Boštjan Slivnik
Gramatike in avtomati

V pojasnilo in opravičilo hkrati naj takoj na začetku zapišem, da se kljub najboljšim namenom ni mogoče izgoniti kaki matematični formuli tu in tam. Taka je narava avtomatov, formalnih jezikov in gramatik. Zainteresiranega bralca to ne bi smelo motiti.

Avtomati in gramatike imajo za človeka z običajno "klasično" izobrazbo kaj malo skupnega. Slovnico spoznamo najprej pri učenju materinega jezika, kasneje pa nekaj malega tudi pri učenju tujih jezikov. O avtomatih običajno pri pouku ne slišimo kaj dosti, v šoli najpogosteje stojijo v kotu in so namenjeni kuhanju kave.

A nič ne de, na nekaj straneh bomo skušali predstaviti, kaj v teoretičnem računalništvu pomenijo besede avtomat, gramatika, formalni jezik. Preden pa se dokončno zakopljemo v težave, si najprej poglejmo zgodovinski potek razvoja tega področja.

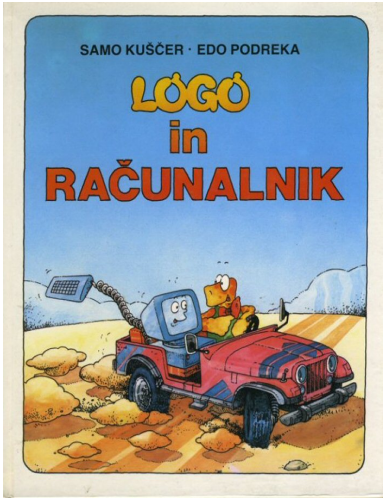
Morda lahko začnemo z Gottfriedom Leibnizem, nemškim matematikom, ki je že v sedemnajstem stoletju uspešno sestavil mehanski računski stroj, nato pa začel razmišljati o stroju, ki bi lahko s preurejanjem simbolov lahko ugotavljal pravilnost matematičnih trditev. Spoznal je, da bi za uporabo takega stroja moral obstajati nek formalen jezik za opis matematičnih trditev, in v nadaljevanju svojega dela se je zato posvetil prav temu. Neka ohlapna zveza med stroji in jeziki, med avtomati in gramatikami, se je očitno rodila že zelo zgodaj.

Precej kasneje, leta 1928, je nemški matematik David Hilbert idejo o ugotavljanju pravilnosti matematičnih trditev formalno opisal kot "odločitveni problem" oziroma po nemško "Entscheidungsproblem". Rešitev tega problema bi bil nek natančen postopek, ki bi za vsako matematično izjavo v logiki prvega reda izračunal, ali je ta izjava vedno resnična ali ne. Vhod v algoritem bi bila torej izjava v logiki prvega reda, izhod pa "da" ali "ne", pač glede na opisano lastnost vhodne izjave.

Bodimo pozorni na to, da je Hilbert zahteval zgolj natančen postopek in ne algortima, saj pojem algoritma leta 1928 sploh še ni bil formalno definiran. To je komaj nekaj let kasneje, leta 1936, uspelo ameriškemu matematiku Alonzu Churchu in angleškemu matematiku Alanu Turingu. Church je definicijo algoritma zasnoval v okviru svojega posebej za rešitev "Entscheidungsproblem"-a razvitega λ-računa, Turing pa na osnovi svojega računskega modela, ki se danes po njem imenuje Turingov stroj. Mimogrede, odličen uvod v računalništvo za osnovnošolce je slikanica Sama Kuščerja, ki predstavi Turingov stroj kot "toaletni računalnik" — zakaj, naj ostane skrivnost kot vzpodbuda, da bralec sam poseže po tej knjigi.

Samo Kuščer, Logo in računalnik,
Velika izobraževalna slikanica,
ilustr. Edo Podreka, Mladinska knjiga, 1987:

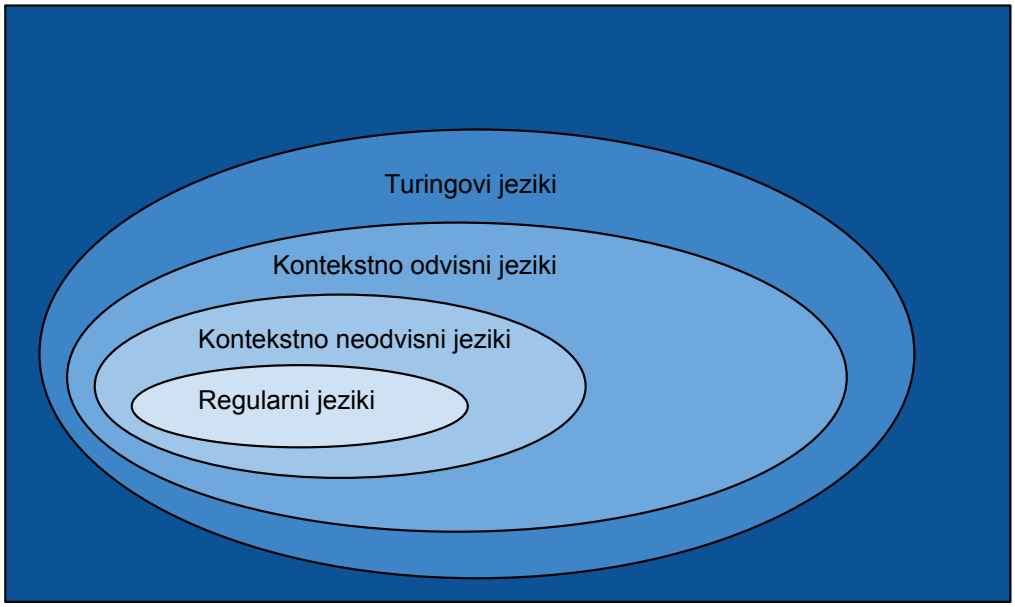
Odlična knjiga za osnovnošolce, ki na lahkoten in zabaven način predstavi delovanje in programiranje današnjih računalnikov.



