FAKULTA SOCIÁLNYCH A EKONOMICKÝCH VIED
KOMENSKÉHO UNIVEZITA V BRATISLAVE

Vladimír Kvasnička
Jiří Pospíchal

Informatika pre sociálne vedy

Júl 2003

Obsah

Úvod

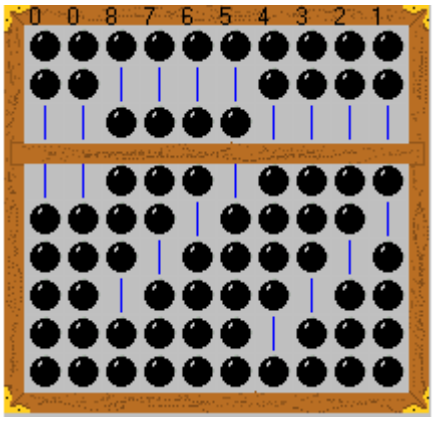
Cieľom tohto učebného textu je výučba informatiky pre potreby sociálnych vied. Pri písaní tohto textu stáli sme pre dilemou, či zredukovať význam informatiky v týchto vedách len na znalosť programového balíka MS OFFICE, ktorý, aj keď je dôležitý pre profesionalizáciu niektorých činností v sociálnych vedách (akými sú, písanie článkov, správ, prezentácia výsledkov, atď.), predsa len netvorí ani len zlomok možností modernej informatiky v sociálnych vedách. V tomto učebnom texte snažíme sa jednoduchým prístupom prezentovať tie teoretické základy informatiky a jej moderné aplikácie a rozšírenia, ktoré majú aj určitý význam v sociálnych vedách. Tak napríklad, skúmanie ľudskej mysle kognitívnou psychológiou (alebo všeobecnejšie, kognitívnou vedou) je už ťažko predstaviteľné bez použitia metafory mozgu ako počítača, ktorý spracováva naše mentálne reprezentácie pomocou formálnych pravidiel. Podobne, počítačové simulácie ekonomických a sociálnych systémov pomocou techník pochádzajúcich z informatiky (multiagentové systémy) tvoria v súčasnosti podklad pre racionálnu argumentáciu pri vysvetľovaní zložitých sociálnych a ekonomických javov, akými sú kooperácia, význam spoločenských noriem, vznik sociálnych štruktúr, atď. Možno konštatovať, že moderná informatika sa stáva mostom medzi prírodnými vedami a sociálnymi vedami, pomocou svojich špecifických techník je schopná študovať procesy, ktoré boli donedávna vyhradené len sociálnym vedám. Veríme, že tento učebný text bude užitočný pre študentov Fakulty sociálnych a ekonomických vied UK, a že nájdu v ňom pomocníka a informátora o širokých možnostiach modernej informatiky pri štúdiu sociálnych vied.

V Bratislave, júl 2003.

Vladimír Kvasnička a Jiří Pospíchal

1 Dejiny počítania

Za prvého moderného realizovaného predchodcu počítača možno považovať rôzne jednoduché pomôcky uľahčujúce počítanie alebo záznam číselnej informácie. Anglické slovo „*calculator*“ pochádza z latinského „*calculus*“ označujúceho malé kamienky používané nielen na hru, ale taktiež aj ako pomôcka pri vykonávaní jednoduchých číselných operácií; jeden kamienok predstavoval napr. jeden kus dobytka. Pomôckou rovnakej "zložitosti" boli aj paličky so zárezmi = rováše[1]. Od zárezov a kamienkov sa prešlo k sofistikovanejším "zariadeniam", ako je abakus (počítadlo), pozri obr. 1.1, pochádzajúci azda z Babylónie a bežne používaný v šiestom storočí pred naším letopočtom v Grécku (zmienený napr. Herodotom), ktorý je doteraz rozšírený v Rusku a na ďalekom východe. Abakus reprezentuje stav výpočtu pozíciou korálok na drôtoch v stĺpcoch (alebo predtým v žliabkoch na drevenej doske)[2]. Pripomína detské počítadlo, pomocou ktorého ale dokážu jeho majstri urobiť veľmi rýchlo aj zložité výpočty (napr. odmocniny). "Program" je v tomto prípade schovaný v mysli týchto expertov, ktorí nad takýmto naučeným postupom nemusia rozmýšľať.



Obrázok 1.1. Znázornenie abakusu s nastavením čísla 87 654 321.

Ľudia začali s postupom času vymýšľať zložitejšie pomôcky a stroje na uľahčenie práce, nech už fyzickej alebo aj duševnej pri výpočtoch. Stroj ale nemusí byť nevyhnutne technickým zariadením podliehajúcim výlučne fyzikálnym zákonom. Keď budeme uvažovať o organizácii práce pri stavbe pyramíd, vojenskú alebo štátnu mašinériu, môžeme povedať, že všeobecným základom strojov je usporiadanie, organizácia malých jednotiek a postupnosť ich práce (nech už sú jednotkami ľudia, ozubené kolieska, alebo tranzistory) [XKel]. Vývoj techniky išiel postupom nahradzovania ľudí usporiadaných v organizačných jednotkách strojmi.

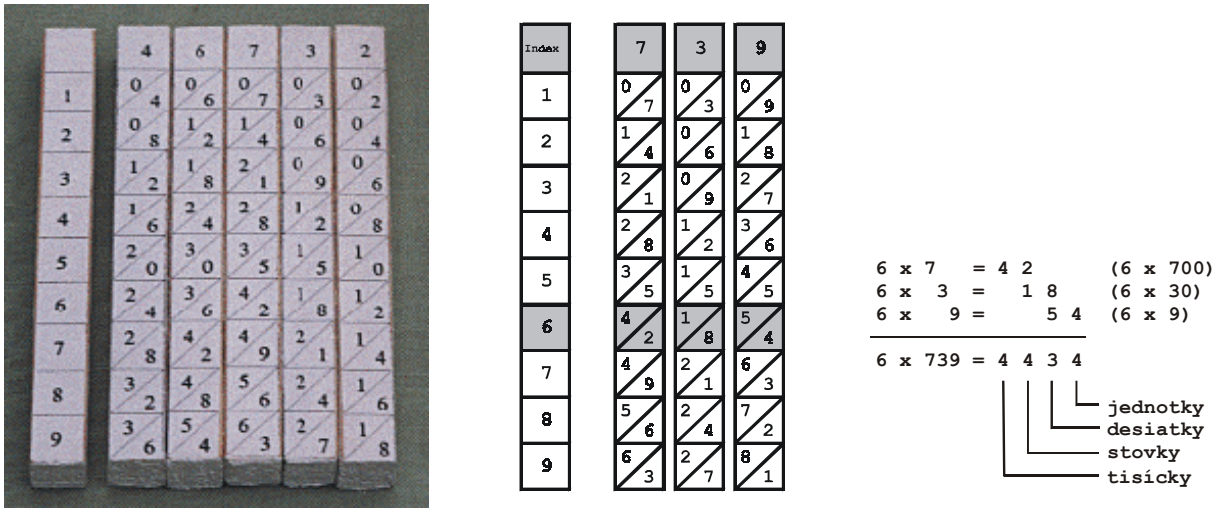
[1] Špeciálne zaujímavá je v tomto ohľade cez 20000 rokov stará vlčia kosť s 55 zárezmi usporiadanými v skupinách po piatich (nález r. 1937, Věstonice na Morave).

[2] Korálky sa považujú za započítané, keď sú posunuté k prostrednému vodorovnému trámiku. Stĺpec napravo reprezentuje jednotky, susediaci stĺpec vľavo desiatky, ďalší stovky, atď. Keď je na dolnej úrovni napočítaných 5 korálok, výsledok je "prenesený" na hornú úroveň, po tom, čo sú obidve korálky na hornej úrovni započítané, výsledok (10) je prenesený do ľavého susedného stĺpca.

Ako prvá ale musela prísť myšlienka, nech aj fantastická. Už v Ílias čítame o zlatých slúžkach "Vedeli rozmýšľať, vravieť, dokonca mali aj silu". V Paríži navrhol Reymund Lullus (1235-1315) "stroj pravdy" na základe presvedčenia o formalizovateľnosti logických operácií. Hovorí sa, že Albert Veliký spolu so svojím žiakom Tomášom Akvinským zhotovili pomocou alchýmie umelého človeka - androida. V 14. storočí sa o Eliovi Chelmovi rozšírilo, že vytvoril z hliny golema, ktorého vraj v 16 storočí oživil pražský rabín Jehuda Lov ben Becalél. Chýrny lekár a alchymista Theophrastus Bombastus von Hohenheim zvaný Paracelsus, ktorý tiež r. 1537 navštívil Bratislavu, opísal postup, ako v sklenenej nádobke zo spermy získať homunkula.

Vyššie uvedené fantastické predstavy a senzácie potom našli svoj odraz v strojoch, keď dostatočne pokročila tak technológia v mechanike, ako aj postupy pri uľahčení výpočtov v matematike.

Heron Alexandrijský v prvom storočí pnl. vyrábal hydraulicky poháňané mechanizmy. Programové riadenie automatov valčekom s klinčekmi sa používalo v 18. storočí v hračkách a orchestriónoch. Keď hovoríme o automatoch, treba spomenúť aj šachový automat Johanna Wolfganga Kempelena, bratislavského rodáka, ktorý sa okrem svojich skutočných a veľmi užitočných vynálezov preslávil aj šachovým automatom, vytvoreným v Bratislave a predstaveným r. 1770 Márii Terézii vo Viedni. Predpokladá sa, že v stroji bol skrytý človek malého vzrastu, ovládajúci šach.



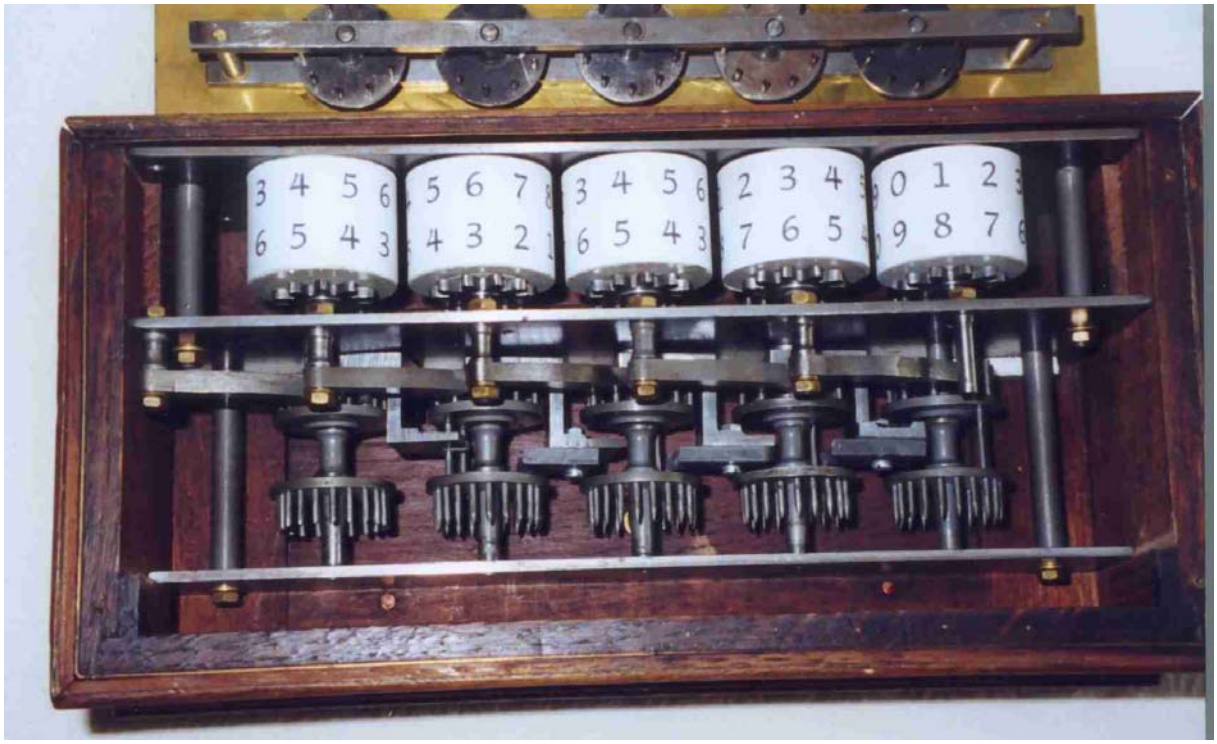
Obrázok 1.2. Znázornenie "Napierových kostí" z r. 1617 a ich schematického zobrazenia potrebného k vysvetleniu násobenia 6×739. Násobenie môžete zjednodušiť na jednoduché sčítanie. Každá kosť obsahuje časť malej násobilky, násobky čísla hore s číslami od 0 po 9. Násobenie 6 x 5 môže byť urobené kosťou s vrchným číslom 5, keď sa pozrieme na šiesty riadok. Číslo riadku môžeme pozrieť na "indexovej kosti". Násobenie čísla (napr. 6) s veľkým číslom (napr. 739) môže byť ľahko urobené pomocou kostí 7, 3 a 9 umiestnených vedľa seba a s indexovou kosťou na prvom mieste. Rad 6 nám dáva výsledok násobku (6 x 739), keď pridáme jednotky, desiatky, stovky a tisícky. Pomocou "/" na kostiach môžete vidieť, ktoré čísla majú byť sčítané.

K významnejšiemu uľahčeniu výpočtov došlo až v priebehu 15. a 16. storočia. V tejto dobe technických objavov a zámorských obchodných ciest bolo potrebné opakovane robiť veľa výpočtov v navigácii a tiež výpočtov týkajúcich sa pohybu planét. V 17 storočí škótsky matematik John Napier vymyslel "Napierove kosti", pomôcku na násobenie (pozri obr. 1.2). Okrem toho vynašiel metódu, akou sa obtiažne procesy násobenia, delenia a výpočtu odmocnín dali previesť pomocou logaritmov na relatívne ľahké procesy sčítania a odčítania. Napier použil za základ logaritmov číslo *e*, ale viac sa rozšírili výpočty pomocou praktickejších tabuliek logaritmov so základom 10 anglického matematika Henryho Briggsa,

kde bolo možné bez problémov meniť rád čísel. Skoro po publikovaní tabuliek došlo aj k objavu logaritmického pravítka, kde sa výpočty robili posúvaním dvoch stupníc. Tak používanie tabuliek, ako aj pravítka bolo potom bežnou súčasťou výučby až do nástupu vreckových kalkulačiek v sedemdesiatych rokoch dvadsiateho storočia.

Keď už bolo jasné, akých princípov pri výpočtoch použiť, bolo iba otázkou času, kedy sa objaví prvé mechanické zariadenie, ktoré tieto postupy bude mať konštrukčne zabudované. Konštrukcia prvého mechanického kalkulátoru je v súčasnosti pripisovaná nemeckému vedcovi Wilhelmovi Schickardovi (1592-1635). Jeho zariadenie robilo sčítanie a odčítanie pomocou pohybu ozubených koliesok spojených s numerickým displejom. Schickard ale predčasne zomrel na mor a jeho vynález bol nadlho zabudnutý.

Vynález prvého zrealizovaného mechanického kalkulátoru je pripisovaný francúzskemu matematikovi a filozofovi Blaisovi Pascalovi (1623-1662). Pascalov otec bol zaťažený vyčerpávajúcou účtovníckou robotou a Pascal ako 19-ročný navrhol otcovi, že mu postaví stroj schopný vykonávať jednoduché výpočty automaticky. Po troch rokoch vyrobil prototyp prvého stroja pomenovaného Pascaline, ktorého sa neskoršie aj cez jeho mechanické nedostatky vyrobilo okolo 50 kusov, pozri obr. 1.3. (Švajčiarsky informatik Niklaus Wirth v druhej polovici 20 storočia nazval svoj programovací jazyk PASCAL na počesť tohto vedca.) Pascal v svojej knihe *Myšlienky* napísal: "*Aritmetický prístroj dáva výsledky, ktoré sa blížia mysleniu viac než všetko, čo robia živočíchy; nerobí však nič, aby sme mohli povedať, že má vôľu ako človek*."



Obrázok 1.3. Pascaline

Konečne Gottfried Wilhelm von Leibnitz, matematik, logik a filozof, spolutvorca diferenciálneho a integrálneho počtu s Newtonom a autor v súčasnosti používaného symbolizmu, postavil r. 1673 stroj, ktorý dokázal automaticky tiež násobiť a deliť. Leibnitz je tiež autorom myšlienky o výhodách dvojkovej sústavy.

V priebehu šestnásteho až sedemnásteho storočia došlo teda k prielomu vo vývoju mechanických zariadení určených k výpočtom, ich princípy tvorili základy mechanických

kalkulátorov využívaných až do polovice dvadsiateho storočia. Tieto kalkulátory ale boli nedokonalé v nasledujúcich aspektoch:

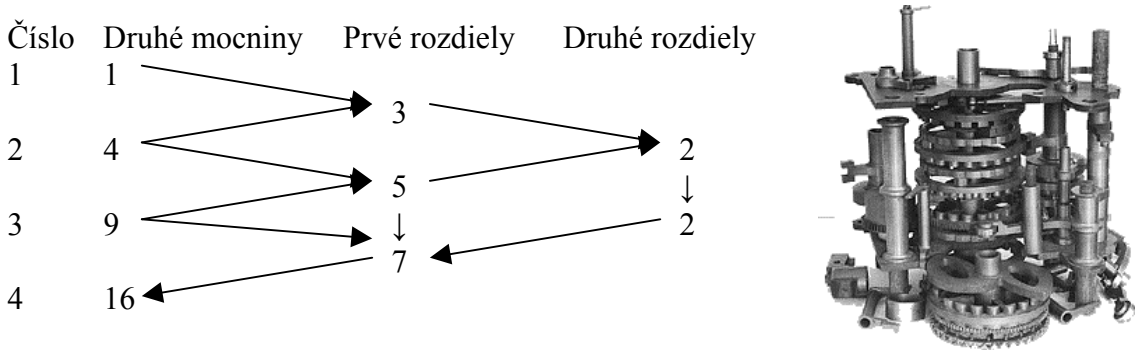
1. Neexistoval u nich koncept programu, takže sada rovnakých krokov pre výpočet využívajúci jedny vstupné dáta musela byť opakovane ručne zadávaná pre iné vstupné dáta.
2. Vo väčšine prípadov tu neexistuje pamäť, medzivýsledky museli byť väčšinou zapísané na papier a znova ručne zadávané
3. Z technologického hľadiska ide čisto o mechanické stroje bez použitia elektroniky.

Je prekvapujúce, že prvé "programy" sa neobjavili u kalkulátorov, ale u tkáčskych stavov. V priebehu osemnásteho storočia boli vyvinuté tkáčske stroje, ktoré boli o mnoho rýchlejšie ako výroba látok na ručne poháňaných tkáčskych stavoch. Spočiatku tkáčske stroje dokázali tkať iba veľmi jednoduché vzory. Prvý krok k programom urobil Jacque de Voucanson, ktorý navrhol, aby sa opakovanie vzorov na látkach pri výrobe na tkáčskych stavoch programovalo pomocou papierových kariet v podobe dierkovanej šablóny. Nápad 70 rokov ostal nepoužitý, až počiatkom 19. storočia vytvoril Joseph Marie Jacquard tkáčsky stroj, ktorý vytváral tkaný vzor pomocou dierkovaných kartičiek. Ihly prechádzali systémom dierok na kartónových kartách, preťahovali nite a tak tkali požadovaný vzor. Umiestením dierok na kartách bol jednoznačne určený tkaný vzor. Kombinovaním "pásky" kariet spojených do postupnosti bolo možné vytvoriť veľmi zložité vzory (napríklad hodvábny portrét Jacquarda bol vytvorený pomocou "programu" z 10000 kariet). Jacqard stroj v roku 1801 úspešne predstavil na svetovom trhu v Paríži a podobná technika tkania sa používa dodnes. Súvislosť s počítačmi je práve v analógii s programami – máme univerzálny stroj a pre rôzne činnosti vytvoríme rôzne programy (kartičky).

V súvislosti s kartičkami časovo preskočíme jednu dôležitú etapu vývoja počítačov (ku ktorej sa neskoršie vrátime), zmienime Holleritovo použitie automatizovania výpočtov. Americký inžinier Hermann Hollerith (1860-1929) použil r. 1890 pri sčítaní obyvateľov USA dierkovacie štítky, kde každému občanovi bol priradený jeden štítok. Vydierkovaný otvor na určenom mieste na štítku znamenal ÁNO pri odpovedi na danú otázku sčítacieho hárku. Ručne dierkované štítky sa vkladali do matrice a ich dierky určovali prechod elektrického prúdu, čo sa prenášalo na panel tabelačného stroja. Na zistenie obsahu štítkov bol použitý elektrický princíp, pomocou drôtených kefiek sa ohmatávanie štítkov zmenilo na kontrolu spojenia v elektrickom obvode - keďže papier je izolant, pri neprítomnosti otvoru je prúd prerušený, otvor zasa tok prúdu obnoví. Stroj umožňoval triedenie dát podľa daného hesla, robil jednoduché výpočtové operácie a bol schopný výsledok vytlačiť alebo znova nadierkovať. 43 strojov spracovalo údaje o 62 miliónoch ľuďoch za niekoľko týždňov, predtým sa údaje o 50 miliónoch ľuďoch spracovávali ručne 500 pracovníkmi po dobu 7 rokov. Holleritom založená firma Tabulating Machine sa r. 1924 premenovala na International Bussiness Machines - slávne IBM.

Okrem spracovania a analýzy dát je ale počítačová technológia spojená aj s výpočtami zložitých algebraických a aritmetických formúl. Prvým zariadením zo súčasťami konceptuálne podobnými zložkám moderných počítačov bol Babbageho diferenčný stroj (Differential engine). V roku 1830 Charles Babbage (1791-1871), anglický matematik navrhol stroj poháňaný parou na výpočet a tlač matematických tabuliek, využívajúci metódu diferencií. Táto metóda prevádzala výpočet polynómov na sčítanie. Presnejšie, diferenčný stroj počítal hodnoty funkcií (ako napr. tabuľky druhých mocnín) na základe rozdielov medzi po sebe nasledujúcimi hodnotami, a potom na základe rozdielov týchto rozdielov. Tento proces je opakovaný, dokiaľ nie je nájdený stĺpec rozdielov, ktorého hodnoty sú všetky rovnaké.

To je založené na nasledujúcom teorému: *n*-té rozdiely polynómu *n*-tého stupňa[3] sú všetky konštantné. Akonáhle je nájdený stĺpec konštantných hodnôt (alebo jeho prijateľná aproximácia), môžeme extrapolovať tabuľky hodnôt predĺžením stĺpca konštánt o jednu hodnotu a potom odvodením susednej hodnoty v stĺpci naľavo pripočítaním novo pridaného konštantného rozdielu k odpovedajúcej hodnote v stĺpci o jedno miesto naľavo hore. Na obr. 1.4. je uvedený príklad pre výpočet druhých mocnín: zo začiatku vypočítame čísla 1, 4, 9, ich rozdiely 3 a 5 a rozdiel rozdielov (druhé rozdiely) 2. Číslo 2 odpíšeme ako ďalší druhý rozdiel, z čísla 5 a 2 vypočítame nový prvý rozdiel 7, a z druhej mocniny 9 a prvého rozdielu 7 vypočítame novú druhú mocninu 16. Takto môžeme sčítaním vyrábať nové hodnoty druhých mocnín ďalších prirodzených čísel.



Obrázok 1.4. Princíp Babbageho výpočtu polynómov (v danom prípade druhej mocniny) v Differential engine a časť tohto stroja.

V Babbageho dobe boli dôležitou pomôckou pri výpočtoch tabuľky hodnôt funkcií, udané vždy na obmedzený počet miest. Pri takto obmedzenej presnosti sa ľubovoľná vtedy tabelovaná funkcia dala nahradiť jej aproximáciou, polynómom. Z matematického hľadiska sa dá povedať, že pomocou polynómu dostatočne vysokého rádu sa s danou presnosťou dajú v zadanom intervale nahradiť hodnoty ľubovoľnej spojitej[4] funkcie. Namiesto tabelácie funkcie teda stačí tabelovať odpovedajúci polynóm - a mechanický postup na tabelovanie je zrejmý z hore uvedenej metódy diferencií. Babbage na ukážku štátnym úradníkom vyrobil stroj ktorý dokázal tabelovať druhé a tretie mocniny a jeden komplikovanejší polynóm. Pomocou prototypu diferenčného stroja Babbage vypočítal a vydal osemmiestne tabuľky logaritmov prirodzených čísel od 1 po 10^5.

Plánovaná konečná verzia diferenčného stroja mala počítať s presnosťou na 20 miest ľubovolný polynóm. Bez toho, aby ale stroj dokončil (pre technické ťažkosti), začal pracovať na návrhu stroja pre univerzálne použitie - Analytický stroj (Analytical Engine). Výpočet plánoval riadiť pomocou diernych štítkov, kedy jeden druh štítkov obsahoval určenie operácie a druhý adresu čísla; stroj mal mať mechanickú adresovateľnú pamäť (až 1000 čísel po 50 cifrách) a "mlynček" = centrálnu aritmetickú jednotku, v ktorej sa mali vykonávať základné operácie. Babbageho mechanické zariadenia požadovali ale presnosť obrábania, ktorá nebola v 19. storočí dostupná. Ako teoretický koncept má Analytický stroj ale obrovský význam. Obsahoval pamäť, procesor a prvýkrát sa objasnilo, že by mechanickým spôsobom bolo možné programovať komplikované algoritmy. O svojom Analytickom stroji Babbage hovoril, že bude predstavovať mechanickú inteligenciu. Cieľom bolo, aby si stroj dokázal sám flexibilne meniť svoj program v priebehu výpočtov.

[3] Polynóm stupňa *n* je funkcia, ktorá má tvar $a_n x_n + a_{n-1}x^{n-1} + ... + a_1x + a_0$

[4] Voľne povedané, funkcia *f* je spojitá v čísle *r* zo svojho definičného oboru, ak hodnoty funkcie *f*(*r*) v číslach blízkych *r* sú blízke hodnote *f*(*r*), spojitá funkcia neobsahuje žiadne "schodíky" alebo "osamotené" body.

Blízkou Babbageho spolupracovníčkou bola dcéra anglického básnika Byrona - grófka Augusta Ada King of Lovelace. Tá pri svojej práci odhalila základy programovania a význam podmieneného vetvenia programu podľa výsledku predchádzajúceho kroku a význam cyklov alebo slučiek v programovaní, napísala aj program na riešenie sústavy lineárnych rovníc a na generovanie Bernoulliho čísel[5]. Špekulovala tiež o možnosti použiť stroj na generovanie hudobných diel pomocou zakódovania zákonov harmónie a kompozície na dierne štítky. Taktiež uvažovala o možnosti použiť analytický stroj v úlohe manipulátora s algebraickými výrazmi.

Videla ale aj obmedzenia počítačov, v dopise svojmu známemu napísala: "*Analytický stroj nemá ambície vymyslieť niečo originálne. Dokáže urobiť iba čokoľvek, o čom vieme, ako mu prikázať, aby to vykonal. Môže postupovať podľa výsledkov analýzy riešenia; nemá ale žiadnu schopnosť vymyslieť akékoľvek analytické vzťahy alebo tvrdenia*." V druhej polovici 20. storočia bol na jej počesť pomenovaný programovací jazyk ADA.

O praktické rozšírenie kalkulátorov sa vo Francúzsku zaslúžil Charles Xavier Thomas, majiteľ poisťovacej spoločnosti, ktorý chcel ušetriť na vypočtároch. Roku 1820 dáva patentovať mechanický počítač Arithmometr. Do r. 1880 to bol jediný vyrábaný počítač pre praktické použitie, umožňoval rátať mocniny, odmocniny, trigonometrické funkcie a zostavoval tabuľky na 20 desatinných miest, ale nemal program ani pamäť a bol poháňaný ručne pomocou kľuky. Na delenie a násobenie osemciferných čísel potreboval asi dvadsať sekúnd. Aritmometrov bolo vyrobených asi 1500.

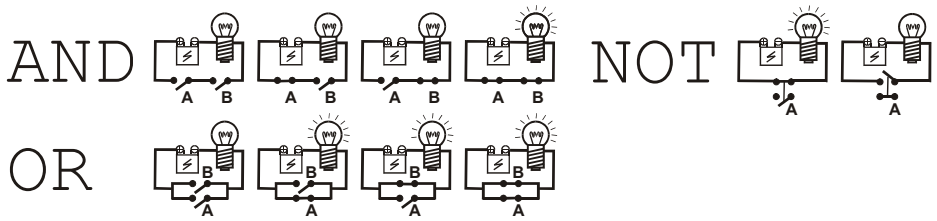
Vedľa zariadení, ktoré počítajú s diskrétnymi hodnotami, vznikali aj analógové stroje. Príkladom je elektromechanický diferenciálny analyzátor, ktorý zostrojil v roku 1930 Vennevar Bush.

Aj keď niektoré z prvých počítačov pracovali tiež analógovým spracovaním signálu, nakoniec zvíťazili počítače s binárnou logikou (teda spracovávajúce na najnižšej úrovni iba logické nuly a jednotky). Základy binárnej logiky položil anglický matematik George Boole (1815-64), vymyslel systém na ohodnotenie pravdivostných hodnôt výrazov zložených z logických spojok AND, OR, NOT a logických premenných nadobúdajúcich iba dve hodnoty – 1 (PRAVDA) a 0 (NEPRAVDA). Boolovská algebra našla použitie pri návrhu technologickej súčasti moderných počítačoch, a to v binárnych klopných obvodoch. Pomocou elektronicky ovládaných hradiel sú reprezentované čísla v binárnej reprezentácii, robia sa výpočty, od sčítania a násobenia až po zložité funkcie. Vhodnou kombináciou klopných obvodov vznikajú programy, ktoré sú schopné počítať pravdivostné hodnoty zložitých funkcií. Hlavná výhoda binárnych systémov je, že manipulácia s binárnymi číslami je ľahko technicky realizovateľná, pričom výsledné elektrické obvody sú neobyčajne rýchle (pozri obr. 1.5). Prakticky všetky moderné výpočtové systémy môžu byť na úrovni tých najzákladnejších operácií popísané v termínoch zložitých sietí binárnych spínacích komponentov.

Prvé z takýchto preklápacích komponentov boli elektromechanické spínacie relé, pozostávajúce z dvoch kovových kontaktov spojených kovovým jazýčkom. Spínače môžu byť ľahko použité napríklad v AND a OR obvodoch, kedy "otvorený" (vypnutý) spínač predstavuje hodnotu NEPRAVDA (FALSE alebo nulu v binárnom kódovaní, neprechádza prúd) a "uzatvorený" (zapnutý) spojený spínač hodnotu PRAVDA (TRUE alebo jednotku v binárnom kódovaní, prechádza prúd). Obvod NOT jednoducho mení pozíciu "otvorený" alebo "uzatvorený" na opačnú, dá sa predstaviť ako pár mechanicky spojených spínačov, kedy vstupný spínač nepatriaci do obvodu, spôsobuje zmenu polohy druhého spínača na opačnú.

[5] Bernoulliho čísla B_k, ktoré zaviedol Jacob Bernoulli, sa dajú vypočítať z rovníc $1 + 2B_1 = 0$, $1 + 3B_1 + 3B_2 = 0$, $1 + 4B_1 + 6B_2 + 4B_3 = 0$, $1 + 5B_1 + 10B_2 + 10B_3 + 5B_4 = 0$, atď., kde môžete rozpoznať začiatky binomických koeficientov (to sú koeficienty u členov po umocnení $(a+b)^n$). Prvých zopár čísel je B_1=-1/2, B_2=1/6, B_3=0, B_4=-1/30, B_5=0, B_6=1/42,.... Tieto čísla sa využívajú v kombinatorike a teórii čísiel.

Pre použitie premennej súčasne v niekoľko obvodoch by ale boli potrebné "vetviace" spínače. Ľubovoľný výraz Boolovej algebry sa dá previesť na usporiadanie spínačov, pri pravdivosti výrazu bude obvodom prechádzať prúd.



Obrázok 1.5. Spínacie obvody AND (sériovo zapojené spínače), NOT, OR (paralelne zapojené spínače) vyrobené z batérie, žiarovky a spínačov. Pri NOT sú dva jazýčky mechanicky prepojené, pre uzatvorený spínač A je obvod otvorený. V uzatvorenom obvode ako výsledok TRUE prechádza prúd a svieti žiarovka.

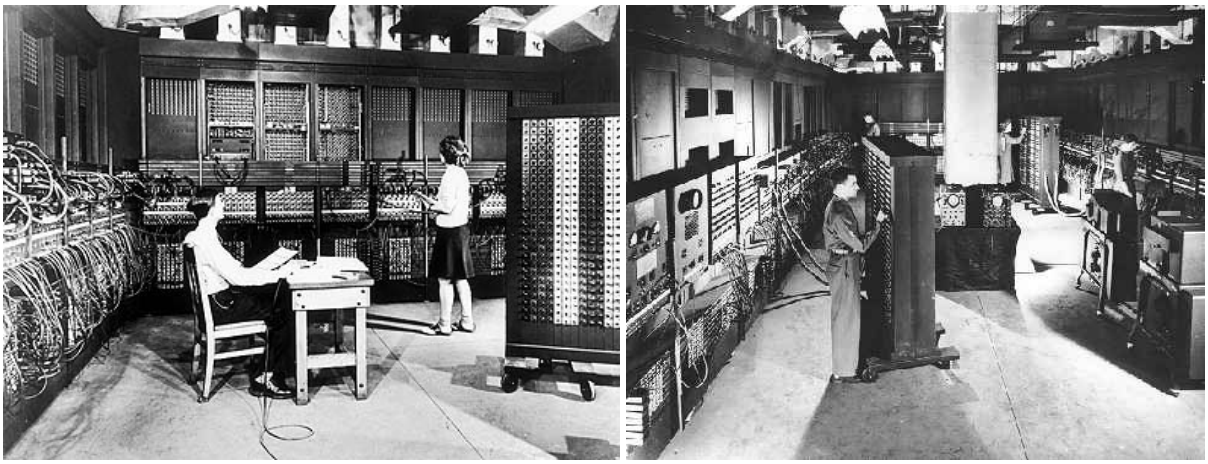
V rokoch 1936-1944 sa zaoberal výstavbou počítačov nemecký inžinier Konrad Zuse v Berlíne. Aj keď jeho prvý počítač Z1 bol ešte mechanický, druhý počítač Z2 už používal elektromagnetické relé na základe nápadu Zuseho kolegu Helmuta Schreyera. Zusom navrhnuté počítače už mali všetky podstatné prvky plánované Babbageom. V priebehu druhej svetovej vojny ale neexistoval o ich nápady záujem. Zusov počítač Z3 postavený r. 1941 už mal pamäť na 64 čísel v dvojkovej sústave o 22 bitoch[6], s plávajúcou desatinnou čiarkou a dokázal násobiť v priebehu 3 sekúnd. Stroj obsahoval aj čítačku diernej pásky s vydierovaným programom a asi 2500 relé. Počítač ešte nemal podmienené skoky.

V USA a Británii zatiaľ prebiehal vývoj nezávisle. Howard Aiken r. 1937 na Harvardskej univerzite začal presadzovať myšlienku počítača, kombinujúceho Babbagove myšlienky reprezentáciu dát diernymi štítkami. IBM v spolupráci s Harwardskou universitou tak medzi r. 1939-44 postavila počítač Mark 1, obsahujúci asi 750000 relé, teda tristokrát viac ako Zusseho počítač, rýchlosť operácií mal ale približne rovnakú ako Z3. Vážil 5 ton a bol 16 m dlhý. Pracoval v desiatkovej sústave s pevnou desatinnou čiarkou a tiež nepoznal podmienené skoky. John Atanasoff a Clifford Berry potom zostavili počítač ABC Athanasoff Berry Computer o 300 elektrónkach na riešenie sústavy lineárnych rovníc a navrhli používať ako pamäťové médium magnetické bubny. John Prosper Eckert (1929-) a John Mauchly r. 1943 presvedčili vládu U.S.A. aby financovala počítač ENIAC Electronic Numerical Integrator and Calculator, postavený z elektroniek (pozri obr. 1.6). Bol to asi prvý mnohoúčelový elektronický počítač, využívaný bol ale predovšetkým na výpočty spojené s výrobou atómovej bomby. Bol dokončený r. 1944, zaberal asi 150 m^2, vážil 30 ton, mal 100 000 súčiastok, bol chladený vrtuľami dvoch leteckých motorov a dokázal násobiť dve čísla za 0,003 sekundy. Pracoval v niekoľko blokoch zaraz, paralelne, ale čísla ale ešte stále kódoval v desatinnej sústave namiesto binárnej. Podobne ako jeho predchodcovia vyžadoval od operátorov po každom výpočte prepínanie drôtov a nulovanie výpočtov, čo zaberalo hodiny.

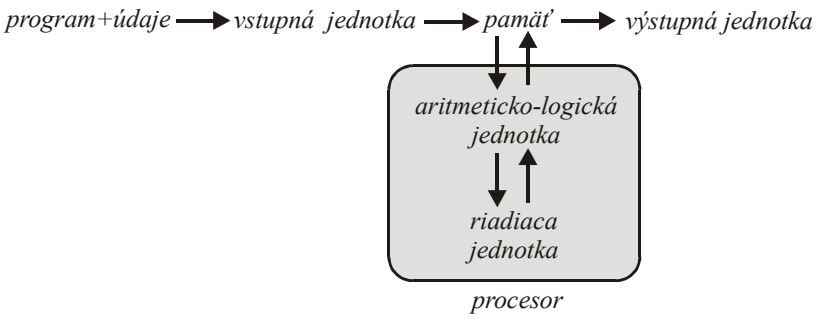
S počítačom ENIAC a jeho nástupcom EDVAC je spojené meno John von Neumann, ktorý je svetoznámy svojou teoretickou prácou o počítačovej architektúre (pozri obr. 1.7). Túto architektúru majú teraz všetky jednoprocesorové stroje, počítal s umiestnením programu aj dát v jednej pamäti, s riadiacou jednotkou a aritmeticko-logickou jednotkou a so vstupným a výstupným zariadením. Pamäť je rozdelená na rovnako veľké bunky, ktoré sú očíslované, cez číslo (adresu) bunky sa dá prečítať alebo meniť jej obsah. Program je uložený do buniek idúcich za sebou, nasledujúca inštrukcia sa zoberie vždy z nasledujúcej bunky s adresou o jednotku vyššou, pokiaľ nepredchádzala inštrukcia skoku. Program dokáže upravovať sám

[6] Bit b (binary digit, dvojková číslica) umožňuje záznam binárnej jednotky alebo nuly, je to najmenšia jednotka informácie. Používa sa v kb=10^3 b. Súčasné počítače reprezentujú čísla, písmena, obrázky, programy aj všetko iné vzormi bitov.

seba v priebehu výpočtu. Všetky dáta aj program sú binárne kódované, dekódovanie zabezpečujú logické obvody riadiacej jednotky.



Obrázok 1.6. ENIAC, mnohoúčelový elektronický počítač z r. 1944



Obrázok 1.7. Schéma von Neumannovskej architektúry, platné od EDVACu až po osobný počítač (zobrazené sú iba hlavné dátové toky)

V Británii zatiaľ prebiehal nezávislý vývoj spojený s prácou anglického matematika Alana Mathiesona Turinga (1912-54) na dekódovaní nemeckých šifier. Turing sa ale okrem výpočtov zamýšľal aj nad použitím počítačov v takých oblastiach, ako lúštenie krížoviek alebo hranie šachu. Bol tiež jedným z prvých ľudí, ktorý sa zamýšľal nad otázkou, či počítače môžu byť inteligentné. Navrhol tzv. Turingov test: Keď človek nie je schopný rozlíšiť, či dostáva odpovede na svoje dotazy od počítača alebo od iného človeka (komunikácia prebieha napr. cez textové správy alebo cez terminál), dá sa taký počítač považovať za inteligentný.

Anglickí výskumníci najprv ako pomôcku na dešifrovanie používali malé stroje založené na reléových spínačoch, ale v r. 1943 vyrobili COLLOSUS o 1800 elektrónkach.

V teoretickej rovine r. 1942 Norbert Wiener opublikoval prvé práce o podobnostiach v činnosti nervovej sústavy živého organizmu a matematického stroja, r. 1948 vydáva dielo "Cybernetics and Communication in the Animal and the Machine". Vzniká tak kybernetika, úzko spojená s počítačovou informatikou. Roku 1950 potom Turing publikuje prácu "Computing Machinery and Intelligence", datujúcu vznik vednej disciplíny umelej inteligencie.

Formálne sa ďalší vývoj veľkých (sálových) počítačov dá rozdeliť do generácií. O tzv. **nultej generácii** sme si už povedali, išlo o prvé počítače na rôznej súčiastkovej základne (väčšinou na základe **relé**) ako ENIAC, Z1 až Z3, Mark I. ENIAC býva zaraďovaný aj do **prvej generácie**, ktorá je charakterizovaná nahradením relé pomocou bistabilných spínacích obvodov zložených z **elektroniek**.

Druhá generácia je spojená so zmenšením rozmerov a spotreby energie spojenej s vynálezom **tranzistoru** r. 1947. Počítače ale boli nekompatibilné a stále slabo rozšírené.

Tretiu generáciu datovanú od r. 1961 charakterizuje nástup integrovaných obvodov, čo znamenalo niekoľko tranzistorov (spočiatku 10-100) na jednej doštičke - **čipu**. To prinieslo výrazné zmenšenie a urýchlenie montáže. Za prvý počítač tretej generácie sa pokladá IBM/360. V socialistických krajinách východného bloku im odpovedali počítače EC1021 a ďalšie, vznikali typové rady počítačov JSEP - Jednotný Systém Elektronických Počítačov. Vzhľadom k tomu, že kybernetika bola zo začiatku v Sovietskom Zväze považovaná oficiálne za buržoáznu pavedu, vývoj v krajinách východného bloku bol opozdený a väčšinou iba kopíroval úspešné modely zo západných krajín a aj to s desaťročným spozdením.

Triapoltá generácia bola charakterizovaná LSI – Large Scale Integration s niekoľko tisíckami tranzistorov na čipe. V tej dobe začali vznikať aj minipočítače firiem DEC - Digital Equipment Company a Hewlett Packard.

Štvrtá generácia sálových počítačov je charakterizovaná **VLSI** - integrovanými obvodmi s veľmi vysokou hustotou (milióny tranzistorov na niekoľko centimetroch štvorcových). Nastupujú superpočítače CRAY, multiprocesorové systémy a pracovné grafické stanice Silicon Graphics (čo sú voľne povedané veľmi "nadupané" počítače veľkosti osobných počítačov, ktoré v súčasnosti môžu mať až stovky procesorov).

Piata generácia je skôr otázna, ide predovšetkým o opustenie von Neumannovy koncepcie s prechodom od jedného procesoru k paralelnému spracovaniu výpočtov. Čo sa týka programového vybavenia, malo by ísť o zvýšený dôraz na postupy umelej inteligencie napodobňujúcej myšlienkové pochody človeka, schopné konverzácie s človekom, automatickej opravy programov a samostatného rozhodovania.

Od r. 1969 sa s vynálezom **mikroprocesoru** datuje nástup domácich a **osobných počítačov**, s počiatkom osemdesiatych rokov to boli IBM PC a Macintosh (Apple). Firma INTEL (Integrated Electronics) r. 1971 vyrobila prvý viacúčelový mikroprocesor Intel 4004, ktorý obsahoval 2300 tranzistorov v štvorbitovej architektúre[7] a dal sa použiť tak do kalkulačky, ako aj na riadenie výťahu.

Osobné počítače sú charakterizované predovšetkým svojou veľkosťou, tým, že sa zmestia na pracovný stôl. Prvé domáce počítače boli r. 1977 Commodory a Apple 1, boli osembitové, s dátami nahrávanými z kazetového magnetofónu a s textovým režimom monitoru. Nasledovali Sinclair, Atari, Robotron, u nás IQ 151, PMD 85 a ďalšie.

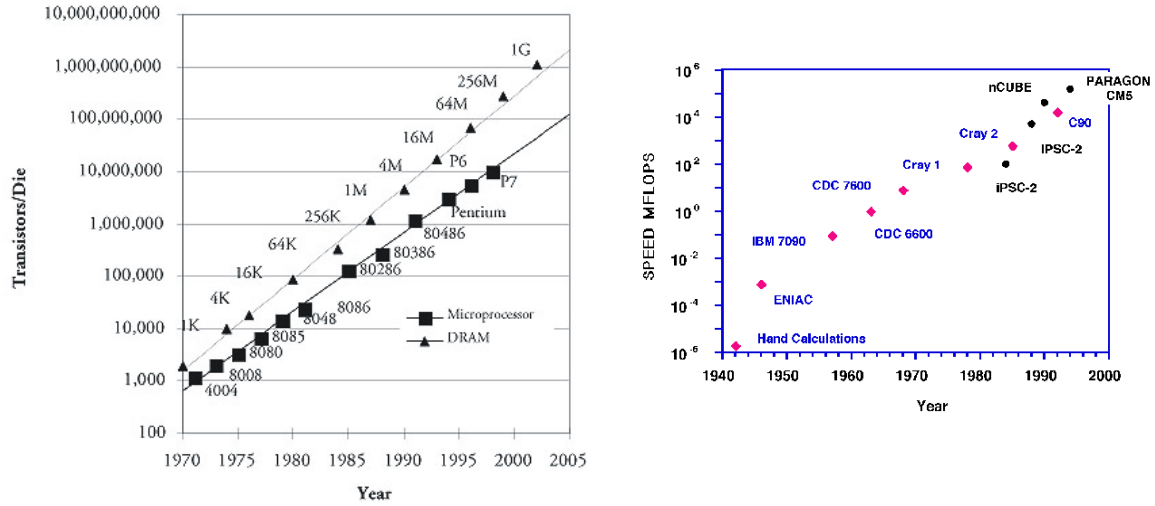
Vznikli diskety, spočiatku osempalcové, potom päť a štvrť palcové, teraz sú najčastejšie tri a pol palcové. Dá sa predpokladať, že v budúcich rokoch sa prejde od diskiet na pamäťové karty, aj keď v pamäťových médiách s nimi budú súťažiť prepisovateľné CD a DVD disky.

Firma IBM sa počiatkom 80-tých rokov rozhoduje, že jej osobné počítače budú montované z dielov od nezávislých dodávateľov, ako procesory firmy Intel a operačný systém od neznámej začínajúcej firmy Microsoft. IBM PC predstavený r. 1981 bol založený na 16 bitovom mikroprocesore Intel 8088, mal 5,25 palcovú disketu, verzia XT už mala aj hard disk s 10 MB pamäťou (MB=megabajt - mega je 10^6 a B=bajt je jednotka pamäti o ôsmich bitoch). Čo sa týka operačnej pamäti, bola 640 kB (k je pre kilo = 1000), sú neslávne známe slová šéfa Microsoftu Billa Gatese "640 kB pamäti musí stačiť každému".

Rýchlosť počítačov sa menila od 1000 operácií za sekundu u prvej generácie až po milión operácií za sekundu u štvrtej generácie. V roku 1964 Gordon Moore formuloval pravidlo, že zložitosť počítačových obvodov sa približne každého jedenapolroka zdvojnásobí.

[7] spracováva naraz 4 bity. V súčasnosti rozšírené 32-bitové počítače používajú na binárnu reprezentáciu čísla kombináciu 32 bitov

Táto téza bola neskoršie nazvaná Moorov zákon, a platí podnes (pozri obr. 1.8). Je ale jasné, že klasická miniaturizácia má svoje fyzikálne hranice. Ich prekonanie sa môže podariť iba použitím nových konštrukčných prvkov, optických, alebo možno aj kvantovo-mechanických, ktoré sú v štádiu výskumov.



Obrázok 1.8. Moorov zákon: počet tranzistorov na mikroprocesore stúpajúci s časom, spolu s veľkosťou pamäti DRAM. Druhý graf ukazuje zvýšenie rýchlosti výpočtu pre rôzne typy počítačov, stúpajúce s časom.

Firma Intel sa odpútava od IBM a vyvíja stále rýchlejšie procesory, r. 1985 32-bitový 80386, r. 1989 80486, r. 1993 Pentium, nasledované ďalšími generáciami.

Výkonnosť počítača (operačná rýchlosť) vyjadruje počet porovnateľných operácií, ktoré je počítač schopný vykonať za sekundu, jednotka tisíc (milión) operácií za sekundu je Kops (Mops). Často sa tiež používa počet náročnejších operácií s číslom s pohyblivou desatinnou čiarkou za sekundu (Flops - floating point operation per second, väčšie jednotky sú Mflops = 10^6 Flops, Gflops =10^9, Tflops - 10^{12}). U multiprocesorových strojov je ale toto merítko niekedy zavádzajúce, nie všetky algoritmy sa dajú dobre sparalelniť a tak multiprocesorovému počítaču s lepšou výkonnosťou môže trvať úloha, ktorá sa nedá sparalelniť, aj dlhšie. S výkonnosťou úzko súvisí aj frekvencia časovača (timer frequency), charakterizujúca rýchlosť procesora. Vyjadruje sa v megaherzoch (MHz=10^6 Hz), a u rovnakého výrobcu zvyčajne zdvojnásobenie frekvencie znamená aj zdvojnásobenie rýchlosti výpočtov (ale aj zvýšenie prehrievania procesora). Pre počítač čas nie je totiž plynulý tok, ale postupnosť prechodov medzi stavmi počítača. U rôznych výrobcov nie sú procesory takto priamo porovnateľné, lepšie zostavený procesor môže aj pri nižšej frekvencii počítať rýchlejšie. Pre praktické účely pri využívaní grafiky je niekedy základným faktorom veľkosť adresovateľnej pamäti RAM (Random Access Memory= pamäť s ľubovoľným prístupom) alebo kvalita grafickej karty a veľkosť vyrovnávacej pamäti "cache".

Pamäťová kapacita je charakterizovaná počtom pamäťových jednotiek, ktoré je možno zaznamenať na príslušný druh pamäte. Zvyčajne sa udáva v bitoch alebo bytoch. Byte (značí sa veľkým písmenom B, číta sa bajt) je súvislá postupnosť najčastejšie 8 bitov, slúži na zobrazenie jedného znaku (8 bitov umožňuje zobraziť 256 znakov). Používa sa v jednotkách kilobyte (KB=2^{10}B=1024 B), megabyte (MB=2^{20}B=1048576 B), gigabyte (GB=2^{30}B =1073741824 B) a terabyte (TB=2^{40}B =1099511627776B).

Operačná rýchlosť a taktiež aj pamäťová kapacita počítačov sa s každým rokom zvyšujú, ale vlastné algoritmy a princípy programového vybavenia sa menia len pomaly. Pre výkonnosť počítačov ale taktiež aj pre prácu užívateľa s nim má podstatný vplyv operačný systém. Operačný systém monitoruje a riadi všetky softwarové a hardwarové aktivity, teda

napr. vstupné a výstupné jednotky. Keď stlačíme klávesu klávesnice, operačný systém neustále cyklicky kontroluje všetky klávesy, zistí, ktorá z nich je stlačená, a prípadne daný znak zobrazí na monitor počítača.

V súčasnosti sú dva základné druhy operačných systémov, a to UNIX (spolu s jeho verziou Linux pre osobné počítače) a Microsoft Windows. Microsoft Windows majú jednu zásadnú výhodu, a to je ich jednoduchá inštalácia. Ich podstatnou nevýhodou je ich vysoká spotreba zdrojov počítača (program môže bežať o polovicu pomalšie, pretože počítač stráca veľa času, aby "riadil sám seba"). Windows boli od začiatku určené pre osobné počítače, aj keď ich novšie verzie, napr. Windows NT sú určené aj pre počítače-servery, ktoré poskytujú sieťové služby napr. na internete. UNIX bol od začiatku spoľahlivý operačný systém určený pre sálové počítače, ktorý bol neskoršie prepísaný na osobné počítače (v tejto verzii sa nazýva Linux). Vzhľadom k tomu, že bol určený pre profesionálov, neobsahuje toľko podpory užívateľovi. Aj keď jeho novšie verzie sa snažia pomôcť užívateľovi širokou ponukou rôznych možností pri inštalácii (napr. distribúcia Linuxu Red Hat), jeho využitie ostáva väčšinou iba pre profesionálov. Pokiaľ chcete na počítači iba písať text alebo spúšťať tabuľkový kalkulátor, je výhodné sa zmieriť s pomalosťou a taktiež aj s nie veľkou spoľahlivosťou operačného systému Windows (v porovnaní s Linuxom), než ako strácať čas štúdiom technických aspektov počítača pre nastavenia Linuxu.

Do začiatku 50-tych rokov 20. storočia bolo ale zadanie problému do počítača veľmi zložité, pretože do počítača bolo treba zadať program aj dáta ako sadu núl a jedničiek. Aby sa predišlo ľudským chybám pri zadávaní programu, vymyslel r. 1950 Maurice Wilkes špecifikáciu programu pomocou primitívnej verzie (z dnešného pohľadu) assembleru, čo boli jednoduché skratky príkazov v angličtine (ako "pričítaj" alebo "ulož do pamäti"), ktoré potom boli automaticky preložené do odpovedajúcich sád bitov použiteľných strojom.

Ďalší vývoj viedol k programovacím jazykom vyššej úrovne. Zatiaľ čo v assembleri sa jeden riadok programu bežne preložil do jednej inštrukcie počítača, jeden príkaz jazyka vyššej úrovne znamenal niekoľko strojových príkazov riadiacej a aritmeticko-logickej jednotke počítača. Takéto programy ale museli byť transformované (kompilované) do sady núl a jedničiek, čo bolo úlohou špecializovaných programov nazvaných kompilátory. Námorná admirálka USA pani Grace Murray Hopper[8] potom r. 1952 vymyslela prvý kompilátor. Jazyky vyššej úrovne umožnili používať počítače ľuďom, ktorí nemajú detailné technické znalosti o operáciách počítača na najnižšej úrovni spracovania bitov.

Vo vedecko-technických aplikáciách je snáď najznámejším programovacím jazykom FORTRAN (FORmula TRANslation), ktorý bol vyvinutý v polovine päťdesiatych rokov a je využívaný až dodnes. Aj keď sa v priebehu rokov objavovali nové verzie jazyka, v princípe sa dodržovala kompatibilita smerom hore, teda staršie programy sa dali spustiť aj na kompilátoroch pre novšie verzie jazyka. Tento trend sa objavil aj u ostatných programovacích jazykov a ostatných programových systémoch. Pretože vo FORTRANe je napísané také množstvo tak rozsiahlych programov, že sa nikomu nechce ich prepisovať do nových jazykov, je pravdepodobné, že FORTRAN sa bude ešte dlho používať.

Medzi ďalšie programovacie jazyky patrí napríklad COBOL, využívaný od počiatku šesťdesiatych rokov k spracovávaní dát obchodnej agendy a riadenia podniku. Pre účely umelej inteligencie bol potom v MIT v r. 1960 (Massechusets Institute of Technology, jedna z najlepších univerzít v USA) vyvinutý programovací jazyk LISP (LISt Processing, študentmi vysvetľovaný ako Lots of Idiotic, Silly Parentheses), používajúci tzv. lamda kalkulus a rekurziu. Tento jazyk je stále široko využívaný v akademickom prostredí, napr. na automatické dokazovanie teorém alebo hranie logicky orientovaných hier, aj keď je relatívne

[8] Jedným z jej prvenstiev bolo prvé "debugovanie" počítača: rozpráva sa, že keď sa počítač Mark I jedného dňa pokazil, vysledovala problém až k more zachytenej v reléovom prepínači.

pomalý a veľmi zle čitateľný. LISP je tzv. neprocedurálny, alebo deklaratívny jazyk, ktorý pomocou logických výrokov opisujú samotný programátorský problém, a na jeho riešenie sa využívajú odvodzovacie, deduktívne algoritmy. V Európe je viac používaný jazyk logického programovania PROLOG. Všetky ostatné zmienené jazyky patria medzi procedurálne, teda opisujú algoritmus riešenia.

Z teraz zaniknutých jazykov je asi najvýznamnejším jazykom ALGOL, ktorý slúžil ako základ pre PASCAL Niklause Wirtha. Tento jazyk je snáď najjednoduchší na výučbu (s výnimkou jazyka BASIC), ale obsahuje všetky podstatné črty vyšších jazykov. Nanešťastie už v súčasnej dobe nie je veľmi podporovaný, teda nevytvárajú sa preň nové kompilátory pre nové operačné systémy. Preto sú v súčasnosti najpoužívanejšími programovacími jazykmi vo vedecko-technických výpočtoch jazyky C a C++. Od tohto jazyka boli tiež odvodené rôzne klony ako JAVA alebo C# určené na výpočty v prostredí internetu.

Ako vznikal internet? V r. 1969 americká agentúra ARPA (Advanced Research Project Agency) prišla s myšlienkou prepojiť počítačové systémy univerzít a federálnych výskumných stredísk medzi sebou. V r. 1972 sa začal používať prenos veľkých súborov dát pomocou služby FTP (File Transfer Protocol). V roku 1974 bol zdarma sprístupnený protokol TCP/IP (Transmission Control Protocol/Internet Protocol). V r. 1979 výskumníci z rôznych univerzít vytvorili CSNET (Computer Science Research Network), ktorá bola o rok neskoršie prepojená protokolom TCP/IP zo sieťou ARPEnet. Prepojením nezávislých sietí sa datuje vznik **internet**u. T. Berners-Lee, pracovník Európskeho laboratória fyziky častíc v Ženeve, prišiel r. 1989 s myšlienkou spojiť prenos súborov na sieti s tvorbou **hypertext**ových dokumentov, v ktorých označené slová odkazujú na ďalšie dokumenty. Vytvorený programový komplet umožňujúci fungovanie hypertextu nazval WEB, od vtedy sa World Wide Web stal súčasťou internetu. Rast počtu počítačov napojených na internet pol explozívny a počet WWW stránok narastá asi o 300000 každý týždeň.

Ukázalo sa, že prepojenie počítačov je vhodné aj pre firmy, a tak okrem tzv. LAN -- Local Area Network, vzniká aj Intranet, ktorý sa vyznačuje predovšetkým odstupňovanými prístupovými právami k dátam.

Súčasne s rozvojom internetu sa objavili aj jeho nepríjemné stránky. Roku 1988 dvadsaťtriročný Robert Morris posiela na internet tzv. červa, ktorý nakazil v priebehu 36 hodín 20000 počítačov. Morris bol za to odsúdený na 3 roky väzenia a naviac dostal pokutu 10000 dolárov. Vírus je program, ktorý sa pripája k iným programom alebo dátam a stáva sa ich súčasťou. Vírus je schopný sa rozmnožovať, a prípadne sa maskovať a škodiť. Na to je ale treba spustiť infikovaný program[9]. Oproti tomu červ (worm) je samostatný program šíriaci sa prostredníctvom siete.

Súčasné osobné počítače majú výkon lepší ako sálové počítače štvrtej generácie, a sálové počítače sa zmenšujú. Novým trendom je tzv. network computing, kedy sa využívajú kapacity momentálne nezaťažených počítačov na zložité sparalelnené výpočty.

Iným smerom je prechod od explicitne programovateľných zariadení k programom alebo zariadeniam vyvíjaným na báze Darwinovskej evolúcie. Aj keď to znie ako nápad z vedecko-fantastického románu, evolučný hardware je posledných niekoľko rokov hlavnou témou mnohých konferencií, aj keď zatiaľ ide iba o jednoúčelové mikroprocesory na riadenie pomocných technických zariadení.

Výpočty pomocou neurónových sietí alebo iných prostriedkov výpočtovej inteligencie ale stále tvoria iba výraznú menšinu v celkovom objeme výpočtov. Počítače samotné zato prenikajú nenápadne do oblastí všedného života, od automatizovaného ovládania

[9] Ako základnú prevenciu je treba si zálohovať vlastné dáta, aby ste napríklad kvôli vírusu neprišli o diplomovú prácu, mať neustále spustený a pravidelne updatovaný svoj antivírusový program a nikdy neotvárať e-mailovú prílohu, ktorá nebola vyžiadaná.

továrenských strojov po práčky, mikrovlné rúry, otváranie dverí na kartu v hoteloch a objednávka jedál v jedálňach, video a DVD prehrávače alebo mobilné telefóny. Toto lavínovité rozšírenie počítačov nie je na prvý pohľad zrejmé, pretože sú prepojené s jednoúčelovými zariadeniami a zaberajú nepatrné rozmery.

V technologickej oblasti sa vyvíja integrovaná optoelektronika, magnetoelektronika a bioelektronika. "Počítače" na základe DNA sú stále ale iba kuriozitou a v súčasnosti vyžadujú dni na riešenie primitívnych problémov.

Revolučnosť počítačov je v tom, že dokážu spracovať ľubovoľnú informáciu, ktorú vieme sformalizovať. Keďže obsah matematiky je prirodzeným spôsobom sformalizovaný, aj počítače prebrali časť práce v matematike, napr. pri dokazovaní teorém matematickej logiky. Jednoduchý a prirodzený formalizmus existuje ale aj v iných oblastiach považovaných za prirodzenú a výlučnú oblasť ľudí, ako je šachová hra alebo komponovaní hudby. Problémom týchto oblastí ale je, že formalizované sú len veľmi povrchové postupy, šachoví veľmajstri alebo hudobní géniovia nie sú schopní sformalizovať postupy prebiehajúce v ich mozgu. Preto kvalita hudobnej kompozície počítačov je iba obmedzená a šachové programy víťazia nad ľuďmi iba "hrubou výpočtovou silou", algoritmy týchto programov celkom isto neodpovedajú postupom skrytým vnútri mozgov veľmajstrov.

René Descartes vo svojej Rozprave o metóde napísal: "Aj keby stroje vykonávali určité úkoly rovnako dobre alebo lepšie ako ktokoľvek z nás, zlyhali by nevyhnutne v iných ... pretože rozum je všeobecný nástroj, ktorého je možno využívať vo všetkých možných prípadoch, zatiaľ čo stroje musia mať nejaké zvláštne uspôsobenie pre každý úkon jednotlivý."

Tieto slová snáď platili donedávna, ale s nástupom umelých neurónových sietí vyvíjaných pomocou evolučných princípov v počítačoch sa zdá, že skepticizmus Pascalov, Descartov a grófky Ady sa o niekoľko desaťročí ukáže ako neoprávnený. Pomocou nových prístupov sú počítače na základe spätnej väzby schopné sa vyvíjať a prispôsobovať (to v prípade evolučného hardvéru), prípadne sú schopné sa učiť (napr. u softvérovo implementovaných neurónových sietí).

Zhrnutie

Prvé známe zariadenie na uľahčenie výpočtov je staroveký abakus. Po logaritmickom pravítku nasledoval mechanický stroj na sčítanie francúzskeho matematika Blaise Pascala z r. 1642 a neskoršie mechanický stroj schopný násobenia nemeckého matematika Gottfrieda Leibnize. Prvý mechanický počítač "analytical engine" bol navrhnutý britským matematikom Charlesom Babbageom r. 1835. Využíval program z derných štítkov, ktorých myšlienka

pochádzala z Jacquardových tkáčskych strojov. Prvým programátorom bola grófka Ada. Americký vynálezca Herman Hollerith použil elektromechanické zariadenie a dierovacie karty pre spracovanie censu r. 1890 (z jeho firmy vzniklo IBM). Prvé elektrónkové počítače boli americký ENIAC a britský Colossus z r. 1943, prielomom je vynález tranzistoru r. 1947. Počítače využívajú logiku George Boola z 19. storočia. Teóriu počítačov rozvinuli britský matematik Alan Turing, a ich architektúru a uloženie programu s dátami v Amerike pracujúci John Von Neumann. Efektívne použitie počítačov potrebuje operačné systémy ako UNIX či Windows a programovacie jazyky, ako Fortran, Pascal, C++, alebo Prolog či Lisp, vyvíjané až od druhej poloviny 20 storočia.

Úlohy

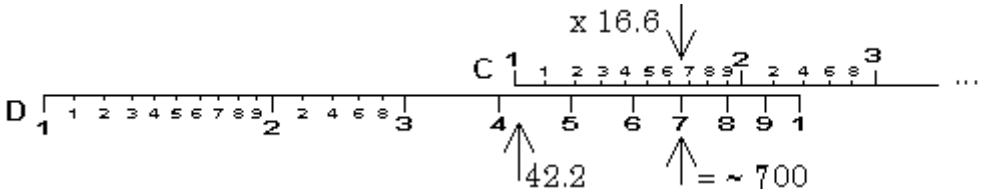
Úloha 1.1. Tu je vysvetlené, ako urobiť násobenie 58 × 43 = 2494 pomocou počítadla abakus (musíte poznať malú násobilku):

1. Vynulujte abakus odtlačením korálok od prostrednému trámiku.
2. Nastavte číslo 58 v stĺpcoch 13 a 12.
3. Nastavte číslo 43 v stĺpcoch 10 a 9.
4. Začnite s násobením 8 × 43. Tento výpočet sa dá rozdeliť na 8 × 3 = 24 a 8 × 4(0) = 32(0). Dajte výsledok 8 × 3 = 24 do stĺpcov 2 a 1. Výsledok 8 × 4 = 32 môže byť dodaný do stĺpcov 3 a 2. Násobenie číslom 8 je teraz ukončené a môžeme tak vynulovať stĺpec 12.
5. Teraz môžeme násobiť číslom 5(0): 5(0) × 43. Opäť môžeme rozdeliť výpočet na 5(0) × 3 =15(0) a 5(0) × 4(0) = 20(00). Výsledok 5 × 3 = 15 bude pripočítaný do stĺpcov 3 a 2 a výsledok 5 × 4 = 20 bude pripočítaný do stĺpcov 4 a 3. Stĺpec 13 môže byť vynulovaný.
6. Výsledok (2494) je v stĺpcoch 4 až 1.

Popíšte, ako by ste na abakusu násobili 39 × 78

Úloha 1.2. Vytvorte si Napierove kosti z papieru a ukážte na nich násobenie 8x635.

Úloha 1.3. Pri násobení pomocou logaritmov bolo nutné vyhľadať logaritmy dvoch čísel, sčítať ich a nájsť číslo, ktorého logaritmus sa rovnal sume. Toto úsilie bolo redukované nakreslením číselnej osy na ktorej sú pozície čísel proporcionálne ich logaritmom. Na vynásobenie dvoch čísel, začiatok hornej škály čísiel C bol posunutý na hodnotu prvého násobeného čísla na dolnej škále D. Užívateľ našiel druhé násobené číslo na škále C a pozrel odpovedajúce číslo na škále D, ktoré bolo výsledkom násobenia. Chýbalo označenie rádu a presnosť záležala na podrobnosti rozdelenia škál. Napr. 16,6×42,2≅700 sa nájde nasledovne:



Vysvetlite, ako by ste pomocou pravítka vydelili 80:19.

Úloha 1.4. Pomocou prvých štyroch hodnôt tretích mocnín vypočítajte ďalších 6 hodnôt pomocou metódy diferencií. Podobne ako u príkladu pre druhé mocniny znázornite postup výpočtov šípkami.

Úloha 1.5. Majme binárne hodnoty dvoch jednobitových premenných A a B, z ktorých chceme dostať dvojbitovú premennú tvorenú dvojicou jednobitových premenných CD, ktorá bude súčtom hodnôt A a B, teda A+B=CD, konkrétne 0+0=00, 0+1=01, 1+0=01, 1+1=10. (Binárna hodnota 10 odpovedá dekadickej hodnote $1*2^1+0*2^0=2$). Skúste vyjadriť binárne hodnoty bitov C a D (každú osobitne) ako pravdivostné hodnoty získané z pravdivostných hodnôt premenných A a B pomocou operácií AND, OR a NOT.

Úloha 1.6. Navrhnite zo spínačov, batérie a žiarovky obvod, ktorý rozsvieti žiarovku, keď pôjdete na pivo, kedy sa rozhodujete na základe vety "Keď mám čas a buď mám peniaze, alebo mi ich niekto požičia, pôjdem na pivo. Použite A,B,C ako logické premenné, ktorých hodnota je vyjadrená zapnutím alebo vypnutím spínačov, kde A je pravdivostná hodnota výroku "mám čas", B je "mám peniaze" a C je " niekto mi požičia peniaze".

Otázky na opakovanie

Otázka 1.1. Používa sa abakus ešte dnes napr. v Japonsku?

Otázka 1.2. Ako sa prevádza relatívne zložitá operácia násobenia pomocou logaritmov na sčítanie?

Otázka 1.3. Kto vymyslel prvý kalkulátor vyrobený vo viacerých exemplároch?

Otázka 1.4. Kto ako prvý došiel na výhody binárnej číselnej sústavy pri výpočtoch?

Otázka 1.5. V ktorom odvetví priemyslu sa objavil prvý predchodca programu uložený na diernych kartách?

Otázka 1.6. Kto je považovaný za prvého programátora na svete, ktorý napr. objavil význam podmieneného príkazu?

Otázka 1.7. Ako sa dá logická operácia OR zrealizovať pomocou dvoch klopných obvodov?

Otázka 1.8. Koľko vážil prvý elektrónkový počítač ENIAC, a o koľkokrát bol asi pomalší ako dnešné PC?

Otázka 1.9. V čom spočíva Turingov test?

Otázka 1.10. Kto je pokladaný za zakladateľa kybernetiky?

Otázka 1.11. Kedy sa rozšíril prvý "červ" na internete, ako sa líši od vírusu?

Otázka 1.12. Čo znamenajú skratky kb a KB, ako sa líšia významom?

2 Algoritmy a počítanie

Všeobecný pohľad na prácu počítača je taký, že je to „zariadenie“, ktoré transformuje vstupné údaje na výstupné údaje. Tak napríklad, počítačový program pre výpočet koreňov kvadratickej rovnice transformuje koeficienty rovnice na korene, t.j. používa známe formule, ktoré určujú korene rovnice pomocou jej koeficientov. To znamená, že tento proces prebiehajúci v počítači obsahuje tieto dve základné časti:

(1) *reprezentáciu*, ktorá určuje symbolické kódovanie použitej informácie (tak vstupnej, ako aj výstupnej, a samozrejmé aj medzivýsledkov), napr. vo forme čísel alebo mien, a
(2) *transformáciu* (predpis, algoritmus, program), ktorá je použitá na výpočet požadovaného výsledku, t.j. transformáciu vstupných údajov na výstupné údaje.

Všeobecný spôsob zápisu transformačného procesu nám umožňuje jeho použitie na rôzne sady dát - keď sa napríklad naučíme násobiť viacciferné čísla, vieme násobiť akúkoľvek dvojicu takých čísiel. Algoritmus je potom popis tejto transformácie, ktorý je určený nielen pre jeho lepšie pochopenie človekom, ale môže slúžiť taktiež aj ako podklad pre jeho kódovanie (napísanie programu) v tvare, v ktorom je „pochopiteľný“ počítaču. Človek ho potom môže aplikovať priamo (s ceruzkou a papierom) alebo zapísať vo forme programu - kódu konkrétneho programovacieho jazyka, ktorý je automaticky spracovateľný počítačom. Algoritmus ako taký umožňuje vznik technickej pomôcky, ktorá na seba preberá časť procesu. Technické prostriedky, ako sú prístroje a tabuľky, sú teda s algoritmami úzko spojené. Počítač bez algoritmov by bol iba komplikovaným kusom kovu. Zmysel počítačom dávajú len algoritmy, ktoré (algoritmy) sú pomocou nich (počítačov) uskutočniteľné. **Univerzálnosť** algoritmu znamená, že je použiteľný pre riešenie veľkej skupiny úloh toho istého typu, líšiacich sa vstupnými údajmi (tak napr. algoritmus pre hľadanie koreňov kvadratickej rovnice je použiteľný pre každú kvadratickú rovnicu).

Algoritmus je predpis, metóda, alebo technika, ktorý špecifikuje postup úkonov potrebných na dosiahnutie riešenia nejakej úlohy. Ako bolo uvedené, zvyčajne sa pod algoritmom chápe predpis určený pre počítač. Pritom algoritmy, aj tie určené pre výrobu programu do počítača, nemusia byť obmedzené na spracovanie čísel. Také usporiadanie zoznamu mien podľa abecedy je tiež algoritmus. Algoritmom môže ale byť aj recept na bábovku, pokiaľ je dosť presne určený.

Samotné slovo algoritmus pochádza od arabského matematika al-Chwárizmího (v angličtine al-Khowarizmi) z ôsmeho storočia n. l., ktorý vo svojej knihe „*Al-jabr wa'l muqabala*“ (podľa ktorej vznikol názov Algebra) používal desiatkovú sústavu a nulu (čo prevzal z indickej matematiky). Pomocou tejto knihy sa tieto dva významné objavy stredovekej matematiky dostali aj do Európy.

