downarrow, g(s,t)\downarrow \ \text{ i } h(f(s,t)) = g(s,t).$$

U tom slučaju pišemo $f(s,t) \overset{h}{\sim} g(s,t)$.

Uz pomoć slabe ekvivalencije sada definiramo i jaku ekvivalenciju.

Definicija 2.3.2. *(Jaka ekvivalencija funkcija)*
Neka su f, g, h parcijalne funkcije na skupu prirodnih brojeva. Kažemo da su funkcije f i g jako ekvivalentne s obzirom na h *ili* $h-$ekvivalentne u jakom smislu *ako za sve $(s,t) \in S \times S$ vrijedi:*

$$f(s,t) = g(s,t) \text{ ili } f(s,t) \overset{h}{\sim} g(s,t).$$

U tom slučaju pišemo $f(s,t) \overset{h}{\cong} g(s,t)$.

Kako bismo sasvim korektno mogli iskazati definiciju pojma virusa, ipak se moramo podsjetiti na rezultat koji se tiče samog indeksa parcijalno rekurzivne funkcije. Naime, na kolegiju smo „Izračunljivost" dokazali egzistenciju izračunljive funkcije koja svakom prirodnom broju pridružuje neku parcijalno rekurzivnu funkciju. Odnosno imali smo: ako je $m \in \mathbb{N}$, onda je definirana funkcija $\{m\}$. Dakle, ako je $m \in \mathbb{N}$ kod nekog Turingova (RAM) stroja[4], onda $\{m\}(x)$ označava izlazni rezultat rada stroja M s programom x. Taj m nazivamo *indeksom* funkcije (koju stroj izračunava). U slučaju da M nikad ne staje ili m nije kod nekog stroja, stavljamo $\{m\}(x)\uparrow$.

Niz $\{\varphi_i\}$ unarnih funkcija nazivamo *Gödelovim prebrajanjem skupa svih unarnih parcijalno rekurzivnih funkcija* ako taj niz ima sljedeća dva svojstva:

[3]Detalje o kodiranju, indeksima i pripadajućim rezultatima možete vidjeti u [9]. Tvrdnje koje su tamo iskazane dane su u kontekstu RAM strojeva.

[4]Kodiranje Turingovih strojeva je također moguće. Taj rezultat može se naći u [8].

1. φ_i je parcijalno rekurzivna. za svaki broj $i \in \mathbb{N}$,

2. za svaku unarnu parcijalno rekurzivnu funkciju f postoji $i \in \mathbb{N}$ tako da vrijedi $f = \varphi_i$.

Definicija 2.3.3. *(Adlemanova definicija virusa)*
Neka je $\{\varphi_i\}$ Gödelovo prebrajanje skupa svih unarnih parcijalno rekurzivnih funkcija i v neka totalna rekurzivna funkcija. Kažemo da je v virus s obzirom na niz $\{\varphi_i\}$ *ako za sve $p, d \in S$ vrijedi barem jedno od sljedećeg:*

1. $(\forall i, j \in \mathbb{N})\ \varphi_{v(i)}(p, d) = \varphi_{v(j)}(p, d)$,

2. $(\forall j \in \mathbb{N})\ \varphi_j(p, d) \overset{v}{=} \varphi_{v(j)}(p, d)$.

U prvom slučaju kažemo da v nanosi štetu. *U drugom slučaju kažemo da virus v* imitira *$(\varphi_j = \varphi_{v(j)})$ ili* inficira *$(\varphi_j \overset{v}{\sim} \varphi_{v(j)})$.*

Uočimo da smo pisali $\varphi(p, d) = \varphi(<p, d>)$ umjesto, recimo, $\varphi(n), n \in \mathbb{N}$. Oznaka je sugestivna u tom smislu što „razlaže" domenu funkcije (programa) φ_i na program(e) i same ulazne podatke. Dakle, govorimo o programima kao nizu znakova na koje možemo infektivno djelovati, i ulaznim podacima koji su sa stajališta virusa inertni. Nadalje, vidimo da definicija opravdava upotrebu termina infekcije (zaraze), odnosno imitacije i oštećenja s obzirom na uvedene definicije slabe i jake ekvivalencije funkcija.

Sada vidimo da je, s jedne strane, virus naprosto program za Turingov stroj - konačan niz simbola zadane abecede koji ima određena svojstva u određenoj okolini (Cohenova definicija virusa). Ovakav pristup nije mogao obuhvatiti razne vrste virusa kakve danas poznajemo. S druge strane, Adleman predstavlja virus kao rekurzivnu funkciju (naravno, s posebnim svojstvima), ne ograničavajući se nekim konkretnim strojem, odnosno konkretnom abecedom, omogućujući uvođenje definicije raznih programa koje danas nazivamo virusima.

Na temelju definicije virusa dajemo njihovu formalnu klasifikaciju.