A hkrati z definicijo algoritma je prišel tudi za tiste čase razmeroma presenetljiv odgovor na "Entscheidungsproblem": rešitve ni! Ko je Hilbert problem definiral, ni niti pomislil na to, da bi morda lahko bil nerešljiv. A Church je dokazal, da ne obstaja algoritem, ki bi ugotovil, ali sta izraza v λ-računu ekvivalentna. Turing je v istem času, le na drugi strani Atlantika, dokazal, da noben Turingov stroj ne more preveriti, ali nek drug Turingov stroj zasnovan tako, da se bo ne glede na vhodne podatke izračun na tem stroju končal v sicer poljubno velikem, a kljub vsemu končnem številu korakov — ta problem je postal znan kot "problem ustavljivosti Turingovega stroja". Oba problema, Churcheva ekvivalenca dveh λ-izrazov in Turingov problem ustavljivosti sta povsem formalno definirana, a nerešljiva z algoritmom, kar pomeni, da postopka za rešitev "Entscheidungsproblem"-a ni.

V formalni dokaz nerešljivosti "Entscheidungsproblem"-a se na tem mestu ne bomo spuščali, saj presega obseg tega besedila in predavanj, ki mu je to besedilo namenjeno. Poleg tega pa oba dokaza, Churchev in Turingov, temeljita na ugotovitvah nemškega matematika Georga Kantorja, ki se mnogim "klasično" izobraženim ljudem na prvi pogled zdijo vsaj nekoliko čudaške. Recimo, da je lihi celih števil enako mnogo kot vseh celih števil skupaj, da je vseh celih števil enako mnogo kot vseh ulomkov, da pa je vseh celih števil manj kot vseh realnih števil. Še več, da obstaja neskončno mnogo različnih neskončnosti. In verjemite, dokaz sploh ni tako zapleten, en sam list papirja zadostuje — le globoko vkoreninjene predstave o "krompirju-u-gajbi" je treba biti pripravljen še enkrat premisliti.

Čeprav sta oba delala povsem neodvisno, se je izkazalo, da sta oba modela računanja, Churchev λ-račun in Turingov stroj, natančno enako močna. Kar lahko izračunamo z enim, lahko tudi z drugim; in obratno. Celo še več: vsi modeli računanja, ki so jih definirali kasneje in ki naj bi bili kar se da močni, hkrati pa bi se jih dalo fizično narediti, so se izkazali za natanko enako močne — vključno z najzmogljivejšim elektronskim superračunalnikom. In pri tem ne pozabimo, da sta Church in Turing definirala vsak svoj modela računanj v času, ko računalnikov še ni bilo.



Slika 1: Chomskyjeva hiearhija formalnih jezikov.

A tako kot sta Church in Turing vsak zase neodvisno definirala pojem algoritma in podala razlago "Entscheidungsproblem"-a, tako se je v drugi polovici dvajsetega stoletja znanje o načinih in mejah formalnega računanja razvijalo po dveh navidez precej ločenih, a v bistvu tesno povezanih poteh.

Po eni poti so kljub dejstvu, da "Entscheidungsproblem" ni rešljiv, ali pa morda prav zaradi tega, so mnogi matematiki in kasneje računalnikarji začeli raziskovati, kakšne probleme je vendarle mogoče reševati s formalnimi postopki računanja oziroma z računalniki. V drugi polovici 20. stoletja je bilo definiranih mnogo modelov računanja različne računske moči, pač glede na trenutne potrebe in možnosti fizične izdelave.

Po drugi poti so se matematiki in jezikoslovci lotili raziskovanja podobnih problemov. Najbolj znan raziskovalec na tem področju je gotovo Noam Chomsky, ki je uvedel idejo univerzalne slovnice, v okviru teh raziskav pa se je razvila teorija formalnih jezikov, ki je ustrezala tako jezikoslovcem kot računalnikarjem.

S stališča teoretičnega računalništva je gotovo najpomembnejši rezultat tega razvoja Chomskyjeva hierarhija formalnih jezikov. Ta definira štiri pomembne razrede formalnih jezikov, kot je to prikazano na sliki 1: regularne jezike, kontekstno neodvisne jezike, kontekstno odvisne jezike in Turingove jezike. Diagram na sliki 1 je predstavljen z množicami: množica vseh regularnih jezikov je prava podmnožica kontekstno neodvisnih jezikov, množica kontekstno neodvisnih jezikov je prava podmnožica kontekstno odvisnih jezikov, in tako naprej. Zunanji pravokotnik na sliki 1 predstavlja univerzalno množico vseh jezikov.

Jezikoslovci so za vsak razred jezikov Chomskyjeve hierarhije definirali razred formalnih gramatik, računalnikarji pa abstraktni matematični model računanja, ki ustreza vsakemu od teh jezikov v Chomskyjevi hierarhiji. Natančneje, Chomskyjevo hierarhijo lahko predstavimo tudi takole:

Turingovi jeziki:
gramatike brez omejitev
Turingovi stroji

Kontekstno odvisni jeziki:
kontekstno odvisne gramatike
linearno omejeni avtomati

Kontekstno neodvisni jeziki:
kontekstno neodvisne gramatike
skladovni avtomati

Regularni jeziki:
linearne gramatike
končni avtomati

V zgornji predstavitvi je za vsak razred najprej naveden razred gramatik, nato pa abstraktni matematični model računanja. Kot je razvidno iz sheme, obstaja zelo lepa povezava med opisom jezika z gramatiko in abstraktnim strojem (avtomatom).

Ne glede na poljudno naravo tega besedila je počasi nujno, da pojme formalnega jezika, gramatike in avtomata določimo natančneje.

Formalni jezik je množica besed končne dolžine, ki so sestavljene iz *simbolov* neke izbrane končne množice simbolov, ki ji rečemo *abeceda*.