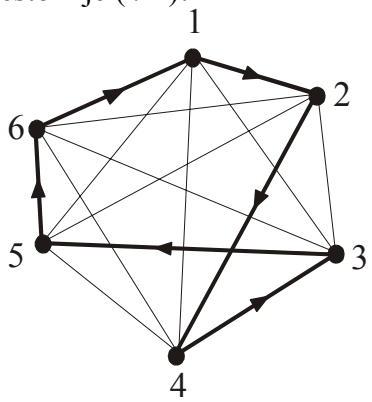
Algoritmus v informatike je jednoznačná, presná a konečná postupnosť operácií, ktoré sú aplikovateľné na množinu objektov alebo symbolov (čísiel, šachových figúrok, ingredencií na bábovku). Počiatočný stav týchto objektov je vstupom, ich koncový stav je výstupom. Operácie odpovedajú zmenám stavu, kedy stav je konfigurácia symbolov alebo objektov,

ktorá sa mení, keď sa na tieto symboly či objekty aplikujú operácie. Operácií je konečný počet, pre jednoduchosť je taktiež výhodné predpokladať, že aj objekty spracovania a vstupy a výstupy sú konečné (aj keď bežne počítame napr. s iracionálnym číslom π, vždy jeho číselnú reprezentáciu obmedzíme pri numerických výpočtoch na konečnú presnosť, napr π=3.14). **Jednoznačnosť** znamená, že každý krok algoritmu musí byť presne definovaný. Nesmie dovoľovať viac výkladov, jednoznačne je určený krok za ním nasledujúci.

U algoritmu by tiež malo byť zaručené, že sa zastaví v konečnom čase. **Rezultatívnosť** znamená, že algoritmus vždy musí po konečnom počte krokov dojsť k nejakému riešeniu.

Ďalšou základnou požiadavkou na algoritmus je **správnosť**, čo znamená, že odpoveď algoritmu je pre každý vstup korektná.

A samozrejme, algoritmus by mal byť efektívny. **Efektívnosť** je možno chápať z rôznych hľadísk, napr. potrebný výpočtový čas a kapacita pamäti nutná na výpočet. Tieto dve hľadiská sú často vzájomne protichodné, vo väčšine algoritmov sa predpokladá, že prístupná pamäť je neohraničená a s nulovým prístupovým časom a uvažuje sa iba výpočtový čas. Algoritmy sa z tohto hľadiska delia na **NP-úplné**[1] a **polynomiálne**. Jedným z najznámejších NP- úplných problémov je napríklad problém obchodného cestujúceho, kde hypotetický obchodný cestujúci má navštíviť *n* miest (každé len raz, pričom na záver sa vracia do východzieho mesta), pričom si musí navrhnúť takú cestu, aby mala minimálnu vzdialenosť. tak, aby sa cestujúci (pozri obr. 2.1). Počet všetkých možných usporiadaní postupností *n* miest s určeným počiatočným mestom je (*n*-1)!



P=(1,2,4,3,5,6)

Obrázok 2.1. Cyklická cesta (tzv. hamiltonovský cyklus) na úplnom grafe, ktorý obsahuje 6 vrcholov (v problému obchodného cestujúceho si vrcholy môžeme predstaviť ako mestá, hrany ako cesty, ktoré môžu byť ohodnotené vzdialenosťou dvojice miest; v grafe záleží iba na tom, či sú mestá prepojené, nezáleží ta tom, ako sa hrana/cesta krúti). Cestu reprezentuje permutácia P=(1,2,4,3,5,6), takáto cesta ale celkom jasne nie je najkratšia.

Keď budeme študovať problém obchodného cestujúceho obsahujúceho napr. 1000 miest, potom nenájdeme najlepšie riešenie, ani keby sme superpočítač nechali bežať miliardy rokov. Povedzme, že máme iba 100 miest a superpočítač dokáže nájsť všetky možnosti za sekundu a pridáme jedno mesto, počet možností sa zvýši 100-krát. Keď pridáme do rozpisu dve mestá, počet možností sa zvýši 100×101 krát ≅ 10000, čo by už počítač riešil takmer 3 hodiny. Aj keď sa snažíme neprehľadávať celkom všetky možnosti až do konca, aj tak najlepší algoritmus má čas úmerný 2^n.

Polynomiálny algoritmus znamená, že čas na výpočet sa dá vyjadriť ako polynóm premennej danej veľkosťou problému. Napríklad aj ten najjednoduchší algoritmus pre zoradenie *n* čísel sa dá vyjadriť polynómom čas_na_výpočet=$a\ n^2 + b\ n + c$ a teda sa o ňom hovorí, že má zložitosť $O(n^2)$. Pre veľmi veľké *n* ale má význam iba najväčšia mocnina, nech sú koeficienty u ostatných mocnín akokoľvek veľké, pre nejakú veľkosť n sa tieto členy stanú bezvýznamné. Ďalšou podstatnou vecou je zanedbanie koeficientu *a* u maximálnej mocniny.

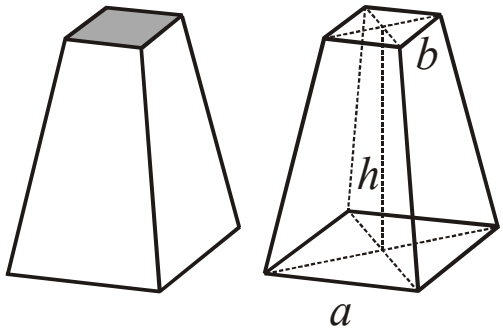
[1] Nondeterministic Polynomially – teda nerozhodnuteľné v polynomiálnom čase.

Tá totiž závisí na tom, ako dobre je algoritmus implementovaný, na akom druhu počítača beží apod. Nás ale u **zložitosti algoritmov** zaujíma, ako sa zvýši výpočtový čas, keď povedzme zdvojnásobíme veľkosť problému. U problému obchodného cestujúceho, ktorý má zložitosť $O(2^n)$, by to bolo zvýšenie 2^n krát (pri prechodu zo 100 na 200 miest sa počet operácií zvýši viac ako 1000000000000000000000000000000 krát). U zoradenia čísel by zvýšenie času nebolo viac ako štyrikrát. Z toho je jasne vidno rozdiel medzi P a NP-úplnými problémami.

Kedy sa objavili prvé algoritmy? Už u starovekých dokumentov z matematiky (napr. zo starého Egypta) sa objavuje prvý algoritmický prístup k riešeniu problémov. Riešenie obsahuje postupnosť príkazov, určujúcich riešiteľovi – užívateľovi, čo musí vykonať. V matematike sa analytické myslenie objavilo až neskoršie u starých Grékov, ale algoritmická forma nikdy celkom nevymizla, a s nástupom počítačov zažíva renesanciu.

Ďalej si uvedieme niektoré známe staroveké algoritmy. Napríklad návod na výpočet objemu zrezaného ihlanu (ihlan so štvorcovou podstavou, ktorého vrchol je zrezaný rovinou rovnobežnou s podstavou, pozri obr. 2.2) je podľa tzv. Moskovského papyrusu (Egypt, 1850 p. n. l.) nasledujúci: Je daná orezaná pyramída, ktorej výška je 6, strana podstavy je 4 a strana hornej základne je 2. Vypočítaj objem tejto pyramídy:

1. Umocni číslo 4 na druhú, dostaneš 16.
2. Číslo 4 zdvojnásob, dostaneš 8.
3. Umocni na druhou číslo 2, dostaneš 4.
4. Tieto čísla 16, 8 a 4 sčítaj, dostaneš 28.
5. Urči tretinu z čísla 6, dostaneš 2.
6. Zdvojnásob číslo 28, dostaneš 56.
7. Celkový výsledok je 56, počítal si dobre.



Obrázok 2.2. Zrezaná pyramída, postup na výpočet objemu tohto telesa vedeli už starí Egypťania.

Aj keď sú príkazy algoritmu zadané s konkrétnymi číslami, ide o návod k výpočtu objemu ľubovolného zrezaného ihlanu. Riešiteľ si iba za dané čísla musí dosadiť svoje konkrétne hodnoty. Použitie symbolov namiesto čísel sa objavilo až u starých Grékov[2]. Moderný algebraický zápis pre zrezaný ihlan s dĺžkou strany a, hornou základnou so stranou b a výškou h by vyzeral ako $V=1/3\ h\ (a^2 + ab + b^2)$.

Egyptský zápis príkazov je ale veľmi podobný príkazom pre počítač vyjadreným dnešným programovacím jazykom.

Aj keď algoritmy vo forme predpisov na výpočty (zväčša zememeračské) boli známe už u starých Egypťanov, prvý pokus formalizovať koncept algoritmu sa dá nájsť v Euklidových *Základoch* (3. storočie p. n. l.), kde boli rozdelené výpočtové procesy na prípustné a neprípustné (napr. pre určitý typ geometrických objektov, ako sú pravidelné

[2] Za prvé použitie symbolickej premennej sa pokladá Aristotelova (žil v 3. storočí p. n. l) formulácia sylogizmov, kde miesto konkrétnych výrokov sa už uvádzali grécke písmená.

mnohouholníky, pripúšťal pri konštrukcii iba konečný počet použitia pravítka a kružidla). V knihe *Základy* uviedol algoritmus pre konštrukciu pravidelného šesťuholníka (6 stranný objekt so všetkými stranami rovnakej dĺžky aj s rovnakými vnútornými uhlami, pozri obr. 2.3): Nech *ABCDEF* je daná kružnica. Úlohou je vpísať rovnostranný a pravidelný šesťuholník do kružnice *ABCDEF*:

1. Nakresli priemer *AD* kružnice *ABCDEF*.
2. Zober stred *G* kružnice. Vytvor kružnicu *EGCH* zo stredom v *D* a polomerom *DG*.
3. Spoj *EG* a *CG* priamkou a nájdi jej priesečníky s kružnicou v bodoch *B* a *F*.
4. Spoj *AB, BC, CD, DE, EF,* a *FA*.

Z úloh, ktoré sú na prvý pohľad primitívne, vychádzajú komplikované problémy. Napríklad o úlohe zadanej Euklidom – kvadratúre kruhu – skonštruovať pomocou kružítka a pravítka štvorec s rovnakou plochou ako má zadaný kruh - bolo až koncom 19 storočia (23 storočí po zadaní úlohy) dokázané, že nemá riešenie.

Známym Euklidovým algoritmom je postup na **výpočet maximálneho spoločného deliteľa** dvoch čísel *m* a *n*:

1. Keď *m* je menšie ako *n*, vymeň ich hodnoty
2. Do *m* daj hodnotu zvyšku po delení *m*/*n* (v modernej terminológii sa to zapisuje ako `m:=m modulo n`)
3. Keď sa *m* nerovná nule, choď na krok 1 s novými hodnotami *m* a *n*
4. Vráť *n* ako výsledok

Ďalším známym starovekým algoritmom je **Eratosthenovo sito** (z 3 storočia p. n. l.) na nájdenie všetkých prvočísiel menších ako *n* (prvočíslo je celé číslo väčšie ako 1, ktoré je bezo zvyšku deliteľné iba sebou samým a číslom 1, v súčasnej dobe sa prvočísla často využívajú napr. pri šifrovaní správ). Výsledok algoritmu pre *n*=50 je na obr. 2.3, Eratosthenov predpis znie:

1. Zapíš do zoznamu všetky čísla od 2 do *n*
2. Nech *k*=2
3. Pre každé číslo *m* medzi *k*+1 a *n* skontroluj, či je *m* presným násobkom *k*, keď áno, vyškrtni *m* zo zoznamu.
4. Do *k* daj najmenšie ešte nevyškrtnuté číslo zo zoznamu
5. Keď *k* je menšie ako *n*, opakuj proces od kroku 3
6. Akékoľvek číslo zo zoznamu, ktoré nebolo vyškrtnuté, je prvočíslo

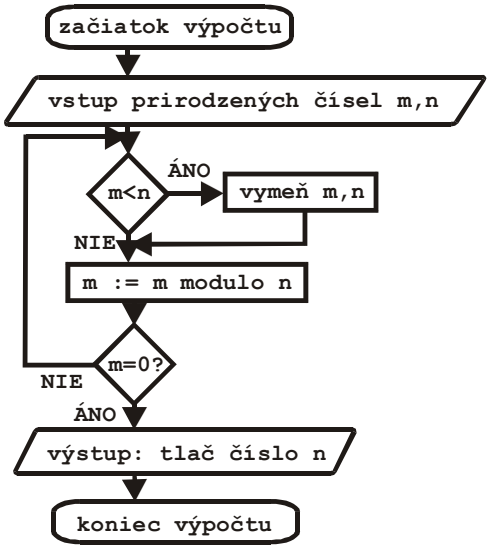
	~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
deliteľné	def.			2		2		2	3	2
	11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
deliteľné		2		2	3	2		2		2
	~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
deliteľné		2		2		2		2		2
	31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
deliteľné		2	3	2	5	2		2	3	2
	41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~
deliteľné		2		2	3	2		2	7	2

Obrázok 2.3. Výsledok Eratosthenova sita pre n=50, zložené čísla sú prečiarknuté, ich delitelia sú uvedení v riadku pod nimi, 1 je prečiarknutá podľa definície. Neprečiarknuté ostávajú prvočísla.

Uvedené algoritmy sa dajú uskutočniť s ceruzkou a papierom, ale iba pre malé problémy, pre väčšie už musíme používať počítač. Aby sme donútili počítač vykonať nejakú úlohu, musíme mu dodať program, čo je implementácia algoritmu. Program je vyjadrený v konkrétnom programovacom jazyku. Je ale možné a aj žiadúce, hovoriť o algoritmoch v "termínoch vyššej úrovne" ako sú príkazy konkrétneho počítačového jazyka. Vývoj algoritmu môže predstavovať intelektuálnu výzvu, zatiaľ čo jeho naprogramovanie by malo byť priamočiare (aj keď urobiť to dobre nemusí byť triviálne).

Najčastejšie používaným zápisom algoritmov sú **vývojové diagramy** (flowcharts) a **pseudokód** (pseudo-code). Tieto spôsoby zápisu sú nezávislé na programovacom jazyku. Vývojový diagram je „grafická štruktúra“ obsahujúca orientované čiary reprezentujúce cesty dát v programe. Začiatok a koniec programu sa označujú oválom, vstup a výstup kosodlžnikom, príkazy/operácie obdĺžnikom a vetvenie programu na základe pravdivosti splnenia nejakej podmienky sa označuje rozhodovacím blokom v tvare kosoštvorca.

Pseudokód je druh “štylizovaného” (alebo “štruktúrovaného”) prirodzeného jazyka. Napríklad, príkaz "`rob X dokiaľ Y`" tvorí "slučku", dá sa preložiť voľnejšie ako "rob príkazy X dokiaľ je splnená podmienka Y". Slučka znamená sadu príkazov, ktoré sa dookola opakujú. "`Keď Y`" znamená "keď je splnená podmienka Y urob nasledujúci príkaz". Pseudokód v obr. 2.4 je aj "štruktúrovaný", t.j. príkazy X vo vnútri slučky sú posunuté doprava, aby bolo vidno, že sú vo vnútri slučky.




```
Vstup dvoch celých čísel: m, n
  rob
    keď m < n vymeň m,n
    m := m modulo n
  dokial n≠0
  vytlač n
```

Obrázok 2.4. Vývojový diagram a pseudokód Euklidovho algoritmu na výpočet maximálneho spoločného deliteľa dvoch čísel m a n.

Ďalej, tak u algoritmov, ako aj u hotových programov je užitočné navrhnúť taký postup riešenia, aby bol ľahko modifikovateľný pri zmene podmienok riešenia. S tým úzko súvisí aj štruktúrovanosť algoritmu. Algoritmus, ktorý je rozdelený na relatívne samostatné, logicky nadväzujúce celky, sa ľahšie pochopí a upravuje. Okrem toho je aj možné zadať jednotlivé časti na naprogramovanie rôznym ľuďom a potom iba zložiť výsledný program z jeho častí. Vyššou úrovňou tohto prístupu je objektovo orientované programovanie, ktoré sa ale pre svoju náročnosť zvyčajne ponecháva na profesionálnych programátorov (napr. v jazyku C++).

V praxi je tiež dôležité, ako často a za akým účelom algoritmus robíte. Keď výsledný program chcete spúšťať iba niekoľkokrát, väčšinou nezáleží na tom, či potrvá sekundy alebo minúty, pokiaľ použitím jednoduchšieho, ale menej efektívneho algoritmu ušetríte váš vlastný čas prípravy programu.

Základom algoritmov je **podmienka**, vzťah relácie medzi dvojicou premenných, konštánt alebo výrazov vyjadrených pomocou relačných operátorov $=$, $<$, $>$, $\leq$, $\geq$, $\neq$. Takéto relácie môžu byť prípadne aj zreťazené pomocou logických spojok konjunkcie (logická spojka AND), disjunkcie (logická spojka OR) a negácie (logická spojka NOT). Výsledkom vyhodnotenia podmienky je jej pravdivostná hodnota „pravda“, keď je podmienka splnená, alebo „nepravda“, keď nie je splnená. Podmienka sa potom používa v rozhodovacom kroku, vetvení programu, kedy sa pri splnenej podmienke robí jedna sekvencia krokov a pri nesplnenej podmienke sa robí iná sekvencia krokov, skáče sa na iné miesto v programe. Skok je prerušením prirodzenej postupnosti priechodu algoritmom a prenesenie spracovania na iné miesto označené návestím.

Pomocou podmienky sa vytvára aj cyklus (slučka, iterácia), keď sa určitá postupnosť krokov opakuje, pokiaľ je splnená podmienka opakovania cyklu. Väčšinou sa cyklus opakuje n-krát, riadiaca premenná i sa zo začiatku nastaví na jednotku a v cyklu sa k i pripočítava jednotka, a cyklus sa ukončí, keď i nadobudne hodnotu rovnú n.

Najpoužívanejšími algoritmami sú algoritmy na **vyhľadávanie a triedenie**. Tu sa dajú aj ukázať rôzne efektívne prístupy k rovnakému problému.

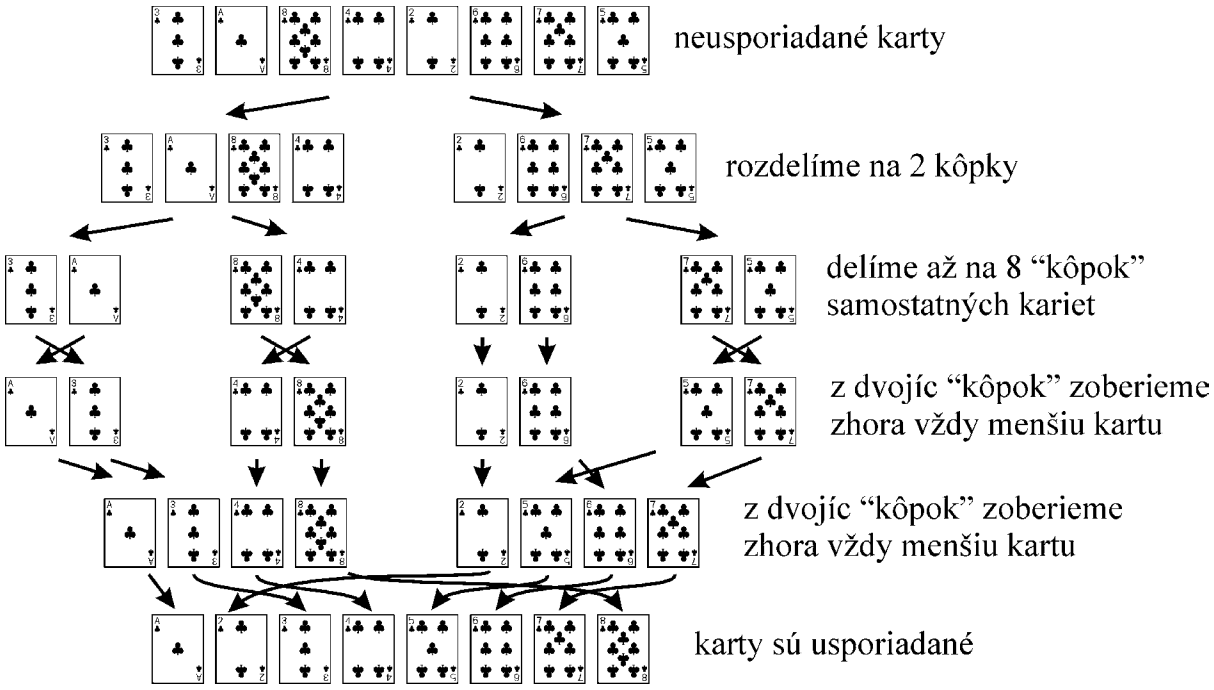
Povedzme, že poznáme telefónne číslo, a máme k nemu vyhľadať v telefónnom zozname meno. To sa nedá urobiť inak, ako pozrieť sa postupne na každé telefónne číslo a porovnať ho s hľadaným číslom, čo sa volá **sekvenčné vyhľadávanie**. Každý ale vie, že keď hľadá v zozname meno, neprehľadáva všetky mená od začiatku do konca. Pokiaľ ani nevieme, ktorý zväzok telefónneho zoznamu máme v ruke, pozrieme sa na začiatok a na koniec zoznamu. Keď máme v ruke správny zväzok, pozrieme sa do polovice zoznamu. Keď je hľadané meno abecedne vpredu, pozrieme sa na meno v polovici prvej polovice (teda v štvrtine) zoznamu, v opačnom prípade sa pozrieme na meno v polovici druhej polovice (teda do troch štvrtín) zoznamu. Takto stále rozdeľujeme zvyšnú časť, ktorú máme prehľadávať, na polovice, dokiaľ nenájdeme hľadané meno. To je **binárne vyhľadávanie**, vtedy počet potrebných porovnaní na nájdenie ľubovoľného mena je hornou celou časťou $\log_2 N$, kde N je počet mien v zoznamu. Keď máme vyhľadať 100 ľudí v zozname občanov Slovenska o 5000000 menách, a porovnanie trvá povedzme 0,000001 s, namiesto takmer päťminútového sekvenčného vyhľadávania nám binárne vyhľadávanie nájde všetky výsledky celkom za 0,002s. Takto sa dá rýchlo vyhľadávať v akomkoľvek utriedenom zozname.

Ako ale zoznam zo začiatku utriediť? Algoritmy triedenia si môžeme ukázať na balíčku kariet. Snáď najpomalšie ale najjednoduchšie na pochopenie je **triedenie priamym výberom**, kde karty v priebehu usporiadania delíme na dve kôpky, jednu už utriedenú a druhú ešte neutriedenú. Na začiatku samozrejme máme iba neutriedenú kôpku. V každom kroku výpočtu vždy zoberieme kartu s najmenšou hodnotou z neutriedenej kôpky a dáme ju na utriedenú kôpku, dokiaľ nie sú všetky karty utriedené. Najmenšiu kartu z neutriedenej kôpky

vyberáme tak, že zoberieme prvú kartu zhora a postupne ju porovnávame zo zvyšnými kartami. Keď je porovnávaná karta z kôpky menšia, vymeníme karty a pokračujeme v porovnávaní kariet až do konca kôpky. Postupne tak pre n kariet vykonávame n-1 +n-2 ...+1=$n(n-1)/2$ porovnávaní.

Existuje ale oveľa rýchlejší algoritmus, volaný **triedenie zlučovaním**. Toto triedenie používa všeobecnejšiu techniku návrhu algoritmov nazývanú podľa zásady rímskych cisárov „**rozdeľ a panuj**“, ktorá spočíva v rozdelení riešeného problému na dva alebo viac nezávisle riešiteľných problémov rovnakého druhu, ako bol pôvodný problém. Tieto problémy ďalej delíme na ešte menšie, až dôjdeme na problémy, ktoré už riešiť vieme priamo. Riešenie týchto malých problémov potom skladáme po úrovniach, dokiaľ nezložíme celkový výsledok. Tento prístup sa nedá použiť všade, vedie ale ku krátkym a väčšinou efektívnym algoritmom rekurzívneho charakteru[3].

Triedenie zlučovaním realizujeme takto: ak je na kôpke iba jedna karta, tá je už usporiadaná, v opačnom prípade rozdeľ zásobník na polovicu, každú polovicu usporiadaj a u dvojice usporiadaných kôpok porovnaj vrchné karty a vyber do výsledné kôpky vždy kartu s menšou hodnotou (pozri obr. 2.5). Pri tomto usporiadaní polovíc kôpok sa používa triedenie zlučovaním, akoby sme začínali od začiatku (podobne utriedime polovice polovíc, atď. – to je podstatou **rekurzie**). Počet porovnaní tohto triedenia pre n kariet (alebo čísel, alebo čohokoľvek iného, čo sa dá usporiadať) je úmerný $n \log n$. Keď máme usporiadať zoznam o 5000000 menách, a porovnanie trvá povedzme 0,000001 s, triedenie priamym výberom by trvalo takmer pol roka, triedenie zlučovaním by trvalo pol minúty.



Obrázok 2.5. Postup triedenia zlučovaním pre 8 kariet. Najprv balíček delíme na polovice, dokiaľ nedostaneme jednotlivé karty. Potom vždy dvojice kôpok utriedime zlučovaním, dokiaľ nedostaneme jednu usporiadanú kôpku. Karty na kôpkach sú tu zobrazené vedľa seba.

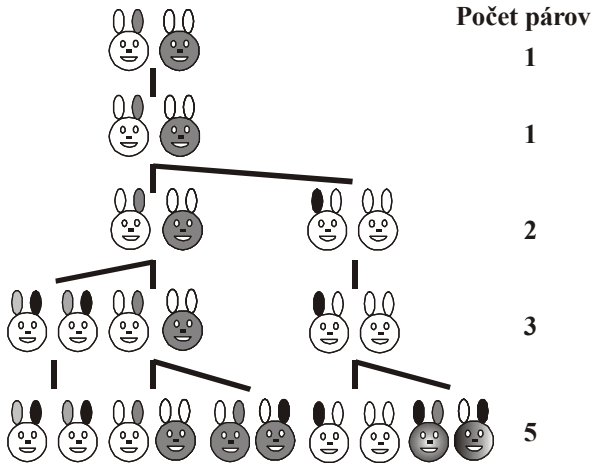
Ukážeme si teraz príklad, keď nie je technika „rozdeľ a panuj“ výhodná. Použijeme na to výpočet **Fibonacciho čísla**. Fibonacci, tiež známy ako Leonardo z Pisy, bol jeden

[3] Rekurzívna funkcia je taká, ktorá volá sama seba s upravenými vstupnými parametrami. Potrebuje v sebe ale mať ako prvý príkaz kontrolu, či už sa pre zadaný vstupný parameter nedá problém riešiť priamo. V takom prípade by funkcia už nevolala seba, ale vrátila by výsledok.

z najvýznačnejších stredovekých matematikov. R. 1202 zadal nasledujúci problém: Králičí samec a samica sa narodia na počiatku roka. Predpokladáme nasledujúce podmienky:

1. Králičí páry nie sú plodné v priebehu prvého mesiaca života, ale potom porodia nový pár samec-samica na konci každého mesiaca.
2. V priebehu roka nedochádza k úmrtiu králikov.

Koľko králikov bude na konci roka?



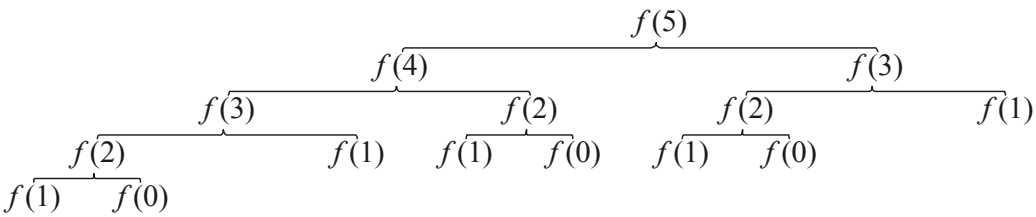
Obrázok 2.6. Výpočet piateho Fibonacciho čísla $f(5)$ alebo rozmnožovanie králikov za 5 mesiacov.

Postupne bude na konci jednotlivých mesiacov 1,1,2,3,5,8,13,21,34,55,89,144 králikov (pre začiatok postupnosti pozri obr. 2.6). Čísla v tejto postupnosti boli pomenované na Fibonacciho počesť. Majú veľa zaujímavých a zvláštnych vlastností, napríklad u mnohých kvetín je počet okvetných lístkov rovný jednému z Fibonacciho čísel, tieto čísla je možné pozorovať v usporiadaniu semien u slnečníc, pre n idúce do nekonečna dáva pomer Fibonacciho čísel $F(n+1)/F(n)$ tzv. zlatý rez 1.618…, ktorý starovekí Gréci považovali u pomeru dĺžky strán obdĺžnika za optimálny.

Pre $n \geq 0$ sa tieto čísla dajú zadefinovať ako

$$f(0)=0;\ f(1)=1;\ f(n)=f(n-1)+f(n-1) \text{ pre } n>1.$$

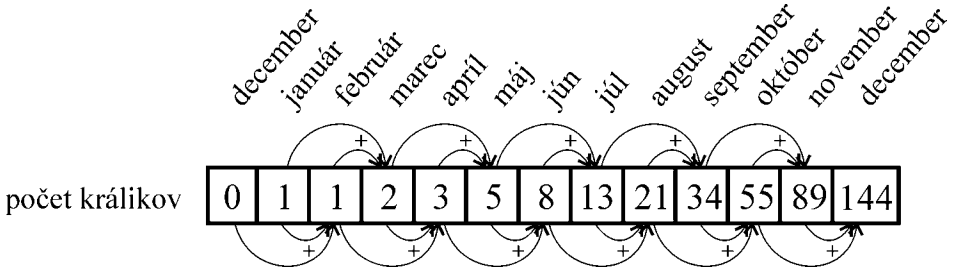
Tieto vzťahy sa jednoducho implementujú rekuzívnym algoritmom. Pokiaľ sa ale pozrieme na obr. 2.8, kde je uvedené schéma rekurzivních volaní funkcie, zistíme, že pri riešení úlohy dochádza k opakovaniu rovnakého výpočtu zbytočne niekoľkokrát, napríklad Fibonacciho číslo $f(2)$ je počítané trikrát. To je podobné u viacerých nevhodných použití rekurzie. Použitie rekurzívneho prístupu rozdeľ a panuj sa dá tiež nazvať prístupom zhora nadol. Pôvod tohto názvu je vidno aj v schéme na obr. 2.7, kde začíname zhora s problémom, ktorý máme riešiť, teda s výpočtom piateho Fibonacciho čísla $f(5)$, ktorého výpočet sa nám rozpadá na menšie podproblémy.



Obrázok 2.7. Schéma výpočtu piateho Fibonacciho čísla $f(5)$ rekurzívnym volaním

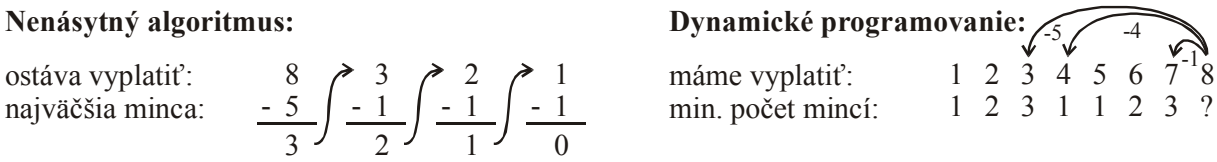
Opačným prístupom, ktorý vypočíta Fibonacciho číslo efektívnejšie, je prístup zdola nahor, kedy začneme s riešením najjednoduchších problémov a krok za krokom vybudujeme

riešenie zložitejších problémov, dokiaľ nedôjdeme na riešenie problému, ktorý hľadáme. V danom prípade je prístup totožný s prístupom nazvaným **dynamické programovanie**. Jednoducho v tomto prípade ideme po mesiacoch od prvého do posledného, pre nultý a prvý mesiac (pozri definíciu Fibonacciho čísla) sú počty určené a od druhého pre každý ďalší mesiac vypočítame počet králikov súčtom hodnôt predchádzajúcich dvoch mesiacov, dokiaľ nedôjdeme do želaného mesiaca, pozri obr. 2.8.



Obrázok 2.8. Schéma výpočtu Fibonacciho čísla *f*(12) dynamickým programovaním. Prvé dve hodnoty 0,1 sa vyplnia podľa definície, ďalej sa postupne počítajú čísla pre jednotlivé mesiace súčtom predchádzajúcich dvoch. Na rozdiel od rekurzívneho prístupu sa žiadna z hodnôt nepočíta zbytočne viackrát.

Okrem prístupu „rozdeľ a panuj“ a dynamického programovania uvedieme ešte zaujímavú techniku a to **„nenásytný“ (greedy) algoritmus**. Tento algoritmus opakuje procedúru, ktorá buduje riešenie postupným použitím momentálne najlepšieho prvku s nádejou, že riešenie zložené z lokálne najlepších prvkov bude aj globálne najlepšie. V niektorých prípadoch tento prístup zaručuje optimálne riešenie, v iných iba prijateľne dobré riešenie vypočítané za krátku dobu.



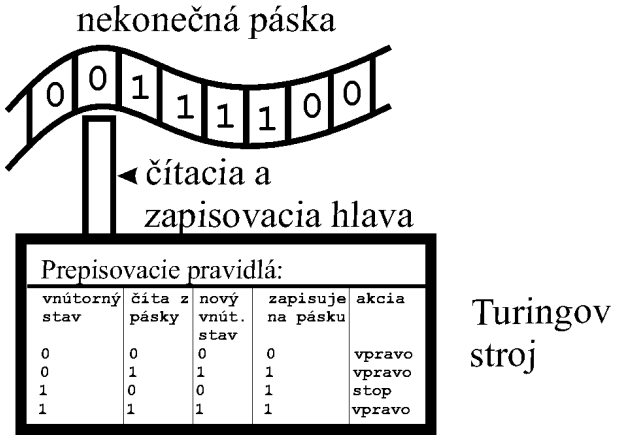
Obrázok 2.9. Problém pokladníčky s štvorkorunáčkou: je treba vyplatiť 8 Sk najmenším počtom mincí hodnôt 5 Sk, 4 Sk a 1 Sk. Nenásytný algoritmus vyplatí vždy najväčšiu mincu menšiu ako zvyšný dlh, teda mince hodnôt 5, 1,1,1. Dynamické programovanie nájde postupne minimálne počty mincí pre výplatu všetkých nižších hodnôt od 1 po 7 a až potom sa pokúša rovnakým spôsobom o nájdenie min. počtu mincí na výplatu 8 Sk. Od hodnoty 8 sa odpočítajú všetky prípustné hodnoty mincí (1,4 a 5) a pre získané hodnoty 7,4 a 3 sa pozrieme, koľkými mincami sa dali vyplatiť. Keďže hodnota 4 sa dala vyplatiť najmenším počtom mincí (jednou) a jednu mincu sme spotrebovali na prechod od hodnoty 8 Sk na 4 Sk, 8 Sk sa dá vyplatiť dvoma mincami.

Nenásytný algoritmus si ukážeme na problému pokladníčky, ktorá sa snaží vyplatiť zadanú sumu za použitia minimálneho počtu mincí. Nenásytný algoritmus problém rieši tak, že postupne vypláca vždy čo najväčšiu mincu, ktorá nepresahuje sumu, ktorú treba ešte zaplatiť. To ale nemusí byť ideálne. Predstavme si, že máme mince v hodnotách 1, 4 a 5 a máme vyplatiť hodnotu 8. Nenásytný algoritmus by najprv zobral mincu s hodnotou 5 a zvyšok by sa musel doplatiť troma mincami s hodnotou 1. Celkovo sa tak spotrebovali 4 mince, zatiaľ čo sme mohli použiť celkovo iba dve mince o hodnote 4. Problém sa dá lepšie riešiť dynamickým programovaním, kedy sa pre platbu o hodnote n postupne snažíme nájsť najmenší počet mincí pre všetky hodnoty platieb 1, 2, 3, …n. Postupujeme tak, že od momentálne skúmanej platby odpočítavame všetky možné hodnoty jednotlivých mincí a pozeráme sa do nami už urobeného zoznamu, koľko mincí sa spotrebovalo na tú zmenšenú platbu. Z možných platieb získaných odpočítaním jednej mince si vyberieme tú, ktorej

odpovedá najmenší počet mincí, k tomuto počtu mincí pripočítame jednotku a zapíšeme výsledok ako počet mincí pre vyplatenie momentálne skúmanej platby.

Môžeme si položiť otázku, dá sa počítačom veriť? Ako si môžeme overiť výsledok výpočtu čísla. π vypočítaného na milión desatinných miest? Aj keď sa pred niekoľkými rokmi objavila v procesore INTEL chyba jeho architektúry, spôsobujúca v špeciálnych prípadoch zlý výpočet, dá sa povedať, že počítačom samotným sa v podstate dá veriť, že urobia presne to, čo im povieme v algoritme. Problém je v tom, že algoritmus samotný býva veľakrát alebo zle navrhnutý, alebo zle prepísaný do počítačového jazyka. Jednou z možností riešenia tohto problému je overenie algoritmu používaného pri výpočtu, kedy samotný program musí byť natoľko jednoduchý, aby sa dalo posúdiť, či naozaj prikazuje vykonať to, čo má. Metódy formálnej analýzy algoritmov sú ale stále ešte v začiatkoch a nedajú sa použiť u rozsiahlych algoritmov z praxe. Ďalšou možnosťou je výpočet daného problému viacerými alternatívnymi cestami a porovnanie výsledkov.

Formálne sa dá algoritmus chápať ako špeciálny typ funkcie - vypočitateľnej funkcie, ktorej vstupy aj výstupy môžu byť produkované tzv. Turingovým strojom. **Turingov stroj** je formálny model počítača, ktorý žiadna firma nikdy nevyrobila.. Takýchto formálnych modelov počítača existuje viac, Turingov stroj je iba jedným z najčastejšie používaných (Alan Mathison Turing bol britský matematik, žijúci v r. 1912-1954). Ide o logický model s jednoduchým usporiadaním základných operácií, ktoré môže vykonávať človek s tužkou a papierom. Postup výpočtu Turingova stroje je pomalý a ťažkopádny, lenže on ani nemá ambície byť rýchly a efektívny. Dôležité je, že všetko, čo sa dá spočítať ľubovolným dosiaľ zostrojeným počítačom[4], ale aj ľubovolným počítačom, o ktorého zostrojenie sa ľudia v budúcnosti len pokúsia, sa dá teoreticky spočítať taktiež Turingovým strojom. Je to automat s konečným počtom stavov, vybavený nekonečnou 1-rozmernou páskou, na ktorej je možné uchovávať informáciu zapisovaním do políčok. Pravidlá sú typu: ak si v stave ‘0‘ a na aktuálnom poličku pásky je symbol '1', zmeň svoj stav na '1', prepíš políčko na '1' a presuň sa o jednotku doprava (pozri obr. 2.10). O takom stroji sa dajú dokazovať matematické vety.



vnútorný stav	číta z pásky	nový vnút. stav	zapisuje na pásku	akcia
0	0	0	0	vpravo
0	1	1	1	vpravo
1	0	0	1	stop
1	1	1	1	vpravo

Obrázok 2.10. Turingov stroj, ktorý pričíta jednotku k unárnemu číslu (prirodzenému číslu *n* odpovedá unárne číslo rovné *n* jedničkám, napr. 2=11, 5=11111). Tento stoj s počiatočným vnútorným stavom 0 ide doprava (akcia "vpravo") po nulách na páske. Až narazí na reťazec jednotiek, dôjde na jeho koniec, prepíše prvú nasledujúcu nulu na jednotku a zastaví sa (akcia "stop").

[4] Počítače, s ktorými sa denne stretávame, sú univerzálne. Pri dostatku času, pamäte a správnom softvéri môžeme každým počítačom simulovať ľubovoľný iný počítač alebo fyzické výpočtové zariadenie. To znamená, že počítač zo správnym softvérom by mal byť schopný simulovať aj výstupy ľudského mozgu. Analógový signál z mozgu (teda reálne číslo) má síce teoreticky nekonečný počet hodnôt, ale spracováva sa iba s obmedzenou presnosťou, ktorá sa dá simulovať zvýšením počtu bitov kódujúcich reálne číslo.

Každý vypočitateľný problém sa dá vyriešiť zapísaním symbolov určujúcich nielen zadanie problému, ale aj metódu jeho riešenia na pásku. Potom spustíme na páske Turingov stroj, ktorý sa na nej posúva vpred a vzad, číta, zapisuje, či prepisuje symboly, a keď sa zastaví, na páske bude zapísané riešenie.

Problém, či je nejaká **funkcia vypočitateľná**, alebo či funkcia bude pre niektoré vstupné hodnoty nedefinovaná, sa dá ukázať na jednej z dosiaľ nevyriešených matematických záhad - problému prvočíselných dvojčiat. To sú dvojice prvočísiel líšiacich sa o 2. Napr. 3 a 5, alebo 17 a 19, alebo 1.000.000.007 a 1.000.000.009. Boli nájdené oveľa väčšie prvočíselné dvojčatá. Nikto však nedokázal, že by žiadne ďalšie prvočíselné dvojča neexistovalo. Nebol by problém napísať program, aj pomocou inštrukcií Turingovho stroja, ktorý by rad za radom bral všetky dvojice čísel a známym algoritmom overoval, či sa jedná o prvočísla. Pokiaľ by sme mali šťastie, program by na prvočíselné dvojča narazil. Pokiaľ ale program bude počítať a počítať, nevieme, či je to tým, že už žiadne ďalšie prvočíselné dvojča neexistuje, alebo tým, že sme ešte nevyskúšali všetky možné čísla.

Tento problém je svojou podstatou príbuzný tzv. **problému zastavenia** (halting problem): Máme vymyslieť univerzálny algoritmus, ktorý by na základe zdrojového textu programu napísaného v nejakom programovacím jazyku, zistil, či program skončí výpočet, alebo sa zacyklí a "spadne". Štúdium problému zastavenia Turingovho stroja ukázalo, že to sa nedá. Predstavte si dva programy, Test zastavenia a Opačný test. Test zastavenia nám povie, či sa program, ktorý je jeho vstupnými dátami (kľudne môže byť zadaný ako sada núl a jedničiek), sa sám zastaví. Program Opačný test bude obsahovať Test zastavenia ako podprogram, a keď Test zastavenia odpovie, že sa vstupný program zacyklí, Opačný test skončí, keď Test zastavenia povie, že vstupný program skončí, Opačný test sa zacyklí. Test zastavenia zlyhá, keď ho spustíme nad programom Opačný test, a teda nepracuje so všetkými programami.

Existujú teda veľmi praktické problémy, na ktoré počítače nemôžu z principiálnych dôvodov stačiť[5]. Fakt, že existujú algoritmicky neriešiteľné problémy, ako bolo uvedené napr. u problému zastavenia, ale neznamená, že by sme sa nemali prestať pokúšať urobiť počítače "inteligentnejšími".

Keď sa pozeráme na efektívne vykonaný výkon, ktorý vyžaduje niečo, čo bežne voláme "inteligenciou" (napr. hru v šachy), považujeme hráča za inteligentného automaticky aj v iných oblastiach, aj keď to vôbec nemusí odpovedať skutočnosti. Pretože počítače môžu byť naprogramované tak, aby zvládali úkoly, ktoré bežne považujeme za intelektuálne náročné, je často laikmi automaticky predpokladané, že sú "inteligentné". Tento prístup k počítačom je podporovaný faktom, že počítače môžu byť nastavené na plnenie rôznych na sebe nezávislých úloh a že v určitom zmysle pracujú nezávisle, bez ľudskej kontroly. Počítače sa už začínajú podobať človeku aj v tom, že ich algoritmy využívajú aj náhodné čísla a heuristiky, ktoré nemusia viesť k absolútne najlepšiemu riešeniu. To sa deje predovšetkým u veľmi zložitých problémov, kde nie je prakticky možné prehľadať všetky možnosti riešení – a je to aj dôvodom, prečo šachový veľmajster má stále ešte šancu počítač poraziť.

Cieľ, aby sa počítače stali skutočne inteligentnými, si dala oblasť informatiky nazvaná "umelá inteligencia" (pozri nasledujúcu 3. kapitolu). Zlá prezentácia reálnych možností umelej inteligencie prispela k pohľadu laickej verejnosti na počítače ako na hrozbu na jednej strane a "inteligentný" stroj na strane druhej. Pri súčasnom stave vývoja počítačov a umelej inteligencie nebude ani jeden z týchto pohľadov ospravedlniteľný prinajmenšom ešte desiatky rokov (či vôbec kedy dosiahne kvalita inteligencie počítača kvalitu bežne rozmýšľajúceho človeka je stále kontroverznou otázkou).

[5] Ďalšou nevypočítateľnou funkciou je rozhodovanie, či je daný matematický výrok pravdivý. Kurt Gödel dokázal r. 1931, že v každom dostatočne „bohatom“ a vnútorne konzistentnom matematickom systéme existujú výroky, ktoré sa nedajú dokázať ani vyvrátiť.

V netradičných prístupoch k výpočtom sa algoritmy spolu s počítačmi teraz rozvíjajú predovšetkým v oblasti neurónových sietí, evolučných algoritmov, fuzzy logiky a kvantových výpočtov (tie majú svoj základ v kvantovej fyzike, a sú zatiaľ iba v štádiu počiatočných experimentov).

Zhrnutie

Algoritmus je jasne definovaný postup na získanie riešenia problému. Mal by byť použiteľný pre riešenie veľkej skupiny úloh, musí dojsť k nejakému riešeniu, a to pokiaľ možno efektívne. Algoritmy sa z tohto hľadiska delia na NP-úplné a polynomiálne. Prvé algoritmy boli zo starého Egypta, ale až Gréci začali používať symbolov namiesto čísel. Známy je Euklidov algoritmus na nájdenie maximálneho spoločného deliteľa alebo Eratosthenovo síto na prvočísla. Algoritmy sa zapisujú napr. vývojovými diagramami alebo pseudokódom. Najčastejšie používanými algoritmami sú vyhľadávanie (sekvenčné, binárne) a triedenie. Mnohé algoritmy sú založené na rekurzii, metóde „rozdeľ a panuj“ alebo dynamickom programovaní či "nenásytnom" prístupe. Na skúmanie teoretických aspektov algoritmov sa užíva Turingov stroj; existujú aj nevypočítateľné problémy, napr. problém zastavenia.

Úlohy

Úloha 2.1. Zápis Euklidovho algoritmu po krokoch je najstručnejším slovným zápisom, ale po skoku na krok 1 obsahuje jednu zbytočnú kontrolu. Skúste navrhnúť vývojový diagram a pseudokód, ktorý bude efektívnejší.

Úloha 2.2. Vytvorte vývojový diagram pre Eratosthenovo sito.

Úloha 2.3. Vytvorte vývojový diagram ľubovoľného (nezmyseľného) programu obsahujúceho nekonečnú slučku.

Úloha 2.4. Vytvorte vývojový diagram pre nájdenie najväčšieho čísla zo sady zadaných čísel.

Úloha 2.5. Nájdite vylepšenie v kapitole uvedeného Eratosthenova síta tak, aby ste nemuseli skúšať deliť všetkými nevyčiarknutými číslami v zozname.

Úloha 2.6. Spočítajte, koľko porovnaní dvojice kariet bolo treba u triedenia zľučovaním na obr. 2.6 a koľko porovnaní by bolo treba pri triedení priamym výberom. Diskutujte, či u týchto metód záleží počet nutných porovnaní na počiatočnom usporiadaní kariet.

Úloha 2.7. Zmeňte pravidlá pre Turingov stroj z obr. 2.10 tak, aby nepridal jednotku len k prvému reťazcu jednotiek, ale aby pokračoval v hľadaní ďalších reťazcov jednotiek.

Úloha 2.8. Skúste posúdiť, či pažravý algoritmus nájde vždy najmenší počet platidiel na výplatu danej sumy pri súčasných hodnotách slovenských bankoviek a mincí.

Otázky na opakovanie

Otázka 2.1. Čo je to algoritmus?

Otázka 2.2. Odkiaľ pochádza slovo algoritmus?

Otázka 2.3. Dá sa celkom presne numericky počítať aj s iracionálnymi číslami, ako je napr. Eulerovo číslo *e*, základ prirodzených logaritmov? Dá sa nazvať postup na výpočet všetkých desatinných miest čísla *e* algoritmom? Diskutujte prečo áno či nie.

Otázka 2.4. Keby ste mali vypočítať všetky možné poradia výsledkov hromadného závodu v behu, ktorého sa účastní niekoľko tisíc účastníkov, ako dlho by vám to na počítači približne trvalo? Ako dlho by počítaču rádovo trvalo, keby ste mali mená týchto pretekárov usporiadať podľa abecedy?

Otázka 2.5. Čo je považované za efektívnejší algoritmus v závislosti na veľkosti problému *n*, ten, čo spočíta výsledok za čas 1000000 n^2, alebo za 0,001 n^3? Diskutujte, kedy by ste vy použili ktorý z týchto algoritmov.

Otázka 2.6. Dá sa pomocou kružítka a pravítka skonštruovať štvorec o rovnakom obsahu ako zadaný kruh?

Otázka 2.7. V akej oblasti sa v súčasnosti prakticky používajú prvočísla?

Otázka 2.8. Čo je to rekurzívna funkcia?

Otázka 2.9. Kto bol Fibonacci, a ako súvisí so zlatým rezom známym z umenia?

Otázka 2.10. Keď máte naplniť prepravnú debnu čo najmenším počtom kusov tovaru tak, aby ste dostali presne *x* kg váhy, a máte druhy tovaru o rozdielnych váhach, je vždy výhodné začať s najťažším druhom tovaru, ktorého váha je menšia ako *x*?

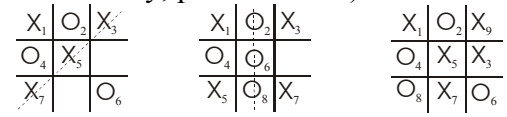
Otázka 2.11. Ako rýchlo počíta typická implementácia Turingovho stroja a kto ho vyrába?

Otázka 2.12. Čo je ľahšie naprogramovať, šachový program, ktorý by vždy porazil aj šachových veľmajstrov, alebo program, ktorý vždy správne zistí, ako dlho bude iný program počítať

3 Umelá inteligencia

Umelá inteligencia (UI) [1-5] je tá časť informatiky, ktorá sa zaoberá problémami obtiažne formulovateľnými a riešiteľnými. Marvin Minsky, jeden zo zakladateľov UI, ju charakterizoval ako vedu, ktorá chce naučiť počítače riešiť problémy, ktoré, ak sú riešené ľuďmi, vyžadujú inteligenciu. V súčasnosti máme dva rôzne prístupy k UI. *Prvý prístup* chápe UI ako inžiniersku disciplínu, ktorá sa snaží urobiť počítače inteligentnejšími. Počítače by napríklad mali komunikovať v prirodzenom jazyku, samostatne riešiť problémy, vyvodzovať nové poznatky, učiť sa na základe svojej predchádzajúcej aktivity a mali by zvládať mnohé ďalšie nemenej zaujímavé problémy. *Druhý prístup* chápe UI ako experimentálnu informatickú disciplínu, ktorá výpočtovými prostriedkami modeluje ľudskú inteligenciu. Pre oba prístupy je spoločné to, že sa snažia riešiť problém ľudskej inteligencie počítačovými prostriedkami. Ako vzdialený cieľ pre oba prístupy je implementácia počítačovej analógie ľudskej mysle, superprogramu, ktorý bude vykazovať podobné aktivity a schopnosti ako ľudský mozog. V tejto kapitole nebudeme diskutovať, či sa takýto program vôbec dá principiálne vytvoriť a aké prekážky musíme zvládnuť pri jeho implementácii. Čitateľa v tomto ohľade odkazujeme na kapitolu 13, ktorá je venovaná kognitívnej vede. Prístupy k štúdiu UI sa delia na dve veľké triedy: (1) štúdium metód riešenia problémov a (2) štúdium reprezentácií vedomostí a manipulácií s nimi. V nasledujúcej časti tejto kapitoly budeme v náznakoch pomocou jednoduchých ilustračných príkladov diskutovať len metódy riešenia problémov.

Pre názornosť a jednoduchosť našich úvah, vysvetlíme základné metódy a postupy UI na jednoduchom príklade hry „piškvorky“[1]. Táto hra vyžaduje dvoch hráčov, prvý hráč je označený symbolom X a druhý hráč symbolom O. Snahou každého hráča je umiestniť na štvorcovej 3×3 hracej doske svoje symboly tak, aby tvorili buď riadok, stĺpec, alebo uhlopriečku (pozri obr. 3.1). Hra je zahájená hráčom X, potom nasleduje ťah hráča O. Toto striedanie hráčov sa opakuje tak dlho, až niektorý z hráčov vytvorí stĺpec, riadok či diagonálu z trojice svojich symbolov, alebo je na hracej doske umiestnených deväť symbolov. Hru vyhráva ten hráč, ktorý prvý vytvoril požadovaný reťazec, alebo hra končí remízou, ak hracia doska obsahuje deväť symbolov a žiaden hráč nevytvoril požadovaný reťazec (pozri obr. 3.1). Priebehy hier môžeme reprezentovať pomocou „stromu riešení“, kde je explicitne ukázané, ktoré ťahy boli použité. Koncové pozície stromu sú buď víťazné alebo remízové, maximálne hĺbka stromu riešení je 9 (teda maximálne deviatimi ťahmi sa dostaneme do situácie, kedy niekto vyhral, alebo sa dosiahlo remízy, pozri obr. 3.2)

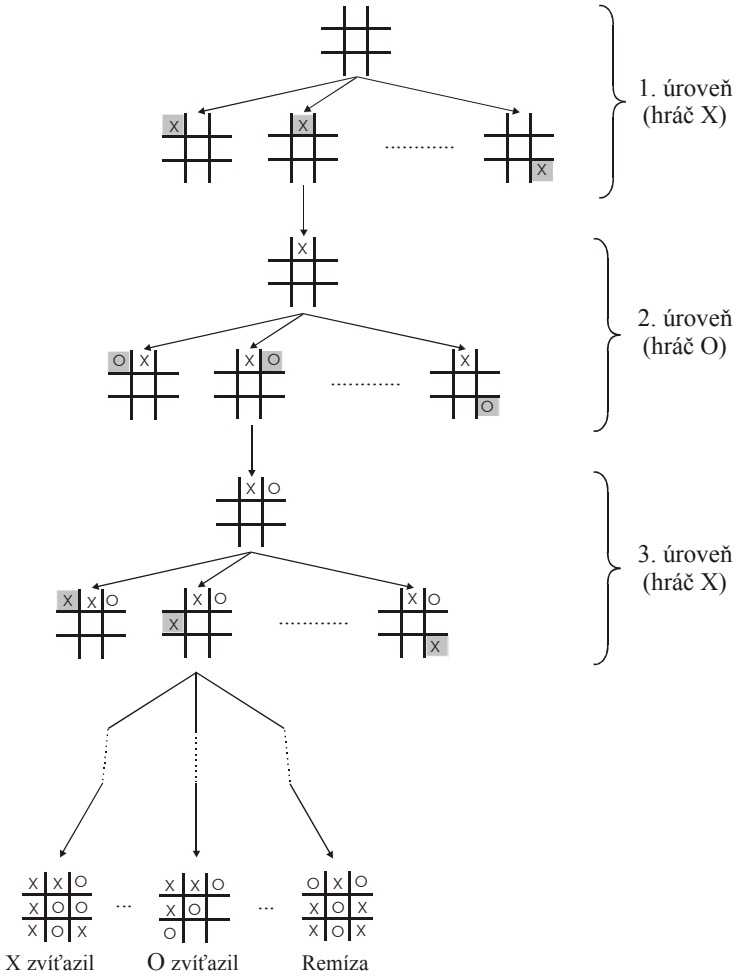


Obrázok 3.1. Znázornenie 3 partií hry piškvorky. Indexy pri jednotlivých symboloch X a/alebo O znamenajú poradie ťahu. V prvých dvoch partiách bolo dosiahnuté víťazstvo, hráč umiestnil svoje symboly do riadku, stĺpca

[1] „Piškvorky“ je český termín pre hru s anglickým názvom „Tic-Tac-Toe“, zatiaľ neexistuje slovenský ekvivalent tohto termínu, preto sme použili český termín.

alebo diagonály, čo je naznačené prerušovanou čiarou. V tretej partii sa žiadnemu hráčovi takto nepodarilo umiestniť svoje symboly, po 9 ťahoch, keď sú obsadené všetky pozície hracej dosky, hra končí remízou.

Hra piškvorky patrí medzi tzv. symetrické hry, z pohľadu druhého hráča je hra identická s hrou prvého hráča (hráči sú rovnocenní, odlišujú sa len v tom, že jeden z nich zahajuje hru). Obaja hráči riešia rovnaký strategický problém, vytvoriť čo najrýchlejšie stĺpec, riadok či diagonálu svojich znakov (maximalizujú svoj zisk) a súčasne zabrániť v tejto akcii súperovi (minimalizujú súperov zisk).



Obrázok 3.2. Znázornenie stromu riešení, ktorého vrchol (koreň) je prázdna hracia doska. Po prvom ťahu, ktorý hral hráč X, je obsadená jedna pozícia (zdôraznená vytieňovaným) symbolom X. V druhej úrovni, hranej hráčom O, je obsadená pozícia (vytieňovaná) symbolom O. Vetva stromu riešení končí vtedy, ak jeden hráč zvíťazil, alebo sú obsadené všetky pozície na hracej doske - hra skončila remízou.

Vo všeobecnosti existujú dva diametrálne odlišné prístupy k riešeniu problémov. *Prvý prístup* je založený na existencii modelu hry, ktorý obsahuje hierarchicky usporiadané pravidlá. Keď tieto pravidlá budeme dodržiavať, mali by sme dospieť ak nie víťaznej pozícii, tak aspoň k remíze. Žiaľ, tento model je zostrojiteľný len pre jednoduché hry, pre zložitejšie hry (napr. šach) už nie sme schopní ho zostrojiť s dostatočnou presnosťou a efektivitou. Obvykle sme v zložitých prípadoch schopní formulovať len rôzne všeobecné pravidlá (heuristiky), ktorých dodržiavanie v priebehu hry by malo viesť k víťazstvu. Tak napr. v šachu také heuristiky obsahujú požiadavky, aby sme sa vyhli strate figúry, budovali pevný stred, obsadzovali volné stĺpce a diagonály, atď. *Druhý prístup* je založený na rozsiahlom prehľadávaní stromu riešení, pomocou ktorého, aspoň teoreticky, sme schopní nájsť optimálny ťah v každej etape hry.

Náznaky tohto prístupu sú znázornené na obr. 3.2, kde sa ľahko môže zistiť, aká sekvencia ťahov vedie k víťazstvu, alebo aspoň k remíze. Žiaľ, aj keď je tento prístup koncepčne veľmi jednoduchý, jeho numerická realizácia v počítači naráža na vážne problémy s enormnou veľkosťou stromu riešení. K numerickému zvládnutiu tohto prístupu musíme zaviesť niektoré podstatné zjednodušenia týkajúce sa hĺbky prehľadávania stromu riešení (tak napr. pri prehľadávaní stromu riešení ideme do maximálnej hĺbky 3 alebo 4). Toto zjednodušenie prehľadávania stromu riešení je obvykle kombinované s rôznymi heuristikami, ktoré ohodnocujú „koncové“ stavy hry. Tak napríklad, štandardný hráč šachu je schopný pre prehľadávaní stromu riešení ísť do hĺbky 3 alebo 4, pričom venuje svoju pozornosť len niektorým vetvám stromu riešení. Z týchto dôvodov, aby sa s určitou pravdepodobnosťou vyvaroval nepríjemných prekvapení, každá koncová pozícia je vyhodnocovaná z pohľadu toho, či napr. získal figúru, alebo nejakú inú strategickú výhodu.

3.1 Model hry piškvorky

Jeden zo základných princípov UI je návrh modelu študovaného problému, ktorý pomocou súboru pravidiel je schopný hrať piškvorky na dobrej úrovni. Uvedieme jednoduchý model, ktorý sa zakladá na tom, že ťah hráča je určený nasledujúcimi šiestimi pravidlami s klesajúcou prioritou:

Pravidlo 1. *Hráč vykoná ťah, ktorý vedie k jeho víťazstvu.*

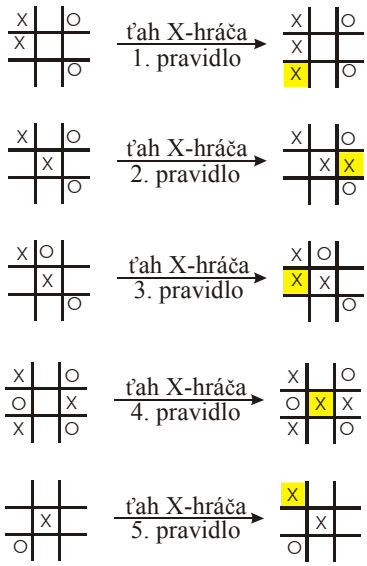
Pravidlo 2. *Hráč vykoná ťah, ktorý zabráni víťazstvu oponenta v nasledujúcom ťahu.*

Pravidlo 3. *Hráč vykoná ťah, ktorý pripraví možnosť dvojnásobného použitia 1. pravidla v nasledujúcom ťahu (tzv. vidlička).*

Pravidlo 4. *Hráč vykoná ťah, ktorým obsadí stredné pole.*

Pravidlo 5. *Hráč vykoná ťah, ktorým obsadí rohové pole.*

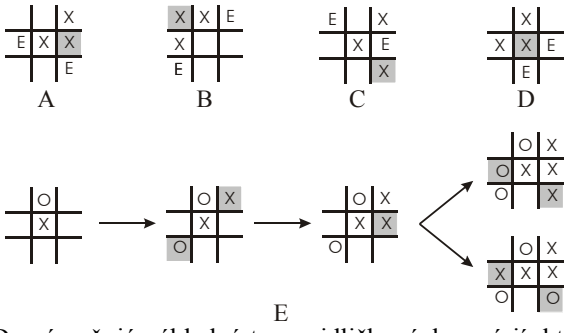
Pravidlo 6. *Hráč vykoná náhodný ťah.*



Obrázok 3.3. Diagramy ilustrujú jednotlivé pravidlá modelu hry. Poznamenajme, že v dôsledku 4. pravidla je prvým ťahom vždy obsadenie prostredného poľa.

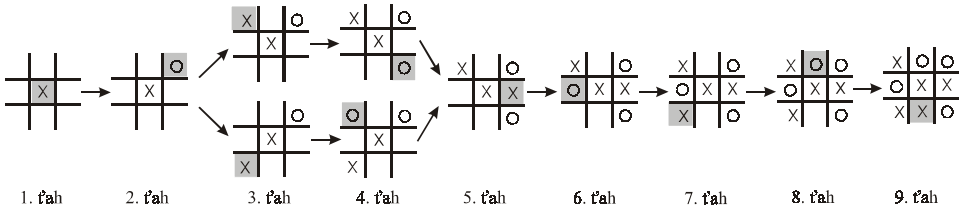
Model nie je plne deterministický, v prípade, že podľa niektorého aplikovaného pravidla existuje niekoľko možností, vyberieme z nich náhodne jednu z nich. Prvé dve

pravidlá odpovedajú jednoduchej rekognoskácii aktuálnej pozície o jeden ťah vopred. Rekognoskácia o dva ťahy vopred je zahrnutá až v treťom pravidle, pripravuje možnosť použitia 1. pravidla. Jednotlivé pravidlá sú ilustrované na obr. 3.3. Tieto pravidlá poskytujú veľmi efektívny model hry piškvorky, ktorý dáva prvému hráčovi pomerne dobrú šancu zvíťaziť, druhý hráč pri presnej hre môže len forsírovať remízu. Na obr. 3.4, diagramy A-D, sú uvedené vybrané východzie pozície, kde je možné vytvoriť vetvenie pozícií, diagram E ilustruje priebeh hry, ktorá už po druhom ťahu druhého hráča je prehraná, prvý hráč môže vynútiť výhru.



Obrázok 3.4. Diagramy A-D znázorňujú základné typy vidličkových pozícií, ktoré sú aplikovateľné použitím pravidla 3. Symboly E znázorňujú príklady dvojice pozícií, ktoré musia byť prázdne, aby sa dala "vidlička" v ďalšom ťahu doplniť na víťaznú trojicu. Diagram E ukazuje pozíciu, ktorá je prehraná pre hráča O už po druhom ťahu. Pravidlá nášho modelu hry používa prvý hráč X.

Hra riadená pre obidvoch hráčov modelom je znázornená na obr. 3.5. Hra je vždy zahájená tým, že prvý hráč umiestni symbol X do stredu (pravidlo 4), v nasledujúcom ťahu druhý hráč umiestni symbol O do rohu (pravidlo 5). V treťom ťahu dochádza k dočasnému štiepeniu pozície, 1. hráč podľa pravidla 4 položí do rohu symbol X. V nasledujúcom 4. ťahu druhý hráč umiestni symbol O do rohu, čím obe pozície splynú do jednej. V piatom ťahu 1. hráč musí zabrániť oponentovi vo výhre (pravidlo 2) a umiestni symbol X medzi dva symboly O, od tohto ťahu je už pozícia remízová. Pomocou tohto jednoduchého príkladu vidíme, že používanie modelu podstatne zjednoduší počítačovú implementáciu hry.



Obrázok 3.5. Znázornenie priebehu hry pomocou modelu, hra skončila remízou.

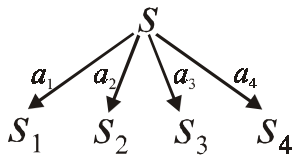
3.2 Prehľadávanie stromu riešení

Pristúpme teraz k formalizácii našich úvah, ktoré boli založené na hre piškvorky. Rozloženie znakov X a O na hracej doske sa nazýva *stav hry*. Ťah hráča, t.j. pridanie znaku X alebo O na hraciu dosky do konkrétnej volnej polohy, sa nazýva *akcia*. Pre daný stav s má hráč k dispozícii množinu prípustných akcií $\mathcal{A}(s)=\{a_1, a_2, \ldots\}$. Hovoríme, že akcia *a transformuje* stav s na nový stav s' , alebo $s' = a(s)$, pozri obr. 3.6. Množina všetkých možných stavov $\mathcal{S}=\{s_1, s_2, \ldots\}$ sa nazýva *stavový priestor* hry. Pomocou akcií sa môžeme pohybovať v tomto stavovom priestore. Nech s_0 je počiatočný stav odpovedajúci prázdnej hracej doske, prvý hráč

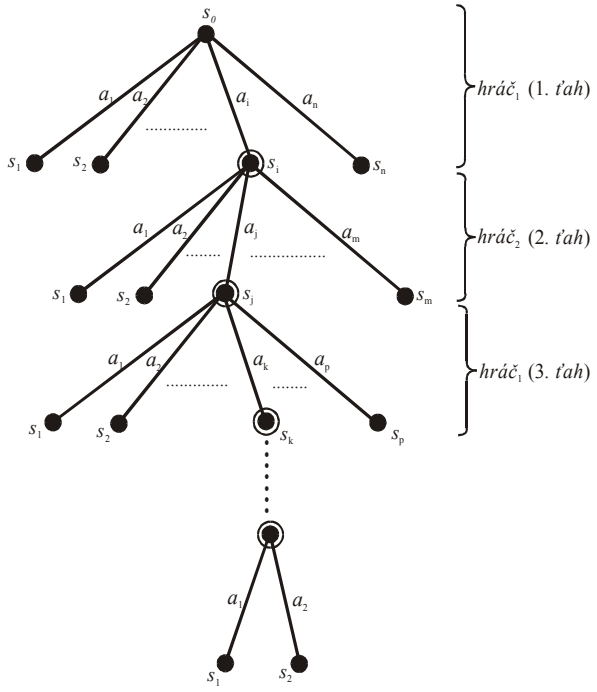
na vybranú pozíciu priloží znak X, dostaneme pozíciu s_1, potom druhý hráč na stav vykoná akciu (priloží na voľnú polohu znak O), takto dostaneme stav s_2, atď. Priebeh hry je popísaný sekvenciou stavov

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 ..... \xrightarrow{a_k} s_k \Rightarrow \left(s_0 s_1 s_2 ... s_k\right)$$

Konštrukcia *stromu riešení* (pozri obr. 3.7) prebieha tak, že pre daný stav (aktuálny vrchol-stav) použijeme všetky možné prípustné transformácie na vytvorenie nových "následníckych" stavov, ktoré sú spojené s pôvodným stavom hranami. Príslušná vetva končí vtedy, keď je pozícia interpretovaná ako víťazstvo/porážka, alebo keď už neexistujú prípustné akcie, ktoré by pretransformovali daný stav na iný (napr. vtedy, keď už všetky pozície sú obsadené).



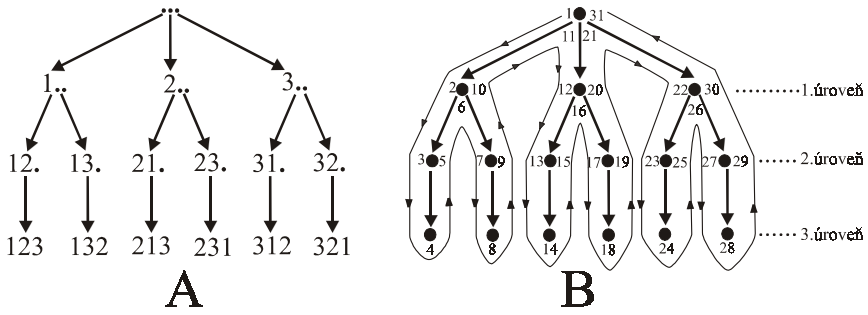
Obrázok 3.6. Schematické znázornenie transformácie stavu s pomocou akcií a_1, a_2, a_3, a_4 na nové stavy s_1, s_2, s_3 resp. s_4 (v tomto prípade indexovanie odpovedá rozdielnym polohám v priestore, nie v čase).



Obrázok 3.7. Zovšeobecnenie obr. 3.2, ktorý znázorňoval hru piškvorky. Prvý hráč vytvorí z počiatočnej pozície (vrchol stromu) všetky možné nasledujúce (1-ťahové) pozície s_1, s_2,…. Na základe určitých (racionálnych) úvah vyberie pozíciu s_i. Druhý hráč z pozície s_i vytvorí nové (2-ťahové) pozície, z nich vyberie pozíciu s_j ako výsledok svojho ťahu – akcie. Oba hráči tento postup opakujú, končí sa vtedy, ak niektorý hráč vyhral, alebo obaja hráči remizovali.

Stojíme pred úlohou, ako formalizovať pohyb po stromu riešení, tento problém je riešený metódou spätného prehľadávania [4,5]. Jej základné princípy sú ilustrované na obr. 3.8, ktorý je pre konkrétnosť urobený tak, že znázorňuje konštrukciu všetkých možných 6 permutácií troch objektov 1, 2 a 3. Metóda je inicializovaná prázdnym riešením (**...**), na prvej úrovni je prázdny symbol '**.**' nahradený postupne všetkými možnými číslicami 1, 2 a 3. V druhej úrovni je druhý prázdny symbol '**.**' nahradený prípustnými číslicami 2, 3 (alebo 1, 3 resp. 1, 2). Tento jednoduchý postup "predlžovania" riešení sa opakuje až do vyčerpania prípustných

symbolov. Potom sa metóda musí vrátiť späť o jednu úroveň vyššie a celý postup sa opakuje. metóda končí vtedy, keď na prvej úrovni máme vyčerpané všetky možnosti použitia číslic 1, 2, a 3. Týmto systematickým spôsobom zostrojíme postupne všetky možné permutácie troch objektov (týchto permutácií je 3!=6). Naznačená metóda spätného prehľadávania je v tejto forme bezprostredne použiteľná aj pre konštrukciu stromu riešení hry piškvorky, týmto systematickým spôsobom môžeme doplniť obr. 3.2, kde je uvedený určitý výsek stromu riešení.



Obrázok 3.8. (A) Strom riešení pre konštrukciu permutácií troch objektov označených 1, 2 a 3. (B) Použitie metódy spätného hľadania ku konštrukcii stromu riešení. Metóda postupne prechádza všetkými vrcholmi – stavmi stromu, jednotlivé etapy „prechádzky“ sú indexované číslicami stojacimi pri vrcholoch. Jednotlivé vetvy stromu sú ukončené vtedy, ak už nemôžeme ďalej pokračovať v predlžovaní riešenia.

Veľkosť stromu riešení hry piškvorky je určená veľkosťou stavového priestoru (t.j. počtom rôznych stavov - pozícií hry), pomocou jednoduchých kombinatorických úvah[2]

$$N = \sum_{p=1}^{5} \binom{9}{p} \left[\binom{9-p}{p-1} + \binom{9-p}{p} \right] - 2 \times 6 \times 13 = 5889$$

kde napr. pre p=1 dostaneme 9×(1+8) pozícií, ktoré obsahujú buď len jeden symbol X, alebo dva symboly X a O. Posledný člen na pravej strane odpovedá skutočnosti, že niektoré členy, obsahujúce jeden stĺpec (riadok) znakov X a jeden stĺpec (riadok) znakov O, nemôžu byť zahrnuté v povolených pozíciách, týchto "nepovolených" pozícií je 156.