Definicija 2.3.4. *(Klasifikacija virusa)*
Neka je $\{\varphi_i\}$ Gödelovo prebrajanje skupa svih unarnih parcijalno rekurzivnih funkcija i v virus s obzirom na taj niz. Neka su $k, j \in \mathbb{N}$ takvi da je $v(j) = k$. Tada redom kažemo:

- *k je* patogen *s obzirom na v te j ako vrijedi:*

$$[\exists d, p \in S\ [\varphi_j(p, d) \overset{v}{=} \varphi_k(p, d)]],$$

- *k je* zarazan *s obzirom na v te j ako vrijedi:*

$$[\exists d, p \in S\ [\varphi_j(p,d) \overset{v}{\sim} \varphi_k(p,d)]],$$

- *k je* bezopasan *s obzirom na v te j ako k nije patogen ni zarazan s obzirom na v te j,*
- *k je* nositelj *s obzirom na v te j ako je k zarazan, ali nije patogen s obzirom na v te j,*
- *k je* trojanski konj *s obzirom na v te j ako je k patogen, ali nije zarazan s obzirom na v te j,*
- *k je* virusan *s obzirom na v te j ako je i patogen i zarazan s obzirom na v te j.*

Ovo poglavlje i rad završavamo intuitivnom nadopunom Cohenovog rezultata koji se tiče detekcije virusa. Rezultat kojeg je Cohen ponudio govori samo o neodlučivosti tog problema, ali je vrlo ograničavajući u smislu detaljnijih informacija, prvenstveno po pitanju složenosti. Znamo da je problem neodlučiv, ali koliko je stvarno težak? Rezultat koji je dokazao Adleman kaže da je odrediti je li neka funkcija virus Π_2-potpun. Što to zapravo znači? Bez formalne definicije samog pojma, a na temelju dosadašnjeg znanja, pogledajmo kako je ova klasa složenosti svakako iznad onog što možemo *efektivno* riješiti. Kao što znamo (vidi [9]), klasa Π_0 zapravo opisuje sve rekurzivne funkcije, dakle u dotičnoj se klasi nalaze svi problemi koje jednostavno možemo riješiti s obzirom na to da su (parcijalno) rekurzivne funkcije izračunljive. No, isto tako znamo da vrijedi $\Pi_i \subsetneq \Pi_{i+1}$. Dakle je $\Pi_0 \subsetneq \Pi_1 \subsetneq \Pi_2$. Da se problem detekcije ne nalazi u Π_0 (čime bismo znali da se efektivno može riješiti) ili pak u Π_1, slijedi upravo iz potpunosti tog problema. Kad bi, dakle, problem detekcije bio i u Π_0 (odnosno Π_1), znali bismo da se svi problemi iz klase Π_2 daju svesti na problem klase Π_0 (odnosno Π_1), a onda bismo imali $\Pi_0 = \Pi_2$ (odnosno $\Pi_1 = \Pi_2$). Iz toga slijedi zaključak kako je navedeni problem svakako „izvan dohvata" efektivnog izračunavanja.

Bibliografija

[1] F. Cohen, *Computer viruses*, Disertacija, University of Southern California, 1986.

[2] F. Cohen, *A Short Course on Computer Viruses*, ASP Press, Pitsburgh, USA, 1986.

[3] E. Filiol, *Computer viruses: from theory to applications*, Springer, Francuska, 2005.

[4] R. Kay i A. Yeun, *Applications of Cellular Automata*, (2009), `http://www.cs.bham.ac.uk/~rjh/courses/NatureInspiredDesign/2009-10/StudentWork/Group2/design-report.pdf`.

[5] S. Srbljić, *Jezični procesori 1*, Element, Zagreb, 2000.

[6] J. Thatcher, *Universality in the von Neumann cellular model*, (1962).

[7] J. von Neumman, *Theory of Self-Reproducing Automata*, University of Illinois Press, 1966.

[8] M. Vuković, *Složenost algoritama*, predavanja, PMF, Zagreb, 2015.

[9] M. Vuković, *Izračunljivost*, predavanja, PMF, Zagreb, 2009.

Sažetak

U ovom radu demonstrirali smo razvoj formalnih koncepata računalnih virusa. Prvo poglavlje daje pregled von Neumannovih staničnih automata koji su sadržavali sve alate kojima se računalni virus mogao definirati i izgraditi. Prvim dijelom drugog poglavlja predstavljamo osnovne rezultate doktorske disertacije Fredericka Cohena koji stvarno započinje razvoj ove teorije. Svakako najvažniji rezultat koji smo dali onaj je koji govori o neodlučivosti virusnog skupa, odnosno virusa. Konačno, drugim dijelom drugog poglavlja otkrivamo način na koji Cohenov mentor, Leonard Adleman, poopćuje teoriju koristeći se rekurzivnim funkcijama.

Summary

In this thesis we have presented formalization of the basic concepts of computer viruses. First chapter deals with von Neumann's cellular automata, which where in themselves sufficient for the definition of a virus and its construction. In the first part of the second chapter we give some of the results Frederick Cohen proved in his PhD thesis. It was Cohen who actually began the development of this theory. The fundemantal result of this theory is undecidability of a viral set, that is the undecidability of a virus. Finally, in the second part of the second chapter we present how Cohen's PhD supervisor, Leonard Adleman, generalized this theory by using recursive functions.

Životopis

Rođen sam 8. ožujka 1993. godine u Zagreb. Godine 1999. upisujem Osnovnu školu Antuna Augustinčića u Zaprešiću, a nakon završetka 2007. i srednju školu Ban Josip Jelačić u istom gradu, smjer opća gimnazija. Srednjoškolsko obrazovanje završavam primitkom nagrade za učenika generacije (2011.), a iste godine upisujem matematiku na Prirodoslovno-matematičkom fakultetu u Zagrebu. Nakon tri godine završavam preddiplomski studij nakon kojeg odmah i upisujem diplomski studij "Računarstvo i matematika" na istom fakultetu. Tijekom posljednje godine studija radio sam u tvrki Corvus info d.o.o. na poziciji programera ("junior software developer"). Kako bih povezao računarstvo i matematiku i u svom završnom radu, kao temu diplomskog rada odaberim "Teorijski model računalnih virusa".