Primer 1 Jezik vseh besed dolžine 3 nad abecedo $\{a, b\}$ je množica

$$\{aaa, aab, aba, abb, baa, bab, bba, bbb\} .$$

Jezik vseh besed nad abecedo $\{a, b\}$, ki vsebujejo enako število a-jev in b-jev, je množica

$$\{\varepsilon, ab, ba, aabb, abab, abba, baab, baba, bbaa, \ldots\} ,$$

pri čemer ε opisuje prazno besedo (torej besedo dolžine 0). Slednji jezik je neskončen, zato smo lahko zapisali le nekaj besed tega jezika. Bistroumni bralci bodo razumeli, kakšne so tudi ostale besede tega jezika, ostali pa naj se potrudijo in izpišejo še vse ostale, ki smo jih v zgornjem zapisu izpustili. ———

Malce poenostavljeno povedano je *gramatika* sistem prepisovalnih pravil, s katerimi iz enega niza simbolov izpeljemo drug niz simbolov. Pravilom pravimo *produkcije*, vsaka produkcija pa

ima levo in desno stran: če v nekem nizu simbolov najdemo podniz, ki ustreza levi strani neke produkcije, lahko ta podniz nadomestimo z nizom simbolov na desni strani produkcije. Če lahko v nekem trenutku uporabimo več produkcij, pač glede na trenuten niz simbolov in leve strani produkcij, potem lahko svobodno izberemo produkcijo in jo uporabimo.

Jezik gramatike sestavljajo vse besede, ki se jih da z ravnokar opisanim načinom uporabe produkcij izpeljati iz nekega vnaprej določenega simbola gramatike.

Primer 2 Vzemimo abecedo $\{a, b\}$ in gramatiko s petimi produkcijami

$$S \longrightarrow \varepsilon \quad S \longrightarrow aAS\,, \quad aA \longrightarrow Aa\,, \quad Aa \longrightarrow bAb \quad \text{in} \quad A \longrightarrow bAb\,.$$

Ob dogovoru, da je S začetni simbol te gramatike, gramatika omogoča mnogo različnih izpeljav, med drugim naslednje tri:

$$S \Longrightarrow \varepsilon$$

$$S \Longrightarrow aAS \Longrightarrow AaS \Longrightarrow baS \Longrightarrow ba$$

$$S \Longrightarrow aAS \Longrightarrow aAaAS \Longrightarrow aAaA \Longrightarrow bAbA \Longrightarrow bbbA \Longrightarrow bbbb$$

Na osnovi teh treh izpeljav lahko zaključimo, da besede ε, ba in $bbbb$ pripadajo jeziku gornje gramatike. Pozorni bralec naj preveri, zakaj so to veljavne izpeljave, in nato še sam zapiše kakšno veljavno izpeljavo. ———————————————————

Gramatika torej *generira* jezik: z vsako izpeljavo, pri kateri uporabljamo produkcije gramatike, izpeljemo neko besedo jezika. Kot smo povedali že pri opisu Chomskyjeve hierarhije, obstaja več vrst gramatik:

1. *Gramatike brez omejitev:* pri teh gramatikah sta lahko na levi in na desni strani produkcije povsem poljubna niz simbolov, le niz na levi strani ne sme biti prazen.

2. *Kontekstno odvisne gramatike:* pri teh gramatikah mora biti niz na desni strani produkcije vedno vsaj tako dolg kot niz na levi strani produkcije.

   Ime "kontekstno odvisne gramatike" izvira iz dejstva, da se da vsako tako gramatiko pretvoriti v njej ekvivalentno gramatiko (ekvivalentno v tem smislu, da generira isti jezik), pri kateri so vse produckije oblike $xAy \longrightarrow x\omega y$, pri čemer sta x in y niza simbolov jezika generiranega jezika, ω pa je niz, ki ga lahko sestavljajo poljubni simboli gramatike (torej tako tisti, ki so v abecedi jezika, kot tisti, ki niso). Na tak način lahko simbol A zamenjamo z nizom ω le v natančno določenem kontekstu, ki ga določata niza x in y.

3. *Kontekstno neodvisne gramatike:* pri teh gramatikah mora biti na levi strani produkcije vedno natanko en simbol gramatike, ki pa ne sme biti simbol abecede jezika, na desni strani produkcije pa je lahko poljuben niz simbolov gramatike.

Ime "kontekstno neodvisna gramatika" izvira iz dejstva, da so vse produkcije oblike $A \longrightarrow \omega$. To pomeni, da sta niza x in y (glede na obliko produkcij kontekstno odvisnih gramatik) prazna in da uporaba produkcij torej ni odvisna od konteksta.

4. *Linearne gramatike:* pri teh gramatikah mora biti na levi strani produkcije vedno natanko en simbol gramatike, ki pa ne sme biti simbol abecede jezika, na desni strani produkcije pa je lahko niz simbolov abecede jezika, ki ima lahko (ali pa ne) na skrajni desni (levi) natanko en simbol gramatike, ki ni simbol abecede jezika.

Različne abstraktne modele računanja, ki se pojavljajo v Chomskyjevi hierarhiji, je precej težje predstaviti enotno (tako kot smo ravnokar predstavili gramatike). Linearno omejene avotmate in Turingove stroje bomo pustili za kakšno drugo priložnost, na tem mustu bomo predstavili le končne in skladovne avtomate:

1. *Končni avtomat* je model, ki ima končno število stanj, med katerimi prehaja glede na posamezne simbole trenutne vhodne besede. Med stanji je natančno eno stanje začetno, vsaj eno stanje pa mora biti označeno kot končno stanje.

   Za vsako podano besedo končni avtomat začne v začetnem stanju, nato pa v končnem številu korakov odgovori na vprašanje, ali ta vhodna beseda pripada jeziku tega avtomata. Na vsakem koraku avtomat na podlagi trenutnega stanja in trenutnega vhodnega simbola preide v novo stanje, trenutni vhodne simbol pa zavrže. Prehodi med stanji so vedno določeni glede na simbole vhodne besede, le v določenih korakih obstaja možnost, da vhodnega simbola za odločitev o prehodu iz enega stanje v drugega ne potrebuje in ga zato pri izvedbi koraka tudi ne zavrže (tak prehod je označen z ε). Natančneje, in morda bolj jasno, bomo delovanje končnega avtomata pokazali s primerom.

