Универзалне Тјурингове машине засноване на ТАГ систему

Мастер рад

Студент:

Борис Вељаноски

Београдски Универзитет

Математички Факултет

Комисија:

проф. др Небојша Икодиновић

проф. др Славко Моцоња

проф. др Филип Марић

Предговор

Тјурингова машина представља математички модел рачунања који дефинише апстрактну машину која манипулише симболима на траци поштујући одговарајућа правила. На њој се заснивају многобројне ствари данашњице који људи узимају здраво за готово попут рачунара и мобилних телефона. Иако је детаљно изучавана, увек постоје нове интриге које заголицају машту, интересантни детаљи које раније нисмо разматрали. Овај рад се заснива на поједностављењу ових гиганата и разматрању најмањих Тјурингових машина које би представљале најједноставнији модел за рачунање, који је довољно моћан да израчуна било коју израчунљиву функцију!

Теза је подељена у три одељка. Први има за циљ да упути читаоца у то шта је заправо универзална Тјурингова машина. Друго упућује мало детаљније о томе шта значи најмања машина, уводи таг систем и описује конструкцију Тјурингове машине која може да симулира рад 2-таг система. Последњи одељак говори о временској сложености 2-таг система и о томе да је ово ефикасна трансформација, јер симулира рад у полиномијалном времену.

Садржај

Глава 1

Универзална Тјурингова Машина

У овој глави биће дефинисана Тјурингова машина и регистар машина, веза између њих и однос са парцијално рекрузивним функцијама.
Биће објашњена примитивна рекурзивност са примерима и детаљно образложен поступак дефинисања аритметизације регистар машине односно формирање универзалне регистар и Тјурингове машине. У раду неће бити разматрана недетрминистичка, већ искључиво детерминистичка Тјурингова машина.

1.1 Тјурингова машина

Конструкција Тјурингове машине рађена је по узору на [3]. За почетак, замислимо бесконачну траку, подељену на ћелије. У сваку ћелију можемо уписати тачно један симбол неког унапред задатог алфабета $\Gamma = \{s_1, \ldots, s_m\}$. У алфабету постоји и специјалан симбол - бланко знак "$\sqcup$".

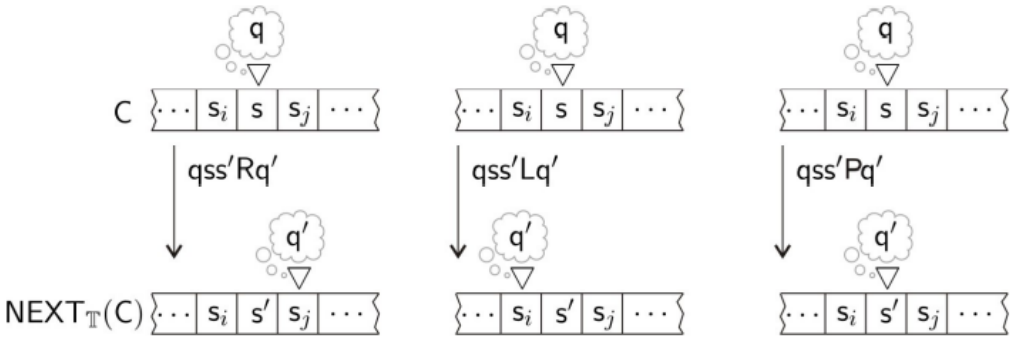
Поред тога, замислимо да постоји и читач инструкција назван глава. Она чита коначни низ инструкција $q_1, \ldots, q_n$ који називамо Тјурингов програм. Под инструкцијом q_i подразумевамо низ уређених петорки чији је q_i почетни симбол. У сваком тренутку глава може да прочита садржај само једне ћелије и затим изврши једну инструкцију. Свака инструкција q_i је следећег облика:

$$\begin{array}{l|l} q_i & s_1 s_1' D_1 q_{j1} \\ & \quad \vdots \\ & s_m s_m' D_m q_{jm} \end{array}$$

или

$$q_i \quad \text{STOP}$$

Значење инструкције је да се учитани симбол s_l брише ($1 \leq l \leq m$), на његово место уписује $s_l^{'}$, да се глава помера на лево (ако је D_l једнако L), десно (ако је D_l једнако R) за једно поље или остаје на истом пољу (ако је D_l једнако P) и да се прелази на извршавање инструкције q_{j_l} ($s_l, s_l^{'} \in \Gamma, 0 \leq j_1, \ldots, j_m \leq n$). Инструкције првог облика можемо записати и као низ уређених петорки $(q_i, s_1, s_1^{'}, D_1, q_{j_1}), \ \ldots, \ (q_i, s_m, s_m^{'}, D_m, q_{j_m})$. Даље у тексту користимо скраћени запис за петорке, тј. $q_i s_1 s_1^{'} D_1 q_{j_1}$. Ознаку инструкције називамо и стањем, при чему ознаку оне којом треба започети извршавање програма (обично q_0) издвајамо као почетно стање. Ознаке свих STOP инструкција називамо завршним стањима. Када кажемо да се глава налази у стању q_i, то значи да треба извршити инструкцију q_i. Конфигурацијом траке називамо садржај свих ћелија траке, положај главе и стање у којем се она тренутно налази. Завршне конфигурације су оне конфигурације у којима је глава у неком завршном стању. Уколико конфигурација C није завршна, онда је Тјуринговим програмом потпуно одређена наредна конфигурација $\mathrm{NEXT}_{\mathbb{T}}(C)$. Пример корака израчунавања Тјурингове машине може се видети на слици 1.1



Слика 1.1: Корак израчунавања приказан у случају да параметар D узима вредности R, L или P редом

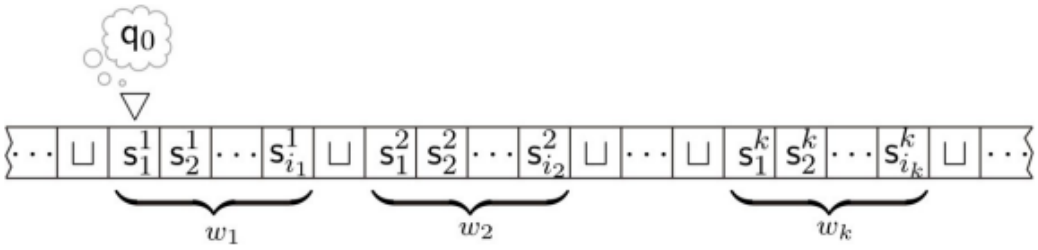
Извршавање Тјурингових програма посматраћемо у ситуацијама када је на почетку на траци уписано коначно много речи унапред одређеног улазног алфабета $\Sigma \subseteq \Gamma \backslash \sqcup$, при чему сматрамо да су:

– симболи сваке речи уписани слева надесно у суседне ћелије траке,

– речи међусобно раздвојене једним бланко знаком и

– у све остале ћелије су уписани бланко знаци.

Овако одређен почетни садржај траке, уз претпоставку да глава у почетном стању скенира прву ћелију слева од које почиње унос улаза називамо почетном конфигурацијом траке. Ако на улазу треба унети k речи алфабета Σ, $w_1 = s_1^1, s_2^1, s_{i_1}^1, \ldots w_2 = s_1^2, s_2^2, s_{i_2}^2, \ldots w_k = s_1^k, s_2^k, s_{i_k}^k$, одговарајућа почетна конфигурација је приказана на слици 1.2.



Слика 1.2: Почетна конфигурација Тјурингове машине

У случају да је w_1 празна реч, глава у почетном стању чита први слева бланко знак испред w_2.

Тјурингова машина (или краће TM) јесте уређена седморка $\mathbb{T} = (Q, q_0, F, \Gamma, \sqcup, \Sigma, \tau)$. Q представља скуп стања, тј. скуп свих инструкција програма, $q_0 \in Q$ је почетно стање, тј прва инструкција којом се започиње извршавање програма, $F \subseteq Q$ скуп завршних стања, тј. скуп STOP-инструкција, Γ је алфабет траке тј. скуп свих симбола који могу бити уписани у ћелију траке, $\Sigma \subseteq \Gamma \setminus \{\sqcup\}$ је улазни алфабет и $\tau : (Q \setminus F) \times \Gamma \to \Gamma \times \{R, L, P\} \times Q$ представља функцију транзиције, тј. Тјурингов програм. Функција транзиције описује инструкције које нису STOP-инсрукције. За уређену петорку $qss'Dq'$ важи $\tau(q, s) = (s', D, q')$.

Најзначајнији део Тјурингове машине јесте њен програм.
Свака Тјурингова машина $\mathbb{T} = (Q, q_0, F, \Gamma, \sqcup, \Sigma, \tau)$ за улаз $\overline{w} = (w_1, \ldots, w_k) \in \Sigma^{*k}$, тј. за одговарајућу почетну конфигурацију $C_{\overline{w}}$, генерише један низ конфигурација:

$$C_{\overline{w}} \to \mathrm{NEXT}_{\mathbb{T}}(C_{\overline{w}}) \to \mathrm{NEXT}_{\mathbb{T}}(\mathrm{NEXT}_{\mathbb{T}}(C_{\overline{w}})) \to \ldots$$

Низ конфигурација је:

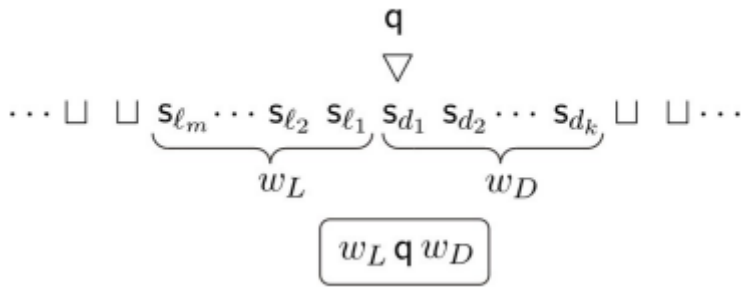
– коначан ако се у низу појави завршна конфигурација; тада пишемо $\mathbb{T}(\overline{w}) \downarrow$ и кажемо $\mathbb{T}$ се зауставља за улаз $\overline{w}$, или

– бесконачан ако се у низу никада не појављује завршна конфигурација; тада пишемо $\mathbb{T}(\overline{w}) \uparrow$ и кажемо $\mathbb{T}$ се не зауставља за улаз $\overline{w}$.

Ако $\mathbb{T}(\overline{w})\downarrow$, коначан низ конфигурација $C_{\overline{w}}, C_1, \ldots, C_d$, називамо израчунавањем машине $\mathbb{T}$ за улаз $\overline{w}$ и пишемо $C_{\overline{w}} \rightarrow_{\mathbb{T}}^{*} C_d$. Ако $\mathbb{T}(\overline{w})\uparrow$, тј. низ конфигурација је бесконачан, израчунавање машине $\mathbb{T}$ се не зауставља за улаз $\overline{w}$ и пишемо $C_{\overline{w}} \rightarrow_{\mathbb{T}}^{*} \infty$.

$C_{\overline{w}}$ је почетна конфигурација одређена улазом $\overline{w}$, две узастопне конфигурације називамо рачунским кораком, а C_d представља завршну конфигурацију.

Увешћемо сада формално појмове конфигурације и функције $\mathrm{NEXT}_{\mathbb{T}}$. Конфигурацију означавамо као реч језика $\Gamma^* Q \Gamma^+$ и имаће облик $w_L q w_D$, односно $w_L \in \Gamma^*$ и $w_D \in \Gamma^+$. Конфигурација је представљена на слици 1.3.



Слика 1.3: Запис конфигурације

У овој формулацији, w_D представља реч коју чине сви симболи десно од главе означене са q, укључујући и симбол на који глава показује. У случају да су сви симболи једнаки бланко знаку, онда је $w_D = \sqcup$; w_L представља реч коју чине сви симболе лево од главе, не укључујући тренутни симбол. Ако су сви симболи једнаки бланко знаку, онда је $w_L = \epsilon$.

Узимајући у обзир овакво означавање конфигурација произвољне Тјурингове машине $\mathbb{T} = (Q, q_0, F, \Gamma, \sqcup, \Sigma, \tau)$, функцију $\mathrm{NEXT}_{\mathbb{T}} : \Gamma^*(Q \backslash F)\Gamma^+ \rightarrow \Gamma^* Q \Gamma^+$ дефинишемо на следећи начин:

$$\mathrm{NEXT}_{\mathbb{T}}(w_L q s w_D) = \begin{cases} w_L s' q' w_D, & \text{ако је } \tau(q,s) = (s', R, q') \text{ и } w_D \neq \epsilon, \\ w_L s' q' \sqcup, & \text{ако је } \tau(q,s) = (s', R, q') \text{ и } w_D = \epsilon, \\ w_L q' s_i s' w_D, & \text{ако је } \tau(q,s) = (s', L, q') \text{ и } w_L = w s_i, \\ q' \sqcup s' w_D, & \text{ако је } \tau(q,s) = (s', L, q') \text{ и } w_L = \epsilon, \\ w_L q' s' w_D, & \text{ако је } \tau(q,s) = (s', P, q'). \end{cases}$$

Дефиниција Функција $f : \Sigma^{*k} \rightarrow \Gamma^*$ је TM-израчунљива ако постоји Тјурингова машина T са улазним алфабетом Σ таква да се за све речи

$w_1, \ldots, w_k \in \Sigma^* : q_0 w_1 \sqcup \cdots \sqcup w_k \to_{\mathbb{T}}^* q_{STOP} f(\overline{w})$

Дефиниција Скуп $A \in N^k$ је ТМ-одлучив ако је ТМ-израчунљива његова карактеристична функција $\chi_A : N^k \to \{0,1\}$.

$$\chi_A(n_1, \ldots, n_k) = \begin{cases} 1, & (n_1, \ldots, n_k) \in A, \\ 0, & (n_1, \ldots, n_k) \notin A. \end{cases}$$

1.2 Регистар Машине

Регистар машине су апстрактне машине „новије генерације". Конструкција регистар машине рађена је по узору на [3]. Замислимо траку која је неограничена само са једне стране и издељена на регистре означене $R_1, R_2, R_3, \ldots$
У сваки регистар може бити уписана читава реч унапред изабраног алфабета $\Sigma = \{s_1, \ldots, s_m\}$. Сем траке, постоји и бројач у који се уноси природни број различит од нуле. Тај број представља редни број инструкције одговарајућег програма.
Замислимо и да постоји механизам који може да чита садржај регистра и да извршава једну од инструкција RM–програма - коначног низа инструкција нумерисаних бројевима од 1 до неког n : $I_1, \ldots, I_n$. Свака инструкција I_i следећег облика:

$$i.R_k^{+s} \mid l \quad (s \in \Sigma, l \geq 1)$$

или

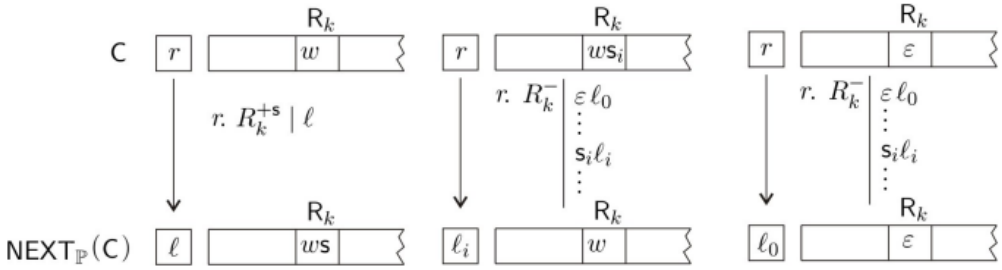
$$i.R_k^{-} \left| \begin{array}{ll} \epsilon & l_0 \quad (l_0, l_1 \ldots, l_m \geq 1) \\ s_1 & l_1 \\ & \vdots \\ s_m & l_m \end{array} \right.$$

Прва инструкција има значење да се у регистар R_k дописује симбол s на крај речи и прелази на извршавање инструкције I_l; друга има значење да ако је празна реч уписана у регистар R_k прелази се на инструкцију I_{l_0}, а у супротном, брише се последњи симбол s_j и прелази на инструкцију I_{l_j}.
Конфигурацију представља низ:

$$(C) \quad r;\ w_1,\ w_2,\ w_3,\ \ldots$$

чији је први члан природни број различит од нуле (уписан у бројач), а остали чланови су речи из Σ^* (уписане редом у регистре R_1, R_2, ...).

Завршна конфигурација програма је она коју тај програм не садржи, односно инструкција r за коју важи $r > n$. Ако (C) није завршна конфигурација програма $\mathbb{P}$, онда је овим програмом одређен јединствени следбеник $\mathrm{NEXT}_{\mathbb{P}}(C)$, тј. конфигурација која се добија завршавањем инструкције I_r. Слика 1.4 представља корак израчунавања регистар машине.



Слика 1.4: Илустрација корака израчунавања регистар машине

Рад RM–програма $\mathbb{P} : I_1, I_2, \ldots, I_n$ посматраћемо искључиво за почетне конфигурације у којима је у бројачу уписан број 1 и у коначно много регистара су уписане неке речи алфабета Σ, а у свим осталим је аутоматски уписана празна реч. Оне су облика:

$$(C_0) \quad 1;\ w_1, \ldots,\ w_k,\ \epsilon,\ \epsilon,\ \epsilon,\ \ldots$$

Сваки RM–програм $\mathbb{P}$ за улаз $(w_1, \ldots, w_k)$ генерише један низ конфигурација:

$$C_0 \to \mathrm{NEXT}_{\mathbb{P}}(C_0) \to \mathrm{NEXT}_{\mathbb{P}}(\mathrm{NEXT}_{\mathbb{P}}(C_0)) \to \ldots$$

Низ конфигурација је:

– коначан ако се у низу појави завршна конфигурација за програм $\mathbb{P}$; тада пишемо $\mathbb{P}(w_1, \ldots, w_k) \downarrow$ и кажемо програм $\mathbb{P}$ се зауставља за улаз $(w_1, \ldots w_k)$; или

– бесконачан ако се у низу никада не појављује завршна конфигурација; тада пишемо $\mathbb{P}(w_1, \ldots, w_k) \uparrow$ и кажемо програм $\mathbb{P}$ се не зауставља за улаз $(w_1, \ldots, w_k)$.

Ако $\mathbb{P}(w_1, \ldots, w_k) \downarrow$, коначан низ конфигурација $C_0, C_1, \ldots, C_d$ називамо израчунавањем програма $\mathbb{P}$ за улаз $(w_1, \ldots, w_k)$ и пишемо $C_0 \to^*_{\mathbb{P}} C_d$.
C_0 је почетна конфигурација одређена улазом $(w_1, \ldots, w_k)$, две узастопне конфи- гурације називамо рачунским кораком и C_d представља завршну

конфигурацију за $\mathbb{P}$. Зарад бољег разумевања регистар машина, уводимо пример израчунавања.

Пример Нека је $\mathbb{P}$ програм над алфабетом $\{0,1\}$:

$1.R_2^-$

ϵ	4
0	2
1	3

$2.R_1^{+0}$ | 1 $\qquad$ $3.R_1^{+1}$ | 1

За улаз (001, 110) програм генерише следећи низ конфигурација:

$$1;001,110 \to 2;001,11 \to 1;0010,11 \to 3;0010,1 \to 1;00101,1$$
$$\to 3;00101,\epsilon \to 1;001011,\epsilon \to 4;001011,\epsilon$$

Како је последња конфигурација наведеног низа и завршна конфигурација за програм $\mathbb{P}$, закључујемо да програм $\mathbb{P}$ конвергира.