Použitím metódy spätného prehľadávania môžeme zostrojiť kompletný strom riešení hry piškvorky. Zistili sme, že má nasledujúci počet koncových pozícií, ktoré sú charakterizované takto

Počet	Typ
131184	víťazstvo hráča X
77904	víťazstvo hráča O
46080	remíza hráčov X a O
255168	celkový počet

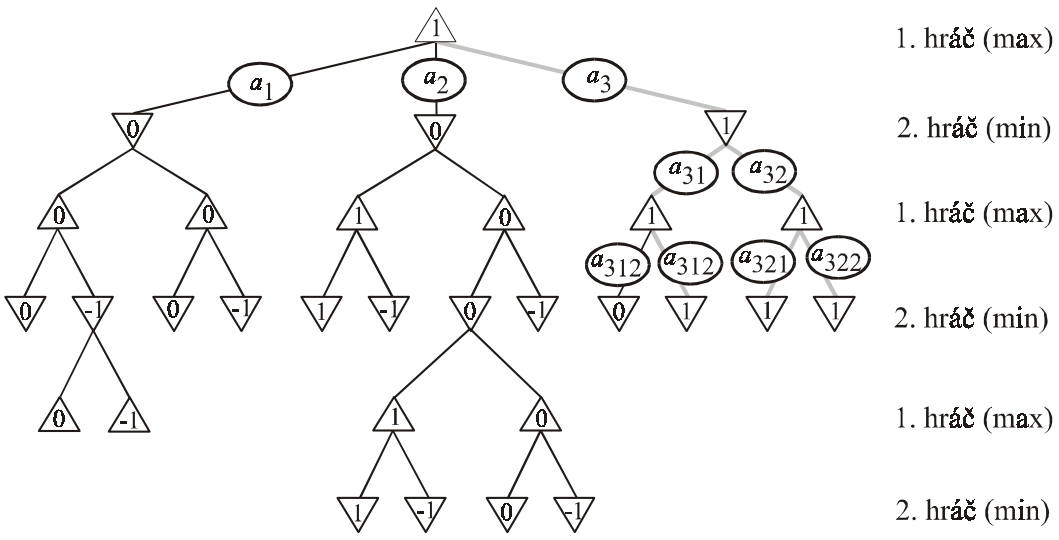
Celkový počet koncových vetví v strome riešení možno jednoducho odhadnúť ako 9!=362880. Niektoré koncové pozície sú totožné, ale cesty, ktorými sa k nim došlo, sa líšia.

Ďalší dôležitý problém pred ktorým teraz stojíme (predpokladáme, že už poznáme úplný strom riešení hry piškvorky) je zistiť víťaznú stratégiu pre prvého hráča. Existujú tieto tri možnosti:

[2] Táto formula bola odvodená J. Pospíchalom.

(1) existuje víťazná stratégia pre prvého hráča, to znamená, že druhý hráč pri použití tejto stratégii musí prehrať,
(2) existuje víťazná stratégia pre druhého hráča, to znamená, že prvý hráč pri použití tejto stratégii musí prehrať,
(3) neexistujú víťazné stratégie, optimálna stratégia poskytuje obom hráčom len remízu.

Jednoduchá modifikácia metódy spätného hľadania pre konštrukciu stromu riešení nám umožňuje zistiť optimálnu stratégiu pre oboch hráčov. Jej základný princíp je veľmi jednoduchý, vychádza zo skutočnosti, že každý hráč volí taký ťah, aby maximalizoval svoj zisk a minimalizoval zisk súpera, preto ho nazývame „*minimax princíp*“. Toto pravidlo je taktiež implicitne obsiahnuté aj vo vyššie uvedenom modelu hry piškvorky (pozri kap. 3.1). Obr. 3.9. znázorňuje strom riešení nejakej hypotetickej hry, všetky možné partie, ktoré sa dajú zohrať, sú znázornené týmto grafom, koncové pozície určujú, ktorý hráč vyhral, alebo či obaja hráči remizovali. Ukážeme metódu, ako sa systematicky pohybovať po stromu riešení, ktorá je založená na tzv. spätnom prehľadávaní. Prvý hráča (označený max) chce vyhrať, to znamená, že bude vyberať také ťahy, aby maximalizoval svoj zisk a minimalizoval zisk súpera (druhého hráča, označeného min). Samozrejmé, je prirodzené očakávať, že aj druhý hráč sa bude správať podobne, ako prvý hráč. Ak koncová pozícia 1. hráča je víťazná (prehraná) tak ju ohodnotíme 1 (-1), v prípade, že koncová pozícia je remíza, potom je ohodnotená 0. Ako už bolo povedané, stratégia 1. hráča je maximalizovať svoj zisk, pričom stratégia 2. hráča je minimalizovať zisk protihráča – 1. hráča (pozri obr. 3.9). Pod optimálnou stratégiou budeme rozumieť takú postupnosť ťahov oboch hráčov, ktorá maximalizuje zisk oboch hráčov. Optimálna stratégia je vyhrávajúca pre jedného hráča, ak postupnosti ťahov vytvorené na základe tejto stratégie vedú k jeho výhre (bez ohľadu na to, aké ťahy použil protivník). Optimálna stratégia je remízová, ak vedie k remíze, ani jeden hráč neprehral.



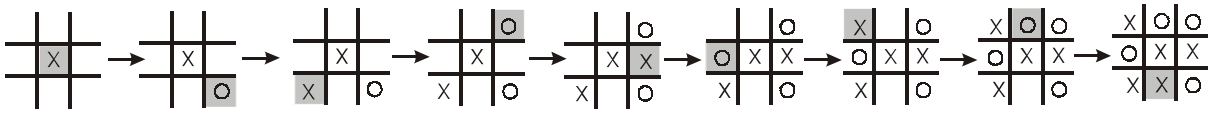
Obrázok 3.9. Strom riešení hypotetickej hry, kde 1. hráč (2. hráč) je reprezentovaný trojuholníkom $\triangle$ (opačným trojuholníkom ∇) . Koncové pozície sú ohodnotené podľa toho, či prvý hráč vyhral (1), prehral (0), alebo remizoval (0). Idúc zdola nahor ohodnocujeme vrcholy stromu riešení tak, že pre 1. hráča (2. hráča) berieme maximálnu (minimálnu) hodnotu susedných dolných vrcholov umiestnených nižšie. V prípade, že horný vrchol stromu (1. hráč) je ohodnotený 1 (0) , potom (neexistujú) existujú také jeho ťahy, že vyhrá bez ohľadu na ťahy druhého hráča (označené na obrázku hrubou čiarou).

Hrany (ťahy) stromu riešení na obr. 3.9 sú označené ťahmi, ktorými boli vytvorené. Pomocou tohto označenia môžeme zrekonštruovať optimálny priebeh hry. Tak napr. z obr. 3.9 vyplýva, že optimálna vyhrávajúca stratégie pre prvého hráča je použitie ako prvého ťahu

a_3, nezávisle od toho, ako bude hrať súper „min“. V druhom ťahu hráč „min“ má výber medzi dvoma ťahmi a_{31} a a_{32}, avšak ani jeden nevedie k jeho víťazstvu, alebo aspoň k remíze. To znamená, že môžeme zostrojiť tri postupnosti ťahov

$a_3 - a_{31} - a_{312}$, $a_3 - a_{32} - a_{321}$ a $a_3 - a_{32} - a_{322}$, ktoré vedú k víťazstvu hráča „max“, proti ktorým, nemá hráč „min“ protihru – obranu.

Pomocou tohto jednoduchého postupu založeného na minimaxovom princípe (pozri obr. 3.9) môžeme riešiť veľkú triedu hier s dvoma hráčmi, ktorí striedavo vykonávajú ťahy, pričom hlavným strategickým zámerom každého hráča je maximalizovať svoj zisk (t.j. minimalizovať zisk súpera). Hlavný problém s použitím tohto algoritmu spočíva v tom, že strom riešení pre zložitejšie hry (napr. šach) má enormnú veľkosť, takže systematické prechádzanie po všetkých jeho vrcholoch- stavoch je nerealizovateľné.



Obrázok 3.10. Znázornenie postupnosti (597364128) pomocou jednotlivých ťahov hry piškvorky.

Ak aplikujeme metódu spätného hľadania kombinovanú s minimax princípom, potom zistíme, že pri obojstranne korektnej hre hráči môžu v najlepšom prípade dosiahnuť len remízu. To znamená, že vrchol stromu riešení je ohodnotený na záver 0, jedna z týchto optimálnych partií je postupnosť ťahov reprezentovaná „permutáciou“ (597364128), pozri obr. 3.10. Jednoducho sa môže skontrolovať, že táto postupnosť ťahov vyhovuje modelu hry piškvorky, ktorý bol diskutovaný v prvej časti tejto kapitoly. Tieto závery môžeme zhrnúť do nasledujúcich viet:

Veta 3.1. Optimálna stratégia hry piškvorky vedie k remíze, neexistuje taká stratégia, pomocou ktorej by jeden hráč vyhral a druhý prehral.

Tento dôležitý záver vyplýva z výsledkov získaných metódou spätného hľadania spolu s „minimax“ princípom. Tým, že metóda prekontrolovala celý strom riešení, môžeme tento výsledok pokladať za konečný a nemenný.

Veta 3.2. Optimálna stratégia hry piškvorky sa pre prvého hráča riadi pomocou modelu hry uvedeného v kapitole 3.1.

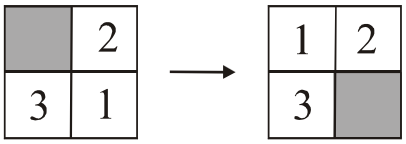
Podobne, ako v predchádzajúcom prípade, aj táto veta je dôsledkom počítačových simulačných výpočtov. Skontrolovali sme optimálne postupnosti ťahov dĺžky 9, zistili sme, že vo všetkých prípadoch model hry je splnený. Pre úplnosť musíme však poznamenať, že sme takto neskontrolovali všetky optimálne sekvencie (ktorých je niekoľko sto). Na vyvrátenie platnosti obdobnej vety pre druhého hráča ale stačí jedna postupnosť ťahov (1583749), kedy prvý hráč vyhráva proti stratégii opísanej v kapitole 3.1 hrajúcej ako druhej. Na skutočne optimálnu stratégiu by sme museli do stratégie z kapitoly 3.1 pridať ďalšie pravidlo, ktoré zabraňuje súperovi pripraviť "vidličku".

Zhrnutie

Umelá inteligencia je časť informatiky, ktorá sa zaoberá algoritmizáciou problémov obtiažne formulovateľných a riešiteľných, ktoré, ak sú riešené ľuďmi, vyžadujú určitú inteligenciu. V súčasnosti sa pre tieto prístupy presadzuje v literatúre neutrálny názov „počítačová inteligencia“. Jedným z hlavných cieľov umelej inteligencie je implementácia „superprogramu“, ktorý bude simulovať ľudskú myseľ. Ako ilustráciu širokého spektra metód umelej inteligencie sme diskutovali metódy riešenia problémov, ktoré sú založené na prehľadávaní stromu riešení.

Úlohy

Úloha 3.1. 4-hlavolam spočíva v tom, že náhodne usporiadané číslice 1-3 musíme pomocou premiestňovania jednotlivých políčok dostať do „konečnej polohy“, pozri obr. 3.11. Zostrojte strom riešení, kde vrchol stromu bude ľavá východzia pozícia hlavolamu a jednotlivé vetvy budú ukončené pravou konečnou pozíciou.

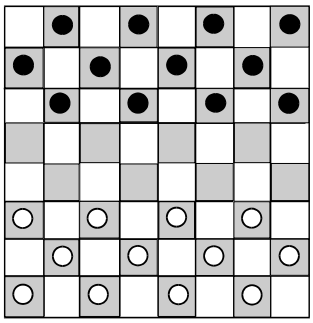


Obrázok 3.11. Znázornenie hry 4-hlavolam. Pomocou premiestňovania jednotlivých políčok musíme dosiahnuť konečný stav znázornený v pravej časti obrázku.

Úloha 3.2. Na obr. 3.12 je znázornená úloha „hanojské veže“, zostrojte pre túto úlohu strom riešení.

Obrázok 3.12. Zjednodušená verzia úlohy „hanojské veže“, kde dva disky sú „navlieknuté“ na vertikálnej tyči. Riešenie úlohy spočíva v tom, aby sme vhodným postupným premiestňovaním jednotlivých diskov z jednej tyče na druhú, použitím aj strednej tyče, dostali koncovú pozíciu.

Úloha 3.3. Zjednodušená hra „dáma“ používa 8×8 šachovnicu, obr. 3.13. Podobne, ako bola navrhnutá stratégia hry „piškvorky“ pomocou systému pravidiel, navrhnite aj pre hru „dáma“ podobný systém pravidiel.

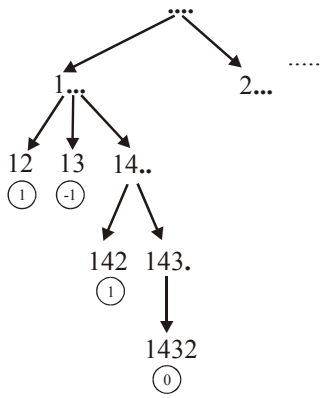


Obrázok 3.13. Zjednodušená hra „dáma“ používa 8×8 šachovnicu, pričom každý hráč ma 12 kameňov. Hra je zahájená hráčom, ktorý má biele kamene. Sú povolené dva druhy ťahov: (1) ťah dopredu kameňom na neobsadené pole, a (2) branie kameňa protihráča umiestneného šikmo, pričom políčko za ním musí byť volné, takto je možné brať viac kameňov naraz, teda jednoduché, dvojité a trojité branie (branie kameňa je povinné). Hru vyhráva ten hráč, ktorý povolenými ťahmi dostal svoj kameň na posledný riadok (t.j. na prvý riadok súpera), alebo ak druhý hráč nemá už žiadne kamene. Hra končí remízou, keď hráč nemôže vykonať ťah (jeho kamene sú zablokované súperovými kameňmi).

Úloha 3.4. Na obr. 3.8 je znázornená metóda spätného hľadania, aplikovaná pre konštrukciu permutácií 3 objektov. Modifikujte tento diagram pre konštrukciu permutácií 4 objektov.

Úloha 3.5. Zostrojte podľa obr. 3.8 strom riešení pre 18 objektov (permutácií alebo začiatkov permutácií), ktorých ohodnotenie je definované pomocou nasledujúcej tabuľky, vrcholy z jednotlivých vrstiev stromu ohodnoťte pomocou minimax princípu.

#	objekt	ohodnotenie	#	objekt	ohodnotenie
1	(1,2)	1	10	(3,1)	1
2	(1,3)	-1	11	(3,2)	-1
3	(1,4,2)	1	12	(3,4,1)	-1
4	(1,4,3,2)	0	13	(3,4,2,1)	-1
5	(2,1,3,4)	0	14	(4,1,2,3)	0
6	(2,1,4,3)	-1	15	(4,1,3)	1
7	(2,3,1,4)	0	16	(4,2,1,3)	0
8	(2,3,4)	1	17	(4,2,3,1)	-1
9	(2,4)	1	18	(4,3)	1



Obrázok 3.14. Výrez zo stromu riešení 18 objektov z tabuľky úlohy 3.5.

Otázky na opakovanie

Otázka 3.1. Čo je umelá inteligencia?

Otázka 3.2. Aké sú dva prístupy k umelej inteligencii?

Otázka 3.3. Ako je charakterizovaná hra piškvorky?

Otázka 3.4. Čo je strom riešení hry piškvorky?

Otázka 3.5. Naformulujte model hry piškvorky pomocou hierarchických pravidiel.

Otázka 3.6. Realizujte jeden priebeh hry piškvorky pomocou jej modelu, v každom ťahu uveďte, ktoré pravidlo bolo použité.

Otázka 3.7. Čo je stav, akcia, stavový priestor, ako sa pomocou týchto pojmov zostrojí strom riešení?

Otázka 3.8. Čo je metóda spätného prehľadávania, ilustrujte ju na príklade konštrukcie permutácií 3 objektov.

Otázka 3.9. Čo je minimax princíp, ilustrujte ho na jednoduchom strome riešení, ktorý má 4 vrstvy.

Otázka 3.10. Čo je optimálna stratégia? Ako ju môžeme nájsť pomocou stromu riešení a minimaxového princípu?

4 Logika

Logika je obvykle charakterizovaná ako analýza metód používaných v ľudskom myslení alebo uvažovaní. Slovo „logika“ sa často vyskytuje v bežnej hovorovej reči v takých spojeniach ako napr. „*to nemá žiadnu logiku*“, „*neúprosná logika vývoja*“, „*ženská logika*“, „*logika veci vyžaduje, aby...*“, a mnoho podobných príkladov. Z týchto jednoduchých príkladov vyplýva, že logika nebude dôležitá len v matematike a informatike, ale aj v našom každodennom bežnom živote. Pýšime sa tým, že ľudské bytosti sú jediný biologický druh na Zemi, ktorý sa správa uvedomele racionálne, dokáže vyvodzovať nové poznatky zo známych poznatkov, dokáže predvídať a plánovať svoje konanie. Racionalita je schopnosť konať a myslieť cestami, ktoré sú vedené rozumom. Na tomto mieste môžeme si položiť otázku, prečo naše konanie a myslenie je racionálne, v čom spočívajú korene našej racionality, je to súčasťou našej genetickej výbavy, alebo sa racionalite musíme učiť? Na túto otázku ponúkame jednoduchú odpoveď, s ktorou nie každý bude súhlasiť. Podstatným zdrojom našej racionality je, že používame logiku pri našom myslení a konaní. Logika je hlavnou hybnou silou našej racionality; ak racionalitu budeme hľadať v iných alternatívnych mimologických zdrojoch, rýchlo dospejeme k „iracionálnym“ riešeniam, ktoré s našou proklamovanou racionálnosťou majú len veľmi málo spoločného.

4.1 Výroková logika

Výroková logika študuje také formy usudzovania, pre ktoré platnosť záverov nezávisí od obsahu a ani od vnútornej štruktúry výrokov, ale výlučne len na pravdivosti či nepravdivosti týchto výrokov. Analyzujme tieto jednoduché vety:

(1) Atóm je fyzikálna štruktúra.
(2) Atóm je sociálna štruktúra.
(3) Vo vesmíre existuje život aj mimo Zeme.
(4) Láska je rádioaktívna.
(5) Rast nášho hospodárstva má neustálu tendenciu.[1]

Medzi uvedenými piatimi vetami sú veľké rozdiely. Možno konštatovať, že veta (1) je pravdivá, zatiaľ čo veta (2) je nepravdivá. O vete (3) zatiaľ nemôžeme rozhodnúť jej pravdivosť alebo nepravdivosť. Veta (4) je síce gramaticky správna, ale je to zrejmý nezmysel vzhľadom k predikátu „rádioaktívny“, čiže nemá zmysel uvažovať o jej pravdivosti alebo nepravdivosti. Konečne, skladba vety (5) je chybná, takže nemá vôbec žiadny zmysel sa pýtať na jej pravdivosť alebo nepravdivosť. Po týchto jednoduchých ilustračných príkladoch môžeme pristúpiť k *definovaniu výroku ako jednoduchej oznamovacej vety, pre ktorú má zmysel pýtať sa či je alebo nie je pravdivá*. Pravdivosť alebo nepravdivosť nejakého výroku budeme označovať za *pravdivostnú hodnotu* daného výroku. Prirodzený jazyk obsahuje spojky (napr. *a*, *alebo*, *ak..., potom...*, *je ekvivalentné*, *nie je pravda, že...*) pomocou ktorých

[1] Veta vyslovená jedným významným federálnym politikom na zjazde jeho rodnej strany.

z jednoduchých výrokov vytvárame zložitejšie výroky, pričom ich pravdivosť alebo nepravdivosť je určená len pravdivostnými hodnotami ich zložiek.

Pristúpme teraz k formalizácii našich úvah o výrokoch, ich pravdivostnej hodnote a spojkách. Ako už bolo povedaná, výroková logika sa zaoberá takými formami usudzovania, ktoré závisia len na pravdivosti alebo nepravdivosti použitých výrokov, a nie na tom, či majú alebo nemajú zmysel. Výroky majú pravdivostnú hodnotu 0 (nepravdivý) alebo 1 (pravdivý). Jednotlivé elementárne výroky (nazývané výrokové premenné, ktoré budeme označovať p, q, r, ...) spájame logickými spojkami do zložitejších výrokov, ktoré môžu byť ďalej spájané spojkami do ďalších výrokov, atď. Nasledujúca tabuľka uvádza základné spojky a ich pravdivostné hodnoty vyskytujúce sa vo výrokovej logike

p	q	$\overline{p}$	$\overline{q}$	$p \wedge q$ (konjunkcia)	$p \vee q$ (disjunkcia)	$p \Rightarrow q$ (implikácia)	$p \Leftrightarrow q$ (ekvivalencia)
0	0	1	1	0	0	1	1
0	1	1	0	0	1	1	0
1	0	0	1	0	1	0	0
1	1	0	0	1	1	1	1

V tejto tabuľke symboly p a q reprezentujú výroky, ich negácie sú označené $\overline{p}$ a $\overline{q}$, binárne symboly 0 a 1 reprezentujú „pravdu“ resp. „nepravdu“.

Formule výrokovej logiky sú definované takto:

(1) Každá výroková premenná je formula,

(2) Ak p a q sú výrokové formule, potom aj $\overline{p},\overline{q},(p \wedge q),(p \vee q),(p \Rightarrow q),(p \Leftrightarrow q)$ sú formule[2].

Obvykle sa ešte zdôrazňuje, že žiadne iné výrazy, ako tie, ktoré môže vzniknúť opakovaným použitím (1) a (2), nie sú formulami výrokovej logiky. Zátvorky sa používajú ako pomocné symboly, pomocou ktorých môžeme odstrániť prípadnú nejednoznačnosť výrokových formúl. Uvažujme formulu $p \wedge q \vee r$, môžeme ju interpretovať dvoma rôznymi spôsobmi $(p \wedge q) \vee r$ alebo $p \wedge (q \vee r)$.

Vyššie špecifikovaný spôsob konštrukcie výrokových formúl nazývame *syntaxom výrokovej logiky*. Podobne ako v prirodzenom jazyku, kde syntax špecifikuje tvar vety, nie všetky vety, ktoré môžeme zostrojiť jednoduchým zreťazením slov, sú syntakticky korektné. Podobne aj vo výrokovej logike, nie každé zreťazenie prípustných symbolov nám definuje formulu, existujú formule, ktoré nie sú syntakticky správne (napr. formula $(p \Rightarrow\Rightarrow q) \vee \wedge$).

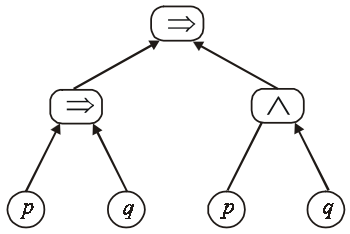
Pre každú formulu výrokovej logiky môžeme zostrojiť tzv. pravdivostnú tabuľku, v ktorej sú postupne počítané hodnoty jej „podformúl“. Pre ilustráciu uvedieme pravdivostnú tabuľku formule $(p \Rightarrow q) \Rightarrow (p \wedge q)$

[2] Znak $\wedge$ označuje logické 'a', $\vee$ logické 'alebo', $p \Rightarrow q$ znamená, že z p vyplýva q (implikácia), $p \Leftrightarrow q$ označuje logickú ekvivalenciu.

1	2	3	4	5
p	q	$1 \Rightarrow 2$	$1 \wedge 2$	$3 \Rightarrow 4$
0	0	1	0	0
0	1	1	0	0
1	0	0	0	1
1	1	1	1	1

Z tejto tabuľky vyplýva, že existujú také pravdivostné hodnoty premenných p a q (p=0, q=0 a p=0, q=1), pre ktoré je pravdivostná hodnota danej výrokovej formule nepravdivá (0). Vo výrokovej logike majú mimoriadne postavenie také formule, ktorých pravdivostná tabuľka je pravdivá pre všetky možné kombinácie pravdivostných hodnôt premenných vo všetkých riadkoch. Takéto formule nazývame *tautológie* a majú postavenie „zákonov“ výrokovej logiky. Ich používanie pri odvodzovaní nových formúl zabezpečuje, že tieto sú taktiež tautológie. Opakom tautológie je *kontradikcia*, t.j. formula, ktorá je pre všetky možné pravdivostné hodnoty jej výrokov nepravdivá. Jednoduchým príkladom kontradikcie je $p \wedge \overline{p}$. Formula, ktorá nie je ani tautológiou a ani kontradikciou, sa nazýva *splniteľná formula*. Splniteľná formula je pravdivá pre niektoré pravdivostné hodnoty jej výrokov, ale je aj nepravdivá pre iné pravdivostné hodnoty jej výrokov.

Každá výroková formula je reprezentovaná pomocou grafického útvaru nazývaného *syntaktický strom,* pozri obr. 4.1.



Obrázok 4.1. Syntaktický strom formule $(p \Rightarrow q) \Rightarrow (p \wedge q)$. Koncové vrcholy stromu reprezentujú výrokové premenné p a q, vrcholy z nasledujúcich vrstiev sú priradené spojkám implikácie a konjunkcie. Vyhodnocovanie tohto stromu prebieha postupne z dolu na hor.

Ako už bolo povedané v predchádzajúcej časti tejto kapitoly, syntax formúl výrokovej logiky je jednoznačne určený spôsobom ich konštrukcie, pomerne ľahko vieme rozhodnúť, či daná formula ma korektný syntax, alebo nemá. Ďalší pojem, dôležitý pre výrokovú logiku je *sémantika*. Pojem pochádza z teórie prirodzených jazykov, kde sémantika špecifikuje význam danej vety (ktorá ma tiež aj svoju syntax). Vo výrokovej logike, ktorá sa zaoberá len pravdivostnými hodnotami premenných a ich formúl, *sémantika* nie je veľmi bohatá. Sémantika výrokovej formule je vlastne tabuľka pravdivostných hodnôt formule pre rôzne hodnoty jej výrokov. Tak napríklad, pre formulu $(p \Rightarrow q) \Rightarrow (p \wedge q)$, ktorá má korektný syntax (napr. reprezentovaný syntaktickým stromom), je jej sémantika plne určená vyššie uvedenou tabuľkou jej pravdivostných hodnôt pre všetky štyri kombinácie výrokov p a q.

Niektoré tautológie sa často používajú nielen v samotnej výrokovej logike, ale aj v bežnom usudzovaní, a sú obvykle označované aj vlastným menom. Väčšinou sa jedná o tautológie tvaru ekvivalencie, ktoré umožňujú nahradzovať jedny formule inými bez straty vlastnosti ich tautologičnosti. Medzi najznámejšie zákony výrokovej logiky patria tieto tautológie:

(1) Zákon totožnosti $p \Rightarrow p$.

(2) Zákon dvojitej negácie $\overline{\overline{p}} \Leftrightarrow p$.

(3) Zákon vylúčenia tretieho $p \vee \overline{p}$.

(4) De Morganov zákon pre konjunkciu $\overline{(p \wedge q)} \Leftrightarrow \overline{p} \vee \overline{q}$.

(5) De Morganov zákon pre disjunkciu $\overline{(p \vee q)} \Leftrightarrow \overline{p} \wedge \overline{q}$.

(6) Zákon ekvivalencie $(p \Leftrightarrow q) \Leftrightarrow ((p \Rightarrow q) \wedge (q \Rightarrow p))$

(7) Zákon hypotetického sylogizmu $(p \Rightarrow q) \Rightarrow ((q \Rightarrow r) \Rightarrow (p \Rightarrow r))$

(8) Distribúcia konjunkcie $(p \vee (q \wedge r)) \Leftrightarrow ((p \vee q) \wedge (p \vee r))$.

(9) Distribúcia disjunkcie $(p \wedge (q \vee r)) \Leftrightarrow ((p \wedge q) \vee (p \wedge r))$.

(10) Zákon kontrapozície $(p \Rightarrow q) \Leftrightarrow (\overline{q} \Rightarrow \overline{p})$

(11) Zákon „reductio ad absurdum“ $(p \Rightarrow q) \Rightarrow ((p \Rightarrow \overline{q}) \Rightarrow \overline{p})$.

(12) Zákon nahradenia implikácie $(p \Rightarrow q) \Leftrightarrow (\overline{p} \vee q)$.

Platnosť všetkých týchto zákonov môžeme prekontrolovať pre všetky pravdivostné hodnoty premenných pomocou tabuľkovej metódy (tj., sú to tautológie).

Výroková formula má *disjunktívny tvar*, keď má formu disjunkcie konečného počtu konjunkcií, z ktorých každá obsahuje len výrokovú premennú alebo jej negáciu, napr. $(p \wedge q \wedge r) \vee (\overline{q} \wedge r) \vee (r \wedge \overline{t})$ alebo $(\overline{p} \wedge q) \vee (p \wedge \overline{q})$. Pomocou zákona nahradenia implikácie a de Morganových zákonov môžeme dokázať dôležitú vetu výrokovej logiky, podľa ktorej platí, že ľubovolnú výrokovú formulu môžeme pretransformovať na disjunktívny tvar.

Veta 4.1. Každá formula môže byť pretransformovaná do ekvivalentného disjunktívneho tvaru.

Na základe tejto vety môžu byť implikácia a ekvivalencia úplne vyeliminované z výrokovej logiky. Ako ilustračný príklad tejto vety uvažujme formulu $(p \Rightarrow q) \Rightarrow (p \wedge q)$, pomocou zákona (12) o nahradení implikácie dostaneme

$$(p \Rightarrow q) \Rightarrow (p \wedge q) \Leftrightarrow \overline{(p \Rightarrow q)} \vee (p \wedge q) \Leftrightarrow \overline{(\overline{p} \vee q)} \vee (p \wedge q) \Leftrightarrow (\overline{\overline{p}} \wedge \overline{q}) \vee (p \wedge q) \Leftrightarrow$$

$$\boxed{(p \wedge \overline{q}) \vee (p \wedge q)}$$

Veta môže byť zosilnená ešte do tvaru, že každá výroková formula môže byť pretransformovaná do ekvivalentného tvaru v ktorom obsahuje buď len konjunkcie a negácie, alebo len disjunkcie a negácie. Táto možnosť plynie priamo z de Morganových zákonov, pomocou ktorých je možno zameniť konjunkciu (disjunkciu) za disjunkciu (konjunkciu). Tak napríklad, vo vyššie uvedenej transformácii môžeme pokračovať tak, že výsledná formula bude obsahovať len disjunkcie a negácie

$$(p \Rightarrow q) \Rightarrow (p \wedge q) \Leftrightarrow (p \wedge \overline{q}) \vee (p \wedge q) \Leftrightarrow \boxed{(\overline{\overline{p} \vee q}) \vee (\overline{\overline{p} \vee \overline{q}})}.$$

4.2 Usudzovanie vo výrokovej logike

Usudzovanie je vyvodzovanie nových poznatkov pomocou zákonov výrokovej logiky z poznatkov v ktorých táto nová informácie nie je explicitne obsiahnutá. Všetky tautológie - zákony výrokovej logike nám môžu slúžiť ako určité „vzory“ pre usudzovanie. V bežnom

živote sa však používa len niekoľko zákonov výrokovej logiky, ktoré boli objavené už gréckymi filozofmi pred viac ako dva tisíc rokmi.

Jeden z fundamentálnych základov usudzovania vo výrokovej logike je zákon „*modus ponens*“, ktorý je založený na tautológii - zákone $\big(p \wedge \big(p \Rightarrow q\big)\big) \Rightarrow q$ (pomocou tabuľkovej metódy sa ľahko presvedčíme, že je to tautológia, pre všetky 4 kombinácie pravdivostných hodnôt premenných p a q dostaneme, že táto formula je pravdivá). Predpokladajme, že p a $\big(p \Rightarrow q\big)$ sú pravdivé, potom aj ich konjunkcia $p \wedge \big(p \Rightarrow q\big)$ je pravdivá. Konečne, ak formula $\big(p \wedge \big(p \Rightarrow q\big)\big) \Rightarrow q$ je pravdivá (je to tautológia) a pravdivý je aj predpoklad (ľavá strana) jej implikácie, potom musí byť pravdivý aj dôsledok q. Tento záver formálne vyjadríme pomocou tejto schémy, ktorá sa z historických dôvodov nazýva *modus ponens*

$$\begin{array}{l} p \Rightarrow q \\ \underline{p} \\ q \end{array}$$

Ako ilustračný príklad modus ponens študujme tieto dva výroky (hlavný - prvý výrok je zložený spojkou implikácie, vedľajší - druhý výrok je elementárny)

Ak mám peniaze, tak si kúpim auto
Mám peniaze

Ak zavedieme výrokové premenné p = *mám peniaze* a q = *kúpim auto,* potom hlavný a vedľajší výrok môžem písať vo forme modus ponens

ak mám peniaze, tak si kúpim auto
mám peniaze
kúpim si auto

Z tohto jednoduchého ilustračného príkladu vyplýva, že modus ponens tvorí vo výrokovej logike „efektívny nástroj“ k vyvodzovaniu nových poznatkov z už známych poznatkov (predpokladov). Modus ponens tvorí základ nášho deduktívneho usudzovania, t.j. vytvárania nových poznatkov z danej „databázy“ poznatkov. Ináč povedané, pri deduktívnom usudzovaní nevzniká nová informácia, len „vyvodíme“ tú, čo je obsiahnutá v našej databáze vedomostí.

Ďalší dôležitý logický nástroj usudzovania je *modus tollens*, ktorý je úzku spriahnutý s modus ponens. Tautológiu $\big(p \wedge \big(p \Rightarrow q\big)\big) \Rightarrow q$ vhodnou substitúciou premenných ($p \to \overline{q}$ a $q \to \overline{p}$) prepíšeme do tvaru $\big(\overline{q} \wedge \big(\overline{q} \Rightarrow \overline{p}\big)\big) \Rightarrow \overline{p}$, implikáciu na ľavej strane prepíšeme pomocou zákona kontrapozície, $\big(p \Rightarrow q\big) \Leftrightarrow \big(\overline{q} \Rightarrow \overline{p}\big)$, dostaneme $\big(\overline{q} \wedge \big(p \Rightarrow q\big)\big) \Rightarrow \overline{p}$. Pomocou tabuľkovej metódy sa ľahko presvedčíme, že táto formula je tautológia. Túto výrokovú formulu vyjadríme schémou usudzovania nazývanou *modus tollens*

$$\begin{array}{l} p \Rightarrow q \\ \underline{\overline{q}} \\ \overline{p} \end{array}$$

Pre ilustráciu tejto schémy, použijeme podobný príklad ako pre modus ponens

ak mám peniaze, tak si kúpim auto
nekúpim si auto
nemám peniaze

Pri tejto príležitosti je potrebné zdôrazniť, že modus tollens vyplýva z modus ponens a zákona kontrapozície, podľa ktorého, *ak chceme implikáciu „obrátiť“, tak potom musíme znegovať aj jednotlivé výroky implikácie*. Je veľkou chybou nášho usudzovania použitie formule $\big(p \Rightarrow q\big) \Rightarrow \big(q \Rightarrow p\big)$, podľa ktorej sa obrátenie implikácie vykonáva jednoducho bez negácií

jednotlivých výrokov. Priamy dôsledkom tejto formule sú nasledujúce dve chybné schémy usudzovania. Prvá sa nazýva *popretie predpokladu*

$$\frac{p \Rightarrow q \quad \overline{p}}{\overline{q}}$$

Druhá sa nazýva *potvrdenie dôsledku*

$$\frac{p \Rightarrow q \quad q}{p}$$

Prvá schéma „popretie predpokladu“ je ilustrovaná príkladom

ak som pekná, tak sa *vydám*

nie som pekná

---

nevydám sa

Záver nie je korektný, môže sa vydať aj vtedy, keď nie je pekná. Druhá schéma „potvrdenie dôsledku môže byť ilustrovaná podobným príkladom

ak som pekná, tak sa *vydám*

vydala som sa

---

som pekná

Chyba v usadzovaní je podobná ako v predchádzajúcom príklade. Ako svedčia mnohé kognitívno-psychologické výskumy [xx], obe tieto schémy, aj keď sú chybné, sa často využívajú v bežnom usudzovaní, možno ich teda pokladať za „klasické chyby“ nášho každodenného uvažovania.

Niekoľko poznámok o používaní uvedených dvoch schém modus ponens a modus tollens v matematike pri dôkazoch. Existujú tri veľké triedy spôsobu dokazovania v matematike:

(1) *priamy dôkaz*, uskutočňovaný pomocou modus ponens,

(2) *nepriamy dôkaz*, uskutočňovaný pomocou modus tollens, a

(3) *dôkaz sporom*, uskutočňovaní pomocou zákona „reductio ad absurdum“.

Metóda „priameho dôkazu“, založená na použití modus ponens, patrí medzi základné techniky dôkazu používané v matematike, preto ju tu nebudeme ilustrovať. Učebnice matematiky sú plné „priamych dôkazov“ rozličných viet.

Nepriamy dôkaz budeme ilustrovať nasledujúcim jednoduchým príkladom. Chceme dokázať výrok (triviálnu vlastnosť prirodzených čísel): Ak *p*=(*súčin dvoch celých čísel* α *a* β *je párny*), potom *q*=(α *alebo* β *sú párne čísla*). Predpokladajme negáciu výroku[3] *q*, $\overline{q} = (\alpha \ a \ \beta$ *sú nepárne čísla*$)$, potom súčin dvoch nepárnych čísel je taktiež nepárne číslo, tým sme dokázali negáciu $\overline{p}$ pôvodného predpokladu našej implikácie. Týmto nepriamym spôsobom sme dokázali výrok $p \Rightarrow q$, tak, že sme dokázali jeho „inverziu“ $\overline{q} \rightarrow \overline{p}$.

Dôkaz sporom je založený na použití zákona „reductio ad absurdum“ $(p \Rightarrow q) \Rightarrow ((p \Rightarrow \overline{q}) \Rightarrow \overline{p})$. Ak z nejakého predpokladu *p* súčasne vyplýva výrok *q* a aj jeho negácia, potom musí platiť negácia *p*. Postupným použitím modus ponens na tento zákon dostaneme zákon „reductio ad absurdum“ v alternatívnom tvare $(p \Rightarrow q) \wedge (p \Rightarrow \overline{q}) \Rightarrow \overline{p}$, ktorý môžeme reprezentovať pomocou schémy

[3] Pri konštrukcii negácie $\overline{q}$ sme použili de Morganov zákon pre disjunkciu.

$$\frac{\begin{array}{l} p \Rightarrow q \\ p \Rightarrow \overline{q} \end{array}}{\overline{p}}$$

Stredoškolský príklad pre aplikáciu schémy „reductio ad absurdum“ je dôkaz, že $\sqrt{2}$ nie je racionálne číslo. Označme výrokovú premennú $p \sim \sqrt{2}$ *je racionálne číslo*, z tohto výroku vyplýva, že jeho alternatívny tvar je $p \sim \sqrt{2} = \alpha/\beta$. Z takto modifikované predpokladu vyplýva výrok $q \sim \alpha,\beta$ *sú celé nesúdeliteľné čísla*, t.j. platí implikácia $p \Rightarrow q$. Úpravou matematického výrazu z výroku p dostaneme formulu $2 = \alpha^2/\beta^2$, z ktorej vyplýva, že čísla α^2 a β^2 sú súdeliteľné, pomocou elementárnych úvah[4] je možné dokázať, že taktiež aj α a β musia byť súdeliteľné. Týmto sme vlastne dokázali $p \Rightarrow \overline{q}$. Súčasne platia teda dve implikácie, $p \Rightarrow q$ a $p \Rightarrow \overline{q}$, použitím schémy „reductio ad absurdum“ dostaneme, že platí negácia ich predpokladu, $\overline{p} \sim \sqrt{2}$ *nie je racionálne číslo*, čo bolo potrebné dokázať.

4.3 Predikátová logika

V hovorovej a odbornej reči sa často vyskytujú obraty:

objekt x má vlastnosť P,
objekty x, y sú vo vzťahu R,

kde vlastnosť P a vzťah – reláciu R budeme označovať ako *predikát*, objekty x a y budeme označovať ako *indivíduum*, formálne môžeme tieto výroky vyjadriť $P(x)$ a $R(x,y)$. Obe formulácie môžeme preformulovať pomocou použitia kvantifikátorov, ktoré kvantifikujú množstvo objektov - indivíduí. Prvá formulácia sa dá vyjadriť napr. dvoma spôsobmi ako

všetky objekty x majú vlastnosť P,
existuje objekt x majúci vlastnosť P.

Spojky „*všetky objekty x*“ a „*existuje objekt x*“ sú vyjadrené pomocou symbolu *všeobecného kvantifikátora* $\forall x$, resp. *existenčného kvantifikátora* $\exists x$. Pomocou týchto symbolov obe kvantifikované formulácie vyjadríme v predikátovej logiky takto

$\forall x\ P(x)$ = *všetky objekty x majú vlastnosť P,*
$\exists x\ P(x)$ = *existuje objekt x majúci vlastnosť P.*

Alternatívna reprezentácia kvantifikátorov sa môže zostrojiť v rámci výrokovej logiky pomocou sekvencie konjunkcií resp. disjunkcií

$$(\forall x)P(x) \Leftrightarrow P(a) \wedge P(b) \wedge \ldots \wedge P(z) \Leftrightarrow \bigwedge_x P(x)$$

$$(\exists x)P(x) \Leftrightarrow P(a) \vee P(b) \vee \ldots \vee P(z) \Leftrightarrow \bigvee_x P(x)$$

Pomocou tohto vyjadrenia ľahko zostrojíme negáciu kvantifikovaného výrazu pomocou de Morganových zákonov

[4] Podľa Základného teorému aritmetiky (hovorí, že každé prirodzené číslo sa dá vyjadriť ako násobok len jednej "sady" prvočísel a ich mocnín) sa číslo 2 musí objaviť v rozkladu čísla p^2 na násobky prvočísiel (pretože sa objaví v tom samom čísle vyjadrenom ako 2 q^2). Pretože 2 je samo o sebe prvočíslo, 2 sa musí teda objaviť v rozkladu čísla p na násobky prvočísiel. Potom ale by sa 2^2 objavilo v rozkladu čísla p^2 na násobky prvočísiel a teda aj v 2 q^2. Po podelení číslom 2 by stále ešte jedna dvojka ostala v v rozkladu čísla q^2 na násobky prvočísiel. Ako predtým, môžeme aj teraz vyvodiť že 2 je prvočíselným deliteľom čísla q. Ale potom by p aj q mali rovnakého prvočíselného deliteľa, teda 2. To je v rozpore s naším predpokladom hore, že p a q nemajú iného spoločného celočíselného deliteľa ako číslo 1.

$$\overline{(\forall x)P(x)} \Leftrightarrow \overline{P(a) \wedge P(b) \wedge ... \wedge P(z)} \Leftrightarrow \overline{P}(a) \vee \overline{P}(b) \vee ... \vee \overline{P}(z) \Leftrightarrow \bigvee_x \overline{P}(x) \Leftrightarrow \boxed{(\exists x)\overline{P}(x)}$$

$$\overline{(\exists x)P(x)} \Leftrightarrow \overline{P(a) \vee P(b) \vee ... \vee P(z)} \Leftrightarrow \overline{P}(a) \wedge \overline{P}(b) \wedge ... \wedge \overline{P}(z) \Leftrightarrow \bigwedge_x \overline{P}(x) \Leftrightarrow \boxed{(\forall x)\overline{P}(x)}$$

Tieto dva dôležité vzťahy pre negáciu formúl s kvantifikátorom sú ilustrované príkladmi

$(\forall x)P(x)$ ~ *každý vojak má zbraň*,

$\overline{(\forall x)P(x)} \Leftrightarrow (\exists x)\overline{P}(x)$ ~ *existuje vojak, ktorý nemá zbraň*

Pomocou kvantifikátorov môžeme efektívne sformalizovať rôzne verbálne tvrdenia. Tak napríklad, výrok „*každý riaditeľ má aspoň jedného podriadeného*“ vyjadríme takto

$$(\forall x)(\exists y)\big(R(x) \Rightarrow P(y,x)\big)$$

kde $R(x)$ znamená, že x je riaditeľ a $P(y,x)$ znamená, že y je podriadený x.

Ďalší ilustračný príklad je z diferenciálneho počtu, kde sa definuje limita postupnosti $a_1, a_2, ..., a_n, ... = \{a_n\}$ takto: hovoríme, že postupnosť $\{a_n\}$ má limitu a, ak pre každé $\varepsilon>0$ existuje taký index n_0, že pre každý index $n > n_0$ platí $|a - a_n| < \varepsilon$. Pomocou formalizmu predikátovej logiky túto definíciu postupnosti zapíšeme v kompaktnom tvare takto

$$(\forall \varepsilon > 0)(\exists n_0 > 0)(\forall n > n_0)\big(|a - a_n| < \varepsilon\big)$$

4.4 Sylogizmy

Sylogizmy [xx] boli vytvorené Aristotelom pred viac ako 2300 rokmi a odvtedy sú obvykle uvádzané ako neodmysliteľný základ racionality ľudského uvažovania. Aristoteles taktiež vypracoval kompletnú teóriu dôkazu, ktorá je založená na transformácii platných módov sylogizmov na platný mód prvej figúry (pozri tabuľky nižšie), ktorý pokladá za evidentne platný a pochopiteľný. Aristotelov revolučný intelektuálny počin je v súčasnosti chápaný nielen ako prvý zaznamenaný vznik logiky v histórii ľudskej civilizácie, ale taktiež ako prvý dokumentovaný výskyt premennej, čo sa v histórii vedy pokladá za jeden z najdôležitejších bodov obratu, ktorým sa napr. grécka civilizácia odlíšila od egyptskej civilizácie, ktorá ku pojmu „premennej“ nedospela. Sylogizmy stratili svoje mimoriadne postavenie v logike až koncom 19. storočia, keď Frege [xx] vypracoval predikátovú logiku, v rámci ktorej sú zaujímavou no nie veľmi dôležitou aplikáciou.

Sylogizmus obsahuje dve *premisy* - predpoklady (hlavný a vedľajší), ktoré obsahujú tri *členy* (premenné) A, B, C pričom *prostredný člen* B sa vyskytuje v obidvoch premisách. Záver sylogizmu má rovnakú štruktúru ako premisy sylogizmu a obsahuje členy A a C. Rozlišujeme 4 možné figúry sylogizmov

typ premisy	1. figúra	2. figúra	3. figúra	4. figúra
hlavná premisa	$A-B$	$B-A$	$A-B$	$B-A$
vedľajšia premisa	$B-C$	$C-B$	$C-B$	$B-C$

Premisy majú 4 rôzne módy

#	mód	stredoveké označenie	označenie v predikátovej logike
1	Všetky A sú B	*AaB*	$(\forall p)\big[A(p) \Rightarrow B(p)\big]$
2	Niektoré A sú B	*AiB*	$(\exists p)\big[A(p) \wedge B(p)\big]$

3	Všetky A nie sú B	AeB	$(\forall p)\left[A(p) \Rightarrow \overline{B}(p)\right]$
4	Niektoré A nie sú B	AoB	$(\exists p)\left[A(p) \wedge \overline{B}(p)\right]$

Počet všetkých možných sylogizmov je určený takto: 4 figúry × 4 módy hlavnej premisy × 4 módy vedľajšej premisy = 64.

Študujme nasledujúce tri výroky (dve premisy a záver)

niektorí kognitívni vedci sú vegetariáni
všetci vegetariáni sú včelári
---
niektorí kognitívni vedci sú včelári

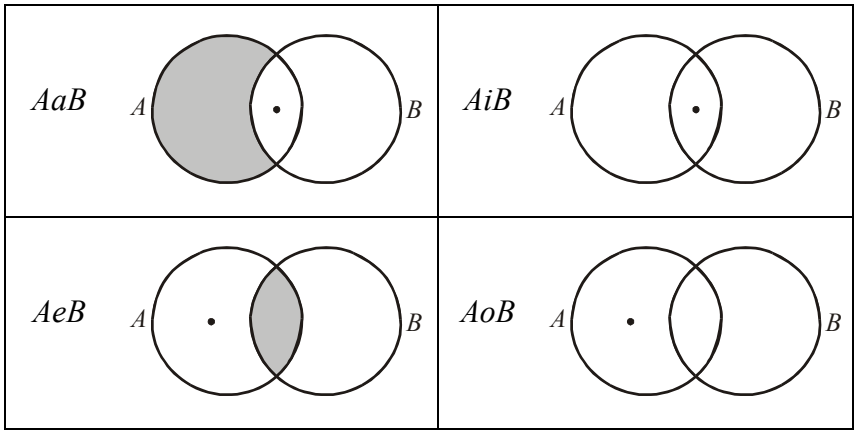
V tomto prípade jednotlivé členy A, B a C môžeme stotožniť s jednotlivými časťami výrok takto

kognitívny vedec ~ A
vegetarián ~ B
včelár ~ C

Takže výroky môžeme prepísať do formálneho tvaru

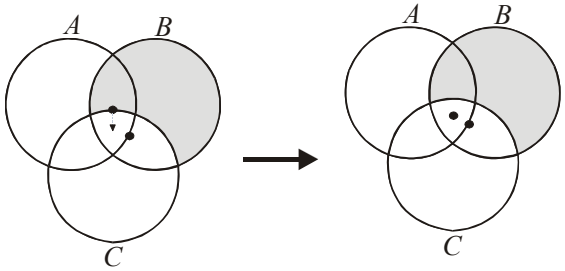
$$\begin{array}{ll} AiB & (\exists x)\left[A(x) \wedge B(x)\right] \\ \underline{BaC} \quad \text{alebo} & \underline{(\forall x)\left[B(x) \Rightarrow C(x)\right]} \\ AiC & (\exists x)\left[A(x) \wedge C(x)\right] \end{array}$$

Štruktúru, ktorá obsahuje tak dve premisy ako aj záver (ktorý má rovnaký formálny tvar ako premisy) budeme označovať ako úplný sylogizmus, ich celkový počet je 64 × 4 módy záveru = 256 úplných sylogizmov. *Hlavný problém teórie sylogizmov je navrhnúť metódu na rýchle zistenie, ktoré z týchto úplných sylogizmov sú pravdivé (platné) a ktoré nepravdivé (neplatné)*.



Obrázok 4.2. Interpretácia módov sylogizmov pomocou Vennových diagramov. Táto interpretácia vychádza z predikátovej formy jednotlivých premís, pričom členy A a B sa chápu ako množiny objektov, pre ktoré je daný člen pravdivý. Tak napríklad premisa „všetky A sú B“ alebo $(\forall p)\left[A(p) \Rightarrow B(p)\right]$ znamená, že množina objektov, ktoré súčasne patria do A a nepatria do B je prázdna ($A - B = \varnothing$) a že existuje aspoň jeden objekt, ktorý súčasne patrí do A a B ($A \cap B \neq \varnothing$). Tieto dve skutočnosti sú znázornené na Vennovom grafe tak, že oblasť $A - B$ je vyšrafovaná a oblasť $A \cap B$ obsahuje jeden bod – objekt (pozri príslušný Vennov graf pre premisu AaB).

Ak pomocou Vennových grafov (pozri obr. 4.2) chceme interpretovať obe premisy súčasne, musíme rozšíriť graf používajúci len dve kruhové oblasti na graf s troma kruhovými oblasťami, potom použitím vyššie uvedenej konvencie zavedieme do jednotlivých podmnožín informáciu, ktorá jednoznačne vyplýva z premís. Tak napr. premisy AiB a BaC majú nasledujúcu grafickú interpretáciu, pozri obr. 4.3.



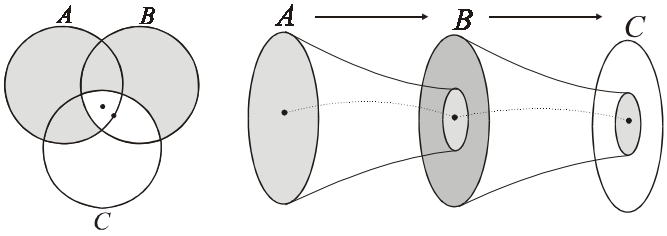
Obrázok 4.3. Grafická interpretácia módov *AiB* a *BaC* pomocou Vennových grafov. Ľavý graf obsahuje bod umiestnený v prieniku $A \cap B$ tak, že pôvodne ležal na hranici s množinou *C*, avšak podmienka prázdnosti *B*-*C* ho z tejto hraničnej polohy posunula do množiny *C*. Poznamenajme, že poloha bodu priamo na hranici dvoch oblastí znamená, že bod sa môže nachádzať buď v jednej, alebo v druhej oblasti. Z pravého grafu vyplýva, že existuje taký objekt (bod), ktorý leží v prekryve všetkých množín *A*, *B* a *C*, z toho vyplýva existencia takého A, ktoré je tiež C, čo odpovedá premise *AiC,* tento výsledok je záver – výsledok daného sylogizmu.

Pre lepšiu ilustráciu prístupu Vennových grafov uvedieme ešte ďalšie tri príklady.

Príklad 1. Uvažujem sylogizmus

$$\begin{array}{l} AaB \\ \underline{BaC} \\ AaC \end{array}$$

Grafická interpretácia má tvar z ktorého vypláva, že grafická interpretácia oboch premís implikuje taktiež grafickú interpretáciu výsledku, takže záver *AaC* je platný, pozri obr. 4.4.

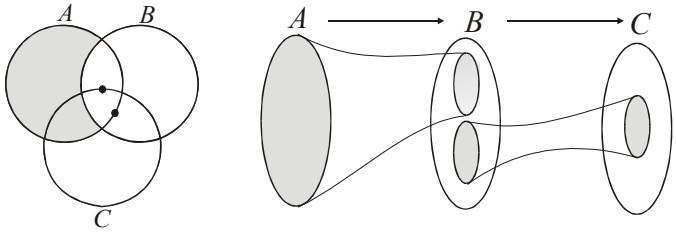


Obrázok 4.4. Pravý graf je analóg Vennovho grafu prekresleného do tvaru dvoch „zobrazení“, a to množiny *A* na množinu *B* (reprezentácia prvej premisy), a množiny *B* na množinu *C* (reprezentácia druhej premisy). Z tohto grafu priamo vidieť, že všetky objekty množiny *A* sú zobrazené na určitú podmnožinu množiny *B*, pričom v ďalšom kroku všetky objekty množiny *B* sú zobrazené na podmnožinu množiny *C*. Môžeme teda povedať, že všetky objekty *A* sú objektmi *C*, alebo naopak, existujú také objekty C, ktoré sú aj objektmi A. Tieto dva závery môžeme pokladať aj za závery, ktoré vyplývajú z daného sylogizmu.

Príklad 2. Uvažujme sylogizmus bez platného záveru

$$\begin{array}{l} AaB \\ \underline{BiC} \\ \varnothing \end{array}$$

jeho grafická interpretácia je znázornená na obr. 4.5.

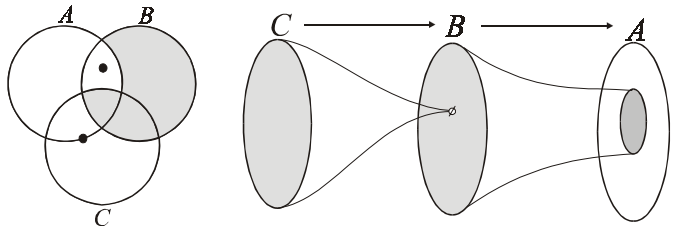


Obrázok 4.5. Graf na pravej strane má veľký heuristický význam pre správne vyhodnotenie sylogizmu. Obe premisy nešpecifikujú či podmnožiny v *B* majú neprázdny prekryv alebo nie, preto je potrebné predpokladať pesimistickejší variant, že podmnožiny majú prázdny prekryv. Z toho potom priamo vyplýva, že neexistuje záver, ktorý by súčasne špecifikoval členy *A* a *C*.

Príklad 3. Tento príklad obsahuje sylogizmus s jedným záporom

$$\begin{array}{l} BaA \\ \underline{CeB} \\ AoC \end{array}$$

s grafickou interpretáciou znázornenú na obr. 4.6.



Obrázok 4.6. Pravý diagram, idúc zľava doprava, reprezentuje druhú premisu, podľa ktorej žiadny objekt *C* nie je zobrazený na podmnožinu množiny *B*, potom je prvá premisa reprezentovaná zobrazením celej *B* na podmnožinu množiny *A*. Z tohto vyplýva záver sylogizmu, že existujú také objekty *A*, ktoré nie sú objektmi *C*.

Zo všetkých možných 64 sylogizmov len 27 má platný výsledok, ostatných 37 sylogizmov nemá platný výsledok.

1. figúra (6 sylogizmov)

1	*AaB* *BaC* *AaC*	2	*AaB* *BeC* *AeC*	3	*AiB* *BaC* *AiC*	4	*AiB* *BeC* *AoC*	5	*AeB* *BaC* *CoA*	6	*AeB* *BiC* *CoA*

2. figúra (6 sylogizmov)

7	*BaA* *CaB* *CaA*	8	*BaA* *CiB* *CiA*	9	*BaA* *CeB* *AoC*	10	*BiA* *CeB* *AoC*	11	*BeA* *CaB* *CeA*	12	*BeA* *CiB* *CoA*

3. figúra (6 sylogizmov)

13	*AaB* *CeB* *AeC*	14	*AaB* *CoB* *CoA*	15	*AiB* *CeB* *AoC*	16	*AeB* *CaB* *AoC*	17	*AeB* *CiB* *CoA*	18	*AoB* *CaB* *AoC*

4. figúra (9 sylogizmov)

19	$\begin{array}{l} BaA \\ \underline{BaC} \\ AiC \end{array}$	20	$\begin{array}{l} BaA \\ \underline{BiC} \\ AiC \end{array}$	21	$\begin{array}{l} BaA \\ \underline{BeC} \\ AoC \end{array}$	22	$\begin{array}{l} BaA \\ \underline{BoC} \\ AoC \end{array}$	23	$\begin{array}{l} BiA \\ \underline{BaC} \\ AiC \end{array}$	24	$\begin{array}{l} BiA \\ \underline{BeC} \\ AoC \end{array}$
25	$\begin{array}{l} BeA \\ \underline{BaC} \\ CoA \end{array}$	26	$\begin{array}{l} BeA \\ \underline{BiC} \\ CoA \end{array}$	27	$\begin{array}{l} BoA \\ \underline{BaC} \\ CoA \end{array}$						

Zhrnutie

Logika tvorí solídny základ našej racionality, študuje zákony podľa ktorých sa riadi naše usudzovanie. Formálna logika je integrálnou súčasťou informatiky, je teoretickým základom počítačových metód pre spracovanie informácie, strojového usudzovania nad databázami poznatkov. Podrobná znalosť zákonov logiky a ich formalizácia nám umožňuje pochopiť a interpretovať procesy ľudského usudzovania, simulovať ich pomocou formálno-symbolických modelov. Znalosť základných logických zákonov patrí k všeobecnému vzdelaniu nielen vo vedách prírodných ale aj humanitných.

Úlohy

Úloha 4.1. Overte pomocou tabuľkovej metódy, ktoré z formúl sú tautológie, kontradikcie a splniteľné: (1) $p \Rightarrow (q \Rightarrow p)$, (2) $\big((p \Rightarrow q) \wedge (q \Rightarrow r)\big) \Rightarrow (p \Rightarrow r)$, (3) $(p \Rightarrow q) \Rightarrow (\overline{q} \Rightarrow \overline{p})$, (4) $(p \wedge q) \Rightarrow (p \vee q)$, (5) $p \Rightarrow (\overline{p} \Rightarrow q)$, (6) $\big((p \Rightarrow q) \Rightarrow p\big) \Rightarrow q$.

Úloha 4.2. Zostrojte syntaktické stromy z úlohy 4.1.

Úloha 4.3. Pretransformujte formule z úlohy 4.1 do disjunktného tvaru.

Úloha 4.4. Doplňte výsledok u nasledujúcich schém usudzovania

(1) $\begin{array}{l} p \Rightarrow q \\ \underline{\overline{p}} \\ ? \end{array}$, (2) $\begin{array}{l} p \Rightarrow \overline{q} \\ \underline{p} \\ ? \end{array}$, (3) $\begin{array}{l} p \Rightarrow q \\ \underline{q} \\ ? \end{array}$, (4) $\begin{array}{l} q \Rightarrow p \\ \underline{\overline{p}} \\ ? \end{array}$, (5) $\begin{array}{l} \overline{p} \Rightarrow \overline{q} \\ \underline{q} \\ ? \end{array}$, (6) $\begin{array}{l} p \Rightarrow \overline{q} \\ \underline{p \Rightarrow q} \\ ? \end{array}$.

Úloha 4.5. Pomocou formálnych prostriedkov výrokovej logiky zapíšte nasledujúce výroky prirodzeného jazyka a potom vykonajte ich negáciu:
(1) Každý lekár je absolventom lekárskej fakulty
(2) Existuje taký ekonóm, ktorý absolvoval technickú alebo ekonomickú univerzitu.
(3) V každom meste je radnica a v niektorých obciach nie je.

Úloha 4.6. Pomocou techniky Vennových diagramov zistite, či nasledujúce sylogizmy sú platné alebo nie, ak áno, tak nájdite riešenie.

$$(1)\ \frac{\begin{matrix}AaB\\BiC\end{matrix}}{?},\quad (2)\ \frac{\begin{matrix}BaA\\CiB\end{matrix}}{?},\quad (3)\ \frac{\begin{matrix}AaB\\CiB\end{matrix}}{?},\quad (4)\ \frac{\begin{matrix}BeA\\BaC\end{matrix}}{?},\quad (5)\ \frac{\begin{matrix}BiA\\BoC\end{matrix}}{?}$$

Otázky na opakovanie

Otázka 4.1. Čo je výrok? Uveďte príklady výrokov. Čo je pravdivostná hodnota výroku?

Otázka 4.2. Aké spojky výrokovej logiky poznáme? Uveďte ich pravdivostné tabuľky.

Otázka 4.3. Ako je definovaná výroková formula, čo je pravdivostná tabuľka výrokovej formuly?

Otázka 4.4. Čo je tautológia, kontradikcia a splniteľná formula?

Otázka 4.5. Čo je disjunktný tvar výrokovej formule?

Otázka 4.6. Čo je modus ponens, modus tollens a dôkaz sporom?

Otázka 4.7. Čo je všeobecný a existenčný kvantifikátor, čo je ich negácia?

Otázka 4.8. Čo je sylogizmus, aké sú jeho základné módy a ich značenie v predikátovej logike?

Otázka 4.9. Aká je reprezentácia jednotlivých módov pomocou Vennových grafov?

5 Mozog a neurónové siete

Metafora ľudského mozgu hrá dôležitú úlohu v modernej informatike. Pomocou tejto metafory boli navrhnuté nové paralelné prístupy k spracovaniu informácie, ktoré sa zásadným spôsobom odlišujú od klasickej sekvenčnej architektúry počítačov podľa von Neumanna (pozri kapitolu 1). Tento nový prístup k spracovaniu informácie je reprezentovaný teóriou umelých neurónových sietí, ktorá v súčasnosti tvorí nielen efektívny informatický nástroj na tvorbu a návrh nových paralelných prístupov k riešeniu problémov umelej inteligencie, ale už je aj integrálnou súčasťou modernej neurovedy, pomocou ktorej sa pristupuje k počítačovým simuláciám procesov prebiehajúcich v mozgu.

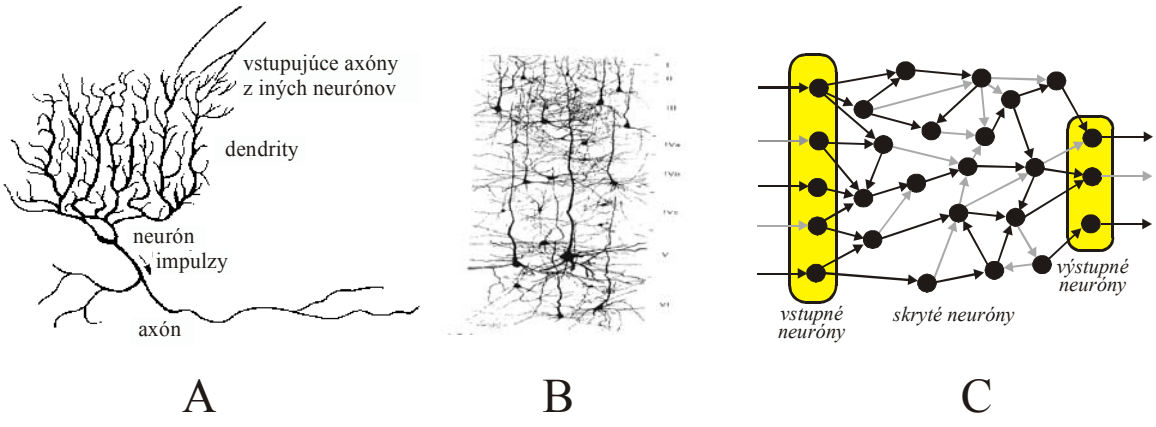
Neurónové siete sú založené na postuláte neurovedy hovoriacom, že základným stavebným kameňom ľudského mozgu je neurón, ktorý má tieto tri najdôležitejšie vlastnosti [xx] (obr. 5.1):

(1) neurón má orgán, pomocou ktorého *prijíma signály* z okolia od ostatných neurónov (dendrity),

(2) neurón *spracováva* (*integruje*) prijaté signály,

(3) neurón má orgán (axón) pomocou ktorého *posiela spracované vstupné signály iným neurónom zo svojho okolia*.

Neuróny sú poprepájané medzi sebou do zložitej sieťovej štruktúry (nazývanej neurónová sieť), pričom jednotlivé *spoje majú buď excitačný alebo inhibičný charakter*. Systém spojov a ich excitačný resp. inhibičný charakter tvoria *architektúru neurónovej siete*, ktorá jediná určuje vlastnosti neurónovej siete (t.j. transformáciu vstupnej informácie na výstupnú informáciu).

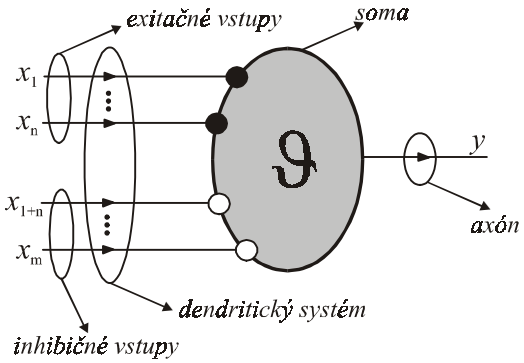


Obrázok 5.1. (A) Typická neurónová bunka, ktorá obsahuje rozsiahly dendritický systém a dlhý vetviaci sa axón. Prostredníctvom dendritického systému do neurónu vstupujú signály z iných neurónov a prostredníctvom axónu z neurónu vystupuje signál charakterizujúci stav neurónu. (B) Vzájomné prepojenie neurónov pomocou spojov medzi dendritmi a axónmi, ktoré je pozorované v mikroanatómii ľudského mozgu. (C) Architektúra neurónovej siete s dvoma druhmi spojov medzi neurónmi, tmavé (svetlé) spoje znázorňujú exitačné (inhibičné) spoje. Neuróny v sieti sú rozdelené na vstupné neuróny (pomocou ktorých do siete vstupuje informácia), výstupné neuróny (pomocou ktorých zo siete vystupuje informácia) a skryté neuróny (spracovávajú vstupnú informáciu na výstupnú informáciu).

5.1 Logické neuróny a teória neurónových sietí

Budeme sa zaoberať najjednoduchším typom neurónových sietí, ktoré boli navrhnuté už pred viac ako polstoročím McCullochom a Pittsom v r. 1943. Ich model neurónovej siete je v súčasnosti pokladaný za významný medzník v rozvoji teórie neurónových sietí. V ich práci bolo ukázané, že neurónové siete sú mocným informatickým simulačným prostriedkom, sú schopným simulovať ľubovoľnú formulu výrokovej logiky.

Elementárnou jednotkou McCullochovej a Pittsovej neurónovej siete je *logický neurón* (výpočtová jednotka), pričom stav neurónu je binárny (tj. má dva možné stavy, 1 a 0). Predpokladajme, že dendritický systém logického neurónu obsahuje tak *excitačné vstupy* (opísané binárnymi premennými x_1, x_2, ..., x_n, ktoré zosilňujú odozvu), ako aj *inhibičné vstupy* (opísané binárnymi premennými x_{n+1}, x_{n+2}, ..., x_m, ktoré zoslabujú odozvu), pozri obr. 5.2.



Obrázok 5.2. Znázornenie logického neurónu, ktorý obsahuje dendritický systém pre vstupné (excitačné alebo inhibičné) aktivity a axón pre výstup neurónovej aktivity. Soma (telo neurónu) je charakterizovaná prahovým koeficientom ϑ.

Aktivita logického neurónu je jednotková, ak suma excitačných vstupných aktivít mínus suma inhibičných aktivít je väčšia alebo rovná prahu ϑ, v opačnom prípade je nulová

$$y = \begin{cases} 1 & \left(x_1 + \ldots + x_n - x_{1+n} - \ldots - x_m \geq \vartheta\right) \\ 0 & \left(\text{inak}\right) \end{cases}$$

Pomocou krokovej funkcie

$$s\left(\xi\right) = \begin{cases} 1 & \left(\text{ak } \xi \geq 0\right) \\ 0 & \left(\text{ak } \xi < 0\right) \end{cases}$$

môžeme aktivitu z vyjadriť takto

$$y = s\left(x_1 + \ldots + x_n - x_{1+n} - \ldots - x_m - \vartheta\right)$$

Tieto vzťahy pre aktivitu logického neurónu môžeme jednoducho interpretovať tak, že excitačné aktivity vstupujú do neurónu cez spoje, ktoré sú ohodnotené jednotkovým váhovým koeficientom (w=1), zatiaľ čo inhibičné aktivity vstupujú do neurónu cez spoje so záporným jednotkovým váhovým koeficientom (w=-1). Potom aktivitu logického neurónu môžeme vyjadriť takto

$$y = s\left(w_1 x_1 + \ldots + w_m x_m - \vartheta\right) = s\left(\sum_{i=1}^{m} w_i x_i - \vartheta\right)$$

Jednoduchá implementácia elementárnych Boolových funkcií[1] *OR*, *AND* a *NOT* (logické spojky disjunkcie, konjunkcie a negácie, pozri ich pravdivostnú tabuľku

[1] Pod Boolovou funkciou rozumieme takú funkciu, ktorej argumenty sú binárne čísla 0 a 1 a funkčné hodnoty sú taktiež binárne čísla 0 a 1. Vo všeobecnosti, Boolova funkcia je zobrazenie $f: \{0,1\}^n \rightarrow \{0,1\}$, ktoré priradzuje n-tici binárnych čísel binárne číslo, $y=f(x_1,x_2,...,x_n)$. Príkladom Boolovej funkcie je logická spojka konjunkcie

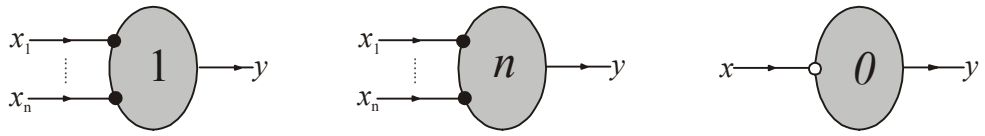
v predchádzajúcej kapitole 4) je znázornená na obr. 5.3. Pre ilustráciu budeme študovať funkciu *OR* pre n=2, použitím formúl z definície logického neurónu dostaneme

$$y_{OR}\left(x_1, x_2\right) = s\left(x_1 + x_2 - 1\right)$$

Funkčné hodnoty tejto Boolovej funkcie sú tieto

x_1	x_2	$y_{OR}(x_1,x_2)$	$x_1 \vee x_2$
0	0	$s(-1)$	0
0	1	$s(0)$	1
1	0	$s(0)$	1
1	1	$s(1)$	1

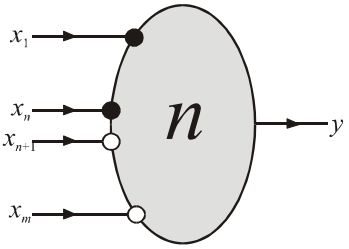
Z tabuľky vyplýva, že Boolova funkcia y_{OR} simuluje Boolovu funkciu disjunkcie *OR*.



Boolova funkcia OR $y=x_1 \vee \ldots \vee x_n$ | *Boolova funkcia AND* $y=x_1 \wedge \ldots \wedge x_n$ | *Boolova funkcia NOT* $y=\overline{x}$

Obrázok 5.3. Tri realizácie logických neurónov pre implementáciu Boolových funkcií *OR, AND a NOT*. Excitačné spoje sú znázornené plným krúžkom, inhibičné prázdnym krúžkom.