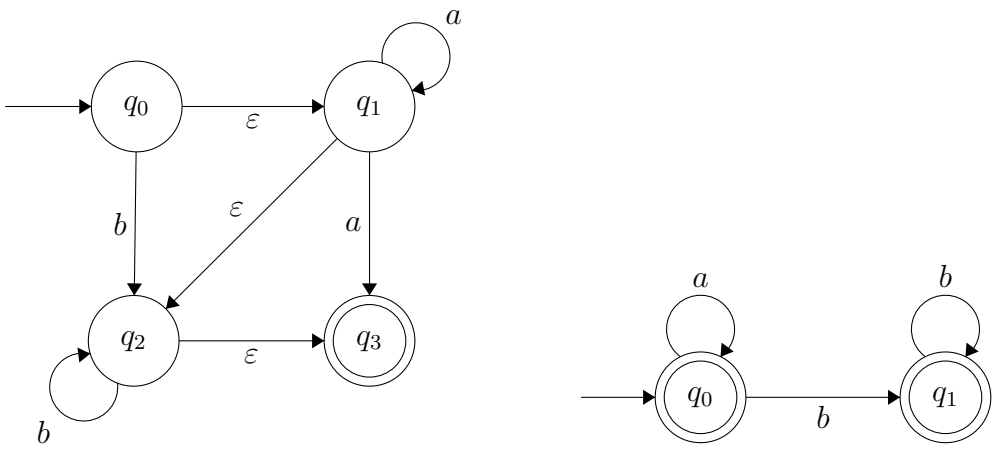
2. *Skladovni avtomat* je model, ki ima poleg končnega števila stanj še neskončno dol sklad, na katerega si lahko med izračunom shranjuje za sklad posebej določene simbole in si na ta način pomaga pri izračunu — sklad na nek način uporablja kot papir za "pomožne račune".

   Skladovni avtomat deluje podobno kot končni avtomat, le da je odločitev o prehodu v naslednje stanje odvisna tudi od simbola na vrhu sklada. Na vsakem koraku torej poleg trenutnega stanja in trenutnega vhodnega simbola za prehod v novo stanje uporabi tudi simbol na vrhu sklada, ki pa ga zato pri vsakem prehodu na vrhu sklada nadomesti z novim nizom skladovnih simbolov (ta niz je lahko seveda tudi prazen).

Da bi si bolje predstavljali prva dva nivoja Chomskyjeve hierarhije, predvsem pa delovanje končnih avtomatov, si oglejmo dva primera.

Primer 3 Najprej si oglejmo primer regularnega jezika. Vzemimo jezik

$$L_{a^n b^m} = \{a^n b^m \mid n \geq 0 \wedge m \geq 0\} \quad ,$$



Slika 2: Nedeterministični in deterministični končni avtomat za jezik $L_{a^n b^m}$.

ki ga sestavljajo vse besede, ki so sestavljene iz poljubnega števila a-jev in b-jev, pri čemer morajo vsi a-ji stati pred vsemi b-ji. Torej:

$$L_{a^n b^m} = \{\varepsilon, a, b, aa, ab, bb, aaa, aab, abb, bbb, \ldots\} \quad .$$

Ker je ta jezik regularen, zanj obstajata vsaj dva različna avtomata, ki sta prikazana na sliki 2. Pri obeh avtomatih se držimo dogovora, da začetno stanje označimo s prazno puščico, ki ne izvira iz nobenega drugega stanja, končna stanja pa označimo z dvojnim krogcem.

Delovanje prvega (na sliki 2 levo) lahko razložimo s primerom, ko mu na vhod damo besedo abb:

> Avtomat začne v svojem začentem stanju, torej v stanju q_0. Prvi simbol vhodne besede je a, zato je edina možnost, da avtomat brez uporabe tega simbola preide v stanje q_1. V stanju q_1 ima avtomat kar tri možnosti: (1) lahko uporabi simbol a in preide (ostane) v q_1, (2) lahko uporabi simbol a in preide v q_3 ali pa (3) simbola a ne uporabi in preide v q_2.
>
> Če izberemo prvo možnost, simbol a umaknemo, nov trenutni simbol postane (prvi) b, zato moramo v naslednjem koraku najprej opraviti prehod v stanje q_2 brez uporabe vhodnega simbola. Nato dvakrat opravimo prehod v q_2 po prvem in drugem simbolu b, nazadnje pa še prehod brez vhodnega simbola v končno stanje q_3. Ker smo končno stanje dosegli, zaključimo, da avtomat sprejme besedo abb oziroma da beseda pripada jeziku tega avtomata.
>
> Če pa bi izbrali drugo možnost, namreč prehod po vhodnem simbolu a v stanje q_3, procesa ne bi mogli nadaljevati, saj iz stanja q_3 ni prehoda po simbolu b, ki sledi a-ju. Ne glede na to, da smo dosegli končno stanje, na osnovi druge možnosti ne moremo zaključiti, da beseda abb priprada jeziku tega avtomata, saj avtomat ni obdelal celotne besede.

Opisana postopka lahko zgoščeno opišemo z naslednjima izpeljavama:

$$q_0abb \Longrightarrow q_1abb \Longrightarrow aq_1bb \Longrightarrow aq_2bb \Longrightarrow abq_2b \Longrightarrow abbq_3 \Longrightarrow abb \in L_{a^nb^m}$$

$$q_0abb \Longrightarrow q_1abb \Longrightarrow aq_3bb \Longrightarrow ???$$

Posebej bodimo pozorni na to, da slednja izpeljava ne zagotavlja, da beseda abb ni v jeziku $L_{a^nb^m}$ — je le zgrešen poskus, s katerim nam ni uspelo pokazati, da ta beseda ni v tem jeziku.

