Математичка гимназија

# МАТУРСКИ РАД
## - из рачунарства -

## Теорија језика и аутомата

Ученик:
Алекса Сотиров IVд

Ментор:
Милош Арсић

Београд, јун 2021.

# Садржај

# 1 Увод

Овај рад је замишљен као базични преглед две наизглед различите, но заправо уско повезане области - теорије формалних језика и теорије аутомата.

Теорија формалних језика израсла је из лингвистике. Ова област проучава најпре синтаксне особине језика, тј. оне особине које се тичу начина на који се структурни елементи тог језика међусобно односе. Сем анализе природних језика (којима се лингвистика бави), теорија формалних језика је значајна и у рачунарству, јер се налази у основи свих програмских језика које користимо.

Теорија аутомата, са друге стране, има своје корене у математици. Она проучава аутомате - системе стања и правила прелаза из једног стања у друго (ово се може репрезентовати као тип усмереног графа). Из овог описа се може наслутити да и ова област има широке примене у рачунарству, од парсера до вештачке интелигенције.

Пресек између ове две области уједно представља и пресек између математике, рачунарства, и лингвистике, те је стога и неупитно занимљиво проучити га. Дакле, главни циљ овог рада биће да читалац стекне слику о овој повезаности између теорија језика и аутомата, а особито о неким значајним теоремама и алгоритмима који нам помажу да их проучавамо.

# 2 Формални језици

## 2.1 Увод у формалне језике

Појам формалног језика се први пут појављује чак 1879. године, у једној књизи из математичке логике немачког математичара Готлоба Фрегеа. Фреге је описао први формални језик, са својим валидним симболима и правилима извођења, и управо из овог језика је израсло оно што је касније постало познато као предикатска логика. Тек касније су за ову област почели да се интересују људи који се баве природним језицима, јер се испоставило да је веома корисна за њихово проучавање.

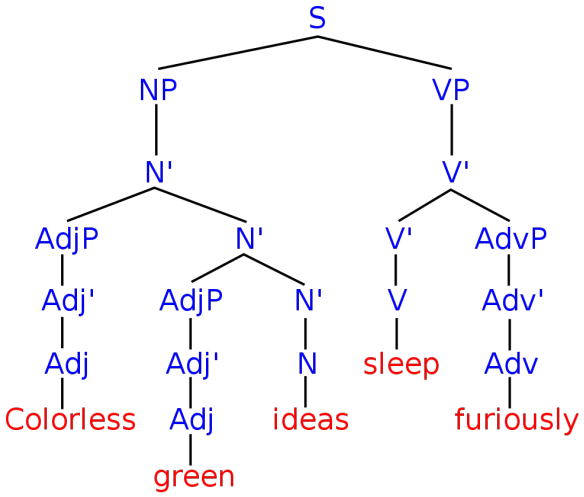
Овде би се требало пажљиво оградити и апсолвирати да теорија формалних језика објашњава функционисање језика на синтаксном, али не и на семантичком нивоу. Другим речима, бави се искључиво структурним односима између елемената језика, али не и смислености значења. Узмимо један класичан пример реченице (Чомски 1957.): "Безбојне зелене идеје спавају бесно." Ова реченица очигледно нема никаквог семантичког смисла - односи у значењу између било које две речи су апсолутно бесмислени. Но, ипак се сви можемо сложити да је ово савршено граматички исправна реченица (за разлику, нпр. од реченице "Спавају безбојне бесно идеје зелене"), па чак можемо и нацртати њено синтаксно стабло, као на слици 2.1. (Више о стаблима касније.)

Дакле, пошто смо дефинисали којим аспектима језика се бавимо, дефинишимо сада шта је то, заправо, језик.

**Дефиниција 1. Азбука** $\Sigma$ је било који коначан скуп симбола, које називамо *словима* те азбуке.

**Дефиниција 2.** Сваки коначан низ слова из азбуке $\Sigma$ називамо **реч (стринг) над азбуком** $\Sigma$.

Над сваком азбуком постоји и **празна реч**, у ознаци $\epsilon$, која не садржи ниједно слово из те азбуке. Из овога видимо да речи можемо дефинисати




**Слика 2.1:** Синтаксно стабло

и рекурзивно:

($i$) Празна реч $e$ је реч над азбуком $\Sigma$.

($ii$) Уколико је $w$ реч над $\Sigma$ и $a$ слово над $\Sigma$, онда је и $wa$ реч над $\Sigma$.

($iii$) Речи над $\Sigma$ могу се добити искључиво коначном применом правила ($i$) и ($ii$).

**Дефиниција 3. Дужина речи** $u$, у ознаци $|u|$, је број слова у тој речи.

Видимо да се и дужина може дефинисати рекурзивно:

($i$) $|e| = 0$.

($ii$) Уколико је $w$ реч над $\Sigma$ и $a$ слово над $\Sigma$, онда је $|wa| = |w| + 1$.

Сада можемо дефинисати основну операцију над речима - конкатенацију.

**Дефиниција 4.** Нека су дате речи $u = x_1x_2...x_n, v = y_1y_2...y_n$, при чему $u, v \in \Sigma$. Конкатенацијом ове две речи добијамо реч $uv = x_1x_2...x_ny_1y_2...y_n$.

Наведимо основна својства конкатенације:

**Теорема 1.** *Скуп речи језика заједно са операцијом конкатенације чини асоцијативни моноид. Односно:*

1° *Празна реч $e$ је неутрални елемент за операцију конкатенације, тј. $ex = xe = x$ за сваку реч $x$ над азбуком $\Sigma$.*

2° *Конкатенација је асоцијативна операција, тј. $(xy)z = x(yz)$ за све $x, y, z$ над $\Sigma$.*

3° *За конкатенацију важе закони скраћивања, односно $xz = yz \Rightarrow x = y$ и $zx = zy \Rightarrow x = y$.*

У аналогији са обичним множењем, можемо дефинисати и нотацију за степеновање речи:

$$w^0 = e, w^1 = w, ..., w^n = \underbrace{ww \dots w}_{n}$$

**Дефиниција 5.** Скуп свих речи над азбуком $\Sigma$ означавамо као $\Sigma^*$ (**Клинијево затворење**). Скуп свих непразних речи над азбуком $\Sigma$ означавамо као $\Sigma^+ = \Sigma^* \setminus \{e\}$.

**Дефиниција 6.** Прозивољан скуп речи над датом азбуком називамо **језиком** над том азбуком. Другим речима, језик над азбуком $\Sigma$ је свако $L \subset \Sigma^*$.

Примећујемо да је онда $\mathcal{P}(\Sigma^*)$ заправо скуп свих језика над неком азбуком, па на њему дефинишемо и следеће операције:

($i$) скуповне: $\cup, \cap, \setminus, '$ (комплемент у односу на $\Sigma^*$)

($ii$) производ: $L_1 \cdot L_2 = \{uv | u \in L_1, v \in L_2\}$

($iii$) степен: $L^0 = e, L^{n+1} = L^n \cdot L$

($iv$) итерација: $L^* = \bigcup_{i \geq 0} L^i$

**Напомена:** Често се уместо уније користи и обично $+$.

Дакле, дефинисали смо све најопштије појмове који се тичу формалних језика. Поиграјмо се сада са неким примерима како бисмо уочили нека битна својства.

**Пример 1.** Решити једначину $001u = u001$ над алфабетом $\{0, 1\}$.

*Решење:* Узмимо одмах празну реч $e$ као тривијално решење. Приметимо да стринг 001 мора бити и суфикс и префикс речи $u$, те ако узмемо

$u = 001v$ и убацимо у полазну једначину имамо $001001v = 001v001$, па из правила скраћивања можемо извући $001v = v001$, односно опет смо се вратили на полазну једначину (еквивалентно бисмо добили кад бисмо узели $u = v001$). Одавде, можемо наслутити да ће решење бити $(001)^*$, а наслућујемо и да ово можемо решити рекурзивно.

$\subseteq$: За свако $u = (001)^n$ видимо да је $001u = 001(001)^n = (001)^{n+1} = (001)^n001 = u001$, па ово јесте решење једначине.

$\supseteq$: Докажимо индукцијом по дужини речи $u$ да је свако решење облика $(001)^n$.

У индукцији ћемо искористити чињеницу да 001 мора бити и префикс и суфикс решења наше једначине како бисмо констатовали да дужина нашег решења мора бити облика $3k, k \in \mathbb{N}_0$, а да у друга два случаја решење не постоји.