Будући да је сваки RM–програм $\mathbb{P}$ коначан низ инструкција наведеног облика, у њему се помиње само коначно много регистара и садржаји само тих регистара могу бити измењени током извршавања програма $\mathbb{P}$. Најмањи природан број $\|\mathbb{P}\|$ такав да се у програму $\mathbb{P}$ помињу само регистри R_i, $i \leq \|\mathbb{P}\|$, назива се дубина програма $\mathbb{P}$. Узимајући у обзир да је на почетку у свим регистрима почев од неког уписана празна реч, закључујемо да ће и током читавог израчунавања садржаји само коначно много регистара бити различити од празне речи. Зато, све достижне конфигурације током рада програма $\mathbb{P}$ можемо означавати коначним низом $i, w_1, \ldots, w_d$, $d = max\{n, \|\mathbb{P}\|\}$, где је n дубина улаза, а $\|\mathbb{P}\|$ дубина програма.

1.2.1 RM–програми за унарни улазни алфабет

Посматрајмо RM–програме за унарни алфабет $\Sigma = \{1\}$. Скуп речи јединичног унарног алфабета $\Sigma = \{1\}$ представља скуп природних бројева $\mathbb{N}$, тј. реч 1^n посматрамо као n. Инструкције RM–програма се за алфабет $\{1\}$ прилично поједностављују.

- Инструкција $R_k^+|l$ значи да се садржају регистра R_k додаје 1 и прелази се на извршавање инструкције чији је редни број l.
- Инструкцију

R_k^-

ϵ	l_0
1	l_1

краће записујемо $R_k^- \mid l_0, l_1$. Ова инструкција значи следеће: ако је садржај регистра R_k једнак 0 прелази се на извршавање инструкције чији је редни број l_0, а ако је садржај регистра R_k различит од 0 умањује се за 1 и прелази се на извршавање инструкције чији је редни број l_1.

Ако је алфабет једнак $\{1\}$, конфигурације постају низови природних бројева чији су сви чланови почев од неког једнаки нули. Нека $Conf$ означава скуп свих конфигурација траке за алфабет $\{1\}$. Сваки RM–програм $\mathbb{P}$ одређује функцију $NEXT_{\mathbb{P}} : Conf \to Conf$. Функција $NEXT$:

- Увећава k-ти регистар за један и прелази на извршавање l-те инструкције, ако је тренутна инструкција $R_k^+ \mid l$,
- Прелази на извршавање l_0-те инструкције, ако је тренутна инструкција $R_k^- \mid l_0, l_1$ и $r_k = 0$,
- Умањује k-ти регистар за један и прелази на извршавање l_1-те инструкције, ако је тренутна инструкција $R_k^- \mid l_0, l_1$ и $r_k \neq 0$,
- Завршава са радом ако тренутна инструкција не постоји

Дефиниција Функција $f : \Sigma^{*k} \to \Sigma^*$ је RM-израчунљива, ако постоји RM–програм $\mathbb{P}$ такав да за све $w_1, \ldots, w_k \in \Sigma^*$ важи $\mathbb{P}(w_1, \ldots, w_k) \downarrow f(w_1, \ldots, w_k)$.

Дефиниција Скуп $A \in N^k$ је RM-одлучив ако је RM-израчунљива његова карактеристична функција $\chi_A : N^k \to \{0, 1\}$.

$$\chi_A(n_1, \ldots, n_k) = \begin{cases} 1, & (n_1, \ldots, n_k) \in A, \\ 0, & (n_1, \ldots, n_k) \notin A. \end{cases}$$

Тврђење Сваки RM–програм може се симулирати Тјуринговом машином.

Тврђење Свака Тјурингова машина може се симулирати RM–програмом.

Конструкција ових програма може се наћи у [3].

Последица За сваку функцију $f : \Sigma^{*k} \to \Sigma^{*k}$ важи:

f је RM-израчунљива акко је TM-израчунљива.

1.3 Парцијалне RM-израчунљиве функције

Сваки RM–програм $\mathbb{P}$ за свако $k \geq 1$ одређује један подскуп скупа $\mathbb{N}^k$ који садржи све оне k-торке природних бројева за које се $\mathbb{P}$ зауставља.

$$W_{\mathbb{P}}^{(k)} = \{(x_1, \ldots, x_k) \in \mathbb{N}^k | \mathbb{P}(x_1, \ldots, x_k) \downarrow\}.$$

Уколико $\mathbb{P}(x_1, \ldots, x_k)$ завршава израчунавање, за резултат узимамо садржај регистра R_1, у завршној конфигурацији. Дакле, програм $\mathbb{P}$, за свако $k \geq 1$ одређује јединствену функцију $\phi_{\mathbb{P}}^{(k)} : W_{\mathbb{P}}^{(k)} \to \mathbb{N}$,

$$\phi_{\mathbb{P}}^{(k)}(\vec{x}) = \text{садржају регистра } R_1 \text{ у завршној конфигурацији израчунавања програма } \mathbb{P}.$$

Узимајући у обзир врсту функција које одређују RM–програми, пажњу усмеравамо на тзв. парцијалне функције и на њих проширујемо појам RM-израчунљивости. Парцијалне k-арне функције су оне које нису дефинисане на целом скупу $\mathbb{N}^k$.

Нека је ϕ парцијална функција арности k. За њен домен D важи $D \subseteq \mathbb{N}^k$. Ако $\vec{x}$ припада домену D кажемо да функција ϕ конвергира, а у супротном дивергира. Скуп вредности функције R је $R \subseteq \mathbb{N}$. Ако функција ϕ конвергира за унос $\vec{x}$ и једнака је вредности y, пишемо $\phi(\vec{x}) \downarrow y$. Формула $\phi(\vec{x}) \simeq \psi(\vec{x})$ значи да или обе вредности нису дефинисане или да су обе вредности дефинисане и једнаке. Једнаке могу бити само две парцијалне функције исте дужине:

$$\phi = \psi \overset{\text{def}}{\Leftrightarrow} \forall \vec{x}(\phi(\vec{x}) \simeq \psi(\vec{x})).$$

Дефиниција Парцијална k-арна функција ϕ је RM-израчунљива ако постоји RM–програм $\mathbb{P}$ такав да за све $\vec{x} \in \mathbb{N}$ важи $\phi(\vec{x}) \simeq \phi_{\mathbb{P}}^{(k)}$, односно:

- ако $\vec{x} \in D$, онда се извршавање програма $\mathbb{P}$ за улаз $\vec{x}$ зауставња и садржај регистра R_1 у завршној конфигурацији је $\phi(\vec{x})$;
- ако $\vec{x} \notin D$, онда се извршавање програма $\mathbb{P}$ за улаз $\vec{x}$ не зауставља.

Лема* Основне функције:

- Нула-функција $0 : \mathbb{N} \to \mathbb{N}, 0(x) = 0$,
- Следбеник $s : \mathbb{N} \to \mathbb{N}, s(x) = x + 1$,
- Пројекције $\Pi_i^k : \mathbb{N}^k \to \mathbb{N}, \Pi_i^k(x_1, \ldots, x_k) = x_i (1 \leq i \leq k)$,

*Доказ се може наћи у [3]

јесу RM-израчунљиве.

Увешћемо опште принципе дефинисања функција за које је скуп свих RM-израчунљивих функција затворен како бисмо показали њихову ширину и моћ.

1.3.1 Супституција

Први принцип који наводимо је супституција. Ако су $g_1, \ldots, g_m$ неке k-арне парцијалне функције и h нека m-арна парцијална функција, онда за k-арну функцију f дефинисану са

$$f(\vec{x}) \simeq h(g_1(\vec{x}), \ldots, g_m(\vec{x})), \vec{x} \in N^k$$

кажемо да је добијена супституцијом функција $g_1, \ldots, g_m$ у h и пишемо $f = \mathrm{Sup}_m^k(h; g_1, \ldots, g_m)$. Ако су $g_1, \ldots, g_m$ неке k-арне функције, које израчунавају редом програми $\mathbb{G}_1, \ldots, \mathbb{G}_m$, и h је нека m-арна RM-израчунљива функција, коју израчунава програм $\mathbb{H}$, онда се може саставити програм $\mathbb{F}$ који за улаз $\vec{x} \in N^k$ рачуна $h(g_1(\vec{x}), \ldots, g_m(\vec{x}))$. Програм $\mathbb{F}$ прати следећу процедуру:

- користећи програме $\mathbb{G}_i$ рачуна функције $g_i(\vec{x})$ и добија резултате $y_i = g_i(\vec{x})$;
- користећи резултате и програм $\mathbb{H}$ рачуна функцију $h(y_1, \ldots, y_m)$ и враћа резултат $z = h(y_1, \ldots, y_m)$

Могуће је да се наведени поступак никада не завршава уколико се не зауставља неки од програма $\mathbb{G}_1, \ldots \mathbb{G}_m$ или $\mathbb{H}$ за одговарајуће улазе.

Формалну дефиницију супституције могуће је наћи у [3].

1.3.2 Примитивна рекурзија

Ако је нека k-арна парцијална функција и h нека $(k+2)$-арна парцијална функција, онда за $(k+1)$-арну функцију f дефинисану са:

$$\left| \begin{array}{l} f(\vec{x}, 0) \simeq g(\vec{x}), \\ f(\vec{x}, y+1) \simeq h(\vec{x}, y, f(\vec{x}, y)), \end{array} \right.$$

кажемо да је добијена примитивном рекурзијом функција g и h и пишемо $f = Rec^{k+1}(g, h)$. Користећи програме $\mathbb{G}$ и $\mathbb{H}$ који редом израчунавају k-арну

функцију g и $(k+2)$-арну функцију h, може се саставити програм $\mathbb{F}$ који за улаз $(\vec{x}, y) \in N^{k+1}$ рачуна функцију $f(\vec{x}, y)$. Програм $\mathbb{F}$ обавља следећи задатак користећи променљиву t, при чему је иницијално $t = 0$:

- користећи програм $\mathbb{G}$ рачуна функцију $g(\vec{x})$ и добија резултат $z = g(\vec{x})$
- ако је $y = 0$ враћа резултат z, у супротном, умањује y за 1
- користећи програм $\mathbb{H}$ рачуна $h(\vec{x}, t, z)$, увећава t за $1(t = t + 1)$ и прелази на други корак.

1.3.3 Неограничена минимизација

Неограничена минимизација одговара поступку потраге за решењем x једначине облика $g(x_1, \ldots, x_k, x) = 0$, где су $x_1, \ldots, x_k$ параметри, а g је нека $(k+1)$-арна RM-израчунљива функција.

Ако је g нека (k+1)-арна парцијална функција, онда за k-арну функцију f такву да је:

$$f_g(\vec{x}) = \mu t(g(\vec{x}, t) = 0) = \begin{cases} t, & t \text{ је најмањи природан број такав да је} \\ & g(\vec{x}, t) = 0 \text{ и важи} \\ & (\forall z < t)(g(\vec{x}, z) \downarrow \wedge g(\vec{x}, z) \neq 0) \\ \uparrow, & \text{иначе.} \end{cases}$$

кажемо да је добијена неограниченом минимизацијом функције g, и пишемо $f = \mathrm{Min}^k(g)$.

Нека је $\mathbb{G}$ RM–програм који израчунава g. Саставимо програм $\mathbb{F}$ који, за задате $x_1, \ldots, x_k$, тражи најмање решење x једначина $g(x_1, \ldots, x_k, x) = 0$. За улаз $\vec{x} \in N^k$, $\mathbb{F}$ користи програм $\mathbb{G}$ да израчуна вредности $g(\vec{x}, i), i \in \{1, 2, 3, \ldots\}$ једну за другом. У тренутку када $g(\vec{x}, i) = 0$, програм враћа излаз i.

Приметимо да је могуће да се наведени поступак никада не заврши и за то постоје два разлога. Први је "заглављивање" у неком кораку због незаустављања програма $\mathbb{G}$. Друга околност може бити та да се успешно рачунају све вредности функције g, али да се никада не добија вредност 0.

Дефиниција Скуп парцијално рекурзивних функција је најмањи скуп у смису инклузије који садржи основне функције (нула-функцију, функцију следбеника и пројекције) и затворен је за супституцију, примитивну рекурзију и неограничену минимизацију.

Теорема Свака парцијално рекурзивна функција је RM-израчунљива.
Доказ. Директна последица претходних разматрања. □

1.4 Примитивно рекурзивне функције и примери

Дефиниција Скуп примитивно рекурзивних функција је најмањи скуп у смислу инклузије који садржи основне функције (нула-функцију, функцију следбеника и пројекције) и затворен је за супституцију и примитивну рекурзију.

Свака примитивно рекурзивна функција је тотална (дефинисана на целом скупу $\mathbb{N}$) и како је парцијално рекурзивна она је и RM односно TM-израчунљива.
Примери примитивно рекурзивних функција:

За произвољне функције $k \geq 1$ и $m \in \mathbb{N}$, консатнтна k-арна функција $m_k = \mathbb{N}^k \to \mathbb{N}, m_k(\vec{x}) = m$ јесте примитивно рекурзивна.
Сабирање, множење и степеновање јесу примитивно рекурзивне функције.
Факторијел је примитивно рекурзивна функција*:

$$\left| \begin{array}{l} 0! = 1, \\ (s(x))! = x! \cdot s(x). \end{array} \right.$$

Претходник је примитивно рекурзивна функција *:

$$\mathrm{pd} : \mathbb{N} \to \mathbb{N} \left| \begin{array}{l} \mathrm{pd}(0) = 0, \\ \mathrm{pd}(s(x)) = x. \end{array} \right.$$

Монус је примитивно рекурзивна фунцкија*:

$$\dot{-} : \mathbb{N}^2 \to \mathbb{N} \left| \begin{array}{l} x \dot{-} 0 = x, \\ x \dot{-} s(y) = \mathrm{pd}(x \dot{-} y). \end{array} \right.$$

Дефиниција Скуп $\mathbb{R} \in \mathbb{N}^k$ (k-арна релација скупа $\mathbb{N}$) је примитивно рекурзиван (-на) ако је таква одговарајућа карактеристична функција $\chi_R : \mathbb{N}^k \to \mathbb{N}$,

$$\chi_R(\vec{x}) = \begin{cases} 1, & \vec{x} \in R, \\ 0, & \vec{x} \in R. \end{cases}$$

Скупови N^+ и $\{0\}$ јесу примитивно рекурзивни. За њихове карактеристичне

*! $= \mathrm{Rec}^1(1, \mathrm{Sup}_2^2(\cdot; \Pi_2^2, \mathrm{Sup}_1^2(s; \Pi_1^2)))$
*pd $= \mathrm{Rec}^1(0, \Pi_1^2)$
*$\dot{-} = \mathrm{Rec}^2(\Pi_1^1, \mathrm{Sup}_1^3(\mathrm{pd}; \Pi_3^3))$

функције χ_{N^+} и $\chi_{\{0\}}$ уводимо посебне ознаке и називе, прву називамо функцијом знака и означавамо sg[*]: $\mathbb{N} \to \mathbb{N}$, а другу називамо функцијом обрнутог знака и означавамо $\overline{\mathrm{sg}}$ [*] : $\mathbb{N} \to \mathbb{N}$:

$$\left| \begin{array}{l} \mathrm{sg}(0) = 0, \\ \mathrm{sg}(s(x)) = 1; \end{array} \right| \left| \begin{array}{l} \overline{\mathrm{sg}}(0) = 1, \\ \overline{\mathrm{sg}}(s(x)) = 0. \end{array} \right.$$

Уређења $\leq$ и $\geq$ су примитивно рекурзивне релације: $\chi_{\leq}(x,y) = \overline{\mathrm{sg}}(x \dot{-} y)$ и $\chi_{\geq}(x,y) = \overline{\mathrm{sg}}(y \dot{-} x)$.
Строга уређења $<$ и $>$ су такође примитивно рекурзивне релације: $\chi_{<}(x,y) = 1 \dot{-} \chi_{\geq}(x,y)$ и $\chi_{>}(x,y) = 1 \dot{-} \chi_{\leq}(x,y)$.

Лема Комплемент примитивно рекурзивног скупа је примитивно рекурзиван скуп.

Доказ. $\chi_{R^C}(\vec{x}) = 1 \dot{-} \chi_R(\vec{x})$ □

Лема Унија и пресек примитивно рекурзивних скупова јесу примитивно рекурзивни скупови.

Доказ. $\chi_{P \cap Q}(\vec{x}) = \chi_P(\vec{x}) \cdot \chi_Q(\vec{x})$; $P \cup Q = (P^C \cap Q^C)^C$ □

Дефиниција Нека је $R \subseteq \mathbb{N}^{k+1}$ нека релација, релације универзалног и егзистенцијалног ограниченог квантификатора $U, E \subseteq \mathbb{N}^{k+1}$ дефинишемо са $U(\vec{x},y) = (\forall z \leq y)R(\vec{x},z)$ и $E(\vec{x},y) = (\exists z \leq y)R(\vec{x},z)$.

Лема Ако је $R \subseteq \mathbb{N}^{k+1}$ примитивно рекурзивна релација, онда су примитивно рекурзивне релације ограничене квантификације.

Доказ. $\chi_U(\vec{x},y) = \prod_{z \leq y} \chi_R(\vec{x},z)$; $\chi_E(\vec{x},y) = \mathrm{sg}\left(\sum_{z \leq y} \chi_R(\vec{x},z)\right)$ □

Дефиниција Нека је $f : \mathbb{N}^{k+1} \to \mathbb{N}$ нека функција, онда функцију $M : \mathbb{N}^{k+1} \to \mathbb{N}$,

$$M_f(\vec{x},t) = (\mu z < t)(f(\vec{x},z) = 0) = \begin{cases} u, & u < t, f(\vec{x},u) = 0 \text{ и} \\ & (\forall v < u) f(\vec{x},v) \neq 0, \\ t, & \text{иначе,} \end{cases}$$

називамо функцијом ограничене минимизације.

Лема Ако је $f : \mathbb{N}^{k+1} \to \mathbb{N}$ примитивно рекурзивна функција, онда је и функција $M : \mathbb{N}^{k+1} \to \mathbb{N}$ примитивно рекурзивна.