Avtomat na sliki 2 levo zahteva, da v določenih korakih izpeljave preprosto uganemo pravo pot — takemu končnemu avtomatu pravimo *nedeterministični končni avtomat*. Kot smo videli ravnokar, tak avtomat za nekatere besede dopušča več različnih začetkov izpeljave, od katerih nas vsi ne pripeljejo do končnega stanja. A velja celo še več, za nekatere besede obstaja celo več različnih izpeljav:

$$q_0aa \Longrightarrow q_1aa \Longrightarrow aq_1a \Longrightarrow aaq_3 \Longrightarrow aa \in L_{a^nb^m}$$

$$q_0aa \Longrightarrow q_1aa \Longrightarrow aq_1a \Longrightarrow aaq_1 \Longrightarrow aaq_2 \Longrightarrow aaq_3 \Longrightarrow aa \in L_{a^nb^m}$$

Da bi se izognili poskušanju pri preverjanju pripadnosti besede jeziku avtomata, lahko nedeterministični avtomat na live strani slike pretvorimo v *deterministični končni avtomat*, ki sprejema isti jezik, a v nobenem trenutku ne dopušča proste izbire v naslednjem koraku: iz vsakega stanja je za vsak vhodni simbol na voljo kvečjemu en prehod v neko novo stanje. Bralec naj za vajo skuša utemeljiti, da oba avtomata na sliki 2 res sprejemata isti jezik.

A ker je jezik $L_{a^nb^m}$ regularen, zanj obstajata desno linearna gramatika

$$S \longrightarrow A\,, \quad A \longrightarrow aA \,|\, B\,, \quad B \longrightarrow bB \,|\, \varepsilon$$

in levo linearna gramatika

$$S \longrightarrow B\,, \quad A \longrightarrow Aa \,|\, \varepsilon\,, \quad B \longrightarrow Bb \,|\, A\,.$$

Takoj lahko zapišemo izpeljavi

$$S \Longrightarrow A \Longrightarrow aA \Longrightarrow aB \Longrightarrow abB \Longrightarrow abbB \Longrightarrow abb$$

$$S \Longrightarrow B \Longrightarrow Bb \Longrightarrow Bbb \Longrightarrow Abb \Longrightarrow Aabb \Longrightarrow abb$$

ostale pa naj si bralec za vajo izpiše sam. ———————————————————

Primer 4 En nivo višje v Chomskyjevi hierarhiji so kontekstno neodvisni jeziki. Vzemimo jezik

$$L_{a^nb^n} = \{a^nb^n \mid n \geq 0\} \quad ,$$

ki je zelo podoben jeziku L_1 iz primera 3, le da morajo biti tu v vsaki besedi natančno enako število a-jev in b-jev. Torej:

$$L_{a^nb^n} = \{\varepsilon, ab, aabb, aaabbb, aaaabbbb, \ldots\} \quad .$$

Ker je jezik konteksnto neodvisen, zanj obstaja kontekstno neodvisna gramatika

$$S \longrightarrow aSb \,|\, \varepsilon \,.$$

Da ta gramatika res generira jezik $L_{a^n b^n}$, nam na prvi pogled zadostuje vzorec prvih nekaj izpeljav

$$\begin{aligned}
&S \Longrightarrow \varepsilon \\
&S \Longrightarrow aSb \Longrightarrow ab \\
&S \Longrightarrow aSb \Longrightarrow aaSbb \Longrightarrow aabb \\
&S \Longrightarrow aSb \Longrightarrow aaSbb \Longrightarrow aaaSbbb \Longrightarrow aaabbb \\
&\quad \vdots
\end{aligned}$$

vse preostale pa si lahko nejeverni bralec izpiše sam. ———————————————

Kot smo že povedali, so vsi regularni jeziki tudi kontekstno neodvisni, obratno pa seveda ni res. Kdor ne verjame, naj poskusi sestaviti končni avtomat za jezik $L_{a^n b^n}$ iz primera 4.

Na tem mestu moramo omeniti še *regularne izraze*, ki predstavljajo izredno učinkovit opis regularnih jezikov. Vsak regularni jezik lahko opišemo z nekim regularnim izrazom, ki ga dobimo na osnovi naslednjih treh pravil:

1. Regularni izrazi $\emptyset$, ε in a, pri čemer je a poljuben simbol abecede, opisujejo regularne jezike $\{\}$, $\{\varepsilon\}$ in $\{a\}$, zaporedoma.

2. Naj bosta r_1 in r_2 regularna izraza. Tedaj regularni izraz $r_1|r_2$ opisuje unijo jezikov regularnih izrazov r_1 in r_2, regularni izraz $r_1 r_2$ pa stik jezikov regularnih izrazov r_1 in r_2.

   V jeziku regularnega izraza $r_1|r_2$ je torej vsaka beseda, ki je v jeziku vsaj enega od obeh jezikov izrazov r_1 in r_2, v jeziku regularnega izraza $r_1 r_2$ pa je vsaka beseda, ki jo dobimo tako, da staknemo eno besedo iz jezika prvega regularnega izraza z besedo iz jezika drugega regularnega izraza.

3. Naj bo r regularni izraz. Tedaj jeziku regularnega izraza r^* pripadajo besede, ki nastanejo tako, da iz jezika regularnega izraza r vzamemo poljubno končno mnogo besed (lahko tudi nič; pri tem izboru se besede lahko tudi ponavljajo) in jih staknemo skupaj.

Dogovorimo se še, da operator * veže najmočneje, operator $|$ pa najšibkeje. Če želimo drugače, pač uporabimo oklepaje.