За $|u| = 0$ знамо да $u = e = (001)^0$.

За $|u| = 1$ и $|u| = 2$ немамо решења (тривијално).

Претпоставимо сада да тврђење важи за $|u| = 3(k-1), 3k-2, 3k-1$. Тада за $|u| = n$ имамо $u = 001v$, где $|v| = n-3$, те ако убацимо то у полазну једначину имамо $001001v = 001v001$, што нам после скраћивања даје $001v = v001$. По индуктивној претпоставци, издвајамо три случаја. Уколико $n = 3k$, онда знамо да је $v = (001)^{k-1}$ решење једначине, те $u = 001(001)^{k-1} = (001)^k$. Уколико је $n = 3k+1$, онда знамо да је $|v| = 3k-2$, те не постоји решење. Исто важи и за $n = 3k+2$. Дакле, тврђење је доказано. $\square$

Примећујемо да смо у изради овог примера користили неке особине суфикса и префикса како бисмо решили једначину. Испоставља се да је ово генерална стратегија за решавање оваквих проблема, те ћемо формализовати овај процес у једну лему.

**Теорема 2** (Левијева лема)**.** *Нека су $a, b, c, d$ речи над азбуком $\Sigma$. Тада једнакост $ab = cd$ важи ако и само ако је задовољен један од следећих услова:*

*(i)* $a = c \wedge b = d$

*(ii)* $(\exists x \in \Sigma^+)(a = cx \wedge d = xb)$

(*iii*) $(\exists x \in \Sigma^+)(c = ay \wedge b = yd)$

*Доказ.* ($\Leftarrow$): Из (*i*) се тривијално изводи $ab = cd$. Из (*ii*) имамо $ab = (cx)b = c(xb) = cd$, док из (*iii*) имамо $ab = a(yd) = (ay)d = cd$. Дакле, овај смер важи.

($\Rightarrow$): Нека је $ab = cd$. Видимо да $a$ и $c$ морају почињати истим словима, те имамо три опције:

(*i*) $a = c$. Онда $ab = ad$, те после скраћивања добијамо $b = d$, што је прва ставка у леми.

(*ii*) $a = cx$. Онда $cxb = cd$, те после скраћивања добијамо $d = xb$, што је друга ставка у леми.

(*iii*) $c = ay$. Онда $ab = ayd$, те после скраћивања добијамо $b = yd$, што је трећа ставка у леми.

□

Поред решавања проблема који се тичу речи над језиком, можемо решавати и скуповне једначине у којима фигурирају читави језици. Ради лакшег решавања неких од ових проблема, можемо увести још једну корисну лему:

**Теорема 3** (Арденова лема)**.** *Нека су $L_1, L_2$ језици такви да $e \notin L_1$ и $L$ језик који задовољава релацију $L = L_1L + L_2$. Тада је $L = L_1^*L_2$.*

*Доказ.* ($\subseteq$): Користимо индукцију по дужини речи $u \in L$. Нека је $u = e$ и претпоставимо да $u \in L = L_1L + L_2$. Пошто $e \notin L_1$, знамо $e \in L_2$, те стога $e \in L_1^*L_2$.

Претпоставимо да за све речи $x \in L$ дужине $|x| \leq n$ важи $x \in L_1^*L_2$. Узмимо сада неко $u \in L, |u| = n + 1$. Прва могућност је да $u \in L_2 \subseteq L_1^*L_2$. Друга могућност је да $u \in L_1L$, односно $u = av$, где је $a \in L_1, v \in L$. Како $a \neq e$, то је $|v| < |u| = n + 1$, тј. $|v| \leq n$, па можемо применити индуктивну претпоставку. Дакле, $v \in L_1^*L_2$, па је $u = av \in L_1L_1^*L_2 \subseteq L_1^*L_2$. Овиме је прва инклузија доказана.

($\supseteq$): Опет користимо индукцију да докажемо $L_1^nL_2 \subset L$, по $n$. За $n = 0$ имамо $L_1^0L_2 = L_2 \subseteq L_1L + L_2 = L$.

Претпоставимо да тврђење важи за $n-1$. Тада за $n$ имамо $L_1^n L_2 = (L_1 L_1^{n-1}) L_2 = L_1(L_1^{n-1} L_2) \subseteq L_1 L \subseteq L_1 L + L_2 = L$. Према томе, $L_1^n L_2 \subseteq L$ за свако $n \in \mathbb{N}_0$, те стога $L_1^* L_2 \subseteq L$. Овиме је и друга инклузија доказана. □

Покажимо примену ове леме на једноставном примеру.

**Пример 2.** Нека су $L_1, L_2 \subseteq \{a, b\}^*$ језици такви да:

$$L_1 = \{e\} + \{a\}L_1 + \{b\}L_2$$

$$L_2 = \{e\} + \{b\}L_2$$

Наћи једноставне репрезентације тих језика.

*Решење:* Једноставном применом Арденове леме на другу једначину имамо:

$$L_2 = \{b\}^*\{e\} = \{b\}^*$$

Са друге стране, из прве једначине имамо:

$$L_1 = \{a\}^*(\{e\} + \{b\}L_2) = \{a\}^*(\{e\} + \{b\}\{b\}^*) = \{a\}^*\{b\}^*$$

□

Из претходних примера већ можемо да наслутимо да вршењем разних операција на неки језик можемо добити неке друге језике, а стекли смо и неку интуицију око тога како то функционише. За формализацију овог процеса ћемо увести појам *формалне граматике*.

## 2.2 Формалне граматике

У претходном одељку смо увели језике као подскупове речи над неком азбуком, и премда смо видели како да манипулишемо њима, ти језици су нам још увек дати "као такви", без неких суштинских правилности. За сада још увек нема оних синтаксних правила које смо помињали у уводу. Да бисмо ово увели, потребна су нам правила извођења и препознавања валидних речи у датом језику.

Изненађујуће, испоставља се да је прва формална граматика дефинисана чак у 5. веку пре наше ере. Тада је индијски филолог Панини у свом *Осмоглавнику* описао тачна математичка правила за граматичку структуру античког санскрит језика. Панинијева теорија да сви језици имају овакву хијерархијску структуру се до данас показала као тачна,

али је западном свету требало око 2500 година чак и да прихвати ову идеју, а камоли да је доведе на овај ниво формализма. Пре него што и ми формално уведемо појам граматике, стекнимо прво интуицију.

Граматику можемо замислити као скуп правила за трансформацију стрингова - у једном кораку један стринг у оквиру речи (леву страну) можемо заменити другим стрингом (десном страном). Разликујемо скуп *терминалних* и *нетерминалних* слова, тј. оних слова од којих се коначне речи језика састоје, и оних слова која нам само помажу у извођењу. Обично се терминални симболи означавају малим словима, а нетерминални великим. Такође издвајамо специјалан нетерминални симбол који се зове стартно (почетно) слово, тј. оно слово од којег крећемо. Језик генерисан граматиком, дакле, је скуп свих речи које можемо добити почевши од старт симбола и примењујући дата правила.

Прикажимо за почетак пар једноставних примера над скупом терминалних симбола $\{a, b\}$ и почетним симболом $S$:

**Пример 3.** Нека су дата правила извођења:

$$S \to aSb$$

$$S \to ba$$

Онда почињемо од $S$ и бирамо које ћемо правило да применимо. Приметимо да бирање другог правила значи да ћемо остати без $S$-ова, а како је $S$ једини нетерминални симбол, то значи да примена другог правила имплицира крај алгоритма. Дакле, посматрајмо онда примену правила 1. То би изгледало овако:

$$S \to aSb \to aaSbb \to aaaSbbb \to ...$$

Ово радимо све док не применимо правило 2, односно скуп валидних речи ће онда изгледати као:

$$L = \{ba, abab, aababb, aaababbb, ...\} = \{a^n bab^n | n \geq 0\}$$

**Пример 4.** Узмимо сада следећа правила:

$$S \to a$$

$$S \to SS$$

$$aSa \to b$$

Овај скуп правила делује донекле комплексније, док не уочимо неке корисне ствари. Прво, применом правила 2 можемо генерисати арбитрарно дугачак низ $S$-ова. Друго, применом правила 1 можемо одабране од тих $S$-ова претворити у $a$-ове. Коначно, уколико применимо правило 2 двапут, затим правило 1 двапут, а затим правило 3, добијамо следеће:

$$S \xrightarrow{(2)} SS \xrightarrow{(2)} SSS \xrightarrow{(1)} aSS \xrightarrow{(1)} aSa \xrightarrow{(3)} b$$

Дакле, произвољно $S$ такође можемо пребацити у $b$. Одавде видимо да заправо овом граматиком можемо генерисати било коју ниску слова $a$ и $b$, односно генеришемо језик:

$$L = \{a, b\}^*$$

Сада можемо увести и формалну дефиницију:

**Дефиниција 7.** Нека су:

- $N$ - коначан скуп нетерминалних симбола
- $\Sigma$ - коначан скуп терминалних симбола, дисјунктан са $N$
- $P$ - скуп правила извођења облика:

$$(\Sigma \cup N)^* N (\Sigma \cup N)^* \to (\Sigma \cup N)^*$$

- $S$ - почетно слово такво да $S \in N$

Тада је **формална граматика** уређена четворка $G = (N, \Sigma, P, S)$.