Jednoduchým zovšeobecnením týchto vyjadrení elementárnych Boolových funkcií *AND* a *NOT* je možnosť reprezentovať logickým neurónom Boolovu funkciu vyjadrenú pomocou konjunkcií konečného počtu výrokových premenných a ich negácií, $x_1 \wedge \ldots \wedge x_n \wedge \overline{x}_{n+1} \wedge \ldots \wedge \overline{x}_m$, pozri obr. 5.4. Táto Boolova funkcia sa rovná 1 (pravdivý výrok) len pre $x_1 = \ldots = x_n = 1$ a $x_{n+1} = \ldots = x_m = 0$, vo všetkých ostatných pravdivostných kombináciách argumentov jej hodnota je 0 (nepravdivý výrok).



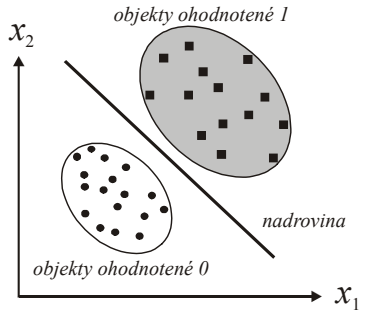
Obrázok 5.4. Logický neurón, ktorý simuluje konjunkciu konečného prvku výrokových premenných alebo ich negácií, $y = x_1 \wedge \ldots \wedge x_n \wedge \overline{x}_{n+1} \wedge \ldots \wedge \overline{x}_m$.

Môžeme si položiť otázku, aké Boolove funkcie je schopný logický neurón vyjadriť? Táto otázka sa dá pomerne jednoducho vyriešiť pomocou geometrickej interpretácie výpočtu prebiehajúceho v logickom neuróne. Výpočtová funkcia logického neurónu rozdeľuje priestor vstupov na dva polpriestory pomocou roviny $w_1x_1+w_2x_2+\ldots+w_nx_n=\vartheta$, pre koeficienty $w_i=0,\pm1$ (pozri diagram B). Hovoríme, že Boolova funkcia $f(x_1,x_2,\ldots,x_n)$ je *lineárne separovateľná*, ak existuje taká rovina $w_1x_1+w_2x_2+\ldots+w_nx_n=\vartheta$, ktorá separuje priestor vstupných aktivít tak, že v

---

(Boolova funkcia *AND*), potom $y_{AND} = f\left(x_1, x_2\right)$, kde funkčná hodnota $y_{AND} = 1$ len pre $x_1 = x_2 = 1$, pre všetky ostatné hodnoty argumentov je rovná 0. V našich úvahách môžeme pokladať termín „Boolova funkcia“ ekvivalentný termínu „výroková formula“.

jednej časti priestoru sú vrcholy ohodnotené 0, zatiaľ čo v druhej časti priestoru sú vrcholy ohodnotené 1 (pozri obr. 5.5).



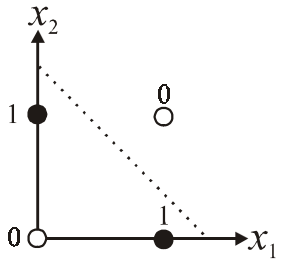
Obrázok 5.5. Schematické znázornenie pojmu lineárnej separovateľnosti, kde okrúhle a štvorcové objekty sú separované nadrovinou $w_1x_1+...+w_nx_n-\vartheta=0$ tak, že v jednom polopriestore sú objekty jedného druhu, zatiaľ čo v druhom polopriestore sú objekty druhého druhu.

Veta 5.1. *Logický neurón je schopný simulovať len Boolove funkcie, ktoré sú lineárne separovateľné.*

Klasický príklad Boolovej funkcie, ktorá nie je lineárne separovateľná, je výroková spojka „*vylučujúca disjunkcia*“[2], ktorá je formálne definovaná ako negácia ekvivalencie $\left(x \dot{\vee} y\right) \Leftrightarrow \left(\overline{x \Leftrightarrow y}\right)$, v informatickej literatúre sa obvykle označuje ako Boolova funkcia *XOR* (e*X*clusive *OR*), $f_{XOR}\left(x,y\right) = x \dot{\vee} y$, jej tabuľka funkčných hodnôt (vo výrokovej logike nazývaná pravdivostná tabuľka) má tvar

#	x	y	$f_{XOR}(x,y)$
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

Ak si jej funkčné hodnoty vynesieme do priestoru *x*-*y* dostaneme obr. 5.6, z ktorého jasne vyplýva, že táto funkcia nie je lineárne separovateľná.



Obrázok 5.6. Znázornenie 4 objektov Boolovej funkcie *XOR* v priestore jej argumentov. Z obrázku jasne plynie, že neexistuje priamka, ktorá by separovala celú rovinu na dve polroviny tak, že objekty (otvorené krúžky) s ohodnotením 0 budú v jednej polrovine, zatiaľ čo objekty s hodnotením 1 (čierne krúžky) budú v druhej polrovine.

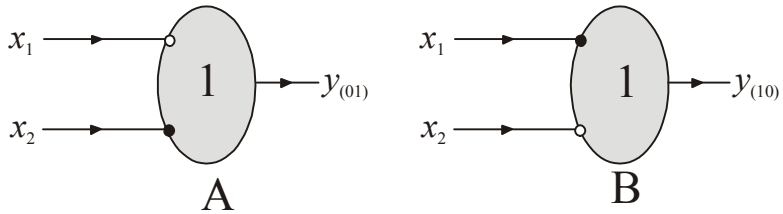
[2] Táto funkcia zohrala dôležitú úlohu v histórii neurónových sietí. Bola spomínaná Minskym a Papertom koncom 60-tych rokov v ich slávnej knihe *Perceptron* [xx], ako lineárne neseparovateľná funkcia, ktorá nemôže byť korektne klasifikovaná jedným neurónom.

Aká je schopnosť neurónových sietí zložených z logických neurónov reprezentovať výrokovú formulu? V predchádzajúcej kapitole 4 sme dokázali, že ľubovolná výroková formula môže byť prepísaná do ekvivalentného disjunktívneho tvaru, ktorý obsahuje disjunkcie formulí zložených z konečného počtu konjunkcií výrokových premenných a ich negácií (pozri veta 4.1). Ukážeme metódu, ako reprezentovať ľubovolnú Boolovu funkciu (teda aj funkciu *XOR*) pomocou 3-vrstvovej neurónovej siete obsahujúcej logické neuróny. Prístup budeme ilustrovať pomocou Boolovej funkcie *XOR*

#	argument	funkčná hodnota
1	(00)	0
2	(01)	1
3	(10)	1
4	(11)	0

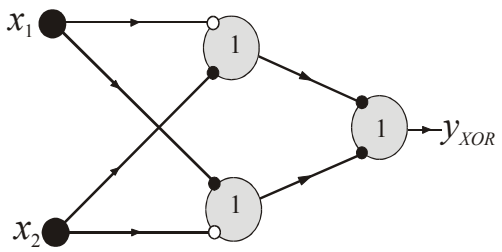
Označené riadky tejto tabuľky zodpovedajú jednotkovým funkčným hodnotám, priradíme im logické neuróny typu *AND* tak, aby vstupné aktivity poskytovali jednotkové výstupné aktivity; tieto riadky sú reprezentované disjunktívnymi formulami tvaru $\overline{x}_1 \wedge x_2$ resp. $x_1 \wedge \overline{x}_2$. Použitím logického neurónu z obr. 5.4 môžeme jednoducho zostrojiť reprezentácie týchto dvoch formúl pomocou logických neurónov (pozri obr. 5.7). Výsledná funkcia *XOR* potom môže byť vyjadrená pomocou disjunkcie dvoch disjunktívnych formúl prislúchajúcich riadkom s pravdivostnou hodnotou 1

$$f_{XOR}(x_1, x_2) = (\overline{x}_1 \wedge x_2) \vee (x_1 \wedge \overline{x}_2)$$



Obrázok 5.7. Reprezentácie disjunktívnych formúl $\overline{x}_1 \wedge x_2$ a $x_1 \wedge \overline{x}_2$, ktoré zodpovedajú označeným riadkom v tabuľke s jednotkovou pravdivostnou hodnotou. Tieto Boolove funkcie majú tú dôležitú vlastnosť, že majú nulovú funkčnú hodnotu pre všetky iné argumenty než ten, pre ktorý boli zostrojené. Tak napríklad Boolova funkcia $\overline{x}_1 \wedge x_2$ má jednotkovú hodnotu len pre (01), pre všetky ostatné argumenty má nulovú funkčnú hodnotu.

Pomocou takto určenej funkcie *XOR* môžeme ľahko zostrojiť neurónovú 3-vrstvovú sieť, ktorá simuluje túto funkciu, pozri obr. 5.8.



Obrázok 5.8. Reprezentácia Boolovej funkcie *XOR* pomocou 3-vrstvovej neurónovej siete. Prvá vrstva (spodná) vyjadruje tzv. vstupné neuróny, ktoré nevykonávajú výpočty, ich úlohou je preniesť vstupné aktivity do ďalšej vrstvy. Druhá vrstva obsahuje tzv. skryté neuróny, ktoré sú reprezentované logickými neurónmi z obr. 5.7. Posledná tretia vrstva obsahuje jeden výstupný logický neurón so vstupmi zo skrytých neurónov, tento neurón vykonáva elementárnu Boolovu funkciu *OR*.

Podobný postup, ktorý bol naznačený vyššie pre Boolovu funkciu *XOR*, môže byť aplikovaný pre konštrukciu 3-vrstvovej neurónovej siete realizujúcej ľubovoľnú Boolovu funkciu obsahujúcej *n* výrokových premenných, $f: \{0,1\}^n \to \{0,1\}$. V prvom kroku zostrojíme neurálnu reprezentáciu všetkých argumentov, ktoré zodpovedajú jednotkovým funkčným hodnotám, a v druhom kroku tieto neuróny spojíme pomocou jedného neurónu reprezentujúceho funkciu *OR*.

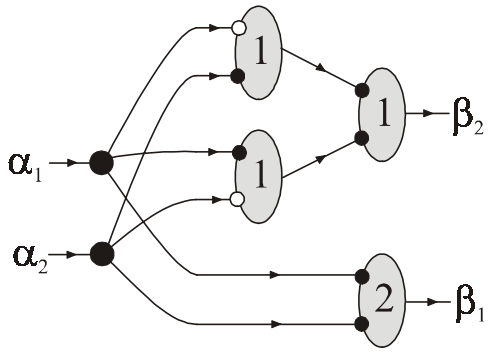
Veta 5.2. *Ľubovoľná Boolova funkcia* $f: \{0,1\}^n \to \{0,1\}$ *sa dá reprezentovať pomocou 3-vrstvovej neurónovej siete.*

Táto veta má pre teóriu logických neurónov navrhnutých už pred viac ako polstoročím McCullochom a Pittsom fundamentálny význam. Inými slovami, 3-vrstvové neurónové siete obsahujúce logické neuróny sú *univerzálne výpočtové zariadenie*, pomocou nich možno reprezentovať každú Boolovu funkciu. Navyše, pretože dôkaz vety bol konštruktívny, poznáme jednoduchý postup, ako túto sieť systematickým spôsobom zostrojiť pre ľubovoľnú Boolovu funkciu. Žiaľ, problém minimálnosti takto zostrojenej siete nie je vetou riešený. Vo všeobecnosti môže existovať sieť, ktorá pre danú Boolovu sieť obsahuje menej logických (skrytých) neurónov ako tá, ktorá bola zostrojená systematickým postupom z dôkazu vety.

Na záver tejto kapitoly o logických neurónoch ukážeme ich jednoduchú aplikáciu na sčítanie dvoch binárnych číslic

$$\begin{array}{r} \alpha_1 \\ \alpha_2 \\ \hline \beta_1\beta_2 \end{array}$$

kde jednotlivé binárne premenné výsledku – sumy sú určené takto: $\beta_2 = \alpha_1 \dot{\vee} \alpha_2$ a $\beta_1 = \alpha_1 \wedge \alpha_2$. Výslednú premennú β_2 prepíšeme do disjunktívnej formy $\beta_2 = (\overline{\alpha}_1 \wedge \alpha_2) \vee (\alpha_1 \wedge \overline{\alpha}_2)$, príslušná neurónová sieť je znázornená na obrázku 5.9.



Obrázok 5.9. Neurónová sieť vykonávajúca súčet dvoch binárnych číslic.

Na záver tejto podkapitoly poznamenajme, že aj keď sú logické neuróny veľmi jednoduché, neurónové siete z nich zložené sú univerzálnym výpočtovým prostriedkom pre Boolove funkcie, a implicitne už obsahujú vlastnosti neurónových sietí, ktoré sú zložené zo zložitejších neurónov. Hlavný nedostatok neurónových sietí zložených z logických neurónov je neprítomnosť procesu učenia v ktorom by sa menila architektúra neurónovej siete a hodnoty váhových a prahových neurónov. V prezentovanom prístupe je architektúra neurónovej siete jednoznačne určená pravdivostnou tabuľkou študovanej Boolovej funkcie; popísali sme jednoduchý postup ako zostrojiť z logických neurónov sieť, ktorá presne simuluje danú Boolovu funkciu. Z pohľadu moderného rozvoja teórie neurónových sietí je úloha postavená tak, že pre náhodne zostrojenú neurónovú sieť (zloženú napr. z logických neurónov) hľadáme

také váhové a prahové koeficienty, aby výstupná hodnota siete y bola totožná s požadovanou hodnotou y_{req}. Tieto a podobné problémy budú študované v nasledujúcej podkapitole.

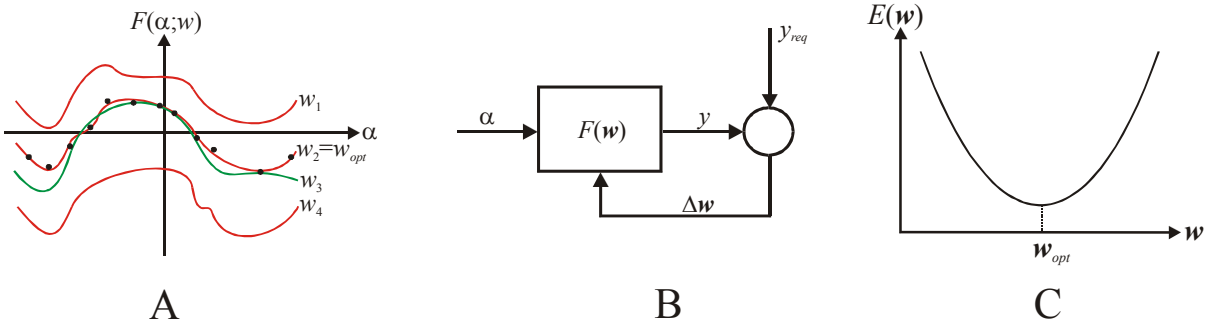
5.2 Plasticita a učenie

Dôležitá vlastnosť ľudského mozgu je tzv. *plasticita*, t.j. mozog je schopný meniť svoju architektúru buď vznikom nových spojov alebo zánikom už existujúcich spojov. Táto výnimočná vlastnosť mozgu sa odráža v schopnosti mozgu *učiť sa*, keď v priebehu učenia mení svoju architektúru tak, aby čo najlepšie klasifikoval požadovaným spôsobom predkladané objekty. V teórii neurónových sietí je koncepcia plasticity automaticky zabudovaná, váhové a prahové koeficienty sú voľne nastaviteľné. Plasticita neurónovej siete úzko súvisí s jej schopnosťou učiť sa, t.j. meniť váhové a prahové koeficienty tak, aby neurónová sieť čo najlepšie vykonávala požadované výpočty.

Pre lepšie pochopenie tejto dôležitej idey v teórii neurónových sietí študujme Boolovu funkciu $f : \{0,1\}^n \rightarrow \{0,1\}$, ktorá každý binárny vektor $\alpha \in \{0,1\}^n$ dĺžky n ohodnocuje buď pravdivostnou hodnotou 1 alebo 0. Táto Boolova funkcia je reprezentovaná pravdivostnou tabuľkou s 2^n riadkami. Táto tabuľka môže byť alternatívne vyjadrená pomocou tréningovej množiny

$$A_{train} = \left\{ \alpha \middle/ y_{req} \, ; \alpha \in \{0,1\}^n , y_{req} = f(\alpha) \right\}$$

Obsahuje všetky možné dvojice zložené z argumentu, α, a z požadovanej funkčnej hodnoty, $f(\alpha)$. Počet elementov v tréningovej množine sa rovná počtu riadkov v pravdivostnej tabuľke, $|A_{train}| = 2^n$.



Obrázok 5.10. Diagram A znázorňuje priebeh funkcie $y = F(\alpha; \boldsymbol{w})$ pre rôzne hodnoty parametru $\boldsymbol{w}$, pre optimálnu hodnotu $\boldsymbol{w}_{opt}$ odpovedajúci graf funkcie F leží najbližšie k bodom tréningovej množiny. Diagram B vyjadruje proces učenia sa spočívajúci v minimalizácii odchýlok medzi vypočítanou výstupnou hodnotou a požadovanou výstupnou hodnotou. V prípade, že táto odchýlka je nenulová, potom oprava $\Delta\boldsymbol{w}$ sa pripočíta k aktuálnej hodnote $\boldsymbol{w}$, tento proces sa opakuje tak dlho, až odchýlky pre všetky dvojice tréningovej množiny sú nulové. Tento postup učenia (nazývaný učenie s učiteľom alebo s dohľadom) predpokladá existenciu metódy výpočtu opráv $\Delta\boldsymbol{w}$ na základe rozdielu medzi vypočítanou hodnotou y a požadovanou hodnotou z tréningovej množiny. Diagram C ukazuje význam optimálnej hodnoty $\boldsymbol{w}_{opt}$, ktorej získanie je cieľom učenia, pretože minimalizuje účelovú funkciu E.

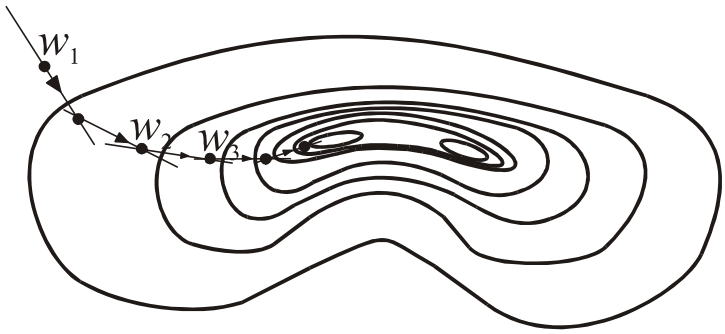
Po týchto prípravných úvahách môžeme už pristúpiť k špecifikácii učenia neurónových sietí (pozri obr. 5.10). Neurónovú sieť môžeme chápať ako funkciu - zobrazenie (závislé od váhových koeficientov w a prahových koeficientov ϑ), ktoré každému binárnemu vektoru $\alpha \in \{0,1\}^n$ priradí binárnu hodnotu $y \in \{0,1\}$, $y = F(\alpha; \boldsymbol{w})$. Definujme účelovú funkciu chýb

$$E(\boldsymbol{w}) = \sum_{A_{train}} \left[F(\alpha; \boldsymbol{w}) - y_{req} \right]^2$$

ktorá je definovaná ako suma štvorcov odchýlok medzi vypočítaným výstupom y a požadovaným výstupom y_{req}. V ideálnom prípade, keď vypočítané hodnoty výstupu sú totožné s požadovaným výstupom, hodnota tejto účelovej funkcie je nulová. Z tohto pohľadu môžeme chápať *učenie ako minimalizačný proces účelovej funkcie* $E(\boldsymbol{w})$, pri ktorom sa hľadá taká optimálna hodnota $\boldsymbol{w}_{opt}$, pre ktorú účelová funkcia nadobudne minimum (pozri diagram C, obr. 5.10).

$$E(\boldsymbol{w}_{opt}) = \min_{w} E(\boldsymbol{w})$$

Môžeme sa zamýšľať ako realizovať učenie neurónovej siete tak, aby navrhnutá metóda bola biologicky prijateľná. Z pohľadu čisto matematického, riešenie minimalizačného problému účelovej funkcie je možné vykonať celou plejádou optimalizačných metód numerickej matematiky, žiaľ len s veľmi malou biologickou prijateľnosťou optimalizačnej metódy. Použitie známej optimalizačnej metódy „najprudšieho spádu“ je ilustrované na obr. 5.11. Táto metóda sa často používa v učení neurónových sietí, avšak tento prístup, ako už bolo poznamenané, má len veľmi malú biologickú prijateľnosť. Preto je v teórii neurónových sietí všeobecnou snahou používať také metódy učenia, ktoré, aj keď priamo neminimalizujú účelovú funkciu E, sú biologicky prijateľné a môžu slúžiť pre štúdium procesov učenia prebiehajúcich v ľudskom mozgu. Nebudeme ďalej diskutovať problém učenia v neurónových sieťach. Jedná sa o veľmi zložitý problém, ktorý, tak na teoretickej, ako aj na experimentálnej úrovni, nie je celkom vyjasnený.



Obrázok 5.11. Znázornenie optimalizačnej metódy *najprudšieho spádu*. Povrch účelovej funkcie je znázornený pomocou vrstevníc, na ktorých má účelová funkcia konštantnú hodnotu. Metóda je inicializovaná náhodne určeným bodom $\boldsymbol{w}_1$ (t.j. váhové koeficienty sú náhodne generované), v tomto bode zistíme smer kolmý na vrstevnicu v ktorom funkcia najrýchlejšie klesá. V tomto smere nájdeme ďalší bod $\boldsymbol{w}_2$, opäť v tomto bode určíme smer kolmý na vrstevnicu, v ktorom účelová funkcia najrýchlejšie klesá a opäť v tomto smere určíme nový bod $\boldsymbol{w}_3$. Tento postup skončíme vtedy, ak aktuálne riešenie predstavuje minimum, t.j. každá výchylka z tohto riešenie už odpovedá len zvýšeniu hodnoty účelovej funkcie. Obrazne povedané, metódu najprudšieho spádu môžeme porovnať s pohybom guľôčky vplyvom gravitácie po povrchu funkcie do najbližšieho minima na povrchu.

5.3 Neurónové siete a vzťah medzi mozgom a mysľou

Ako už bolo poznamenané v úvodnej časti kapitoly 3 (venovanej umelej inteligencii), jeden z hlavných dlhodobých cieľov informatiky je implementácia počítačového „superprogramu“, ktorý bude analógiou ľudskej mysle, bude vykazovať podobné aktivity a schopnosti ako ľudský mozog. V tejto súvislosti stojíme pred zložitými otázkami, čo je ľudská myseľ, aký je jej vzťah k mozgu, môžeme chápať ľudskú myseľ ako program implementovaný v počítači mozgu, aká je architektúra tohto programu? Takýchto a podobných otázok môžeme si položiť

veľmi mnoho. Žiaľ odpovedať na ne nie je vôbec jednoduché, preto v súčasnosti sú stále predmetom záujmu oblasti filozofie nazývanej „*filozofia mysle*“ [xx]. Cieľom tejto záverečnej podkapitoly je poskytnúť jednoduchý informatický pohľad na vzťah medzi mysľou a mozgom, ktorý je založený na všeobecných predstavách teórie neurónových sietí (nazývanej v kognitívnej vede[3] „*konekcionizmus*“) [xx].

V rámci *konekcionistického prístupu* [xx] k štúdiu vzťahu medzi mozgom a mysľou je tento problém riešený pomocou koncepcií modernej neurovedy a neurónových sietí [xx]. Podľa jednej zo základných paradigiem modernej kognitívnej vedy [xx] *je mozog/myseľ charakterizovaný ako počítač*

(1) ktorý transformuje symboly pomocou syntaktických pravidiel na iné symboly, pričom

(2) myšlienky sú symbolické reprezentácie implementované pomocou jazyka myslenia, a

(3) mentálne procesy sú postupnosti symbolov (medzi ktorými sú príčinné vzťahy) generované syntaktickými pravidlami.

Použitie termínu „počítač“ obvykle evokuje predstavu sekvenčného počítača von neumannovskej architektúry (napr. personálne počítače majú túto architektúru), kde je možné striktne oddeliť hardware od software; kde na tom istom počítači – hardware môže byť vykonávaných nepreberné množstvo naprosto rozdielnych programov – softvérov. Pre tieto počítače existuje striktná dichotómia medzi počítačom a programom – t. j. medzi hardvérom a softvérom. Žiaľ, paradigma mysle ako počítača implikuje u mnohých ľudí predstavu, že je možné oddeliť mozog od mysle, ako dva „nezávislé“ fenomény, kde mozog hrá úlohu hardvéru, zatiaľ čo myseľ je softvér (vykonávaný na hardvéru – mozgu). V tejto súvislosti sa tiež hovorí o sociokultúrnom zdroji ľudskej mysle, kde mozog akoby bol len irelevantným „hardwarovým“ vehiklom mysle.

Obráťme našu pozornosť na moderný prístup k chápaniu vzťahu medzi mozgom a mysľou [xx], ktorý je založený na konekcionistickom poňatí tak mozgu, ako aj mysle. Základná predstava o mozgu (založená na experimentálnych neurovedných poznatkoch) je, že je tvorený z neurónov navzájom poprepájaných pomocou jednosmerných synaptických spojov. Ľudský mozog vykazuje neobyčajnú plasticitu (pozri obr. 5.1), v priebehu učenia neustále vznikajú (ale taktiež aj zanikajú) synaptické spoje. Architektúra mozgu je určená spojmi medzi neurónmi, ich inhibičným, alebo excitačným charakterom, a taktiež aj ich intenzitou. Možno konštatovať, *že schopnosť mozgu vykonávať nielen kognitívne aktivity, ale byť aj pamäťou, je plne zakódovaná do jeho architektúry*. Na základe týchto neurovedných poznatkov bazálneho charakteru môžeme konštatovať, že počítačová paradigma ľudského mozgu/mysle sa musí formulovať tak, že mozog je paralelný distribuovaný počítač (obsahujúci mnoho miliárd neurónov, elementárnych procesorov, ktoré sú medzi sebou poprepájané do zložitej neurónovej siete). Program v tomto paralelnom počítači je priamo zabudovaný do architektúry neurónovej siete, t. j. ľudský mozog je jednoúčelový paralelný počítač reprezentovaný neurónovou sieťou, ktorý nie je možné preprogramovať bez zmeny jeho architektúry. Z týchto všeobecných úvah vyplýva, *že myseľ s mozgom tvoria jeden integrálny celok; myseľ je v tomto prístupe možné chápať ako program vykonávaný mozgom, avšak tento program je špecifikovaný architektúrou distribuovanej neurónovej siete reprezentujúcej mozog.* Mozog a myseľ tvoria dva rôzne pohľady na ten istý objekt: (1) keď hovoríme o mozgu, myslíme tým „hardwarovú“ štruktúru, biologicky realizovanú neurónmi a ich synaptickými spojmi (formálne reprezentovanú neurónovou sieťou), v opačnom prípade,

[3] Kognitívna veda je charakterizovaná v kapitole 11.

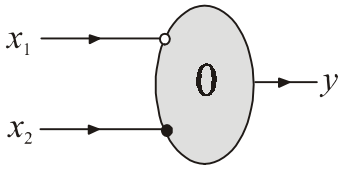
(2) keď hovoríme o mysli, myslíme tým kognitívne a iné aktivity mozgu, realizované výpočtami neurónovej siete reprezentujúcej mozog.

Zhrnutie

Metafora ľudského mozgu má svoje formálne naplnenie v informatike v teórii neurónových sietí. Neurónové siete sú tvorené z jednoduchých prahových procesných jednotiek nazývaných neuróny, ktoré sú poprepájané prostredníctvom jednosmerných spojov do paralelnej architektúry. Ak sú neuróny realizované pomocou jednoduchých logických neurónov, potom neurónové siete nadobúdajú vlastnosť univerzálnosti nad Boolovými funkciami, t.j. ľubovolná Boolova funkcia je implementovateľná pomocou trojvrstvovej neurónovej sieti. Učenie neurónovej siete spočíva v systematickom nastavovaní váhových koeficientov tak, aby neurónová sieť vykazovala požadovanú výstupnú odozvu na vstupné obrazce - informáciu z tréningovej množiny. Učenie v neurónových sieťach je možné naformulovať ako minimalizačný problém účelovej funkcie chýb, ktorá reprezentuje rozdiely medzi vypočítanými a požadovanými výstupnými aktivitami. Konekcionistický prístup k mozgu a mysli rieši ich vzájomný vzťah tak, že mozog a myseľ tvoria dva rôzne pohľady na ten istý objekt. Keď hovoríme o mozgu, myslíme tým „hardwarovú“ štruktúru, biologicky realizovanú neurónmi a ich synaptickými spojmi (formálne reprezentovanú neurónovou sieťou); v opačnom prípade, keď hovoríme o mysli, myslíme tým kognitívne a iné aktivity mozgu, realizované výpočtami neurónovej siete reprezentujúcej mozog.

Úlohy

Úloha 5.1. Zistite akú výrokovú spojku (Boolovu funkciu) reprezentuje logický neurón



Úloha 5.2. Vyjadrite pomocou logického neurónu z úlohy 5.1 a neurónu konjunkcie výrokovú spojku ekvivalencie.

Úloha 5.3. Tabuľka

x_1	x_2	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

obsahuje všetky množné (16) Boolove funkcie dvoch premenných. (1) Identifikujete medzi týmito funkciami štandardné výrokové spojky a (2) zistite, ktoré funkcie sú lineírne separovateľné.

Úloha 5.4. Zostrojte neurónovú sieť, ktorá simuluje Boolovu funkciu zadanú tabuľkou

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Úloha 5.5. Zostrojte tabuľku Boolovej funkcie $(\beta_1,\beta_2) = f(\alpha_1,\alpha_2,\alpha_3)$, ktorá určuje súčet troch binárnych číslic

$$\begin{array}{r} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \hline \beta_1\,\beta_2 \end{array}$$

Zostrojte neurónovú sieť, ktorá simuluje túto Boolovu funkciu. (pozri obr. 5.9).

Otázky na opakovanie

Otázka 5.1. Aké najdôležitejšie vlastnosti má neurón?

Otázka 5.2. Ako je definovaný logický neurón?

Otázka 5.3. Aký je rozdiel medzi excitačným a inhibičným spojom v neurónovej sieti?

Otázka 5.4. Ako sú určené logické neuróny, ktoré simulujú logické spojky konjunkcie, disjunkcie a negácie?

Otázka 5.5. Ako je definovaná lineárne separovateľná Boolova funkcia?

Otázka 5.6. Ukážte na príklade spojky vylučujúcej disjunkcie prípad Boolovej funkcie, ktorá je lineárne neseparovateľná.

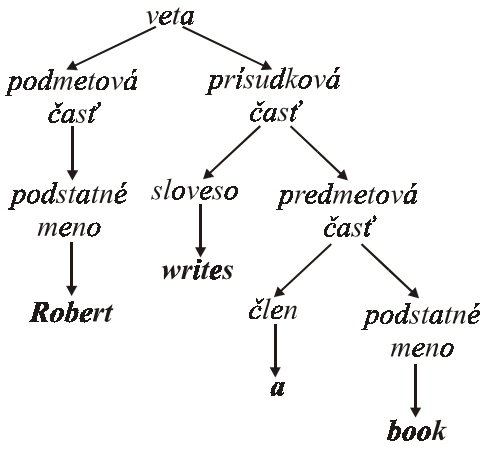
Otázka 5.7. Aký je disjunktívny tvar logickej spojky vylučujúcej disjunkcie? Aký má tvar neurónová sieť reprezentujúca spojku vylučujúcej disjunkcie?

Otázka 5.8. Ako je charakterizovaná univerzálnosť neurónových sietí?

6 Gramatiky a jazyk

Motivujúca idea teórie formálnych gramatík spočíva v tom, že poskytujú efektívny prístup k pochopeniu nielen ľudského jazyka, ale aj prekladačov programovacích jazykov (Basic, Pascal, ...). Jazyk môže byť chápaný ako súbor štruktúrovaných symbolických výrazov, pričom generatívne gramatiky reprezentujú jednoduchý spôsob konštrukcie týchto výrazov.

Štruktúra vety prirodzeného jazyka je definovaná gramatickými pravidlami. Jednoduchá slovenská veta obsahuje podmetovú časť a prísudkovú časť. To znamená, že veta je definovaná pomocou iných dvoch jednotiek, ktorých význam zatiaľ nepoznáme, a preto ich musíme definovať. Podmetovú časť možno definovať ako podstatné meno a prísudkovú časť ako sloveso, za ktorým nasleduje predmetová časť. Základnou vlastnosťou tohto prístupu je, že niektoré jednotky jazyka definujeme pomocou iných jednotiek. Aby bol jazyk správne definovaný, musí byť tento proces taký, aby sme v konečnom dôsledku definovali každú jednotku pomocou veličín, ktoré sú známe – elementárne, také čo sa nakoniec objavia v skutočnej vete. Celý proces možno pre vetu „Robert writes a book“ graficky znázorniť pomocou obr. 6.1.



Obrázok 6.1. Gramatická štruktúra vety „*Robert writes a book*“

Ako vidieť z obr. 6.1, elementárne jednotky sú na najnižšej úrovni. Na odlíšenie elementárnych jednotiek (nazývaných terminálové symboly) od jednotiek definovaných pomocou iných jednotiek (nazývaných neterminálové symboly) používame zvyčajne odlišné typy písma, ohraničujúce symboly a pod. Gramatické pravidlá, pomocou ktorých definujeme jednotlivé jednotky – symboly, majú tento tvar:

$$\langle \textit{definovaný symbol} \rangle \rightarrow \textit{symboly, pomocou ktorých definujeme symbol na ľavej strane}$$

Vetu prirodzeného jazyka možno teraz definovať nasledujúcimi gramatickými pravidlami:

R_1 : ⟨veta⟩ → ⟨podmetová časť⟩ ⟨prísudková časť⟩
R_2 : ⟨podmetová časť⟩ → ⟨podstatné meno⟩
R_3 : ⟨podstatné meno⟩ → ***Robert***

R_4 : ⟨prísudková časť⟩ → ⟨sloveso⟩ ⟨predmetová časť⟩
R_5 : ⟨sloveso ⟩ → ***writes***
R_6 : ⟨predmetová časť⟩ → ⟨člen⟩ ⟨podstatné meno⟩
R_7 : ⟨člen⟩ → ***a***
R_8 : ⟨podstatné meno⟩ → ***book***

Na prvý pohľad by sa mohlo zdať, že prirodzený jazyk možno veľmi jednoducho formalizovať. Žiaľ, opak je však pravdou. Stačí si zobrať napr. vetu „*Robert wrote a book*“, kde prítomný čas slovesa „*writes*“ bol nahradený minulým časom slovesa „*wrote*“. To znamená, že pri slovese z prísudkovej časti už treba uvažovať o čase slovesa (prítomný a minulý), naviac, vytváranie minulého času je komplikované existenciou nepravidelných slovies, a pod.

Množina všetkých gramatických pravidiel jazyka tvorí gramatiku jazyka. Gramatika jazyka nám umožňuje generovať vety jazyka, ktoré majú základnú vlastnosť: sú gramaticky správne.

Gramatickými pravidlami – gramatikou – určujeme syntax jazyka, to znamená, že určujeme prípustnú štruktúru viet jazyka a elementárne jednotky, ktoré možno na danom mieste použiť. Ak ľubovolná veta spĺňa podmienky definované gramatikou, patrí do daného jazyka. Význam viet je určený sémantikou. Syntax a sémantika navzájom úzko súvisia. Syntax určuje štruktúru vety, ktorí je základom pre určenie sémantiky (významu) vety.

Syntaktická správnosť vety nezaručuje jej sémantickú správnosť. Možno to ilustrovať na dvoch vetách „*Robert writes a book*“ a „*Book writes a Robert*“. Zatiaľ čo prvá veta je tak syntakticky, ako aj sémanticky správna, druhá veta je len syntakticky správna, ale je sémanticky nezmyselná.

6.1 Definícia generatívnej gramatiky

Všeobecné úvahy z úvodnej časti tejto kapitoly teraz vyústia v definícii generatívnej gramatiky[1] pomocou usporiadanej štvorice

$$G = (V_N, V_T, R, S)$$

kde jednotlivé prvky štvorice sú špecifikované takto:

(1) V_N je množina obsahujúca neterminálové symboly,
(2) V_T je množina obsahujúca terminálové symboly, pričom predpokladáme, že tieto dve množiny nemajú spoločné symboly, $V_N \cap V_T = \varnothing$,
(3) R je množina prepisovacích pravidiel $\alpha \to \beta$, kde $\alpha \in V_N$ je neterminálový symbol a β je reťazec obsahujúci tak terminálové, ako aj neterminálové symboly,
(4) $S \in V_N$ je začiatočný symbol.

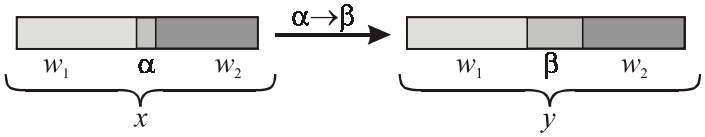
Základným cieľom gramatiky je pomocou prepisovacích pravidiel systematické generovanie reťazcov – formúl, ktoré obsahujú len terminálové symboly. Proces generovania formúl je inicializovaný „protoformulou“ obsahujúcou len začiatočný symbol *S*, použitím prepisovacích pravidiel je táto „protoformula“ pretransformovaná do tvaru obsahujúceho len

[1] Pre jednoduchosť budeme definovať len tzv. bezkontextovú gramatiku (podľa Chomského klasifikácie [xx]), ostatné typy generatívnych gramatík (napr. kontextové) nebudú v tejto kapitole diskutované, pretože sú podstatne menej dôležité.

terminálové a/alebo neterminálové symboly. Proces je ukončený vtedy, keď formula obsahuje len terminálové symboly, k takýmto formulám sú totiž prepisovacie pravidlá už neaplikovateľné.

Obráťme našu pozornosť na špecifikáciu prepisovacích pravidiel, ukážeme na ich principiálnu úlohu v generatívnych gramatikách. Nech *x* je reťazec symbolov (terminálových, alebo neterminálových); formálne túto skutočnosť vyjadríme formulou $x \in (V_N + V_T)^*$, kde „výraz" $(A)^*$ vyjadruje množinu všetkých možných reťazcov zostrojených zo symbolov obsiahnutých v množine *A*. Predpokladajme, že reťazec *x* má tvar $x = w_1 \alpha w_2$, potom aplikácia prepisovacieho pravidla $r = (\alpha \to \beta) \in R$ k reťazcu *x* vygeneruje nový reťazec $y = w_1 \beta w_2$, formálne $y = r(x)$. Tak napríklad, uvažujme reťazec $x = 00A11A00$ a pravidlo $r = (A \to 01)$, aplikácia tohto pravidla na reťazec *x* poskytuje dva rôzne reťazce $y_1 = 00\underline{01}11A00$ a $y_2 = 00A11\underline{01}00$, kde podtrhnuté binárne podreťazce vznikli aplikáciou pravidla *r* substitúciou neterminálového symbolu *A* reťazcom 01 (pozri obr. 6.2). Potom hovoríme, že formula *y* je *priamo odvoditeľná* z formule *x*, čo zapisujeme takto: $x \to y$. Vo všeobecnosti, formula *y* je (len) *odvoditeľná* z formule *x* (formálne, $x \xrightarrow{*} y$), ak existuje taká postupnosť formúl $x_0, x_1, x_2, ..., x_n$, kde $x_0 = x$ a $x_n = y$, že susedné formule x_{i-1}, x_i $(pre\ i = 1,2,...,n)$ sú priamo odvoditeľné

$$\left(x \xrightarrow{*} y\right) \Leftrightarrow_{def} \left(x = x_0 \to x_1 \to ... \to x_{n-1} \to x_n = y\right)$$



Obrázok 6.2. Prepísanie formuly $x = w_1 \alpha w_2$ na formulu $y = w_1 \beta w_2$ pomocou pravidla $\alpha \to \beta$.

Všetky možné formule A_T^*, ktoré sú vygenerované gramatikou a ktoré obsahujú len terminálové symboly, tvoria *jazyk L* na gramatike *G* (tiež hovoríme, že jazyk *L* je *indukovaný* gramatikou *G*), označený $L(G)$

$$L(G) = \left\{x \in A_T^* \,;\, S \xrightarrow{*} x\right\}$$

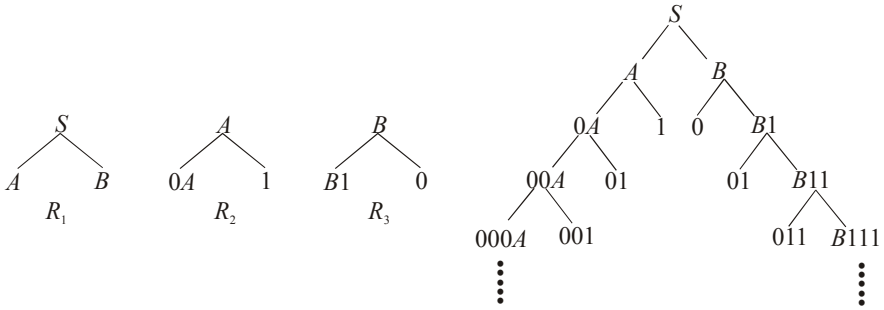
To znamená, že gramatika pomocou prepisovacích pravidiel generuje reťazce, ktoré vo všeobecnosti obsahujú tak terminálové ako aj neterminálové symboly. Tie formule, generované gramatikou, ktoré obsahujú len terminálové symboly, tvoria jazyk na danej gramatike (hovoríme, že jazyk *L* je *indukovaný* gramatikou *G*). Tvorba formúl jazyka je pomerne priamočiary proces, postupným aplikovaním prepisovacích pravidiel musíme vždy dospieť k formule, ktorá obsahuje len terminálové symboly a teda patria do jazyka nad danou gramatikou. Opačný problém, rekognoskácia toho či nejaká formula patrí do daného jazyka, nie je triviálny, bude mu venovaná nasledujúca podkapitola.

Príklad 6.1. Nech množiny neterminálových a terminálových symbolov sú $V_N = \{A, B, S\}$ a $V_T = \{0,1\}$ (pripomeňme, že tieto množiny sú disjunktné, spĺňajú požadovanú podmienku $V_N \cap V_T = \varnothing$). Množina *R* obsahuje tieto prepisovacie pravidlá

$$\left.\begin{array}{lll} R_{1,1} & : & S \to A \\ R_{1,2} & : & S \to B \end{array}\right\} \Rightarrow R_1 \; : \; S \to A \,|\, B \qquad \left.\begin{array}{lll} R_{2,1} & : & A \to 0A \\ R_{2,2} & : & A \to 1 \end{array}\right\} \Rightarrow R_2 \; : \; A \to 0A \,|\, 1$$

$$\left.\begin{array}{lll} R_{3,1} & : & B \to B1 \\ R_{3,2} & : & B \to 0 \end{array}\right\} \Rightarrow R_3 \; : \; B \to B1 \,|\, 0$$

kde bola použitá konvencia, že ak existuje viac pravidiel s rovnakou ľavou stranou, tak tieto sa integrujú do jedného pravidla, ktoré má alternatívne pravé strany oddelené vertikálnou čiarou. Spôsob generovania reťazcov – formúl je znázornený na pravej časti obr. 6.3, ktorý je založený na reprezentácii prepisovacích pravidiel z ľavej časti obr. 6.3. Proces je možné znázorniť pomocou grafu – *stromu riešení*[2], ktorého koreň (najvyšší vrchol, ktorý má len svojich nasledovníkov) je označený začiatočným symbolom *S*. Proces generovania formúl je zahájený v prvom kroku použitím pravidla $R_1 \; : \; S \to A \,|\, B$, ktorým je začiatočný vrchol substituovaný neterminálovým symbolom *A* resp. *B*. V druhom kroku sa použije pravidlo $R_2 \; : \; A \to 0A \,|\, 1$ alebo $R_3 \; : \; B \to B1 \,|\, 0$, pomocou ktorých vetvu stromu môžeme neustále predlžovať alebo taktiež aj ukončiť. Tento proces aplikácie týchto dvoch pravidiel sa neustále opakuje, čím dostávame stále dlhšie binárne formule obsahujúce len terminálové symboly. Jednotlivé vrcholy v tomto strome riešení sú reprezentované aktuálnymi formulami, ktoré sú odvoditeľné zo symbolu *S* pomocou prepisovacích pravidiel. Poznamenajme, že takto definovaný strom riešení je nekonečný v dôsledku rekurzie spôsobenej výskytom neterminálových symbolov na pravej strane prepisovacích pravidiel; tento formálny nedostatok môže byť odstránený tak, že postulujeme maximálnu dĺžku reťazcov.



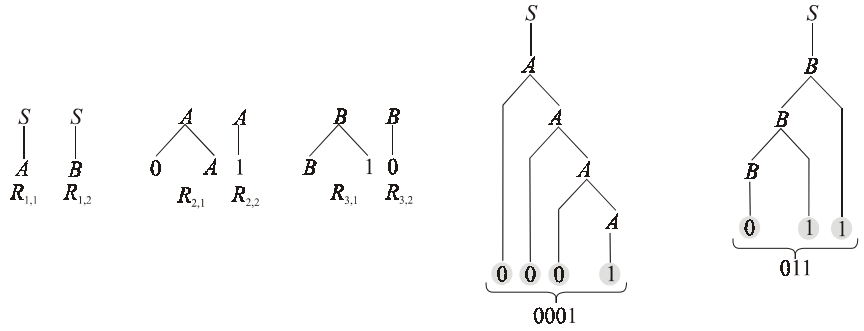
Obrázok 6.3. Ľavá časť obrázku znázorňuje diagramatickú reprezentáciu jednotlivých prepisovacích pravidiel z príkladu 6.1. Vrcholom každého grafu je príslušný neterminálový symbol z ľavej strany pravidla, v dolných častiach grafu sú uvedené pravé strany prepisovacích pravidiel. Pravá časť obrázku je *strom riešení*, jeho vrcholy sú reprezentované formulami, ktoré sú odvoditeľné zo začiatočného symbolu *S*. Koncové vrcholy sú ohodnotené formulami obsahujúcimi len terminálové symboly, ich všeobecný tvar je 0^n1 a 01^n (pre $n \geq 0$).

Z tejto jednoduchej diagramatickej reprezentácie postupného používania prepisovacích pravidiel vyplýva, že sa vytvárajú formule obsahujúce terminálové symboly len tvaru 0^n1 a 01^n (pre $n \geq 0$)[3]. To znamená, že jazyk indukovaný gramatikou obsahuje len tieto reťazce, $L(G) = \{0^n1, 01^n \,;\, n \geq 1\}$.

[2] Podobný názov sme použili aj v kapitole 3, kde stavy hry sú usporiadané do stromovej štruktúry, pričom koreň stromu je stav priradený začiatočnému stavu hry. Podobne aj teórii formálnych jazykov môžeme chápať formuly odvoditeľné v rámci danej gramatiky ako stavy „jazyka“, pričom jednotlivé formuly - stavy sú organizované do stromovej štruktúry. Koreňom tohto stromu je začiatočný symbol *S*, koncové vrcholy odpovedajú formulám generovaným gramatikou.

[3] V tomto prípade *n* neznamená mocninu, ale počet opakovaní symbolu 0 či 1.

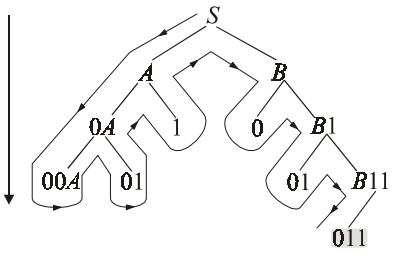
Existuje aj druhý, podrobnejší prístup k diagramatickej reprezentácii generovania formúl pomocou prepisovacích pravidiel. V tomto prípade sú jednotlivé prepisovacie pravidlá diagramaticky reprezentované spôsobom znázorneným na ľavej časti obr. 6.4. Postupným aplikovaním týchto jednoduchých diagramatických prepisovacích pravidiel dostaneme tzv. *strom odvodenia* (derivačný graf, alebo graf derivácie) danej formuly (pozri pravú časť obr. 6.4). Proces je opakovaný tak dlho, dokiaľ už žiadne pravidlo nemôže byť aplikované, t.j. koncové vrcholy stromu obsahujú len terminálové symboly. Pomocou týchto stromov *odvodenia* je každá „lineárna" formula reprezentovaná stromovým grafom, ktorý sa chápe ako reprezentácia syntaxu formuly. Takýto stromový graf je taktiež nápomocný pri zostrojení sémantického významu formuly.



Obrázok 6.4. Ľavé tri grafy reprezentujú podrobnú diagramatickú reprezentáciu jednotlivých prepisovacích pravidiel z príkladu 6.1. Vrcholom každého grafu je príslušný neterminálový symbol z ľavej strany pravidla, v dolných častiach grafu sú uvedené jednotlivé symboly (idúc zľava doprava) z pravej strany pravidiel. Pravé dva grafy znázorňujú stromy odvodenia formúl 0001 a 011. Generovaný reťazec znakov sa zostrojí tak, že ideme zľava doprava a zaznamenávame jednotlivé koncové symboly. Koncové symboly sú znázornené tmavými kolečkami.

6.2 Syntaktická analýza formúl

Syntaktická analýza rieši problém, či daná formula x patrí do jazyka $L(G)$ indukovaného gramatikou G. V predchádzajúcej podkapitole sme študovali inverzný problém, pre danú gramatiku G sme postupným aplikovaním vhodnej sekvencie prepisovacích pravidiel zostrojili formulu x, ktorá automaticky patrila do jazyka $L(G)$.

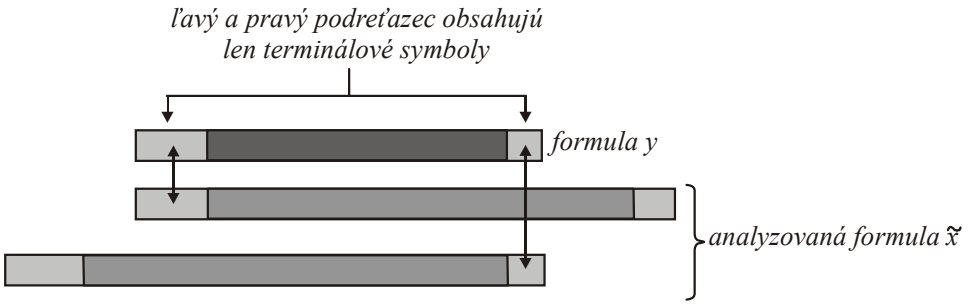


Obrázok 6.5. Schematické znázornenie metódy spätného prehľadávania „zhora-nadol" pri konštrukcii redukovaného stromu riešení, ktorý odpovedá cieľovej formule 011. Konštrukcia je inicializovaná vrcholom S, použitím pravidiel odvodzovania predlžujeme jeho konštrukciu. Daná vetva stromu odvodenia je ukončená ako nenádejná ak vyprodukuje formulu, ktorí je buď dlhšia ako analyzovaná formula, alebo jej neterminálové symboly nie sú kompatibilné s analyzovanou formulou. Tento druhý prípad predčasného ukončenia konštrukcie stromu odvodenia sa vyskytuje v ľavej vetvi stromu odpovedajúcej formule 00A, ktorej prvé dva ľavé terminálové symboly 00 nie sú kompatibilné s analyzovanou formulou 011. Odvodenie tejto formule môžeme znázorniť sekvenciou formúl: $S \to B \to B1 \to B11 \to 011$.

Nech x je formula, ktorá obsahuje len terminálové symboly. Naším cieľom je zistiť, či $x \in L(G)$. Tento proces rozpoznávania, či daná formula patrí do jazyka alebo nie, sa nazýva *syntaktická analýza formule*. Vo všeobecnosti, syntaktická analýza formule môže byť implementovaná metódou spätného prehľadávania, ktorej základné princípy boli už študované v kapitole 3. Metóda spätného prehľadávania, v tomto špecifickom prípade produkujúca len tzv. *redukovaný strom riešení*, spočíva v systematickom pohybe po strome odvodení, ktorého vrcholy priamo odpovedajú jednotlivým formulám, ktoré sú odvoditeľné zo začiatočného symbolu S (metóda zhora-nadol) , pozri obr. 6.5. Pri postupnom vytváraní stromu odvodenia opakovane aplikujeme prepisovacie pravidla r k formule x, ktorej výsledkom je formula y, $y=r(x)$. Táto metóda je ohraničená nasledujúcimi dvoma podmienkami:

(1) ak výsledná formula y obsahuje na svojich okrajoch terminálové symboly, potom tieto terminálové symboly musia byť rovnaké s terminálovými symbolmi analyzovanej formule $\tilde{x}$. V opačnom prípade, ak tieto terminálové symboly sú rozličné, potom analyzovaná formula $\tilde{x}$ nemôže byť odvodená z formule y, pozri obr. 6.6,

(2) dĺžka formuly y nemôže byť väčšia ako dĺžka analyzovanej formuly $\tilde{x}$.

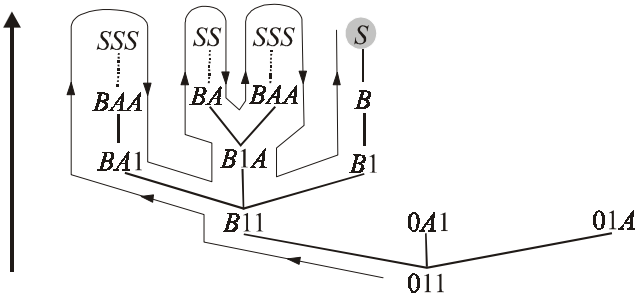


Obrázok 6.6. Ak aktuálna formula y na svojom ľavom/pravom okraji obsahuje terminálové symboly, potom tieto musia byť obsiahnuté aj v analyzovanej formule $\tilde{x}$. V opačnom prípade, ak táto jednoduchá podmienka nie je splnená, potom aplikácia prepisovacieho pravidla $y = r(x)$ nemôže byť aplikovaná k tvorbe stromu odvodenia ako nenádejná.

Obe tieto podmienky podstatne obmedzujú veľkosť množiny kandidátov prepisovacích pravidiel. Pri predlžovaní stromu odvodenia musia byť odmietnuté mnohé z prepisovacích pravidiel potenciálne aplikovateľných k formule x v dôsledku porušenia niektorej z vyššie uvedených dvoch podmienok.

V prípade, že existuje redukovaný strom riešení pre danú formulu x, potom z cesty, ktorá spája začiatočný symbol S s analyzovanou formulou x, dostaneme sekvenciu prepisovacích pravidiel, pomocou ktorých zo symbolu S vytvoríme formulu x. Z tejto sekvencie prepisovacích pravidiel ľahko vytvoríme strom odvodenia formuly x.

Metóda spätného prehľadávania pre konštrukciu stromu odvodenia môže byť taktiež realizovaná opačným spôsobom „zdola-nahor“. Koreň stromu je teraz samotná analyzovaná formula, aplikovaním jednotlivých pravidiel sa ju snažíme prepísať do kratšieho tvaru pomocou neterminálových systémov. Metóda je úspešná vtedy, keď koncový vrchol stromu odpovedá začiatočnému symbolu S, pozri obr. 6.7.



Obrázok 6.7. Metóda spätného prehľadávania pre konštrukciu redukovaného stromu riešení spôsobom „zdola-nahor“. V tomto prípade je koreň stromu identifikovaný priamo s analyzovanou formulou, použitím prepisovacích pravidiel sa transformuje aktuálna formula na tvar s klesajúcim počtom terminálových symbolov. V prípade, že koncový vrchol je ohodnotený začiatočným symbolom *S*, potom analyzovaná formula je v rámci danej gramatiky odvoditeľná z S, t.j. patrí do jazyka indukovaného danou gramatikou.

Zhrnutie

Formálne gramatiky tvoria významnú časť teoretickej informatiky, kde tvoria teoretický základ pre štúdium a implementáciu tak umelých, ako aj prirodzených jazykových systémov. V tejto kapitole sme sa obmedzili len na štúdium tzv. bezkontextovej gramatiky *G*, pre ktorú ľavé strany prepisovacích pravidiel obsahujú jeden neterminálový symbol. Jazyk gramatiky *L*(*G*) je zložený zo všetkých možných formúl – reťazcov, ktoré obsahujú len terminálové symboly. Dva rôzne spôsoby grafickej vizualizácie sú použité pri interpretácii postupov a výsledkov v teórii formálnych gramatík.

Prvý prístup ja založený na grafických objektoch nazývaných *stromy riešení*, kde dva vrcholy stromu, ktoré sú priradené dvom rôznym formulám sú spojené hranou vtedy, ako jedna z formúl je priamo odvoditeľná z druhej pomocou prepisovacieho pravidla gramatiky *G*. Tento prístup je v tesnej analógii s podobnými objektmi z umelej inteligencie, kde grafické štruktúry „stromy riešení“ reprezentujú postup pri riešení problému spočívajúceho v transformácii východzieho stavu na požadovaný konečný stav. Strom riešení sa zostrojí pomocou metódy spätného prehľadávania dvoma rôznymi prístupmi, a to buď prístupom „zhora nadol“ alebo „zdola nahor“. Použitá metóda spätného prehľadávania môže byť podstatne urýchlená rôznymi pravidlami založenými na tvaru finálnej formuly, ktorej odvodenia sa hľadá.

Druhý prístup ku grafickej vizualizácii postupov v teórii formálnych gramatík ja založený na *stromoch odvodenia*, ktorých nekoncové vrcholy sú ohodnotené len jednotlivými neterminálovými symbolmi (nie formulami ako v stromoch riešení). Ak je nejaká formula odvoditeľná zo začiatočného symbolu *S*, potom existuje graf – strom odvodenia, ktorého koncové vrcholy tvoria usporiadanú množinu (idúc zľava doprava) terminálových vrcholov z analyzovanej formuly. Pomocou grafu odvodenia je každá formula (lineárna postupnosť symbolov) patriaca do jazyku *L*(*G*) interpretovaná grafom, ktorý špecifikuje *syntax* danej formuly. Strom odvodenia je dôležitou formálnou štruktúrou nielen pre popis syntaxu danej formuly, ale pomocou neho sa zostrojuje aj jej sémantický význam, a pomocou neho sa dá plne pochopiť aj informačný obsah formule.

Úlohy

Úloha 6.1. Pre gramatiku z príkladu 6.1 zostrojte redukované stromy riešení tak v prístupe zhora-nadol, ako aj v prístupe zdola-nahor pre formulu 0001.

Úloha 6.2. Pre gramatiku

$$G = \left(V_N = \{S, A, B\}, V_T = \{0,1\}, R = \{S \to A \mid B, A \to B0 \mid 1, B \to 1A \mid 0\}, S\right)$$

zostrojte strom riešení a pomocou neho zostrojte jazyk tejto gramatiky.

Úloha 6.3. Pre gramatiku z príkladu 6.2 zostrojte redukované stromy riešení formúl 1100 a 1110, tak pomocou prístupu „zdola-nahor“, ako aj „zhora-nadol“ .

Úloha 6.4. Pre gramatiku z príkladu 6.2 zostrojte stromy odvodení formúl 1100 a 1110.

Otázky na opakovanie

Otázka 6.1. Ako je definovaná gramatika? Čo znamená, že gramatika je bezkontextová?

Otázka 6.2. Ako je definovaná aplikácia prepisovacieho pravidla $\alpha \to \beta$ na formulu *x*?

Otázka 6.3. Čo je to jazyk *L* na gramatike *G*?

Otázka 6.4. Čo je strom riešení? Ako zostrojíme zo stromu riešení jazyk gramatiky?

Otázka 6.5. Čo je strom odvodenia? Reprezentuje formulu jazyka jednoznačne?

Otázka 6.6. Čo je syntaktická analýza formuly? Čo je redukovaný strom riešení pre danú formulu?

Otázka 6.7. Ako sa odlišujú prístupy „zhora-nadol“ a „zdola-nahor“ ku konštrukcii redukovaných stromov riešení?

7 Univerzálny darwinizmus a genetické algoritmy

V súčasnosti je paradigma darwinovskej evolúcie široko aplikovaná v rôznych oblastiach ľudského poznania (poznáme evolučnú ekológiu, evolučnú ekonomiku, evolučnú psychológiu, evolučnú lingvistiku, evolučnú epistemológiu, evolučnú výpočtovú vedu, evolučnú fyziku,...). Spoločným leitmotívom všetkých týchto nových vedných oblastí je použitie princípov darwinovskej evolučnej teórie, ktorá je v súčasnosti [xx] charakterizovaná ako *univerzálny algoritmus* s platnosťou nielen v biológii, ale aj vo všetkých oblastiach ľudského poznania, kde sme schopní vyabstrahovať informačné entity - replikátory, ktoré majú schopnosť reprodukovať sa s variáciami a medzi ktorými prebieha prírodný výber. Prvý, kto na túto univerzálnosť darwinovskej evolúcie upozornil, bol americký populačný genetik Sewall Wright. Ten v 30. rokoch minulého storočia charakterizoval *evolúciu ako optimalizáciu na povrchu fitnes funkcie* (fitness landscape), kde sa hľadá genotyp odpovedajúci globálnemu maximu. Z tohto pohľadu môžeme darwinovskú evolúciu charakterizovať ako univerzálnu paradigmu schopnú riešiť komplikované optimalizačné problémy vyskytujúce sa v širokom spektre vedných oblastí, od prírodných vied až po spoločenské vedy. V informatike našla táto zaujímavá idea svoj odraz už pred viac ako 30 rokmi, keď John Holland vynašiel genetické algoritmy, ktoré je možno chápať ako algoritmizáciu darwinovskej evolúcie a ktoré sa stali v súčasnosti búrlivo sa rozvíjajúcou oblasťou informatiky. Ich použitie umožňuje riešiť mnohé problémy, ktoré sa donedávna zdali len ťažko (ak vôbec) riešiteľnými. V umelej inteligencii použitie genetických algoritmov otvára úplne nové možnosti pre tvorbu stratégií a modelov (ktoré reprezentujú obvykle veľmi zložité problémy pre človeka), ich tvorba je prenechaná počítaču pomocou prístupu darwinovskej evolúcie „in silico“.

7.1 Darwinovské systémy a univerzálny darwinizmus

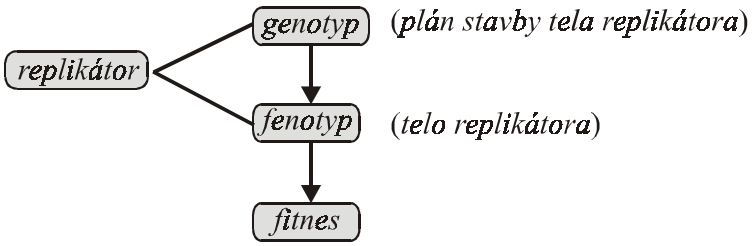
Najvšeobecnejšia formulácia základnej idey univerzálneho darwinizmu je uskutočnená pomocou koncepcie ***darwinovského systému*** založeného na nasledujúcich dvoch postulátoch:

(1) *Darwinovský systém sa skladá z populácie replikátorov* – jedincov/objektov, ktoré za určitých vhodných podmienok sú schopné *replikácie* - rozmnožovania. Replikačný proces spočíva v „kopírovaní“ jedincov do populácie, pričom toto „kopírovanie“ sa uskutočňuje s *určitými malými chybami*[1]. Z tohto predpokladu nepresnej replikácie (s malými chybami) vyplýva, že *populácia nie je homogénna*, obsahuje jedincov – replikátory, ktoré vykazujú určitú malú variabilitu.

[1] Vo všeobecnosti, presnosť kopírovania závisí tak od spôsobu kopírovania, ako aj od spôsobu kódovania kopírovanej informácie. V súčasnosti je najpresnejšie digitálne kopírovanie informácie, ktorá je taktiež digitálne kódovaná (napr. kopírovanie CD nosiča pomocou „napalovačky“ na počítači). Podstatne menej presné je analógové kopírovanie informácie kódovanej taktiež analógovo (napr. kopírovanie xerox kópie pomocou xeroxu, alebo magnetofónovej nahrávky pomocou magnetofónu).

(2) Každý replikátor populácie je ohodnotený *fitnes*, hodnotenie vyjadruje schopnosť replikátora prežiť a úspešne vstupovať do replikačného procesu. *Prírodný výber* v darwinovskom systéme spočíva v tom, že jedinci populácie nie sú vyberané do replikačného procesu náhodne, ale s *pravdepodobnosťou úmernou ich fitnes* (hovoríme, že výber sa deje kvázináhodne).

V definícii darwinovského systému sa vyskytuje niekoľko pojmov, ktoré musia byť bližšie objasnené. Pojem „replikátor" je potrebné chápať ako relatívne samostatnú entitu, ktorá existuje v nejakom prostredí a vytvára spolu s ostatnými replikátormi populáciu. Najjednoduchší prístup k implementácii replikátora je jeho stotožnenie s nejakou informáciou, ktorá kóduje jeho „telo". Z tohto dôvodu môžeme využívať biologickú terminológiu a rozlišovať dve rôzne špecifikácie replikátora, jeho *genotyp* a *fenotyp*. Genotyp je informácia o architektúre replikátora, zatiaľ čo fenotyp je „telo" replikátora[2]. Vzťah medzi genotypom a fenotypom nebudeme bližšie špecifikovať, obvykle je silne závislý na typu darwinovského systému. Čo bude pre nás dôležité, je skutočnosť, že fitnes replikátora je určený schopnosťou fenotypu prežívať v danom prostredí a vstupovať do výhodných interakcií s inými replikátormi z populácie, pozri obr. 7.1.



Obrázok 7.1. Replikátor je charakterizovaný tak genotypom, ako aj fenotypom. Fitnes replikátora je určený schopnosťou fenotypu prežívať v danom prostredí a vstupovať do výhodných interakcií s inými replikátormi z populácie.

Proces replikácie jedinca – replikátora je formálne chápaný ako kopírovanie jeho genotypu a vytvorenie nového fenotypu určeného kopírovaným genotypom. To znamená, že fenotyp – organizmus replikátora – môžeme chápať ako nosič (vehikel) genotypu, ktorý umožňuje jeho replikáciu. Pre zjednodušenie našich úvah chápeme proces replikácie len ako kopírovanie genotypu, pričom tento proces kopírovania je „fyzicky" uskutočnený fenotypom replikátora. Je potrebné poznamenať, že sa jedná o veľmi silnú idealizáciu, ale umožní nám zaviesť pomerne jednoduchú algoritmizáciu univerzálneho darwinizmu.

Musíme však podotknúť, že môžu existovať darwinovské systémy, kde odlíšenie fenotypu od genotypu neplatí, kde sa genotyp kopíruje – replikuje iným zariadením, ako vlastným fenotypom. Dobrým príkladom tejto situácie sú biologické a počítačové vírusy, ktoré k vlastnej replikácii využívajú systémy v ktorých parazitujú.

Postulujme, že replikátor je reprezentovaný svojím *genotypom* $\boldsymbol{x}$, ktorý, ako už bolo poznamenané, obsahuje úplnú informáciu o architektúre a vlastnostiach replikátora. *Populácia* replikátorov je multimnožina[3] genotypov

$$P = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_p\}$$

[2] Podobne, ako vo fyzike mikrosveta, kde častice majú duálny častico-vlnový charakter, môžeme aj fenotyp a genotyp chápať ako duálne charakteristiky replikátora. V určitých situáciách pod replikátorom rozumieme jeho genotyp, zatiaľ čo v iných situáciách pod replikátorom myslíme jeho fenotyp.

[3] Multimnožina je zovšeobecnenie pojmu množiny, v ktorej sa prvky môžu vyskytovať viackrát.

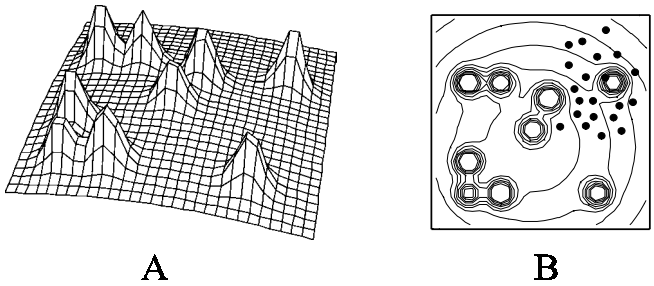
Vzájomný vzťah medzi triádou „genotyp – fenotyp – fitnes“ je reprezentovaný postupnosťou dvoch zobrazení (pozri obr. 7.2)

$$G \xrightarrow{\text{fenotyp}} F \xrightarrow{\text{fitnes}} [0,\infty)$$

To znamená, že prvotnou veličinou je genotyp, ktorý je zobrazený na fenotyp, potom fenotyp je zobrazený na fitnes. Prvé zobrazenie priradí každému genotypu telo – organizmus, nazývaný fenotyp. V biológii toto zobrazenie predpokladá existenciu procesu nazývaného embryogenéza, t. j. tvorba organizmu zo zárodku – genotypu. Skrátená forma dvojstupňového zobrazenia má tvar (pozri obr. 7.2)

$$G \xrightarrow{f} [0,\infty)$$

kde nové zobrazenie *f* vznikne zložením dvoch zobrazení, zobrazenia genotypu na fenotyp a zobrazenia fenotypu na fitnes. V tejto súvislosti si môžeme položiť otázku, prečo je užitočné explicitne uvažovať fenotyp ako sprostredkovateľa medzi genotypom a fitnes. Táto otázka je plne legitímna z pohľadu matematika, avšak je potrebné poznamenať, že koncepcia fenotypu reprezentuje veľmi efektívnu a plodnú heuristiku[4] pre interpretáciu univerzálneho darwinizmu. Cieľom evolúcie nie je obvykle požadovaný genotyp, ale fenotyp vykazujúci určité vlastnosti. Z tohto pohľadu môžeme povedať, že darwinovská evolúcia je reprezentovaná postupnosťou fenotypov, ktorých vlastnosti sú bližšie a bližšie cieľovým vlastnostiam fenotypu.



Obrázok 7.2. Znázornenie povrchu fitnes (A), ktorý vyjadruje závislosť fitnes replikátorov na zložení ich genotypu. Kontúrový graf (B) je priradený povrchu fitnes, oblak bodov znázorňuje populáciu replikátorov. Ako už bolo povedané v úvode tejto kapitoly, podľa Sewalla Wrighta je evolúcia *optimalizačný proces na povrchu fitnes funkcie*, kde sa hľadá genotyp odpovedajúci globálnemu maximu.

Replikačný proces budeme rozlišovať ako unárny (asexuálny) a binárny (sexuálny). Pri unárnej replikácii jeden replikátor - rodič sa zúčastňuje procesu, zatiaľ čo pri binárnej replikácii dva replikátory – rodičia sa zúčastňujú procesu. Rodičia (rodič) sú kvázináhodne vybraní z populácie v závislosti na ich fitnes (replikátory s väčším fitnes s väčšou pravdepodobnosťou vstupujú do replikácie) a produkujú nové replikátory - potomkov. Budeme rozlišovať tieto tri zložky replikačného procesu

(1) *selekcia* rodičov,
(2) *replikácia* rodičov, pričom vznikajú potomkovia, a
(3) *návrat* potomkov do populácie.

V prvom kroku binárnej replikácie sa vyberú pomocou stochastického operátora O_{select} z populácie P dva replikátory $\boldsymbol{x}_1^{old} = O_{select}(P)$, $\boldsymbol{x}_2^{old} = O_{select}(P)$ tak, že pravdepodobnosť ich výberu je úmerná ich fitnes. V druhom kroku použitím stochastického operátora replikácie O_{repli} z rodičov dostaneme

[4] Heuristika v informatike znamená pravidlo, väčšinou intuitívne zavedené, ktoré poskytuje určitý návod, ako sa správať v zložitom prostredí, ktorého presný model nepoznáme. Tak napríklad, v neznámom lese je veľmi užitočnou heuristikou pravidlo, že sa používajú len označené chodníky. V mnohých prípadoch tieto označené chodníky nereprezentujú optimálnu cestu, ale zabezpečia nám, že sa dostaneme do cieľa.

potomkov $\boldsymbol{x}_1^{new}$ a $\boldsymbol{x}_2^{new}$.

$$\left(\boldsymbol{x}_1^{new}, \boldsymbol{x}_2^{new}\right) = O_{repli}\left(\boldsymbol{x}_1^{old}, \boldsymbol{x}_2^{old}\right)$$

V unárnej (asexuálnej) replikácii sa na tvorbe potomkov podieľa len jeden replikátor – rodič. Formálne, tento proces môžeme vyjadriť takto

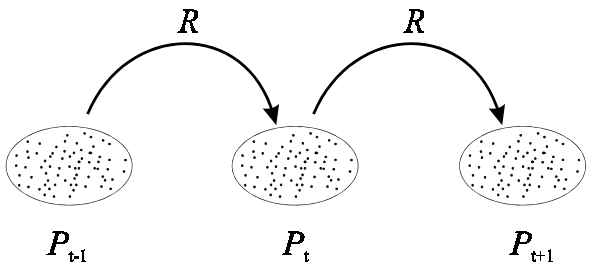
$$\boldsymbol{x}^{old} = O_{select}(P) \text{ a } \boldsymbol{x}^{new} = O_{repro}\left(\boldsymbol{x}^{old}\right)$$

V prvom kroku je kvázináhodne vybraný rodič, v druhom aplikáciou stochastického operátora reprodukcie O_{repro} sa z replikátora - rodiča vytvorí replikátor – potomok. Podobne ako v predchádzajúcej binárnej replikácii, v treťom kroku sa rieši problém návratu potomka do populácie. V nasledujúcej časti tejto kapitoly si ukážeme, za akých podmienok je potrebné uvažovať reprodukčný proces s účasťou dvoch rodičov alebo len jedného rodiča.

```
P:=náhodne vygenerovaná populácia
   replikátorov;
t:=0;
pokiaľ t<tmax urob
   t:=t+1;
   Q:=∅;
   pokiaľ |Q|<|P| urob
      x1:=Oselect(P);
      x2:=Oselect(P);
      (x1',x2'):=Orepli(x1,x2);
      Q:=Q∪{x1',x2'};
   P:=Q;
```

Algoritmus 7.1. Pseudokód univerzálnej Darwinovej evolúcie. Algoritmus je inicializovaný náhodnou generáciou populácie `P`. Evolúcia populácie je obsiahnutá vo vnútri **`pokiaľ-urob`**-cyklu, ktorý sa opakuje $t_{\max}$-krát. V rámci jednej epochy sa vytvorí nová populácia `Q`, ktorá sa tvorí opakovanou replikáciou dvoch rodičovských genotypov, ktoré majú vysoký fitnes. Nová populácia `Q` nahradí pôvodnú populáciu `P`.