[*]$\mathrm{sg} = \mathrm{Rec}^1(0, 1_2)$
[*]$\overline{sg} = \mathrm{Rec}^1(1, 0_2)$

Доказ.

$$M_f(\vec{x}, t) = \sum_{v<t} \prod_{u \leq v} \mathrm{sg} f(\vec{x}, u))$$

□

Целобројни количник, уз договор да је $\left\lfloor \frac{y}{0} \right\rfloor = 0$, јесте примитивно рекурзивна функција:

$$\left\lfloor \frac{y}{x} \right\rfloor = \mathrm{sg}(x) \cdot (\mu k \leq y)(y < (k+1)x).$$

Остатак, уз договор да је $\mathrm{rm}(0, y) = y$, јесте примитивно рекурзивна функција:

$$\mathrm{rm}(x, y) = y \dot{-} \left(\left\lfloor \frac{y}{x} \right\rfloor \cdot x \right).$$

Дељивост је примитивно рекурзивна релација, јер важи

$$\chi_{|}(x, y) = \overline{\mathrm{sg}}(\mathrm{rm}(x, y)).$$

Скуп простих бројева $\mathbb{P} = \{2, 3, 5, 7, 11, \dots\}$ је примитивно рекурзиван.

$$\mathbb{P}(x) \Leftrightarrow x \geq 2 \wedge (\forall y \leq x)(y|x \Rightarrow y = 1 \vee y = x).$$

Низ простих бројева је примитивно рекрузиван, јер се функција $p : \mathbb{N} \to \mathbb{N}$, неформално дефинисана са

$$p(x) = p_x = \text{“}(x+1) - \text{и прост број“},$$

може описати на следећи начин:

$$\begin{aligned} p_0 &= 2, \\ p_{x+1} &= (\mu y \leq p_x! + 1)(\mathbb{P}(y) \wedge y > p_x). \end{aligned}$$

Функција растављања на просте чиниоце

$$(x, y) \to (y)_x = (\mu k \leq y)(p_x^k | y \wedge \neg(p_x^{k+1} | y))$$

је примитивно рекурзивна.
Функција кодирања парова природних бројева, $\langle ., . \rangle : \mathbb{N}^2 \overset{\text{1-1}}{\underset{\text{na}}{\to}} \mathbb{N}$,

$$\langle x_1, x_2 \rangle = 2_1^x(2x_2 + 1) - 1,$$

јесте примитивно рекурзивна. Инверзна функција $\langle .,. \rangle^{-1}$ одређена је примитивно рекурзивним функцијама $\langle . \rangle_1, \langle . \rangle_2 : \mathbb{N} \to \mathbb{N}$,

$$\langle x \rangle_1 = (x+1)_0, \langle x \rangle_2 = \left\lfloor \frac{\frac{x+1}{2^{(x+1)_0}} - 1}{2} \right\rfloor, x \in \mathbb{N}.$$

Бинарна репрезентација броја: Сваки $x \in \mathbb{N}$ има јединствени приказ облика $x = \sum_0^\infty c_{i,x} 2^i, c_i \in \{0,1\}$. Низ (c_i, x) јесте заправо једна примитивно рекурзивна функција $c : \mathbb{N} \times \mathbb{N} \to \{0,1\}, c(i,x) = \mathrm{rm}(2, \left\lfloor \frac{2}{2^i} \right\rfloor)$. Ако је $x > 0$, онда постоји јединствени приказ облика

$$x = 2^{b_1} + 2^{b_2} + \ldots + 2^{b_k}, 0 \leq b_1 < b_2 < \ldots \langle b_k, k \rangle = 1. \qquad (*)$$

Функција lh: $\mathbb{N} \to \mathbb{N}$, којом се за $x > 0$ одређује k у горњем приказу (*):

$$\mathrm{lh}(x) = \begin{cases} k \text{ у приказу } (*) \text{ }, x > 0, \\ 0, \qquad\qquad x = 0, \end{cases}$$

јесте примитивно рекурзивна: $\mathrm{lh}(x) = \sum_{i<x} c(i,x)$. Таква је и функција $b : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, којом се одређују изложиоци b_i:

$$b(i,x) = \begin{cases} b_i, \text{ у приказу } (*), & x > 0 \text{ и } 1 \leq i \leq \mathrm{lh}(x), \\ 0, & \text{иначе.} \end{cases}$$

$$= \begin{cases} (\mu k < x) \left(\sum_{j \leq k} c(j,x) = i \right), & x > 0 \text{ и } 1 \leq i \leq \mathrm{lh(x)}, \\ 0, & \text{иначе.} \end{cases}$$

Кодирање коначних низова природних бројева:

$$\begin{aligned} \lceil () \rceil &= 0 (\text{кôд празног низа је } 0) \\ \lceil x_1, \ldots, x_k \rceil &= 2^{x_1} + 2^{x_1+x_2+1} + \cdots + 2^{x_1+x_2+\ldots+x_k+k-1} \\ &= (1 \underbrace{0 \ldots 0}_{x_k \text{ нула}} 1 \ldots 0 1 \underbrace{0 \ldots 0}_{x_1 \text{ нула}})_2 \end{aligned}$$

Функција $\lceil . \rceil$ није примитивно рекурзивна будући да је њен домен $\cup_{k \geq 0} \mathbb{N}^k$, али постоје примитивно рекурзивне функције којма се реконструише низ чији је кôд задат:

$$\mathrm{lh}(x) = \text{дужина низа чији је кôд x},$$

док нам следећа примитивно рекурзивна функција омогућава одређивање чланова низа чији је кôд задат:

$$(i,x) \to \lfloor x \rfloor_i = \begin{cases} \text{i-ти члан низа чији је кôд x, } x > 0, 1 \leq i \leq \mathrm{lh}(x), \\ 0, \qquad\qquad\qquad \text{иначе.} \end{cases}$$

$$= \begin{cases} b(i,x) \dot{-} (b(i \dot{-} 1, x) + 1), & x > 0, 1 \leq i \leq \mathrm{lh}(x), \\ 0, \qquad\qquad\qquad\qquad\quad \text{иначе.} \end{cases}$$

Ако је $x > 0$, онда је $x = \lceil \lfloor x \rfloor_1, \ldots, \lfloor x \rfloor_{\mathrm{lh}(x)} \rceil$.

1.5 Свака RM-израчунљива функција је парцијално рекурзивна

Сада доказујемо да је свака RM-израчунљива функција парцијално рекурзивна. Посматрамо функцију $\mathrm{NEXT}_{\mathbb{P}} : \mathrm{Conf} \to \mathrm{Conf}$,
$\mathrm{NEXT}_{\mathbb{P}}(i; r_1, \ldots, r_k, \ldots, r_d, 0 \ldots)$

$$= \begin{cases} (l; r_1, \ldots, r_k + 1, \ldots, r_d, 0, \ldots), & i\text{-та инструкција у } \mathbb{P} \text{ је } R_k^+|l, \\ (l_0; r_1, \ldots, r_k, \ldots, r_d, 0, \ldots), & i\text{-та инструкција у } \mathbb{P} \text{ је } R_k^-|l_0, l_1 \text{ и } r_k = 0, \\ (i; r_1, \ldots, r_k - 1, \ldots, r_d, 0, \ldots), & i\text{-та инструкција у } \mathbb{P} \text{ је } R_k^-|l_0, l_1 \text{ и } r_k \neq 0, \\ (0; r_1, \ldots, r_k, \ldots, r_d, 0, \ldots), & i\text{-та инструкција у } \mathbb{P} \text{ не постоји.} \end{cases}$$

Скуп свих конфигурација Conf идентификујемо са скупом свих низова природних бројева чији су чланови почев од неког једнаки нули, па можемо кодирати све конфигурације природним бројевима (свакој конфигурацији једнозначно придружујемо кôд те конфигурације):

$$(i, r_1, r_2, \ldots, r_d, 0, 0, 0, 0 \ldots) \to p_0^i p_1^{r_1} \ldots p_d^{r_d}$$

Нека је c тренутна конфигурација

$$c = p_0^i p_1^{r_1} \cdots p_k^{r_k} \cdots$$

Користећи уведено кодирање дефинисаћемо примитивно рекурзивну функцију $\text{next}_{\mathbb{P}} : \mathbb{N} \to \mathbb{N}$, такву да за сваку конфигурацију C важи $\text{next}_{\mathbb{P}}(\text{кôд}(\text{C})) = \text{кôд}(\text{NEXT}_{\mathbb{P}}(\text{C}))$.

- Ако је у програму $\mathbb{P}$ i-та инструкција $P_k^+|l$:

$$\text{next}_{\mathbb{P}}(c) = \frac{c}{p_0^{(c)_0}} \cdot p_0^l \cdot p_k$$

- Ако је у програму $\mathbb{P}$ i-та инструкција $P_k^-|l_1, l_2$ и $r_k \neq 0$:

$$\text{next}_{\mathbb{P}}(c) = \frac{c}{p_0^{(c)_0} \cdot p_k} \cdot p_0^{l_2}$$

- Ако је у програму $\mathbb{P}$ i-та инструкција $P_k^-|l_1, l_2$ и $r_k = 0$:

$$\text{next}_{\mathbb{P}}(c) = \frac{c}{p_0^{(c)_0}} \cdot p_0^{l_1}$$

- Ако је у програму $\mathbb{P}$ не постоји i-та инструкција:

$$\text{next}_{\mathbb{P}}(c) = \frac{c}{p_0^{(c)_0}}$$

Лема 1 За сваки RM–програм $\mathbb{P}$ функција $\text{next}_{\mathbb{P}} : \mathbb{N} \to \mathbb{N}$ јесте примитивно рекрузивна.

Теорема Свака RM-израчунљива функција је парцијално рекурзивна. Доказ се може наћи у [3].

Последица Функција је RM-израчунљлива акко је парцијално рекурзивна.

1.6 Клинијева теорема о нормалној форми

Парцијана функција f је интуитивно израчунљива ако постоји „коначан опис поступка израчунавања" вредности те функције. Према претходним резултатима постоје различити међусовбно еквивалентни проступи у дефинисању коначног описа поступка израчунавања - TM, RM,... Најважнија чињеница својствена сваком од приступа јесте постојање универзалног поступка реализације било којег од „коначног описа" за задати улаз.

Универзални поступак ћемо описати једном парцијално рекурзивном функцијом, када природним бројевима искодирамо све што чини RM–програме и њихова израчунавања.

1.6.1 Кодирање инструкција RM–програма

Програми су коначни низови инструкција, стога ћемо прво њих кодирати природним бројевима. Подсетимо се да је $\langle \cdot, \cdot \rangle$ функција кодирања парова са 14. стране. Дефинишемо биијекцију $[\cdot]$ која свакој RM-инструкцији додељује природан број - кôд те инструкције:

$$[R_k^+|l] = \langle 2(k-1), l\rangle, k \geq 1, l \geq 0, \text{ и}$$
$$[R_k^-|l_1, l_2] = \langle 2(k-1)+1, \langle l_1, l_2\rangle\rangle, k \geq 1, l_1, l_2 \geq 0.$$

За сваки природај број n, једноставно је наћи инструкцију чији је кôд n. Декодирање $[\cdot]^{-1}$ је одређено следећим једнакостима:

$$[n]^{-1} = \begin{cases} R^+_{\left[\frac{\langle n\rangle_1}{2}\right]+1}\big|\langle n\rangle_2, & \text{ако } 2 \mid \langle n\rangle_1 \\ R^-_{\left[\frac{\langle n\rangle_1}{2}\right]+1}\big|\langle\langle n\rangle_2\rangle_1, \langle\langle n\rangle_2\rangle_2, & \text{ако } 2 \nmid \langle n\rangle_1 \end{cases}$$

1.6.2 Кодирање RM–програма

Користећи уведено кодирање RM-инструкција, сваком RM–програму придружујемо кôд низа који чине кôдови инструкција тог програма: програму $\mathbb{P} = I_1, I_2, \ldots I_m$ додељујемо кôд

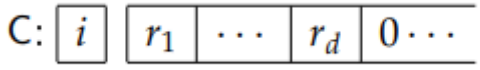
$$[\![\mathbb{P}]\!] = \ulcorner [I_1], [I_2] \ldots, [I_m] \urcorner = 2^{[I_1]} + 2^{[I_1]+[I_2]+1} + \ldots + 2^{[I_1]+\cdots+[I_m]+m-1}.$$

Функција $[\![\cdot]\!]$ јесте бијекција скупа свих RM–програма и скупа природних бројева. За било који природан број n, да бисмо одредили који програм одговара датом кôду n, најпре треба одредити низ чији је кôд једнак n, затим одредити инструкције чији су кодови чланови тог низа:

$$[\![n]\!]^{-1} = \begin{cases} \text{кôд прве инструкције програма чији је кôд једнак } n, \text{ тј. } [\lfloor n \rfloor_1]^{-1} \\ \vdots \\ \text{кôд последње инструкције програма чији је кôд једнак } n, \\ \text{тј. } [\lfloor n \rfloor_{\mathrm{lh}(n)}]^{-1} \end{cases}$$

1.6.3 Кодирање конфигурација и функција next

Свака конфигурација може се посматрати као низ природних бројева који су почев од неког једнаки нули, илустрација се може видети на слици 1.5. Конфигурације кодирамо природним бројевима већим од нуле:



Слика 1.5: Слика конфигурације регистар машине

$$[\mathrm{C}] = p_0^i p_1^{r_1} \ \ldots \ p_d^{r_d}$$

Свакој конфигурацији придружујемо јединствени кôд, а сваки природан број различит од нуле јесте кôд једне конфигурације коју добијамо растављањем броја на просте чиниоце.

Универзални поступак заправо је заснован на функцији NEXT чији је први аргумент RM–програм, а други нека конфигурација. За сваки RM–програм $\mathbb{P}$ и сваку конфигурацију C,

$$\mathrm{NEXT}(\mathbb{P}, C) = \text{конфигурација која се добија из C извршавањем одговарајуће инструкције програма } \mathbb{P}.$$

Разматрања из Леме 1 указују на универзалност аритметизације свих функција $\mathrm{NEXT}_{\mathbb{P}}$. То доводи до аритметичке верзије функције NEXT, тј. до функције next: $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$, такве да за сваки програм и сваку конфигурацију C важи једнакост $\mathrm{next}([\![\mathbb{P}]\!], [\![C]\!]) = [\![\mathrm{NEXT}(\mathbb{P}, C)]\!]$:

$next(e,c)$ = кôд наредне конфигурације у односу на конфигурацију
чији је кôд c током извршавања програма чији је кôд e.

Лема Постоји примитивно рекурзивна функција next: $N^2 \to \mathbb{N}$ таква да за сваки програм $\mathbb{P}$ и сваку конфигурацију C важи једнакост next($[\![\mathbb{P}]\!], [\![C]\!]$) = $[\![\text{NEXT}(\mathbb{P}, C)]\!]$.
Доказ се може наћи у [3].

1.6.4 Кодирање израчунавања и Клинијеви предикати

Сваком коначном низу конфигурација $J = C_1, C_2, \ldots, C_n$ придружујемо кôд низа који чине кôдови појединих чланова $[\![J]\!] = \lceil [C_1], [C_2], \ldots, [C_n] \rceil$. Поставља се питање, да ли коначан низ конфигурација $C_1, \ldots C_n$ представља израчунавање програма $\mathbb{P}$ за улаз $\vec{x} \in N^k$.

- Прво проверавамо да ли је C_1 одговарајућа конфигурација за улаз $\vec{x}$.
- Затим да ли је $\text{NEXT}(\mathbb{P}, C_i) = C_{i+1}, 1 \leq i < n$.
- На крају, да ли је C_n завршна конфигурација за програм $\mathbb{P}$.

Коришћењем досадашњих кодирања, налазимо аритметичку варијанту овог поступка познату као Клинијев предикат:
$T_k(e, \vec{x}, z) \overset{\text{def}}{\Leftrightarrow} z$ је кôд израчунавања програма чији је кôд e за улаз $\vec{x}$

$\Leftrightarrow$ • $\lfloor z \rfloor_1$ је кôд почетне конфгируације за улаз $\vec{x}$, тј. у бројачу је уписано 1, $\vec{x}$ је уписано у $R_1, \ldots R_k$, а у осталим регистрима, који се помињу у програму e, је уписана 0,

$$(\lfloor z \rfloor_1)_0 = 1,$$
$$(\lfloor z \rfloor_1)_i = x_i, 1 \leq i \leq k,$$
$$\forall j \leq \text{lh}(e)(1 \leq j \wedge \left[\frac{\langle \lfloor e \rfloor_j \rangle_1}{2}\right] \geq k \Rightarrow (\lfloor z \rfloor_1)_{\left[\frac{\langle \lfloor e \rfloor_j \rangle_1}{2}\right]+1} = 0)$$

- за $1 \leq i < \text{lh}(z)$, $\lfloor z \rfloor_{i+1}$ је кôд следбеника конфигурације $\lfloor z \rfloor_i$ током

извршавања програма чији је кôд e,

$$\text{next}(e, \lfloor z \rfloor_i) = \lfloor z \rfloor_{i+1}, 1 \leq i < \text{lh}(z)$$

- $\lfloor z \rfloor_{\text{lh}(z)}$ је кôд завршне конфигурације програма чији је кôд e.

$$\left(\lfloor z \rfloor_{\text{lh}(z)}\right)_0 > \text{lh}(e) \vee \left(\lfloor z \rfloor_{\text{lh}(z)}\right)_0 = 0$$

Предикат $T_k(e, \vec{x}, z)$ је конјукција свих наведених формула, стога је примитивно рекурзиван. Резултат израчунавања неког RM–програма за задати улаз сматрамо садржај регистра R_1 у завршној конфигурацији, дефинишимо функцију $U : \mathbb{N} \to \mathbb{N}$:

$U(z) =$ садржај регистра R_1 у последњој конфигурацији чији је кôд z,

тј. $U(z) = (\lfloor z \rfloor_{\text{lh}(z)})_1$.

Теорема (Клинијева теорема о нормалној форми) Постоји примитивно рекурзивна функција $U : \mathbb{N} \to \mathbb{N}$ и за свако $k \geq 1$ примитивно рекурзивна $(k+2)$-арна релација T_k тако да за свако $\vec{x} \in \mathbb{N}^k$ важи:

- $\vec{x} \in \text{dom}(\phi_e^{(k)}) \Leftrightarrow \exists z T_k(e, \vec{x}, z)$,
- $\phi_e^{(k)} \simeq U(\mu z T_k(e, \vec{x}, z))$.

Последица За свако $k \geq 1$, парцијална $(k+1)$-арна функција

$$(e, \vec{x}) \to U(\mu z T_k(e, \vec{x}, z)) \qquad (*)$$

јесте пацијално рекурзивна. Функцију (*) означавамо $\Phi_U^{(k)}$ и називамо универзалном функцијом за k-арне парцијално рекурзивне функције, јер је

$$\Phi_U^{(k)}(e, \vec{x}) \simeq \phi_e^{(k)}(\vec{x}), \text{ за све } e \in \mathbb{N}, \vec{x} \in \mathbb{N}^k.$$

$\Phi_U^{(k)}(e, \vec{x})$ је резултат извршавања RM–програма чији је кôд e за улаз $\vec{x}$ ако се то израчунавање зауставља, а у супротном $\Phi_U^{(k)}(e, \vec{x})$ није дефинисано.

Можемо кодирати све могуће улазе тако што сваком $\vec{x} \in \cup_{k \geq 1} \mathbb{N}^k$ доделимо кôд $\lceil \vec{x} \rceil$. То би нас довело до рекурзивног предиката $T \subseteq \mathbb{N}^3$:

$T(e, x, z) \Leftrightarrow z$ је кôд израчунавања програма чији је кôд e за улаз чији је кôд x.

Бинарна функција Φ дефинисана са

$$\Phi(e,x) \simeq U(\mu z T(e,x,z)),$$

јесте парцијално рекурзивна, па постоји RM-програм U који је израчунава. Овај програм називамо универзалним RM–програмом.
Ово се истиче као кључно запажање које је покренуло изградњу модерних рачунара.

1.7 Универзална Тјурингова машина

Пре дефинисања универзалне Тјурингове машине, морамо све Тјурингове машине да кодирамо речима унапред задатог алфабета.

Све ТМ могу се кодирати речима алфабета $\{0,1\}$. Стања Тјурингове машине означавамо са q_i, а симболе траке са s_i, за неки природан број i. За сваку Тјурингову машину $\mathbb{T} = \{Q, q_0, F, \Gamma, \sqcup, \Sigma, \tau\}$, могу се изабрати природни бројеви $m \geq 0$ и $n \geq 0$, и скуп $\{i_1, \ldots, i_k\} \subseteq \{0, \ldots, m\}$ такви да важи:

$$\begin{aligned} Q &= \{q_0, \ldots, q_m\}, \\ F &= \{q_{i_1}, \ldots, q_{i_k}\}, \\ \Gamma &= \{s_0, \ldots, s_n\}, \\ \sqcup &= s_0, \\ \Sigma &= \{s_1, \ldots, s_n\}. \end{aligned}$$

Функцију прелаза τ представља низ уређених петорки облика $(Q \backslash F) \times \Gamma \times \Gamma \times \{R, L, P\} \times Q$. Свака петорка овог низа мора да има јединствен пар прве две координате.