Дефинишимо неколико релација које ће нам бити корисне за формално записивање извођења.

**Дефиниција 8.** Нека је дата граматика $G = (N, \Sigma, P, S)$. Тада дефинишемо бинарне релације:

- "$y$ се непосредно изводи из $x$" као: $(x \underset{G}{\Rightarrow} y) \iff (\exists u, v, p, q \in (\Sigma \cup N))(x = upv \wedge y = uqv \wedge (p \to q) \in P)$
- "$y$ се изводи у $k$ корака из $x$" као: $(x \overset{k}{\underset{G}{\Rightarrow}} y) \iff x \underset{G}{\Rightarrow} x_1 \wedge x_1 \underset{G}{\Rightarrow} x_2 \wedge \ldots \wedge x_{k-1} \underset{G}{\Rightarrow} b$

- "$y$ се посредно изводи из $x$" или "$y$ се изводи у 0 или више корака из $x$" као рефлексивно и транзитивно затворење релације $\underset{G}{\Longrightarrow}$, у ознаци $x \overset{*}{\underset{G}{\Longrightarrow}} y$

**Дефиниција 9.** Нека је дата граматика $G = (N, \Sigma, P, S)$. Тада је **језик генерисан граматиком** $G$ дефинисан као:

$$L(G) = \{w | w \in \Sigma^* \wedge S \overset{*}{\underset{G}{\Longrightarrow}} w\}$$

Наш главни задатак из ове области би ишао у једном од два смера: или бисмо из дате граматике морали да пронађемо језик генерисан њоме, или бисмо за дати језик морали да конструишемо граматику која га генерише. Оба ова случаја ће се, у суштини, сводити на интуитивни "убод" решења, а потом на доказивање два смера инклузије математичком индукцијом. Примера ради, прикажимо овде један пример другог типа, тј. конструисања граматике на основу језика.

**Пример 5.** Конструисати граматику која генерише језик $W = \{a^i b^j | i \geq j > 0\}$

*Решење:* Нека је $G = (N, \Sigma, P, S)$, где је:

$$N = \{S, A\}$$

$$\Sigma = \{a, b\}$$

$$P = \{S \to aAb, A \to aAb, A \to aA, A \to e\}$$

Доказаћемо да је $W = L(G)$.

($\subseteq$) : Ако је $w \in W$, онда имамо следеће извођење преко граматике $G$:

$$S \overset{1}{\Rightarrow} aAb \overset{2\cdot(j-1)}{\Longrightarrow} a^j Ab^j \overset{3\cdot(i-j)}{\Longrightarrow} a^i Ab^j \overset{4}{\Rightarrow} a^i b^j$$

($\supseteq$) : Узмимо било које извођење у граматици $G$ које не садржи корак 4 (тривијално можемо извести да тај корак просто обуставља извођење и не мења облик финалне речи). Морамо доказати прво да ће сваки корак резултирати у стрингу из скупа $\{a\}^*\{A\}\{b\}^*$, а затим и да је број $a$-ова већи или једнак од броја $b$-ова (у ознаци $\#_a \geq \#_b$). Докажимо, дакле, математичком индукцијом по дужини извођења $n$, да ћемо увек имати

$a^k Ab^l, k \geq l$.

За $n = 1$ тврђење очигледно важи. Зато претпоставимо да важи за неко $n$, и погледајмо случај $n+1$. Знамо да $S \overset{n}{\underset{G}{\Longrightarrow}} w' \underset{G}{\Longrightarrow} w$. По индуктивној претпоставци знамо да је онда $w' = a^k Ab^l, k \geq l$. Како правило 1 мора да има $S$ на левој страни, то је овај последњи корак морао бити или типа 2 или типа 3. У случају 2, видимо да добијамо $a^{k+1}Ab^{l+1}$, и $k+1 \geq l+1$, те овде тврђење важи. У случају 3, добијамо $a^{k+1}Ab^l$, а знамо да је $k+1 \geq l$, те и овде тврђење важи.

Стога, када још применимо правило 4 на овако добијену реч, добићемо $a^k b^l, k \geq l$, а то припада језику $W$. □

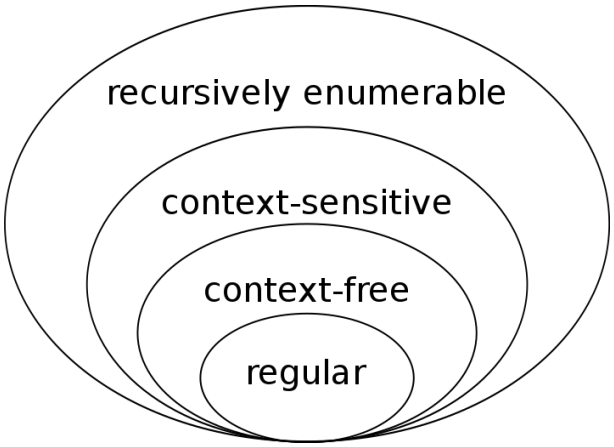
Из примера датих у овом одељку, можемо видети да међу формалним језицима постоји велика разноврсност, и да она у великој мери зависи од тога каква их граматика генерише. При решавању проблема из ове области, показаће се значајна потреба за класификацијом ових језика у односу на особине граматика које их генеришу. Сада ћемо описати ову класификацију, која се јавља у облику хијерархије.

## 2.3 Хијерархија Чомског

Ноам Чомски (1928-) познат је као отац савремене лингвистике из неколико разлога. Сем тога што је први успео да теорију о урођености језика учини широко прихваћеном и што је започео дубља истраживања у везу између језика и когниције, он је такође један од главних покретача теорије формалних граматика. Између осталог, он је први направио класификацију језика преко својстава њихових формалних граматика, која се и дан-данас користи у рачунарству.

Сама класификација се може замислити као скуп концентричних кругова, као на слици 2.2. Спољни круг представља језике у најширем смислу, док сваки следећи круг додаје све строже рестрикције на граматике тих језика, тако да што више идемо уз хијерархију, то ће број језика који су генерисани тим граматикама бити ужи.

Сем језика и граматике, са тачке гледишта рачунарства нам је такође веома битно да знамо какав тип машине, односно система, је способан да препозна сваки од ових класа. Под препознавањем језика, подразумевамо




**Слика 2.2:** Хијерархија Чомског

да у машину можемо као инпут убацити произвољну ниску, и да ће нам она као аутпут вратити да ли је дата ниска валидна реч у том језику или не.

Опишимо сада сваки од нивоа у овој хијерархији.

### 2.3.1 Рекурзивно-пребројиви језици

Рекурзивно-пребројиви језици су такође познати као *језици типа 0*. Ово је управо зато што граматике које генеришу рекурзивно-пребројиве језике на себи немају апсолутно никакве рестрикције, сем тога да постоје. Другим речима, језик је рекурзивно-пребројив уколико постоји алгоритам који може да преброји све ниске које он садржи.

Из овога се може видети да је систем који препознаје рекурзивно-пребројиве језике ни више ни мање него Тјурингова машина, те се ови језици зато и другачије зову *Тјуринг-препознатљиви језици.* Као што и обично бива, недостатак рестрикција над граматикама ових језика их чини не тако занимљивим за проучавање, бар на тренутном нивоу.

### 2.3.2 Контекстно-сензитивни језици

Контекстно-сензивитни (негде контекстно-зависни) језици су сви они језици који су генерисани граматикама типа 1, тј. оним граматикама чија су сва правила извођења облика:

$$\beta A\gamma \to \beta\alpha\gamma$$

где је $A$ нетерминалан симбол, $\alpha, \beta, \gamma$ ниске терминалних или/и нетерминалних симбола, и $\alpha$ није празна реч. Другим речима, правила извођења могу да узимају у обзир произвољну околину оног симбола који се трансформише.