Pseudopascalovský kód takto určenej Darwinovej evolúcie je daný Algoritmom 7.1. Pripomeňme si Bennetovu myšlienku [XX], že Darwinova evolúcia je algoritmus. Tým chcel zdôrazniť jej univerzálnosť, nezávislosť od materiálnej realizácie replikátorov - živých systémov.



Obrázok 7.3. Schematické znázornenie rekurzívnosti Darwinovej evolúcie, nová populácia sa vytvára z predchádzajúcej populácie aplikáciou stochastického operátora reprodukcie R.

Darwinova evolúcia môže byť interpretovaná ako *rekurentný proces*, v ktorom nasledujúca populácia je vytvorená reprodukciou predchádzajúcej populácie (pozri obr. 7.3)

$$P_{t+1} = R(P_t)$$

Funkcia R kvázináhodne (vzhľadom k fitnes replikátorov) priradí k populácii P_t nasledujúcu populáciu P_{t+1}. Cieľom tejto rekurzie - evolúcie darwinovského systému – je spontánna emergencia replikátorov s vysokým fitnes, ktoré sa vyvinuli z počiatočných, náhodne generovaných replikátorov, tvoriacich populáciu P_0. Hybnou silou evolúcie darwinovského systému je (1) prírodný výber a (2) spontánne mutácie. Z tohto pohľadu môžeme povedať, že evolúcia systému prebieha na hrane chaosu a poriadku. Deterministické mechanizmy selekcie najlepšie prispôsobených replikátorov pomocou prírodného výberu a neustále sa meniaca populácia replikátorov dôsledkom stochastických mutácií pre replikačnom procese, sú hlavným zdrojom evolúcie systému, postupného zvyšovania priemerného fitnes v celej populácii.

7.2 Genetické algoritmy[5]

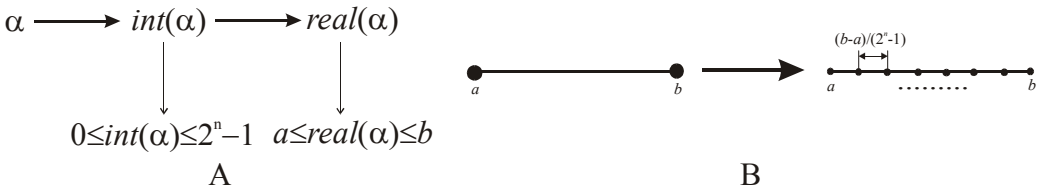
Genetické algoritmy boli vynájdené Johnom Hollandom [xx] počiatkom 70-tych rokov minulého storočia. Po určitej nábehovej 10-ročnej perióde rozpakov a mlčania v komunite informatikov sa stali jednou z najbúrlivejšie rozvíjajúcich sa oblastí informatiky a umelej inteligencie. Možno konštatovať, že spolu s neurónovými sieťami tvoria jadro novovznikajúcej oblasti nazývanej počítačová inteligencia, ktorá je schopná riešiť už praktické inžinierske problémy z informačných technológií, ktoré majú vysoký stupeň „inteligentnosti“[6]. V tejto podkapitole ukážeme základné princípy genetických algoritmov [xx] a ukážeme niektoré ich úspešné aplikácie. Genetické algoritmy budú formulované ako špeciálna aplikácia darwinovského systému k optimalizácii funkcií v binárnej reprezentácii premenných.

Základný pojem genetických algoritmov je binárna reprezentácia reálnych čísel. Nech binárny vektor α dĺžky k $\alpha = (\alpha_1\alpha_2...\alpha_k) \in \{0,1\}^k$ je interpretovaný ako celé číslo

$$int(\alpha) = \sum_{i=1}^{k} \alpha_i 2^{k-i} = \alpha_1 2^{k-1} + \alpha_2 2^{k-2} + ... + \alpha_{k-1} 2 + \alpha_k$$

K tomuto celému číslu jednoduchým spôsobom priradíme reálne číslo, ktoré môže byť chápané ako aproximácia reálneho čísla $x \in [a,b]$ (pozri obr. 7.4)

$$x \approx real(\alpha) = a + \frac{b-a}{2^k - 1} int(\alpha)$$



[5] Genetické algoritmy (GA) tvoria širokú triedu algoritmov [xx], ktorých spoločnou črtou je ich základ v univerzálnom darwinizme a využívanie binárnej reprezentácie riešeného problému.

[6] Pod inteligentnými informačnými technológiami rozumieme dnes také riešenia, ktoré vykazujú vysoký stupeň samostatnosti, adaptability a učenia sa na zmenené prostredie, schopnosť komunikovať v prirodzenom jazyku, a pod.

Obrázok 7.4. (A) Znázornenie prechodu od binárneho reťazca k reálnemu (racionálnemu) číslu. (B) Pri prechode z reálnej reprezentácie na binárny reprezentáciu, reálne čísla ležiace na úsečke $[a,b]$ sú vyjadrené (aproximované) bodmi, medzi ktorými je rovnaká vzdialenosť $(b-a)/(2^n-1)$ (pre i=0,1,…,2^n-1)

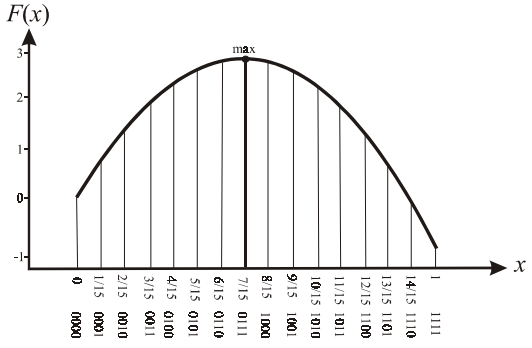
Tento vzťah medzi celými číslami v binárnej reprezentácii a ich reálnych (resp. racionálnych) reprezentácií pre jednoduchý prípad k=4 a interval [0,1] je znázornený na nasledujúcej tabuľke

No.	α	*int*(α)	*x*=*real*(α)	*F*(*x*)
1	0000	0	0.00000 (0/15)	0.00000
2	0001	1	0.06667 (1/15)	0.86667
3	0010	2	0.13333 (2/15)	1.60000
4	0011	3	0.20000 (3/15)	2.20000
5	0100	4	0.26667 (4/15)	2.66667
6	0101	5	0.33333 (5/15)	3.00000
7	0110	6	0.40000 (6/15)	3.20000
8	0111	7	0.46667 (7/15)	3.26667
9	1000	8	0.53333 (8/15)	3.20000
10	1001	9	0.60000 (9/15)	3.00000
11	1010	10	0.66667 (10/15)	2.66667
12	1011	11	0.73333 (11/15)	2.20000
13	1100	12	0.80000 (12/15)	1.60000
14	1101	13	0.86667 (13/15)	0.86667
15	1110	14	0.93333 (14/15)	0.00000
16	1111	15	1.00000 (15/15)	-1.00000

V druhom stĺpci tabuľky sú uvedené všetky možné (16) binárne reťazce dĺžky k=4, tretí stĺpec obsahuje celočíselnú interpretáciu binárnych reťazcov, štvrtý stĺpec obsahuje reálne (racionálne) čísla z intervalu [0,1], ktoré sú podľa vyššie uvedeného vzorca priradené binárnym reťazcom. Posledný piaty stĺpec obsahuje funkčné hodnoty jednoduchej kvadratickej funkcie

$$F\left(x\right)=14x-15x^2$$

pre $0\le x\le 1$. Priebeh tejto funkcie je znázornený na obr. 7.5, kde sú taktiež ukázané funkčné hodnoty v bodoch odpovedajúcich binárnej reprezentácii čísel z intervalu [0,1] reťazcami dĺžky k=4.



Obrázok 7.5. Priebeh kvadratickej funkcie $F\left(x\right)=14x-15x^2$ na intervale [0,1]. Na horizontálnej osi sú znázornené jednotlivé body, ktoré vznikli binárnou reprezentáciou reálnej premennej reťazcami dĺžky k=4. Funkcia má maximálnu hodnotu pre x_{opt}=7/15, s funkčnou hodnotou $F(x_{opt})$=3.266667.

.

Ohodnotenie replikátorov veličinou fitnes v genetickom algoritme sa robí pomocou funkcie, ktorú chceme optimalizovať. Nech $F(x)$ je účelová funkcia, definovaná na intervale $[a,b]$, na ktorom budeme hľadať jej maximálnu hodnotu

$$F\left(x_{opt}\right) = \max_{x\in[a,b]} F\left(x\right)$$

Predpokladajme, že reálne čísla z intervalu $[a,b]$ sú aproximované binárnymi reťazcami dĺžky k, potom spojitý optimalizačný problém je prevedený na diskrétny optimalizačný problém nad binárnymi reťazcami dĺžky k

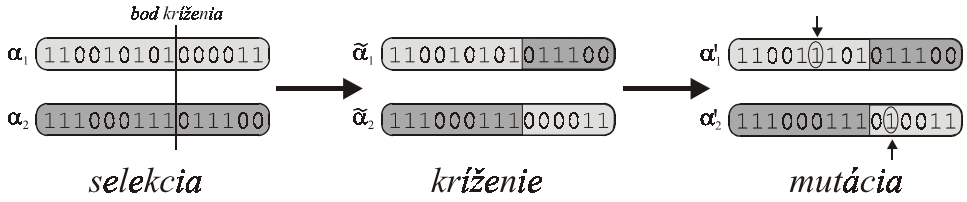
$$F\left(\alpha_{opt}\right) = \max_{\alpha\in\{0,1\}^k} F\left(\alpha\right)$$

Majme dva replikátory α_1 a α_2, ich ohodnotenie fitnes musí vyhovovať podmienke $F\left(\alpha_1\right) > F\left(\alpha_2\right) \Rightarrow f\left(\alpha_1\right) > f\left(\alpha_2\right)$, kde $f(\alpha)$ je fitnes priradené replikátoru – reťazcu α. Tento spôsob výberu fitnes vyplýva zo skutočnosti, že zostrojujeme genetický algoritmus, ktorý bude hľadať optimálne riešenie maximalizačného problému účelovej funkcie $F(x)$. Preto, ak pre dva replikátory α_1 a α_2 platí $F\left(\alpha_1\right) > F\left(\alpha_2\right)$, potom fitnes priradený α_1 musí byť väčší, ako fitnes priradený α_2, $f\left(\alpha_1\right) > f\left(\alpha_2\right)$, t.j. pre nasledujúcu evolúciu populácie P genotyp α_1 je nádejnejší ako genotyp α_2 . Najjednoduchšie riešenie problému priradenia fitnes replikátorom na základe ich hodnôt účelovej funkcie je jednoduchá identifikácia týchto dvoch veličín, $f(\alpha)=F(\alpha)$. V poslednom stĺpci vyššie uvedenej tabuľky sú dané funkčné hodnoty kvadratickej funkcie pre argumenty získané z binárnych reťazcov dĺžky k=4 , Tieto funkčné hodnoty môžu byť priamo stotožnené s fitnes replikátorov. To znamená, že v darwinovskej evolúcii prežívajú prednostne tie replikátory, ktoré majú vyšší fitnes t.j. vyššiu funkčnú hodnotu účelovej funkcie. Z tohto pohľadu potom môžeme očakávať, že v populácii replikátorov budú spontánne vznikať také replikátory, ktoré odpovedajú optimálnemu riešeniu optimalizačného problému, t.j. replikátory, ktoré sú priradené maximálnej funkčnej hodnote účelovej funkcie.

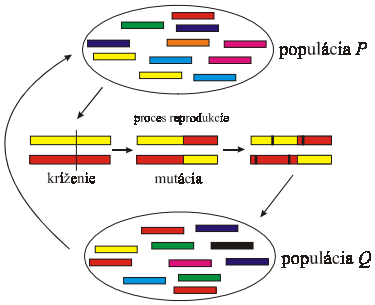
Všeobecný operátor replikácie O_{repli}, definovaný v rámci univerzálneho darwinizmu, má v prípade genetických algoritmov dve časti: *kríženie* a *mutáciu*. V prvom kroku rodičovské replikátory α_1 a α_2 (ktoré boli vybrané z populácia P s pravdepodobnosťou úmernou ich fitnes) sa krížia na nové replikátory $\tilde{\alpha}_1$ a $\tilde{\alpha}_2$, takto vzniklé nové replikátory sa ešte „zašumia“ mutáciami, vzniknú replikátory – potomkovia α_1' a α_2' (pozri obr. 7.6)

selekcia	$\alpha_1 = O_{select}\left(P\right)$ a $\alpha_2 = O_{select}\left(P\right)$
kríženie	$\left(\tilde{\alpha}_1,\tilde{\alpha}_2\right) = O_{cross}\left(\alpha_1,\alpha_2\right)$
mutácia	$\alpha_1' = O_{mut}\left(\tilde{\alpha}_1\right)$ a $\alpha_2' = O_{mut}\left(\tilde{\alpha}_2\right)$

Implementácia genetického algoritmu sa zhoduje s implementáciou evolúcie darwinovského systému zhrnutého v Algoritme 7.1. Replikačný proces v genetickom algoritme obsahuje kríženie a mutáciu, ich špecifikácia pre binárne reťazce je znázornená na obr. 7.5. Diagramatická interpretácia genetického algoritmu je znázornená na obr. 7.6.



Obrázok 7.5. Jednotlivé etapy binárnej (sexuálnej) replikácie. V prvej etape sú vybrané dva replikátory – rodičia s pravdepodobnosťou úmernou ich fitnes. V druhej etape, pri krížení, si replikátory pri kopírovaní prehodia určité časti svojich genotypov – binárnych reťazcov. V tretej etape, pri mutácii, sa v nových replikátoroch – potomkoch – v niektorých polohách reťazca s malou pravdepodobnosťou zmenia hodnoty genotypu.

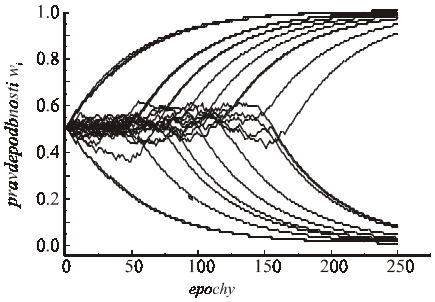


Obrázok 7.6. Diagramatická vizualizácia genetického algoritmu, implementovaného pomocou algoritmu 7.1. Z aktuálnej populácie *P* vyberieme dva replikátory – rodičov s pravdepodobnosťou úmernou ich fitnes. Tieto dva rodičovské replikátory sa v priebehu replikačného procesu krížia a potom mutujú, vzniklé dva nové replikátory – potomkovia tvoria novú populáciu *Q*. Ak populácia *Q* má rovnaký počet replikátorov ako pôvodná množina *P*, potom táto nová populácia nahradí pôvodnú populáciu, $Q \rightarrow P$.

Na záver tejto podkapitoly je potrebné ešte diskutovať problém zastavenia genetického algoritmu. Obvykle je inicializovaný populáciou replikátorov, ktoré sú náhodne generované. Evolúcia je zložená z opakovanej obnovy populácie pomocou rekurzie $P_{t+1} = R(P_t)$, kde populácia P_t je nahradená novou populáciou P_{t+1} pomocou procesu reprodukcie R. Proces prírodného výberu ponechá v populácii len tie replikátory, ktorých ohodnotenie je veľmi blízke k aktuálnemu maximálnemu fitnes (alebo je s ním totožné). Veľmi názornou ilustráciou priebehu genetického algoritmu je graf na obr. 7.7, ktorý obsahuje priebehy zložiek tzv. pravdepodobnostného vektora $\boldsymbol{w} = (w_1, w_2, ..., w_k)$. Tieto zložky w_i sú definované ako priemerné hodnoty *i*-tých zložiek binárnych replikátorov pre aktuálnu populáciu $P_t = \{\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, ..., \boldsymbol{\alpha}_p\}$, kde *i*-tý replikátor má tvar $\boldsymbol{\alpha}_i = (\alpha_1^{(i)}, \alpha_2^{(i)}, ..., \alpha_k^{(i)})$,

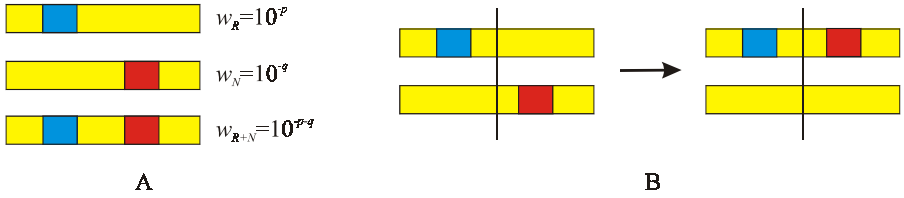
$$w_i = \frac{1}{p}\sum_{j=1}^{p} \alpha_i^{(j)} \quad (i = 1, 2, ..., k)$$

Aký je význam týchto zložiek pravdepodobnostného vektora? Veličina $0 \le w_i \le 1$ popisuje pravdepodobnosť výskytu '1' v *i*-tej polohe vo všetkých replikátoroch populácie P_t. V prípade, že $w_i \to 0$ (alebo $w_i \to 1$), potom všetky replikátory v *i*-tej polohe obsahujú 0 (alebo 1). V limitnom prípade, ak $w_i \to 0.5$, potom výskyt v *i*-tej polohe '0' alebo '1' je úplne náhodný. Priebeh týchto veličín je znázornený na obr. 7.7.



Obrázok 7.7. Priebeh pravdepodobnosti w_i pre evolúciu populácie replikátorov v genetickom algoritme. Populácia P_0 bola inicializovaná náhodne generovanými binárnymi replikátormi, v priebehu evolúcie jednotlivé pravdepodobnosti sa asymptoticky blížia buď k jednotkovej hodnote, alebo k nulovej hodnote.

Pravdepodobnosti w_i sú vhodné na kvantifikovanie homogenity celej populácie; ak tieto pravdepodobnosti sú skoro rovné 1 (alebo 0), potom replikátory populácie majú len malú variabilitu, z čoho vyplýva, že ďalší priebeh evolúcie nemôže priniesť nič podstatného (s veľmi malou pravdepodobnosťou). V tomto prípade môžeme genetický algoritmus zastaviť a replikátor s najväčším fitnes (ktorých je v populácii prevážna väčšina) reprezentuje finálne riešenie optimalizačného problému účelovej funkcie $F(x)$.



Obrázok 7.9. Znázornenie významu kríženia pre akceleráciu vzniku komplexných vlastností fenotypu v priebehu evolúcie. Diagram A znázorňuje tri rozdielne replikátory, z ktorých jeden má vlastnosť A, druhý má vlastnosť B a tretí má obe vlastnosti A a B. Nech tieto vlastnosti sú v genotype reprezentované spojitými oblasťami, ktoré sa neprekrývajú. Pravdepodobnosť spontánneho vzniku týchto jednotlivých oblastí je veľmi malá, t.j. pravdepodobnosť spontánneho vzniku oboch vlastností v jednom genotype je potom skoro zanedbateľná. Diagram B ukazuje význam kríženia pre spojenie oboch vlastností do jedného genotypu. Môžeme konštatovať, že kríženie replikátorov podstatne akceleruje rýchlosť evolúcie tým, že umožňuje spájanie spontánne vzniklých samostatných vlastností v genotype.

Položme si otázku, aký význam má operácia kríženia pri replikácii genotypov a aký má vlastne význam pre genetický algoritmus? Za akých podmienok je táto operácia významná, kedy môže byť ignorovaná? Pri hľadaní odpovedí na tieto a podobné otázky obrátime pozornosť na biologickú argumentáciu, ktorá vysvetľuje dôvody prečo je darwinovská evolúcia úspešná pri vzniku nových druhov lepšie prispôsobených k boju o prežitie v populácii. Predpokladajme, že v populácii spontánne vznikli replikátory - jedinci, ktorí majú dve nezávislé vlastnosti (obrazne môžeme povedať, že majú buď "ruky" alebo "nohy"), cieľom evolúcie nech je vznik jedincov, ktorí majú súčasne obe tieto vlastnosti. Pravdepodobnosť w_{R+N} spontánneho vzniku chromozómu, ktorý súčasne obsahuje jednu vlastnosť a aj druhú vlastnosť je v porovnaní s pravdepodobnosťami spontánneho vzniku chromozómov obsahujúcich len prvú vlastnosť w_R alebo len druhú vlastnosť w_N, potom veľmi malá, $w_R = 10^{-p}, w_N = 10^{-q}, w_{R+N} = 10^{-p-q}$. Táto nepriaznivá situácia sa dramaticky zmení, keď uvažujeme zapojenie "kríženia" v procese reprodukcie, pozri obr. 7.9. Z týchto jednoduchých úvah vyplýva skutočnosť, že v GA je kríženie chromozómov základnou "hnacou silou" evolúcie populácie smerom k populáciám obsahujúcim optimálny chromozóm s častou frekvenciou výskytu vtedy, ak existujú dôležité vlastnosti, ktoré sú reprezentované

v genotype navzájom sa neprekrývajúcimi blokmi. Ak táto vlastnosť nie je splnená, potom použitie kríženia v genetických algoritmoch je veľmi otázne, ba až problematické.

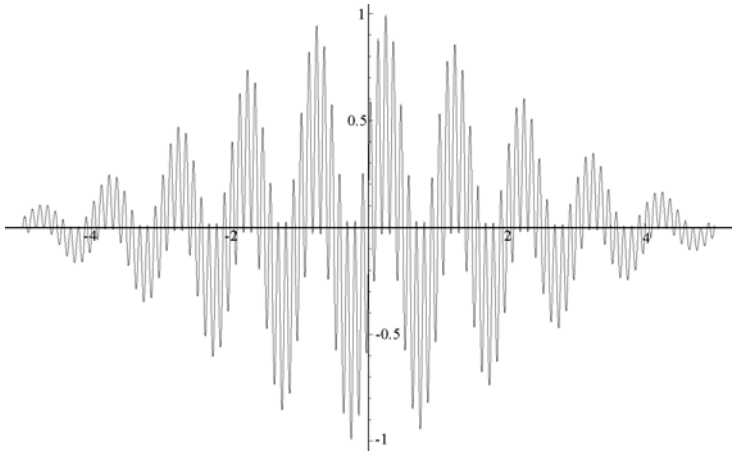
7.2 Aplikácia genetického algoritmu ako globálneho optimalizátora

Študujme funkciu

$$F(x) = e^{-0.1x^2}\, sin(10\pi x) cos(8\pi x)$$

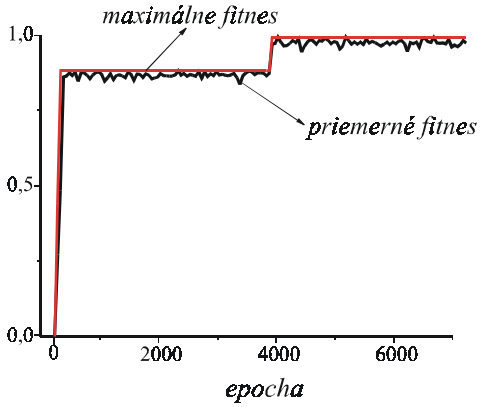
na uzavretom intervale [-5,5], priebeh tejto funkcie je znázornený na obr. 7.10, pomocou ktorého môžeme aj odhadnúť globálne maximum funkcie, jeho presná hodnota je získaná použitím Newtonovej metódy

$$f(x_{opt}) = 0.99377,\ x_{opt} = 0.249969$$



Obrázok 7.10. Priebeh funkcie $F(x) = e^{-0.1x^2}\, sin(10\pi x) cos(8\pi x)$ na intervale [-5,5]. Funkcia je silne multimodálneho charakteru, má množstvo extrémov (maxím a miním), pre klasické optimalizačné metódy je hľadanie globálneho maxima (alebo minima) obtiažny problém.

V našich simulačných výpočtoch sme predpokladali, že populácia obsahuje 200 binárnych replikátorov dĺžky k=30. Dá sa ukázať, že takto zvolená binárna reprezentácia špecifikuje reálne číslo z intervalu [-5,5] s presnosťou na 7 dekadických miest za desatinnou bodkou. Fitnes replikátorov bude stotožnený priamo s funkčnou hodnotou účelovej funkcie $f(\alpha) = F[real(\alpha)]$. Výskyt mutácií v replikačnom procese je určený pravdepodobnosťou jednobodovej mutácie P_{mut}=0.001, ktorá sa interpretuje tak, že idúc po reťazci replikátora, každá jeho binárna hodnota je zmenená (zmutovaná) s pravdepodobnosťou P_{mut}. Populácia bola inicializovaná replikátormi – binárnymi reťazcami obsahujúcimi len nuly, $\alpha = 000...0$. Priebeh evolúcie je znázornený na obr. 7.11, na ktorom sú grafy priemerného fitnes a maximálneho fitnes pre jednotlivé etapy evolúcie $1 \le t \le 7000$. Z tohto obrázku vyplýva, že genetický algoritmus bol úspešný v nájdení globálneho riešenia.



Obrázok 7.11. Priebeh priemerného a maximálneho fitnes v priebehu genetického algoritmu aplikovaného k hľadaniu globálneho maxima multimodálnej funkcie $F(x) = e^{-0.1x^2}\, sin(10\pi x) cos(8\pi x)$ na intervale [-5,5]. Priebehy fitness tvoria schodové funkcie, na ktorých existujú pomerne dlhé etapy neutrality, v ktorých sa čaká na vhodnú mutáciu, ktorá zvýši funkčnú hodnotu. Približne od epochy t=4000 populácia väčšinou obsahovala replikátory, ktoré odpovedali optimálnemu riešeniu $f(x_{opt}) = 0.99377,\ x_{opt} = 0.249969$.

Zhrnutie

Univerzálny darwinizmus je zovšeobecnenie pôvodných Darwinových myšlienok o evolúcii pomocou koncepcie darwinovského systému. Darwinovský systém obsahuje populáciu replikátorov, ktoré sú schopné vlastnej replikácie, pričom ich pravdepodobnosť vstupovania do replikačného procesu je úmerná fitnes replikátorov. Replikátory sú charakterizované genotypom a fenotypom. Genotyp je tvorený informáciou, ktorá je obsiahnutá v stabilnom nosiči, a je kopírovaná do nosičov jeho potomkov. Fenotyp tvorí viditeľné, dynamické a aktívne telo replikátora. Darwinovský systém poskytuje formálne prostriedky pre jednoduchú algoritmizáciu darwinovskej evolúcie. Genetické algoritmy tvoria špecifickú počítačovú formu univerzálneho darwinizmu, pričom tvoria efektívny prostriedok pre hľadanie globálnych riešení zložitých optimalizačných problémov. V umelej inteligencii genetické algoritmy umožňujú jednoduchú implementáciu darwinovskej evolúcie počítačových systémov.

Úlohy

Úloha 7.1. Prepíšte algoritmus 7.1 do tvaru, v ktorom je použitý unárny (asexuálny) replikačný proces.

Úloha 7.2. Lamarckovská evolúcia sa líši od darwinovskej tým, že umožňuje genetický prenos získaných vlastností do nasledujúcich generácií replikačným procesom. Táto hypotetická vlastnosť replikátora sa vyjadrí tak, že sa naň aplikuje operátor O_{opt} pomocou ktorého sa lokálne optimalizuje štruktúra genotypu; t.j. $\alpha' = O_{opt}(\alpha)$, kde nový genotyp môže

mať mierne vyššie fitnes, $f(\alpha') \geq f(\alpha) + \delta$, kde δ je malé kladné číslo. Prepíšte algoritmus 7.1 do formy umožňujúcej lamarckovský prenos získaných vlastností.

Úloha 7.3. Pretransformujte reálne čísla x=0.5, 0.55, 1.0 do binárnej reprezentácie dĺžky k=4 a na intervale [-1,1].

Úloha 7.4. Transformujte binárne reťazce 001100,101010 a 000111 na reálne čísla z intervalu [0,2].

Úloha 7.5. Uskutočnite binárny (sexuálny) replikačný proces pre dvojice binárnych reťazcov $\alpha_1 = 001100, \alpha_2 = 111100$ a $\alpha_1 = 000011, \alpha_2 = 001100$. Znázornite získané výsledky pomocou reálnej reprezentácie tak rodičovských reťazcov, ako aj nových reťazcov - potomkov.

Úloha 7.6. Simulujte ručne genetický algoritmus pre populáciu obsahujúcu 10 replikátorov dĺžky k=4, ktorých ohodnotenie fitnes je uvedené v tabuľke nad obr. 7.5, pričom replikačný proces je unárny (asexuálny). Vykonajte toľko evolučných epoch, aby populácia obsahovala niekoľko epoch po sebe aspoň 7 replikátorov s rovnakým fitnes. Aké majú fitnes tieto rezultujúce replikátory?

Úloha 7.7. Homogenitu populácie dobre popisuje parameter usporiadania definovaný takto

$$\chi(P) = \frac{4}{k} \sum_{i=1}^{k} (w_i - 0.5)^2$$

Aké vlastnosti má tento parameter v priebehu evolúcie?

Otázky na opakovanie

Otázka 7.1. Ako je definovaný darwinovský systém?

Otázka 7.2. Aký je vzťah medzi genotypom, fenotypom a fitnes?

Otázka 7.3. Ktoré sú základné prvky procesu replikácie?

Otázka 7.4. Akú má podobu pseudokód univerzálnej Darwinovej evolúcie?

Otázka 7.5. Aká je binárna reprezentácia reálneho čísla?

Otázka 7.6. Aký je vzťah medzi ohodnotením binárnych replikátorov pomocou fitnes a hodnotou účelovej funkcie v GA?

Otázka 7.7. Aké sú základné prvky replikácie binárnych reťazcov – replikátorov v GA?

Otázka 7.8. Aký je význam kríženia binárnych replikátorov v GA?

8 Umelý život

Otázka je, čo je to vlastne umelý život ("Artificial Life")? Pretože definícia tejto novej búrlivo sa rozvíjajúcej oblasti informatiky nie je vôbec jasná, dá sa umelý život definovať skôr tým, čo doň rôzni ľudia zahŕňajú. Najčastejšie ide o simuláciu životných prejavov jednotlivcov alebo skupín. Umelý život zasahuje do viacerých už zavedených oblastí - okrem biológie a informatiky aj do fyziky, sociológie, psychológie, ekonómie (modelovanie situácií, kedy si firmy konkurujú alebo kedy spolupracujú, zapojenie teórie hier, riadenie priemyselných komplexov) a filozofie. Témou umelého života sa zaoberajú tak populárno-vedecké knihy [XX], vedecké knihy, ako aj početné odborné konferencie a časopisy. Centrom aktivít skupín zaoberajúcich sa umelým životom je USA, presnejšie povedané Santa Fe Institute, laboratóriá v Los Alamos a MIT, a menšie centrá v Japonsku a v Európe. Na Slovensku pracuje v oblasti umelého života napr. skupina vedená doc. Júliusom Csontóm v Košiciach [Al] a v Českej republike profesorom Jozefom Kelemenom v Opave [KogUI, StrAg]. Do oblasti umelého života zasahuje aj oblasť chaosu, predovšetkým vo forme modelovania pomocou fraktálov a celulárnych automatov, a oblasť simulácie sociálnych systémov; o týchto témach sa ale pojednáva v kapitolách 9 a 10.

Existujú dva prístupy k umelému životu:

Prvý je "*vizionársky*" - tvrdiaci, že umelý život znamená vytvorenie a vylepšovanie účinnosti nových foriem života, iba v inom médiu ako sú uhľovodíky, teda väčšinou na báze kremíkových čipov v počítačoch (ako napr. počítačové vírusy) alebo v "železe" predstavovanom (nano)robotmi. Proces biologickej evolúcie viedol ku genotypu vytvárajúcemu fenotyp, schopný manipulovať s vlastným genotypom priamo: kopírovať ho, meniť alebo vytvárať celkom nový. Tento prístup síce funguje zatiaľ iba pri umelom živote prezentovanom počítačovými programami, no pri terajšom prudkom rozvoji biotechnológií a génových manipulácií možno predpovedať, že v polovici 21. storočia už môže byť reálny aj "biologický" umelý život (teda pokiaľ ako taký nebudeme chápať už dnes vytvárané rastlinné druhy s geneticky vnesenou informáciou pre vytváranie protilátok proti škodcom, alebo produkujúce látky použiteľné ako vakcína proti cholere či malárii). S týmto prístupom sa prelína z opačnej strany aj "technizácia" ľudského života, od existujúcich transplantátov končatín reagujúcich na nervové povely alebo kardiostimulátorov v srdci až po dnes ešte vzdialenú možnosť priameho napojenia človeka na počítač, doteraz realizovanú iba primitívnymi prostriedkami virtuálnej reality.

Druhý, "*prízemnejší*" smer tvrdí, že cieľom umelého života je vypracovať nové metódy na štúdium života, vedúce k lepšiemu pochopeniu tohto biologického fenoménu. Umelý život poskytuje možnosť robiť experimenty, ktoré by boli v tradičnej biológii extrémne komplikované alebo nemožné. Výhodná je tu možnosť presne opakovať experiment a tiež rýchlosť simulácií v porovnaní s experimentmi. Zatiaľ čo klasická biológia sa snaží život analyzovať, umelý život smeruje k syntéze aspoň niektorých čŕt života zo základných vlastností živej hmoty. Kľúčovým rysom tohto oboru je formulovanie hlavných pravidiel správania sa jednotlivých ”organizmov” a ich replikácie, pričom cieľom je dosiahnuť

evolúciu organizmov rovnako ako vzťahov medzi organizmami (”emergent[1] behavior”), ako je napr. súperenie, spolupráca, parazitizmus, vzťah dravec - korisť alebo vývoj druhov. Veľa prác smeruje k počítačovej simulácii umelých bytostí a umelých svetov, v ktorých tieto bytosti žijú. Predmetom štúdia je teda nielen napodobovanie pozemského života, ale aj simulácia, ako by život mohol vyzerať za iných podmienok.

Zakladateľ umelého života Christopher Langton na prvej konferencii venovanej tomuto odboru, ktorá sa konala r. 1987 definoval umelý život ako štúdium umelých systémov, ktoré vykazujú správanie charakteristické pre prírodné živé systémy: samoorganizáciu, adaptáciu, evolúciu, koevolúciu, metabolizmus atď. Je to snaha o vysvetlenie fenoménu života v akejkoľvek forme, bez ohľadu na špecifické príklady, ktoré sa vyvinuli na Zemi. To zahŕňa biologické a chemické experimenty, počítačové simulácie a čisto teoretické rozbory. Predmetom výskumu sú procesy prebiehajúce na molekulárnej, sociálnej a evolučnej úrovni. Cieľom je extrahovať základné princípy organizácie živých systémov a ich vývoja.

Vytvorenie týchto procesov v inom prostredí (inom ako v počítačoch) ich sprístupňuje novým druhom experimentálnej manipulácie a testovania. Pritom evolúcia hardvéru a syntetická evolúcia biomolekúl sú zaujímavé aj priamo pre prax. Na druhej strane, populácie dátových štruktúr v počítači sa dajú použiť na reprezentovanie biologických jedincov (dravca a koristi, mravcov, buniek apod.). Vo väčšine príkladov umelého života populácie dátových štruktúr nereprezentujú priamo žiaden žijúci organizmus alebo proces, ale skôr sa podriaďujú umelo vytvoreným zákonom abstrahovaným zo zákonov platných pre živé organizmy. Najväčší nadšenci pritom tvrdia, že tieto dátové štruktúry sú fakticky živé. Steven Levy [1] dôvodí takto: ”Život je dynamický fyzikálny proces a keď dokážete duplikovať tieto procesy na inak neživom materiále - stvorili ste život. Ten môže byť nezávislý na materiále. Môže dokonca vzniknúť v počítači.” Vzhľadom na to, že dosiaľ neexistuje žiadna celkom kvalitná definícia života, je toto tvrdenie zdrojom búrlivých diskusií.

Základný problém, pri ktorom je ťažko hľadať riešenie, je odpoveď na otázku **ako definovať život**? Prakticky všetky definície majú svoje výnimky a obmedzenia. Napríklad: život musí byť schopný autoreprodukcie - mulica (sterilný kríženec kobyly a osla, ktorý nie je principiálne schopný ďalšieho rozmnožovania) teda nie je živá?

Vlastnosti definujúce život môžu byť napr. nasledujúce:

1. Život je skôr systém existujúci v čase a priestore ako špecifický materiálny objekt.
2. Je schopný sa reprodukovať, či už sám alebo s pomocou hostiteľského organizmu.
3. Obsahuje informáciu, pomocou ktorej je vybudovaný.
4. Obsahuje metabolizmus premeny jedného druhu hmoty na iný spojený s produkciou energie.
5. Funkčne interaguje s okolím.
6. Jeho časti sa navzájom funkčne dopĺňajú.
7. Je stabilný v meniacom sa prostredí.
8. Je schopný vyvíjať sa.
9. Rastie.

Obrazy organizmu ako stroja a stroja ako organizmu sú staré stáročia. No až teraz sa obidva tieto obrazy pomaly začínajú prekrývať.

Základnými vlastnosťami života, ktoré sa snažíme vytvoriť umelo, sú autoreplikácia, samostatné rozhodovanie = seba-riadenie, limitovaná seba-obnova, evolúcia a učenie. Týmto približujeme systémy tvorené v umelom živote živým organizmom. No zatiaľ sa nedarí napodobniť všetky tieto vlastnosti súčasne. Väčšina príkladov umelého života sa zameriava na simuláciu aspoň jednej vlastnosti charakteristickej pre život. Zo všeobecného pohľadu

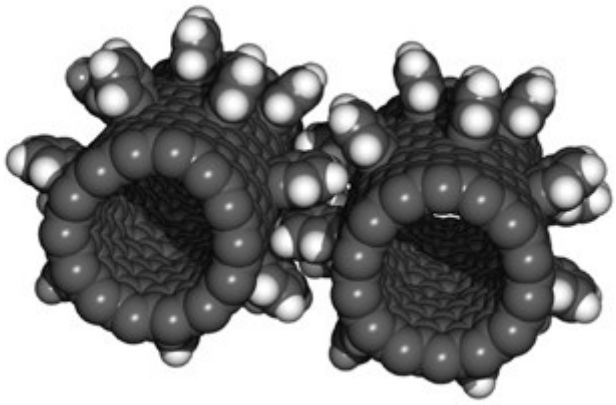
[1] "Emergencia" patrí medzi kľúčové termíny v prístupoch umelého života. Chápe sa ako spontánne vynáranie zložitejších štruktúr v systéme, ktoré tam pôvodne neboli prítomné.

môže ako príklad výtvorov podobných životu slúžiť celosvetová telefónna sústava, vývoj počítačových vírusov, virtuálna realita v počítači alebo počítačové modely rozvoja ekonómie a/alebo ekológie. Priemysel smeruje k používaniu metód založených na biologických princípoch, tieto sú menej náročné na spotrebu materiálu a energie, zložitosť vytváraných vecí a procesov môže dosahovať hranice biologickej integrácie.

Aplikácia metód genetického inžinierstva, aj keď po stáročia v praxi využívaných šľachtiteľmi ovocia alebo domácich zvierat, robí život až teraz inžiniersky spracovateľným.

Živé systémy majú vnútornú štruktúru vykazujúcu niekoľko základných vlastností: absenciu striktného centrálneho riadenia, čiastočnú autonómiu nižších podštruktúr, vysokú úroveň prepojenosti medzi podštruktúrami, nelineárne vzájomné ovplyvňovanie podštruktúr v sieti.

Z týchto vnútorných vlastností potom vyplývajú vonkajšie vlastnosti živých systémov: adaptabilita (v rámci učenia jedného živočícha, drobné "taktické" zmeny), vývoj (v rámci druhu, zmena základných génových vlastností alebo zmena celkového zamerania vývoja - z adaptácie génov k adaptácii kultúry), odolnosť spôsobená redundanciou, schopnosť narastania systému bez nutnosti meniť jeho základné vlastnosti (kŕdeľ vtákov alebo kobyliek môže narastať rádovo bez nutnosti meniť systém správania sa - jediná topológia schopná neobmedzeného rastu a neriadeného učenia je sieť). Prehľadávanie priestoru možných riešení evolúciou ale môže byť aj nevýhodné, pokiaľ sa životné prostredie nemení. Pri využití životu-podobných systémov vo vedecko-technických aplikáciách je treba zvážiť potrebu miery adaptibility navrhovaného systému v súvislosti s mierou premenlivosti riešenej úlohy v čase, rovnako ako stupeň samostatnosti systému v porovnaní s požiadavkou potreby priebežného zasahovania do systému užívateľom.



Obrázok 8.1. Ozubené kolesá z atómov použiteľné v nanorobotoch

Štúdium životu-podobných systémov môže byť užitočné napr. aj pre praktické účely softvérovej evolúcie inteligentných robotov, nanotechnológie (robotiky a výroby počítačových komponent alebo napríklad lekárskych ”nástrojov” na molekulárnej úrovni, pozri obr. 8.1) a robustných a adaptabilných počítačových programov pri využití životu podobných procesov.

Vo vzťahu ku klasickej umelej inteligencii je umelý život orientovaný ”opačným smerom”. Umelá inteligencia postupuje "zhora nadol", základom je dopredu zadaný komplexný problém s presne stanovenými pravidlami ako hranie šachu, chápanie textu alebo stanovenie diagnózy. Program musí byť inteligentný hneď od začiatku a závisí od programátora, ako tento cieľ zrealizovať prostriedkami informatiky. Umelý život, na rozdiel od programov umelej inteligencie, vzniká "zdola nahor". Začína od najjednoduchších elementárnych jednotiek a postupne sa dostáva evolúciou k zložitejším systémom. Je určený

na "prežitie" v komplexnom prostredí s meniacimi sa podmienkami. Umelý život sa snaží vytvoriť iba základné pravidlá, ktoré budú tak dobre vybrané, aby zložité správanie sa (a možno v budúcnosti aj istá ”inteligencia”) vzniklo (emergovalo) po čase spontánne.

Umelá inteligencia a umelý život majú k sebe najbližšie v type programov nazvaných **autonómny agent** [KelemenovXXX]. To sú programy, ktoré obsahujú špecifický druh vnímania okolitého prostredia. Na základe týchto vstupných informácií ovplyvňujú vlastný stav a stav susedných agentov alebo prostredia samotného. Agenti pracujú v operačnom systéme, databáze alebo v počítačovej sieti. Agent buď zberá informácie alebo ovplyvňuje svoje okolie, individuálne aj ”v tímovej práci”. Po počiatočnom nastavení pracuje bez ďalšieho ovplyvňovania programátorom. Pod názvom ”distribuovaná umelá inteligencia” tento prístup rieši praktické problémy, akými sú napr. riadenie letovej prevádzky letiska alebo prevádzky továrne, kedy každý stroj má svojho ”agenta” starajúceho sa o dodávky energie, surovín a o odvoz hotových produktov.

Zatiaľ najrozšírenejšími príkladmi umelého života (aj keď mnohí by ich za umelý život nepovažovali) sú počítačové vírusy a bunkové (celulárne) automaty, ktoré sú v najjednoduchšom prípade reprezentované Conwayovou hrou "život".

Počítačový vírus sa môže považovať aj za autonómneho agenta. V praxi sa počítačový vírus líši od autonómneho agenta tým, že je jednoduchší a má len jeden hlavný cieľ: rozmnožovať sa a rozširovať do ďalších počítačov. Okrem toho je tu otázka ničivých vedľajších účinkov vírusov. Pre umelý život sú vírusy špeciálne zaujímavé tým, že majú vlastnosti veľmi podobné biologickým vírusom. Akcie vírusov by teoreticky mohli byť aj pozitívne, napríklad udržiavanie integrity databázy.

Základným dôvodom na to, aby sa počítačové vírusy považovali za živé, je ich schopnosť autoreplikácie. Počítačový vírus je sekvenciou strojového kódu, ktorá sa skopíruje na jeden či viac "hosťovských" programov, keď sú tieto zaktivované. Keď sa tieto infikované programy spustia, kód vírusu sa aktivuje a vírus sa šíri ďalej. Podobne ako biologické vírusy ani počítačové vírusy nie sú schopné "žiť" samostatne, potrebujú hostiteľa.

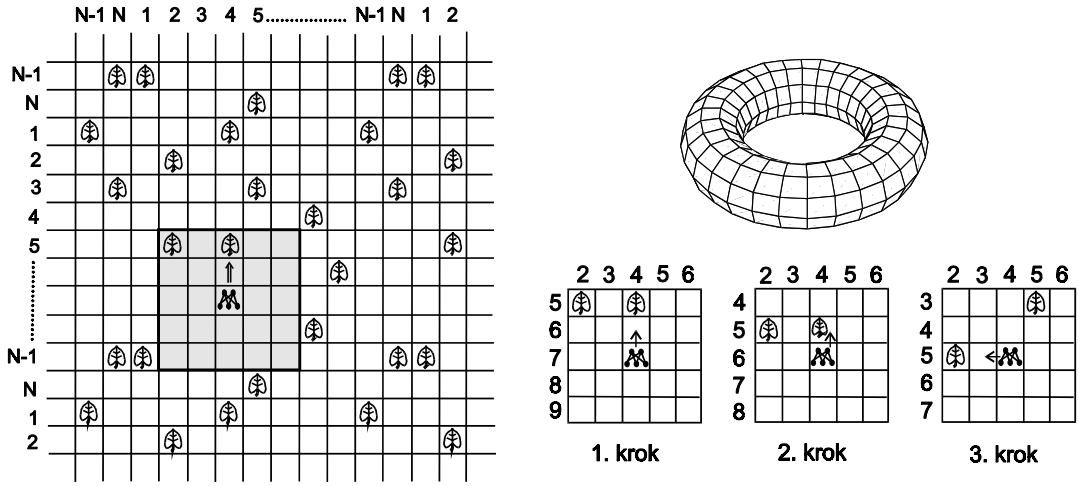
Základná vlastnosť, ktorá chýba vírusom, je schopnosť samostatne sa vyvíjať. Vírus, ktorý by toho bol schopný, by musel byť príliš veľký, a teda ľahko rozpoznateľný a zničiteľný antivírovými programami.

Virtuálna realita [40] má tiež blízko k umelému životu. Na rozdiel od umelého života je vo virtuálnej realite užívateľ aktívnym účastníkom procesov v umelo vytvorenej (simulovanej) realite. Na rozdiel od virtuálnej reality v simuláciách umelého života jedinci - programy po počiatočnom nastavení zvyčajne už nekomunikujú s užívateľom, ktorý je len pozorovateľom dejov vo virtuálnom svete, do ktorého nezasahuje.

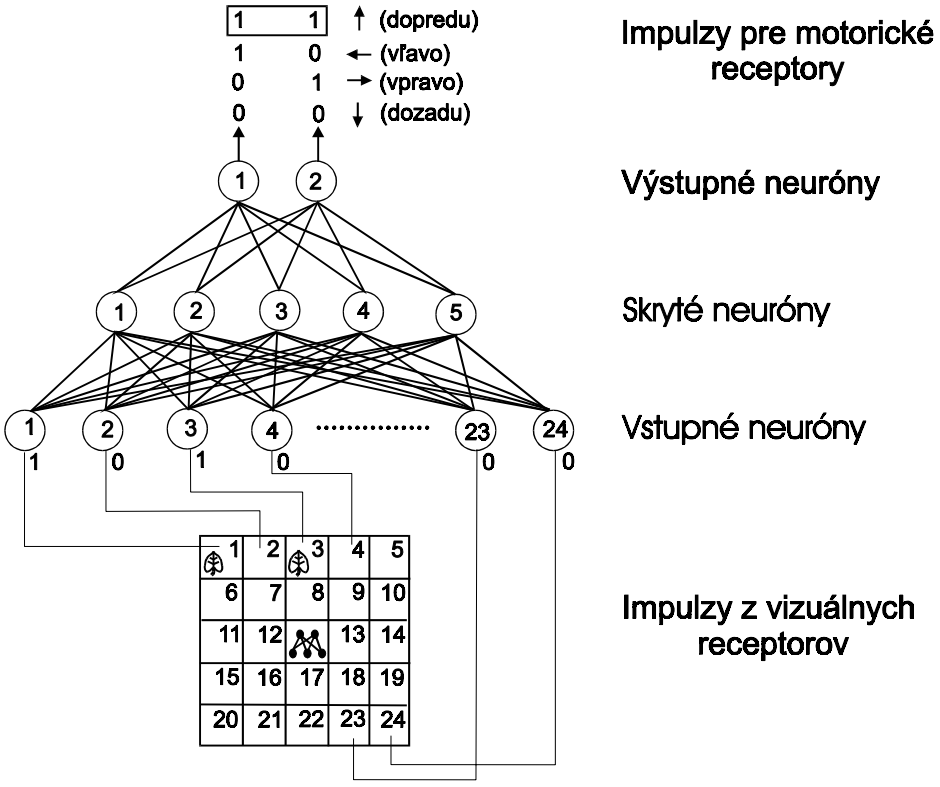
Základom väčšiny simulácií umelého života sú algoritmy dovoľujúce umelým ”bytostiam” vyvíjať sa a/alebo meniť svoje prostredie. Každý z týchto algoritmov je vedným oborom sám osebe so širokými vedeckými a priemyselnými aplikáciami. Základné algoritmy spadajú do dvoch odvetví: učiace sa algoritmy reprezentované neurónovými sieťami a evolučné algoritmy reprezentované predovšetkým genetickými algoritmami.

Neurónové siete (pozri kapitolu 5) majú hodnotu samy osebe ako univerzálne aproximátory a dá sa povedať, že samy predstavujú pole výskumu ďaleko rozsiahlejšie ako v súčasnej dobe umelý život. Pre autonómnych agentov potom neurónové siete predstavujú jednu z možných metód rozhodovania a súčasne učenia sa ako komunikovať s okolitým prostredím alebo rozpoznávať informácie.

Veľa aplikácií umelého života vybavuje svoje organizmy neurónovou sieťou, ktorá slúži ako umelý mozog schopný učiť sa z príkladov s vopred zadaným výsledkom. Typická je neurónová sieť zostavená zo vstupných neurónov, ktoré sú napojené na informácie získavané z vonkajšieho prostredia, zo skrytých vrstiev neurónov, ktoré vykonávajú väčšinu výpočtov, a z výstupných neurónov, ktoré určujú ďalšiu akciu organizmu - agenta (viď. obr. 8.2 a 8.3).



Obrázok 8.2. Ilustratívny príklad organizmu, ktorý je riadený neurónovou sieťou a nachádza sa na pravouhlej mriežke typu $N \times N$. Organizmus sa živí potravou znázornenou náhodne rozmiestnenými lístkami a má za cieľ pozbierať za svoj život (daný počtom horizontálnych alebo vertikálnych krokov na susedné pole) čo najväčší počet lístkov. Je schopný vnímať iba svoje najbližšie (vytieňované) okolie. Pri rekognoskácii prostredia musí určiť smer, kde je najviac najbližších lístkov, a pohybovať sa týmto smerom. Pravouhlá mriežka je stočená do toroidu - pneumatiky, aby sa nemohlo stať, že organizmus dôjde na okraj mriežky. Vpravo dole sú znázornené prvé tri kroky, pričom predpokladáme, že neurónová sieť vždy rozhodla správne a organizmus získal celkovo jeden lístok.

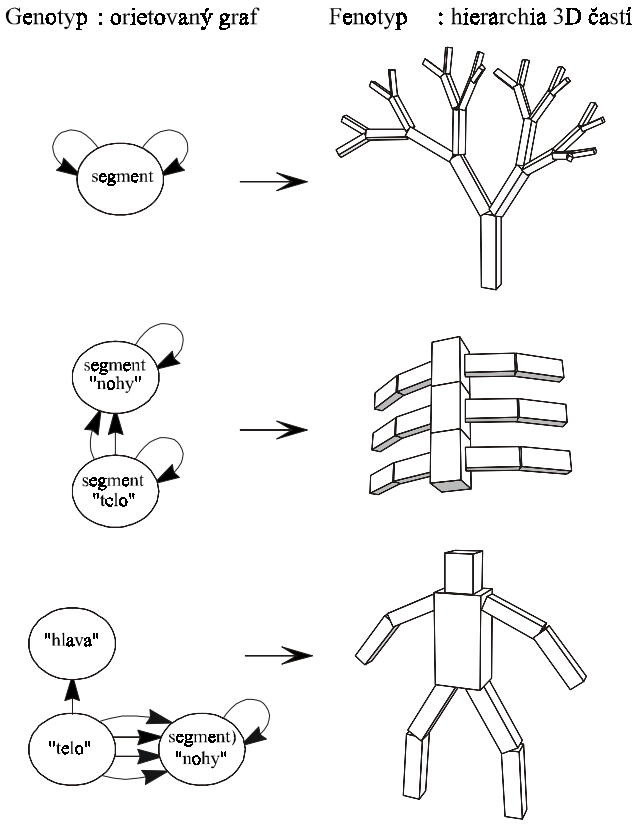


Obrázok 8.3. Znázornenie trojvrstvovej neurónovej siete organizmu z obr. 9.1. Táto riadiaca jednotka sa realizuje trojvrstvovou neurónovou sieťou. Vstupné neuróny získavajú signály z vizuálnych receptorov pre okolité políčka (0 pre neprítomnosť lístku, inak 1). Skryté neuróny pomáhajú spracovať prijatú informáciu pre výstupné neuróny, ktoré rozhodujú o smere budúceho pohybu organizmu. Spoje medzi neurónmi sú ohodnotené váhami, ktoré tak ovplyvňujú "inteligenciu" neurónovej siete. Takýchto organizmov je viac, ich úspešnosť v zbere potravy sa ohodnocuje, neúspešné zanikajú a tie najúspešnejšie sa množia, vymieňajú si vzájomne časti "dedičnej informácie" - váhy medzi neurónmi, takže organizmy sa stávajú stále efektívnejšími v zbere potravy.

Trénovanie (učenie) neurónovej siete pozostáva z nastavovania sily spojenia medzi jednotlivými neurónmi. Toto trénovanie možno robiť rôznymi prístupmi, medzi iným aj genetickými algoritmami. Niektoré techniky z evolučných algoritmov (kríženie a mutácia), ktorým sme sa venovali v 7. kapitole, slúžia na prenos informácie medzi organizmami a zabezpečujú evolúciu celej generácie - populácie organizmov - agentov.

Veľmi zaujímavou aplikáciou sú v tomto ohľade napr. umelé bytosti nazvané **Nornovia**, ktorí sú základom počítačovej hry nazvanej Creatures, vytvorenej r. 1996. Sú schopné učenia a evolúcia, sú vybavené systémom komplikovaných programov včítane neurónových sietí, obsahujú aj simuláciu fyziológie včítane hormónov. Ako príklad ich praktického využitia je možné uviesť ich virtuálne pilotovanie lietadiel EuroFighter v rámci projektu britského ministerstva obrany. Nornovia tu boli schopní v simulovanom prostredí vyvinúť známe bojové stratégie a aj niektoré zaujímavé nové, napr. stabilizáciu lietadla rotáciou okolo pozdĺžnej osi.

Karl Sims použil genetický systém na vývoj "umelých bytostí" kódovaných pomocou "genotypových grafov", kde odpovedajúci fenotyp je zložený z blokov, spojených kĺbmi, poháňaných svalmi a riadených "obvodmi", pozri obr. 8.4. Tieto umelé bytosti sa potom testovali na schopnosť pohybovať sa po povrchu, skákať, plávať a pod., pričom sa vyvinula veľká variabilita bytostí a nových typov pohybov.

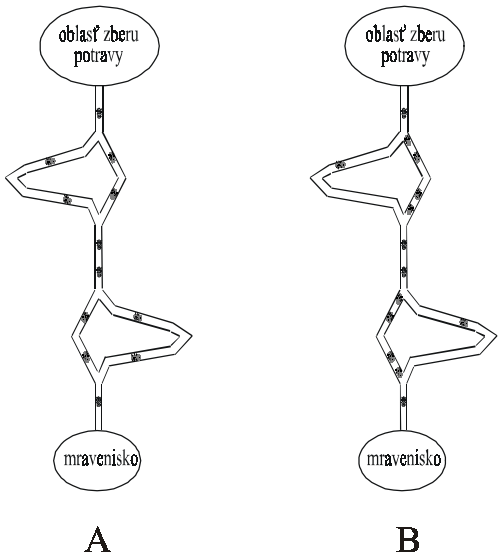


Obrázok 8.4. Navrhnuté príklady genotypových grafov a odpovedajúcich morfológií umelých bytostí

Zaujímavé je aj použitie simulátorov umelého života vo výučbe princípov biológie pre deti. Výrazný úspech tu dosiahol napr. Mitchell Resnick. Najpopulárnejší v tomto ohľade je asi program SimLife.

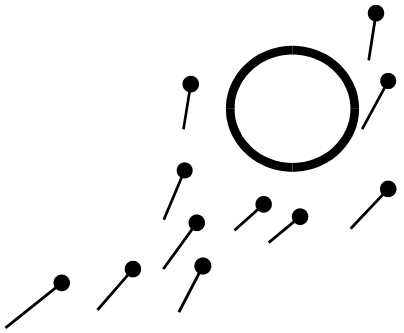
Typickým cieľom mnohých simulačných programov je **modelovanie kolónie mravcov**. Je veľa programov, ktoré takéto správanie napodobňujú. Mravce totiž pri relatívnej jednoduchosti svojich jednotlivcov vykazujú komplikované skupinové správanie. Sú tak schopné vykonávať úlohy, ktoré idú ďaleko nad rámec možností jednotlivého mravca. Robia

tak bez vzájomného spojenia ”telefónnou linkou”, bez centrálneho riadenia a pri častých poruchách informácie alebo zmenách prostredia. Mravčie kolónie stavajú cesty medzi hniezdom a zdrojom potravy, alebo vytvárajú živé mosty pri migrácii. Z opačného pohľadu sa dá takéto správanie využiť aj pre optimalizačné výpočty (pozri obr. 8.5).



Obrázok 8.5. Postupná optimalizácia cesty mravcov od mraveniska k zdroju potravy a späť od menej výhodnej A ku kratšej B. Mravce spolu nekomunikujú priamo a nemusia ako jednotlivci poznať najkratšiu cestu, vypúšťajú ale po ceste feromón (čím kratšia cesta, tým koncentrovanejší feromón) podľa ktorého sa riadia ostatné mravce pri rozhodovaní na križovatkách.

Ďalším príkladom emergentných vlastností je správanie sa vtáčieho kŕdľa bez centrálneho riadenia skúmané v prácach Craiga Reynoldsa (pozri obr. 8.6). Ten vytvoril základný model "vtáka", nazvaného "**boid**". Do počítačového modelu sa potom vložila populácia "vtákov" riadiacich sa troma základnými vlastnosťami: vyhýbať sa kolízii s okolitými "vtákmi", prispôsobiť rýchlosť okolitým "vtákom", snažiť sa držať sa blízko okolitých "vtákov" (môže sa pridať základný smer, ktorým vtáky letia, nutnosť vyhýbať sa prekážkam, pridať malé náhodné zmeny smeru a rýchlosti vtákov apod.). Každý "vták" pritom vidí iba svojich najbližších susedov. Tieto základné vlastnosti sú dostačujúce na vytvorenie prirodzeného správania sa kŕdľa. Keď sa napr. objaví prekážka, kŕdeľ sa bez akéhokoľvek centrálneho riadenia rozdelí na dve časti, ktoré sa po obletení prekážky spoja. Tieto princípy sa napr. používajú na realistickú animáciu netopierov alebo vtákov vo filmoch.



Obrázok 8.6. Zobrazenie ”žubrienok” (čiernych bodov s ”chvostíkom”), ktoré sa pohybujú smerom vpravo nahor a vyhýbajú sa prekážke (kruhu). Model je inšpirovaný správaním sa vtáčieho kŕdľa podľa Craiga Reynoldsa.

Existuje aj množstvo simulačných nástrojov umožňujúcich do veľkých podrobností simulovať životné prostredie. Takéto modely sú využívané napr. pre ekologické simulácie vplyvu zmien životného prostredia na ekosystém, čo je využiteľné pri plánovaní.

V **mikrobiológii** sa takéto simulátory dajú použiť napríklad na modelovanie zelených rias používaných na odstraňovanie ťažkých kovov z životného prostredia [XXCsonto].

Vznik života evolúciou neživého žiaľ v laboratórnych podmienkach púhou emergenciou asi nenapodobíme, preto je aj pre biológiu zaujímavé skúmať emergenciu "organizmov" v značne zjednodušenom prostredí. Virtuálny svet Toma Raya z University of Delaware nazvaný **Tierra** je prechodom od kódov počítačových vírusov k experimentálnemu štúdiu evolúcie umelých organizmov na úrovni genómu. Snaží sa poskytnúť prostredie, v ktorom môže darwinovská evolúcia prebiehať v počítači bez priameho riadenia alebo intervencie človeka. Ray vytvoril zjednodušený "počítač" v počítači s menším množstvom inštrukcií v umelom kóde napodobňujúcom strojový kód. V tomto kóde Ray potom napísal 80 bytový "vírus", ktorý bol schopný rozmnožovania. Takýto organizmus je lineárnym reťazcom inštrukcií a žije postupným vykonávaním postupnosti svojich príkazov. Avšak Ray urobil aj niečo navyše - pri kopírovaní vírusu pridal s určitou pravdepodobnosťou náhodné preklopenie niekoľkých bitov, čo odpovedá mutácii v prírode. Základný súbor strojových príkazov je vhodne vybraný tak, že pri akejkoľvek mutácii zostane organizmus vykonávateľným programom. Organizmy sa teda môžu vyvíjať pomocou genetických operátorov. Každému ”vírusu" bol riadiacim operačným systémom pridelený určitý strojový čas na beh, a teda rozmnožovanie. Na druhú stranu Ray pridal vek, takže staršie vírusy boli operačným systémom odstránené s väčšou pravdepodobnosťou. V Tierre nie je presne definovaná účelová funkcia. Vírusy súťažia o "prírodné zdroje" svojho prostredia, teda o čas CPU a pamäť. Tierra sa tak stáva umelým prostredím, v ktorom sa organizmy riadia vlastnými zákonmi. Organizmy, ktorým sa nejakým spôsobom podarí zabrať väčšinu týchto zdrojov, majú vyššiu schopnosť prežitia. Keď sa pamäť zaplní organizmami, operačný systém ”zabíja” najmenej schopné organizmy, teda napríklad tie, ktorých kód v nejakom kontexte generuje chybu. "Svet" bol do tej miery umelo vytvorený, že pokiaľ si vírus napr. vytvoril nekonečnú slučku, bol umelo zastavený. Čoskoro sa v programoch objavili lepšie "vírusy", ako bol schopný sám Ray zostrojiť, a dokonca sa objavil aj "parazitizmus" - vírus bol krátky a využíval časť kódu iného vírusu. Žiaľ, implementácia vyšších funkcií ako inteligencia, vnímanie a komunikácia je v organizmoch Tierry spojená s ťažkosťami. Získanie týchto vlastností prirodzeným vývojom Tierry by trvalo veky. Zatiaľ sa nepokročilo významne ani v programovej verzii viacbunkových organizmov. Roku 1996 Andrew Pargellis vytvoril podobný virtuálny svet s ešte obmedzenejším počtom inštrukcií, v ktorom sa vytvoril sebareplikujúci sa "vírus" sám, bez nutnosti vytvoriť prvý vírus "božským" zásahom človeka. Podobný systém ako Tierra je Avida vytvorená Christophom Adamim na dvojrozmernej mriežke s interakciami susedov.

Moderným teóriám vzniku života predchádzal pokus vtedajšieho chicagského študenta chémie **Stanleyho Millera** r. 1953, ktorý jednoducho zatvoril do 5-litrovej fľaše vodu, metán, amoniak a vodík, fľašu týždeň zahrieval a nechal vo vnútri generovať elektrické iskry simulujúce blesky. Výsledkom bola zmes komplikovaných zlúčenín aminokyselín potrebných na vznik života. Nemecký biofyzikálny chemik **Manfred Eigen** spolu s teoretickým chemikom **Petrom Schusterom** potom vytvorili **teóriu hypercyklov** vysvetľujúcu vznik života pomocou siete replikujúcich sa molekúl RNA. Problémom tu je, že RNA dosť ťažko môže vytvoriť svoju kópiu bez pomoci enzýmov, ktoré by mali byť produkované pomocou RNA - a RNA sama sa normálne vytvára prepisom z DNA.

Trocha pozmenenú teóriu vzniku života vytvoril jeden zo zriedkavých špičkových biológov, ktorí sa zaoberajú umelým životom, **Stuart Kaufmann** zo Santa Fe inštitútu, študujúci procesy **samoorganizácie sietí** spôsobené komplexným prepojením lokálnych

reakcií. Tento prístup sa dá použiť na ”zapínanie” a ”vypínanie” génov pri diferenciácii buniek, na správanie imúnneho systému, na evolučný proces, ale predovšetkým na modelovanie vzniku života z hypercyklov reakcií jednoduchých aminokyselín a iných zlúčenín prítomných v ”prebiotickej polievke”. Z týchto jednoduchých zlúčenín potom vznikajú navzájom sa spájajúce a katalyzujúce polyméry, ktoré vytvoria stabilnú ”sieť reakcií”. Nemalo by teda ísť iba o zlúčeniny RNA.

Podobné zjednodušené príklady napodobujúce princípy chemických reakcií dali vzniknúť aj samostatnému odboru v rámci umelého života, ktorý sa volá umelá alebo **algoritmická chémia**. Umelá chémia je definovaná súborom objektov - "molekúl" a súborom pravidiel - reakcií určujúcich spôsob interakcie týchto "molekúl", spolu s popisom aplikácie týchto pravidiel definujúcich dynamiku "populácie molekúl". Pomocou takého systému sa dajú modelovať nielen chemické reakcie, ale napríklad aj správanie sa robota riadeného symbolicky formulovaným ekvivalentom difúzie. Na rozdiel od celulárnych automatov "molekuly" nemusia byť na mriežke a počet objektov (ktorý by inak odpovedal stavom bunky) nemusí byť konečný. Objektmi môžu byť napr. čísla, binárne a znakové reťazce a iné dátové štruktúry. Reakcie potom môžu predstavovať jednoduché aritmetické operácie, maticovú manipuláciu, zlučovanie reťazcov, spájanie do boolovských sietí alebo prepájanie obvodov v procesoroch. Dynamika sa potom môže riadiť napr. explicitnými simuláciami kolízií, obyčajnými diferenciálnymi rovnicami alebo celulárnym automatom.

Jednoduchým príkladom ilustrujúcim skutočnosť, že umelá chémia môže mať so skutočnou chémiou len veľmi málo spoločného, je konštruktívna "čísla deliaca chémia", produkujúca populáciu prvočísel z náhodnej počiatočnej populácie kladných celých čísel. Objekty sú v nej prirodzené čísla okrem jednotky, $S=\{2,3,...\}$. Reakčným pravidlom je $s_1 + s_2 \rightarrow s_1 + s_3$, kde $s_3 = s_2 / s_1$ pre s_2 mod $s_1 = 0$ a $s_2 \neq s_1$, v opačnom prípade $s_3 = s_2$; toto reakčné pravidlo vyjadruje funkcia reakcia(s_1,s_2) dávajúca výsledné s_3. Proces začína náhodným generovaním populácie P [1], P [2], ..., P [M] čísel z množiny S. Potom nasleduje cyklus daný v algoritme 8.1. Funkcia náhodnéCeléČíslo(1,M) produkuje náhodné celé čísla od 1 po M.

```
dokia• nie je splnená podmienka_ukon•enia_cyklu opakuj s•u•ku
za•iatok s•u•ky
   i1:=náhodnéCelé•íslo(1,M);
   s1:=P[i1];
   i2:= náhodnéCelé•íslo(1,M);
   s2:=P[i2];
   s3:=reakcia(s1, s2);
   ak s3≠0 potom P[i2]:= s3;
koniec s•u•ky;
```

Algoritmus 8.1. Konštrukcia prvočísel pomocou konštruktívnej čísla-deliacej chémie. Pre veľkosti populácie M =100, počet cyklov cez 5000 a prvú populáciu P generovanú z obmedzeného súboru čísel od 2 do 100 skonverguje k populácii obsahujúcej takmer iba prvočísla.

Má umelý život niečo spoločné s robotmi? **Plánovanie činnosti robotov** bolo jednou z ústredných tém klasickej umelej inteligencie založenej na logickom rozbore situácie a modelu okolia v „mozgu“ robota. Problémom bolo, že cez veľkú snahu takto riadené roboty neboli schopné sa vyporiadať so zložitosťou reálneho prostredia. Aj keď dlhodobé plánovanie stále asi bude doménou klasického logického rozboru situácie, rýchle reakcie na zmeny v okolí u mnohých moderných robotov sú založené na neurónových sieťach, prípadne fuzzy logike vyvíjanej pomocou evolučných algoritmov. Rýchla reakcia robota je tu založená na vneme, ktorý sa priamo prevedie na akciu, čo sa volá **reaktívne správanie**. To je podobné

našim reakciám – keď sa spálime, nemusíme rozmýšľať nad tým, či odtrhneme ruku, robíme to automaticky.

Prvým, kto sa takýmto prístupom zaoberal, bol Walter v Neurologickom inštitúte v Bristole v na prelomu 40tych a 50tych rokov. Jeho primitívne roboty vybavené iba zopár senzormi (napr. fotobunkou) a riadené niekoľko neurónmi vykazovali zložité správanie.

Ďalším, kto na Waltera nadviazal, bol až **Brooks**, ktorý v MIT propagoval tzv. subsumpčnú architektúru, kedy moduly nižšej úrovne reagujú na okamžité prekážky a na vyššej úrovni sa snažia splniť vytýčené ciele. Moduly na rozdiel od klasickej UI pracujú asynchrónne a nezávisle, iba občas sa základné príkazy z vyššej úrovne dostanú na nižšiu alebo potlačia príkaz nižšej úrovne. Neexistuje tu centrálne riadenie. Jednou z „aplikácií“ bol robot, schopný potulovať sa po miestnosti a zbierať plechovky od nápojov. Hlavným zdrojom Brooksovej inšpirácie je hmyz, roboty takejto inteligenčnej úrovne by sa dali použiť napr. na prieskum Mesiaca alebo Marsu.

V robotike sa pokladajú za budúcnosť mikroroboty - rýchle, lacné a neriadené. Využitie: planetárny výskum, baníctvo a žatva, výstavba "na diaľku". Na rozdiel od klasického prístupu je adaptívna kontrola pohybu robota schopného sa učiť v premenlivom prostredí typickou praktickou aplikáciou “umelého života”.