Стање q_i, кодирамо речју $q$bin(i) тј. $q_0 \equiv q0, q_1 \equiv q1, q_2 \equiv q10, q_3 \equiv q11, \ldots$ а симбол s_i речју $s$bin(i), аналогно. Сваку ТМ можемо задати као реч алфабета $\{0,1,q,s,R,L,P,;\}$ која је следећег облика:

$$q_0\text{bin}(m); q_0; q\text{bin}(i_1)\text{bin}(i_k); s_0\text{bin}(n); s_0; s\text{bin}(1)\text{bin}(n); \overline{\tau} \qquad (*)$$

при чему је $\overline{\tau}$ реч добијена надовезивањем речи

$$q\text{bin}(i)s\text{bin}(j)s\text{bin}(j')Dq\text{bin}(i')$$

које су преводи одговарајућих петорки $q_i s_j s_{j'} D q_{j'}$ низа τ. Кодирањем симбола алфабета $\{0, 1, q, s, R, L, P, ;\}$ трословним речима:

$$\begin{pmatrix} 0 & 1 & q & s & R & L & P & ; \\ 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \end{pmatrix}$$

реч облика (*) преводимо у реч алфабета $\{0, 1\}$ коју ћемо назвати кôдом Тјурингове машине $\mathbb{T}$ и обележавати је $[\![\mathbb{T}]\!]$. Постоји бесконачно много речи над $\{0, 1\}$ које нису кодови ниједне Тјурингове машине.

Скуп

$$\text{Syntax} = \{t \in \{0,1\}^* | t \text{ је кôд неке ТМ}\}$$

је рекурзиван. Скица доказа у [3].

На основу наведеног кодирања ТМ кодирамо и речи улазног алфабета. Улаз $\overline{w} = (w_1, \ldots, w_k) \in \Sigma^{*k}$ који представља одговарајући почетни садржај траке

$$w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k \qquad (+)$$

описујемо наводећи редом кодове свих симбола који се појављују у (+) и на тај начин формирамо кôд улаза $[\![\overline{w}]\!] \in \{0, 1\}^*$.

Лема Језик

$$\text{Input} = \{(t, w) \in \{0,1\}^{*2} | t \text{ је кôд неке ТМ и } w \text{ је кôд коначног низа речи улазног алфабета Тјурнгове машине чији је кôд } t\}$$

је рекурзиван.

Сваку реч C која може бити конфигурација неке ТМ кодирамо на исти начин. Тај кôд означавамо са $[\![\mathrm{C}]\!]$.

Теорема Постоји Тјурингова машина $\mathbb{U}$ таква да за сваку Тјурингову машину $\mathbb{T}$ и коначни низ $\overline{w}$ речи њеног улазног алфабета важи:

$$\mathbb{U}([\![\mathbb{T}]\!], [\![\overline{w}]\!]) \uparrow \text{ акко } \mathbb{T}(\overline{w}) \uparrow;$$

ако $\mathbb{U}([\![T]\!], [\![\overline{w}]\!]) \downarrow$, онда је завршна конфигурација овог израчинавања једнака $q^{\mathbb{U}}_{stop}[\![\mathrm{C}]\!]$, где је C завршна конфигурација израчунавања машине $\mathbb{T}$ за улаз $\overline{w}$. Доказ ове теореме следи из постојања универзалног RM-програма и односу ТМ и RM-програма.

Глава 2

Најмање универзалне Тјурингове машине и Таг систем

У овом поглављу ће бити објашњено шта то представљају појмови најмање универзалне Тјурингове машине, таг система и веза између парцијалних рекурзивних функција и таг система. Затим ћемо описати конструкцију Тјурингове машине која симулира рад 2-таг система.

2.1 Најмање машине

Дефинисали смо универзалну Тјурингову машину. Она је заправо коначни аутомат са бесконачном траком која симулира било коју другу машину уколико јој је дат њен кôд.

Због овога је могуће да посматрањем једне ТМ закључимо нешто ново о некој другој ТМ. Универзална Тјурингова машина може израчунати било коју рекурзивну функцију и препознати било који рекурзиван језик. Према Черч-Тјуринговој тези, проблеми решиви универзалном Тјуринговом машином су управо они за које постоји алгоритам или ефективни метод израчунавања.

Фасцинантно је да тако нешто не само да постоји, већ може да буде и поприлично једноставно у својој структури. Клод Шенон* је први експлицитно поставио питање налажења најмањих могућих машина које су универзалне Тјурингове машине 1956. године. Доказао је да постоји машина која користи само два стања ако имамо довољно симбола траке и обратно. Такође је доказао да увек можемо размењивати симболе траке за стања, као и чињеницу да не

*Claude Shannon - амерички математичар, инжењер и криптограф

постоји универзална Тјурингова машина са једним стањем. Марвин Мински* је открио универзалну Тјурингову машину са 7 стања и 4 симбола користећи 2-таг системе 1962. године. Ако означимо са (m, n) класу универзалних Тјурингових машина са m стања и n симбола, до сада су откривене: (15,2), (9,3), (6,4), (5,5), (4,6), (3,8) и (2,18).
Ако уопштимо модел стандардне Тјурингове машине, још мање универзалне машине су могуће. Једна од генерализација била би дозвољавање бесконачно понављајуће речи с једне или обе стране уноса Тјурингове машине. Ово бисмо назвали "полу-слабом", односно "слабом" универзалношћу. Мала слабо универзална Тјурингова машина која може да симулира Правило 110* може се направити са (6,2), (3,3) и (2,4) стање-симбол паром.

2.2 Таг систем

Таг систем је детерминистички модел израчунавања који је дефинисао Емил Пост 1943. године. Представља апстрактну машину, названу Постов таг систем, која има коначан број стања и бесконачну траку која функциоинише по FIFO принципу тако што чита симбол са врха реда, брише одређени број елемената са врха и додаје на реп низа секвенцу елемената која у потпуности зависи од учитаног симбола. У некој од ранијих верзија проблема, таг систем није брисао елементе након читања главе, него је постојао знак који се помера за одређени број позиција надесно и одређује докле смо стали са обрадом речи. Стога је решавање проблема било утврђивање да ли ће знак сустићи десни крај речи, као дечија игра 'шуге'- па је тако и настало име†.

2.3 Дефиниција

Таг систем се састоји од алфабета симбола $a_{1,m}$, и скупа трансформација у зависности од тих симбола. За сваки симбол a_i постоји секвенца $a_{i_1}, \ldots, a_{i_{n_i}} = P_i$ симбола из истог алфабета. Секвенца P_i садржи n_i симбола и дужина може да варира од симбола до симбола. Таг систем такође карактерише и цео број P. Претпоставимо да је дата почетна реч. Чита се најлевљи симбол. Ако је то симбол a_i, додаје се P_i на крај речи. Затим се брише првих P симбола са

*Marvin Minsky - амерички професор електронике и информатичар.
*аутомат који је за одређени унос Тјуринг комплетан
†tag - шуга

почетка речи и итерира се процес. Процес се прекида ако цела реч нестане или ако наиђемо на посебан заустављајући симбол a_H.

Пример Посматрајмо 2-таг систем са азбуком $\Sigma = \{a, b, c, H\}$. Нека су правила:

$$a \to ccbaH$$
$$b \to cca$$
$$c \to cc$$

За унос $w = baa$, поступак израчунавања ће бити следећи:

$$baa \to acca \to caccbaH \to ccbaHcc \to baHcccc$$

$$\to Hcccccca \qquad \text{(halt)}$$

2.4 Еквивалентност таг система са парцијално рекурзивним функцијама

За сваку парцијално рекурзивну функцију $T(k)$ можемо наћи таг систем еквивалентан $T(k)$ у следећем смислу - нека азбука таг система садржи бар следеће симболе a, A, b, B са особином: Ако систем применимо на Aa^{2^k} и ако је вредност $T(k)$ дефинисана, таг систем ће генерисати реч $Bb^{2^{T(k)}}$, при чему симболом B не почиње ни једна друга ниска - B је заустављајући симбол који завршава процес. Таг систем и парцијална функција $T(k)$ су еквивалентни. Овај резултат користимо да сведемо проблем конструкције УТМ-а на еквивалентан проблем конструкције ТМ-а који може да симулира рад произвољног таг система.

Наводимо пример 2-таг система који одговара проблему Колацове* хипотезе*.

*Lothar Collatz - немачки математичар

*Колатцова хипотеза - понављајући наредне две операције над бројем, ћемо ли икада трансформисати произвољан цео број у јединицу:

$$f(n) = \begin{cases} \frac{n}{2}, & \text{if } n \equiv 0 (\text{mod} 2) \\ 3n+1, & \text{if } n \equiv 1 (\text{mod} 2) \end{cases}$$

Пример Посматрајмо 2-таг систем са азбуком $\Sigma = \{a, b, c\}$. Нека су правила:

$$a \to bc$$
$$b \to a$$
$$c \to aaa$$

За унос $w = aaa$, поступак израчунавања ће бити следећи:

$$aaa \to abc \to cbc \to caaa \to aaaaa$$
$$\to aaabc \to abcbc \to cbcbc \to cbcaaa$$
$$\to caaaaaa \to aaaaaaaa \to aaaaaabc \to aaaabcbc$$
$$\to aabcbcbc \to bcbcbcbc \to bcbcbca \to bcbcaa$$
$$\to bcaaa \to aaaa \to aabc \to bcbc \to bca$$
$$\to aa \to bc \to a \qquad \text{(halt)}$$

2.5 Конструкција (6,6) Тјурингове машине која симулира рад 2-таг система

С обзиром да за сваку парцијално рекурзивну функцију постоји таг систем који јој је еквивалентан, проблем тражења универзалне Тјурингове машине можемо заменити проблемом тражења Тјурингове машине која симулира рад произвољног таг система.
У наредним секцијама ћемо се бавити конструкцијом Тјурингове машине која симулира рад 2-таг система и објашњавањем како она функционише.

Табела прелаза (6,6) Тјурингове машине је приказана у продужетку:

2.5.1 Опис траке

Трака универзалне машине је подељена на седам регија. Почевши слева, налази се:

1. Терминирајућа регија:

учитани симбол	стање					
	q_1	q_2	q_3	q_4	q_5	q_6
O	XL	XRq_3	ALq_4	XRq_5	YLq_4	$HALT$
I	BR	BRq_1	BR	ILq_2	BR	IR
A	XLq_2	BR	AR	AL	AR	ORq_1
B	IL	IL	-	IL	-	AR
X	YR	BLq_6	XR	XL	XR	OR
Y	XL	XL	YR	YL	YR	OR

Табела 2.1: Свака ћелија представља уписани симбол, смер кретања и ново стање (уколико се стање мења)

O	X	I	A	A	O

Ова секција се користи за враћање услова на почетне након брисања речи са почетка ниске над којом тренутно радимо, као и за коначно заустављање када наиђемо на завршни симбол.

2. Регија брисања:

I	I	I	I	$\cdots$	I	I	I

Ову регију карактерише веома велик број, Q, I-ова. Користи се при брисању.

3. Продукциона регија:

$$(P_m)(P_{m-1})\cdots(P_1)$$

Овај регион садржи шифру за произвођење ниски за сваки сибмол a_i. Састоји се од O-ова и I-ова. Сваком симболу a_i таг система, пристаје број N_i који ћемо дефинисати мало касније. Нека је $a_{i_1},\ldots,a_{i_{n_i}}$ низ симбола у који се a_i пресликава у таг систему, тј. $a_i \to a_{i_1},\ldots,a_{i_{n_i}}$. Тада ће стринг P_i у продукционој регији имати облик

$$(P_i) \;=\; IIO^{N_{in_i}}OI\cdots IO^{N_{i_2}}OIO^{N_{i_1}}O$$

P_i садржи $\overline{n_i+1}$ I-ова.

4. Раздвајајућа регија: Ова регија садржи велики број R, I-ова.

$$I\ I\ I\cdots I\ I\ I = I^R$$

Служи да направи довољно велике бројеве N_i , дефинисаћемо их касније.

5. Обрисана регија: Састоји се од O-ова и представља места где је таг систем обрисао симболе са почетка ниске.

6. Регија тренутне речи: Ова регија садржи симболичну репрезентацију ниске $a_\alpha, \ldots, a_\omega$ над којом таг систем оперише. Има облик:

$$Y^{N_\alpha} A Y^{N_\beta} A Y^{N_\gamma} \cdots A Y^{N_\omega}$$

7. Празан простор: Ово је остатак десног дела траке, садржи O-ове.
Цела трака има следећи облик:

$OXIAAO$	I^Q	$(P_m)\cdots(P_1)$	I^R	$OOO\cdots OOO$	$Y^{N_\alpha}A\cdots AY^{N_\omega}$	$OOO\cdots$
I	II	III	IV	V	VI	VII

2.6 Опис процеса

Један корак израчунавања таг система можемо поделити на четири концептуална корака рада Тјурингове машине:

(1) читање првог симбола радне ниске и лоцирање одговарајућег продукционог (P_i) региона

(2) прављење копије продукционе ниске (P_i) са десне стране радне ниске

(3) брисање P симбола, са почетка радне ниске и

(4) ажурирање услова на стање као пре почетка овог процеса тј. припрема за наредну итерацију

Фаза (1) се извршава уклањањем Y-а са почетка регије 6 и тражењем одговарајуће продукционе ниске у регији 3. Пажљиво темпирани број N_α једнозначно одређује локацију продукционе ниске. Завршавамо фазу наилажењем на прво A региона 6.

Фаза (2), тј. фаза копирања, врши копирање продукционе ниске (P_i) лоциране у претходном кораку, у регију 7. Прво O регије празног простора претвара у A, док остале O-ове претвара у Y-е. У тренутку наилажења на двоструко I (назнаке краја (P_i)), копирање се прекида.

Сада почиње фаза (3). Она брише P симбола са почетка радне ниске. Један симбол је већ обрисан у првој фази. Брисање се врши механизмом из прве

фазе. Брише се један по један симбол са почетка речи над којом радимо и "тражи" се одговарајућа продукциона ниска. Због конструкције машине, сваки симбол од другог до P-тог (први симбол у првој фази успева!) неће наићи на своју продукциону ниску, већ ће се наћи у регији 2 - регији брисања. У њој се не налазе O-ови, што значи да се никакво копирање неће извршити. Дужина регија 2 и 4 је одређена тако да обрише тачно P симбола и након тога пробија у регију 1.

Последња, фаза (4), долази у регију 1, затим враћа траку у стандардни формат и цео процес се понавља. Заустављање процеса ће се десити у две околности: ако машина стане тачно на најдешњем O прве регије што се дешава само уколико наиђемо на заустављајући симбол a_H оригиналне Тјурингове машине $\mathbb{T}$, или ако нестану сви симболи радне ниске.

2.7 Детаљи описа траке

Продукциона ниска (P_i) садржи $(n_i + 1)$ I-јева, за једно више од броја симбола ниске у коју се симбол a_i таг система пресликава. Дефинишимо N_i на следећи начин:

$$N_i = \sum_{j=1}^{i-1}(n_j + 1) + R,$$

R ћемо мало касније дефинисати. Када бисмо пребројали N_i I-јева почевши од десне стране четврте регије налево, стигли бисмо до почетка продукционе ниске (P_i) на траци. Стога користимо секвенце Y^{N_i} за одговарајуће a_i у регији 6. Дефинишимо број S као број свих I-јева у региону 3:

$$S = \sum_{i=1}^{m}(n_i + 1)$$

и дефинишимо $R = PS$, где је P број таг система (2 у случају 2-таг система). За свако i важи следеће: $R \leq N_i < R + S$. Посматрамо произвољни низ од P симбола, $N_{i_1}, N_{i_2}, \cdots N_{i_P}$ и добијамо следећу релацију:

$$n_i + 1 + \sum_{j=1}^{P-1} N_{i_j} < S + (P-1)(R+S) = S + (P-1)R + (P-1)S =$$

$$(P-1)R + PS = PR,$$

док с друге стране имамо

$$n_i + 1 + \sum_{j=1}^{P} N_{i_j} > n_i + 1 + PR > PR.$$

Изрази с леве стране представљају број I-јева означен при обради редом $P-1$ и P симбола. Поставимо $Q = (P-1)R - S$, да би тоталан број I-јева у регионима 2, 3 и 4 био тачно PR. Ове две неједнакости обезбеђују да завршимо у регији 1 након P симбола и притом гарантују да се то не може догодити након $P-1$ симбол! Ове чињенице осигуравају да се тачно P симбола брише у свакој итерацији процеса.

2.7.1 Симбол заустављања

Да би се извршавање процеса завршило, потребан нам је посебан симбол да, када га учитамо, машина прекине са радом. Заустављање може да се догоди у два случаја од којих се први дешава када нестане цела радна ниска. Тада машина креће из прве регије надесно у стању q_6, наилази на прво O празне регије и завршава са радом.
Друга околност се догађа ако машина пристигне на најдешње O регије 1 у стању q_2. Ово се догађа ако претходна фаза означи тачно последње I са леве стране друге регије, тј. ако машина означи тачно PR I-ова у тренутној итерацији процеса. Симбол N_i је специјално одабран да се ово не догађа у регуларним околностима, то нам гарантују неједнакости

$$\sum^{P-1} < PR < \sum^{P}.$$

Да бисмо издејствовали заустављање, морамо да натемпирамо да се означавање заустави тачно на крају регије 2. То ћемо постићи користећи специјалну терминирајућу секвенцу, продукциону ниску

$$(P_H) = I\ I\ O^{PR}\ O\ I\ O^{PR}\ O$$

Претпоставимо да се продукција ископира и да се ниска $AY^{PR}AY^{PR}$ налази у радној регији. Када стање q_1 процесуира први Y^{PR} блок, или ће завршити са радом на левом крају региона 2, остварујући услов заустављања, или ће у стању q_1 ући у регион 1, почистити и вратити траку у почетно стање. У другом случају, након оперисања над другим AY^{PR} блоком, маркирање ће се

завршити тачно на жељеном месту, и заустављање ће наступити. Стога (P_H) користимо као продукциону ниску која одговара симболу заустављања a_H.

2.7.2 Детаљи функционисања

Помоћ разумевању и сликовитијем разматрању детаља функционисања биће испис програма (кôд програма се може наћи у додатку) који симулира рад Тјурингове машине на конкретном примеру са слике ??.
Наиме, азбуку чине $\Sigma = \{a, b, c\}$, правила прелаза су $a \to bc$, $b \to a$ и $c \to aaa$, а почетна реч је $w = aaa$.
Константе редом одређујемо: $P = 2$, $n_i = \{2, 1, 3\}$, $S = 9$, $R = 18$, $Q = 9$ и $N_i = \{18, 21, 23\}$.
Иницијална конфигурација траке је дата на слици 2.7.

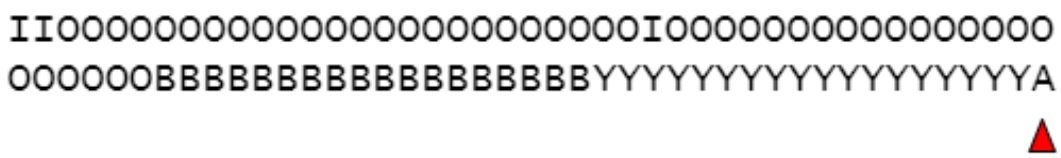
Процес почиње у стању q_1 на првом Y у регији 6; Ово Y прелази у X и машина се мрда налево (мењајући O у X) све док не наиђе на I, које мења у B. Тада креће надесно (мењајући X у Y) док не наиђе на Y.