Primer 5 Ponovno izberimo abecedo $\{a, b\}$. Regularni jezik $L_{a^n b^m}$ lahko opišemo z regularnim izrazom

$$a^* b^* \, .$$

Zakaj? Izraz a opisuje jezik $\{a\}$, zato regularni izraz a^* opisuje jezik $\{\varepsilon, a, aa, aaa, aaaa, \ldots\}$. Po analogiji izraz b^* opisuje jezik $\{\varepsilon, b, bb, bbb, bbbb, \ldots\}$, stik teh dveh jezikov, kot ga opisuje regularni izraz $a^* b^*$ pa zato vsebuje besede, ki jih dobimo tako, da vzamemo neko besedo iz jezika $\{\varepsilon, a, aa, aaa, aaaa, \ldots\}$ in jo staknemo z neko besedo jezika $\{\varepsilon, b, bb, bbb, bbbb, \ldots\}$. Dobimo recimo besede $\varepsilon = \varepsilon\ \varepsilon$, $a = a\ \varepsilon$, $ab = a\ b$, $aaab = aaa\ b$, $\ldots$

Zapišimo še nekaj regularnih izrazov in jezikov, ki jih ti izrazi opisujejo:

$$\begin{aligned} a^* b a^* \quad &\ldots \quad \{b, aba, abaa, aaba, aabaa, \ldots\} \\ a^* (ba)^* \quad &\ldots \quad \{\varepsilon, a, ba, aa, aaba, ababa, baba, \ldots\} \\ (abb^* a)|bb \quad &\ldots \quad \{bb, aba, abba, abbba, \ldots\} \end{aligned}$$

Bralec naj se potrudi in poišče razlago sam.

Da pa ne bi ostalo vse skupaj samo pri teoriji, se vprašajmo, kje lahko formalne jezike, predvsem regularne in kontekstno neodvisne, tudi zares uporabimo.

Običajnemu programerju so gotovo najbližji regularni izrazi, saj jih pri programiranju zelo pogosto uporabljamo. Ne samo, da vsi uporabni urejevalniki besedil (ne, MS Word ne sodi v to skupino) omogočajo iskanje besed z uporabo regularnih izrazov, tudi v programih samih uporabljamo regularne izraze za iskanje podatkov v besedilih in podobnih zbirkah podatkov. Danes praktični ni več nobenega jezika, ki bi ne podpiral dela z regularnimi izrazi — bodisi so regularni izrazi del jezika bodisi del standardne knjižnice.

Primer 6 Recimo, da iščemo v neki datoteki vse vrstice, ki jih vsebujejo katerokoli letnico med 1980 in 2000. To lahko opišemo z regularnim izrazom

$$(19(9|8)(0|1|2|3|4|5|6|7|8|9))|2000$$

na računalniku pa z ukazom

```
$ grep "(19[89][0-9])|(2000)" datoteka
```

Na računalniku so regularni izrazi zapisani nekoliko drugače, saj so pri programiranju obogateni z večjim številom operatorjev. Med posameznimi izvedbami regularnih izrazov obstajajo določene razlike, zato je v vsakem primeru bolje pogledati na Google, kaj nam posamezno orodje ali posamezni programski jezik nudi v zvezi z regularnimi izrazi.

```
statement
        : labeled_statement
        | compound_statement
        | expression_statement
        | selection_statement
        | iteration_statement
        | jump_statement
        ;

labeled_statement
        : IDENTIFIER ':' statement
        | CASE constant_expression ':' statement
        | DEFAULT ':' statement
        ;

compound_statement
        : '{' '}'
        | '{' statement_list '}'
        | '{' declaration_list '}'
        | '{' declaration_list statement_list '}'
        ;

declaration_list
        : declaration
        | declaration_list declaration
        ;

statement_list
        : statement
        | statement_list statement
        ;

expression_statement
        : ';'
        | expression ';'
        ;

selection_statement
        : IF '(' expression ')' statement
        | IF '(' expression ')' statement ELSE statement
        | SWITCH '(' expression ')' statement
        ;

iteration_statement
        : WHILE '(' expression ')' statement
        | DO statement WHILE '(' expression ')' ';'
        | FOR '(' expression_statement expression_statement ')' statement
        | FOR '(' expression_statement expression_statement expression ')' statement
        ;

jump_statement
        : GOTO IDENTIFIER ';'
        | CONTINUE ';'
        | BREAK ';'
        | RETURN ';'
        | RETURN expression ';'
        ;
```

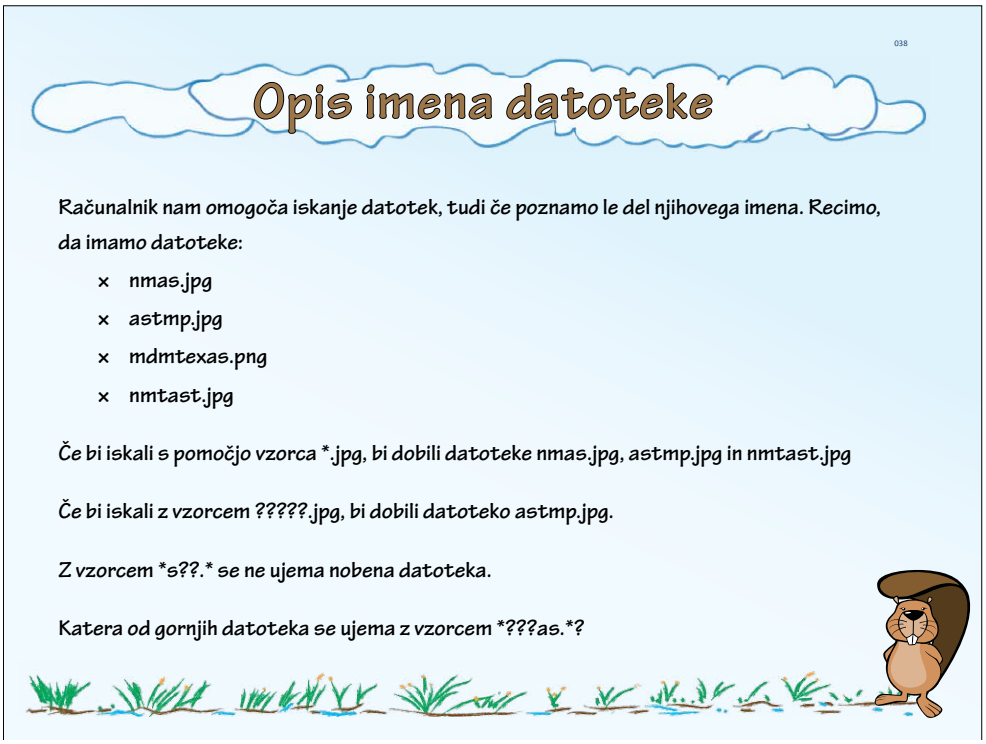
Slika 3: Del kontekstno neodvisna gramatike jezika C.