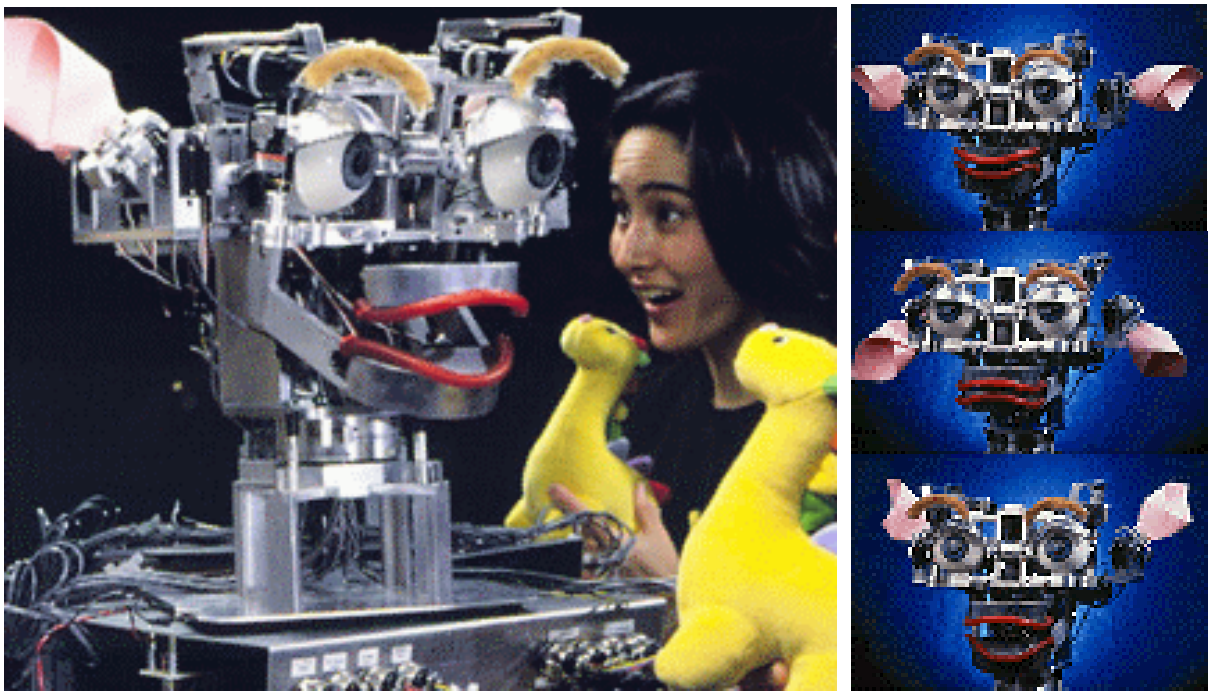
Laboratórium Rodneyho Brooka pre výskum robotov stanovilo tieto zásady:

1. Začať s jednoduchými úlohami.
2. Naučiť sa ich robiť bezchybne.
3. Pridávať ďalšiu úroveň riadenia nad výsledkami jednoduchých úloh.
4. Nemeniť jednoduché základné veci, vyššie úrovne môžu maximálne potlačiť výstup nižších vrstiev.
5. Naučiť novú vrstvu vykonávať veci tak dobre, ako to len bude možné.
6. Opakovať dookola.

Zásadou je postupné budovanie zložitého systému, nie budovanie zložitého systému naraz. Príkladom je napríklad ľudský mozog; prvotné sú základné funkcie ako riadenie dýchania a potom možno pridať funkcie vyššej úrovne ako myslenie. Riadenie musí byť čo najviac decentralizované a komunikácia musí obsahovať iba najnutnejšie príkazy - podobne ako je to pri objektovo orientovanom programovaní.

Napr. vozík Pathfinder skúmajúce r. 1997 Mars bol čiastočne autonómny a sondy typu DEEP SPACE vypúšťané od r. 1999 sú ovládané iba základnými niekoľko príkazmi, namiesto niekoľkostočlenného pozemského personálu ovládajúceho „normálne“ vesmírne sondy.

Ďalším cieľom je schopnosť robota vytvárať **sociálne interakcie s človekom**. Takýto robot potrebuje tak rozpoznávať, ako aj „cítiť“ **emócie** a správať sa emotívne. Príkladom je robot Brooksova laboratória Kismet (pozri obr. 8.7), schopný udržiavať vizuálny kontakt a rozoznávať hlasové prejavy a ich emocionálne zafarbenie. Emócie sú modelované z hľadiska psychologickej funkčnosti v modeloch a motorický systém potom ovláda výraz tváre a „gestá“.



Obrázok 8.7. Cynthia Brazeal z MIT: Interakcie s robotom Kismet, vyjadrenie: radosť, smútok, prekvapenie.

Kolektívne správanie robotov a evolučnú robotiku potom robia Maja Matarič na MIT a Dareo Floreano v taliansku, v súčasnosti ale stále ešte ide o základný výskum vzdialený od aplikácií.

Zhrnutie

Umelý život (Artificial life) využíva počítačových simulácií a matematických princípov spracovania informácie prevzatých z prírody na štúdium života vo všeobecnosti, s dôrazom na štúdium pozemských foriem života. Snaží sa vysvetliť prejavy a vlastnosti živých organizmov, od biochemického metabolizmu až po koevolúciu stratégií správania sa, študuje aj abstraktné vlastnosti života ako takého ("život aký by mohol byť"). Na rozdiel od umelej inteligencie je stavaný "zdola nahor", zaujímavé dynamické vlastnosti celkov "vyššej úrovne" podobné prejavom živých organizmov by mali emergovať z jednoduchých pravidiel interagujúcich na "nízkej úrovni". Princípy umelého života nachádzajú aplikácie od animácie pohybu umelých vtákov alebo iných živočíchov vo filmoch po štúdium vzniku života na Zemi alebo pri riadení sond vo vesmíre.

Úlohy

Úloha 8.1. Vymyslite aplikáciu, kde by sa dala použiť simulácia "boidov" v plánovaní.

Úloha 8.2. Navrhnite architektúru neurónovej siete a kódovaní jej výstupu tak, aby sa ňou riadený živočích mohol posúvať na šachovnici nielen doľava, doprava a hore a dole, ale aby mohol ísť tiež po uhlopriečkach. Aké tvary polí (okrem štvorcového u šachovnice a

obdĺžnikového) sú ešte prípustné, aby pokrývali rovnomerne plochu? Ako by ste rozkódovali výstup siete na príkazy pre pohyb živočícha?

Úloha 8.3. Skúste vytvoriť pseudokód pre modelovanie boidov, aby sa vyhýbali kolízii s okolitými "vtákmi", prispôsobili rýchlosť okolitým "vtákom", snažili sa držať sa blízko okolitých "vtákov - všetky tieto tri vlastnosti vyjadrite pomocou funkcií, ktoré budú mať ako vstup pozíciu vlastnú aj okolitých vtákov v danom momente a v minulej iterácii - na výpočet rýchlosti a ako výstup ďalšiu vlastnú pozíciu do budúcej iterácie.

Úloha 8.4. V umelej chémii použite reakciu $s_3 := (s_1 + s_2)$ modulo 2^{32} pre 10000 molekúl. Spočiatku populáciu tvorí iba 10 rôznych druhov náhodne vygenerovaných celých čísel. Vyrobte graf pomeru počtu rôznych čísel k celkovej veľkosti populácie čísel v závislosti na počtu generácií (generácia je rovná počtu reakcií rovnému počtu molekúl). Výpočet môžete urobiť v ľubovoľnom programovacom jazyku včítane príkazov v rámci MS Excelu.

Otázky na opakovanie

Otázka 8.1. Ako sa dá definovať život?

Otázka 8.2. Aké sú dva základné prístupy k umelému životu?

Otázka 8.3. Aké sú výhody a nevýhody živých systémov?

Otázka 8.4. Aké sú aplikácie umelého života?

Otázka 8.5. Čím sa líši umelá inteligencia od umelého života? V ktorých typoch programov sa stretávajú tieto dva prístupy?

Otázka 8.6. Čo sú to "prírodné zdroje" v systéme Tierra?

Otázka 8.7. Aké sú základné stavebné prvky umelej chémie?

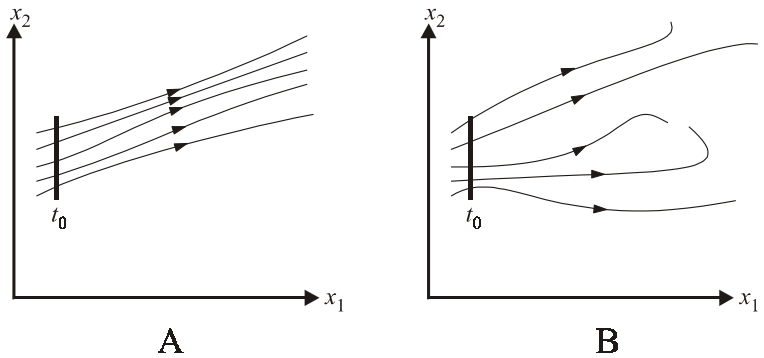
Otázka 8.8. Čo je to reaktívne správanie a subsumpčná architektúra?

Otázka 8.9. Kde sú systémy umelého života využívané vo výskume vesmíru?

Otázka 8.10. Existujú programy rozpoznávajúce emočné zafarbenie hlasu?

9 Chaos, celulárne automaty a fraktály

Deterministický chaos je prírodovedný fenomén, kde existujú zdanlivo chaotické zmeny generované nelineárnymi systémami podľa zákonov dynamiky. Je dôležité zdôrazniť, že chaotické správanie v tomto zmysle nevyplýva z externých zdrojov "šumu" alebo z neurčitosti z kvantovej fyziky. Namiesto toho je zdrojom nepravidelností prudký nárast rozdielov pôvodne blízkych stavov, ktorý je presne určený rovnicami. Táto citlivosť na počiatočné podmienky sa niekedy volá "motýli efekt"(pozri obr. 9.1); môžeme si predstaviť, že zatrepotanie motýlích krídel v Brazílii môže mať o mesiac za dôsledok tornádo v Texase. Praktickým dôsledkom existencie chaosu je obtiažnosť akejkoľvek dlhodobej predpovedi dynamického systému, ktorého počiatočné podmienky sa v praxi dajú určiť len s obmedzenou presnosťou a ktorých chyba narastá exponenciálne rýchlo.

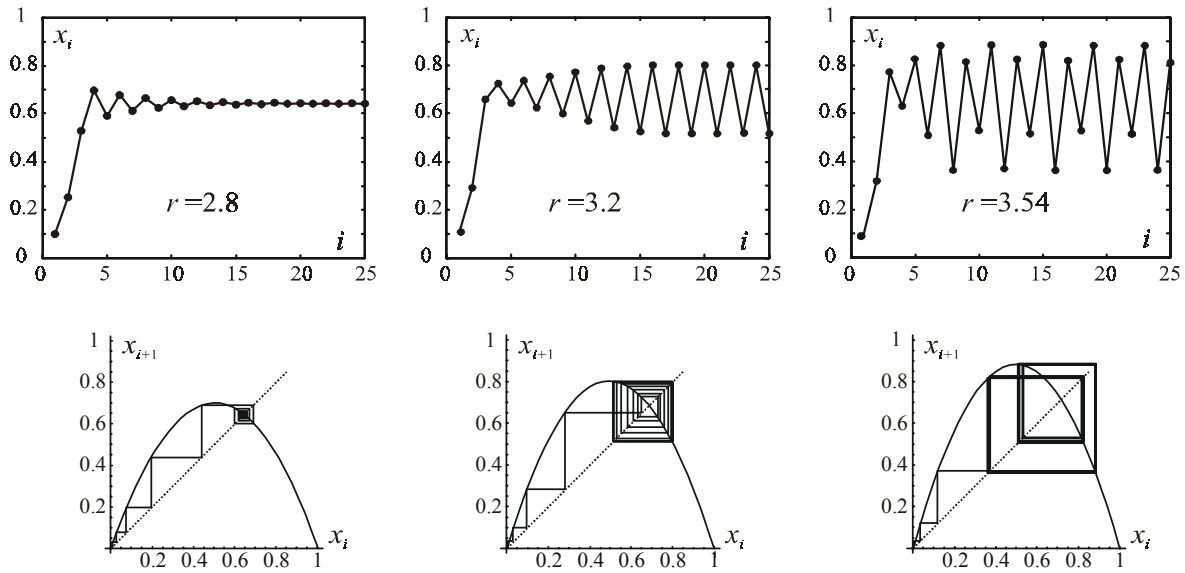


Obrázok 9.1. (A) Trajektórie dynamického systému nie sú veľmi citlivé na malé zmeny počiatočných podmienok. (B) Trajektórie dynamického systému je veľmi citlivá na zmeny počiatočného stavu. V tomto prípade, malé zmeny v trajektórii v počiatočnom čase t_0 majú veľký dopad na koncové stavy trajektórii. Táto vlastnosť dynamického systému citlivosti na počiatočné stavy trajektórií môže viesť k fenoménu nazývanom „deterministický chaos“, kde zdanlivo malé až zanedbateľné zmeny v trajektórii dynamického systému môže spôsobiť veľké zmeny v nasledujúcich častiach trajektórie.

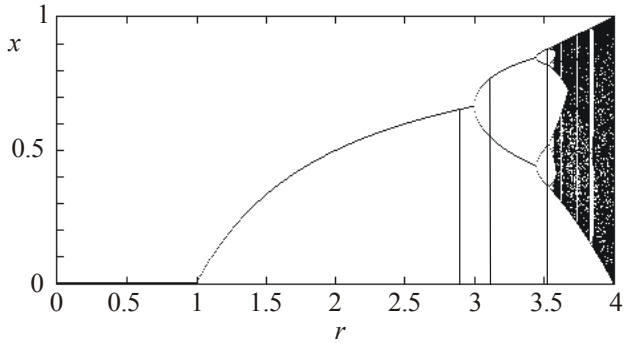
Prvé náznaky, že nie je niečo v poriadku s našimi modelmi prírody, sa objavili už koncom 19 storočia. Vtedy klasická mechanika (a dynamika vôbec) začala mať vážne problémy. Zatiaľ čo pohyb sústavy dvoch hmotných telies (napr. Slnka a Jupitera) sa dal vypočítať analyticky (teda na ľubovoľný čas dopredu dosadením do jedného vzorčeka), už výpočet iba 3 podobne ťažkých telies bolo dokázané, že analytické riešenie neexistuje. V riešení tohto problému sa objavili nestability, ktoré viedli k dosť nepravidelnému (teda chaotickému) pohybu, aj keď rovnice boli presne dané. Výskyt chaosu sa po čase začal študovať aj v iných súvislostiach.

Jedným z najjednoduchších príkladov deterministického chaosu je správanie sa tzv. **logistickej rovnice**. Je to tzv. diferenčná rovnica $x_{i+1}=r\ x_i\ (1-\ x_i)$, ktorej premenná x_i môže vyjadrovať napríklad veľkosť populácie v časovom okamihu i. Veľa druhov zvierat plodí potomkov iba v určitej časti roku. Preto v tomto prípade prechod z počtu zvierat x_i v jednom roku na počet zvierat x_{i+1} v druhom roku je prirodzenejší ako (diferenciálna) rovnica

popisujúci veľkosť populácie v každom časovom okamihu. Keďže nie všetky samice budú mať potomkov, ale zasa niektoré ich môžu mať viac, zväčšenie populácie bude nejakým násobkom *r* súčasnej populácie, kde *r* sa volá rýchlosť rastu alebo plodnosť ($r \in (0,4)$). Keď je populácia malá, môže rasť lineárne $x_{i+1}=r\,x_i$, člen $r\,x_i^2$ je taký malý, že ho môžeme zanedbať, ale pri veľkej populácii nastáva nedostatok zdrojov a priestoru a veľkosť populácie sa znižuje kvadraticky úmerne premnoženiu.



Obrázok 9.2. Typické behy iterácií logistickej rovnice pre rôzne hodnoty *r* a ich "pavučinové" diagramy.



Obrázok 9.3. Bifurkačný diagram logistickej rovnice ukazuje, ku ktorých hodnotám *x* pre zadané *r* logistická rovnica konverguje, alebo medzi ktorými osciluje. Z čiar vidno, že pre *r*=2.8 sa iterácie blížia k jednej hodnote, pre *r*=3.2 k dvom hodnotám a pre *r*=3.54 k štyrom hodnotám.

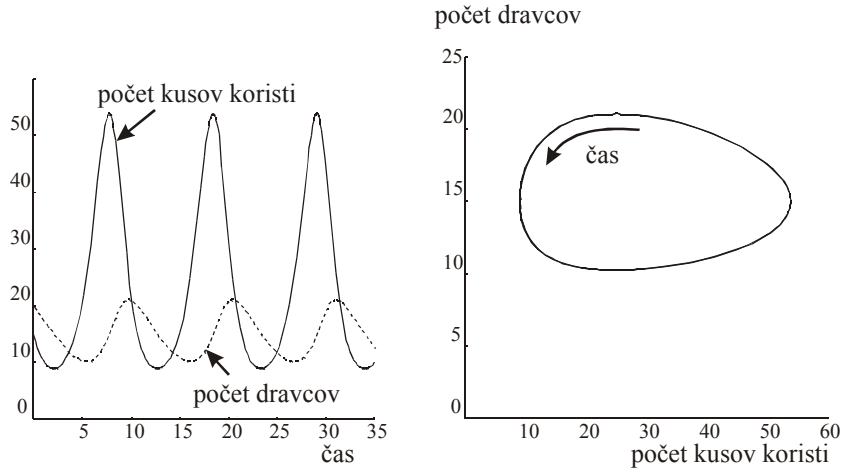
Pre x_0 rovné nule alebo jednotke dostávame v ďalšom časovom okamihu nulu a maximum ostávame pre 1/4 *r*. Prechod z x_i (=*x*) na x_{i+1} (=*y*) je vlastne rovnica paraboly ($y=r\,x - r\,x^2$), kde parameter r mení výšku paraboly, ale nie okraje. Všetky počiatočné podmienky nakoniec vedú na jeden z troch typov správania sa.

1. **Stála hodnota**: pre $0<r\leq 1$ populácia po čase vyhynie, pre $1<r\leq 2$ populácia monotónne stúpa či klesá na konečnú hodnotu a pre $2<r\leq 3$ sa populácia tiež ustáli na jednej hodnote, ale až po perióde tlmených kmitov (pozri obr. 9.2, prvý diagram).
2. **Periodické správanie sa**: veľkosť populácie alternuje medzi dvoma alebo viac stabilnými hodnotami. Pre $3 < r < 1+\sqrt{6}$ sa striedajú vždy dve hodnoty *x*, s ďalším rastom *r* sa objavujú vždy ďalšie "bifurkácie", teda zdvojovanie počtu hodnôt, medzi ktorými veľkosť populácie preskakuje (pozri obr. 9.2, druhý a tretí diagram). Pomery

za sebou idúcich rozdielov hodnôt *r*, kedy nastávajú bifurkácie, konvergujú k tzv. Feigenbaumovej konštante rovnej 4.669201.

3. **Chaotické**: veľkosť populácie postupne nadobudne každú z hodnôt medzi (0, 1) pre r=3.56994..., ale aj pre väčšie r sa ešte nájdu úzke intervaly hodnôt *r*, kedy nastávajú kmity s ustálenou periódou (pozri obr. 9.3).

Oscilácie ale môžu vznikať aj iným spôsobom. Jedným z najznámejších príkladov je populačný systém dravec - korisť reprezentovaný Lotkovou-Volterrovou diferenciálnou rovnicou[1] patriacou do ekologickej a *evolučnej dynamiky*. Ide o vyjadrenie vzťahu medzi počtom dravcov a množstvom lovnej zveri v závislosti od času. V ideálnom prípade by tieto počty mali byť stabilné, no pri určitej efektivite útokov dravcov a rozmnožovaní dravcov aj koristi nastávajú oscilácie množstva koristi, nasledované osciláciami počtu dravcov (pozri obr. 9.4). Dá sa to vysvetliť nasledovne: pri počiatočnej chorobe sú dravci zdecimovaní a korisť sa premnoží. Dravci potom majú veľa koristi, bez problémov sa môžu (po zániku choroby) rozmnožovať a ich počet sa tak zväčší ešte výraznejšie ako množstvo koristi. Teraz sú zasa premnožení dravci, ktorým sa podarí natoľko zredukovať množstvo koristi, že sami začnú umierať od hladu. Oscilácie množstva koristi a dravcov sa potom opakujú už bez nutnosti počiatočného vychýlenia počtu dravcov z rovnovážneho stavu nejakou chorobou. Takéto oscilácie možno napríklad pozorovať u počtov ulovených králikov a rysov vo viac ako storočie starých záznamoch Spoločnosti Hudsonovho zálivu, kde sú maximá počtov ulovených králikov vzdialené od seba vždy desať rokov nasledované maximami počtov ulovených rysov.



Obrázok 9.4. Prvá závislosť ukazuje oscilácie počtu dravcov a množstva koristi v čase, druhý obrázok je trajektóriou sústavy diferenciálnych rovníc - ukazuje vzájomnú závislosť počtu dravcov a množstva koristi, kde sa tieto počty pohybujú v čase stále dookola po zobrazenej trajektórii.

Podobné modely sa dajú využívať nielen v biológii, ale aj v sociológii či ekonomike, kde namiesto dravcov môžu byť firmy a namiesto koristi pre výrobu potrebné obnoviteľné zdroje.

Oscilácie sa dajú získať aj pre logistickú rovnicu pre r menšie ako 2, pokiaľ ale zavedieme do modelu náhodné čísla. Tým ale už prechádzame od deterministického chaosu k skutočnému šumu. Ku konštante *r* sa dá pri každom výpočte novej iterácie pripočítať náhodné číslo (napr. s nulovou strednou hodnotou a malou varianciou, N(0,0.1) tak, aby zmenená

[1] $dN_1(t)/dt = N_1(r_1 - b_1 N_2)$, $dN_2(t)/dt = N_2(-r_2 + b_2 N_1)$, kde $N_1(t)$ je počet kusov koristi (alebo hostiteľa) a $N_2(t)$ je počet kusov dravcov (alebo parazitov) v čase t. Za neprítomnosti dravcov sa korisť rozmnožuje rýchlosťou r_1, zatiaľ čo za absencie koristi dravci hynú rýchlosťou r_2. Schopnosti dravcov požierať korisť vyjadruje b_1 a vplyv množstva koristi (a teda nasýtenia dravcov) na ich rozmnožovanie b_2.

hodnota r bola stále medzi 0 a 4), takže nepravidelné oscilácie získavame aj pre rovnicu x_{i+1}=min(max((r+N(0,0.1)),4),0) x_i (1- x_i). Tu už ale ide o skutočne stochastický proces.

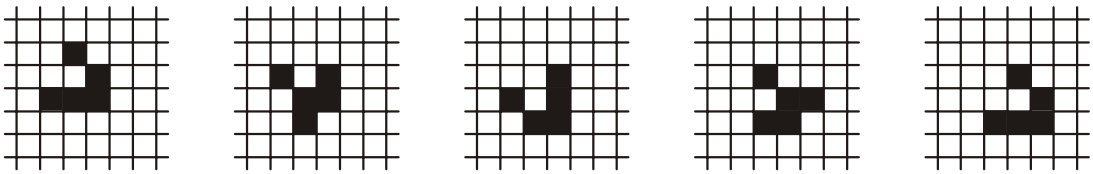
Pri uvedených modeloch populácie ale zatiaľ išlo iba o "chaos" v hodnote jednej premennej v časovej postupnosti. Deterministický chaos v priestore a v čase sa modeluje pomocou tzv. celulárnych (alebo bunkových) automatov.

Celulárne automaty sú diskrétne nelineárne dynamické systémy zložené z rovnakého typu "buniek", ktorých správanie je úplne určené ich momentálnym vlastným stavom a stavmi "buniek" v ich najbližšom okolí. Čo je považované za okolie, závisí od definície. Čas, priestor a stav systému sú diskrétne. Bunky typicky nemajú pamäť a sú rozložené na pravidelnej mriežke (jedno- či viacrozmernej, pri dvojrozmernej môže ísť o štvorcovú, trojuholníkovú alebo šesťuholníkovú mriežku). Mriežka mení svoj vzhľad po etapách. Môžeme si predstaviť, že stavy všetkých buniek sa v priebehu jedného časového kroku menia súčasne. Stav bunky v mriežke sa mení z "generácie na generáciu" pomocou pravidiel berúcich do úvahy blízke okolie bunky. Bunky majú konečný počet stavov. Stav každej bunky je charakterizovaný niekoľkými bitmi informácie; čas sa mení v diskrétnych krokoch, pričom nový stav bunky je definovaný napr. podľa "tabuľky" obsahujúcej všetky možné stavy "centrálnej" bunky a jej susedov v predchádzajúcom časovom kroku. Každému prípadu v tabuľke zodpovedá nový stav "centrálnej" bunky. Všetky bunky používajú rovnakú prechodovú funkciu.

Zákony zmien sú teda lokálne a uniformné. Na základe veľmi jednoduchých zákonov však vznikajú veľmi zložité štruktúry a vzory správania vyšších celkov (zložitých vzorov) jednoduchých buniek.

Celulárne automaty boli ako prvé zavedené von Neumannom počiatkom 50-tych rokov 20 storočia ako jednoduché modely biologickej autoreprodukcie. Celulárne automaty sú základnými modelmi komplexných systémov a procesov pozostávajúcich z veľkého množstva identických, jednoduchých, lokálne interagujúcich prvkov. Štúdium týchto systémov pritiahlo na seba v priebehu rokov značnú pozornosť pre ich schopnosť generovať bohaté spektrum veľmi zložitých vzorov správania sa zo súboru relatívne jednoduchých pravidiel. Naviac zachycujú veľa podstatných rysov komplexného samoorganizujúceho sa kooperatívneho správania pozorovaného v reálnych systémoch. Existujú ich početné aplikácie vo fyzike, biológii, chémii, biochémii, geológii, a v ďalších disciplínach. Používajú sa napríklad pri modelovaní turbulencie v kvapalinách a oscilačných chemických reakcií, pri modelovaní fibrilácií srdca, rastu rastlín a dendritckého rastu kryštálov, v ekológii, DNA evolúcii, modelovaní rozširovania infekčných chorôb, sociálnej dynamiky mestskej populácie, šírenia správ, imitačného správania, rastu populácie, vývoja mesta alebo dopravných zápch, šírenia lesných požiarov, alebo pre modelovanie biologických fenoménov, ako sú hmyzie spoločenstvá, bunkové kolónie, sietnica – vnútorný obal očnej gule, imunitný systém, v geológii na modelovanie zemetrasenia, sopiek a pretvárania zemského povrchu, v ekonomike pri fluktuáciách cien.

Evolúcia celulárnych automatov môže slúžiť aj na návrh hardvérových riešení. Tzv. *evolvér* (evolware) je systém založený na celulárnom programovaní, v ktorom sa paralelné celulárne automaty (s implementovaným genetickým algoritmom, kde nie všetky pravidlá musia byť rovnaké pre všetky bunky) používajú na riešenie výpočtových problémov. Takéto systémy sú už pokusne implementované aj hardvérovo. Výhodou takých systémov je okrem iného aj väčšia tolerancia k vstupným chybám aj k chybám pri výpočte, ktoré pri stovkách tisícok výpočtových prvkov ("mikroprocesorov") nemusia byť až také zanedbateľné.



Obrázok 9.5. Štruktúra “klzák” z hry “Life” (primitívneho celulárneho automatu). Táto štruktúra sa vždy po štyroch krokoch opakuje, pričom sa pohybuje doprava nadol. Existuje veľa takýchto objektov, ktoré vzájomnými zrážkami môžu zanikať alebo tvoriť iné druhy štruktúr. “Klzák” si možno predstaviť ako bit 1 (jeho neprítomnosť ako bit 0) a smer jeho pohybu ako drôt. Z takýchto a podobných štruktúr potom možno zostaviť celý virtuálny počítač.

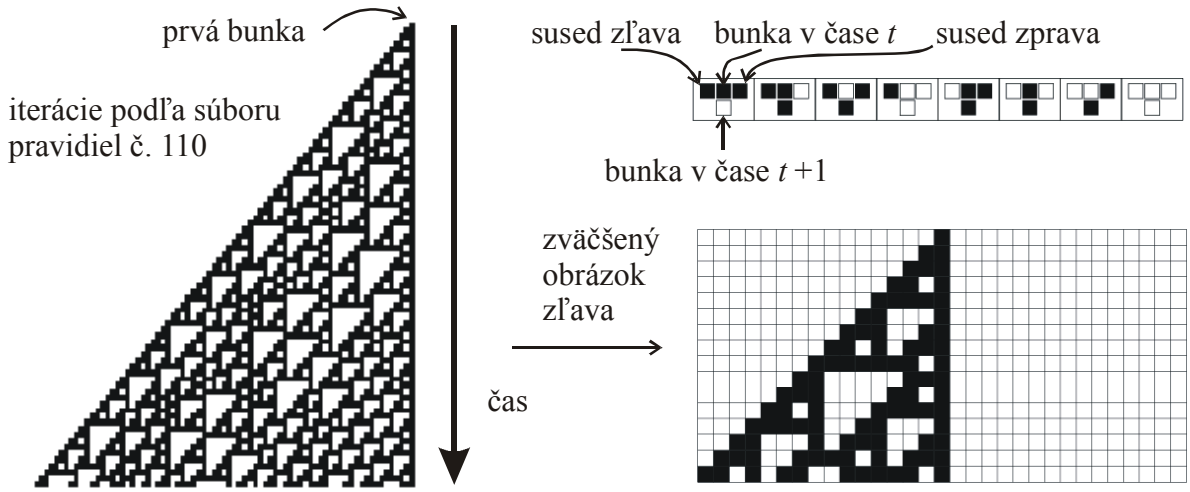
Najpopulárnejším celulárnym automatom je hra "Život" (Life), navrhnutá r. 1970 Johnom Hortonom Conwayom, mladým matematikom z Gonville and Caius College v Cambridgi. Von Neumannov automat bol síce univerzálnym počítačom (mohol simulovať akúkoľvek opísateľnú funkciu akéhokoľvek počítača pomocou súboru logických pravidiel), no bol zbytočne zložitý. Conway zjednodušil počet možných stavov bunky z von Neumannových 29 iba na 2: bunka mohla byť alebo živá, alebo mŕtva.

Conway vyskúšal veľa pravidiel pre vznik a prežitie bunky. Chcel pritom dosiahnuť, aby veľmi jednoduché vzory automaticky nerástli donekonečna, no niektoré s "divokým" a nepravidelným správaním by mohli rásť do nekonečna a mali by existovať aj stabilné štruktúry. Conwayove bunky sú rozmiestnené na štvorcovej mriežke a za susedov štvorčeka (bunky) považujeme štvorčeky nielen priamo susediace hranou, teda horný, dolný, vľavo a vpravo, no aj ďalšie štyri susediace "rohom". Hráč tu vlastne nerobí nič, iba pozoruje štruktúry, ktoré vytvára počítač.

Pravidlá:
1. STABILNÝ STAV: Keď má daná bunka dvoch živých susedov (nech už je sama živá či "mŕtva”), ostáva v rovnakom stave ako predtým aj v ďalšej generácii.
2. RAST: Keď má bunka presne troch živých susedov, bude v ďalšej generácii živá bez ohľadu na jej momentálny stav.
3. SMRŤ: Keď má bunka počet susedov 0, 1, 4-8, bude v nasledujúcej generácii "mŕtva”. Bunka teda “zomrie na podchladenie alebo na prehriatie, keď má primálo alebo priveľa susedov”.

Populácia buniek sa mení z generácie na generáciu a je možné tvoriť najrôznejšie štruktúry. Conway sa zaujímal o možnosť rastu donekonečna. Ako jednu z hypotetických možností navrhol tzv. "klzákovú pušku" (glider gun) - stabilný objekt, ktorý by "vystreľoval" vzory nazývané "klzák" (glider, viď obr. 9.5). R.W. Gosper na základe výzvy v populárnom časopise Scientific American vytvoril na počítači takúto štruktúru a tiež veľa ďalších.

Pri použití "klzákov" ako reprezentácie bitov bol Conway schopný vytvoriť ekvivalenty "and"-brán, "or"-brán a "not"-brán, sčítač, ako aj analógiu vnútornej pamäte počítača. Tak sa preukázalo, že hra Life je vlastne Turingovým počítačom, rovnako ako ním môžu byť aj iné celulárne automaty.



Obrázok 9.6. Wolframov[2] jednorozmerný celulárny automat začínajúci jedinou čiernou štvorcovou bunkou v riadku buniek. Odhora dolu sú riadky od prvej po 74 iteráciu. Prepisovacie pravidlá buniek pri prechodu od jedného diskrétneho času k druhému sú uvedené hore napravo.

Zaujímavé správanie je ale možné pozorovať aj u veľmi jednoduchých jednorozmerných celulárnych automatov, ktorých druhý rozmer pri zobrazení v dvojrozmernej mriežke je čas. Zložité obrazce pripomínajúce vzory na morských mušliach sú napríklad na obr. 9.6, kde je zakódovaný jednorozmerný automat pravidlami určujúcimi stav bunky v ďalšej iterácii momentálnym stavom bunky a jej susedov zľava a sprava. Pravidlo 110 je dekadickým zakódovaním binárneho čísla 01101110 určujúceho pravidlá premeny, nula je biela, jednotka čierna bunka, horná trojica stavu bunky a jej susedov zľava a sprava ide pritom od 111 po 000, ako vidno na obrázku vpravo hore. Aj keď je to neuveriteľné, takýto celulárny automat je tiež univerzálny, teda môže emulovať výpočty Turingovho stroja a tým aj ľubovoľný výpočet na ktoromkoľvek počítači.

Celulárne systémy sú schopné vykazovať stabilitu, cyklické správanie aj chaos. Kedy u takýchto systémov začne prevažovať dynamika informácie vykazujúca komplexné správanie sa?

Treba zdôrazniť, že oblasť komplexného správania je v skutočnosti veľmi malá v porovnaní s množstvom chaoticky sa správajúcich automatov, alebo s periodicky sa opakujúcimi stavmi. (Treba tiež zdôrazniť, že vzhľadom na konečnosť pravidiel a automatu sa každý automat správa periodicky, otázka je iba, či sa rovnaký stav opakuje hneď v ďalšej iterácii, za desať iterácií alebo za milión iterácií. Zaujímavé štruktúry sú také, kde aj keď nepoznáme pravidlá a poznáme iba momentálny stav, sme schopní odhadnúť budúci krok s nenáhodnou, no nie príliš veľkou pravdepodobnosťou.)

Fráza "na hrane chaosu" odkazuje na myšlienku, že veľa komplexných adaptívnych systémov[3], včítane života samotného, sa prirodzene vyvíja smerom k dynamike, ktorá je niekde medzi poriadkom a totálnym chaosom. Táto metafora naznačuje ekvivalenciu dynamiky fázového prechodu (napr. typu prechod ľad-voda) s dynamikou spracovania informácií. Podobne ako pri výpočtoch, aj pri vývoji prírodných systémov "hrana chaosu" hovorí, že biologické druhy by nemali byť v svojom vývoji príliš metodické ani príliš náhodne správajúce sa, aby sa dokázali úspešne adaptovať. Takéto druhy majú najlepšiu

[2] Stephen Wolfram, fyzik a informatik, je autorom programového systému Mathematica, schopného vykonávať symbolické matematické operácie ako derivácie, integrovanie apod.

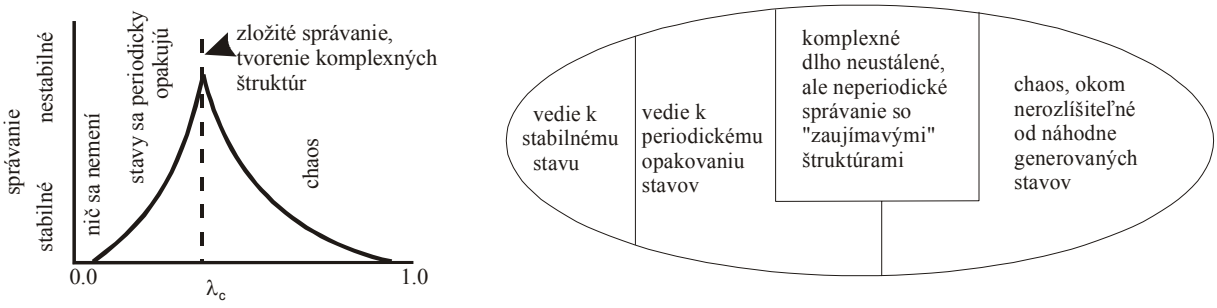
[3] Tento pojem zaviedol americký informatik Holland [XX] pri štúdiu vlastností zložitých biologických a ekonomických systémov. Ich základná vlastnosť je, že napriek intenzívnej výmene hmoty, energie a informácie s okolím sú stabilné a dokážu sa adaptovať na zmenené okolie.

schopnosť sa vyvíjať, a pritom neskĺznuť do "anarchie", ktorou je v termodynamickom zmysle smrť.

Pojem emergencie sa potom týka "vynárania sa" vlastností a správania sa systému na vyššej úrovni. Tieto vlastnosti majú jednoznačne pôvod v interakciach prvkov systému, ale pritom z nich nie sú priamo odvoditeľné. Vo vzťahu k obrázku 9.6 to znamená, že emergujú väčšie či menšie trojuholníky, ale normálny človek to dopredu neodhadne iba na základe daných ôsmich pravidiel, rovnako ako z vlastností molekúl kyslíku a vodíku neodvodí existenciu tornád, zo zrnka piesku existenciu lavín, zo semena nemožno odhadnúť konečnú podobu rastliny či z jednotlivého neurónu existenciu vedomia. Emergentné správanie sa je typicky nové a neočakávané. Najrýchlejším spôsobom ako zistiť "výstup" semena je nechať ho vyklíčiť.

Christopher Langton považoval niektoré z celulárnych automatov za "nudné", pretože po niekoľkých generáciách všetky bunky vymreli, alebo sa usporiadali do jednoduchých pravidelných štruktúr - boli vysoko usporiadané. Iné automaty zasa boli príliš chaotické a nepredpovedateľné, nedali sa rozpoznať od náhodne vygenerovaného šumu. Niektoré celulárne automaty však vykazovali zaujímavé komplikované správanie sa na hranici medzi poriadkom a úplným chaosom. Langton definoval parameter nazvaný lambda, pomocou ktorého sa dá predpovedať, či daný celulárny automat bude usporiadaný, chaotický alebo na hranici medzi poriadkom a chaosom.

Nech je ďalšie správanie bunky určené funkciou so vstupom stavov buniek v okolí s počtom buniek N (včítane bunky samotnej) a výstupom funkcie je nový stav bunky. Nech celkovo existuje K možných stavov bunky. Vyberme si jeden stav bunky, ktorý nazvime "neaktívny". Nech počet pravidiel funkcie, ktoré vedú k tomuto neaktívnemu stavu, je n. Nech ostatných $K^N - n$ prechodových pravidiel funkcie je vybraných náhodne a uniformne zo zvyšných $K - 1$ stavov bunky. Potom je parameter lambda vyjadrený ako $\lambda=(K^N-n)/K^N$. Keď je $n = K^N$, lambda je 0, všetky bunky okamžite vyhynú. Pre $n = 0$ je lambda rovné 1 a žiadna bunka nehynie. Keď sú prechody na všetky stavy v pravidlách funkcie rozložené rovnomerne, potom $\lambda=1.0 - 1/K$. Medzi lambdou rovnou nule (usporiadaným "mŕtvym" stavom) a rovnomerne rozloženými stavmi v pravidlách (chaosom) leží oblasť zaujímavého správania - na hranici chaosu.

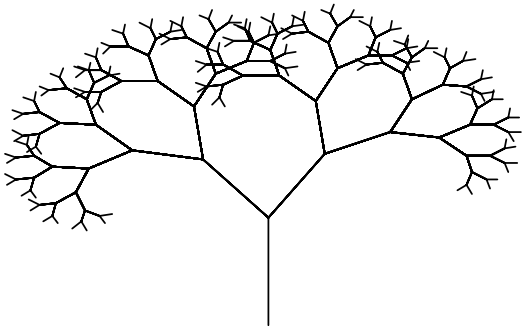


Obrázok 9.7. Schematické znázornenie závislosti komplexnosti od parametra λ v priestore celulárnych automatov, ukazujúce vzájomný vzťah medzi Wolframovými triedami (idúcimi od nemenného stavu cez periodicitu a komplexné správanie po chaos) a fázovým priechodom. Vedľa je schematické znázornenie priestoru pravidiel celulárnych automatov. Treba poznamenať, že "meradlo stability" celulárneho automatu pred a za fázovým priechodom označeným λ_c sa líši spôsobom merania.

Parameter lambda ale dobre opisuje zmeny dynamických režimov iba pre väčšie K a N. Na obr. 9.7 je na osi y "čas na dosiahnutie stabilného stavu", ale stabilný stav je rôzne definovaný pre hodnoty pred a za fázovým prechodom. Pred fázovým prechodom je dosiahnutie stabilného stavu definované tak, že všetky bunky sú v stave, ako boli už v niektorom z predchádzajúcich časových odsekov, a teda sa začína situácia periodicky

opakovať (s väčšou či menšou periódou). Naproti tomu pre lambda po fázovom prechode je stabilný stav štatisticky vyjadrený pomocou histórie tak, že priemerná "hustota" obsadenia buniek sa usadila tak, že sa nelíši viac ako o 1 percento od svojho dlhodobého priemeru.

Ďalším príkladom komplexného systému vznikajúceho z jednoduchých pravidiel sú po celulárnych automatoch aj **fraktály**. To sú geometrické tvary, ktoré majú veľmi komplikovanú štruktúru s nekonečne podrobnými detailmi. Keď si zväčšíte nejakú sekciu fraktálu, vidíte rovnaké množstvo detailov ako u celého fraktálu. Fraktály sú rekurzívne definované a ich malé časti sú podobné veľkým častiam (hovorí sa o tzv. sebapodobnosti).



Obrázok 9.8. Lindenmayerov systém, počiatočný symbol je F a produkčné pravidlo $F \rightarrow F\ [+F][-F]$, interpretovaný je ďalej uvedenými pravidlami a rovnicami, s rekurziou do úrovne 7.

Jednými z najpopulárnejších fraktálových štruktúr, príbuzných aj bezkontextovej gramatike preberanej v kapitole 6 su tzv. **Lindenmayerove systémy** produkčných pravidiel používaných na modelovanie rastu a rozvoja organizmov. Tieto systémy sa vyskytujú v mnohých simulátoroch a tiež v počítačovej animácii. Pri simulácii týchto systémov zanedbávame vnútornú štruktúru skúmaných systémov a nahrádzame ju zjednodušenými pravidlami tak, aby výsledný efekt bol čo najbližší skúmanému biologickému efektu.

Príkladom je tvorba "kríka" (pozri obr. 9.8), kde je L-systém daný nasledovnými pravidlami:

Počiatočný bod F; Produkčné pravidlo $F \rightarrow F\ [+F][-F]$

F znamená nakresli čiaru (začíname kolmou čiarou z počiatočného bodu so súradnicami 0,0 do koncového bodu so súradnicami 0,10)

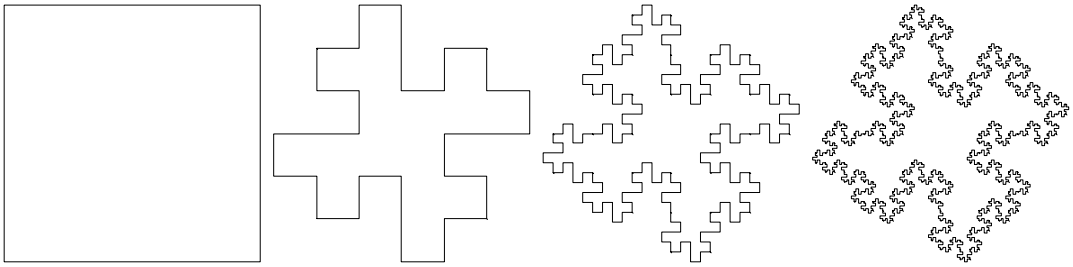
[znamená zapamätať si súradnice koncového bodu a zodpovedajúcu zmenu súradníc $dx = x_{koncový} - x_{počiatočný}$, $dy = y_{koncový} - y_{počiatočný}$

] znamená vrátiť sa na zapamätané súradnice bodu a zodpovedajúce dx a dy

+ znamená otočiť sa doľava; použili sme prepis pre súradnice nového bodu $x_{koncový} = x_{počiatočný} + 0.7(\cos(40^o)dx - \sin(40^o)dy)$; $y_{koncový} = y_{počiatočný} + 0.7(\sin(40^o)dx + \cos(40^o)dy)$

– znamená otočiť sa doprava; použili sme prepis pre súradnice nového bodu $x_{koncový} = x_{počiatočný} + 0.7(\cos(33^o)dx + \sin(33^o)dy)$; $y_{koncový} = y_{počiatočný} + 0.7(\sin(33^o)dx - \cos(33^o)dy)$

Na obrázku 9.8 sme použili rekurziu pravidiel do úrovne 7. Uhol otáčania 40^o doľava v a 33^o doprava vo vyššie uvedených rovniciach bol zvolený na začiatku náhodne. Fraktálovú štruktúru vytvorenú pomocou trocha pozmenených pravidiel môžeme vidieť na obr. 9.9.



Obrázok 9.9. Lindenmayerov systém pre počiatočný bod F+F+F+F, produkčné pravidlo F → F+F-F-FF+F+F-F. Uhol otáčania v hore uvedených rovniciach bude 90^o a zmenšenie sa použije iba pri rekurzii. Dostávame tzv. kvadratický Kochovej ostrov, čo je fraktálová štruktúra (môžeme ísť s rekurziou ľubovoľne hlboko, a pokiaľ dostaneme zobrazený odsek obrázku bez udanej mierky, nie je možné povedať, aký veľký je to odsek). Je to podobné ako pri predchádzajúcom strome, pokiaľ dostaneme obrázok odrezaného konca vetvičky bez mierky, nie je možné povedať, či rekurzia na obr. 9.8 išla do 7. úrovne alebo do miliontej.

Fraktály sa typicky používajú na modelovanie prírodných objektov, napr. fraktálové scenérie a krajiny sa objavujú často v sci-fi filmoch. Fraktály ale majú aplikácie aj v biológii na modelovanie štruktúr a javov v živých organizmoch, ako je sieť ciev a kapilár v ľudskom tele. Tá je taká hustá, že žiadna bunka nie je od nej vzdialená viac ako 3-4 bunky (počet kapilár je asi 10 miliárd), ale sieť nezaberá viac ako 5% objemu tela. Fraktálmi sa dá modelovať aj systém alveol (vzduchových komôrok) v ľudských pľúcach, ktorých je asi 300 000 000 a ktoré spôsobujú, že 1 cm^3 pľúc odpovedá aktívnej respiračnej ploche 300 cm^2. Podobný systém je tiež vo vnútrajšku ľadvín, ktoré spracujú okolo 1800 litrov krvi za deň pomocou asi 1 000 000 základných filtračných jednotiek, alebo v srdci u Purkyněho vlákien elektrochemicky regulujúcich tep. Využitie fraktálov sa zdá byť jedným zo základných princípov morfogenézy a fraktály sa objavujú aj pri geometrickej interpretácii dynamických pochodov (podivné atraktory), čo sa dá využiť napr. pri biorytmoch na konštrukciu defibrilátorov na potlačenie infarktu. Fraktálmi sa tiež dajú popísať štruktúry a javy na nižšej, molekulárnej úrovni, napr. u proteínov alebo DNA.

Pokiaľ ide o aplikácie mimo biológie, napríklad v informatike sa fraktály používajú medzi inými u metód fraktálnej kompresie dát. Tieto metódy využívajú sebapodobných miest v súboroch, napr. v obrázkoch. Na podobnom princípe je založené aj chaotické šifrovanie, kedy príjemca kódovanej správy musí mať chaotický algoritmický kľúč.

Jedným z pojmov, ktorý priťahuje pozornosť, je koncept fraktálovej dimenzie. Na jeho vysvetlenie je potrebné pochopiť, čo vlastne máme na mysli, keď hovoríme o normálnej dimenzii. Takže prečo hovoríme, že úsečka je jednorozmerná a štvorec dvojrozmerný? Povšimnite si, že obidva tieto objekty sú sebapodobné. Úsečku môžeme rozdeliť na štyri sebapodobné intervaly, každý o rovnakej dĺžke a každý môže byť štyrikrát zväčšený na to, aby vytvoril pôvodnú úsečku. Obecne, rovnako dobre môžeme úsečku rozdeliť na N sebapodobných dielov, a každý z nich zväčšiť N-krát.

So štvorcom je to trocha iné. Ten môžeme rozdeliť na štyri sebapodobné podštvorce, ale je potrebné iba ich dvojnásobné zväčšenie na to, aby sme dostali pôvodný štvorec. Pre rozdelenie na 9 dielikov je potrebné trojnásobné zväčšenie. Obecne, štvorec môže byť rozdelený na N^2 sebapodobných kópií, z ktorých každá musí byť zväčšená iba N-krát, aby sme dostali pôvodný štvorec.

Podobne kváder môžeme rozdeliť na N^3 sebapodobných kópií, z ktorých každá musí byť zväčšená iba N-krát, aby sme dostali pôvodnú kváder.

Dimenziu teda môžeme spočítať pomocou vzorčeka

dimenzia=log(počet sebapodobných kópií)/ log(faktor zväčšenia)

Pre štvorec to je dimenzia=log(N^2)/ log(N) =2 log(N)/ log(N) =2. Pre fraktálový obrazec tzv. Sierpinského sita z obr. 9.10 je dimenzia=log(3)/ log(2) ≅1.26, čo je neceločíselná dimenzia.

Obrázok 9.10. Postupná tvorba fraktálovej štruktúry volanej Sierpinského sito. Generuje sa z rovnostranného trojuholníka, kde spojíme stredy strán, z 4 nových trojuholníkov opakujeme tento proces iba pre 3 externé. Táto štruktúra môže byť vyjadrená aj ako Lindenmayerov systém s počiatkom F+F+F a pravidlom F $\rightarrow$ F+F-F-F+F. Uhol otáčania v rovniciach podobných ako u obrázku 9.8 bude 120^o.

Keby sme zmerali plochu Sierpinského sita, ako ho postupne budujeme, bude sa zmenšovať a zmenšovať, až k nulovej hodnote. Naopak, keby sme merali obvod "Kochovej[4] ostrova", a používali na meranie menšiu a menšiu mierku, bude sa obvod zväčšovať až do nekonečna. Pritom každá "rozumná" krivka, ktorá má dimenziu jedna (môžeme ju napodobniť napr. niťou) má konečnú dĺžku. Nekonečne "kostrbaté" pobrežie "Kochovej ostrova" musí isto zaberať v rovine viac miesta ako obyčajná krivka, ale zasa menej miesta ako normálny dvojrozmerný obrazec. Definícia neceločíselnej, zobecnenej dimenzie tento náhľad odráža, pričom pre čisto jednorozmerné, dvojrozmerné a trojrozmerné objekty ich dimenziu zachováva. Fraktálové objekty, „hustejšie“ ako hladká jednorozmerná krivka, budú mať dimenziu medzi 1 a 2, podobne to platí aj pre viacrozmerné objekty. Takáto dimenzia súvisí s rôznou rýchlosťou, s akou "dĺžka týchto kriviek raste do nekonečna. Tato dimenzia sa volá fraktálna od slovného základu "frakcia", teda zlomok, a útvary, s takouto dimenziou sa od toho volajú fraktály. Tento termín navrhol matematik Benoît B. **Mandelbrot**, ktorý je tiež autorom azda najslávnejšieho fraktálu.

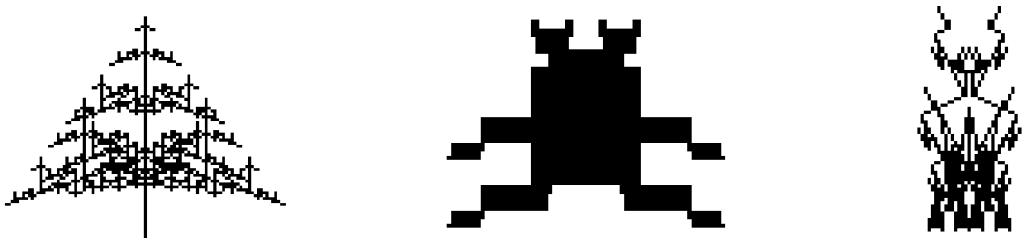
Sierpinského sito (a iné fraktály) z obr. 9.10 môžeme vytvoriť aj iným spôsobom ako bolo ukázané na obr. 9.10, a to hrou chaosu: zvolíme bod (x_0,y_0), patriaci atraktoru (v našom prípade je atraktorom[5] želaný fraktál) na neho aplikujeme náhodne zvolené pravidlo zo súboru daných lineárnych transformačných pravidiel $ax_i+by_i+e=x_{i+1}$, $cx_i+dy_i+f=y_{i+1}$, čím dostávame súradnice nového bodu (x_1,y_1). Iterácie opakujeme. Pre generovanie Sierpinského sita platia konštanty a=0.5, b=0, c=0, d=0.5, a konštanty e, f si vyberáme s rovnakou pravdepodobnosťou z dvojíc e=0, f=0; e=1, f=0; e=0.5, f=0.8660254, potom dostávame postupne vynárajúci sa obrazec z bodiek ako na obr. 9.11. Pre generovanie Barleyho paprade sú tieto konštanty rovné

a	*b*	*c*	*d*	*e*	*f*	*Pravdepodobnosť*
0.85	0.04	-0.04	0.85	0	0.3	0.85
-0.15	0.28	0.26	0.24	0	0.0825	0.07
0.2	-0.26	0.23	0.22	0	0.3	0.07
0	0	0	0.16	0	0	0.01

[4] Útvar je pomenovaný podľa jeho autorky, matematičky Helgy Kochovej, zaoberajúcej sa špeciálnymi funkciami bez derivácie r. 1906

[5] Napríklad vo fyzike je atraktor nejaký konečný stav systému, ku ktorému smeruje vývoj systému. Ako ilustratívny príklad môže slúžiť detská hra v guľôčky, kde najnižší bod jamky hrá úlohu atraktora. Niektoré dráhy hodených guľočiek v tomto bode končia, pozri obr. 10.3.

Obrázok 9.11. Postupné generovanie Sierpinskeho sita z 1000 bodov a Barleyho paprade z 100000 bodov pomocou hry chaosu.



Obrázok 9.12. Strom, žaba a samuraj, obrázky vytvorené algoritmom založeným na rekurzii a pravidlách podobných Lindenmayerovým systémom, kde však sú pravidlá zakódované chromozómami, ktoré prechádzajú drobnými zmenami. Užívateľ má napr. na výber z 8 možných obrázkov a ten, ktorý si vyberie, tvorí základ pre ďalších 8 obrázkov vytvorených pomocou náhodne pozmenených pravidiel. Po niekoľkých desiatkach cyklov výberov sa potom dajú získať zaujímavo vyzerajúce obrázky.

Netypickým prístupom spojenia Lindenmayerových systémov a fraktálov s evolučnými algoritmami je použitie estetického kritéria pri výberovej funkcii u vývoja obrázkov "bytostí". S tým začal zoológ Richard Dawkins vo svojej populárnej knižke *Slepý hodinár* [XX], "bytosti" nazval "biomorfy" podľa surrealistických obrázkov Desmonda Morrisa podobných zvieratám. Pomocou náhodných mutácií sa vždy vytvoril súbor obrázkov, z ktorých si užívateľ vybral ten, čo sa mu najviac páčil, a z toho sa generoval ďalší súbor. Tieto stromom podobné štruktúry boli zakódované vo forme génov, kódujúcich uhol vetvenia, dĺžku vetiev, hĺbku rekurzie, tvar ako priamka, elipsa, obdĺžnik a pod. Takto vznikli napr. nasledujúce obrázky ihličnatého stromu, žaby alebo samuraja (viď. obr. 9.12).

Zhrnutie

Deterministický chaos sú na prvý pohľad nepravidelné alebo "chaotické" zmeny bez rozpoznateľnej závislosti generované nelineárnymi systémami podľa zákonov dynamiky, ktoré jednoznačne určujú stav systému v ktoromkoľvek okamihu zo znalosti predchádzajúcej histórie systému. Chyba predpovedi ich stavu narastá exponenciálne rýchlo („motýli efekt“, populačná logistická rovnica), oscilácie ale dostávame aj u nechaotickej evolučnej dynamiky Lotkových-Volterových rovníc. **Celulárne automaty** sú diskrétne nelineárne dynamické systémy zložené z rovnakého typu "buniek", ktorých správanie je úplne určené ich momentálnym vlastným stavom a stavmi "buniek" v ich najbližšom okolí. Conwayova hra Life a Wolframove jednorozmerné automaty môžu modelovať aj Turingov počítač. Ich

správanie „na hrane chaosu“ je blízke životu. **Fraktály** sú sebapodobné geometrické tvary s nekonečne podrobnými detailmi vznikajúce z jednoduchých pravidiel. K nim patrí aj Lindenmayerove systémy, používané na modelovanie prírodných štruktúr.

Úlohy

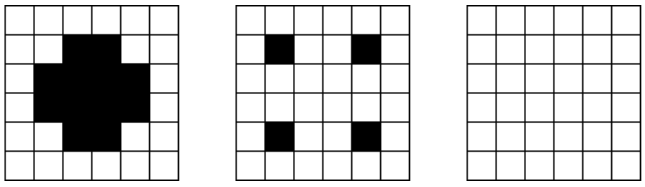
Úloha 9.1. Pre $1<r \leq 3$ vypočítajte hodnotu, na ktorej sa ustáli veľkosť populácie, ktorá sa riadi logistickou rovnicou.

Úloha 9.2. Pre $x_0=0.01$ a sadu hodnôt $r=2.9, 3.1, 3.56$ vyrobte graf závislosti veľkosti populácie (ktorá sa riadi logistickou rovnicou) na generácii pre 100 generácií.

Úloha 9.3. Pre $x_0=0.01$ a počiatočnú hodnotu $r=2.8$, ktorá je pri každej generácii menená podľa vzorca $\min(\max((r+N(0,0.1)),4),0)$, kde $N(0,0.1)$ je náhodné číslo z normálneho rozdelenia s nulovou strednou hodnotou a varianciou 0.1, vyrobte graf závislosti veľkosti populácie (ktorá sa riadi logistickou rovnicou) na generácii pre 100 generácií.

Úloha 9.4. Začnite u hry Life zo štruktúrou  a zistite, aké rôzne obrazce sa vytvorí a kedy sa hra ustáli, aký bude jej atraktor.

Úloha 9.5. Nájdite taký *najmenší* (teda používajúci najmenší počet živých buniek) netriviálny obrazec, ktorý pri hre Life kompletne zanikne už po prvej iterácii. Netriviálnosť spočíva v tom, že aspoň jedna bunka musí vymrieť z „prehriatia“, teda z toho, že má 4 alebo viac susedov. Ako príklad uvádzame obr. 9.13, kde obrazec vymiera po dvoch iteráciach.



Obrázok 9.13. Štruktúra hry Life vymierajúca už po dvoch generáciach.

Úloha 9.6. Zobrazte súbor stavov jednorozmerného celulárneho automatu, kde sú bunky iba v jednom riadku (formálne spojenom do cyklu, t.j. začiatok riadku susedí s koncom) a možné stavy buniek 0 = biela, a 1 = čierna. Prechodové pravidlá pre stav buniek berú do úvahy iba momentálny stav bunky (v prostriedku) a jej ľavého a pravého suseda. Pravidlá sú zadané takto: $1,1,1 \Rightarrow 0$; $1,1,0 \Rightarrow 0$; $1,0,1 \Rightarrow 1$; $1,0,0 \Rightarrow 0$; $0,1,1 \Rightarrow 1$; $0,1,0 \Rightarrow 1$; $0,0,1 \Rightarrow 0$; $0,0,0 \Rightarrow 1\}$. Napr. pravidlo $1,1,0 \Rightarrow 0$ znamená, že čierna bunka s čiernym ľavým a bielym pravým susedom bude v ďalšom časovom kroku biela. Zobrazte súbor riadkov v časovej postupnosti, t.j. riadok v čase 0, pod ním riadok v čase 1 atď. Počiatočný stav riadku v čase 0 má byť náhodne generovaný.

Úloha 9.7. Použite pravidlo č. 90 na generovanie jednorozmerného celulárneho automatu, ktorý je inicializovaný jedinou čiernou bunkou v prvom riadku. Ktorý fraktál vám výsledok pripomína?

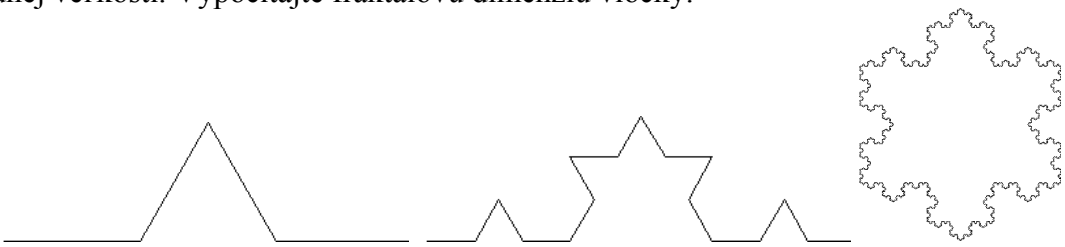
Úloha 9.8. Vypočítajte Langtonov lambda parameter pre Conwayovu hru Life. Podľa tejto hodnoty, do ktorej triedy kompexnosti patrí táto sada pravidiel.

Úloha 9.9. Pomocou limity spočítajte obvod ostrova Kochovej, kde rekurzia použitia produkčného pravidla ide do nekonečna.

Úloha 9.10. Pomocou limity spočítajte plochu Sierpinského sita, kde rekurzia použitia produkčného pravidla ide do nekonečna. Spočítajte počet trojuholníkov pri stej iterácii produkčného pravidla (nájdite iteračný vzorec).

Úloha 9.11. Pascalov trojuholník je trojuholník z čísel usporiadaných v posunutých radoch tak, že $a_{nr} = \frac{n!}{r!(n-r)!} = \binom{n}{r}$, kde posledný výraz je binomický koeficient. Trojuholník bol študovaný Blaisom Pascalom (tým, čo navrhol kalkulátor Pascaline, pozri kapitolu 1), ale tento trojuholník bol študovaný už 500 rokov predtým čínskym matematikom Yanghuim a Perzským astronómom a básnikom Omarom Chajjámom. Vytvorte aspoň 8 riadkov tohto trojuholníka a vyfarbite čiernou všetky nepárne čísla. Ktorý preberaný fraktál vám vyfarbený útvar pripomína? Ukážte na základe vlastností súčtov párnych a nepárnych čísel, že takéto farbenie je ekvivalentné pravidlu č. 90 Wolframovho jednorozmerného celulárneho automatu z úlohy 9.7.

Úloha 9.12. Roku 1906 Helga von Kochová vynašla krivku „snehová vločka“, pozri obr. 9.14. Táto sa zostrojí z rovnostranného trojuholníku vztýčením na strednej tretine každej strany trojuholník jednej tretiny pôvodnej veľkosti, a opakovaním tohoto postupu do nekonečna. Spočítajte obvod u prvých desiatich iterácií snehovej vločky. Jedna strana snehovej vločky je vyrobená z štyroch kópií seba samej, každá kópia má jednu tretinu pôvodnej veľkosti. Vypočítajte fraktálovú dimenziu vločky.



Obrázok 9.14. Prvé 2 iterácie pre jednu stranu Kochovej vločky a výsledná vločka po veľa iteráciach.

Úloha 9.13. Pomocou hry chaosu skúste zostrojiť obrázok Sierpinského sita ako je na obrázku 9.11.

Otázky na opakovanie

Otázka 9.1. Je Brownov pohyb častice pylu deterministickým chaosom?

Otázka 9.2. Môže veľkosť populácie kolísať aj bez vonkajších náhodných vplyvov a prítomnosti predátorov?

Otázka 9.3. Ktorý z priekopníkov počítačov zaviedol celulárne automaty?

Otázka 9.4. Kde sa všade používa modelovanie pomocou celulárnych automatov?

Otázka 9.5. Čo je špecialitou štruktúry klzák hry Life a jako môže byť použitá v modelu počítača?

Otázka 9.6. Čo znamenajú jednotlivé riadky u zobrazení výsledkov jednorozmerného celulárneho automatu a jako sa interpretuje číslo pravidiel u Wolframovho automatu?

Otázka 9.7. Čo je to emergencia, hrana chaosu a Langtonov lamda koeficient?

Otázka 9.8. Aká je súvislosť medzi fraktálmi a filmami?

Otázka 9.9. Ktoré štruktúry ľudského tela sa modelujú pomocou fraktálov?

Otázka 9.10. Ako je definovaná fraktálová dimenzia?

Otázka 9.11. Ktorým už v minulých kapitolách preberaným matematickým štruktúram sú príbuzné Lindenmayerove systémy?

Otázka 9.12. Aké je kritérium výberu pri optimalizácii Dawkinsových Biomorfov?

10 Simulácia sociálnych javov

Autonómny alebo adaptívny agent je typicky počítačový program (môže byť prepojený s reálnym zariadením), ktorý dostáva vstupy z okolitého komplexného dynamického prostredia a reaguje naň so zámerom splniť sadu cieľov. Medzi jeho vlastnosti patrí, že má schopnosť spracovávať vstupnú informáciu a rozhodovať sa, pričom môže predvídať budúce stavy a možnosti na základe vnútorného modelu okolia. Tieto jeho vnútorné modely môžu byť často nesprávne alebo neúplné, ako u človeka. Svojimi akciami agenti často mení celé prostredie, ktorého sú súčasťou. Ciele agenta môžu byť napríklad dosiahnutie želaných lokálnych stavov alebo globálnych konečných cieľov. Pretože hlavná časť okolia agentov pozostáva z ostatných agentov, agenti strávia veľa času vzájomným prispôsobovaním sa. Tým sú príbuzné živým systémom.

Modely sociálnych systémov zostavené z agentov odpovedajú aj na otázky, ktoré klasické modely na základe diferenciálnych rovníc nezvládnu: Ako heterogénny mikrosvet správaní indivíduí generuje globálne makroskopické pravidelné črty spoločnosti?

Podobné multiagentové systémy, ako sa používajú pre simuláciu životného prostredia (zmienené v kapitole 8 o umelom živote) sa dajú použiť aj v sociálnych vedách, napríklad v **antropológii,** kde bol podobný softvér využitý na modelovanie indiánskeho kmeňa Anasazi pre overenie teórií o živote dediny, pričom výsledky modelovania sú porovnávané s archeologickými nálezmi.

Pri štúdiu dynamiky sociálnych vzťahov sa často využívajú pojmy z biológie. Evolúcia je adaptácia zameraná na naplnenie vlastných potrieb. Koevolúcia je vo väčšej miere adaptácia na naplnenie potrieb navzájom medzi druhmi, ktoré fungujú ako svoje životné prostredie. Je napríklad proti záujmu ako dravej zveri, tak aj koristi, aby bol druhý druh vyhubený. Aj keď pre korisť na prvý pohľad vyzerá zánik nepriateľa priaznivo, dravá zver v skutočnosti väčšinou likviduje najslabšie kusy koristi a tým prispieva k zdokonaľovaniu genofondu a zabraňuje degenerácii. Podobné pojmy z biológie často môžu nájsť paralelu napr. v analýze ekonomických vzťahov.

10.1 Sociálne dilemy

Ako vyplýva z teórie hier v ekonómii, aplikovanej tiež často na analýzu pretekov v **zbrojení medzi štátmi**, zjednodušenej zo skúmania ďalej preberaného "problému väzňov" (prisoner's dilemma), kooperácia sa môže vynoriť zo sebeckého záujmu, keď je výhodná pre obidvoch. Vo všeobecnosti, keď hľadáme racionálne dôvody, či pri našej interakcii s niekým iným (s našim obchodným alebo iným partnerom) pristúpiť na kooperáciu, alebo nie, naše rozhodnutie by malo byť založené na racionálnych argumentoch a nie na predsudkoch a ideologických dôvodoch.

Medzi sociálne dilemy napríklad patrí oblasť životného prostredia (keď ho znečistím, negatívny vplyv na mňa je malý a úspora veľká, aj keď súčet negatívnych vplyvoch pre všetkých je významnejší ako moja úspora), obchodné vojny, vyjednávanie medzi krajinami a sociálne interakcie. Tu všade je výber medzi altruistickým správaním, kooperatívnym, alebo

egoistickým. Na modelovanie toho sa používajú hry, ktoré dovoľujú analýzu rôznych scenárov.

Ľudské spoločnosti predstavujú jedny z najkomplexnejších kooperatívnych systémov, porovnateľné iba so sociálnym hmyzom. Určite všetci niekedy máte pocit, že spoločnosť okolo vás je riadená v svojom rozhodovaní iba krátkodobými sebeckými cieľmi, ale v skutočnosti tomu tak asi nie je. Ako kontrapríklad si môžete predstaviť situáciu, že by človeka vysadili do volnej prírody. Ako dlho by asi prežil a ako by sa mu darilo? Z tejto predstavy môžeme usúdiť, že kooperácia je v rozvinutej spoločnosti nevyhnutná.

Je najskôr potrebné hľadať niečo podobné ako "neviditeľnú ruku tržných síl", ibaže vedúcu namiesto súťaženia k spolupráci. Tieto vplyvy najskôr nebudú určené prevážne geneticky na rozdiel od hmyzu, kde je víťazstvo kooperácie udržiavané genetickou homogénnosťou spoločenstiev - z pohľadu tzv. sebeckého génu [XX] pomoc blízkemu príbuznému znamená pomoc sebe samému. Ľudská spoločnosť je ale z tohto pohľadu príliš heterogénna, vysvetlenie spolupráce "sebeckým génom" asi nie je to pravé.

V žargónu teórie hier poznáme súťažné interakcie ako "hry s nulovým súčtom", čo jeden získa, druhý stratí. Tento model je ale pre prax príliš obmedzujúci, zdroje nie sú konštantné. Keď sebeckosť jedincov spôsobuje, že ich zdatnosť ako jednotlivcov aj ako celku sa znižuje, namiesto toho aby sa zvyšovala, ide o "hry so záporným súčtom". Zmena systému tiež môže viesť k využití nevyužívaných zdrojov, kooperácia alebo symbióza odpovedá "hre s kladným súčtom".

V tejto kapitole ukážeme na jeden z možných prístupov k vysvetleniu vzniku kooperácie, ktorý je založený simulačných výpočtoch pomocou hry nazývanej „***väzňova dilema***“.

Obrázok 10.1. Väzenská dilema ako sociálna dilema či statická[1] hra s nenulovým súčtom

Nech dvaja zlodeji boli zatknutí políciou pre vylúpenie banky *Prvá Národná Poctivá Sporiteľňa*, a boli separovane umiestnený do cely predbežného zadržania[2] (pozri obr. 10.1). Každému z páchateľom záleží o mnoho viac na tom, aby bol oslobodený on, ako na tom, aby bol oslobodený jeho spolupáchateľ. Pretože polícia nemala dostatok dôkazov k súčasnému usvedčeniu oboch páchateľov, chytrý prokurátor dal každému tento „diabolský“ návrh: „*Môžete si vybrať, buď sa priznať k lúpeži, alebo mlčať. Ak sa prizná te vy a váš komplic sa neprizná, tak potom celú vinu za lúpež dám na vášho komplica, ktorý bude odsúdený na mnohoročný trest a vy budete oslobodený. Podobne, ak vy budete mlčať a váš komplic sa prizná činu, potom on bude oslobodený a vy dostanete mnohoročný trest. Ak budete obaja mlčať a nepriznáte sa, potom navrhnem súdu, aby vás oboch odsúdil na trest, ktorý tiež bude mnohoročný, ale kratší, ako v prípade, že by sa len jeden z vás priznal a druhý mlčal. V opačnom prípade, ak budete obaja spolupracovať so mnou a priznáte sa, navrhnem vaše*

[1] statická hra je taká, keď sa všetci hráči rozhodujú súčasne bez znalosti aktuálnych ťahov ostatných hráčov

[2] Táto historka je založená na americkom práve, kde páchateľ môže vystupovať ako svedok obžaloby, pričom za túto „službu“ mu prokurátor môže sľúbiť zníženie trestu, alebo až oslobodenie.

odsúdenie, ale sľubujem vám predčasné omilosťnenie. Ak sa mienite priznať, dozorcovi nič nepovedzte, na druhý deň ráno vás znovu navštívim a môžete mi oznámiť vaše rozhodnutie“.

“Dilema”, pred ktorou stoja obaja väzni, spočíva v tom, že nech sa druhý z nich rozhodne akokoľvek, vždy je lepšie nemlčať a priznať sa, než ako nepriznať svoju vinu a prizná sa komplic, potom celá vina padne na mňa. Táto dilema ilustruje konflikt medzi individuálnou a skupinovou racionálnosťou. Skupine ktorej členovia preferujú svoj vlastný záujem obvykle skončí zle, ako tá skupina, kde jej členovia uprednostňujú skupinové záujmy. Vo všeobecnosti, ak záujmy jednotlivcov nereprezentujú záujmy celej skupiny, potom takáto skupina je viac úspešnejšia ako skupina, ktorej jednotliví členovia uprednostňujú vlastné záujmy a ciele. Tento problém bol navrhnutý a široko diskutovaný už počiatkom 50-tych rokov 20 storočia Merrilom Floodom a Melvinom Dresherom vo vtedy veľmi známej Rand Corporation inštitúcii, kde sa vedci snažili použiť teóriu hier k odhaleniu zákonitostí globálnej nukleárnej stratégii. Väzňova dilema[3] sa stala často používaným prístupom k vysvetlenie vzniku kooperácie v rôznych sociálnych, ekonomických a iných systémoch, kde sa pomocou matematickej teórie hier a/alebo počítačových simulácií hľadajú „racionálne dôvody“ k tomu, prečo a za akých okolností je výhodnejšie spolupracovať, než ako nespolupracovať.