Систем који препознаје ове језике је *линеарно-ограничена недетерминистичка Тјурингова машина* - простије речено, Тјурингова машина чија трака није бесконачна, већ има свој почетак и крај. Из овог разлога, иако је ово најближа апроксимација природног језика међу формалним језицима, ипак је изузетно ретко виђена у лингвистици, из простог разлога што је механички изузетно непрактична.

### 2.3.3 Контекстно-слободни језици

Контекстно-слободни (негде контекстно-независни) језици су они језици који су генерисани граматикама типа 2, тј. оним граматикама чија су сва правила извођења облика:

$$A \to \alpha$$

где је $A$ нетерминални симбол, а $\alpha$ ниска терминалних или/и нетерминалних симбола. Као што видимо, главна разлика између контекстно-слободних и -сензитивних језика је то што код контекстно-слободних валидност трансформације не зависи од околине трансформисаног слова.

Машина која препознаје ове језике је *недетерминистички потисни аутомат*. Више о овим системима, као и о контекстно-слободним језицима генерално, чућемо у поглављу 4. За сада би било корисно да уочимо да су ови језици заправо они који улазе у основу велике већине програмских језика, како у смислу њиховог дефинисања, тако и у контексту парсирања.

### 2.3.4 Регуларни језици

Регуларни језици су они језици који су генерисани граматикама типа 3, тј. оним граматикама у којима су или сва правила извођења облика:

$$A \to a, A \to aB$$

или су сва правила извођења облика:

$$A \to a, A \to Ba$$

где су $A, B$ нетерминални симболи, а $a$ терминални симбол. У првом случају кажемо да је граматика *десно-линеарна*, а у другом да је *лево-линеарна*. У некој литератури се једино десно-линеарне граматике воде као регуларне, но ово уме да буде доста збуњујуће.

Видимо да се ови језици изводе преко инкременталног додавања појединачних симбола, те су стога и најстрожи међу свим формалним језицима. Примера ради, језик $\{a^n | n \geq 1\}$ је регуларан, док већ језик $\{a^n b^n | n \geq 1\}$ то није (али јесте контекстно-слободан). Машине које препознају регуларне језике су заправо коначни аутомати, који су тема другог дела овог рада, те би било лепо да се овим језицима мало детаљније побавимо.

# 3 Регуларни језици

## 3.1 Регуларни изрази, језици и граматике

У претходном поглављу, констатовали смо да је главна карактеристика регуларних језика то што они "пишу и читају" своје речи слово по слово, и то увек прилазећи са исте стране. Из овога се одмах види колико су овакви језици корисни у рачунарству, те није ни чудо да се прво интересовање за ову област показано управо у том контексту.

Конкретно, регуларни језици се испостављају као идеално решење за имплементацију алгоритама претраге (као што је опција "*find and replace*"), алгоритама провере валидности уноса, али и лексичке анализе. Зарад овакве имплементације, често се узима и једна мало другачија дефиниција регуларности, преко увођења појма *регуларног израза*.

**Дефиниција 10.** Нека је $\Sigma$ коначна азбука. Регуларан израз над $\Sigma$ се онда дефинише рекурзивно, и то на следећи начин:

($i$) Празан скуп је регуларни израз.

($ii$) Празна реч $e$ је регуларни израз.

($iii$) Слово $a \in \Sigma$ је регуларан израз.

($iv$) Уколико су $R_1, R_2$ регуларни изрази, тада су регуларни и изрази $R_1 \cup R_2, R_1R_2, R_1^*$.

($v$) Регуларни изрази над $\Sigma$ се добијају искључиво коначном применом правила $(i) - (iv)$.

Приказаћемо неколико једноставних примера регуларних израза, ради интуиције.

**Пример 6.** Регуларни израз који представља скуп свих бинарних речи где се подреч 001 јавља барем једанпут: $(0+1)^*001(0+1)^*$.

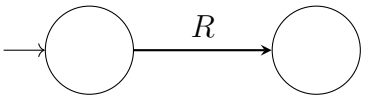
**Пример 7.** Регуларни израз који представља све бинарне бројеве дељиве са 4: $1(00)^*$.

**Пример 8.** Регуларни израз који представља све валидне и-мејл адресе: $((a-z)+(A-Z)+(0-9))^*@(a-z)^*.(a-z)^*$.

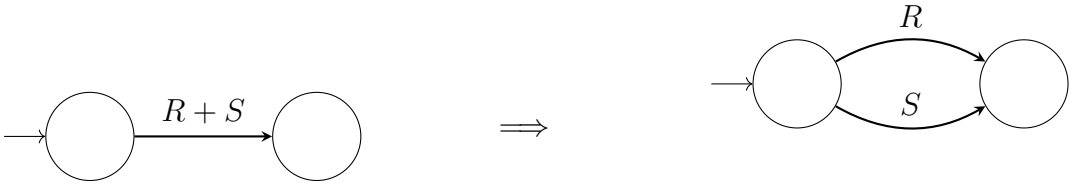
Из ових примера, видимо како је нотација коју смо увели постала изузетно корисна за баратање овим језицима. Мотивисани овиме, природно нам долази да сада уведемо нов начин репрезентовања регуларних израза - преко усмерених графова.

Узмимо произвољни регуларни израз $R$. Желимо да направимо граф у којем ће свако теме представљати неко "стање", а свака стрелица ће представљати додавање извесног симбола на десну страну стринга (присетимо се десно-линеарних језика). Можемо ли овако представити израз $R$? Ево скице једног покушаја алгоритма.

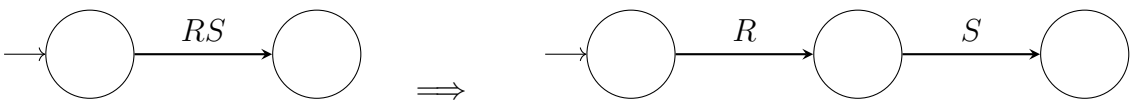
Први корак ће бити да конструишемо почетно и крајње стање, а између њих стрелицу означену са $R$.



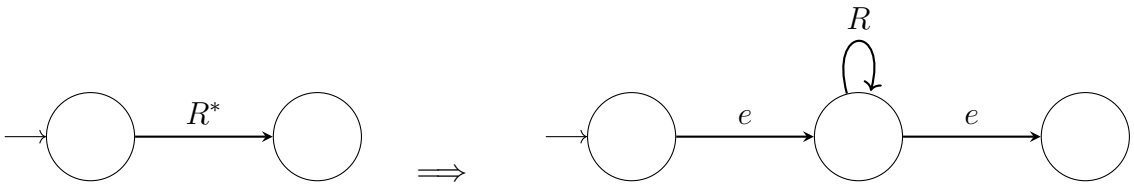

Затим ћемо увести трансформације за различите операције над регуларним изразима. Сваку грану означену са $R+S$, односно унијом два регуларна израза, мењамо са две гране са истом почетном и крајњом тачком.




За сваку грану означену са $RS$ ћемо додати ново стање између два стања, а затим повезати их на следећи начин:



Коначно, за сваку грану означену са $R^*$, додаћемо стање између, на којем ћемо имати петљу:



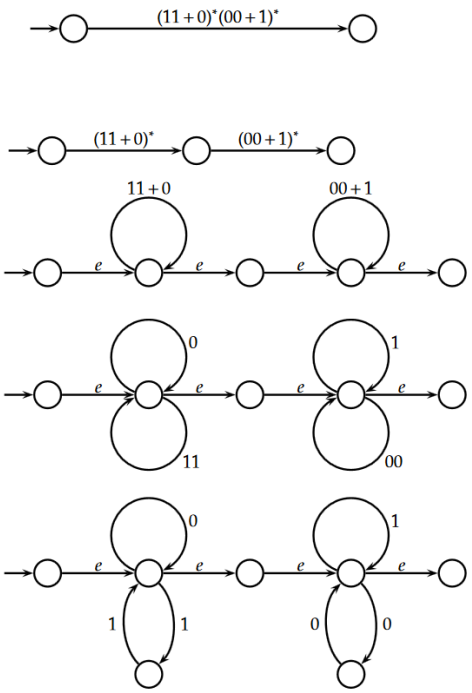

Напослетку, обришемо све гране које имају ознаку $\emptyset$, и добили смо усмерени граф који "препознаје" дати регуларни израз. Како смо имплементирали сва ова правила, знамо да ћемо моћи да доведемо граф до тачке где је свака стрелица означена тачно једним словом. Погледајмо како то изгледа на примеру:

**Пример 9.** Конструисати граф $G(r)$ регуларног израза $(11+0)^*(00+1)^*$.

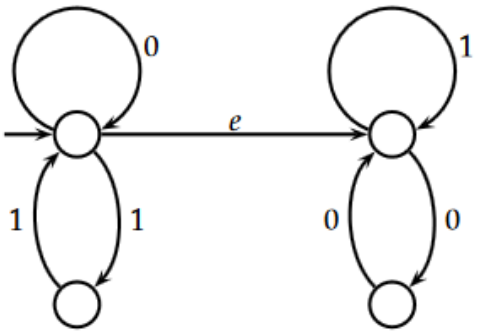
Решење је скицирано на слици 3.1.