Končni avtomati so neposredno uporabni v računalniških komunikacijah, saj z njimi opisujemo protokole, uporabljamo jih tako pri snovanju logičnih vezij kot pri načrtovanju telefonskih central, pri sestavljanju modelov delovanja posameznih enot programa, ...

Kar se tiče kontekstno neodvisnih jezikov in gramatik, lahko zapišemo, se morda ne pojavljajo na toliko različnih področij računalništva, so pa zato povsem nepogrešljive za opis sintakse programskih jezikov in s prevajanjem programskih jezikov. A to je veliko: praktično ni področja računalništva, ki ne bi temeljilo na programiranju, za programiranje pa seveda potrebujemo prevajalnike.

Primer 7 *Kontekstno neodvisna gramatika za opis poljubnega programskega jezika vsebuje preveč produkcij, da bi jo lahko predstavili v celoti. Samo za občutek so na sliki 3 predstavljene produkcije, s katerimi v gramatiki programskega jezika C opišemo kontrolne stavke.*

"Simbolične majice" so tipičen primer naloge, kjer je podan vzorec, za katerega moramo ugotoviti, kateri nizi mu ustrezajo. Vzorec je podan s poenostavljenim regularnim izrazom, mali Bober pa pri reševanje take naloge gotovo ne bom imel nič več težav kot odrasel Bober.



Naloga “Opis imena datoteke” je lep primer naloge, pri kateri si lahko reševalec pomaga z znanjem regularnih izrazov. Čeprav so v nalogi uporabljeni precej enostavnejši prijemi kot v pravih regularnih izrazih (le dva enostavna operatorja), pa že samo dejstvo, da vemo za obstoj operatorjev v regularnih izrazih, pomaga pri razvozlanju in rešitvi te naloge.

Ključ za pravilno rešitev naloge je namreč v tem, da ugotovimo, da

1. zvezdica (znak `*`) predstavlja poljubno zaporedje črk,
2. vprašaj (znak `?`) pa poljubno eno črko.

Od tod naprej je rešitev enostavna:

1. Vzorec `*???as.*?` zahteva vsaj tri znake pred `a`-jem (zaradi treh vprašajev), a ime `nmas.jpg` ima le dva (namreč `nm`) in zato ne more biti rešitev naloge. Isto velja za ime `astmp.jpg`, ki pred `a`-jem nima sploh nobenega znaka.
2. Ime `nmtast.jpg` prav tako ne more biti rešitev naloge, saj vzorec `*???as.*?` zahteva piko za znakom `s`, v imenu `nmtast.jpg` pa piki sledi znak `t`.
3. Ostane še ime `mdmtexas.png`. Vzorcu `*???as.*?` se ime `mdmtexas.png` prilagodi tako, da prva zvezdica vzorca predstavlja `mdm`, trije vprašaji za prvo zvezdico predstavljajo `tex`, druga zvezdica predstavlja `pn` in zadnji vprašaj predstavlja `g`. Torej je ime `mdmtexas.png` rešitev naloge.

Osnovno vprašanje pri razlagi te naloge je naslednje: kako naj otrok ugotovi, da zvezdica in vprašaj delujeta kot operatorja (četudi tega ne poimenuje tako). Bolje je začeti z razlago regularnih izrazov, kot je opisano zgoraj, nato pa otroku prepustiti, da sam odkrije, da so v tej nalogi regularni izrazi pač nekoliko drugačni.

Kot zanimivost lahko mimogrede zapišemo, da je naloga lahko današnjim osnovnošolcem videti zapletena, čisto drugače pa bi jo sprejeli njihovi vrstniki pred dobrim desetletjem. V tistem času so namreč namesto oken in najrazličnejših brskalnikov morali uporabljati zgolj ukazno vrstico in ukaze, ki jih današnji osnovnošolci sploh ne poznajo. A če so recimo želeli prenesti skupino datotek iz ene diskete na drugo (ja, le povejmo jim, kako je bilo to takrat), so si to lahko olajšali le, če so znali uporabljati natančno tak opis datotek, kot je uporabljen v tej nalogi. Ali povedano drugače, `COMMAND.COM` je bilo praktično vse, kar so imeli na voljo, da so brkljali po disketah in diskih (ne, USB "ključkov" še ni bilo).

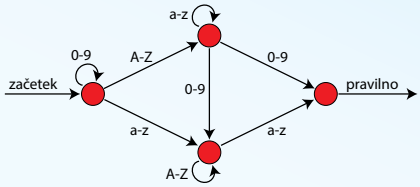
091

Preverjanje gesel

Bobrčki v neki šoli si morajo izbrati gesla za dostop do računalnikov. Da bi bilo geslo varno, mora prestati testni stroj, ki deluje takole: začnemo pri krogcu, ki ga označuje puščica »začetek«. Nato po vrsti jemljemo znake gesla (sestavljeno mora biti iz številk in črk), ki povedo, po kateri poti moramo iti z vsakega krogca. Geslo je pravilno, če končamo na krogcu, iz katerega vodi puščica »pravilno«.

Eno izmed naslednjih gesel ni pravilno? Katero?