Мења Y у X и поново креће налево тражећи ново I. Када иде налево, мења Y и O у X и B у I, а када иде надесно мења X у Y и I у B. За прву фазу одређивања локације продукционе ниске довољно је једно стање. Слика 2.1 илуструје како за свако Y машина означава једно левље I.

```
IIIIIIIIIIIIIIIIXYYYYYYYYYYYYYYYYA
IIIIIIIIIIIIIIIIBXYYYYYYYYYYYYYYYA
IIIIIIIIIIIIIIIIBYYYYYYYYYYYYYYYA
IIIIIIIIIIIIIIIIBYXYYYYYYYYYYYYYA
IIIIIIIIIIIIIIIIBXXYYYYYYYYYYYYYA
IIIIIIIIIIIIIIIIIXXYYYYYYYYYYYYYA
IIIIIIIIIIIIIIIIBIXXYYYYYYYYYYYYA
IIIIIIIIIIIIIIIIBBXXYYYYYYYYYYYYA
IIIIIIIIIIIIIIIIBBYXYYYYYYYYYYYYA
IIIIIIIIIIIIIIIIBBYYYYYYYYYYYYYYA
IIIIIIIIIIIIIIIIBBYYXYYYYYYYYYYYA
IIIIIIIIIIIIIIIIBBYXXYYYYYYYYYYYA
IIIIIIIIIIIIIIIIBBXXXYYYYYYYYYYYA
IIIIIIIIIIIIIIIIBIXXXYYYYYYYYYYYA
IIIIIIIIIIIIIIIIIIXXXYYYYYYYYYYYA
IIIIIIIIIIIIIIIIBIIXXXYYYYYYYYYYA
```

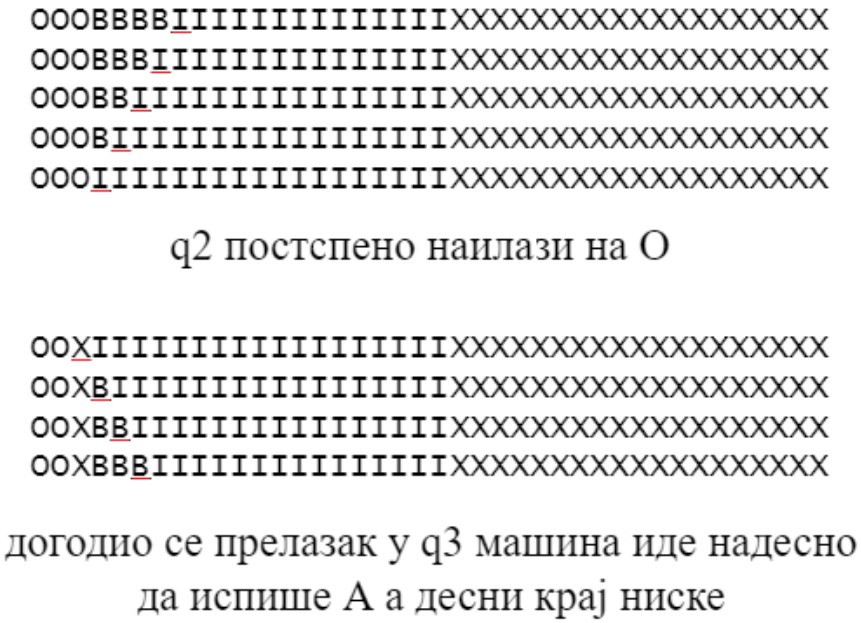
Слика 2.1: Сваки ред представља део траке у суседним итерацијама израчунавања. Црвена линија приказује померање главе Тјурингове машине

У тренутку наилажења на прво слово A, машина мења стање на q_2 и почиње фаза копирања, илустрација на слици 2.2. У том тренутку је означено свако I до почетка одговарајуће продукционе ниске.



Слика 2.2: Део траке је подељен у два реда зарад прегледности. Црвена стрелица означава положај главе. У тренутку наиласка на A прелази се у стање q_2, а симболи B директно упућују на локацију од које почиње копирање

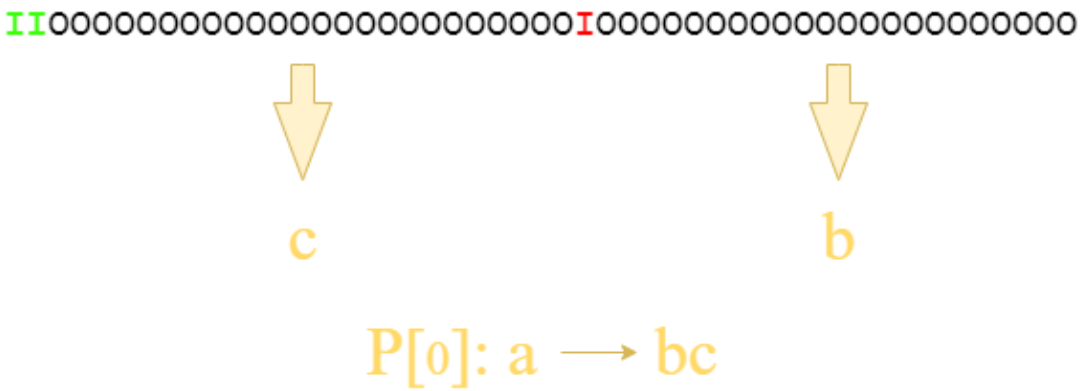
Фаза (2) процеса израчунавања користи стања q_2, q_3, q_4 и q_5. Стања q_3, q_4 и q_5 се баве копирањем, док q_2 служи да одлучи када се копирање зауставља. Машина се помера налево у стању q_2 док не наиђе на O, тада прелази у стање q_3, илустрација на слици 2.3. У том стању се креће надесно и исписује A на почетак празне регије. Затим прелази у стање q_4.



Слика 2.3: Прелазак из стања q_2 у q_3

Стање q_4 се креће налево тражећи O или I. Ако наиђе на O, прелази у q_5. У том стању исписује Y у празан регион и прелази у стање q_4. Пар стања

(q_4,q_5) копира било који број O-ова продукционе регије у Y-оне празне регије. Ако q_4 крећући се налево наиђе на I, враћа се у стање q_2, и помера се налево. Ако q_2 прочита O, исписује се ново A у празној регији и наставља се копирање (копирамо нови симбол продукционе ниске). У супротном, ако прочита I, то значи да смо наишли на крај продукционе ниске тј. на двоструко I, па прелазимо у стање q_1. Илустрација дела продукционе регије које се односи на прво правило 2-таг система из примера може се видети на слици 2.4

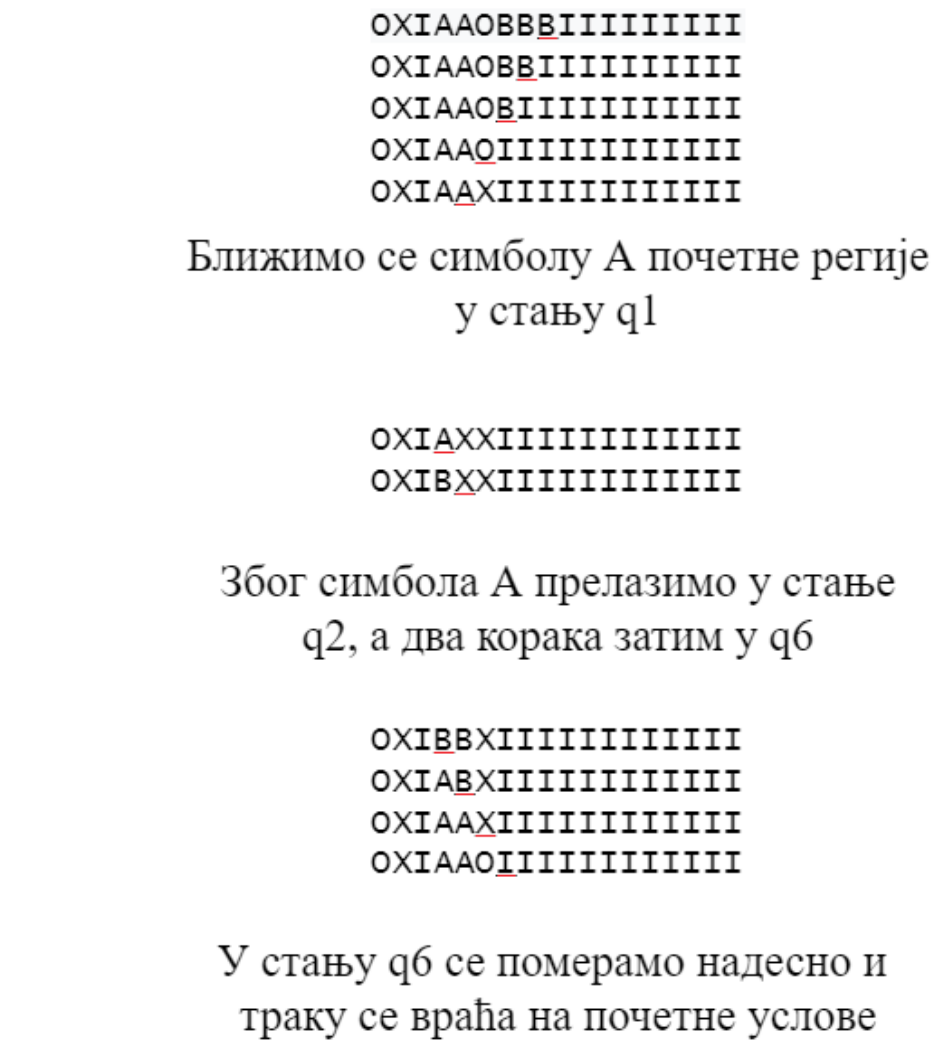


Слика 2.4: Ако се вратимо на слику једне од продукционих ниски, видимо разлику између једног I и двоструког II. Двоструко означава крај продукционе ниске, а само једно представља размак између различитих симбола истог правила

Фаза (3), односно фаза брисања, је већ садржана у фази (1). Фаза (3) ће "појести" још $P-1$ симбол, али за разлику од фазе (1) која завршава у продукционој регији и копира O-ове на десни крај радне ниске, она завршава у регији два, и нема шта да копира (јер се у њој налазе само симболи I).

Фаза враћања услова се догађа када машина наиђе на прво A региона 1 у стању q_1, илустрација на слици 2.5. Исписаће X које ће бити детектовано два корака касније у стању q_2, и послаће машину у стање q_6. B-ови који су управо написани ће бити измењени A-овима, сви O-ови и I-ови враћени у почетно стање и сви неискоришћени Y-они са почетка шесте регије ће бити обрисани. При сусрету са првим A из регије рада тренутне ниске, машина га брише, враћа се у почетно стање q_1 и отпочиње наредни корак израчунавања. Производ рада машине у стању q_6 може се видети на слици 2.6.

Стање q_6 може да наиђе на O само у специјалним околностима с обзиром да су O-ови из регија од 2 до 6 смењени X-овима у претходним фазама. O које



Слика 2.5

q_6 стање може видети је почетно O, у околностима да смо ушли у ту регију у стању q_2, или прво O регије празног простора у случају да је таг систем "прогутао" целу радну ниску, чиме је израчунавање завршено.

2.7.3 Пример (4, 7) машине

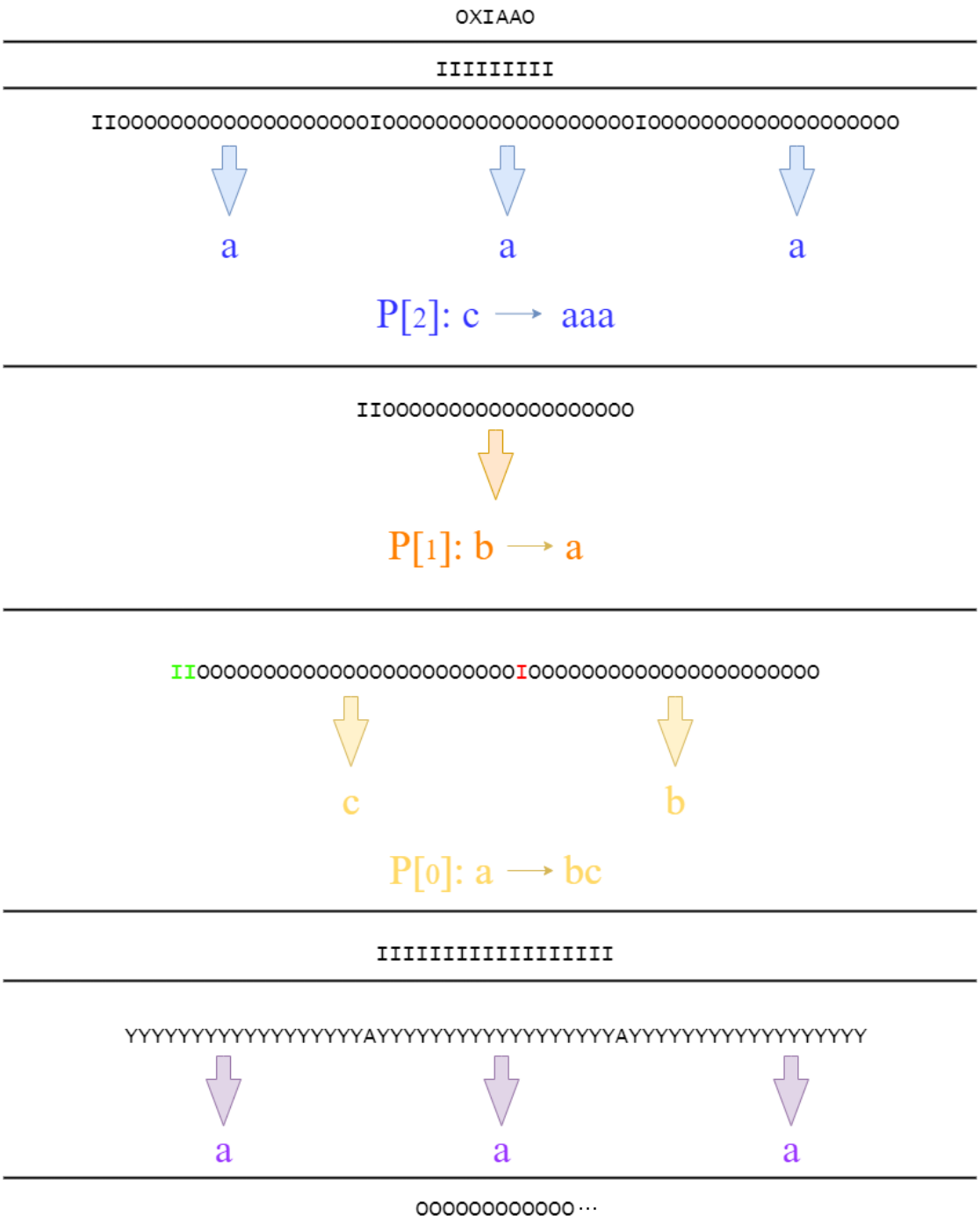
Марвин Мински је успео да конструише и једноставнију машину са мањим производом стања и симбола (4,7). Ово се заснивало на открићу њега и Џона Коука* да постоје универзални Таг системи са бројем брисања P=2.

*John Cocke - амерички информатичар.

Слика 2.6: Траке су подељене у два дела ради прегледности. Прва трака представља почетак рада стања q_6, док друга представља крај. У тренутку мењања A у O машина прелази у стање q_1 и почиње наредни корак израчунавања. Зеленом бојом је означен најдешњи крај продукционе регије, црвеном регија раздвајања, док је наранџастом означен почетак регије тренутне речи, која је у овом тренутку обрисана и постала обрисана регија (ово значи да се ради о првом брисању, односно крају управо првог корака израчунавања 2-таг система).

Елиминисане су 1, 2 и 4 регија, осим тога стање траке је исто сем што се сваки продукциони блок P_i завршава са I, N_i је $1 + \sum_{j=1}^{i-1}(n_i + 1)$ и (P_H) је $IIOIOI$. Структура машине је

учитани симбол	стање						
	q_1	q_2	q_3	q_4	q_5	q_6	q_7
Y	OL	OLq_2	YL	YL	YR	YR	OR
O	OL	YR	$HALT$	YRq_5	YLq_3	ALq_3	YRq_6
I	ILq_2	AR	AL	ILq_7	AR	AR	IR
A	IL	YLq_3	ILq_4	IL	IR	IR	ORq_2



Слика 2.7:
Прва слика означава почетни регион
Друга означава регион брисања
Наредне три представљају продукциони регион, по једна за свако правило и дата је илустрација која приказује која продукциона ниска одговара којем правилу
Идућа означава регион раздвајања
(обрисана регија не постоји, јер смо тек почели извршавање израчунавања)
Седма означава регион у којем радимо над задатом речи. Приказано је да та реч на нашем примеру гласи "aaa"
Последња означава празан простор

Глава 3

Временска сложеност симулације 2-таг системом

Отворено питање које је мучило људе четрдесет година било је, "Да ли најмање познате универзалне Тјурингове машине могу ефикасно симулирати рад Тјурингових машина?". Ово је тесно везано са проблемом временске сложености 2-таг система. Коук и Мински су показали да 2-таг систем симулира Тјурингову машину, али експоненцијално споро.
Мала УТМ симулира рад Тјурингове машине пратећи следећи низ симулација:

$$\text{ТМ} \rightarrow \text{2-таг систем} \rightarrow \text{мала УТМ} \qquad (*)$$

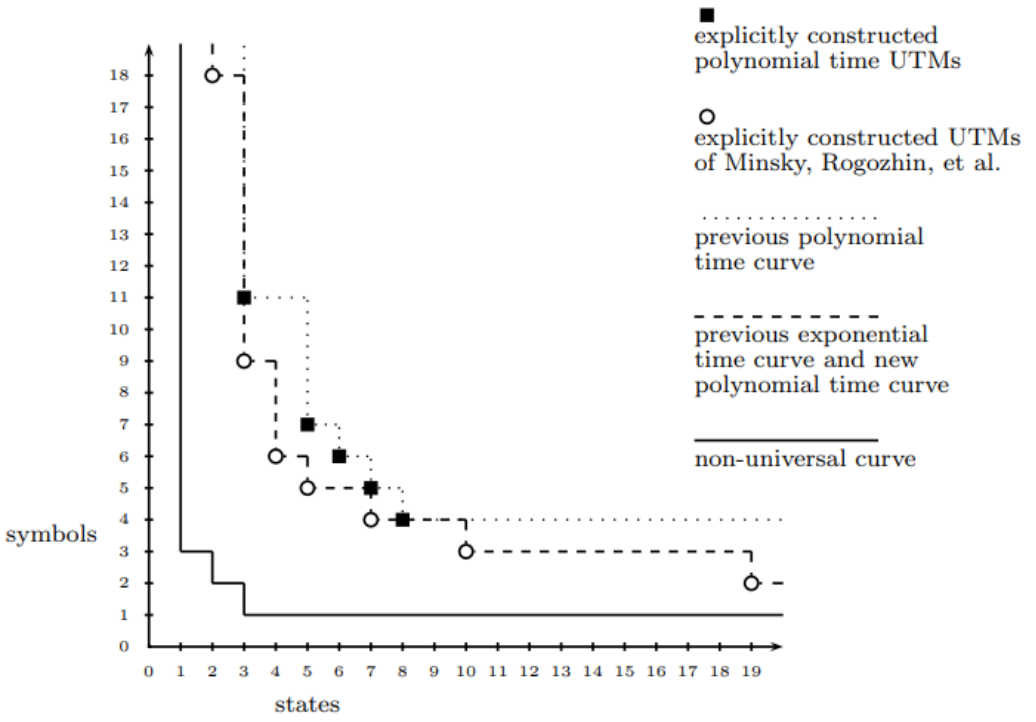
где А $\rightarrow$ Б значи да је А симулирано од стране Б. Користећи технику симулације (*) Рогожин* и Мински су открили УТМ-ове разних стање-симбол парова (3.1) који ефикасно симулирају 2-таг системе. Али с обзиром да је тада симулација Тјурингових машина 2-таг системима била експоненцијално спора, није се знало да ли најмање УТМ могу да симулирају рад у полиномијалном времену.