Једноставно можемо показати да се свака $e$-грана графа која је јединствена излазна свом почетном чвору или јединствена улазна свом крајњем чвору може скупити у један чвор тако да се скуп израза које тај граф препознаје не мења, те стога ово решење можемо додатно упростити, као на слици 3.2. □

Ова репрезентација преко усмерених графова се показује као изузетно згодна за описивање регуларних израза, а особито је корисна зато што служи као репрезентација потенцијално бесконачног скупа израза преко коначног скупа стања. Стога, било би пожељно да формализујемо овај концепт. Ови усмерени графови заправо представљају репрезентацију једног коначног скупа стања и прелаза између тих стања, а то је управо оно што зовемо *коначни аутомат*.




**Слика 3.1:** Решење примера 9




**Слика 3.2:** Боље решење примера 9

## 3.2 Коначни аутомати

У претходном поглављу смо увели метод који нам омогућава да направимо усмерени граф за произвољни регуларни израз. Уколико уз оно што смо већ увели издвојимо још и једно почетно стање и подскуп финалних стања, оно што ћемо добити је коначни аутомат.

**Дефиниција 11.** Нека су:

- $Q$ - непразан коначан скуп стања
- $\Sigma$ - улазна азбука
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ - функција преласка
- $q_0 \in Q$ - почетно стање
- $F \subseteq Q$ - скуп финалних стања

Онда уређену петорку $M = (Q, \Sigma, \delta, q_0, F)$ називамо **коначним аутоматом**.

Функционисање овог аутомата можемо да замислимо на два начина - генеративни и акцепторски. У првом случају, улазимо у почетно стање са празним стрингом, и онда пролазимо кроз све могуће путеве у графу, дописујући симболе с десне стране, док не дођемо до финалног стања. Скуп свих стрингова које добијамо у финалним стањима је језик над тим аутоматом. У другом случају, улазимо у почетно стање са датом речју, и онда пролазимо кроз граф, бришући означени карактер са леве стране при сваком прелазу. Кад дођемо до празне речи, уколико смо у финалном стању то значи да аутомат прихвата тај стринг, а у супротном не прихвата. Еквиваленција између ова два погледа је очигледна.

Сетимо се како смо при анализи формалних граматика проширили релацију непосредног извођења на посредно извођење, односно извођење у нула или више корака. Било би добро да сада сличну ствар урадимо и за функцију преласка у аутомату.

**Дефиниција 12.** Функцију $\delta$ са домена $Q \times \Sigma$ индуктивно проширујемо на $Q \times \Sigma^*$ на следећи начин:

$(i)$ $\delta(a, e) = a$

$(ii)$ за свако $u \in \Sigma^*$ и $x \in \Sigma$, уколико је дефинисано $\delta(a, u)$, важи:

$$\delta(a, ux) = \delta(\delta(a, u), x)$$

**Табела 3.1:** Функција прелаза за пример 10

| $\delta$ | $q_0$ | $q_1$ | $q_2$ | $q_3$ |
|---|---|---|---|---|
| 0 | $q_1$ | $q_2$ | $q_2$ | $q_3$ |
| 1 | $q_3$ | $q_3$ | $q_2$ | $q_3$ |

Одавде се лако изводи и да за произвољне $u, v \in \Sigma^*$ важи:

$$\delta(a, uv) = \delta(\delta(a, u), v)$$

**Дефиниција 13.** Нека је дат аутомат $M$. Тада скуп $L(M) = \{u \in X^* | \delta(q_0, u) \in F\}$ називамо **језиком аутомата** $M$.

Приметимо да је кодомен функције прелаза партитивни скуп скупа стања $Q$. Ово значи да, уколико се налазимо у неком стању и желимо да додамо/поједемо неки симбол, може се десити да такав прелаз или не постоји, или пак да их постоји више, те морамо да бирамо. Пошто ово делује као компликован проблем, упростимо га прво на специјалан случај где је кодомен функције прелаза само скуп $Q$. Дакле, за дато стање $q$ и карактер $a$, постоји тачно једно стање $p$ такво да $\delta(q, a) = p$. Овакве аутомате зовемо **детерминистички аутомати**.

### 3.2.1 Детерминистички коначни аутомати

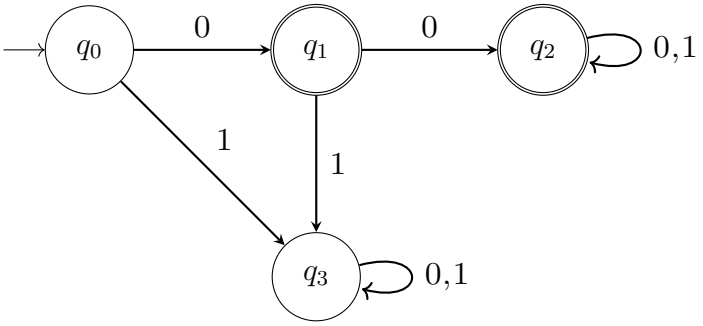
Прикажимо принцип функционисања детерминистичког аутомата на једном примеру.

**Пример 10.** (а) Конструисати граф прелаза аутомата $M = (Q, \Sigma, \delta, q_0, F)$, где је $Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{0, 1\}$, $F = \{q_1, q_2\}$, и функција прелаза $\delta$ дата табелом 3.1.

(б) Да ли аутомат $M$ прихвата речи 000 и 010?

(в) Одредити језик аутомата $M$.

*Решење:* (а) Граф прелаза конструишемо лако, уз напомену да финална стања обележавамо двоструким кружићима.




(б) Пут означен са 000 даје:

$$\delta(q_0, 000) = \delta(\delta(q_0, 0), 00) = \delta(q_1, 00) = \delta(\delta(q_1, 0), 0) = \delta(q_2, 0) = q_2 \in F$$

дакле аутомат прихвата реч 000. Са друге стране, 010 даје:

$$\delta(q_0, 010) = \delta(\delta(q_0, 0), 10) = \delta(q_1, 10) = \delta(\delta(q_1, 1), 0) = \delta(q_3, 0) = q_3 \notin F$$

дакле аутомат не прихвата реч 010.

(в) Најпре, приметимо да су $q_2$ и $q_3$ "ћорсокаци", тј. ако у њих уђемо не можемо да изађемо, с тим што је $q_2$ прихватајуће стање, а $q_3$ није. Стога, све речи које почињу са 1 или 01 нису прихваћене. Дакле, чим почнемо са 0, ушли смо у завршно стање, дакле $0 \in L(M)$. Затим морамо опет да направимо 0 корак, те завршавамо у финалном ћорсокаку $q_2$, где можемо да додамо произвољно нула и јединица. Дакле, језик овог аутомата је скуп:

$$L(M) = 0 + 00(0 + 1)^*$$

Овај проблем смо такође могли формалније решити постављањем система једначина и применом Арденове леме. Наиме:

$$L(M) = L_1 + L_2$$
$$L_2 = L_1 0 + L_2(0 + 1)$$
$$L_1 = L_0 0$$
$$L_0 = e$$

Одакле можемо извести редом:

$$L_1 = e0 = 0$$

$$L_2 = L_2(0+1) + 00 = 00(0+1)^*$$
$$L(M) = 0 + 00(0+1)^*$$

□

Сетимо се примера из претходног одељка где смо свели аутомат заснован на регуларном изразу на једноставнију репрезентацију истог. Очигледно, неће сваки аутомат који конструишемо за дати језик бити у свом најједноставнијем могућем облику. Из овог разлога, један од значајних проблема у овој области је проблем *минимизације аутомата*. Овде ћемо неформално описати најефикаснији алгоритам минимизације - *Хопкрофтов алгоритам*.