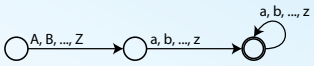
- × 123aNNa
- × Peter3ABCd
- × 2010Bober4EVEr
- × bENN0Zzz



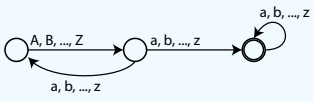
031

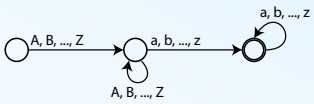
Uporabniška imena

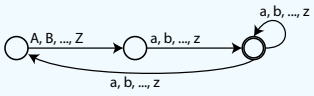
Ime se začne z veliko črko, ki mu sledi ena ali več malih črk. To opišemo takole:

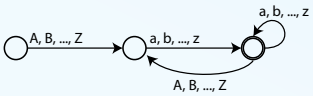


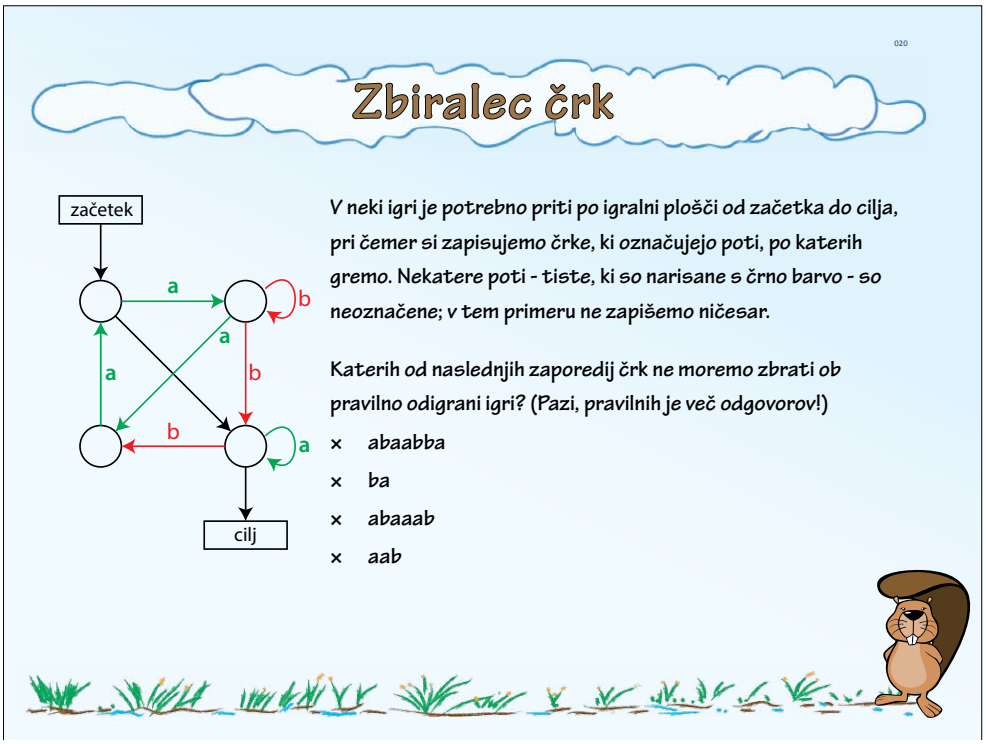
Shemo bi radi razširili tako, da bi lahko z njo opisali tudi osebe z več imeni; pisali bi jih brez presledkov, npr. FrancJožef ali MarijaTerezija. Katera od spodnjih shem je pravilna?



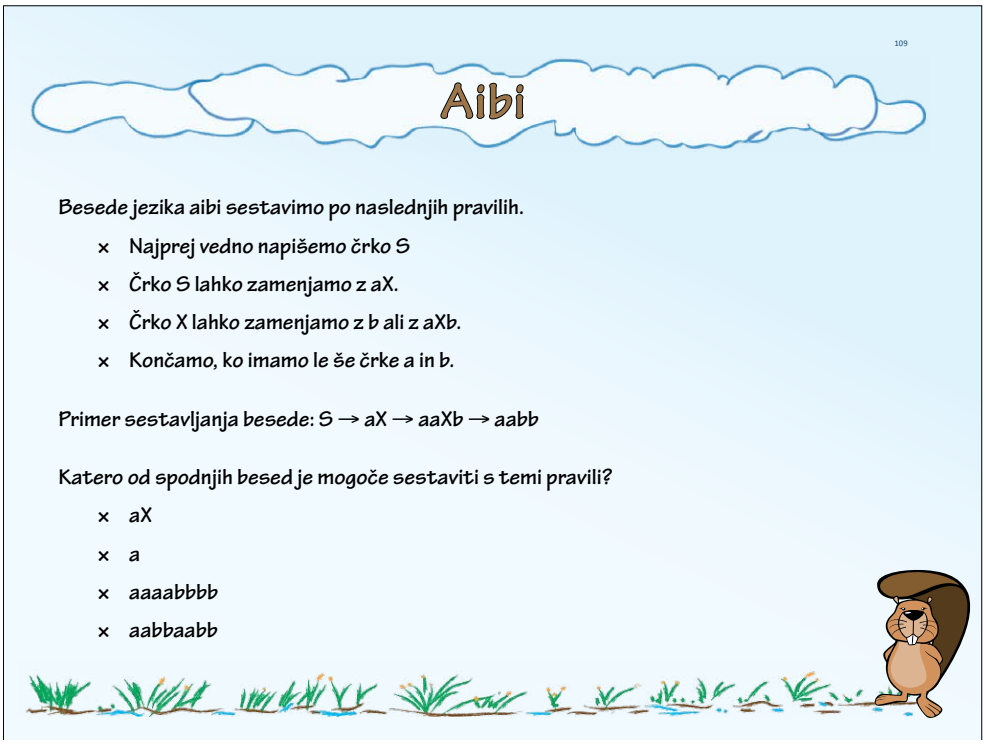








Vse tri naloge, “Preverjanje gesel”, “Uporabniška imena” in “Zbiralec črk”, so povsem klišejske naloge, ki temeljijo na ročnem izračunu delovanja končnega avtomata, kot je bilo opisano zgoraj.



"Aibi" je primer suhoparne naloge, kjer je podana kontekstno neodvisna gramatika, reševalec pa mora za podane besede poskusiti najti izpeljavo ali pa dokazati, da izpeljava ne obstaja.