3.1 Приступ

Да бисмо доказали да је симулација Тјурингове машине малом УТМ-ом ефикасна, посматраћемо следећи низ симулација:

Тјурингова машина $\rightarrow$ циклични таг систем $\rightarrow$ 2-таг систем $\rightarrow$ мала УТМ

*Yurii Rogozhin - информатичар



Слика 3.1: График који приказује временску сложеност малих УТМ-ова.

Полиномијална симулација ТМ-а цикличним таг системом доказана је у [2]. Овде ћемо показати ефикасну симулацију 2-таг система цикличним таг системом.

3.1.1 Уводна терминологија

Нека је $\mathbb{N} = \{0, 1, 2 \ldots .\}$, Σ означава коначну азбуку и ϵ означава празну реч. Функција $|| \cdot || : \Sigma^* \times \Sigma^* \to \mathbb{N}$ написана као $||w_1, w_2||$ означава број појављивања речи w_2 у w_1. На пример $||1101001, 01|| = 2$. Сви логаритми имају основу 2. Симболи a и $\#$ су константе и x и $x_i \in \{0, 1\}$. Дефинишемо парност речи да буде непарна ако је њен најлевљи симбол без тачке, парна у супротном.

Овде ће бити формулисана алгебарска дефиниција 2-таг система.

Дефиниција (2-таг систем) Таг систем се састоји од коначне азбуке Σ, коначног скупа правила $R : \Sigma \to \Sigma^*$ и бројем брисања $\beta \in \mathbb{N}$, $\beta \geq 1$. Код 2-таг система $\beta = 2$. Посматрамо искључиво детерминистичке 2-таг системе. Израчунавање 2-таг система врши се над речју $w = \sigma_1 \sigma_2 \ldots \sigma_l$. Целу

конфигурацију представља реч w. У кораку израчунавања, симболи $\sigma_1\sigma_2$ се бришу и ако постоји правило за σ_1, нпр. $\sigma_1 \to \sigma_{l+1}\ldots\sigma_{l+c}$, суфикс $\sigma_{l+1}\ldots\sigma_{l+c}$ је додат на реч w. Записујемо $w_1 \vdash w_2$ када конфигурација w_2 проистекне из w_1 пратећи један корак израчунавања:

$$\sigma_1\sigma_2\sigma_3\ldots\sigma_l \vdash \sigma_3\ldots\sigma_l\sigma_{l+1}\ldots\sigma_{l+c}$$

при чему $\sigma_1 \to \sigma_{l+1}\ldots\sigma_{l+c} \in R$. 2-таг систем завршава израчунавање ако:

1. $|w| < \beta$
2. уђемо у низ понављајућих конфигурација
3. не постоји правило за најлевљи симбол.

Мерење временске и просторне сложености је стандардно. За дату реч w, користимо термин заокружити да опишемо $\lceil \frac{|w|}{2} \rceil$ корака израчунавања да прођемо кроз w тачно једном. Углавном записујемо симболе 2-таг система у паровима. Други (парни) симбол има тачку да би се разликовао од првог. У продужетку кодирамо бинарне бројеве на следећи начин - 1 кодирамо са $1\dot{1}$ и 0 са $0\dot{0}$. Такође јединствени пар симбола разликујемо од осталих надлинијом: $\bar{0}\dot{\bar{0}}$ или $\bar{1}\dot{\bar{1}}$. Пример кодирања речи 11010 је $\bar{1}\dot{\bar{1}}1\dot{1}0\dot{0}1\dot{1}0\dot{0}$.

Лема Нека $w = \bar{x}_0\dot{\bar{x}}_0x_1x_2\ldots x_l \in \{\bar{0}\dot{\bar{0}}, \bar{1}\dot{\bar{1}}\}\{0,1\}^*$. Онда постоји 2-таг систем T који проверава да ли је $|w|$ паран или непаран у тачно $\lfloor \frac{|w|}{2} \rfloor + 1$ корака.

Доказ. Нека 2-таг систем има 6 правила, $R = \{\bar{x} \to \underset{1}{\bar{x}}\underset{1}{\dot{\bar{x}}}, x \to \epsilon, \underset{1}{\bar{x}} \to \underset{2}{x}\}$ где $x \in \{0,1\}$. На почетку T чита најлевљи симбол $\bar{x}_0$ речи w. Након једног корака, ако прочитани симбол има тачку, онда је излаз симбол $\underset{1}{\dot{\bar{x}}}$ означавајући да је $|w|$ непарна. У супротном је излаз $\underset{2}{x}$ означавајући да је $|w|$ парна. □

3.2 Циклични таг системи

Цикличне таг системе је осмислио Кук* [1].

Дефиниција (циклични таг систем) Циклични таг систем $C = \alpha_0, \alpha_1, \ldots, \alpha_{p-1}$ је листа бинарних речи $\alpha_m \in \{0,1\}^*$ названих продужеци.

Конфигурација цикличног таг система се састоји од маркера који показује на појединачни продужетак a_m у C, и речи $w = x_0x_{1l} \in \{0,1\}^*$. Листа C

*Matthew Cook - информатичар

представља програм на који маркер показује у инструкцији α_m. На почетку рада, маркер показује на продужетак α_0, а w представља бинарну улазну реч.

Дефиниција (корак израчунавања цикличног таг система) Корак израчунавања је детерминистичан и може се извршити на један од два начина:

- ако је $x_0 = 0$ онда x_0 бришемо и померамо маркер на продужетак $\alpha_{(m+1)\bmod_p}$.
- ако је $x_0 = 1$ онда x_0 бришемо, реч α_m додајемо на десни крај речи w, а маркер померамо на продужетак $\alpha_{(m+1)\bmod_p}$.

Циклични таг систем завршава израчунавање ако је w празна реч или ако уђе у понављајући низ конфигурација. Временску и просторну сложеност дефинишемо на уобичајен начин.

Пример (израчунавање цикличног таг система) Нека је $C = 00, 010, 11$ циклични таг систем са почетном речју 011. Испод је наведено првих неколико корака израчунавања. С леве стране налазе се продужеци, маркирани продужетак је у загради, а са десне стране налази се реч над којом радимо.

$$\begin{aligned} &(00), 010, 11 \quad 011 \vdash 00, (010), 11 \quad 11 \\ \vdash\ &00, 010, (11) \quad 1010 \vdash (00), 010, 11 \quad 01011 \\ \vdash\ &00, (010), 11 \quad 1011 \vdash \cdots \end{aligned}$$

Корак израчунавања у општем случају има следећи облик

$$\begin{aligned} &\alpha_0, \ldots, \alpha_{m-1}, (\alpha_m), \alpha_{m+1}, \ldots, \alpha_{p-1} \quad x_0 x_{1l} \\ \vdash &\alpha_0, \ldots, \alpha_{m-1}, \alpha_m, (\alpha_{m+1}), \ldots, \alpha_{p-1} \quad x_0 x_{1l} x_{l+1} \cdots x_{l+c} \end{aligned} \tag{1}$$

при чему $x \in \{0,1\}$, ако је $x_0 = 0$ онда $x_{l+1l+c} = \epsilon$, у супротном ако је $x_0 = 1$ онда је $x_{l+1l+c} = \alpha_m \in \{0,1\}^c, c \in \mathbb{N}$.

Теорема 1 Нека је M детерминистична Тјурингова машина са једном траком, која врши израчунавање у времену t. Онда постоји циклични таг систем C_M који симулира израчунавање M-а за време $O(t^3 \log t)$.
Више о овоме се може наћи у [2].

3.3 2-таг системи ефикасно симулирају циклични таг систем

3.3.1 Кодирање

Циклични таг систем користи бинарни алфабет. Извршавање програма контролише учитани симбол и вредност маркера инструкција. 2-таг систем изгледа општије од цикличног таг система, јер је произвољно велика азбука дозвољена. С друге стране, 2-таг системи имају више ограничења по питању контроле тока, јер зависе само од учитаног симбола. Због овог ограничења користимо велики број симбола у конструкцији. Број симбола је константа независна од дужине уноса, али зависна од алгоритма симулације и величине програма симулираног цикличног таг система. У кодирању, симболима додајемо тачке ($\dot{x}, \ddot{x}$), надцрте ($\bar{x}$) и под-индексе ($\underset{j}{x}$). Ови додаци се користе за алгоритам контроле тока.

Дефиниција 1 (кодирање уноса 2-таг система) Почетна ниска цикличног таг система $w = x_0 x_{1n} \in \{0,1\}^*$ је кодирана као реч 2-таг система

$$\hat{w} = \underset{1}{\bar{x}_0}\underset{1}{\vec{x}_0}\underset{1}{x_1}\underset{1}{\dot{x}_1}\ \underset{1}{x_2}\underset{1}{\dot{x}_2}\ \ \ldots\ \underset{1}{x_n}\underset{1}{\dot{x}_n}\ \underset{11}{a\dot{a}}\ \underset{11}{a\dot{a}}\ \ \ldots\ \underset{11}{a\dot{a}}$$

где је број $a\dot{a}$ парова у $\hat{w}$ $||\hat{w}, a\dot{a}|| = 2^{\lceil \log(n+1) \rceil}$ и додатне белине између парова симбола су стављене искључиво због прегледности.
Ово кодирање је израчунљиво у логаритамском простору. Подреч $a\dot{a}\ a\dot{a}\ \ldots\ a\dot{a}$ ће се користити као бројач и његова вредност је $||\hat{w}, a\dot{a}||$. Произвољна (не нужно почетна) циклична радна реч је кодирана слично дефиницији (1) сем што се бројач налази 'унутар' w. Конкретно, ако је $w = x_0 x_1 \ldots x_l$ онда

$$\hat{w} = \underset{j}{\bar{x}_0}\underset{j}{\vec{x}_0}\underset{j}{x_1}\underset{j}{\dot{x}_1}\ \ \ldots\ \underset{j}{x_i}\underset{j}{\dot{x}_i}\ \underset{jj}{a\dot{a}}\ \ \ldots\ \underset{jj}{a\dot{a}}\ \underset{j}{x_{i+1}}\underset{j}{\dot{x}_{i+1}}\ \ \ldots\ \underset{jj}{x_l}\underset{jj}{\dot{x}_l}$$

за неко $i \in \{0 \ldots l\}$ и $j \in \mathbb{N}$. Даље, вредност бројача је степен двојке, стриктно већи од l

$$||\hat{w}, a\dot{a}|| = 2^{\lceil \log l' \rceil} \tag{3.1}$$

где $l' \in \mathbb{N}, l' > l$.

У овој симулацији, бројач неће превише расти. Као што ће бити показано, вредност бројача никада неће бити већа од двоструке дужине најдуже речи која се појављује у израчунавању цикличног таг система.

3.4 Симулација

Показаћемо да постоји 2-таг систем који симулира рад произвољног корака израчунавања цикличног таг система дефинисаног у изразу (1). Расчлањујемо израз (1) у три концептуална корака:

1. ако је $x_0 = 1$ онда симулирамо правило $x_0 \to \alpha_m$ тако што додајемо $\alpha_m = x_{l+1} \dots x_{l+c}$,
2. подешавамо x_1 да буде нови учитани симбол и бришемо x_0,
3. увећавамо маркер m тако да следећи продужетак буде $\alpha_{(m+1) \bmod p}$.

Почињемо тако што симулирамо (2). Неформално лема (1) тврди да постоји 2-таг систем који ефикасно мрда надцрту напред за један пар симбола. Главна потешкоћа је разликовање $\underset{1\ 1}{x_1\dot{x}_1}$ од других симбола без надцрте $\underset{1\ 1}{x_2\dot{x}_2}, \dots, \underset{1\ 1}{x_l\dot{x}_l}$. Ова лема важи у случају да је вредност бројача следећи степен двојке већи од дужине шифроване речи.

Лема 1 Нека је дата реч облика

$$\hat{w} = \underset{1\ 1\ 1\ 1}{\bar{x}_0\dot{\bar{x}}_0 x_1\dot{x}_1} \ \dots \ \underset{1\ 1\ 1\ 1}{x_i\dot{x}_i\, a\dot{a}} \ \dots \ \underset{1\ 1}{a\dot{a}}\ \underset{1\quad 1}{x_{i+1}\dot{x}_{i+1}} \ \dots \ \underset{1\ 1}{x_l\dot{x}_l}$$

постоји 2-таг систем T који израчунава

$$\hat{\hat{w}} = \underset{6\ 6}{\bar{x}_1\dot{\bar{x}}_1} \dots \ \underset{6\ 6\ 6\ 6}{x_i\dot{x}_i\, a\dot{a}} \ \dots \ \underset{6\ 6}{a\dot{a}}\ \underset{6\quad 6}{x_{i+1}\dot{x}_{i+1}} \ \dots \ \underset{6\ 6}{x_l\dot{x}_l}$$

у времену $O(l \log l)$. Овде $||\hat{w}, a\dot{a}|| = ||\hat{\hat{w}}, a\dot{a}|| = 2^{\lceil \log(l+1) \rceil}$ и $i \in \{0\}$.

Идеја доказа - Постоји 5 фаза овог алгоритма. Нека је $\hat{w}_0 = \hat{w}$ и нека $\hat{w}_k$ означава излаз из k-те итерације 5 фаза. У фазама од 1 до 3 k-те итерације рачунамо $\lceil \frac{||\hat{w}_{k-1}, x\dot{x}||}{2} \rceil$, тако што означавамо сваки други пар $x\dot{x}$ симбола (означавамо парне парове). Онда у фазама 4 и 5 половимо бројач $||\hat{w}_{k-1}, a\dot{a}||$ тако што означавамо сваки други пар $a\dot{a}$ симбола (опет означавамо парне парове). Враћамо се у фазу 1 и итерирамо све док $||\hat{w}_k, a\dot{a}|| = 1$. Тада бројач има непарну вредност по први пут. Број потпуно извршених итерација и коначна вредност k је $k = \log ||\hat{w}, a\dot{a}||$. У овом тренутку $x_1\dot{x}_1$ је једини пар неозначених $x\dot{x}$ симбола у $\hat{w}$, тако да је $x_1\dot{x}_1$ јединствен од осталих парова симбола у речи $\hat{w}$. Бришемо $\bar{x}_0\dot{\bar{x}}_0$. Јединственост $x_1\dot{x}_1$ дозвољава успешно извршавање правила $\{x_1 \to \bar{x}_0\dot{\bar{x}}_0\}$.

Детаљи доказа: Нека x припада $\{0,1\}$. Овде конкретизујемо правила 2-таг

система и показујемо њихову прву итерацију. Зарад илустрације нека је $i = l$. У фази 1 почињемо са речју облика

$$\hat{w} = \underset{1}{\bar{x}_0}\underset{1}{\vec{x}_0}\underset{1}{x_1}\underset{1}{\dot{x}_1}\underset{1}{x_2}\underset{1}{\dot{x}_2} \ \ldots \ \underset{1}{x_l}\underset{1}{\dot{x}_l}\underset{11}{a\dot{a}}\underset{11}{a\dot{a}} \ \ldots \ \underset{11}{a\dot{a}}$$

Фаза 1 се састоји од правила:

$$\left\{ \underset{1}{\bar{x}} \to \underset{22}{\bar{x}\vec{x}}, \quad \underset{1}{x} \to \underset{222}{x\dot{x}\ddot{x}}, \quad \underset{\not 1}{\dot{x}} \to \underset{\not 2\not 2}{\dot{x}\ddot{x}}, \quad \underset{1}{a} \to \underset{22}{a\dot{a}}, \quad \underset{\not 1}{\dot{a}} \to \underset{\not 2\not 2}{\dot{a}\ddot{a}} \right\}$$

уз још пар правила која ће бити наведена накнадно. Након прве фазе добијамо

$$\underset{2}{\bar{x}_0}\underset{2}{\vec{x}_0}\underset{2}{x_1}\underset{2}{\dot{x}_1}\underset{2}{\ddot{x}_1}\underset{2}{x_2}\underset{2}{\dot{x}_2}\underset{2}{\ddot{x}_2} \ \ldots \ \underset{2}{x_l}\underset{2}{\dot{x}_l}\underset{2}{\ddot{x}_l}\underset{22}{a\dot{a}}\underset{22}{a\dot{a}} \ \ldots \ \underset{22}{a\dot{a}} \tag{3.2}$$

Правила фазе два су

$$\left\{ \begin{array}{l} \underset{2}{\bar{x}} \to \underset{33}{\bar{x}\vec{x}}, \quad \underset{2}{x} \to \underset{33}{x\dot{x}}, \quad \underset{2}{\dot{x}} \to \underset{\not 3\not 3}{\dot{x}\ddot{x}}, \quad \underset{2}{\ddot{x}} \to \epsilon, \quad \underset{\not 2}{\dot{x}} \to \underset{\not 3\not 3}{\dot{x}\ddot{x}}, \quad \underset{\not 2}{\ddot{x}} \to \underset{\not 3\not 3}{\dot{x}\ddot{x}}, \\ \underset{2}{a} \to \underset{33}{a\dot{a}}, \quad \underset{2}{\dot{a}} \to \underset{33}{a\dot{a}}, \quad \underset{\not 2}{\dot{a}} \to \underset{\not 3\not 3}{\dot{a}\ddot{a}}, \quad \underset{\not 2}{\ddot{a}} \to \underset{\not 3\not 3}{\dot{a}\ddot{a}} \end{array} \right\}$$

настављајући од (3.2), након једног пролажења видимо да је сваки други (паран) пар $x\dot{x}$ означен

$$\underset{3}{\bar{x}_0}\underset{3}{\vec{x}_0}\underset{3}{x_1}\underset{3}{\dot{x}_1}\underset{\not 3}{x_2}\underset{\not 3}{\dot{x}_2}\underset{3}{x_3}\underset{3}{\dot{x}_3}\underset{\not 3}{x_4}\underset{\not 3}{\dot{x}_4} \ \ldots \ \underset{3}{x_{l-1}}\underset{3}{\dot{x}_{l-1}}\underset{\not 3}{x_l}\underset{\not 3}{\dot{x}_l}\underset{33}{a\dot{a}}\underset{33}{a\dot{a}} \ \ldots \ \underset{33}{a\dot{a}}$$