10.2 Základné princípy hry "väzeňskej dilemy"

"Väzňská dilema" **(**VD) [XX] je hrou medzi dvoma hráčmi, kde každý z nich má výber z dvoch možností: spolupracovať (cooperate – *C*) alebo podvádzať (defect – *D*). Keď oba hráči spolupracujú, potom dostanú „odmenu“ *R* (reward), ktorá by mala byť väčšia ako bodový zisk „trest“ *P* (punishment) získaný obidvoma hráčmi, keď podvádzajú. Ale keď jeden hráč podvádza (hrá *D*) zatiaľ čo ten druhý spolupracuje (hrá C), potom podvádzajúci získa výplatu bodov *T* (temptation - pokušenia) ktorá je väčšia ako *R*, zatiaľ čo spolupracujúci dostáva počet bodov *S* (sucker – ten, čo naletel) ktorá je menšia ako *P*. Platby spĺňajú nerovnosti[4] $T > R > P > S$ a $2R>T+S$. Druhá nerovnosť znamená, že celková výplata pre dvoch hráčov je väčšia keď obaja spolupracujú v porovnaní so súčtom bodov keď jeden spolupracuje a druhý nie. Formálne môže určiť funkciu platieb $f:\{C,D\}^2 \rightarrow \{0,1,3,5\}$ ako $f(C,C)=3$, $f(C,D)=0$, $f(D,C)=5$, $f(D,D)=1$.

Uvažujme hráča – agenta *a*, jeho stratégia $S(a)$ je súbor pravidiel, ktoré použije pri výbere ťahu *C* alebo *D* (spolupráce alebo podvádzania) proti druhému hráčovi, na základe predošlej histórie hry. Pri iba jednej iterácii by samozrejme bolo výhodné zradiť pri polovičnej šanci stretnúť spolupracujúceho, no keď sa iterácie s tým istým spoluväzňom opakujú a vopred nie je známy počet iterácií, metóda ako sa zachovať nie je jasná. Hra sa opakuje t_{max}-krát (dôležitý parameter hry), na záver hry sa spočítajú platby jednotlivých hráčov. Funkcia `Stratégia(x,t)` poskytuje stratégiu hráča x(= *a*, *b*) v čase *t*. Pre jednoduchosť v našich ďalších úvahách budeme používať stratégiu, pri ktorej hráč zvolí ťah na základe predchádzajúceho ťahu súpera. Stratégia je určená pomocou 3-bitového vektora (t.j. hráči si pamätajú predchádzajúci ťah oponenta) $\boldsymbol{s}=\{s_1, s_2, s_3\} \in \{0,1\}^3$, kde jednotlivé zložky (0 – kooperácie, 1 – nekoooperácia) majú túto interpretáciu:

(1) s_1 určuje prvý ťah hráča: ak $s_1=0$, potom hráč kooperuje, v opačnom prípade, ak $s_1=1$, nekooperuje.

(2) s_2 určuje ťah hráča v prípade, že druhý hráč v predchádzajúcom kroku kooperoval: ak $s_2=0$, potom hráč kooperuje, v opačnom prípade, ak $s_2=1$, nekooperuje.

---

[3] samotný názov „väzňova dilema“ pochádza od známeho matematického ekonóma Alberta Tuckera

[4] v prípade, že platí $R>T>P>S$, ide o tzv. hru so zárukou (assurance game). Keď $R>T>S>P$, ide o tzv. hru na kurča. Princíp hry spočíva v rýchlej jazde dvoch aut oproti sebe. Kto prvý uhne, stratí tvár a je "kurča" ("chicken"= slangovo zbabelec). Podvod je v tomto prípade neuhnúť, spolupráca uhnúť. Keď žiadny neuhne, sú na tom horšie ako ten, čo jediný uhne a je označený za zbabelca.

(3) s_3 určuje ťah hráča v prípade, že druhý hráč v predchádzajúcom kroku nekooperoval: ak s_3=0, potom hráč kooperuje, v opačnom prípade, ak s_3=1, nekooperuje.

Takáto stratégia vyžaduje len krátkodobú pamäť hráčov, požaduje len znalosť posledného kroku protivníka - druhého hráča. Celkový počet rôznych 1-stratégií je 2^3=8. Keby si väzni mali pamätať viac krokov do minulosti, s každým takým krokom by exponenciálne narastala zložitosť zápisu stratégie.

Pseudo pascalovská implementácia algoritmu pre dvoch hráčov – agentov *a* a *b* je nasledovná:

```
Opakuj pre čas t:=1 po t_max slučku
Začiatok slučky
      ťah_a[t]:=Stratégia(a,t);
      ťah_b[t]:= Stratégia(b,t);
      platba[a]:=platba[a]+f(ťah_a[t], ťah_b[t]);
      platba[b]:=platba[b]+f(ťah_b[t], ťah_a[t]);
koniec slučky;
```

Ilustračný príklad 1-stratégie
Majme dve stratégie $s(a)$=(010) a $s(b)$=(101), potom nasledujúcich 5 ťahov hráčov *a* a *b* a taktiež aj jednotlivé platby na základe matice platieb (1) sú určené pomocou nasledujúcej tabuľky

ťah	agent *a*		agent *b*	
	ťah	platba	ťah	platba
1	0	0	1	5
2	0	3	0	3
3	1	5	0	0
4	1	1	1	1
5	0	0	1	5
suma:		9		14

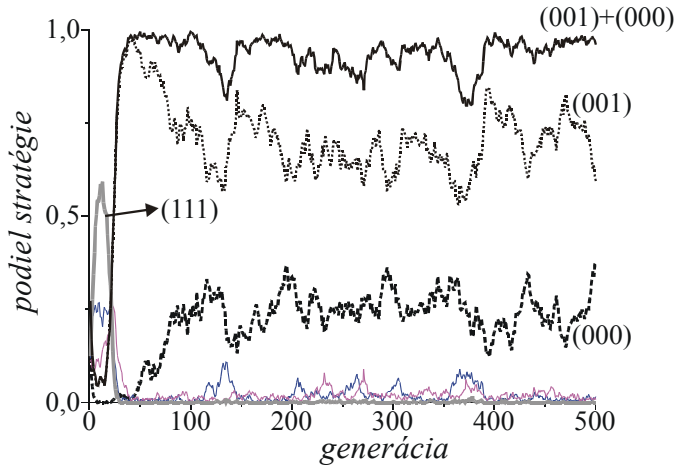
V prvom ťahu oba hráči použijú ťahy, ktoré sú určené prvou zložkou ich stratégií. V nasledujúcich ťahoch už sú determinovaní ťahom súpera v predchádzajúcom ťahu.

10.3 Simulácia emergencie stabilnej stratégie pomocou genetického algoritmu

Pri plánovaní stratégie na základe predchádzajúcich skúseností s protihráčom sa často používa genetický algoritmus. Táto aplikácia genetického algoritmu prvýkrát vznikla zo spolupráce amerického politológa Roberta Axelroda [XX] s Johnom Hollandom a rozvíjala sa ďalej pre zložitejšie modely medzinárodnej bezpečnosti odpovedajúce napr. na otázku, koľko z rozpočtu vydávať na obranu. V časovom priebehu je zaujímavé sledovať vývoj kooperatívnych a nekooperatívnych stratégií. Kooperatívne stratégie sa spočiatku ukázali menej výhodné, no v neskorších štádiách prevážili.

Základné princípy genetického algoritmu (pozri kapitolu 7) použitého k štúdiu emergencie „najvýhodnejšej“ stratégie sú rovnaké ako u klasického genetického algoritmu.. Populácia je zložená z chromozómov - stratégií. Algoritmus je inicializovaný náhodne vytvorenou populáciou stratégií. Fitnes chromozómov sa určí pomocou "megaturnaja", kde všetky možné dvojice hrajú VD hru, celková platba každého chromozómu získaná v tomto megaturnaji je fitnes. Štandardná realizácia reprodukčného procesu obsahuje selekciu dvoch stratégií na základe ich fitnes (stratégie s väčším fitnes majú väčšiu pravdepodobnosť byť vybraté do reprodukčného procesu), ktorá sa obvykle realizuje pomocou Goldbergovej rulety.

Vybraté stratégie vstupujú do vlastného procesu reprodukcie, ktorý obsahuje kríženie a mutáciu. Výsledné stratégie – potomkovia tvoria novú populáciu, ktorá keď obsahuje rovnaký počet potomkov, ako mala pôvodná populácia počet rodičov, nahradí rodičovskú populáciu populáciou potomkov. Numerické výsledky genetického algoritmu sú znázornené na obr. 10.2. Ako výsledná stratégia emergujú dve podobné stratégie (001) a (000), ktoré obe zahajajú hru kooperáciou a na kooperáciu odpovedajú kooperáciou. Líšia sa v reakcii na podvod, prvá stratégia (001) reaguje na podvod podvodom, zatiaľ čo druhá stratégia (000) „hlúpo“ reaguje na podvod kooperáciou. Stratégia (001) sa nazýva podľa Axelroda tit-for-tat (vo volnom preklade, ako ty mne, tak ja tebe). Menšinová stratégia „dobrák“ (000) má šancu prežívať iba popri stratégii tit-for-tat.[5]



Obrázok 10.2. Výsledky genetického algoritmu použitého pre emergenciu výslednej stratégie, pričom populácia stratégií bola inicializovaná náhodne. V priebehu evolúcie algoritmu spontánne emergovali dve „kooperatívne“ stratégie (001) a (000), ktoré zahajujú hru kooperáciou a na kooperáciu súpera v predchádzajúcom ťahu odpovedajú kooperáciou v nasledujúcom ťahu. V dôsledku toho, že väčšinová stratégia (001) neprodukuje „podvod“, menšinová stratégia (000) nemá príležitosť reagovať spoluprácou na podvod.

10.4 Evolučne stabilné stratégie

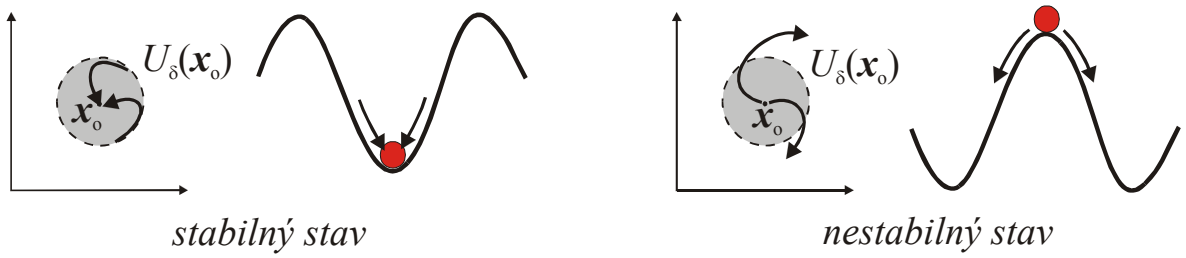
Termín "evolučne stabilná stratégia" bol pôvodne zavedený Maynardom Smithom v rámci jeho biologicky orientovaných aplikácií teórie evolučných hier. Pre lepšie pochopenie pojmu evolučnej stability budeme vychádzať zo štandardnej teórie dynamických systémov, kde existuje pojem asymptoticky stabilný resp. asymptoticky nestabilný bod s jednoduchou fyzikálnou interpretáciou, pozri Obr. 10.3.

Nech populácia P obsahuje v prevažnej miere stratégiu $\boldsymbol{s}_0$ a v malej miere aj inú tzv. votreleckú stratégiu s_1.

$$P = P_0 \cup P_1, \quad P_0 = \{\boldsymbol{s}_0, \boldsymbol{s}_0, ..., \boldsymbol{s}_0\}, \quad |P_0| = p, \quad P_1 = \{\boldsymbol{s}_1, \boldsymbol{s}_1, ..., \boldsymbol{s}_1\}, \quad |P_1| = q$$

pričom predpokladáme, že počet predstaviteľov stratégie $\boldsymbol{s}_0$ je podstatne väčší, ako počet predstaviteľov "votreleckej" stratégie $\boldsymbol{s}_1$, t.j. $p>>q>>0$. Stratégia $\boldsymbol{s}_0$ sa nazýva *evolučne stabilná*, ak v priebehu evolúcie predstavitelia inej stratégie vymiznú. Ak v priebehu evolúcie dochádza k zániku stratégie s_0, táto stratégia s_0 sa nazýva *evolučne nestabilná*.

[5] ďalšia úspešná stratégia založená na trocha inom princípe sa volá Pavlov. Keď jedinec dostane odmenu T alebo R, opakuje svoj krok, keď dostane P alebo S, prepne z nespolupráce na spoluprácu alebo naopak. Ešte zložitejšie sú pravdepodobnostné stratégie.



Obrázok 10.3. Stabilný stav dynamického systému je názorne reprezentovaný guličkou v jamke. Malé vychýlenie systému z rovnovážnej polohy vyvolá procesy, ktoré ho vrátia do pôvodnej rovnovážnej polohy. V hornej časti obrázku je táto verbálna formulácia reprezentovaná pomocou trajektórie znázorňujúcej evolúciu systému. Ak počiatočný bod trajektórie je vybraný z blízkeho okolia rovnovážneho stavu, potom trajektória vždy končí v rovnovážnom bode. Pre nestabilný stav platí, že už malá výchylka stavu z rovnovážnej polohy vyvolá procesy, ktoré vedú k tomu, že systém nenávratne opustí rovnovážny stav. Potom, ak počiatočný bod trajektórie je vybraný z blízkeho okolia rovnovážneho stavu, potom trajektória nenávratne opúšťa rovnovážne okolie.

Sila stratégií z populácie P je určená vzájomným turnajom stratégií

$$fitnes(\boldsymbol{s}_0) = (p-1) f(\boldsymbol{s}_0, \boldsymbol{s}_0) + qf(\boldsymbol{s}_0, \boldsymbol{s}_1)$$
$$fitnes(\boldsymbol{s}_1) = (q-1) f(\boldsymbol{s}_1, \boldsymbol{s}_1) \; + pf(\boldsymbol{s}_1, \boldsymbol{s}_0)$$

Pretože evolúcia populácie prebieha na základe prirodzeného výberu, podmienka evolučnej stability stratégie $\boldsymbol{s}_0$ má tvar

$$fitnes(\boldsymbol{s}_0) > fitnes(\boldsymbol{s}_1)$$
$$(p-1) f(\boldsymbol{s}_0, \boldsymbol{s}_0) + qf(\boldsymbol{s}_0, \boldsymbol{s}_1) > (q-1) f(\boldsymbol{s}_1, \boldsymbol{s}_1) + pf(\boldsymbol{s}_1, \boldsymbol{s}_0)$$

Z predpokladu $p>>q>>0$ vyplývajú tieto dve podmienky pre evolučnú stabilitu stratégie $\boldsymbol{s}_0$

$$f(\boldsymbol{s}_0, \boldsymbol{s}_0) > f(\boldsymbol{s}_1, \boldsymbol{s}_0), \text{ alebo } f(\boldsymbol{s}_0, \boldsymbol{s}_0) = f(\boldsymbol{s}_1, \boldsymbol{s}_0) \text{ a } f(\boldsymbol{s}_0, \boldsymbol{s}_1) > f(\boldsymbol{s}_1, \boldsymbol{s}_1)$$

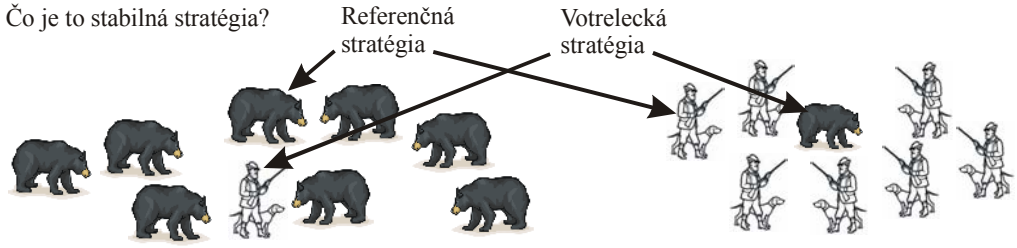
Týmto sme dostali jednoduché podmienky, ktoré špecifikujú či daná stratégia s_0 je v prítomnosti malého počtu votreleckých stratégií s_1 evolučne stabilná. Podmienky požadujú len znalosť funkcie platieb pre jednotlivé stratégie. Tieto platby urobené pre priemernú platbu na ťah z desiatich ťahov za sebou pre dvojicu stratégií by ukázali, že iba stratégie 001 a 111 majú platný vzorček $f(\boldsymbol{s}_0, \boldsymbol{s}_0) \geq f(\boldsymbol{s}_1, \boldsymbol{s}_0)$, teda by teoreticky mohli byť stabilné, keby nebolo stratégií 000 pri stratégii 001 a 101 pri stratégii 111. Tieto dvojice stratégií majú pri vzájomnej hre rovnaké ohodnotenie. Stratégia 000 sa teda popri stratégii 001 môže náhodnými mutáciami rozšíriť. Potom, čo sa rozšíri, môže náhodne vzniknúť ďalšia stratégia, ktorá ich využije na svoje rozšírenie. Podobne to platí aj pre dvojicu stratégií 101 a 111.

Keby stratégie 000 a 101 boli zakázané, potom by stratégie 001 a 111 boli evolučne stabilné. Každá dvojica by bola v tzv. Nashovej rovnováhe[6], teda stratégia, ktorá by nahradila jednu stratégiu z dvojice (001,001) alebo z dvojice (111,111) by si zhoršila platbu. V prípade nahradenia tit-for-tat nespolupracujúcim by síce nespolupracujúci skončil horšie ako predtým, ale relatívne lepšie ako jeho spoluhráč. Vzhľadom k tomu, že sa ale fitnes porovnáva v rámci celej populácie, nespolupracujúci by na tom bol stále zle, jeho spoluhráč tit-for-tat by nabral fitnes v interakciách s ostatnými tit-for-tat (pozri obr. 10.4).

Podobne sa dajú študovať aj stratégie agentov, ktorí si pamätajú dva alebo viac krokov do minulosti. Spolu so zložitosťou stratégií ale narastá tendencia k nespolupráci a chaotické správanie populácie, aj keď je možné tieto tendencie prekonať napríklad ustanovením „administrátora“, ktorý by náhodne trestal nespoluprácu.

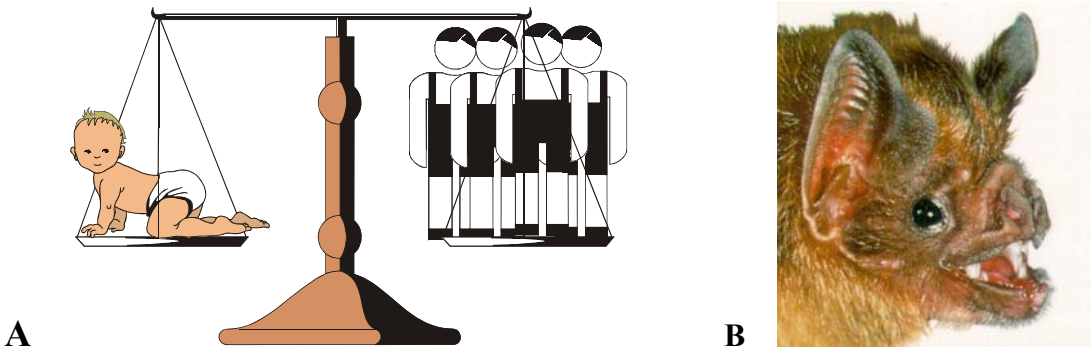
---

[6] Keď existuje súbor stratégií s takou vlastnosťou, že žiaden hráč nemôže profitovať na zmene svojej stratégie, zatiaľ čo ostatní hráči svoju stratégiu nezmenia, potom tento súbor stratégií je v Nashovej rovnováhe (spomeňte si na film Čistá duša – Beautiful mind).



Obrázok 10.4. Príklad toho, že stabilná stratégia závisí na množstve rovnakých spolupracujúcich stratégií najmenej tak závažne ako na kvalite stratégie. Keď je referenčnou stratégiou medveď, tak lovec ako votrelecká stratégia medzi veľkou skupinou medveďou asi neprežije a medveď je stabilnou stratégiou. Opačne to platí aj pre medveďa obkoleseného lovcami.

V prírode sa úzka spolupráca nachádza veľmi často, ale väčšinou je vysvetliteľná pomocou dedičnosti (pozri obr. 10.5). Ale napríklad každý deň asi 8 % dospelých jedincov vampíra nosatého (netopiera *Desmondus rotundus*) nedokáže zohnať potravu, v takom prípade ju s nimi zdieľajú úspešnejší vyvrhnutím časti krvi (čiastočne na základe príbuznosti, čiastočne recipročne). Recipročný altruizmus je aj u samcov paviána (*Papio anuhis*), tí sa združia do dvojice proti protivníkovi, aby jeden z nich sa spojil so samicou (teda dve opakovania koalície tvoria hru, aby každý z nich dostal svoju časť výhry).



Obrázok 10.5. A: V prírode sa väčšina spolupráce dá vysvetliť teóriou sebeckého génu: Z pohľadu genetickej informácie by vaše vlastné dieťa pre Vás malo mať rovnakú váhu ako 4 bratanci, pretože dieťa obsahuje polovicu Vašej genetickej informácie, zatiaľ čo bratanci iba osminu. V prípade klónov by ste mal pomáhať súrodencom ako sebe samému (čo geneticky funguje u hmyzu). **B:** Vampír - neočakávaný príklad recipročného altruistu

10.5 Tragédia spoločného

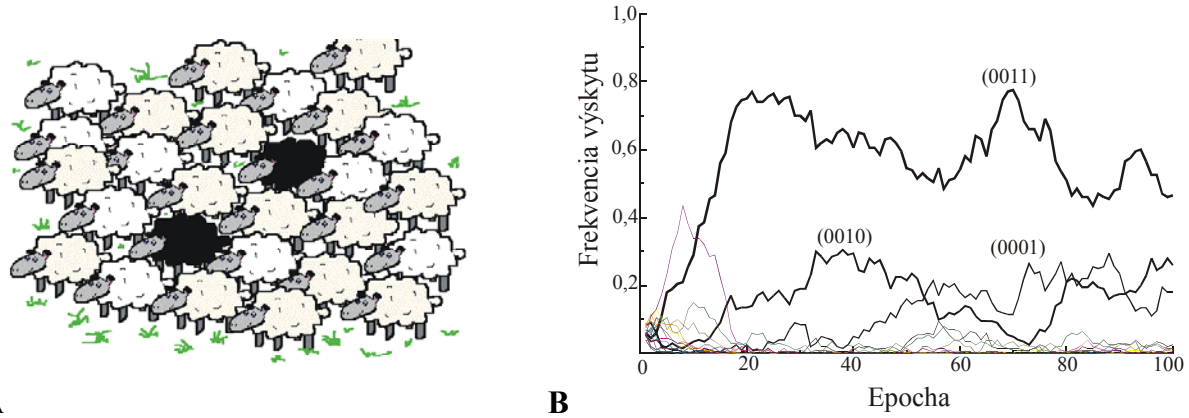
Modelový príklad egoistických tendencií erodujúcich spoluprácu je tzv. tragédia spoločného definovaná Garrettom Hardinom r. 1968 [XX]:

"Predstavte si pastvinu prístupnú pre všetkých. Dá sa očakávať, že každý pastier sa pokúsi držať na spoločnej pastvine toľko dobytka, ako sa len dá. Ako racionálny jedinec, každý pastier sa snaží maximalizovať svoj zisk. Explicitne alebo implicitne sa pýta: "Akú zmenu výnosu pre mňa znamená prídavok jedného alebo viac zvierat k môjmu stádu?"

1. Pozitívna časť je funkciou prínosu jedného zvieraťa. Pretože pastier dostáva všetok zisk z predaja zvieraťa, možný príspevok sa blíži + 1.
2. Negatívna časť je funkciou prídavku prílišného spásania pridaným zvieraťom. Pretože ale výsledky prílišného spásania sú zdieľané všetkými pastiermi, negatívny príspevok pre ktoréhokoľvek rozhodujúceho sa pastiera je iba zlomkom z - 1.

Keď si racionálny pastier spočíta pozitívnu a negatívnu časť prínosu, vychádza mu, že jediné rozumné riešenie je preňho pridať ďalšie zviera k stádu. A ešte ďalšie ...
Ale toto je záver každého racionálneho pastiera zdieľajúceho pastvinu. V tom je tá tragédia. Každý človek je uväznený v systéme, ktorý ho núti zväčšovať svoje stádo bez obmedzenia - v svete ktorý je limitovaný. ... Sloboda v užívaní spoločnej pastviny prináša skazu pre všetkých."

Vyše uvedený príklad sa dá modelovať tiež pomocou väzenskej dilemy, kedy sa namiesto dvoch väzňov interagujú traja alebo viac. Zovšeobecnená väzenská dilema pre viac hráčov (konkrétna stratégia je pre troch) môže vyzerať napríklad takto: Stratégia (0 1 1 0) v prvom kroku kooperuje, nekooperuje, keď v predošlom kroku všetci ostatní kooperovali, nekooperuje, keď v predošlom kroku jeden spoluhráč nekooperoval, kooperuje, keď v predošlom kroku všetci ostatní kooperovali. Výsledok je obr. 10.3.



Obrázok 10.6. A) Tragédia spoločnej pastviny a **B**) modelovanie troma pastiermi. Výsledok genetického algoritmu = zobrazenie frekvencie výskytu stratégií pre spoluprácu troch hráčov. Ako je vidno, vyhráva stratégia 0011, ktorá znamená, že na začiatku je výhodné spolupracovať, a pokiaľ všetci spolupracujú, aj pre mňa je výhodné spolupracovať. Pokiaľ ale iba jeden nespolupracuje, je aj pre mňa výhodné nespolupracovať. Ku koncu vývoja sa vyskytuje aj stratégia 0001 a 0010, ktorá pripúšťa možnosť spolupráce, aj keď jeden alebo dvaja spoluhráči z turnaja nespolupracujú. To je ale výchylka spôsobená tým, že ku koncu vývoja sa vyskytujú väčšinou jedinci zo stratégiou 0011, a teda situácia stretnutia s nespolupracujúcimi jedincami prakticky nenastáva.

Tabuľka pre prvých päť ťahov zovšeobecnenej väzenskej dilemy pre troch hráčov, ktorých stratégie sú určené ako (0110), (1100), (0011)

No.	Ťah			Skóre		
	1.hráč	2. hráč	3. hráč	1.hráč	2. hráč	3. hráč
1	0	1	0	3	10	3
2	1	1	1	2	2	2
3	0	0	1	3	3	10
4	1	0	0	10	3	3
5	1	0	1	6	0	6
Celkové skóre:				24	18	24

Všeobecné vlastnosti víťaznej stratégie: hra je **zahájená kooperáciou**, ak v predchádzajúcom ťahu **všetci oponenti kooperovali, potom kooperuj**, ak v predchádzajúcom ťahu **aspoň jeden oponent nekooperoval, potom nekooperuj**.

Pretože vo väčších celkoch sa vždy nájde nekooperujúci jedinec, nastáva všeobecná nespolupráca - tragédia spoločnej pastviny napr. u rybolovu v medzinárodných vodách, pri rúbaní pralesov, starosti o spoločné zavodňovanie alebo znečisťovaní atmosféry. Politológ Axelrod tvrdí, že v spoločnosti musia vznikať inštitúcie, ktoré kontrolujú - penalizujú nekooperáciu.

Okrem zavedenia administrátora sa emergencie spolupráce dá dosiahnuť aj priestorovým rozmiestnením stratégií, ktoré potom môžu interagovať iba s blízkymi stratégiami, alebo zavedením "dobrého mena" jedincov, kedy si jedinci vymieňajú informácie o spolupráci či nespolupráci ostatných, a k tým, čo nespolupracovali s ostatnými, sa správajú, ako keby nespolupracovali s nimi - tzv. moralistická agresia [XX]. Ďalšou možnosťou je aj tzv. skupinový výber, kde súťaž prebieha nielen medzi jedincami v rámci populácie, ale aj medzi populáciami. Populácia s príliš veľa sebcami má oproti ostatným populáciám nižšie priemerné skóre u jedincov, preto zanikne. U biológov je táto predstava uznávaná iba obmedzene, pretože rýchlosť výmeny jedincov je väčšia ako rýchlosť výmeny populácií, a teda populácia sa skôr nakazí egoizmom, ako by mala zvíťaziť nad inou už egoistickou populáciou. U spoločností ale toto vysvetlenie môže mať svoj význam.

10.6 Kde sa dá využiť koevolúcia v informatike?

Aj zdanlivo sociobiologické koncepty ako koevolúcia sa môžu s výhodou použiť napr. pri evolúcii programov ako sú triediace algoritmy pre zoznamy čísel. Testovanie všetkých možných triediacich sietí, ktoré dokážu korektne porovnať iba 16 čísel pomocou povedzme 64 porovnaní, by vyžadovalo vyskúšať 10^{154} sietí. Preto sa Hillis [XXX] pokúšal vyvíjať algoritmy a ohodnocovať ich podľa toho, koľko zoznamov zoradili správne. Takto vyvinuté algoritmy boli však ďaleko od optima. Keď sa však zároveň nechali vyvíjať aj zoznamy, ktoré boli ohodnotené tým lepšie, čím viac algoritmov ich nezoradilo správne (v biológii takéto zoznamy odpovedajú parazitom), výsledné algoritmy boli oveľa efektívnejšie.

Bublinkové triedenie Výsledok evolúcie

Obrázok 10.7. Korektné siete na triedenie sady čísel podľa veľkosti. Jeden z často používaných algoritmov na triedenie, tzv. bublinkové triedenie (bubble sort), dokáže zoradiť 6 čísel s použitím 15 porovnaní, zatiaľ čo jednej z minimálnych korektných sietí napravo stačí iba 12 porovnaní. Takýchto minimálnych korektných sietí existuje viac (môžeme napríklad prehodiť prvé dve porovnania).

Triediaca sieť opisuje algoritmus na zoraďovanie sady čísel podľa veľkosti od najmenšieho (hore) k najväčšiemu (dole). Každá vodorovná čiara v sieti reprezentuje umiestnenie čísla v zozname. Každá zvislá úsečka zakončená štvorčekmi predstavuje porovnanie dvoch čísel v daných umiestneniach a ich prípadnú výmenu, pokiaľ je v "hornej pozícii" väčšie číslo ako v "dolnej". V prípade, že je mechanizmus usporiadania korektný, mal by správne zoradiť všetky možné permutácie danej veľkosti (viď. obr. 10.7). Keď je sieť minimálna, neexistuje korektný algoritmus s menším počtom porovnávaní.

10.7 Kultúra, mémy a emoční agenti

Ako ľudia získavajú spoločné hodnoty ako myšlienky, vieru, názory alebo obľubu? Jeden z možných mechanizmov je šírenie spoločných hodnôt cez populáciu "nákazlivým" procesom, ktorý nastáva, keď spolu ľudia interagujú. Takáto imitačná interakcia sa volá *kultúrny prenos* alebo *sociálne učenie*. To sa dá jednoducho modelovať: národ je charakterizovaný sadou

vlastností. Systém sa vyvíja po krokoch a pri interakcii si dva národy porovnajú navzájom vlastnosti. Keď sú rovnaké, nedeje sa nič. Keď sa líšia, pravdepodobnosť *kultúrnej interakcie* je priamo úmerná ich *kultúrnej podobnosti*. Interakcia spočíva v nahradení jednej z rozdielnych hodnôt jednej vlastnosti národov novou hodnotou. Toto sa stále opakuje.

Do podobných modelov sa dá zahrnúť aj sociálny status a preberanie vzoru. Keď dvaja ľudia alebo národy nerovného sociálneho statusu interagujú kultúrne, je pravdepodobnejšie, že jedinec s nižším statusom prevezme mém[7] od jedinca s vyšším statusom. Keď podobný postup aplikujeme na vyššie uvedený model, konvergencia k väčším oblastiam podobnosti je oveľa silnejšia. Ďalšie oveľa rýchlejšie formovanie súvislých oblastí rovnakých vlastností oblastí je možno dosiahnuť tým, že sa indivídua snažia mať podobné vlastnosti ako ich okolie.

Konečne najnovším trendom v modelovaní je aj zahrnutie emócií. Výpočtové modely emócií sa vytvárali už od 60–tých rokov 20. storočia, ale až v poslednej dekáde sa začali prudko rozvíjať (pozri obr. 10.8). Začínajú sa využívať aj u emocionálnych umelých agentov napríklad v psychológii pri modelovaniu a kontrole psychologických teórií, v umelej inteligencii na vytvorenie motivácie agentov a ako pomôcka pri rozhodovaní v komplexných situáciách a v týmoch agentov a konečne v interakcii človek–počítač, kde ide hlavne o analýzu snímaného a generovanie umelého obličaje a tónu hlasu. Asi najviac zaujímavé nie sú vlastné modely, ale reakcia ľudí na ne (napr. konverzačný program Eliza [XX] je časťou ľudí braný ako ľudský partner v diskusii). Väčšinou sú ale emócie je zabudované v rámci schémy klasickej umelej inteligencie, teda v zložito človekom budovaných agentoch

Obrázok 10.8. Jonathan Klein, 3D animácia obličaja umelého agenta pomenovaného Bruzard

10.8 Záverom

Aj keď model väzenskej dilemy vyzerá byť veľmi primitívny, je študovaný už od roku 1950,. Počet článkov, ktoré sa týmto a jemu príbuznými modelmi zaoberajú, stále narastá, no tieto modely sú stále dosť vzdialené reálnej skutočnosti na to, aby boli spoľahlivými nástrojmi na rozhodovanie o spolupráci či nespolupráci. Môžu byť ale užitočnou pomôckou poukazujúcou na niektoré logické trendy vývoja.

Rovnako je to v podobných, aj keď väčšinou oveľa komplikovanejších modeloch v sociológii. U podrobných modelov je ale treba dávať pozor, aby do nich zlým odhadom nezaniesli chybu (u každého modelu je treba niečo zanedbať), čo by spôsobilo nepresnú predikciu. Napriek tomuto nebezpečiu je multiagentový prístup spolu s„natvrdo“ vystavanými modelmi založenými na diferenciálnych rovniciach jedinou rozumnou alternatívou, ako verifikovať často vágne sociologické teórie.

[7] mém je jednotka kultúrnej informácie, ako pesnička, zvyk alebo myšlienka, ktorá je prenášaná z jednej mysle do druhej na základe slovného vyjadrenia alebo pozorovaním akcie

Zhrnutie

Multiagentové systémy nám dávajú odpovede na otázku, ako heterogénny mikrosvet správaní indivíduí generuje globálne makroskopické pravidelné črty spoločnosti. Sú využívané napr. v soviológii, antropológii, ekonomike, pri modelovaní vzťahov medzi štátmi. Jednou z oblastí modelovania je existencia spolupráce či nespolupráce u sociálnych dilem, ako tragédia spoločného pri znečisťovaní životného prostredia alebo ako sú obchodné vojny. Jedným z najviac využívaných modelov je väzenská dilema. V rámci teórie hier sa skúma stabilita a efektívnosť jednotlivých stratégií riešenia problému spolupráce. Koevolúcia a koncept parazitizmu z biológie sa dá výhodne využiť aj v informatike pri optimalizácii. Konečne multiagentové systémy sa využívajú aj pri skúmaní prenosu kultúry a zložitejšie agenty aj pri skúmaní emócií.

Úlohy

Úloha 10.1. Táto úloha sa týka dvoch párov spolubývajúcich, Janka a Fera, a Ľuby a Anky. Každý z nich sa musí rozhodnúť, či venovať čas na upratanie izby alebo nie. Úroveň spokojnosti jednotlivcov závisí jednak na ich rozhodnutiu, jednak na rozhodnutiu ich spolubývajúceho. Janko aj Fero sa zhodnú na tom, že štyri možné situáciu zoradia nasledovne:

Najlepšia - môj spolubývajúci upratuje všetko, ja nerobím nič
Druhá najlepšia - obidvaja sa podieľame na upratovaní
Druhá najhoršia - nikto z nás neupratuje
Najhoršia - ja upratujem všetko, môj spolubývajúci nerobí nič

Na druhej strane Ľuba a Anča zoradia situácie takto:

Najlepšia - moja spolubývajúca upratuje všetko, ja nerobím nič
Druhá najlepšia - obidve sa podieľame na upratovaní
Druhá najhoršia - ja upratujem všetko, moja spolubývajúca nerobí nič
Najhoršia - nikto z nás neupratuje

Dá sa "hra" medzi Jankom a Ferom opísať ako väzenská dilema alebo hra na kurča? Ako je to s "hrou" Ľuby a Anči?

Úloha 10.2. Keď ako hráč z dvojice hráčov vo väzenskej dileme viem, že môj spoluhráč si zo začiatku zvolí spoluprácu a bude spolupracovať, dokiaľ ja budem spolupracovať, ako sa mám logicky zachovať pri 10 ťahoch, keď je mojím cieľom získať čo najviac? Ako sa mám zachovať, keď nie je určený počet ťahov, ale ťah so spoluhráčom sa bude opakovať s určitou pravdepodobnosťou p? Odvoďte vzorček, kedy spolupracovať, na základe veľkosti p.[8]

Úloha 10.3. Diskutujte, či niektoré z nasledujúcich situácií by mohli byť modelované ako väzenská dilema pre n väzňov: vložiť do zlej banky peniaze na vysoký úrok, keď je štátna záruka vrátenia 100 percent v prípade krachu banky; stať na sedadle pri rockovom koncerte; súkromne telefonovať zadarmo z práce. Čo by v týchto prípadoch bola stratégia kooperácie a podvádzania? Je individuálny hráč na tom lepšie, keď podvádza, bez ohľadu na to, čo robia ostatní? Sú všetci hráči na tom lepšie, keď všetci kooperujú než keď všetci podvádzajú?

[8] Ukážte najprv, že $1+p+p^2+...+p^{m-1}=(1-p^m)/(1-p)$, a $1+p+p^2+...=1/(1-p)$

Úloha 10.4. Keď je veľa stratégií tit-for-tat (001) a jeden "zloduch" (111), zloduch má najmenšiu fitnes. Keď je veľa stratégií "zloduch" (111) a jeden tit-for-tat (001), tit-for-tat má najmenšiu fitnes. Vyrobte graf pre 10 stratégií, kde na ose *x* je počet stratégií tit-for-tat od 1 po 9 (ostatné sú zloduch), na ose *y* je počet opakovaní ťahov pre jednotlivé dvojice (od 1 po 10), a na ose *z* je zobrazený súčet ohodnotení jedinca zo stratégiou tit-for-tat pre daný počet opakovaní mínus súčet ohodnotení jedinca so stratégiou zloduch. Vyhodnoťte, kedy je ktorá z dvoch uvedených stratégií stabilná, teda odolná proti vpádu druhej, votreleckej stratégie, a kedy ktorá zo stratégií získa viac bodov.

Úloha 10.5. Keď hrá stratégia tit-for-tat (001) a "zloduch" (111) spolu 10 ťahov a keď je mojím cieľom získať čo najviac, ktorú stratégiu si vyberiem v prípade, že administrátor (povedzme kanadská jazdná polícia) s pravdepodobnosťou *p* zoberie nespolupracujúcemu interagujúcemu so spolupracujúcim práve získané body *T*? Odvoďte vzorček, ktorý určí priemerne získané body pre tit-for-tat a zloducha v závislosti na *p*. Závisí výhra jedného nad druhým na počtu ťahov?

Úloha 10.6. Uvažujme situáciu študentov: Výborná známka bude ohodnotená 20 bodmi; priemerná 5 bodmi; zlá 0 bodmi. Veľa času na štúdium stojí ekvivalent 10 bodov; stredne času na štúdium 6 bodov.
Pre veľa času na štúdium a výslednú výbornú známku je teda výsledný efekt 10 bodov. Pre veľa času na štúdium a výslednú priemernú známku je teda výsledný efekt -5 bodov. Pre stredne času na štúdium a výslednú priemernú známku je teda výsledný efekt -1 bodov. Pre stredne času na štúdium a výslednú zlú známku je teda výsledný efekt -6 bodov.

Ďalej predpokladajme, že skúšky sú klasifikované nie na základe absolútnych vedomostí študenta, ale v porovnaniu jeho znalostí s ostatnými študentmi. Keď každý preukáže rovnaké vedomosti, všetci dostanú priemernú známku. Keď Jano študuje tvrdo a nikto iný nie, dostane výbornú známku a ostatní dostanú zlú. Keď Jano študuje stredne veľa a všetci ostatní študujú veľa, dostane zlú známku a ostatní výbornú známku.

Čo by v takomto modeli znamenala kooperácia a čo podvádzanie? Zostavte maticu platieb, zodpovedá podmienkam väzenskej dilemy? Keď sú iba dvaja študenti a idú na párny počet skúšok, aká by mala byť ich ideálna stratégia? Aká by mala byť ideálna stratégia *n* študentov idúcich na *n* skúšok?

Úloha 10.7. Pre každú z triediacich sietí na obr. 10.7 napíšte sadu permutácií, cez ktoré sa mení permutácia (3,2,4,1,6,5) na permutáciu (1,2,3,4,5,6).

Úloha 10.8. Na základe postupu triedenia z kapitoly 2 skúste navrhnúť čo triediacu sieť s čo najmenším počtom porovnaní pre triedenie 8 čísel.

Úloha 10.9. Keď sú permutácie používané ako parazity pri vývoji triediacich sietí, nájdite najneúspešnejšieho parazita danej dĺžky.

Úloha 10.10. Nájdite minimálnu sieť, ktorá dokáže usporiadať permutáciu (4,3,2,1) a všetky permutácie, ktoré pre túto sieť budú úspešnými parazitmi.

Otázky na opakovanie

Otázka 10.1. Skúste definovať, čo je autonómny agent a na akú základnú otázku v sociológii a ekonomike odpovedajú multiagentové systémy.

Otázka 10.2. Ktoré všetky druhy hier boli spomenuté v kapitole?

Otázka 10.3. Opíšte stratégiu tit-for-tat a Pavlov u väzenskej dilemy?

Otázka 10.4. Čo je evolučne stabilná stratégia?

Otázka 10.5. Čo je to Nashova rovnováha?

Otázka 10.6. Skúste vysvetliť myšlienku sebeckého génu?

Otázka 10.7. Vysvetlite podstatu tragédie spoločného. Keď niekto na zastávke potrhá cestovné poriadky autobusov, jedná sa tiež o tragédiu spoločného?

Otázka 10.8. Aké sú prístupy na podporu spolupráce?

Otázka 10.9. Ako sa dá využiť koncept parazitizmu v informatike?

Otázka 10.10. Čo je to mém?

11 Metódy usudzovania

Jeden z kľúčových problémov modernej informatiky, umelej inteligencie a kognitívnej vedy je usudzovanie (inferencia) nových poznatkov z danej databázy vedomostí. Klasická výroková logika obsahuje mnoho rôznych „deduktívnych“ pravidiel (modus ponens, modus tollens, sylogizmy, a pod.), kde pravdivosť záverov vyplýva z pravdivosti premís. To znamená, že pri používaní deduktívnych pravidiel sa obsah informácie nezvyšuje, ale zostáva konštantný, odvodené nové poznatky sú už nejakým spôsobom obsiahnuté vo východzích premisách. Preto našu pozornosť obrátime hlavne na nededuktívne mechanizmy usudzovania, kde môže vznikať nová informácia mechanizmami zovšeobecnenia alebo tvorby nových hypotéz, tieto dva inferenčné mechanizmy neustále používame k prijateľnému vysvetleniu a interpretácii nášho vonkajšieho sveta. Usudzovanie, podľa toho či je uvedomelá alebo nie, sa obvykle rozdeľuje na dva druhy:

(1) *reflexívna usudzovanie*, kde bez vedomého úsilia interpretujeme náš okolitý svet, o ktorom bez prestania dostávame stále nové a nové informácie (napr. pri interpretácii počutého prirodzeného jazyka), a

(2) *reflektívna usudzovanie*, kde s vedomým úsilím vytvárame náš obraz o okolitom svete (napr. tvorba vedeckej teórie).

Pre ilustráciu dôležitosti reflexívnej usudzovania uvedieme klasický príklad interpretácie rozprávky o malom dievčatku nazývanom *Červená Čiapočka*, ktorá sa rozpráva malým deťom vo veku 3-6 rokov. Týmto jednoduchým príkladom chceme zdôrazniť, že elementárne, tak deduktívne, ako aj nededuktívne inferenčné mechanizmy, musia byť vlastné už aj malým deťom, aby boli schopné správne pochopiť a interpretovať túto rozprávku.

Obrázok 11.1. Červená čiapočka [XX]

Nech malý poslucháč je práve v tom bode rozprávky, kde Vlk sa kontaktuje s Červenou Čiapočkou (ČČ). V nasledujúcej časti rozprávky malý poslucháč získa túto informáciu: „*Vlk počul zvuky sekier blízkych drevorubačov a preto sa rozhodol počkať*“. Predpokladajme, že malý poslucháč spontánne (a reflexívne) pochopí túto vetu. Avšak jej pochopenie vyžaduje celú postupnosť reflexívnych inferencií, ktoré neformálne môžeme popísať takto (poznámky v zátvorke vyjadrujú základné vedomosti v pozadí, ktoré tvoria východzie premisy pre inferenčný proces):

(1) Vlk sa priblíži k ČČ (aby niečo zjedol, čo nie je moc vzdialené);
(2) ČČ bude kričať (pretože deti kričia, keď sa k nim priblíži zlé zviera);
(3) drevorubači, keď budú počuť krik, budú vedieť, že dieťa je v nebezpečí (pretože krik dieťaťa znamená, že je v nebezpečí);
(4) drevorubači prídu k dieťaťu a pokúsia sa ČČ zachrániť pred Vlkom (ľudia ochraňujú deti, ktoré sú v nebezpečenstve, čo je však veľmi nebezpečné pre Vlka);
(5) preto drevorubači môžu napadnúť vlka (ochrana pre zvieraťom vyžaduje fyzickú silu...);
(6) Vlk sa rozhodol počkať (pretože zviera nechce byť napadnuté ľuďmi).

Z tohto jednoduchého ilustratívneho príkladu vyplýva, že už malé dieťa musí byť schopné nededuktívnej reflexívnej interferencie, aby bolo schopné pochopiť príbeh rozprávky (alebo vo všeobecnosti, pochopiť prirodzený jazyk dospelých, ktorý už obsahuje implicitne skryté východzie premisy a teórie).

Aké sú mechanizmy našej inferencie pri vzniku nových poznatkov, t.j. v prípadoch keď odvodené poznatky majú väčší informačný obsah ako ich východzie premisy? Tieto metódy inferencie, ktoré sa nazývajú *nededuktívne metódy*, sú založené na zovšeobecnení východzích poznatkov (napr. výrok „*všetci bratislavskí informatici sú inteligentní*“ je zovšeobecnený na „*všetci informatici sú inteligentní*“), na analógii, generovaní hypotéz a pod.

Americký filozof a logik Charles S. Pierce, ktorý je známy ako spoluzakladateľ filozofického smeru pragmatizmu (veľmi populárneho v 1. polovici minulého storočia hlavne v USA) sa stal známym svojou klasifikáciou nededuktívnych metód inferencie, od už známej indukcie oddelil tzv. abdukciu, ako metódu nezávislú od indukcie, často využívanú pri tvorbe hypotéz vysvetľujúcich nejaké pozorované javy (napr. lekárskej diagnózy).

Pierceho úvahy vychádzali z pravidla modus ponens a z jeho možných ďalších kombinácií (bez ohľadu na to, či sú platné alebo nie, menovite v druhom a v treťom stĺpci)

dedukcia	***indukcia***	***abdukcia***
$\begin{array}{l} p \Rightarrow q \\ \underline{p\quad\quad} \\ q \end{array}$	$\begin{array}{l} p \\ \underline{q\quad\quad} \\ p \Rightarrow q \end{array}$	$\begin{array}{l} p \Rightarrow q \\ \underline{q\quad\quad} \\ p \end{array}$
$\begin{array}{l} B\ a\ A \\ \underline{C\ i\ B\quad} \\ C\ i\ A \end{array}$	$\begin{array}{l} C\ i\ B \\ \underline{C\ i\ A\quad} \\ B\ a\ A \end{array}$	$\begin{array}{l} B\ a\ A \\ \underline{C\ i\ A\quad} \\ C\ i\ B \end{array}$

V dolnej časti tabuľky sú uvedené sylogizmy, ktoré boli použité Pierceom na alternatívnu klasifikáciu inferenčných módov. Prvý stĺpec je priradený štandardnému deduktívnemu módu inferencie založenom na pravidle modus ponens, jeho sylogizmová alternatíva v spodnom riadku odpovedá slávnemu stredovekému príkladu „*všetci ľudia sú smrteľní a Sokrates je človek, teda Sokrates je smrteľný*“.

Je potrebné poznamenať, že táto tabuľka má hlavne heuristický význam, bez požadovania hlbšieho významu jednotlivých módov inferencie. Pierceov prístup použijeme hlavne pre jeho jednoduchosť a názornosť, aj keď z pohľadu moderných prístupov k štúdiu inferencie sa už javí trochu „ťažkopádny“. Moderný pohľad na nededuktívnu inferenciu vznikol hlavne zásluhou informatiky (umelej inteligencie), kde sa intenzívne študujú metódy vyvodzovania nových poznatkov pomocou zovšeobecnenia a tvorby hypotéz nad databázou poznatkov.

11.1 Dedukcia

Podľa Piercea je dedukcia charakterizovaná zákonom „modus ponens“ alebo platným sylogizmom $B\ a\ A \wedge C\ i\ B \Rightarrow C\ i\ A$ (pozri kap. 4). Uvediem jeho klasický príklad s fazuľami (pozri obr. 11.1).

teória	:všetky fazule z tohto vrecka sú biele
pozorovanie	:tieto fazule sú z tohto vrecka
záver	:tieto fazule sú biele

Pomocou kvantifikátorov môžeme tento príklad prepísať do tvaru

teória	$:(\forall x \in$ *fazule*$)$	$vrecko(x) \Rightarrow biela(x)$
pozorovanie	$:(\forall x \in$ *tieto_fazule*$)$	$vrecko(x)$
záver	$:(\forall x \in$ *tieto_fazule*$)$	$biela(x)$

kde „*tieto_fazule*“ je podmnožina množiny „*fazule*“.

Obrázok 11.1. Grafické znázornenie módu dedukcie pomocou platného sylogizmu $B\ a\ A \wedge C\ i\ B \Rightarrow C\ i\ A$, ktorý môže byť pretransformovaný na zákon „modus ponens“. Máme dve premisy: prvá je, že všetky fazule z tohto vrecka sú biele, druhá premisa je „tieto fazule sú z tohto vrecka“, záver z týchto dvoch premís je: tieto fazule (pretože sú z vrecka) sú biele. Podobné príklady deduktívnej inferencie môžu byť urobené aj pomocou ostatných zákonov výrokovej logiky a sylogizmov, diskutovaných v kapitole 4.

Pre deduktívnu inferenciu je podstatné, že platnosť (pravdivosť) záverov vyplýva z platnosti premís, nemusí sa analyzovať „obsahový význam“ premís, či a kedy sú platné, a potom, na základe ich platnosti, či a kedy je záver platný. Z tejto skutočnosti vyplýva, že pri deduktívnej inferencii nevzniká nová informácia, ktorá by nebola už obsiahnutá v premisách. Pretože máme hlavne záujem o také inferenčné módy, kde informácia záveru už nie je bezprostredne obsiahnutá v premisách (pričom sa musia vykonávať určité „mimologické“ činnosti – operácie), ale je novo vytvorená, obrátime našu pozornosť na ďalšie inferenčné módy uvedené Pierceom , a to na indukciu a dedukciu.

11.2 Indukcia

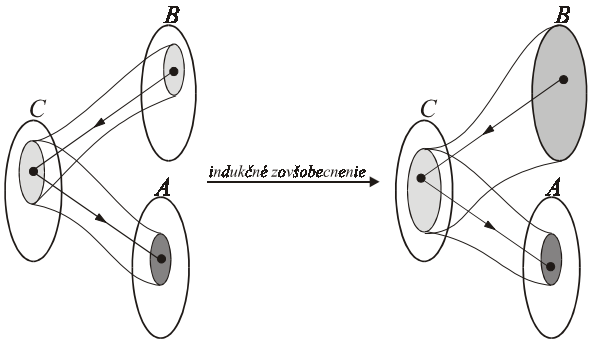
Pierce charakterizuje indukciu pomocou schémy

$$\frac{\begin{array}{l} p \\ q \end{array}}{p \Rightarrow q}$$

ktorej záver je $p \Rightarrow q$ pravdivý, ak premisy p a q sú pravdivé, alebo $(p \wedge q) \Rightarrow (p \Rightarrow q)$. Z tejto schémy vlastne vyplýva, že ak súčasne sú pravdivé premisy p a q, tak potom tieto premisy môžu vytvárať „asociáciu“ $p \Rightarrow q$. Charakterizovanie indukcie pomocou sylogizmu (pozri obr. 11.2) má tvar

$$\frac{\begin{array}{l} C\ i\ B \\ C\ i\ A \end{array}}{B\ a\ A}$$

Tento sylogizmus však nie je platný (z pohľadu predikátového počtu, t.j. nie je jedným z tabuľky 27 platných sylogizmov uvedených v 4. kapitole).



Obrázok 11.2. Grafické znázornenie sylogizmu $(C\ i\ B) \wedge (C\ i\ A) \Rightarrow B\ a\ A$. Ľavý diagram znázorňuje pôvodný sylogizmus, pomocou ktorého môžeme zostrojiť platný sylogizmus $(C\ i\ B) \wedge (C\ i\ A) \Rightarrow B\ i\ A$. Zovšeobecnením tohto sylogizmu dostaneme pravý diagram, ktorý platí pre každé B, t.j. platí záver “všetky B sú A“.

„Fazuľový“ tvar Pierceho sylogizmu indukcie je (pozri obr. 11.3)

teória	: tieto fazule sú z tohto vrecka	
pozorovanie	: tieto fazule sú biele	
záver	: všetky fazule z tohto vrecka sú biele	

alebo

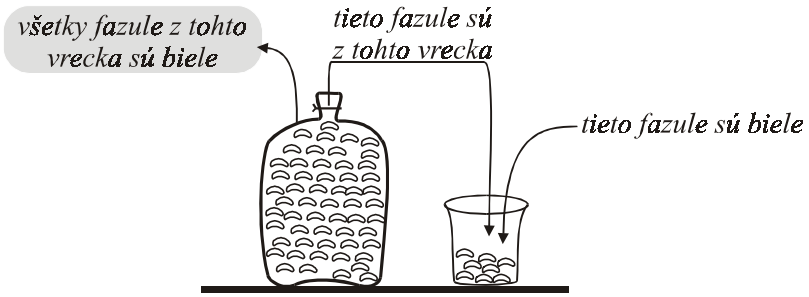
teória	: $(\forall x \in$ *tieto_fazule*$)$	$vrecko(x)$
pozorovanie	: $(\forall x \in$ *tieto_fazule*$)$	$biela(x)$
záver	: $(\forall x \in$ *fazule*$)$	$vrecko(x) \Rightarrow biela(x)$

Tvar tohto sylogizmu vyjadreného pomocou formálnych predikátorov je

teória	: $(\forall x \in F_0)$	$V(x)$
pozorovanie	: $(\forall x \in F_0)$	$B(x)$
záver	: $(\forall x \in F)$	$V(x) \Rightarrow B(x)$

Vyššie uvedený sylogizmus nie je vo všeobecnosti platný, platným sa stáva vtedy, keď univerzálny kvantifikátor dôsledku obsahuje len objekty z podmnožiny F_0. Je potrebné zdôrazniť, že „indukčné zovšeobecnenie“ spočíva práve v rozšírení F_0 na „nadmnožinu“ $F \supseteq F_0$. Budeme rozlišovať dva procesy nad týmto indukčným zovšeobecnením, prvý sa nazýva *verifikácia* a druhý je *falzifikácia*. Verifikácia znamená, že neustále hľadáme také $x \in F$ pre ktoré platí implikácia $V(x) \Rightarrow B(x)$. Proces verifikácie záveru je „never ending story“, neustále môžeme hľadať nové a nové ilustratívne príklady platnosti záveru, žiaľ tento verifikačný proces len nepatrne zvyšuje naše poznanie. O mnoho dôležitejší je proces falzifikácie, stačí nájsť jeden príklad $x \in F$ pre ktorý neplatí implikácia $V(x) \Rightarrow B(x)$, potom indukčné zovšeobecnenie $(\forall x \in F)\ V(x) \Rightarrow B(x)$ je nepravdivé.

Na tento dôležitý moment falzifikácie indukčného zovšeobecnenia prvý upozornil rakúsko – anglický filozof Karl Popper [xx], ktorý charakterizoval vedu ako postupnosť falzifikovania hypotéz. V súčasnej filozofii vedy sa pod falzifikáciou rozumie idea, že pokrok vo vede sa uskutočňuje prostredníctvom slobodnej kritiky, ktorá nie je ničím a nikým obmedzovaná. Len hypotézy, ktoré sú schopné porovnania s experimentmi umožňujú vedecký pokrok, napr. hypotéza „*zlato je rozpustné v kyseline chlorovodíkovej*“ je vedecká (aj keď nepravdivá) hypotéza; „*niektoré homeopatické lieky sú vskutku účinné*“ je nevedecká hypotéza (aj keď môže byť pravdivá). Prvá hypotéza je vedecká preto, že môže byť falzifikovaná jednoduchým experimentom. Druhá hypotéza o homeopatikách je nevedecká z dôvodu, že jej experimentálna falzifikovateľnosť je veľmi problematická, môže byť falzifikovaná (čo sa aj často deje) pomocou našich všeobecných predstáv o fyzikálnych a chemických vlastnostiach hmoty[1]. Nefalzifikovateľné teórie sa podobajú počítačovému programu, ktorý nemá výstup, teda nemáme šancu výstup testovať. Falzifikovateľné teórie môžeme „kontrolovať“ pomocou chýb, ktoré produkujú, keď sú aplikované k situáciám reálneho sveta.



Obrázok 11.3. Grafické znázornenie módu indukcie pomocou sylogizmu $C\ i\ A \wedge C\ i\ B \Rightarrow B\ a\ A$. Pri ilustrácii tohto sylogizmu vychádzame z dvoch premís: prvá premisa je „tieto fazule sú z tohto vrecka“, druhá premisa je „tieto fazule sú biele“, záver z týchto dvoch premís je: „všetky fazule z tohto vrecka sú biele“. Zovšeobecnie spočíva v tom, že farba vybraných fazúľ bola zovšeobecnená na všetky fazule.

11.3 Abdukcia

Tretí inferenčný mód navrhnutý Peirceom je *abdukcia*, ktorá je špecifikovaná schémou

[1] Pre vedeckosť hypotézy by bolo treba špecifikovať, ktoré homeopatické lieky majú byť účinné. Samotné vedecké overenie by potom vyžadovalo štatistickú analýzu dostatočne veľkej vzorky pacientov liečených jednak homeopatikom, jednak čistou vodou (tzv. slepý pokus), aby sa odstránil "placebo" efekt.

$$\frac{p \Rightarrow q \quad q}{p}$$

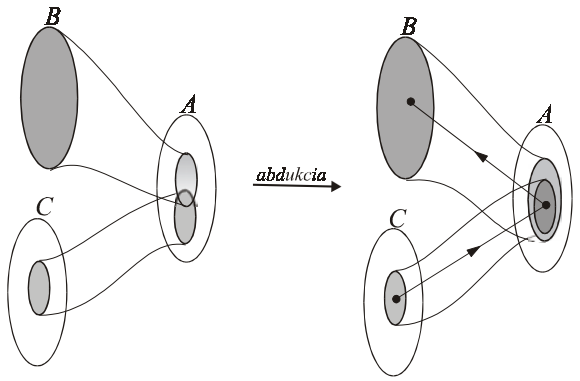
ktorá nie je logickým zákonom (v 4. kapitole táto schéma bola nazvaná ako *potvrdenie dôsledku*). To znamená, že z platnosti premís $p \Rightarrow q$ a q chceme vyvodiť aj platnosť p, čo vo všeobecnosti nie je možné a vyššie uvedená schéma sa pokladá za príklad chybného uvažovania. Podívajme sa na túto schému alternatívnym spôsobom navrhnutým Pierceom. Premisu $p \Rightarrow q$ pokladajme za teóriu, ktorá spája do kauzálneho reťazca p a q, výskyt „*príčiny*“ p zaručuje aj výskyt „*príznaku*“ q. Na tento kauzálny reťazec je možný aj „inverzný“ pohľad (stanovenie diagnózy), výskyt „príznaku“ q je odôvodnený výskytom príčiny p. Táto špecifikácia abdukcie vedie niektorých autorov aj k tomu, že nazývajú abdukciu ako spätný modus tollens. Pierceho sylogizmus pre abdukciu má tvar

$$\frac{B\ a\ A \quad C\ i\ A}{C\ i\ B}$$

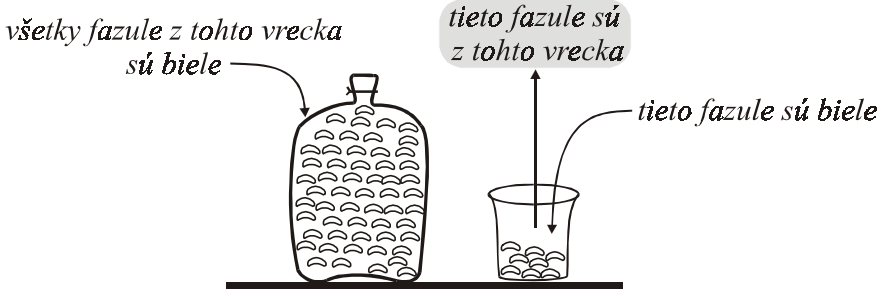
jeho grafická interpretácia je znázornená na obr. 11.4.

Alternatívny Pierceho prístup k špecifikácii abdukcie je pomocou „fazuľového“ sylogizmu (pozri obr. 11.5)

teória	:	všetky fazule z tohto vrecka sú biele
pozorovanie	:	tieto fazule sú biele
hypotéza	:	tieto fazule sú z tohto vrecka



Obrázok 11.4. Grafické znázornenie sylogizmu $(B\ a\ A) \wedge (C\ i\ A) \Rightarrow C\ i\ B$. Ľavý diagram znázorňuje pôvodný sylogizmus, pomocou ktorého môžeme zostrojiť platný sylogizmus abdukcie spočívajúci v tom, že sa predpokladá špecifický tvar „projekcií“ $C \rightarrow A$ a taktiež $B \rightarrow A$.



Obrázok 11.5. Grafické znázornenie módu abdukcie pomocou sylogizmu $(B\ a\ A) \wedge (C\ i\ A) \Rightarrow C\ i\ B$. Pri ilustrácii tohto sylogizmu vychádzame z dvoch premís: prvá premisa - teória je „všetky fazule z tohto vrecka sú biele“, druhá premisa - pozorovanie je „tieto fazule sú biele“, záver - hypotéza z týchto dvoch premís je: „tieto fazule sú z tohto vrecka (pretože sú biele). Abdukcia spočíva v tom, že farba vybraných fazúľ bola použitá ako základ pre tvorbu hypotézy, že tieto fazule pochádzajú z vrecka.

Formalizácia tohto sylogizmu pomocou predikátov má tvar

teória	$(\forall x \in F)$	$vrecko(x) \Rightarrow biela(x)$
pozorovanie	$(\forall x \in F_0)$	$biela(x)$
hypotéza	$(\forall x \in F_0)$	$vrecko(x)$

kde F_0 sú vybrané fazule, ktoré teoreticky nemusia byť z vrecka.

Pri tvorbe hypotéz pomocou abduktívnej inferencie je možné vždy niekoľko alternatívnych príkladov, ktoré sú konzistentné s danou teóriou. Tak napríklad, vyššie uvedený „fazuľový“ ilustračný príklad môže taktiež obsahovať hypotézu „tieto fazule sú od Jana“, ktorá taktiež konzistentne vysvetľuje pozorovanie „tieto fazule sú biele“. Máme teda dve hypotézy

h_1: tieto fazule sú z vrecka
h_2: tieto fazule sú od Jana

Stojíme pred problémom, ktorú hypotézu vybrať ako prijateľnejšiu. Je to zložitý „filozoficko-metodologický“ problém, ktorý sa dotýka priamo základov vedy. V tejto súvislosti sa často spomína kritérium „Ockhamova britva“, pripisované anglickému stredovekému scholastikovi – františkánskemu mníchovi Williamovi z Ockhamu, podľa ktorého veci majú byť len tak zložité, ako je to potrebné (*entia non sunt multiplicanda praeter necessitatem*). Druhá hypotéza h_2 obsahuje novú entitu „Jano“ , ktorá sa nevyskytuje v teórii, preto druhá hypotéza má slabšiu platnosť ako prvá hypotéza, pretože zavádza ad-hoc novú entitu „Jano“. V súčasnosti princíp „Ockhamova britva“ sa taktiež nazýva „princíp jednoduchosti“, podľa ktorého „jednoduchšie vysvetlenie je lepšie ako iné zložitejšie vysvetlenie“, alebo „hypotézy nemajú byť zbytočne zložité“. Toto kritérium jednoduchosti sa často používa vo filozofii vedy vtedy, keď si máme vybrať medzi hypotézami, ktoré majú rovnakú vysvetľujúcu schopnosť. Švajčiarsky spisovateľ von Däniken[2] môže mať pravdu v tom, že pravekých ľudí mimozemšťania učili umeniu a inžinierstvu, avšak nemusíme predpokladať ich návštevu k tomu, aby sme vysvetlili schopnosť pravekých ľudí maľovať a zostrojovať kamenné stavby a jednoduché mechanické zariadenia.

11.4 Moderná formulácia indukcie a abdukcie

Budeme hľadať spoločné vlastnosti indukcie a abdukcie. Obe obsahujú premisu – teóriu, pomocou ktorej sa na základe pozorovania vytvára buď záver – zovšeobecnenie alebo záver – hypotéza. V čom sa podstatne odlišujú je, že pre indukciu obvykle máme vždy len jeden záver – zovšeobecnenie, pre abdukciu obvykle vytvárame na základe premisy – teóriu celú množinu alternatívnych hypotéz. To taktiež znamená, že abdukcia musí mať ako svoju integrálnu časť nejakú procedúru výberu výslednej hypotézy z množiny alternatívnych hypotéz.

[2] Erich von Däniken (nar. 1935) je švajčiarsky publicista, ktorý napísal niekoľko veľmi populárnych kníh (preložených aj do slovenčiny, napr. „Spomienky na budúcnosť“), kde rozvíjal ideu, že ľudstvo v dávnej minulosti absolvovalo návštevu mimozemšťanov, ktorí usmernili jeho vývoj do súčasnej podoby.

Pristúpme najprv k všeobecnej formulácii indukcie, ktorá je založená na nasledujúcich pojmoch [xx]:

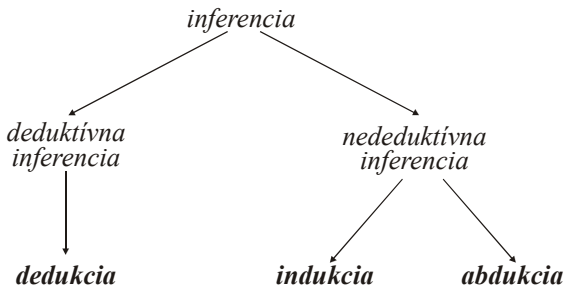
- pozorované dáta (príklady) Δ,
- teória (v pozadí) Γ, ktorá tvorí teoretický základ vysvetlenia pozorovaných javov a tvorby záveru,
- záver Φ, ktorý pomocou teórie Γ deduktívne vysvetľuje príklady Δ

Budeme postulovať, že tieto tri pojmy vyhovujú nasledujúcim podmienkam: (1) $\Gamma \not\models \Delta$, pozorované dáta nie sú deduktívne vysvetliteľné pomocou teórie Γ, ak by pozorované dáta priamo deduktívne vyplývali z teórie Γ, potom (2) $\Gamma \cup \Delta$, rozšírenie teórie o dáta je konzistentné, a (3) $\Gamma \cup \Phi \models \Delta$, pozorované dáta sú deduktívne vysvetliteľné pomocou rozšírenia teórie o hypotézu. Potom hovoríme, že záver Φ ***induktívne vyplýva z teórie*** Γ.

Špecifikácia abdukcie vyžaduje taktiež rovnaké tri pojmy: dáta, teóriu a záver, ktorý teraz je substituovaný množinou alternatívnych hypotéz $\{\Phi\}$. Potom ***abduktívnym vysvetlením*** pozorovaní Δ je hypotéza $\Phi_{opt} \in \{\Phi\}$, ktorá najlepšie vysvetľuje spolu s teóriou Γ pozorovania Δ

$$\Phi_{opt} = arg \max_{\Phi' \in \{\Phi\}} \textbf{\textit{vhodnosť'}}(\Phi')$$

kde ***vhodnosť'***$(\bullet)$ je funkcia, ktorá ohodnocuje vhodnosť alternatívnych hypotéz.



Obrázok 11.6. Klasifikácia inferenčných módov. V prvom najvšeobecnejšom prístupe delíme módy inferencie na deduktívne a nededuktívne. V druhom prístupe, nededuktívne módy sa delia na indukciu a abdukciu.

Z týchto dvoch všeobecných špecifikácií indukcie a abdukcie vyplýva, že medzi indukciou a abdukciou je len malý „formálny" rozdiel, a to ten, že pri abdukcii sa explicitne uvažuje množina alternatívnych hypotéz, zatiaľ čo pri indukcii sa uvažuje len jeden záver (hypotéza), pozri obr. 11.6.

Klasifikačné schéma nededuktívnej inferencie môže mať dve limitné podoby. Prvá je taká, ktorá chápe abdukciu ako špeciálny prípad indukcie, druhá je opačná, chápe indukciu ako špeciálny prípad abdukcie.

11.5 Abdukcia v každodennom živote

Ako už bolo uvedené v úvodnej časti tejto kapitoly, nededuktívna inferencia je súčasťou nášho každodenného života. Našu pozornosť teraz usmerníme na význam abdukcie. Uvedieme tri jednoduché príklady, ktoré dobre ilustrujú jej význam v našom živote, pri interpretácii a pochopení prostredia v ktorom žijeme a komunikácie v prirodzenom jazyku s inými ľuďmi.

1. Uvažujme nasledujúci rozhovor:

A: *Prečo sme zaparkovali pri tejto benzínovej pumpe*?

B: *Pretože máme skoro prázdnu benzínovú nádrž.*
A: *Na základe čoho takto usudzuješ*?
B: *Pretože indikátor upozorňuje na skutočnosť, že benzínová nádrž je skoro prázdna. Nemám dôvod domnievať sa, že indikátor je pokazený a taktiež už uplynul dlhý čas od vtedy, čo som naposledy naplnil nádrž benzínom.*

Za určitých okolností, „skoro prázdna benzínová nádrž“ je najlepšie možné vysvetlenie skutočnosti, že svieti indikátor stavu paliva v nádrži. Môžeme prijať aj iné vysvetlenie (napr. že indikátor je pokazený), ale toto vysvetlenie je podstatne menej prijateľné ako akceptované vysvetlenie (nádrž je prázdna).

2. Pri odchode z práce autom sme si všimli, že už dlhšiu dobu za nami ide červené auto. Urobíme dve neočakávané zabočenia autom do iného smeru, po chvíli zistíme, že za nami je opäť červené auto, aj keď už nie je tak blízko ako predtým. Naraz si spomenieme, že sme zabudli v práci svoj notebook, tak sa rozhodneme vrátiť sa preň do práce. Po zložitom manévrovaní sa nám podarí dostať auto na trasu do našej práce. Po niekoľkých okamžikoch znovu spozorujeme, že za nami je rovnaké červené auto. Prvá naša hypotéza pre vysvetlenie týchto skutočností je, že sme sledovaní; avšak nevieme si predstaviť dôvody toho, že by sme mali byť sledovaní. Preto prijmeme druhú alternatívnu hypotézu, že sa jedná o neuveriteľnú náhodu.

3. Príklad zo súdu. Predpokladáme, že výpoveď svedka je pravdivá, akceptovanie tejto hypotézy je založené na nasledujúcich dôvodoch: (1) usudzujeme, že hovorí pravdu, pretože mu veríme, (2) usudzujeme, že je tomu tak, pretože výpoveď v ktorej popísal danú situáciu bola veľmi dôveryhodná. Naša viera v dôveryhodnosť výpovede je založená na našich záveroch, že pokladáme výpoveď svedka za najlepšie akceptovateľné vysvetlenie (hypotézu) danej situácie. Naša dôvera vo výpoveď sa podstatne zmení, ak začneme pokladať za prijateľné iné hypotézy pre vysvetlenie výpovede svedka, napríklad, že svedok chce si získať našu dôveru. Pre tento typ usudzovania je podstatná skutočnosť, že pri vysvetľovaní pozorovaní nepoužívame len možné hypotézy, ale „optimálne“ hypotézy, ktoré najlepšie vysvetľujú skutočnosti v porovnaní s inými hypotézami.