Приликом минимизације, постоје три типа стања која нам праве проблем:

- недостижна стања - стања до којих не можемо доћи из почетног стања; идеја је да их обришемо
- мртва стања - стања из којих не можемо доћи до финалног стања; идеја је да их све спојимо у једно мртво (тј. неприхватајуће) стање
- нераспознатљива стања - неке групе стања која су међусобно еквивалентна; идеја је да скуп стања испартиционирамо у класе еквивалентних стања

Недостижна и мртва стања је прилично једноставно решити. Остаје још питање нераспознатљивих стања. Суштина Хопкрофтовог алгоритма је да кренемо од две најгрубље класе еквиваленције - скупова $F$ и $Q \setminus F$ - и затим да полако раздвајамо те класе на мање класе уколико пронађемо два нееквивалентна стања.

Формирајмо радну листу $W$, у којој чувамо све класе које морамо проверити у датом кораку. Затим узимамо једну класу $A$ и вадимо је из $W$, те за свако слово $c$ у улазној азбуци нађемо скуп $X$ стања која су достижна из $A$ преко $c$. Затим, тражимо оне класе $Y$ из тренутне партиције за које је $X \cap Y \neq \emptyset$ и $Y \setminus X \neq \emptyset$, тј. оне $Y$ из којих постоје и стања која воде у $A$ и која не воде у $A$. Оваква стања $Y$ ћемо у партицији поделити на $X \cap Y$ и $Y \setminus X$, док ћемо у радну листу убацити или иста та два скупа (уколико је $Y$ непроверен), или мањи од та два (уколико је $Y$ проверен). Ово радимо чисто да бисмо избегли непотребне провере. Овиме смо поделили "проблематичну" класу $Y$ на две нееквивалентне, те понављамо цео процес за неко друго $A$, све док нам се радна листа $W$

не испразни.

Применом овог алгоритма добијамо партицију скупа стања на неекививалентне класе, те смо се решили свих проблематичних стања. Дакле, овиме смо добили минимални детерминистички аутомат.

### 3.2.2 Недетерминистички коначни аутомати

До сада смо се бавили специјалним случајем детерминистичких аутомата, али шта ако узмемо општији, недетерминистички случај? Како над овим аутоматима има много мање рестрикција, очекивали бисмо да ће нам рад са њима бити "генералнији", тј. да ћемо уз помоћ њих моћи да генеришемо неке нове језике.

Ипак, испоставља се да то није случај. Наиме, недетерминистички и детерминистички аутомати су еквивалентни, што нам говори следећа теорема:

**Теорема 4** (Робин-Скотова теорема)**.** *За језик $L$ постоји недетерминистички коначни аутомат који га прихвата ако и само ако постоји детерминистички коначни аутомат који га прихвата.*

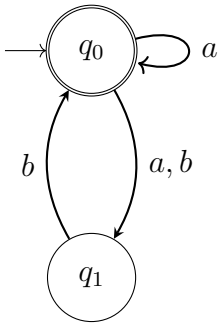
*Доказ.* Сваки детерминистички аутомат је недетерминистички, дакле довољно је доказати да сваки недетерминистички аутомат $N$ можемо трансформисати у детерминистички аутомат $D$ такав да $L(D) = L(N)$.

Како из сваког стања у $N$ применом одређеног слова долазимо у неки скуп стања, то ће се наш доказ заснивати на конструисању аутомата $D$ чија су стања елементи партитивног скупа од скупа стања аутомата $N$, и ићи ћемо корак по корак.

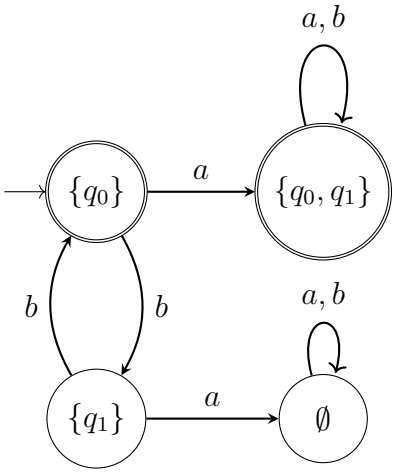
Након нула прочитаних карактера, $N$ се може налазити или у почетном стању $q_0$, или у неком од стања до којих се може стићи применом само симбола $e$. Овај скуп обележавамо са $E(q_0) = \{q \in \delta(q_0, e)\}$, и ово је почетно стање аутомата $D$. Узмимо сада неко стање $q_i \in E(q_0)$ и рецимо да учитавамо карактер $a_i$, дакле $N$ иде у стање $q \in \delta(q_i, a_i)$, а затим опет можемо применити $e$-правила колико год можемо. Дакле, можемо доћи у стања из скупа $S = \bigcup_{q \in \delta(q_i, a_i)} E(q)$, те у аутомату $D$ повлачимо стрелицу између $E(q_0)$ и $S$. Исто ово примењујемо док нисмо исцрпили све конфигурације, и онда смо добили детерминистички аутомат из недетерминистичког, чиме је доказ комплетиран. □

Прикажимо функционисање ове трансформације на простом примеру.

**Пример 11.** Конструисати детерминистички аутомат који одговара следећем недетерминистичком:




*Решење:* Како нема $e$-стрелица, то је иницијално стање новог аутомата $q_0$. Уколико из $q_0$ применимо $a$, можемо доћи у стање из скупа $\{q_0, q_1\}$. Уколико из $q_0$ применимо $b$, сигурно идемо у $q_1$. Уколико из $q_1$ применимо $a$, идемо у празан скуп. Уколико из $q_1$ применимо $b$, идемо назад у $q_0$. Уколико из $\{q_0, q_1\}$ применимо $b$, идемо назад у исти чвор. Уколико из $\{q_0, q_1\}$ применимо $a$, добијамо опет исти тај скуп. Из празног скупа не можемо изаћи. Немамо више нових скупова (прошли смо цео партитивни скуп), дакле нови аутомат изгледа овако:




□

Поред анализе формалних језика, теорија аутомата је веома корисна и применљива у разним областима. Већина аутоматизованих уређаја (аутомати за кафу, лифтови, семафори, браве) функционишу по принципу аутомата. Алгоритми понашања ентитета у видео-игрицама и сличним

медијима су програмирани преко аутомата. Наравно, и примене у рачунарству су небројене. Ипак, дошло је време да се вратимо на тему регуларних језика, јер нам је остала још једна последња мистерија.

## 3.3 Лема о пумпању

На почетку овог поглавља смо дефинисали регуларне изразе на рекурзиван и чисто теоретски начин. Овај начин дефинисања нам је био прилично згодан, но када се дође до суштине видимо да заправо није нарочито експлицитан. Наиме, преко ове дефиниције тешко можемо да утврдимо да ли је неки језик заправо регуларан или није.

Овде ће нам теорија аутомата бити од велике помоћи. Откако смо увели аналогију између регуларних израза и усмерених графова, а потом то формализовали у аутомате, успели смо да добијемо неке прилично експлицитне резултате. Стога, хајде да формализујемо ту аналогију коју смо већ скицирали преко једне од кључних теорема из ове области.

**Теорема 5** (Клинијева теорема)**.** *Језик $L$ је регуларан ако и само ако постоји коначан аутомат који га препознаје.*

Ова теорема ће нам помоћи да изведемо можда и најважнију лему када је испитивање регуларности језика у питању, а то је *лема о пумпању.*

**Теорема 6** (Лема о пумпању)**.** *За сваки регуларан језик $L$ постоји број $p$, такав да свака реч $s \in L$ дужине бар $p$ може бити записана као $s = xyz$, тако да важи $|xy| \leq p, |y| > 0$, и $xy^kz \in L$ за свако $k \in \mathbb{N}_0$.*

*Доказ.* Нека је $A$ детерминистички аутомат који препознаје језик $L$, и нека је $|Q| = p$ број његових стања. Узмимо било коју реч $s = c_1c_2...c_{|s|}$ дужине $p$, и погледајмо низ стања кроз које $A$ пролази кад унесемо $s$:

$$q_1 \xRightarrow{c_1} q_2 \xRightarrow{c_2} ... \xRightarrow{c_{|s|}} q_{|s|+1}$$

Овај низ има више елемената него што аутомат има стања, па постоји неко стање које се понавља бар два пута у првих $p + 1$ елемената. Нека су то $q_i = q_j, i < j \leq p + 1$. Онда можемо да поставимо $x = c_1c_2...c_{i-1}, y = c_i...c_{j-1}, z = c_j...c_{|s|}$.