где (зарад приказа) претпостављамо да је l парно. Правила фазе 3 су:

$$\left\{ \begin{array}{l} \underset{3}{\bar{x}} \to \underset{44}{\bar{x}\vec{x}}, \quad \underset{3}{x} \to \underset{44}{x\dot{x}}, \quad \underset{\not 3}{\dot{x}} \to \underset{\not 4\not 4}{\dot{x}\ddot{x}}, \quad \underset{3}{a} \to \underset{44}{a\dot{a}}, \quad \underset{\not 3}{\dot{a}} \to \underset{\not 4\not 4}{\dot{a}\ddot{a}}, \\ \underset{3}{\dot{\bar{x}}} \to \underset{44}{\#\bar{x}\vec{x}}, \quad \underset{3}{\dot{x}} \to \underset{44}{x\dot{x}}, \quad \underset{\not 3}{\dot{x}} \to \underset{\not 4\not 4}{\dot{x}\ddot{x}}, \quad \underset{3}{\dot{a}} \to \underset{44}{a\dot{a}}, \quad \underset{\not 3}{\dot{a}} \to \underset{\not 4\not 4}{\dot{a}\ddot{a}}, \end{array} \right\}$$

Улазимо у фазу 3 читајући симбол са тачком (остао је непаран број неозначених $x\dot{x}$ парова после фазе 1) или симбол без тачке (остао је паран број неозначених $x\dot{x}$ парова после фазе 1). Фаза три почиње провером овога у једном кораку; ако је број паран, онда 2-таг систем чита симболе са тачком, тада додаје симбол $\#$ и враћа парност на непар након први фазе. Након завршетка фазе 3, читамо симбол без тачке:

$$\underset{4}{\bar{x}_0}\underset{4}{\vec{x}_0}\underset{4}{x_1}\underset{4}{\dot{x}_1}\underset{\not 4}{x_2}\underset{\not 4}{\dot{x}_2}\underset{4}{x_3}\underset{4}{\dot{x}_3}\underset{\not 4}{x_4}\underset{\not 4}{\dot{x}_4} \ \ldots \ \underset{4}{x_{l-1}}\underset{4}{\dot{x}_{l-1}}\underset{\not 4}{x_l}\underset{\not 4}{\dot{x}_l}\underset{44}{a\dot{a}}\underset{44}{a\dot{a}} \ \ldots \ \underset{44}{a\dot{a}}$$

У фазама 4 и 5 k-те итерације половимо вредност бројача (рачунамо $||\hat{w}_k, a\dot{a}|| = \frac{||\hat{w}_{k-1}, a\dot{a}||}{2}$), на сличан начин као у фазама 1 до 3. Правила фазе 4 су:

$$\left\{ \underset{4}{\bar{x}} \to \underset{55}{\bar{x}\dot{\bar{x}}}, \quad \underset{4}{x} \to \underset{55}{x\dot{x}}, \quad \underset{\not 4}{\dot{x}} \to \underset{\not 5\not 5}{\dot{x}\ddot{x}}, \quad \underset{4}{a} \to \underset{555}{a\dot{a}\ddot{a}}, \quad \underset{\not 4}{\dot{a}} \to \underset{\not 5\not 5}{\dot{a}\ddot{a}} \right\}$$

што нам након извршавања даје

$$\underset{5}{\bar{x}_0}\underset{5}{\vec{x}_0}\underset{5}{x_1}\underset{5}{\dot{x}_1}\ \underset{\not 5}{x_2}\underset{\not 5}{\dot{x}_2}\ \ldots\ \underset{\not 5}{x_l}\underset{\not 5}{\dot{x}_l}\ \underset{555}{a\dot{a}\ddot{a}}\ \underset{555}{a\dot{a}\ddot{a}}\ \ldots\ \underset{555}{a\dot{a}\ddot{a}}$$

Правила фазе 5 полове вредност бројача:

$$\left\{ \begin{array}{l} \underset{5}{\bar{x}} \to \underset{11}{\bar{x}\dot{\bar{x}}}, \quad \underset{5}{x} \to \underset{11}{x\dot{x}}, \quad \underset{\not 5}{\dot{x}} \to \underset{\not 1\not 1}{\dot{x}\ddot{x}}, \quad \underset{5}{\dot{x}} \to \underset{11}{x\dot{x}}, \quad \underset{5}{\dot{x}} \to \underset{\not 1\not 1}{\dot{x}\ddot{x}}, \\ \underset{5}{a} \to \underset{11}{a\dot{a}}, \quad \underset{5}{\dot{a}} \to \underset{\not 1\not 1}{\dot{a}\ddot{a}}, \quad \underset{5}{\ddot{a}} \to \epsilon, \quad \underset{\not 5}{\dot{a}} \to \underset{\not 1\not 1}{\dot{a}\ddot{a}}, \quad \underset{\not 5}{\ddot{a}} \to \underset{\not 1\not 1}{\dot{a}\ddot{a}} \end{array} \right\}$$

Настављајући наше израчунавање добијамо:

$$\underset{1}{\bar{x}_0}\underset{1}{\vec{x}_0}\underset{1}{x_1}\underset{1}{\dot{x}_1}\ \underset{\not 1}{x_2}\underset{\not 1}{\dot{x}_2}\ \ldots\ \underset{\not 1}{x_l}\underset{\not 1}{\dot{x}_l}\ \underset{11}{a\dot{a}}\ \underset{\not 1\not 1}{\dot{a}\ddot{a}}\ \underset{11}{a\dot{a}}\ \underset{\not 1\not 1}{\dot{a}\ddot{a}}\ \ldots\ \underset{11}{a\dot{a}}\ \underset{\not 1\not 1}{\dot{a}\ddot{a}}$$

и враћамо се у фазу 1. Свака итерација ових 5 фаза полови вредност бројача. Након $\log||\hat{w}, a\dot{a}||$ итерација вредност бројача ће бити 1. Због овога излаз из фазе 4 ће бити непаран по први пут. Парност добија парну вредност за време фазе 5, што детектујемо на почетку фазе 1 правилима:

$$\{\underset{1}{\dot{\bar{x}}} \to \#, \underset{1}{\dot{x}} \to \underset{66}{\bar{x}\dot{\bar{x}}}, \quad \underset{\not 1}{\ddot{x}} \to \underset{66}{x\dot{x}}, \quad \underset{1}{\dot{a}} \to \underset{66}{a\dot{a}}, \quad \underset{\not 1}{\ddot{a}} \to \underset{66}{a\dot{a}}\}$$

Прво правило брише x_0 у једном кораку. Друго пребацује надцрту на следећи пар симбола, а наредна два правила скидају ознаку са преосталих симбола.

$$\underset{6}{\bar{x}_1}\underset{6}{\vec{x}_1}\underset{6}{x_2}\underset{6}{\dot{x}_2}\ \ldots\ \underset{6}{x_l}\underset{6}{\dot{x}_l}\ \underset{66}{a\dot{a}}\ \underset{66}{a\dot{a}}\ \ldots\ \underset{66}{a\dot{a}}$$

Симбол # враћа парност на непар тако да читамо симболе без тачке (ово је везано за касније израчунавања). На примеру је претпостављено да је $i = l$. Ако је реч $\hat{w}$ општијег облика $i \in \{0, \ldots, l\}$, можемо убацити бројач унутар речи и исти доказ ће важити с обзиром да то не утиче на парност и контролу тока алгоритма. □

Наредна лема ће нам обезбедити већину механизама потребних за симулацију додавања продужетка (прву тачку поглавља 3.4). Симулација додавања је

праволинијска, највећи проблем је одржавање једнакости израза (3.1). Стога, након додавања продужетка, ако симулирана реч има дужину бар једнаку дужини бројача, удуплаћемо бројач да би задовољили израз (3.1). Следећа лема је формулисана са претпоставком да је иницијална вредност бројача први степен двојке већи од дужине кодиране речи.

Лема 2 Нека је дата реч облика

$$\hat{w} = \underset{0\,0}{\bar{1}}\underset{0}{\dot{\bar{1}}}\underset{0}{x_1}\underset{0}{\dot{x}_1} \quad \ldots \quad \underset{0\,0}{x_i}\underset{0}{\dot{x}_i}\,\underset{0\,0}{a\dot{a}} \quad \ldots \quad \underset{0\,0}{a\dot{a}}\,\underset{0}{x_{i+1}}\underset{0}{\dot{x}_{i+1}} \quad \ldots \quad \underset{0\,0}{x_l}\underset{0}{\dot{x}_l}$$

при чему $||\hat{w}, a\dot{a}|| = 2^{\lceil \log(l+1) \rceil}$ и $i \in \{0, \ldots, l\}$, онда постоји 2-таг систем T који израчунава

$$\hat{w} = \underset{8\,8}{\bar{1}}\underset{8}{\dot{\bar{1}}}\underset{8}{x_1}\underset{8}{\dot{x}_1} \quad \ldots \quad \underset{8\,8}{x_i}\underset{8}{\dot{x}_i}\,\underset{8\,8}{a\dot{a}} \quad \ldots \quad \underset{8\,8}{a\dot{a}}\,\underset{8}{x_{i+1}}\underset{8}{\dot{x}_{i+1}} \quad \ldots \quad \underset{8\,8}{x_l}\underset{8}{\dot{x}_l}\,\underset{8}{x_{l+1}}\underset{8}{\dot{x}_{l+1}} \quad \ldots \quad \underset{8}{x_{l+c}}\underset{8}{\dot{x}_{l+c}}$$

где $c \leq 2^{\lceil \log(l+1) \rceil}$, $||\hat{\hat{w}}, a\dot{a}|| = 2^{\lceil \log(l+c+1) \rceil}$. T извршава ово израчунавање у времену $O(l \log l)$.

Доказ. Примењујући правило

$$\left\{ \underset{0}{\bar{1}} \rightarrow \underset{0}{x_{l+1}}\underset{0}{\dot{x}_{l+1}} \ldots \underset{0}{x_{l+c}}\underset{0}{\dot{x}_{l+c}}\underset{1}{\bar{1}}\underset{1}{\dot{\bar{1}}} \right\}$$

на $\hat{w}$ добијамо реч, означићемо је $\hat{w}_0$. Она је слична $\hat{\hat{w}}$ осим што $||\hat{w}_0, a\dot{a}|| = 2^{\lceil \log(l+1) \rceil}$, то јест бројач још увек није ажуриран на вредност $||\hat{\hat{w}}, a\dot{a}|| = 2^{\lceil \log(l+c+1) \rceil}$. Остатак доказа ће се бавити ажурирањем бројача. Нека $\hat{w}_k$ означава k-ту итерацију од 4 фазе. Правила за фазе од 1 до 4 су сличне онима из претходне леме па их нећемо поново наводити. Почнимо израчунавање над $\hat{w}_0$. За време прве две фазе од k-те итерације, рачунамо $||\hat{w}_k, a\dot{a}|| = \frac{||\hat{w}_{k-1}, a\dot{a}||}{2}$, означавајући сваки други пар $a\dot{a}$ симбола (означавамо све парне парове). У фазама 3 и 4 израчунавамо

$$||\hat{w}_k, x\dot{x}|| = \left\lfloor \frac{||\hat{w}_{k-1}, x\dot{x}||}{2} \right\rfloor \tag{3.3}$$

тако што означавамо сваки други пар $x\dot{x}$ симбола (овде означавамо непарне парове). Затим се враћамо на фазу 1 и итерирамо поступак све док $||\hat{w}_k, a\dot{a}|| = 1$. Бројач сада има непарну вредност по први пут и то детектујемо у фази 3. Број потпуно извршених итерација ће бити k, при чему $k = \log||\hat{w}_k, a\dot{a}|| = \lceil \log(l+1) \rceil$. Додатна фаза враћа парност (тако што уводи додатни симбол #

и брише га након једне рунде) да бисмо читали симболе без тачке. Затим $\hat{w}_k$ је облика

$$\hat{w} = \underset{5}{\bar{1}}\underset{5}{\dot{1}}\underset{5}{x}\underset{5}{\dot{x}}\ \ldots\ \underset{5}{x}\underset{5}{\dot{x}}\underset{5}{a}\underset{5}{\dot{a}}\ \ldots\ \underset{5}{q}\underset{5}{\dot{q}}\ \underset{5}{x_{i+1}}\underset{5}{\dot{x}_{i+1}}\ \ldots\ \underset{5}{x_l}\underset{5}{\dot{x}_l}\ \underset{5}{x_{l+1}}\underset{5}{\dot{x}_{l+1}}\ \ldots\ \underset{5}{x_{l+c}}\underset{5}{\dot{x}_{l+c}}$$

У овом тренутку број $||\hat{w}_k, x\dot{x}||$ неозначених $x\dot{x}$ парова задовољава $0 \leq ||\hat{w}_k, x\dot{x}|| \leq 1$. Да видимо ово, приметимо да $||\hat{w}_0, a\dot{a}|| = 2^{\lceil \log(l+1) \rceil}$ и $c \leq 2^{\lceil \log(l+1) \rceil}$, решавајући израз (3.3) за $k = \lceil \log(l+1) \rceil$ добијамо 0 акко

$$0 \leq ||\hat{w}_0, x\dot{x}|| = l + c < 2^{\lceil \log(l+1) \rceil}$$

и 1 акко

$$2^{\lceil \log(l+1) \rceil} \leq ||\hat{w}_0, x\dot{x}|| = l + c < 2^{\lceil \log(l+c+1) \rceil}.$$

Не постоје друге могуће вредности за $||\hat{w}_0, x\dot{x}||$ па треба да проверимо да ли је $||\hat{w}_0, x\dot{x}||$ једнако 0 или 1. Да бисмо се припремили, у две наредне фазе примењујемо правила

$$\left\{ \underset{5}{\bar{1}} \to \underset{6}{\bar{1}}\underset{6}{\dot{1}}, \underset{5}{x} \to \underset{6}{x}\underset{6}{\dot{x}}\underset{6}{\ddot{x}}, \underset{5}{\not{x}} \to \underset{6}{\not{x}}\underset{6}{\dot{\not{x}}}, \underset{5}{a} \to \underset{6}{a}\underset{6}{\dot{a}}, \underset{5}{q} \to \underset{6}{q}\underset{6}{\dot{q}} \right\}$$

и

$$\left\{ \begin{array}{c} \underset{6}{\bar{1}} \to \underset{7}{\bar{1}}\underset{7}{\dot{1}}, \underset{6}{x} \to \underset{7}{x}\underset{7}{\dot{x}}, \underset{6}{\ddot{x}} \to \epsilon, \underset{6}{\not{x}} \to \underset{7}{x}\underset{7}{\dot{x}}, \underset{6}{\dot{\not{x}}} \to \underset{7}{x}\underset{7}{\dot{x}}, \\ \underset{6}{a} \to \underset{7}{a}\underset{7}{\dot{a}}, \underset{6}{\dot{a}} \to \underset{7}{a}\underset{7}{\dot{a}}, \underset{6}{q} \to \underset{7}{a}\underset{7}{\dot{a}}, \underset{6}{\dot{q}} \to \underset{7}{a}\underset{7}{\dot{a}} \end{array} \right\}$$

Ова правила померају читајућу главу на тачкасте симболе акко $||\hat{w}_k, x\dot{x}|| = 1$. Додатно, ова правила одозначавају све означене симболе, јер то више није потребно. Постоје два случаја:

Први - ако $||\hat{w}_k, x\dot{x}|| = 0$ онда нема потребе да мењамо вредност бројача да бисмо задовољили израз (3.1). У овом случају детектујемо $||\hat{w}_k, x\dot{x}|| = 0$ читајући надвучени симбол без тачке $\underset{7}{\bar{1}}$. Да завршимо израчунавање примењујемо правила

$$\left\{ \underset{7}{\bar{1}} \to \underset{8}{\bar{1}}\underset{8}{\dot{1}}, \underset{7}{x} \to \underset{8}{x}\underset{8}{\dot{x}}, \underset{7}{a} \to \underset{8}{a}\underset{8}{\dot{a}} \right\}$$

Други - ако $||\hat{w}_k, x\dot{x}|| = 1$ дуплирамо бројач да бисмо задовољили израз (3.1). У овом случају детектујемо $||\hat{w}_k, x\dot{x}|| = 1$ тако што читамо надвучени симбол са тачком $\underset{7}{\dot{1}}$. Затим враћамо парност на непар и дуплирамо вредност бројача

примењујући следећа правила:

$$\left\{ \underset{7}{\bar{1}} \to \underset{88}{\#\bar{1}\dot{\bar{1}}}, \underset{7}{x} \to \underset{88}{x\dot{x}}, \underset{7}{a} \to \underset{8888}{a\dot{a}a\dot{a}} \right\}$$

□

3.5 Временска сложеност

Теорема 2 За задати циклични таг систем C који врши израчунавање у времену $t(n)$, унос дужине n (где n представља бар дужину C-овог најдужег продужетка), постоји 2-таг систем T_C који симулира израчунавање C у времену $O(t^2(n)\log t(n))$.

Доказ. Претпостављамо да је C-ов унос дужине n дугачак бар колико његов најдужи продужетак. Као део шифровања уноса 2-таг система у дефиницији (1), мање уносе ћемо допунити до ове дужине. Подсетимо се декомпозиције корака израчунавања цикличног таг система у три подкорака са почетка секције (3.4) .

Лема (2) обезбеђује алгоритам за први корак у случају да је $\bar{x}_0 = \bar{1}$. У случају да важи $\bar{x}_0 = \bar{0}$ прескачемо први корак. Разлучивање између ова два случаја је лако - подешавањем парности на пар акко $\bar{x}_0 = \bar{1}$.

Лема (1) обезбеђује главни алат за други подкорак. Након примене леме (1) надвучени пар $x\dot{x}$ је или са леве ($i > 0$ из тврђења леме) или прескочи преко бројача и налази се са десне стране ($i = 0$). Лема (2) претпоставља да се глава налази са леве стране. То треба обезбедити како бисмо је применили. Стога остатак другог дела другог подкорака је следећи - у једном пролажењу прочитамо целу реч $\hat{w}$ и репликујемо је, сем што подиндексе увећамо и дужину надвучене подречи повећамо са 2 на 3 правилом $\bar{x} \to \bar{x}\dot{\bar{x}}\ddot{\bar{x}}$.

Ако у следећем пролажењу читамо тачкаст симбол бројача ($\dot{a}$), онда је глава с леве стране бројача; у наредном пролажењу подешавамо парност на непар и тиме завршавамо посао.

С друге стране, ако читамо симболе бројача без тачке (a), онда подешавамо парност на непар у једном пролажењу, а потом читамо бројач још једном како бисмо га поставили на десну страну у односу на главу.

Леме (1) и (2) тврде да вредност бројача треба да буде први степен двојке већи од l дужине речи C-а. Ови докази важе и за општији случај, где је вредност бројача било који степен двојке већи од l. То је разлог због којег

доказе можемо применити чак и ако се дужина речи C-а драстично смањи. У том случају временска сложеност неће зависити више од l, већ од дужине бројача. Обзиром да је вредност бројача $O(t(n))$ ово не утиче на анализу комплексности.