Пошто је $j \leq p + 1$ знамо $|xy| \leq p$, а пошто $i < j$ то $|y| > 0$, а видимо још и да $y$ можемо да пропустимо кроз аутомат произвољно пута, и наш стринг ће и даље бити прихваћен, јер се увек на крају вратимо у исто стање. Дакле, лема о пумпању је доказана. □

Тврђење леме о пумпању можемо интуитивно формулисати на следећи начин: У регуларном језику, унутар сваке довољно дугачке речи можемо да издвојимо један стринг који можемо да множимо у недоглед и да и даље добијемо важећу реч.

Приметимо да импликација у овој леми иде само у једном смеру, што значи да из важења леме о пумпању не можемо закључити да је језик регуларан. Ипак, она ће нам бити од велике помоћи у решавању једног другог проблема, а то је доказивање да неки језик није регуларан.

Пар илустративних примера:

**Пример 12.** Доказати да језик $\{a^n b^n | n \geq 0\}$ није регуларан.

*Решење:* Претпоставимо да је језик регуларан, тј. да постоји неко $p$ које задовољава услове леме о пумпању. Узмимо $w = a^p b^p \in L$, што је реч дужа од $p$, те по леми о пумпању постоји $w = xyz$ такво да $|xy| \leq p, |y| \geq 1, xy^i z \in L$ за свако $i \in \mathbb{N}_0$. Из првог услова знамо да $y$ садржи само слово $a$, а из другог да садржи бар једно $a$. Но, онда $xy^2z$ садржи више $a$ него $b$, па не припада језику, што је у супротности са трећим условом. Контрадикција. Дакле, језик није регуларан. □

**Пример 13.** Доказати да језик $\{x^{k^2} | k \in \mathbb{N}\}$ над азбуком $X = \{x\}$ није регуларан.

*Решење:* Претпоставимо да је језик регуларан, тј. да постоји неко $p$ које задовољава услове леме о пумпању. Узмимо $w = x^{p^2} \in L$, што је реч дужа од $p$, те по леми о пумпању постоји $w = ayb$ такво да $|xy| \leq p, |y| \geq 1, xy^i z \in L$ за свако $i \in \mathbb{N}_0$. Из прва два услова знамо $1 \leq |y| \leq p$, па испитујемо број слова речи $ay^2b$.

$$p^2 = |w| = |ayb| < |ay^2b| = |ayb| + |y| \leq p^2 + p < p^2 + 2p + 1 = (p+1)^2$$

Видимо да је дужина напумпане речи између два квадрата, те она дефинитивно не може припадати језику $L$. Контрадикција. Дакле, језик није регуларан. □

У овом поглављу смо се позабавили регуларним језицима и коначним аутоматима - најужим подскупом формалних језика. За крај, идемо један степеник навише у хијерархији, и бацамо поглед на *контекстно-слободне језике*.

# 4 Контекстно-слободни језици

Претходно поглавље се бавило регуларним језицима, за које би требало да смо до сада стекли интуицију да се препознају по принципу "слово по слово". Ово је сјајна ствар за компјутере, али не толико за људска бића. Наиме, истраживања у последњих 70 година су показала да људски ум заправо перципира језик као хијерархијску структуру, која се састоји од више рекурзивних "блокова". Ради успостављања овакве структуре су заправо први пут проучавани контекстно-слободни језици, премда су године показале да се природни језици у хијерархији Чомског ипак пре налазе у околини контекстно-сензитивног нивоа. Но, како је то сувише комплексно за нас, задржаћемо се на контекстно-слободним за сада, те ћемо у овом поглављу описати нека занимљива својства оваквих језика на чисто рекреативном нивоу.

Присетимо се прво дефиниције контекстно-слободне граматике. То су оне граматике чија су сва правила извођења облика:

$$A \to \alpha$$

где је $A$ нетерминални симбол, а $\alpha$ произвољна реч. Другим речима, контекстно-слободна граматика мења нетерминалне симболе стринговима, без обазирања на контекст око тих симбола. Неколико једноставних примера:

**Пример 14.** Скуп речи конкатенираних са својом инверзном речју, нпр. генерисан граматиком $G = (\{S\}, \{a, b\}, P, S)$, где $P$ садржи:

$$S \to aSa$$
$$S \to bSb$$
$$S \to e$$

Извођење у овом језику би, на пример, било:

$$S \to aSa \to abSba \to abbSbba \to abbbbba$$

**Пример 15.** Формуле исказне логике, генерисане граматиком:

$$G = (\{Izraz\}, \{p, q, r, \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, (, )\}, P, Izraz)$$

где $P$ садржи следећа правила:

$$Izraz \to p$$
$$Izraz \to q$$
$$Izraz \to r$$
$$Izraz \to \neg Izraz$$
$$Izraz \to Izraz \wedge Izraz$$
$$Izraz \to Izraz \vee Izraz$$
$$Izraz \to Izraz \Rightarrow Izraz$$
$$Izraz \to Izraz \Leftrightarrow Izraz$$
$$Izraz \to (Izraz)$$

Извођење у овом језику би, на пример, било:

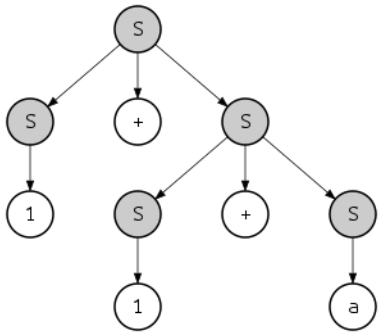
$$\begin{aligned} & Izraz \\ \to\ & Izraz \wedge Izraz \\ \to\ & (Izraz) \wedge Izraz \vee Izraz \\ \to\ & (Izraz \Rightarrow Izraz) \wedge q \vee \neg Izraz \\ \to\ & (p \Rightarrow r) \wedge q \vee \neg r \end{aligned}$$

**Пример 16.** Мали подскуп српског језика, генерисан граматиком:

$$G = (\{Recenica, ImSint, GlagSint, Im, Glag, Prid\},$$
$$\{Marko, macke, voli, sutira, male, debele\}, P, Recenica)$$

где $P$ садржи правила:

$$Recenica \to ImSint\ GlagSint$$
$$ImSint \to Prid\ ImSint$$
$$ImSint \to Im$$
$$GlagSint \to Glag\ ImSint$$
$$GlagSint \to Glag$$




**Слика 4.1:** Стабло извођења

$$Im \rightarrow Marko$$

$$Im \rightarrow macke$$

$$Glag \rightarrow voli$$

$$Glag \rightarrow sutira$$

$$Prid \rightarrow male$$

$$Prid \rightarrow debele$$

Пример извођења у овом језику би био, на пример:

$$Recenica$$
$$\rightarrow ImSint\ GlagSint$$
$$\rightarrow Im\ Glag\ ImSint$$
$$\rightarrow Im\ Glag\ Prid\ ImSint$$
$$\rightarrow Im\ Glag\ Prid\ Prid\ ImSint$$
$$\rightarrow Im\ Glag\ Prid\ Prid\ Im$$
$$\rightarrow Marko\ voli\ male\ debele\ macke$$

Често се уместо овакве нотације користе *стабла извођења*, односно *парсирајућа стабла*, која су мало лепши визуелни приказ извођења одређене речи. Конкретан пример за извођење стринга $1 + 1 + a$ приказан је на слици 4.1.

Приметимо да ово стабло није јединствено, тј. могли бисмо за исти стринг да имамо више извођења. Стога, често је корисно да доведемо сва правила до неке нормалне форме.

**Теорема 7** (Нормална форма Чомског)**.** *Свака контекстно-независна граматика се може свести на граматику чија су сва правила извођења облика:*

$$A \to BC$$

$$A \to a$$

$$S \to e$$

*где су $A, B, C$ нетерминални симболи, а терминални симбол, и $S$ стартни симбол.*

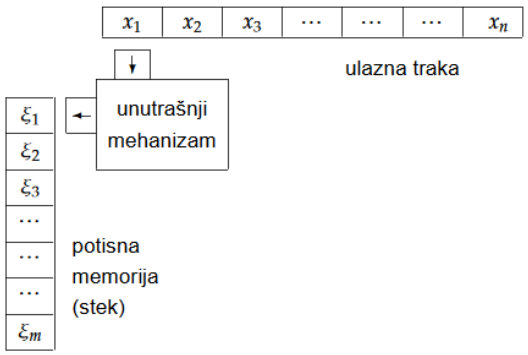
*Доказ.* Скицираћемо укратко правила којима можемо свести било коју граматику у овај облик.

- Елиминисање почетног симбола са левих страна: Додајемо нови почетни симбол $S_0$ и правило $S_0 \to S$, и проблем је решен.
- Елиминисање комбинације нетерминалних и терминалних са десних страна: За сваки терминални симбол $a$ који се појављује на десној страни, направимо нови нетерминални $N_a$, те онда свако правило $A \to \alpha a \beta$ заменимо са $A \to \alpha N_a \beta$ и $N_a \to a$.
- Елиминисање десних страна са више од 2 нетерминална симбола: Свако $A \to X_1 X_2 ... X_n$ мењамо са $A \to X_1 A_1$, $A_1 \to X_2 A_2, ..., A_{n-2} \to X_{n-1} X_n$
- Елиминисање десних страна са само једним нетерминалним симболом: Ако имамо $A \to B$, онда на свако $B \to \alpha$ додајемо и $A \to \alpha$.

□

Ова форма је веома корисна као метод припреме извесног контекстно-слободног језика за парсирање. Природно је поставити питање како парсирамо ове језике, тј. која је машина која их препознаје? Испоставља се да ће и ово бити један тип аутомата, али нешто комплекснији од оних коначних са којима смо се сусретали код регуларних језика - *потисни аутомати.*

Потисни аутомат је сличан обичном коначном аутомату, уз једну битну разлику - присуство потисне меморије, односно стека. Дијаграм потисног аутомата приказан је на слици 4.2.




**Слика 4.2:** Дијаграм потисног аутомата

Као што видимо, потисни аутомат при прелазу не узима у обзир само тренутно стање и симбол који се чита, већ и симбол који се налази у стеку. Такође, при вршењу преласка, он поред "једења" симбола са улазне траке такође може да манипулише стеком, тако што или "поједе" актуелни симбол у стеку, или потисне стек на доле и дода нови симбол (стога се и зове потисни аутомат). Ова додатна меморија нам омогућава шири опсег при генерисању и препознавању језика.

Формална дефиниција потисног аутомата је ужасно компликована, те је стога у овом рекреативном поглављу нећемо наводити. Но, заинтересовани је врло лако могу пронаћи у било којој литератури на тему контекстно-слободних граматика.

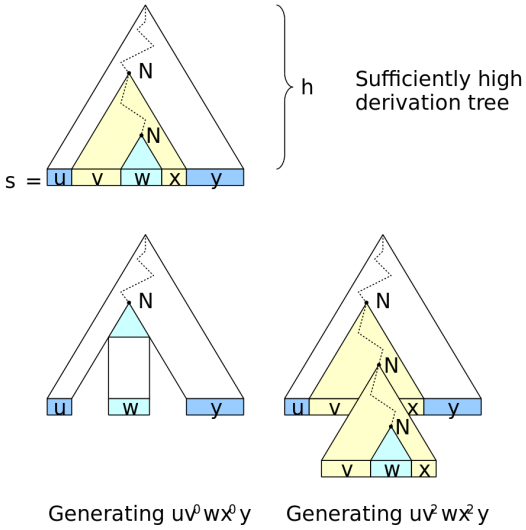
Слично као код регуларних језика, ово се може искористити за доказ нове варијанте леме о пумпању:

**Теорема 8** (Лема о пумпању за контекстно-слободне језике)**.** *За сваки контекстно-слободан језик $L$ постоји број $p$, такав да свака реч $s \in L$ дужине бар $p$ може бити записана као $s = uxyzv$, тако да важи $|xyz| \leq p, |xz| > 0$, и $ux^kyz^kv \in L$ за свако $k \in \mathbb{N}_0$.*

Визуелну репрезентацију ове варијанте леме о пумпању можете видети на слици 4.3.

За крај, имајући у виду све ове изузетно специфичне теореме и алгоритме које смо изложили овде, било би занимљиво видети колико проблема на тему контекстно-слободних језика су заправо одлучиви. Изложићемо овде неколико питања и видети како се будемо провели.

- **Парсирање:** За дату контекстно-слободну граматику, можемо ли




**Слика 4.3:** Скица нове леме о пумпању

проверити да ли дата реч припада језику који она генерише? Као што смо поминьали, ово је дефинитивно одлучив проблем. Познати алгоритми за решавање овога су Кук-Јангер-Касами алгоритам и Ерлијев парсер. ✓

- **Достижност:** Можемо ли за дати симбол утврдити да ли се може достићи из неког другог симбола? Слично као горе, одговор је да дефинитивно можемо. ✓

- **Празност и коначност:** Можемо ли закључити да ли је језик генерисан граматиком празан или коначан? Хопкрофт и Улман су 1979. формализовали алгоритме за решавање ових проблема, дакле и они су одлучиви. ✓

- **Регуларност:** Овде већ наилазимо на проблем. Наиме, из дефиниције регуларних граматика је очигледно да је регуларност једне контекстно-слободне граматике одлучив проблем. Но, уколико нам је дат контекстно-слободан језик, утврдити да ли је он регуларан је неодлучив проблем. (Премда нам лема о пумпању доста помаже да "изнесемо ђубре"). ×

- **Универзалност:** Генерише ли дата граматика скуп свих речи над датом азбуком? Ово се заправо своди на халтинг проблем код

Тјурингових машина, који је познат пример неодлучивог проблема, дакле и овај проблем је неодлучив. ×

- **Једнакост:** Генеришу ли две граматике исти језик? Неодлучивост овог проблема је директна последица неодлучивости претходног. Наиме, немогуће је утврдити ни једнакост језика генерисаног граматиком са скупом свих речи над азбуком, а камоли међусобну једнакост. ×

- **Инклузија:** Може ли једна граматика генерисати све стрингове које може и друга? Одлучивост овог проблема би значила одлучивост проблема једнакости (јер је једнакост скупова заправо спој две инклузије), а како једнакост није одлучива, то није ни инклузија. ×

- **Дисјунктност:** Јесу ли језици генерисани двема граматикама дисјунктни? Одлучивост овог проблема би значила и одлучивост познатог Постовог проблема дописивања, који је доказано неодлучив. Стога, ни дисјунктност није одлучива. ×

Као што видимо, поред свог формализма ове теорије, већина проблема који се јављају у контекстно-независним језицима (а и формалним језицима генерално) су неодлучиви. Ипак, контекстно-слободни језици су се показали као вероватно најзанимљивија класа међу формалним језицима, како за програмере тако и за лингвисте.

# 5 Закључак

Циљ овог рада био је да пружи читаоцу темељит увод у теорију језика, али са што мање оног претераног формализма и компликованог записа који (вероватно оправдано) доминира над свим скриптама из ове области које је аутор виђао.

У поглављу 2, видели смо како језик може да се формализује као математички систем и увели основне појмове његовог проучавања. Увели смо формалне граматике како бисмо језику дали структуру, а на основу особина граматике смо и класификовали све језике у хијерархију Чомског.

У поглављу 3, бавили смо се најужим степеником у хијерархији - регуларним језицима. Објаснили смо шта су регуларни изрази и какве везе имају са усмереним графовима, те смо тако дефинисали и наше прве аутомате. Научили смо како ове машине функционишу, те смо научено применили на регуларне језике.

Напослетку, у поглављу 4 смо се неформално осврнули на доста шири скуп контекстно-слободних језика. Уочили смо њихову корисност по питању природних језика, објаснили њихову анализу помоћу потисних аутомата, и на крају направили кратак преглед основних проблема у вези са њима.

Захвалио бих се свом ментору Милошу Арсићу што ми је предложио ову дивну тему и лекторисао рад, Игору Енгију што ме је свих ових година учио LaTeX, и Ноаму Чомском што постоји.

# Литература


[1] И. Спасић. П. Јаничић. *Теорија алгоритама, језика и аутомата.* 2000.

[2] М.Д.Ћирић. Ј.Д. Игњатовић. *Теорија алгоритама, аутомата и језика.* 2012.

[3] *Formal languages and automata theory.* `https://www.gopalancolleges.com/gcem/course-material/computer-science/course-plan/sem-V/formal-languages-and-automata-theory-10CS56.pdf`

[4] А. Милојевић. *Коначни аутомати и регуларни изрази.* 2019.

[5] J. Hopcroft, J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* 1979.

[6] N. Chomsky. *Syntactic Structures.* 1957.