За трећи подкорак уводимо нове ознаке за симболе 2-таг система. До овог тренутка их је било q, али сада повећавамо тај број на pq, где p представља број продужетака C-а. Направићемо нови скуп симбола означавајући сваки симбол $y \in \{\bar{x}, \dot{\bar{x}}, x, \dot{x}, \ddot{x}, \not{x}, \dot{\not{x}}, a, \dot{a}, \ddot{a}, \not{a}, \dot{\not{a}}, \#\}$ 2-таг система целим бројем m за свако $0 \leq m < p$. Користећи ово, наше кодирање произвољног цикличног таг система ће добити облик

$$\underset{j\ j}{\bar{x}_0}\underset{m\ m}{\dot{\bar{x}}_0}\ \underset{j\ \ j}{x_1}\underset{m\ \ m}{\dot{x}_1}\ \cdots\ \underset{j\ \ j}{x_i}\underset{m\ \ m}{\dot{x}_i}\ \underset{j\ \ j}{a}\underset{m\ \ m}{\dot{a}}\ \underset{j\ \ j}{a}\underset{m\ \ m}{\dot{a}}\ \cdots\ \underset{j\ \ j}{a}\underset{m\ \ m}{\dot{a}}\ \underset{j\ \ \ j}{x_{i+1}}\underset{m\ \ \ m}{\dot{x}_{i+1}}\ \cdots\ \underset{j\ \ j}{x_l}\underset{m\ \ m}{\dot{x}_l}$$

Подкораци 1 и 2 се симулирају игноришући вредност m. Затим j добија вредност која сигнализира да су извршена прва два подкорака. Трећи подкорак (инкрементовање програмског маркера) се симулира правилима облика:

$$\left\{ \underset{j}{\bar{x}} \rightarrow \underset{j\ \ j}{\bar{x}}\underset{m'\ m'}{\dot{\bar{x}}}\ , \underset{j}{x} \rightarrow \underset{j\ \ j}{x}\underset{m'\ m'}{\dot{x}}\ , \underset{j}{a} \rightarrow \underset{j\ \ j}{a}\underset{m'\ m'}{\dot{a}} \right\}$$

где $m' = (m+1) \mathrm{mod}_p$. Примена ових правила за дато k захтева $O(l)$ временских корака. У анализи временске сложености израчунавања T_C-а, произвољан временски корак C-а процењујемо l-ом, $l = O(t(n))$. Стога примењујући леме (2),(1) и овај доказ, закључујемо да T_C симулира један корак C-овог израчунавања у времену $O(t(n)\ \log t(n))$. □

Када бисмо директно применили теореме (1) и (2), добили бисмо да је временска сложеност симулације Тјурингове машине сложености $O(t)$, 2-таг системом једнака $O(t^6(\log t)^3)$. Пажљивијом анализом, могли бисмо да добијемо ограничење наведено у наредној теореми.

Теорема За дату детерминистичку Тјурингову машину T са једном траком, која врши израчунавње у времену t, постоји 2-таг систем T_M која симулира израчунавање T у времену $O(t^4(\log t)^2)$.

Последица Најмање Тјурингове машине Минског и осталих су полиномијални $O(t^8(\log t)^4)$ симулатори Тјурингових машина.

Додатак

Програм који исписује израчунавања Тјурингове машине која симулира 2-таг систем

Овде је дат кôд програма који је коришћен за анализу 2-таг система из другог поглавља. Исписан је у програмском језику *Java*.

```
1 // Zakomentarisan kod je koriscen kao pomagalo u svrhe
    analize rezultata
2 public class masterTezaAutomat {
3
4  /**
5   * Funkcija koja vraca novu traku u kojoj na poziciju
    pokazanoj glavom
6   * Tjuringove masine upisuje simbol c.
7   * @param traka Traka Tjuringove masine
8   * @param glava Broj trenutne celije
9   * @param c     Novi simbol koji ce biti upisan na traku
10  */
11 public static String zameni(String traka, int glava, char
    c) {
12  return traka.substring(0,glava) + c + traka.substring(
    glava+1);
13 }
14
15 /**
16  * Pojenostavljeni ispis trake koji za svaki niz od tri
    ili vise istih
```

```
17   * simbola ispisuje dva, sa tri tacke izmedju njih.
18   * npr AAAAAABBBBBBCCCCCBA -> A ... AB ... BC ... CBA
19   * @param traka Traka Tjuringove masine
20   */
21  public static void ispisTrake(String traka) {
22   int i=0;
23   try {
24    for(i = 0; i < traka.length() ; i++) {
25     // ispisujemo trenutni simbol
26     System.out.print(traka.charAt(i));
27     // u trenutku kada ima tri simbola ista zaredom, gutamo
      simbole sve dok ne naidjemo na razliciti
28     if(traka.charAt(i)==traka.charAt(i+1))
29      if(traka.charAt(i+1)==traka.charAt(i+2)) {
30       System.out.print(" ... ");
31       while(traka.charAt(i+1)==traka.charAt(i)) {
32        i++;
33       }
34      // ispisujemo poslednji simbol od istih i onda
     prelazimo na naredni u sledecoj iteraciji
35      System.out.print(traka.charAt(i));
36      }
37
38    }
39   }catch(IndexOutOfBoundsException e) {
40    System.out.print(traka.charAt(i)+ "\n");
41    //System.out.println("verujem da smo stigli do ovde");
42   }
43  }
44
45  /**
46   * Funkcija kao i prethodna, samo sto ispisuje traku
     crvenim slovima.
47   */
48  public static void ispisTrakeCrveno(String traka) {
49   int i=0;
```

```
50  try {
51   for(i = 0; i < traka.length() ; i++) {
52    // ispisujemo trenutni simbol
53    System.err.print(traka.charAt(i));
54    // u trenutku kada ima tri simbola ista zaredom, gutamo
     simbole sve dok ne naidjemo na razliciti
55    if(traka.charAt(i)==traka.charAt(i+1))
56     if(traka.charAt(i+1)==traka.charAt(i+2)) {
57      System.err.print(" ... ");
58      while(traka.charAt(i+1)==traka.charAt(i)) {
59       i++;
60      }
61     // ispisujemo poslednji simbol od istih i onda
    prelazimo na naredni u sledecoj iteraciji
62     System.err.print(traka.charAt(i));
63     }
64
65   }
66  }catch(IndexOutOfBoundsException e) {
67   System.err.print(traka.charAt(i)+ "\n");
68   //System.out.println("verujem da smo stigli do ovde");
69  }
70 }
71
72 public static void main(String[] args) {
73  // simboli tag sistema su ai = { a, b, c }
74  // a-> bc
75  // b-> a
76  // c-> aaa
77  // pocetna rec je "aaa"
78
79  // ako je debug = 1 ispisuju se neke informacije poput
    izgleda produkcione regije
80  int debug = 0;
81  // ako je ispisPriPromeniStanja = 1 ipsiujemo infomracije
     o traci svaki put kada se promeni stanje
```

```
82  int ispisPriPromeniStanja = 1;
83  String pocetak = "OXIAAO";
84  String brisanje = "";
85  String produkcija = "";    // (Pi) = II O^Ni,ni O I... I O
     ^Ni,1 O
86              // ima ni+1 I-ova
87  String razdvajanje = "";  // I^R
88  //String obrisana = "";
89  String sinteza = "";  //Y^ Nalfa A ^ Nbeta ... A Y^Nomega
90  String prazanProstor = "";
91
92
93
94  int N[] = {0, 0, 0}, P = 0, R = 0, S = 0, ni[] = {2, 1,
     3}, Q = 0;
95  String Pi[] = {"II","II","II"};
96
97  /**********************************
98   * ODREDJIVANJE KONSTANTI SPQR   *
99   **********************************/
100 // radi se o 2-tag sistemu
101 P = 2;
102
103 for (int niTmp : ni) {
104  S += niTmp+1;
105 }
106 // Konstanta regije razdvajanja
107 R = P * S;
108 // Konstanta regije brisanja
109 Q = (P-1)*R-S;
110 for(int i=0; i < ni.length ; i++) {
111  int tmpSum = 0;
112  for (int j=0 ; j < i; j++) {
113   tmpSum+=ni[j]+1;
114  }
115  N[i] = tmpSum + R;
```

```
116     if (debug == 1)
117      System.out.println("N"+i+" = " + N[i]);
118   }
119   if (debug == 1) {
120    System.out.println("Q = " + Q);
121   }
122   if (debug == 1) {
123    System.out.println("R = " + R);
124   }
125   if (debug == 1) {
126    System.out.println("S = " + S);
127   }
128
129   /**********************************
130    * KONSTRUKCIJA PRODUKCIONE REGIJE *
131    **********************************/
132
133   Pi[0]="II";
134   //dodajemo c
135   for(int i=0;i<23+1;i++)
136    Pi[0]+="O";
137   Pi[0]+="I";
138   // dodajemo b
139   for(int i=0;i<21+1;i++)
140    Pi[0]+="O";
141
142   Pi[1] = "II";
143   // dodajemo a
144   for(int i=0;i<18+1;i++) {
145    Pi[1]+="O";
146   }
147   Pi[2] = "II";
148   // dodajemo a
149   for(int i=0;i<18+1;i++) {
150    Pi[2]+="O";
151   }
```

```
152  Pi[2]+="I";
153  // dodajemo a
154  for(int i=0;i<18+1;i++) {
155   Pi[2]+="O";
156  }
157  Pi[2]+="I";
158  // dodajemo a
159  for(int i=0;i<18+1;i++) {
160   Pi[2]+="O";
161  }
162
163  produkcija = Pi[2] + Pi[1] + Pi[0];
164  if(debug == 1)
165   System.out.println(produkcija);
166  /*************************************
167   * KONSTRUKCIJA REGIJE TRENUTNE NISKE *
168   *************************************/
169  for(int i = 0; i < 3; i++) {
170   for(int j = 0; j < N[0]; j++) {
171    sinteza += "Y";
172   }
173   sinteza += "A";
174  }
175  sinteza = sinteza.substring(0,sinteza.length()-1);
176  if (debug == 1)
177   System.out.println("Regija sinteze je " + sinteza);
178
179  /*************************************
180   * KONSTRUKCIJA REGIJE BRISANJA      *
181   *************************************/
182  for (int i = 0; i < Q; i++)
183   brisanje += "I";
184
185  /*************************************
186   * KONSTRUKCIJA REGIJE RAZDVAJANJA     *
187   *************************************/
```

```
188  for (int i = 0; i < R; i++)
189   razdvajanje += "I";
190
191  /************************************
192   *KONSTRUKCIJA REGIJE PRAZNOG PROSTORA*
193   ************************************/
194  // 10000/6 - 600 je empirijski dobijena aproksimacija
      potrebnog prostora za ovaj konkretan slucaj tag sistema
195  for (int i = 0; i < (10000/6) -600; i++)
196   prazanProstor += "O";
197
198  /************************************
199   *       KONSTRUKCIJA TRAKE         *
200   ************************************/
201  if(debug == 1) {
202   System.out.println("Regije trake redom:");
203   System.out.println(pocetak);
204   System.out.println(brisanje);
205   System.out.println(produkcija);
206   System.out.println(razdvajanje);
207
208   System.out.println(sinteza);
209   System.out.println(prazanProstor);
210
211  }
212
213  String traka = "";
214  traka += pocetak;
215  traka += brisanje;
216  traka += produkcija;
217  traka += razdvajanje;
218  traka += sinteza;
219  traka += prazanProstor;
220  ispisTrakeCrveno(traka);
221  //14 20
222  /************************
```

```
223     * KONSTRUKCIJA AUTOMATA *
224     ************************/
225
226   int glava = pocetak.length() + brisanje.length()+
       produkcija.length()
227   +razdvajanje.length();
228   String stanje = "q1";
229   //System.out.println(zameni("papaja",2,'c'));
230   int i=0;
231   if(ispisPriPromeniStanja == 1) {
232    System.out.println("Promena stanja u " + stanje + ", a
       glava pokazuje na celiju " + glava+". Iteracija je broj
       " + i);
233    System.out.println(traka);
234   }
235   System.out.println();
236   System.out.println(traka);
237   System.out.println();
238
239   // kraj       na 2,221,856 iteraciji izracunavanja
240   //while(i<50000) {
241   while(true) {
242    i++;
243    switch(traka.charAt(glava)) {
244    case 'O':
245     switch(stanje) {
246     case "q1":
247      traka = zameni(traka, glava, 'X');
248      glava--;
249      break;
250     case "q2":
251      traka = zameni(traka, glava, 'X');
252      glava++;
253
254      stanje = "q3";
255      if(ispisPriPromeniStanja == 1) {
```

```
256       System.out.println("Promena stanja u " + stanje + ",
      a glava pokazuje na celiju " + glava+". Iteracija je
      broj " + i);
257       System.out.println(traka);
258      }
259      break;
260     case "q3":
261      traka = zameni(traka, glava, 'A');
262      glava--;
263      stanje = "q4";
264      if(ispisPriPromeniStanja == 1) {
265       System.out.println("Promena stanja u " + stanje + ",
      a glava pokazuje na celiju " + glava+". Iteracija je
      broj " + i);
266       System.out.println(traka);
267      }
268      break;
269     case "q4":
270      traka = zameni(traka, glava, 'X');
271      glava++;
272      stanje = "q5";
273      if(ispisPriPromeniStanja == 1) {
274       System.out.println("Promena stanja u " + stanje + ",
      a glava pokazuje na celiju " + glava+". Iteracija je
      broj " + i);
275       System.out.println(traka);
276      }break;
277     case "q5":
278      traka = zameni(traka, glava, 'Y');
279      glava--;
280      stanje = "q4";
281      if(ispisPriPromeniStanja == 1) {
282       System.out.println("Promena stanja u " + stanje + ",
      a glava pokazuje na celiju " + glava+". Iteracija je
      broj " + i);
283       System.out.println(traka);
```

```
284       }
285       break;
286      case "q6":
287       System.out.println(i);
288       System.out.println(traka);
289       System.out.println("HALT! Kraj kakav treba biti!");
290       return;
291      default:
292       System.err.println("Neispravno stanje kada je glava na
      karakteru O");
293       return;
294      }
295      break;
296     case 'I':
297      switch(stanje) {
298      case "q1":
299       traka = zameni(traka, glava, 'B');
300       glava++;
301       break;
302      case "q2":
303       traka = zameni(traka, glava, 'B');
304       glava++;
305       stanje = "q1";
306       if(ispisPriPromeniStanja == 1) {
307        System.out.println("Promena stanja u " + stanje + ",
      a glava pokazuje na celiju " + glava+". Iteracija je
      broj " + i);
308        System.out.println(traka);
309       }
310       break;
311      case "q3":
312       traka = zameni(traka, glava, 'B');
313       glava++;
314       break;
315      case "q4":
316       traka = zameni(traka, glava, 'I');
```

```
317       glava--;
318       stanje = "q2";
319       if(ispisPriPromeniStanja == 1) {
320        System.out.println("Promena stanja u " + stanje + ",
      a glava pokazuje na celiju " + glava+". Iteracija je
      broj " + i);
321        System.out.println(traka);
322       }
323       break;
324      case "q5":
325       traka = zameni(traka, glava, 'B');
326       glava++;
327       break;
328      case "q6":
329       traka = zameni(traka, glava, 'I');
330       glava++;
331       break;
332      default:
333       System.err.println("Neispravno stanje kada je glava na
       karakteru I");
334       return;
335      }
336      break;
337     case 'A':
338      switch(stanje) {
339      case "q1":
340       traka = zameni(traka, glava, 'X');
341       glava--;
342       stanje = "q2";
343       if(ispisPriPromeniStanja == 1) {
344        System.out.println("Promena stanja u " + stanje + ",
      a glava pokazuje na celiju " + glava+". Iteracija je
      broj " + i);
345        System.out.println(traka);
346       }
347       break;
```

```
348     case "q2":
349      traka = zameni(traka, glava, 'B');
350      glava++;
351      break;
352     case "q3":
353      traka = zameni(traka, glava, 'A');
354      glava++;
355      break;
356     case "q4":
357      traka = zameni(traka, glava, 'A');
358      glava--;
359      break;
360     case "q5":
361      traka = zameni(traka, glava, 'A');
362      glava++;
363      break;
364     case "q6":
365      traka = zameni(traka, glava, 'O');
366      glava++;
367      stanje = "q1";
368      if(ispisPriPromeniStanja == 1) {
369       System.out.println("Promena stanja u " + stanje + ",
    a glava pokazuje na celiju " + glava+". Iteracija je
    broj " + i);
370       System.out.println(traka);
371      }
372      break;
373     default:
374      System.err.println("Neispravno stanje kada je glava na
     karakteru A");
375      return;
376     }
377     break;
378    case 'B':
379     switch(stanje) {
380     case "q1":
```

```
381       traka = zameni(traka, glava, 'I');
382       glava--;
383       break;
384     case "q2":
385       traka = zameni(traka, glava, 'I');
386       glava--;
387       break;
388     case "q3":
389       System.err.println("Nemoguca konfiguracija, B, q3.");
390       return;
391     case "q4":
392       traka = zameni(traka, glava, 'I');
393       glava--;
394       break;
395     case "q5":
396       System.err.println("Nemoguca konfiguracija, B, q5");
397       break;
398     case "q6":
399       traka = zameni(traka, glava, 'A');
400       glava++;
401       break;
402     default:
403       System.err.println("Neispravno stanje kada je glava na
      karakteru B");
404       return;
405     }
406     break;
407   case 'X':
408     switch(stanje) {
409     case "q1":
410       traka = zameni(traka, glava, 'Y');
411       glava++;
412       break;
413     case "q2":
414       traka = zameni(traka, glava, 'B');
415       glava--;
```

```
416       stanje = "q6";
417       if(ispisPriPromeniStanja == 1) {
418        System.out.println("Promena stanja u " + stanje + ",
      a glava pokazuje na celiju " + glava+". Iteracija je
      broj " + i);
419        System.out.println(traka);
420       }
421       break;
422      case "q3":
423       traka = zameni(traka, glava, 'X');
424       glava++;
425       break;
426      case "q4":
427       traka = zameni(traka, glava, 'X');
428       glava--;
429       break;
430      case "q5":
431       traka = zameni(traka, glava, 'X');
432       glava++;
433       break;
434      case "q6":
435       traka = zameni(traka, glava, 'O');
436       glava++;
437       break;
438      default:
439       System.err.println("Neispravno stanje kada je glava na
      karakteru X");
440       return;
441      }
442      break;
443     case 'Y':
444      switch(stanje) {
445      case "q1":
446       traka = zameni(traka, glava, 'X');
447       glava--;
448       break;
```

```
449     case "q2":
450      traka = zameni(traka, glava, 'X');
451      glava--;
452      break;
453     case "q3":
454      traka = zameni(traka, glava, 'Y');
455      glava++;
456      break;
457     case "q4":
458      traka = zameni(traka, glava, 'Y');
459      glava--;
460      break;
461     case "q5":
462      traka = zameni(traka, glava, 'Y');
463      glava++;
464      break;
465     case "q6":
466      traka = zameni(traka, glava, 'O');
467      glava++;
468      break;
469     default:
470      System.err.println("Neispravno stanje kada je glava na
     karakteru Y");
471      return;
472     }
473     break;
474    default:
475     System.err.println("Nepoznat karakter na traci");
476     return;
477    }
478    //ispisTrake(traka);
479    //if(i>2230000)
480     System.out.println(traka);
481    //System.out.println("");
482   }
```

```
483 //System.err.println("Izasli smo iz while petlje silom
      prilika");
484 }
```

Захвалнице

Захвалио бих се превасходно ментору, Небојши Икодиновићу, на сарадњи и помоћи у писању мастер рада, а потом породици и пријатељима, који су уз мене у добру и злу, кад зашкрипи, а и кад сунце сија. Без јаке подршке и великих леђа која чине, не бих ништа постигао!
Специјална захвалност другарици Ани Митровић што је упркос многобројним животним обавезама издвојила гомилу времена да помогне око ревизије и савета за писање овог рада, хвала ти.:)

Литература

[1] M. Cook. Complex Systems. Universality in elementary cellular automata, 2004. 40

[2] T. Neary and D. Woods. P-completeness of cellular automaton Rule 110. International Colloquium on Automata Languages and Programming (ICALP), July 2006. 39, 41

[3] Н. Икодиновић. Теорија алгоритама. 1, 5, 8, 9, 10, 17, 20, 23