11.6 Vzťah medzi indukciou a abdukciou

V predchádzajúcej časti tejto kapitoly sme zaviedli klasifikáciu inferencie na dve široké podtriedy deduktívnu a nededuktívnu, pričom nededuktívna inferencia sa (podľa Pierceho) rozdeluje na induktívnu a abduktívnu. Moderný pohľad na nededuktívnu inferenciu je taký, že veľmi nerozlišuje medzi indukciou a abdukciou, v tomto odseku uvedieme niektoré argumenty [xx] podľa ktorých je indukcia špeciálny prípad abdukcie. Uvažujme nasledujúce jednoduché inferenčné schéma indukcie – zovšeobecnenia

$$\frac{\textit{všetky pozorované } A \textit{ sú } B}{\textit{všetky } A \textit{ sú } B}$$

alebo pomocou kvantifikátorov

$$\frac{(\forall x \in M_0)\, A(x) \Rightarrow B(x)}{(\forall x \in M)\, A(x) \Rightarrow B(x)}$$

kde $M \supseteq M_0$. Zovšeobecnenie „*všetky A sú B*“ chápeme ako hypotézu, ktorá je lepšia a prijateľnejšia než iné možné hypotézy (napr. že niekto riadi naše experimenty tak, aby .sme sa domnievali, že všetky *A* sú *B*). Samozrejmé, môže nastať taká úplne nová situácia, že iné hypotézy sa stanú prijateľnejšími, potom nemusíme robiť zovšeobecnenie „*všetky A sú B*“ na základe predchádzajúcej korelácie medzi objektmi *A* a *B*. Ukázali sme teda, že indukcia môže byť chápaná ako špeciálny prípad abdukcie, ktorý je platný vtedy, keď hypotéza „zovšeobecnenia“ je prijateľnejšia než ako iné hypotézy.

Vzťah medzi indukciou a abdukciou je o mnoho problematickejší hlavne vtedy, keď sa snažíme zovšeobecňovať na základe malej vzorky výberu pozorovaní (napr. vtedy, keď $|M_0| \ll |M|$, t.j. počet pozorovaní je podstatne menší ako veľkosť množiny *M* objektov zovšeobecnenia). V tomto prípade môže existovať mnoho alternatívnych hypotéz, ktoré rovnako dobre vysvetľujú pozorované javy. Sledujme nasledujúci príbeh: Fero bol dvakrát v reštaurácii „Mariova Pizzeria“, po každej návšteve mal žalúdočné ťažkosti. Vo všeobecnosti, Fero mával občas žalúdočné ťažkosti, ale nie príliš často. Čo môžeme povedať o súvislosti medzi jedením pizzi a žalúdočnými ťažkosťami u Fera? Takmer nič, pretože počet návštev pizzerie je veľmi malý, než aby sme boli schopní vyvodiť nejaký seriózny záver z našich pozorovaní. Predpokladajme, že Fero navštevoval Mariovu Pizzeriu naďalej, pričom po 80 návštevách mal len 12 krát žalúdočné ťažkosti. Aké môžu byť teraz vyvodené závery pre vzťah medzi navštevovaním Mariovej pizzerie a Ferovými žalúdočnými ťažkosťami? K vysvetleniu môžeme použiť abduktívnu inferenciu založenú na dvoch hypotézach:

1. **hypotéza.** Súvislosť medzi jedením pizze a žalúdočnými ťažkosťami je náhodná (napr. vírovou infekciou, ktorej sprievodné javy boli aj žalúdočné ťažkosti).
2. **hypotéza.** Existuje určitá súvislosť medzi Ferovými návštevami Mariovej pizzerie a jeho žalúdočnými ťažkosťami, Fero je alergický na Mariovu špeciálnu omáčku.

Na základe vysokého počtu Ferových návštev Mariovej pizzerie, môžeme vylúčiť induktívne zovšeobecnenie ako neprijateľné a akceptovať druhú hypotézu, podľa ktorej Ferove žalúdočné ťažkosti sú spôsobené Mariovou špeciálnou omáčkou, ktorú si občas dá ako prílohu na pizzu so slimákmi.

Zhrnutie

Podľa Pierceho klasifikácie sa inferencia rozdeľuje na tri základné mechanizmy – dedukciu, indukciu a abdukciu. ***Dedukcia*** je také odvodzovanie nových poznatkov, ktoré je založené na logických zákonoch typu *modus ponnens* a pod., kde pravdivosť výsledku je plne určená len pravdivosťou východzích premís. Pri dedukcii nedochádza k zvyšovaniu informácie, výsledky obsahujú len takú informáciu, ktorí už bola obsiahnutá v premisách. ***Indukcia*** je taká inferencia, ktorá zovšeobecňuje pozorovania. Verifikácia indukcie znamená hľadanie ďalších pozorovaní, ktoré podporujú zovšeobecnenie, falzifikácia indukcie spočíva v náleze aspoň jedného pozorovania, ktoré vyvracia zovšeobecnenie. Podľa filozofa Karla Poppera, falzifikácia patrí medzi základné hybné sily rozvoja vedy. ***Abdukcia*** je inferenčný mód pomocou ktorého sa vytvárajú hypotézy vysvetľujúce pozorovanie. Z množiny alternatívnych hypotéz vyberáme tú, ktorá je najprijateľnejšia. Dôkladnejšia analýza ukazuje, že medzi indukciou a abdukciou je len veľmi malý rozdiel, indukcia môže byť považovaná za zvláštny prípad abdukcie.

Úlohy

Úloha 11.1. Majme formálny systém špecifikovaný teóriu obsahujúcou tieto pravidlá
(1) $(\forall x)(\forall y)(\forall z)\left[give(x,y,z) \Rightarrow own(y,z)\right]$, *typ x* : darca; *typ y* : príjemca; *typ z* : vec,
(2) $(\forall y)(\forall z)\left[buy(y,z) \Rightarrow own(y,z)\right]$, *type y* : kupujúci; *typ z* : vec,
(3) $(\forall y)(\forall z)\left[own(y,z) \Rightarrow can\_sell(y,z)\right]$, *typ y* : vlastník; *typ z* : vec,
a troma pozorovaniami
(1) $give(John,Mary,book)$, (2) $own(Mary,ball)$ (3) $buy(John,something)$.
Aké sú deduktívne závery z tohto systému?

Úloha 11.2. Študujme úlohu zistenia vzorca, ktorý určuje sumu prvých *n* prirodzených čísel $\Sigma(n) = 1+2+3+...+n$, kde pre prvé prirodzené čísla táto suma má tieto hodnoty: $\Sigma(1) = 1$, $\Sigma(2) = 3$, $\Sigma(3) = 6$, ... Postulujme, že táto suma pre všeobecné *n* má tvar

$$\Sigma(n) = \frac{n(n+1)}{2}$$

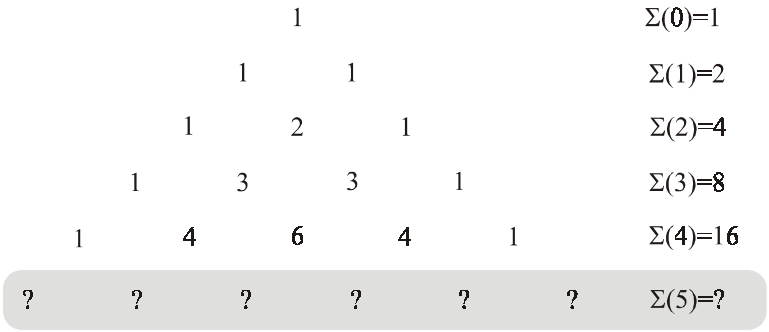
Jednoduchými verifikáciami sa presvedčíme, že tento vzorec je platný, žiaľ túto skutočnosť nemôžeme pokladať za dôkaz formuly. Dokážte, že platnosť formule pre $\Sigma(n)$ implikuje jej platnosť $\Sigma(n+1)$, t.j. platí pre každé prirodzené *n*.

Úloha 11.3. Váš priateľ Fero vám oznámil, že sa mu podaril odvodiť vzorec na konštrukciu *n*-tého prvočísla a_n pomocou predchádzajúcich troch prvočísiel

$$a_n = -6\, a_{n-3} + 3\, a_{n-2} + 2\, a_{n-1}$$

Žiaľ, nevie ho dokázať. Pokúste sa mu ho falzifikovať.

Úloha 11.4. Na obrázku je znázornený Pascalov trojuholník, ktorý obsahuje v každom *n*-tom riadku *n* prirodzených čísel. Doplňte čísla v 6. riadku Pascalovho trojuholníka a čomu sa rovná suma jeho členov. Pokúste sa zovšeobecniť indukciou vaše pozorovania o Pascalovom trojuholníku pre *n*-tý riadok, ako sú definované .



Úloha 11.4. Ráno v škole zistíte, že vaša peňaženka je prázdna, aj keď večer bolo v nej ešte niekoľko sto korún. Navrhnite niekoľko hypotéz, ktoré vysvetľujú skutočnosť, že ráno bola peňaženka prázdna, pokúste sa určiť z daných hypotéz najpravdepodobnejšiu hypotézu.

Otázky na opakovanie

Otázka 11.1. Aká je Pierceho klasifikácia inferencií?

Otázka 11.2. Ako je špecifikovaná dedukcia? Dochádza k nárastu informácie pri deduktívnej inferencii?

Otázka 11.3. Ako je špecifikovaná indukcia? Čo je verifikácia a falzifikácia záveru získaného indukciou?

Otázka 11.4. Ako je špecifikovaná abdukcia? Akým spôsobom vyberáme výslednú hypotézu?

Otázka 11.5. Aká je všeobecná formulácia indukcie a abdukcie, aký je rozdiel medzi nimi?

Otázka 11.6. Ako je špecifikovaná dedukcia? Dochádza k nárastu informácie pri deduktívnej inferencii?

12 Teória rozhodovania s obmedzenou racionalitou

Problém riadenia zložitých systémov, ktorých modely dobre nepoznáme (ak poznáme, tak len vo veľmi nedokonalej forme), sú tradičným problémom kybernetiky[1]. Metódy modernej informatiky (menovite umelej inteligencie) sú schopné poskytnúť nový pohľad na tento štandardný problém kybernetiky, poskytujú nielen jeho novú interpretáciu, ale umožňujú aj jeho implementáciu pomocou neurónových sietí. Z dôvodov neutrality našich záverov budeme používať termín „agent“ na označenie inteligentného jedinca s kognitívnym orgánom (mozgom), ktorého cieľom je riadiť systém (stroj, prostredie, pohyb stroja, zložitý ekonomický systém,...). Kognitívny orgán agenta (informatický termín pre ľudský mozog) je schopný vykonávať zložité kognitívne aktivity, akými sú učenie, plánovanie, vyhodnocovanie, riešenie problémov, vnímanie okolia, a pod. Tieto schopnosti kognitívneho orgánu sa využívajú pri tvorbe vnútorných modelov riadeného systému, ktoré tvoria ústrednú zložku využívanú pri riadení zložitých systémov alebo pri rozhodovaní.

Systém (napr. fabrika, banka, systém zásobovania veľkomesta potravinami, vysoká pec na výrobu železa,...) je špecifikovaný vstupom, výstupom a vnútornými stavmi. Agent prostredníctvom svojich orgánov vnímania rozpoznáva aktuálne stavy systému a taktiež môže priamo ovplyvňovať vstup systému. *Cieľom riadenia systému (t.j. úloha pre agenta) je pre dané vnútorné stavy systému meniť vstup systému tak, aby boli dosiahnuté požadované hodnoty výstupu* (pozri obr. 12.1).

Obrázok 12.1. (A) Komplex agent – systém. Agent transformuje vstupný zámer a stav systému na akciu, ktorá tvorí vstup pre systém. Odozva systému na akciu je zmena jeho stavu a výsledok. Zámer a akcia sú pre agenta *blízke premenné*, ktoré je schopný bezprostredne kontrolovať, zatiaľ čo stav systému a výsledok sú *vzdialené premenné*, ktoré môže ovplyvňovať len nepriamo prostredníctvom akcie. (B) V prípade, že zámer agenta je nemenný a agent je schopný predvídať výsledok systému, potom komplex agent-systém môže byť zredukovaný o dve externé premenné. Medzi agentom a systémom vtedy existuje len spätnoväzobná dvojica veličín akcia - stav.

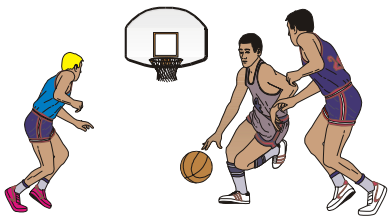
[1] *Kybernetika* je veda o riadení strojov, zariadení a systémov. Vznikla v r. 1947, keď Norbert Wiener takto pomenoval disciplínu ležiacu na rozhraní elektrického inžinierstva, matematiky, biológie a neurovedy. Prvé aplikácie kybernetiky boli zamerané na riadenie fyzikálnych systémov (riadenie delostreleckej paľby, návrh elektrických obvodov, pohyb jednoduchých robotov). V súčasnosti sa mnohé oblasti, ktoré boli pôvodne predmetom záujmu kybernetiky, presunuli do informatiky a umelej inteligencie.

Zámer agenta (určený vstupom – zámerom), určuje ciele, ktoré chce agent dosiahnuť. V prípade, že agent má konštantný zámer (nemennosť svojich cieľov), potom agent môže mať len jeden vstup, ktorý je identický so stavom systému. Podobne, ak výstup systému – výsledok je ignorovaný v dôsledku toho, že agent je schopný rekognoskovať okrem stavu systému aj výsledok. Týmto spôsobom sa pôvodná dvojica agent – systém zredukuje len na vnútorné väzby medzi agentom a systémom, externé väzby sú v tomto priblížení ignorované.

Aká je úloha učiteľa (supervisora) v tomto rozšírenom prístupe k učeniu agenta interagujúceho so systémom? V štandardnom prístupe k učeniu agenta učiteľ vyhodnocuje výstupy agenta, ktoré je agent schopný ovládať svojím výstupom (blízkou premennou). V ďalej študovanom rozšírenom prístupe učiteľ vyhodnocuje až výsledok systému, ktorý agent môže ovládať len sprostredkovane pomocou akcií. Týmto dochádza k podstatnému posunu významu učenia, v štandardnom prístupe učiteľ hodnotil bezprostredný výstup agenta, ktorý je schopný agent ovládať svojou intenciou, v rozšírenom prístupe učiteľ hodnotí vzdialený výstup systému, ktorý agent môže ovládať len sprostredkovane pomocou svojich akcií.

Niekoľko „psychologických“ poznámok o priebehu učenia sa v komplexe agent – systém. Proces učenia je možné rozdeliť na dve etapy. V *prvej etape* si agent pomocou svojho kognitívneho orgánu vytvára tzv. vnútorný model systému, učí sa predvídať pre daný stav výstupy na základe svojich akcií. V *druhej etape* agent adaptuje svoj kognitívny orgán (vzhľadom k vnútornému modelu) aby mohol transformovať predvídané výstupy pomocou svojich akcií. Podobný dvojetapový prístup bude použitý aj pri našom štúdiu rozšíreného učenia komplexu agent-systém; v prvej etape sa vytvorí priamy model systému a v druhej etape kognitívny aparát agenta bude adaptovaný vzhľadom k zafixovanému priamemu systému (vytvorenému v predchádzajúcej etape učenia). V tejto druhej etape vzniká inverzný model systému, pomocou ktorého agent dokáže predvídať akcie pre požadované výstupy systému.

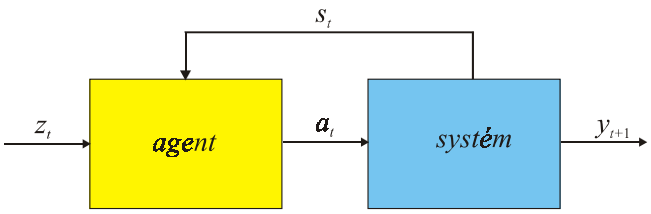
Naznačený dvojetapový model učenia, kedy sa agent učí riadiť komplexný systém pomocou svojich akcií a zámerov, má dôležité postavenie v informatike a v kognitívnej vede. Môže byť užitočný pre pochopenie reflexívnych aktivít človeka pri riadení takých komplexných zariadení akými je napr. auto alebo lietadlo. Neobyčajne veľká rýchlosť, akou človek reaguje pri riadení komplexného zariadenia, podporuje hypotézu, že človek sa učí riadiť systém v dvoch etapách. V prvej etape sa vytvára v mozgu priamy model systému, ktorý dokáže s dobrou presnosťou predpovedať výstupy zariadenia na základe aktivít človeka. Tento priamy model má veľký význam pre vznik reflexívneho riadenia celého zariadenia na základe zámeru a aktuálneho výstupu a stavu systému. V druhej etape učenia už sa človek nemusí neustále obracať na systém ako taký, predvída potrebné akcie ako odozvu na požadovaný výstup pomocou inverzného modelu. Toto umožňuje vznik reflexívneho (veľmi rýchleho) „modulu“ v mozgu pre riadenie zložitých zariadení.



Obrázok 12.2. Jednoduchý ilustračný príklad dvojice agent – systém. Agent je reprezentovaný hráčom s loptou, zatiaľ čo systém je reprezentovaný ihriskom a polohou ostatných hráčov. Intenciou hráča – agenta je hodiť loptu do koša, výstupom systému sú audiovizuálne efekty doprevádzajúce hodenie lopty do koša.

Ako jednoduchý príklad študovaného komplexu uvedieme hráča basketbalu (agent), ktorý na basketbalovom ihrisku obsadenom inými hráčmi (systém) má intenciu hodiť loptu do

koša (pozri obr. 12.2). Systém je charakterizovaný stavom, ktorý je reprezentovaný polohami iných hráčov, pričom výstup zo systému – výsledok sú audiovizuálne efekty doprevádzajúce hodenie lopty do koša. Agent – hráč je schopný rekognoskovať stav systému, t.j. postavenia ostatných hráčov, túto skutočnosť svojím kognitívnym orgánom pretransformuje na príkazy pre svoje motorické centrá, pomocou ktorých vykoná sekvenciu elementárnych pohybov. Toto môže byť chápané ako akcia agenta, ktorá mení stav systému. Vyššie uvedené dvoj-etapové učenie sa agenta v komplexe so systémom znamená, že agent – hráč si v prvej etape vytvára model systému (basketbalovej hry), v ktorom je schopný viac-menej presne predpovedať dôsledky svojho pohybu – akcie na stav systému, ako naň zareagujú ostatní hráči. V druhej etape už agent neočakáva neustále reakciu systému na svoju akciu, ale pomocou svojho vnútorného priameho modelu adaptuje svoj kognitívny orgán (vzniká inverzný model) na zmenu stavu systému a svoj zámer transformuje na akciu. Vo všeobecnosti môžeme očakávať dobré výsledky aj vtedy, keď vnútorný priamy model systému nie je celkom presný, adaptáciou kognitívneho orgánu sa môžu odstrániť možné nedostatky tohto modelu.



Obrázok 12.3. Dvojica zložená z dvoch častí: z agenta, ktorý transformuje intenciu a stav systému na akciu, a zo systému, ktorý transformuje akciu na výsledok (pre daný stav).

Predpokladajme, že zámer (z), akcia (a), stav systému (s) a výsledok (y) sú špecifikované pre každý časový krok t. Dvojica agent a systém je schematicky reprezentovaná na obr. 12.3. Tento obrázok je v podstate totožný s obrázkom 12.1, v tejto novej schematickej reprezentácii sú explicitne uvedené jednotlivé premenné komplexu a ich časové indexy (predpokladáme diskrétny čas). Ako už bolo poznamenané, cieľom agenta je riadiť výsledok (výstup systému) pomocou svojich akcií. Pretože táto premenná je pre agenta „vzdialená“, nemôže ju bezprostredne ovplyvňovať, učenie sa dvojice agent – systém sa označuje ako *učenie s dohľadom*. Tréningová množina pre tento prípad má tvar $A_{train} = \left\{ z_t \middle/ \boldsymbol{y}^*_{t+1} ; t = 1,2,...,t_{max} \right\}$, kde $\boldsymbol{y}^*_{t+1}$ je požadovaný výstup pre zámer z_t. V tomto prípade si môžeme predstaviť, že riadime auto a v každom časovom okamihu máme predpísané, akú rýchlosť máme dosiahnuť. Táto množina spolu so špecifikovaným počiatočným stavom systému s_0 tvorí základ adaptácie s učiteľom (ktorý poskytuje požadované výsledky).

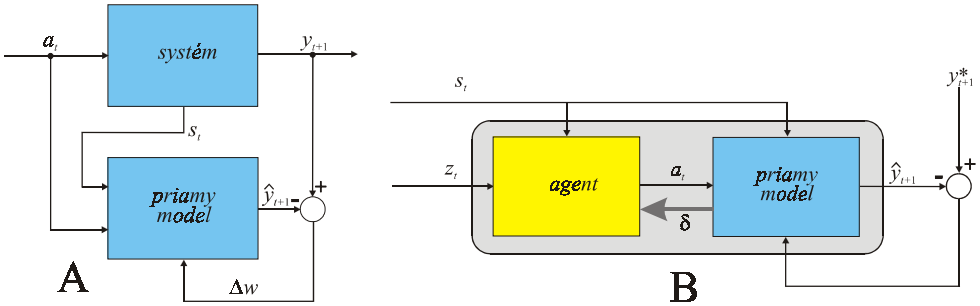
Poznamenajme, že konštrukcia tejto tréningovej množiny je pomerne problematická a reprezentuje netriviálny problém. Existencia tréningovej množiny implikuje existenciu učiteľa, ktorý je schopný rozfázovať pohyby hráča basketbalu na elementárne zložky a každú ohodnotiť. Je takmer nepredstaviteľné, že by tréner basketbalu učil hrať basketbal týmto spôsobom. Preto sa obrátiť na inú možnosť, vhodnými tréningovými metódami vedie hráčov k tomu, aby si v rámci svojho kognitívneho aparátu vygenerovali model hry, pomocou ktorého budú schopní vnútornou inferenciou vyvodiť dôsledky svojej akcie na systém, čo je podstatne rýchlejšie a hlavne efektívnejšie, než sa neustále obracať svojimi percepčnými orgánmi na systém.

12.1 Priamy model systému

Tento model je chápaný ako parametrická funkcia[2] *G,* ktorá zobrazuje akciu a_t a stav systému s_t na výsledok označený $\hat{y}_{t+1}$ (pozri obr. 12.4, diagram A).

$$\hat{y}_{t+1} = G(a_t, s_t; w)$$

Pomocou modelu systému môžeme vytvoriť tzv. komplex, kde systém je nahradený jeho priamym modelom, pričom stavy systému sa pokladajú za externý vstupný parameter, pozri obr. 12.4, diagram B). Učenie sa tohto komplexu môže prebiehať v podstate štandardnou metódou s dohľadom, pričom požadované výstupné aktivity poskytuje priamo systém. Je však potrebné poznamenať, že konštrukcia priameho modelu musí predchádzať učenie sa komplexu, pričom výsledný priamy model nemusí byť perfektný. V teórii riadenia je zvykom priamy model nazývať *identifikácia systému*.

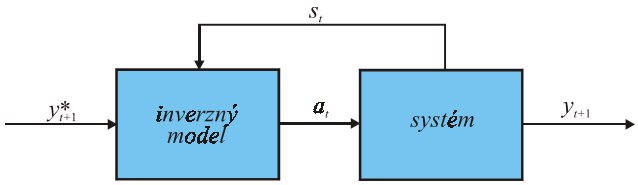


Obrázok 12.4. (A) Priamy model systému je chápaný ako parametrická funkcia, ktorá zobrazuje intenciu a stav na výsledok. (B) Modifikovaná sústava agent – systém, kde systém je nahradený jeho priamym modelom. Učenie sa tejto dvojice je založené na predpoklade, že priamy model je zafixovaný, adaptuje sa len kognitívny orgán agenta. Rozdiel medzi vypočítaným výsledkom a požadovaným sa nechá prebehnúť" cez priamy model, vyprodukovaný vektor „chýb" δ je vstupom do učenia sa agenta.

Obráťme teraz našu pozornosť na učenie sa komplexu zloženého z agenta (jeho kognitívneho orgánu určeného parametrami ***w***) a z priameho modelu systému. Vstupom do komplexu je dvojica tvorená intenciou i a stavom s systému, výstupom je výsledok $\hat{y}$, účelová funkcia má v tomto prípade tento jednoduchý tvar

$$E(w) = \frac{1}{2}(\hat{y} - y^*)^2$$

Pomocou derivácie tejto účelovej funkcie podľa parametrov kognitívneho orgánu zisťujeme, ako veľmi sa má zmeniť akcia agenta. Táto zmena musí byť tiež úmerná „chybe" δ vygenerovanej priamym modelom v bloku kognitívneho orgánu agenta (pozri obr. 12.4, diagram B).



Obrázok 12.5. Sériové zapojenie inverzného modelu spolu so systémom, požadovaný výstup tohto zapojenia je identické zobrazenie, t.j. $y^*_{t+1} = y_{t+1}$.

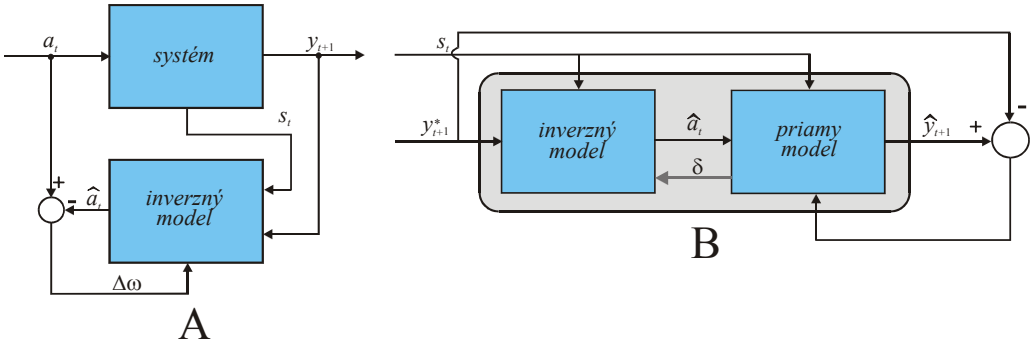
12.2 Inverzný model systému

[2] Parametrická funkcia $f(x,w)$, obsahuje okrem nezávisle premennej x aj parameter w, pomocou ktorého môžeme modifikovať vlastnosti funkcie f.

Inverzný model systému je taký vnútorný model, ktorý produkuje akciu a_t ako odozvu na aktuálny stav systému s_t a požadovaný výsledok y^*_{t+1}. Tento model je definovaný tak, že poskytuje identické zobrazenie, ak je sériovo vložený pred systém (okolie), pozri obr. 12.5. V tejto zostave môže byť inverzný model chápaný ako nejaké riadiace zariadenie, ktoré kontroluje systémový blok; požadované výstupy zo systému tvoria vstupy do riadiaceho zariadenia, ktorého výstup je aktivita pre systém, ktorá zmení jeho stav

$$\hat{a}_t = F\left(y_{t+1}, s_t; \omega\right)$$

V tejto súvislosti si môžeme položiť otázku o jednoznačnosti určenia inverzného systému pomocou daného systému. Pretože systém ako taký netvorí jedno-jednoznačné zobrazenie aktivity na stav a výsledok, potom vo všeobecnosti inverzné zobrazenie k nemu nemôže byť určené jednoznačne. Ďalší všeobecný problém s inverzným modelom je, že jeho určenie pomocou diagramu A na obr. 12.6 nie je cieľovo orientované. To znamená, že nemusíme nájsť takú akciu, ktorá produkuje požadovaný výstup systému.

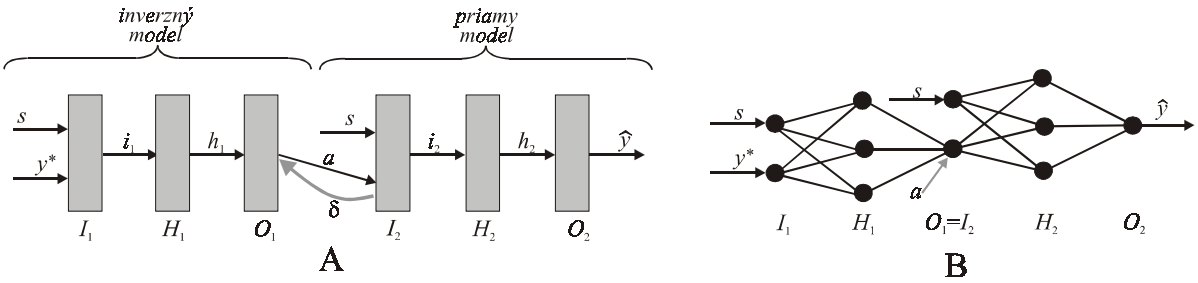


Obrázok 12.6. (A) Učenie sa inverzného modelu, pričom vstupy do modelu sú tvorené výstupmi systému. (B) Komplex tvorený z inverzného modelu a z priameho modelu. V tomto komplexe priamy model nemusí byť perfektný, učenie sa komplexu prebieha tak, že parametre priameho modelu sú zafixované, adaptácia prebieha len v rámci parametrov inverzného modelu. Rozdiel medzi vypočítaným výsledkom a požadovaným sa nechá „prebehnúť" cez priamy model, vyprodukovaný vektor „chýb" δ je vstupom do učenia sa inverzného modelu.

Učenie sa komplexu vytvoreného z inverzného a z priameho modelu (pozri obr. 12.6, diagram B) sa môže vykonať podobne ako učenie sa komplexu obsahujúceho agenta a priamy model v predchádzajúcej podkapitole, (pozri obr. 12.6., diagram B).

12.3 Konekcionistický model

Komplex zobrazený na obr. 12.6, diagram B, je interpretovaný tak, že každý blok modelu (inverzný a priamy) je reprezentovaný parametrickým zobrazením $F(\boldsymbol{\omega})$ resp. $G(\boldsymbol{w})$. V rámci konekcionistického prístupu tieto modely môžu byť reprezentované pomocou trojvrstvových neurónových sietí s priamym šírením signálu (poznamenajme, že tieto jednoduché neurónové siete už majú vlastnosť univerzálneho aproximátora, pozri podkapitolu 6.9). Učenie sa takejto neurónovej siete je štandardnou záležitosťou teórie neurónových sietí [xx].



Obrázok 12.7. (A) Neurónová sieť zostrojená zo 6 vrstiev neurónov, pričom prvé tri vrstvy (I_1, H_1 a O_1) reprezentujú inverzný model, ďalšie tri vrstvy (I_2, H_2 a O_2) reprezentujú priamy model (porovnaj s obr. 12.6, diagram B). Spoje medzi neurónmi zo susedných vrstiev I_1-H_1 , H_1-O_1, resp. I_2-H_2, H_2-O_2 sú realizované všetkými možnými spôsobmi. (B) Implementácia pomocou neurónovej siete obsahujúcej tri skryté vrstvy neurónov, na rozdiel od diagramu A sú vrstvy O_1 a I_2 stotožnené. V prvej etape učenia sa tohto komplexu sa adaptuje len pravá časť (zložená z vrstiev I_2, H_2 a O_2) , pričom ako vstup do tejto podsiete je dvojica (a,s), požadovaný výstup je y^*. V druhej etape budeme predpokladať, že váhové koeficienty z priameho modelu sú zafixované, adaptujú sa len váhové koeficienty z inverzného modelu.

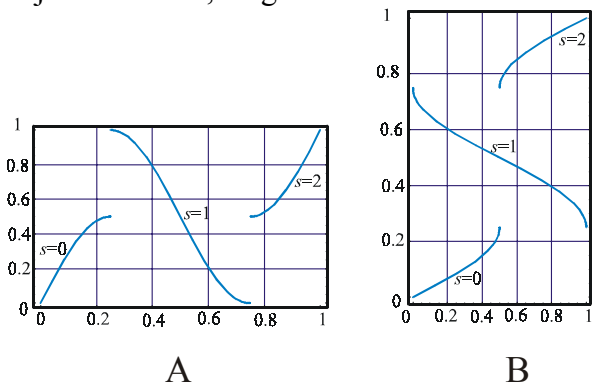
Neurónová sieť obsahuje dve časti, prvá sieť je tvorená vrstvami I_1, H_1 a O_1, táto sieť odpovedá inverznému modelu, druhá sieť, priradená priamemu modelu, je tvorená vrstvami I_2, H_2 a O_2. Učenie sa tohto komplexu dvoch neurónových sietí prebieha v dvoch etapách. V prvej etape učíme len samotnú neurónovú sieť odpovedajúcu priamemu modelu. Po skončení učenia sa je adaptovaná neurónová sieť vložená do komplexu a potom nasleduje druhá etapa učenia sa. V tejto etape predpokladáme, že váhové koeficienty časti priradenej priamemu modelu sú konštantné, t.j. menia sa len váhové koeficienty z inverzného modelu.

12.4 Ilustračný príklad

Majme "systémovú" funkciu určenú vzťahom (pozri obr. 12.8, diagram A)

$$y = f_s(a) = \begin{cases} \frac{1}{2} sin(2\pi a) & (s = 0,\ a < 1/4) \\ \frac{1}{2}\left(1 + sin(2\pi a)\right) & (s = 1,\ 1/4 < a < 3/4) \\ 1 + \frac{1}{2} sin(2\pi a) & (s = 2,\ a > 3/4) \end{cases}$$

Pre jednoduchosť budeme postulovať, že funkcie mimo svoj definičný obor sú nulové; tento predpoklad podstatne zjednoduší adaptáciu príslušnej časti neurónovej siete z jej implementácie znázornenej na obr. 12.8, diagram B.



Obrázok 12.8. (A) Priebeh funkcie $f_s(a)$ pre tri rôzne hodnoty stavového parametra s. (B) Priebeh inverznej funkcie $f_s^{-1}(y)$ taktiež pre tri rôzne hodnoty stavového parametra s. Poznamenajme, že funkcie mimo svojich oborov definície majú nulovú funkčnú hodnotu.

Funkcia $f_s(a)$ má tri rôzne inverzné funkcie v závislosti od funkčnej hodnoty a stavu (pozri obr. 12.8, diagram B):

$$a = f_s^{-1}(y) = \begin{cases} \frac{1}{2\pi} arcsin(2y) & (s=0,\ y \le 1/2) \\ \frac{1}{2} - \frac{1}{2\pi} arcsin(2y-1) & (s=1,\ 0 \le y \le 1) \\ 1 + \frac{1}{2\pi} arcsin(1-2y) & (s=2,\ y > 1/2) \end{cases}$$

Naším cieľom bude zostrojiť tak priamy model systému (ktorý transformuje akciu a a stav s na výsledok y), ako aj inverzný model (transformujúci stav s a výsledok y na akciu a). Ako už bolo poznamenané vyššie v tejto podkapitole, tento cieľ sa realizuje dvojetapovo. V prvej etape sa použije pravá časť komplexu na obr. 12.7, diagram B, ku konštrukcii priameho modelu obsahujúceho vrstvy I_2, H_2 a 0_2 , kde tréningová množina je realizovaná „stochasticky“ takto

$$s = random(3)$$

$$a = \begin{cases} \frac{1}{4} random[0,1] & (s=0) \\ \frac{1}{4} + \frac{1}{2} random[0,1] & (s=1) \\ \frac{3}{4} + \frac{1}{4} random[0,1] & (s=2) \end{cases}$$

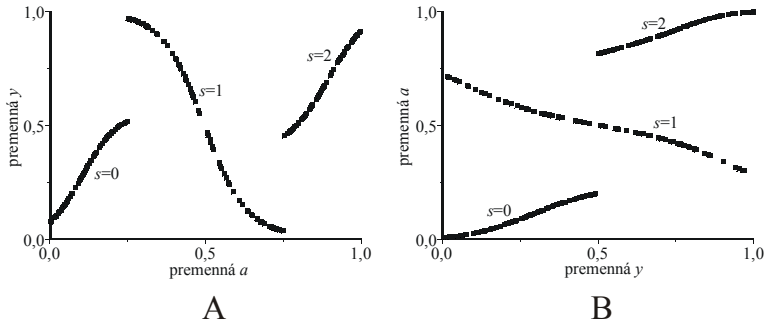
kde $random[0,1]$ je náhodný generátor čísel z intervalu $[0,1]$ s rovnomernou distribúciou, podobne $random(n)$ je náhodný generátor celých čísel z množiny $\{0,1,2,...,n-1\}$ s rovnomernou distribúciou, požadované hodnoty y sa spočítajú pomocou vyššie špecifikovanej funkcie $f_s(a)$. Adaptačný proces neurónovej siete je po určitom počte krokov ukončený, potom komplex v rámci druhej etapy je použitý na konštrukciu inverzného modelu pomocou ľavej časti komplexu (I_1, H_1 a O_1), pričom váhové koeficienty priameho modelu (pravej časti) sú nemenné. V tejto druhej etape adaptačný proces vyžaduje znalosť „chybového signálu“ δ, ktorý je vytvorený štandardnou metódou spätného šírenia signálu cez pravú priamu časť. Tréningová množina v druhej etape je taktiež realizovaná stochasticky

$$s = random(3)$$

$$y^* = \begin{cases} \frac{1}{2} random[0,1] & (s=0) \\ random[0,1] & (s=1) \\ \frac{1}{2} + \frac{1}{2} random[0,1] & (s=2) \end{cases}$$

(v tejto druhej etape už nepotrebujeme poznať požadované hodnoty akcií a).Výsledkom adaptačného procesu v druhej etape bude neurónová sieť priradená inverznému modelu (vzhľadom k danému priamemu modelu z prvej etapy), ktorá realizuje viac-menej presne

inverznú funkciu $f_s^{-1}(y)$, ktorú teraz nepotrebujeme poznať explicitne, bude sa používať len na vyhodnotenie presnosti vyprodukovanej inverznej neurónovej siete.



Obrázok 12.9. (A) Zobrazenie realizované neurónovou sieťou adaptovanou v prvej etape. Sieť reprezentuje priamy model systému (ktorý bol realizovaný pomocou zobrazenia (6.105)). (B) Zobrazenie realizované neurónovou sieťou adaptovanou v druhej etape, reprezentujúcou inverzný model systému (realizovaný pomocou zobrazenia (6.106)).

Komplex inverzného a priameho modelu (pozri obr. 12.6, diagram B) môže byť teraz chápaný ako zložená funkcia

$$F_s(y) = f_s\left[f_s^{-1}(y)\right] = \begin{cases} y & pre\ y < 1/2, \quad 0 \quad pre\ y \geq 1/2 \quad (s=0) \\ y & (s=1) \\ y & pre\ y > 1/2, \quad 0 \quad pre\ y \leq 1/2 \quad (s=2) \end{cases}$$

To znamená, že v tomto konkrétnom prípade komplex znázornený na obr. 12.7 je možné chápať ako jednotkové zobrazenie len v zúženom zmysle, a to na vhodných intervaloch. Táto skutočnosť je veľmi dôležitá pre správnu realizáciu adaptácie neurónovej siete tak v prvej, ako aj v druhej etape.

Získané numerické výsledky simulácie realizované pomocou doprednej neurónovej siete obsahujúcej 15 skrytých neurónov sú zosumarizované na obr. 12.9. Diagram A na tomto obrázku znázorňuje presnosť adaptovanej siete pre priamy model, porovnaním s obr. 12.8, diagram A zistíme, že priebehy jednotlivých funkcií $f_s(a)$ sú veľmi podobné priebehom produkovaným neurónovou sieťou. Podobne, diagram B na obr. 12.9 vyjadrujúci experimentálne priebehy inverzného modelu vypočítané príslušnou časťou neurónovej siete komplexu zodpovedá obr. 12.8, diagram B.

12.5 Riadenie s ohraničenou racionálnosťou (problém návštevnosti baru „*El Farol*“ v Santa Fe)

V predchádzajúcej časti tejto kapitoly bol študovaný problém riadenia zložitých systémov pomocou vnútorných modelov (priameho a inverzného) systému. Tieto modely sa zostrojujú pomocou učenia, ktoré môžeme chápať ako zovšeobecňovanie skúseností pomocou indukcie. Tento pohľad na riadenie systémov ako na problém induktívneho učenia je veľmi dôležitý v informatike a v príbuzných vedách. Otvára nové možnosti heuristických prístupov k problému riadenia zložitých systémov, ktorých matematické modely nepoznáme, alebo ak poznáme, tak len vo veľmi ohraničenej miere. V tejto súvislosti môžeme pristúpiť k zavedeniu pojmu „riadenie s ohraničenou racionálnosťou“, pod ktorým budeme rozumieť

taký typ riadenia, kde nepoznáme presný model riadeného systému, pomocou indukcie vytvárame vnútorný model systému.

Riadenie s ohraničenou racionálnosťou je založené na dvoch základných metódach logického usudzovania: indukcii a abdukcii (pozri predchádzajúcu 11. kapitolu). Indukcia a abdukcia sa objavuje všade tam, kde sa jedná o neuvedomelé (reflexívne) usudzovanie a interpretáciu vnemov prichádzajúcich z nášho okolia. Tak napríklad, pri interpretácii čítaného textu, reflexívne vytvárame množstvo abdukcií, ktoré sú potrebné k tomu, aby sme pochopili čítaný text. Podobná situácia je aj pri pochopení a interpretácii nášho okolia na základe našich vnemov, neustále si musíme vytvárať reflexívne rôzne hypotézy, pomocou ktorých interpretujeme naše okolie (vidíme na ulici cúvať auto, okamžite vytvoríme hypotézy, že auto chce zaparkovať). Abdukcia je dôležitá aj pri vedomom (reflektívnom) usudzovaní. Ako príklad na vedomé použitie abdukcie môžeme uviesť lekára, ktorý vytvára rôzne hypotézy o diagnóze pacienta, vyhodnocuje ich, aby získal najspoľahlivejšiu diagnózu – hypotézu, ktorá vysvetľuje pacientov zdravotný stav.

V súčasnej informatickej literatúre zameranej na štúdium riadenia zložitých ekonomických systémov, pre ktoré neexistuje matematicky model (alebo ak existuje, tak je veľmi nedokonalý), je často študovaný problém riadenia týchto systémov pomocou *stratégie*. Stratégia je súborom heuristických pravidiel, pomocou ktorých riadime zložitý systém na základe jeho predchádzajúcich stavov a výstupov. V tejto podkapitole budeme ilustrovať tento prístup pomocou úlohy nazývaného „*El Farol Bar*“, ktorú vymyslel W. B. Arthur [xx] ako ilustráciu rozhodovania s obmedzenou racionalitou.

> Bar[3] “El Farol” v Santa Fe každú sobotu večer uvádza írsku hudbu. Do baru sa zmestí asi 60 ľudí, pričom počet celkových záujemcov je okolo 100 ľudí. K tomu, aby sa poslucháči sobotňajšieho predstavenia vyhli návalu v baru, každý si hľadá svoju stratégiu (heuristiku), ako odhadnúť na základe predošlých predstavení návštevnosť aktuálneho predstavenia. Poslucháči medzi sebou nekomunikujú o tom, či navštívia, alebo nenavštívia bar.

Pokúsme sa túto úlohu naformulovať presnejšie. Nech časová rada sobotňajšej návštevnosti baru El Farol má tvar : ..... $x_{t-k}, x_{t-k+1}, ..., x_{t-1}$; kde posledné číslo x_{t-1} špecifikuje počet návštevníkov predstavenia ostatnú sobotu. Stojíme pred úlohou ako odhadnúť dnešnú návštevnosť (x_t) pomocou tohto časového radu. Predpokladajme, že dnešná návštevnosť je určená funkciou návštevností za ostatných k sobôt

$$x_t = f\left(x_{t-1}, x_{t-2}, ..., x_{t-k}\right)$$

kde k sa nazýva “veľkosť okna do minulosti”. Podobný prístup sa napr. využíva pri odhade kurzov valút na základe časovej rady kurzov danej meny. Tvar funkcie nie je určený vopred, vyjadruje sa pomocou funkcií, ktoré obsahujú mnoho parametrov (tzv. parametrická regresná analýza), pričom parametre sa určia tak, aby sa dostala čo najlepšia zhoda medzi vypočítanou a skutočnou návštevnosťou.

Druhý postup, ako predvídať aktuálnu hodnotu časového radu na základe „okna do minulosti“ sa uskutočňuje pomocou „stratégie“, ktorej základné myšlienky boli už prezentované v kapitole 11 venovanej počítačovým simuláciám kooperácie v sociálnych systémoch. Majme spoločenstvo 100 agentov, pričom každý agent je reprezentovaný stratégiou – binárnym vektorom

[3] Bar El Farol skutočne existuje v meste Santa Fe, štát Nové Mexiko, USA, dokonca existuje jeho domovská stránka na internete.

$$\boldsymbol{s} = \left(s_1, s_2, ..., s_8\right) \in \left\{0,1\right\}^8$$

kde jednotlivé zložky stratégie sú špecifikované nasledujúcou tabuľkou

#	*história*	#	*história*
s_1	(*UUU*)	s_5	(*CUU*)
s_2	(*UUC*)	s_6	(*CUC*)
s_3	(*UCU*)	s_7	(*CCU*)
s_4	(*UCC*)	s_8	(*CCC*)

Symbol *C* (*U*) znamená – *crowded – preplnený* (*uncrowded – nepreplnený*). Tak napríklad, história (*UUC*) je v čase *t* interpretovaná tak, že pre predchádzajúce tri časové okamžiky (soboty) platí

$$x_{t-3} \le 60\left(U\right), x_{t-2} \le 60\left(U\right), x_{t-1} > 60\left(C\right)$$

Jednotlivé zložky binárneho vektora stratégie ***s*** sú popísané takto (pozri hore uvedenú tabuľku)

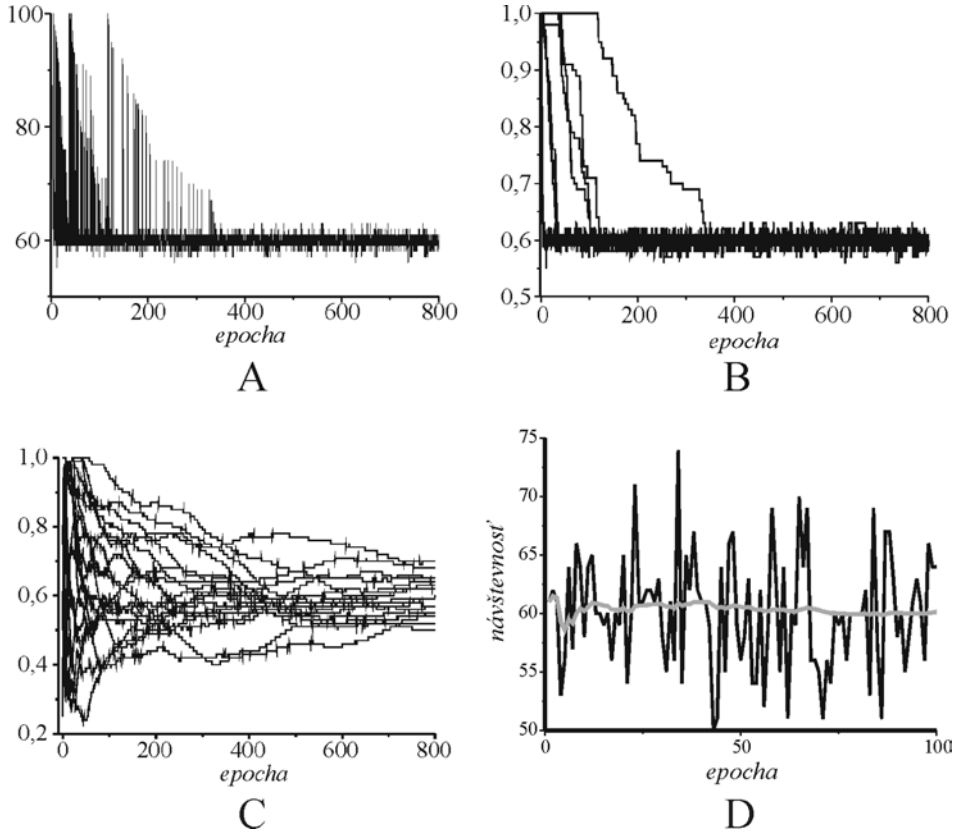
s_i=0 $\Rightarrow$ na základe *i*-tej histórie agent ***nenavštívi*** predstavenie

s_i=1 $\Rightarrow$ na základe *i*-tej histórie agent ***navštívi*** predstavenie

Pre lepšie pochopenie tohto spôsobu kódovania stratégie, študujme binárny vektor ***s***=(1,1,0,1,1,0,0,1), jednotlivé zložky stratégie majú tento význam

(1) s_1=1, história (*UUU*) $\Rightarrow$ agent navštívi bar
(2) s_2=1, história (*UUC*) $\Rightarrow$ agent navštívi bar
(3) s_3=0, história (*UCU*) $\Rightarrow$ agent nenavštívi bar
(4) s_4=1, história (*UCC*) $\Rightarrow$ agent navštívi bar
(5) s_5=1, históriu (*CUU*) $\Rightarrow$ agent navštívi bar
(6) s_6=0, história (*CUC*) $\Rightarrow$ agent nenavštívi bar
(7) s_7=0, história (*CCU*) $\Rightarrow$ agent nenavštívi bar
(8) s_8=1, história (*CCC*) $\Rightarrow$ agent navštívi bar

Nech $P_t = \left\{\boldsymbol{s}_1, \boldsymbol{s}_2, ..., \boldsymbol{s}_{100}\right\}$ je množina, ktorá obsahuje stratégie všetkých 100 agentov v epoche *t*. Táto množina nemôže byť homogénna (obsahujúca len jeden druh stratégie), ak by takéto boli, potom *všetci* agenti by sa na základe predchádzajúcej histórie rozhodli buď ísť alebo neísť v sobotu podvečer do baru El Farol na predstavenie. Z týchto dôvodov každý agent musí mať svoju vlastnú stratégiu ***s***, podľa ktorej sa rozhoduje, či ísť, alebo neísť do baru. Jedna epocha v simulácii znamená sobotný podvečer, keď sa každý agent populácie musí rozhodnúť, či ísť alebo neísť na predstavenie do baru El Farol. Aby v populácii samovoľne vznikol stav, kde približne 60 agentov navštívi každú sobotu bar, musíme pripustiť proces učenia stratégií. Na záver každej epochy agenti vyhodnocujú svoju úspešnosť alebo neúspešnosť pomocou procesu učenia. Tento proces spočíva v tom, že ak sa daný agent *rozhodol zle* na základe *i*-tej zložky svojej stratégie $\boldsymbol{s} = \left(..., s_i, ...\right)$ (t.j. podľa s_i mal ísť, ale bolo obsadené, alebo nemal ísť a bolo neobsadené), tak potom s malou pravdepodobnosťou ($P_{learn} = 0.01$) zmení túto zložku na jej komplementárnu hodnotu, $s_i - 1 \to s_i$. Epochy sa opakovali 800 krát, pričom bolo pozorované, že v populácii v priebehu asi 300 epoch samovoľne vzniknú také stratégie, že priemerná návštevnosť bola 60 ± 2, pozri obr. 12.10.



Obrázok 12.10. (A) Priebeh návštevnosti baru El Farol pre jednotlivé epochy. Algoritmus bol inicializovaný tak, že každý agent mal stratégiu (11111111), t.j. za každých podmienok v sobotu navštívil bar. Proces učenia stratégií umožnil vznik novým stratégiám, ktoré už poskytovali priemernú návštevnosť okolo 60 hostí (čo je optimálny výsledok). (B) Frakcie používania jednotlivých histórií v stratégiách celej populácie. Z grafu vyplýva, že sa ustálil „rovnovážny" stav, kde každá história (UUU, ..., CCC) sa využíva približné 60%. (C) Znázornenie priebehov frekvencie návštev jednotlivých agentov. Priemerná návštevnosť je približne 60%, avšak existujú agenti, ktorí na základe svojej stratégie navštevujú bar častejšie (zriedkavejšie). (D) Znázornenie priebehu návštevnosti baru, ak sa návštevníci rozhodujú tak, že si vygenerujú náhodné číslo $0 \le rand \le 1$, ak je toto spĺňa (nespĺňa) podmienku $rand \le 0.6$, tak bar navštívi (nenavštívi). Výsledkom tohto jednoduchého rozhodovacieho procesu je, že priemerná návštevnosť rýchlo konverguje k 60%, avšak na rozdiel od rozhodovania so stratégiou (pozri diagramy A a B) sú v tomto prípade veľmi veľké fluktuácie.

Ako interpretovať tieto výsledky? El Farol problém je typický reprezentant takých stochastických problémov, ktoré nemajú deterministický model. Vo všeobecnosti, jedna z najjednoduchších stratégií je taká, že každý agent si v sobotu podvečer náhodne vygeneruje číslo s rovnomernou distribúciou $0 \le rand \le 1$, ak toto číslo je menšie ako 0.6, potom navštívi bar, v opačnom prípade, ak číslo je väčšie ako 0.6), potom bar nenavštívi. Samozrejme, dlhodobý priemer návštevností bude konvergovať k 0.6 (čo je očakávaný výsledok), avšak s veľkými fluktuáciami (okolo ± 10), pozri diagram D, obr. 12.10. Iný, „jemnejší", prístup k riešeniu problému návštevnosti baru El Farol je ponechať agentom možnosť rozhodovať sa pomocou svojich stratégií, ktoré nie sú fixné – nemenné, ale sa menia na základe skúseností jednotlivých agentov. Tento proces modifikácie stratégií na základe predchádzajúcich skúsenosti môžeme chápať ako proces učenia agentov, v ktorom sa snažia pomocou indukcie a abdukcie vytvárať hypotézy, ktoré stále lepšie a lepšie predpovedajú návštevnosť baru v aktuálny sobotný podvečer. Samozrejme, toto „vynáranie" (emergencia) optimálnych stratégií v spoločenstve všetkých 100 agentov je možné len vtedy, ak každý agent sa správa „racionálne", t.j. problém sobotňajšej návštevy baru El Farol rieši pomocou nejakého svojho vnútorného modelu, ktorý nie je pevne daný, ale mení sa v priebehu agentovej histórie na základe jeho úspešnosti. To znamená, že sa jedná o veľmi

zložitý dynamický riadiaci problém, v ktorom sa súčasne rieši nielen adaptácia vnútorného modelu každého agenta, ale aj prispôsobenie tohto modelu k zmenám vnútorných modelov ostatných agentov (táto skutočnosť sa v umelej inteligencii nazýva „problém pohyblivého cieľa“, kde cieľ rozhodovania nie je fixne daný, ale je v čase premenlivý).

Zhrnutie

Riadení zložitých systémov a rozhodovanie v zložitých situáciách, keď nepoznáme modely týchto situácií, patrí medzi aktuálnu problematiku súčasnej informatiky a umelej inteligencie. Problém riadenia zložitých systémov je študovaný pomocou koncepcie dvojstupňového vnútorného modelu, ktorý si vytvára agent vo svojom kognitívnom orgáne. V prvej etape vzniká tzv. priamy model, ktorý s určitou vernosťou simuluje správanie sa systému, jeho odozvu na akciu pre dané vnútorné stavy, ktorá spočíva v zmene výstupu riadeného systému. V druhej etape si agent pomocou svojho priameho modelu vytvára tzv. inverzný model, ktorý rieši problém, ako je pre daný stav systému výstup transformovaný na požadované akcie. Oba tieto modely vytvárajú v kognitívnom orgáne jeden komplex, pomocou ktorého je agent schopný reflexívne riadiť systém (napr. auto alebo bicykel), nemusí sa neustále obracať na riadený systém a skúmať jeho reakcie na rôzne akcie a ich vplyv na výstup systému. Tento dvojstupňový model sa pomerne jednoducho implementuje pomocou neurónových sietí.

Ďalší (podobný) problém, ktorý bol študovaný v kapitole, bolo rozhodovanie s ohraničenou racionálnosťou. V tomto prípade, podobnom problému riadenia zložitých systémov, každý agent (alebo spoločenstvo agentov) musí riešiť problém rozhodovania v situáciách, ktoré nie sú jasne popísané a ktoré závisia aj od rozhodnutí ostatných agentov. V týchto prípadoch sa agenti nemôžu správať „racionálne“, t.j. maximalizovať svoje zisky, ale len s „ohraničenou racionálnosťou“, kedy si pomocou učenia (indukcie a abdukcie) vytvárajú stratégie (heuristiky) rozhodovania. Prístup bol ilustrovaný pomocou jednoduchého stochastického problému návštevnosti baru El Farol tak, aby sa minimalizoval výskyt preobsadenosti baru návštevníkmi (agentmi).

Úlohy

Úloha 12.1. Predstavte si hypotetickú situáciu, že ste generálnym riaditeľom fabriky na výrobu automobilov. Špecifikujte v tomto prípade výsledok systému, stav systému a pôsobenie akcie na zmenu týchto veličín. Aký je zámer agenta (generálneho riaditeľa) pri riadení systému, pomocou čoho odhaduje svoje akcie, aby bol splnený jeho zámer. Popíšte význam priameho modelu a aj inverzného modelu pre generálneho riaditeľa pri riadení systému.

Úloha 12.2. Preformulujte problém návštevnosti baru El Farol na prípad, keď spoločenstvo agentov opakovane využíva obmedzený ale obnoviteľný zdroj (napríklad kosenie spoločnej lúky, alebo výlov rýb v spoločnom teritóriu v oceáne).

Otázky na opakovanie

Otázka 12.1. Ako je definovaný komplex agent – systém, čo je cieľom riadenia systému?

Otázka 12.2. Aký je psychologický model riadenia systému v komplexe agent – systém?

Otázka 12.3. Ako je definovaný priamy model systému?

Otázka 12.4. Ako je definovaný inverzný model systému?

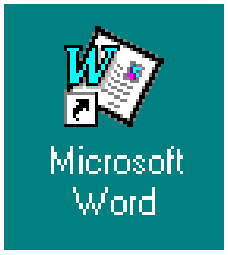
Otázka 12.5. Aký je konekcionistický model komplexu agent – systém, akým spôsobom prebieha jeho adaptácia?

Otázka 12.6. Čo je indukcia a abdukcia, aký je ich význam pri rozhodovaní s ohraničenou racionálnosťou?

Otázka 12.7. Ako je definovaný problém návštevnosti baru El Farol, čo je cieľom rozhodovania v tomto prípade?

Otázka 12.8. Ako je definovaná stratégia agenta v probléme baru El Farol, čo je učenie stratégie, aký má význam stratégia pri rozhodovaní?

13 MS Word 97



Písanie a spracovanie textových informácií patrí medzi najčastejšie sa vyskytujúce kancelárske práce. Texty najrôznejšieho druhu sú súčasťou odborných prác. Všade treba písať listy, protokoly, správy, dokumentáciu a ďalšie textové informácie.

Klasická tvorba dokumentu je v podstate postupnosťou niekoľkých krokov. Prvý z nich je vytvorenie rukopisu. V druhom kroku rukopis upravujeme - niektoré časti vynecháme alebo prehodíme na iné miesto, vsúvame do textu nové vety, formulujeme iné väzby slov, atď. V treťom kroku tvoríme čistopis, čo je vlastne prepis rukopisu na písacom stroji. Aj v tomto kroku robíme chyby, preklepy, vynecháme riadok, odstavec a pod. Odstránenie týchto chýb je zdĺhavé, ba aj nemožné. Niekedy treba celý dokument znovu pozornejšie prepísať.

Textové editory racionalizujú túto činnosť a urýchľujú celú prácu s dokumentom. Umožňujú spracovať dokument najprv na obrazovke počítača - prepisovať písmená, slová, vymieňať odstavce, mazať slová, riadky, meniť typ použitých písmen, meniť celkový vzhľad dokumentu. Tlačiareň vytlačí hotový, bezchybný a esteticky vhodne upravený dokument.

Editovaný dokument môžeme uložiť do súboru na disk a hocikedy v budúcnosti s ním pracovať. Na obrazovke sa nám zobrazí tak, ako sme ho predtým uložili. V súbore sú okrem samotného textu uložené aj informácie o formáte dokumentu.

Spomedzi dostupných textových editorov vyniká svojou všestrannosťou Word, ktorý zďaleka nie je určený len na jednoduché písanie a editovanie textu. Možnosti tohto programu sú omnoho väčšie ako väčšina bežných užívateľov vyžaduje.

Word okrem vlastností textových editorov dokáže nahradiť i jednoduchý databázový program a tabuľkový procesor, tj. vie spracovať jednoduchú databázu a prepočítavať jednoduché tabuľky a vytvoriť na základe ich údajov elegantné grafy. Okrem toho obsahuje účinné prostriedky na automatizáciu najrôznejších činností - od opráv bežných preklepov až po schopnosť samostatného naformátovania rozsiahlych dokumentov. Naviac dovoľuje do textu vložiť obrázok alebo napísať zložité matematické výrazy. Word disponuje veľkým užívateľským komfortom, je mimoriadne intuitívny.

Textový editor Microsoft Word 97 je súčasťou programového balíka Microsoft Office 97. Túto verziu považujeme za dostatočnú na výukové účely, vzhľadom k tomu. že väčšina užívateľov ani veľkú časť možností tohto editora nevyužije a funkcie novších verzií editora neprinášajú podstatné vylepšenie.

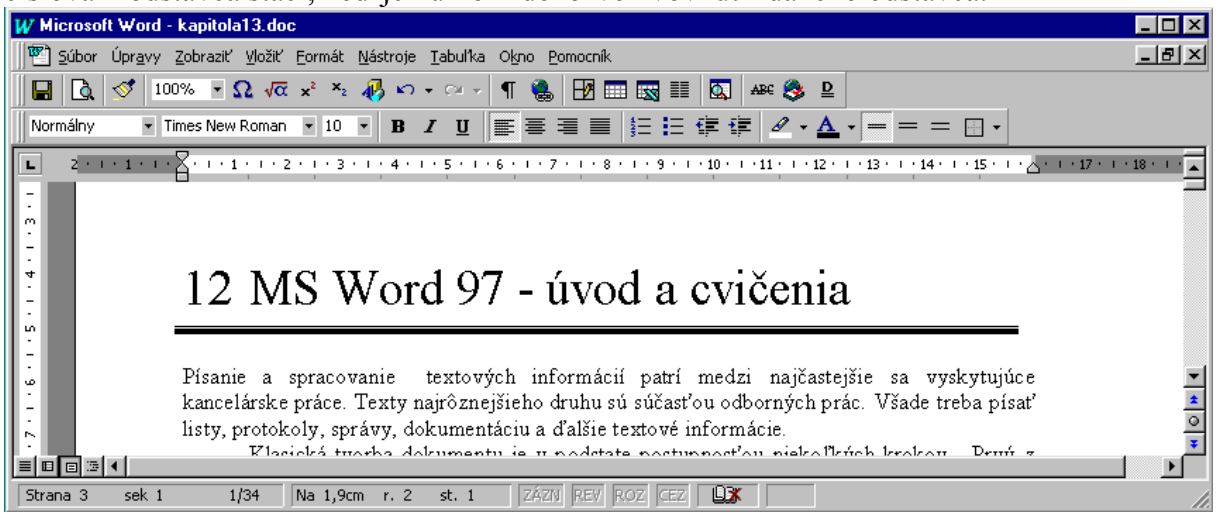
Aký je rozdiel medzi hladkým textom a editovaným textom?

```
Toto je hladký needitovaný text, ktorý neobsahuje rôzne rezy
písmen (fonty), formule a obrázky
```

Toto je editovaný text, **pričom použitý** rez písma je „*Times New Roman*“, obsahuje formule $s = \frac{1}{2} g t^2$ a taktiež obsahuje aj grafiku

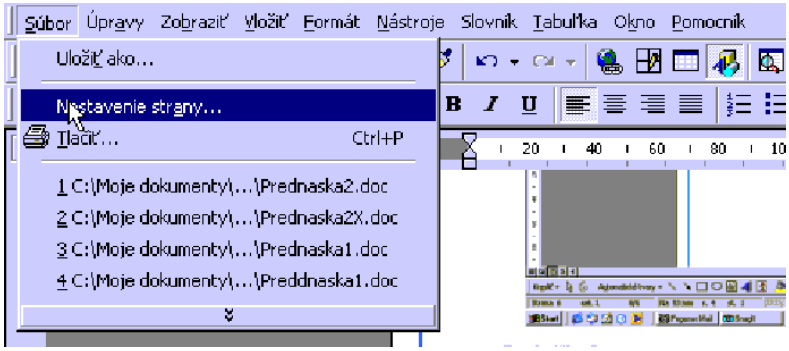
Hlavné okno

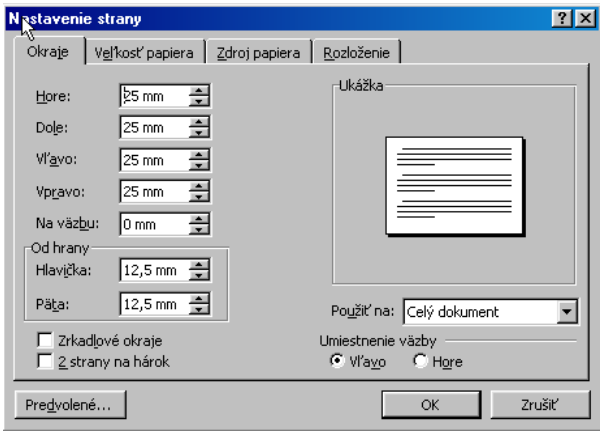
sa objaví po spustení MS Wordu kliknutím na úvodnú ikonu a zobrazuje nám základné informácie, lišty a tlačidlá na spracovanie dokumentu. Horná lišta vždy ukazuje názov dokumentu, nasleduje základná ponuka a formátovacie panely. Pri podržaní šípku kurzora na ikonách na lište sa pod ním objaví žltý obdĺžnik s popiskom funkcie ikony. Vľavo a hore môže byť pravítko, vpravo a dole posuvníky a celkom dole informácie, na ktorej strane, sekcii, riadku a stĺpci sa nachádzame. Pretože Word je konfigurovateľný, výzor hlavného okna sa môže meniť podľa nastavenia uživateľa. Ukladanie a načítanie súboru sa robí štandardným spôsobom z lišty voľbou `Súbor - Otvoriť`, respektíve `Súbor - Uložiť ako`. K tomu, aby formátovanie fungovalo pre daný text, je treba ho najprv vysvietiť pomocou kliknutia, podržania tlačidla myši a potiahnutia. Pri zarovnaní alebo číslovaní odstavca stačí, keď je kurzor kdekoľvek vovnútri daného odstavca.

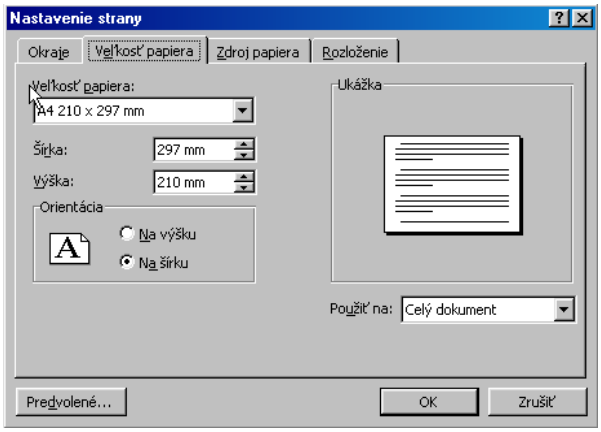


Voľba rozsahu stránky

Na nasledujúcich obrázkoch vidíte, ako sa dá nastaviť vzdialenosť textu od okrajov strany. Hlavička a päta sú dôležité parametre pre nastavenie vzdialenosti čísla strany od okraja strany. Číslo strany ale musíme vložiť voľbou `Vložiť - Čísla strán` zo základnej ponuky.

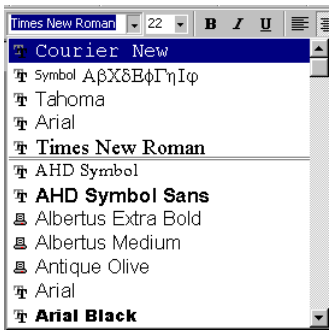






Rezy písmen a ich veľkosť

Už od počiatku kníhtlače sa datujú snahy mať v knihách čo najkrajšie vyzerajúce písmo. To viedlo k špecializácii typografov, vysoko vážených umelcov navrhujúcich typy písma. Napríklad aj francúzsky kráľ Ludovít XV sa zaoberal typografiou. Existuje veľmi veľa typov písma, ale štandardne sa používa okolo desiatky. Meniť typ písma môžeme po vysvietení textu pomocou myši, bez vysvietenia sa zmena typu prejaví až pre novo písaný text.

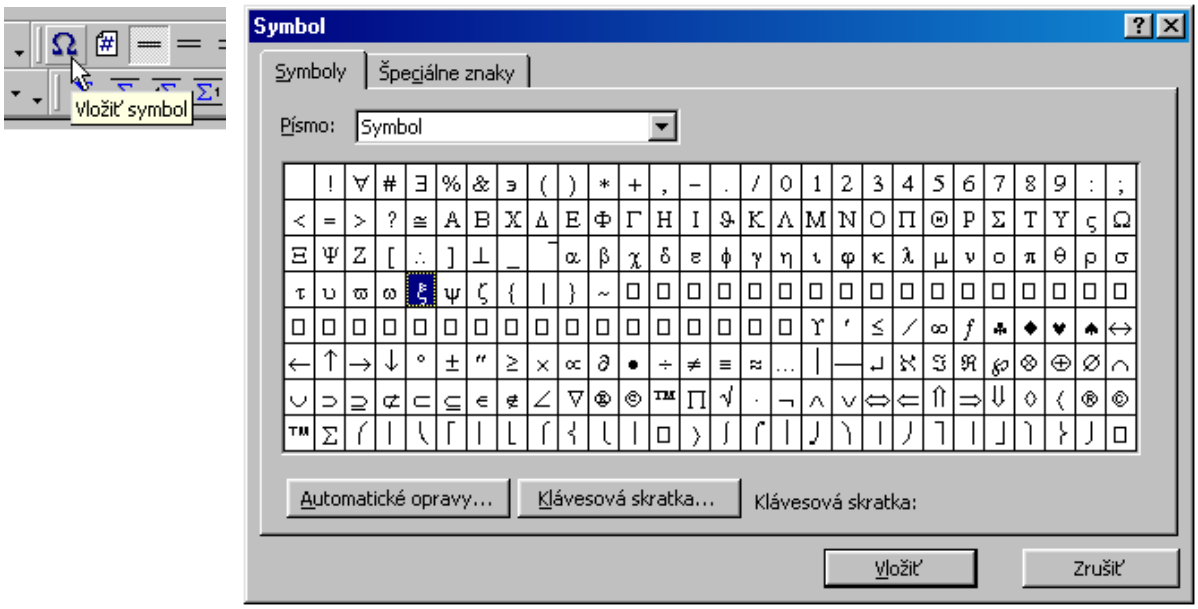


Nasledujú príklady typov (rezov, "fontov") písmen:
Times New Roman
Courier New
Σψμβολ
Arial
Book Antiqua
Century Schoolbook
Tahoma

Hneď vedľa nastavenia mena písma sa dá nastaviť aj jeho veľkosť. Najčastejšie používanou veľkosťou v bežnom texte je 10 alebo 12 (výška sa meria v tzv. bodoch, 1 bod je asi 0.35 mm) a typ písma **Times New Roman**. Na počítačové programy sa často používa písmo Courier, ktoré má tú výhodu, že každé písmenko je rovnako široké, napr. mmm a iii, narozdiel od

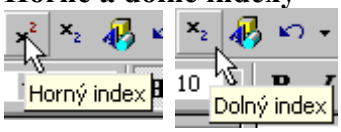
Príklady veľkosti rezov písmen sú veľkosť 28, veľkosť 18, veľkosť 12, veľkosť 10, veľkosť 8, veľkosť 6.

Relatívne často sa používajú aj špeciálne znaky, symboly. Jednou možnosťou, ako ich vložiť, je kliknúť na nasledujúcu ikonku, pokiaľ ju máme na lište, inak postupujeme z hlavnej ponuky voľbou Vložiť - Symbol, potom dva razy klikneme na vybraný symbol



Relatívne často používané sú napríklad matematické symboly $\in \pm \neq$, grécke písmená π, šípky $\rightarrow \Rightarrow$, alebo iné znaky ©. Namiesto typu písma Symbol si môžeme vybrať aj iné typy, napríklad Windings so zaujímavými symbolmi.

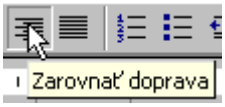
Horné a dolné indexy



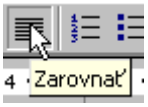
Pokiaľ chceme vyrobiť horný či dolný index, vysvietime text indexu a alebo klikneme na hore uvedené ikonky, pokiaľ ich máme na lište, alebo použijeme `Formát - Písmo` a začiarkneme dole voľbu `Horný index` alebo `Dolný index` z ponuky. Výsledky môžu vyzerať napríklad takto: A_{12}=0.123, A^{12}=0.123, A_1^2=0.123. Treba si všimnúť, že nie je možné umiestniť horný index priamo nad dolný index, to je možné až v pokročilejšom Editore rovníc.

Zarovnávanie textu

Tento text je zarovnaný doľava. Tento spôsob sa často používa pri písaní kníh, kde technickí redaktori chcú mať tento formát zarovnávania

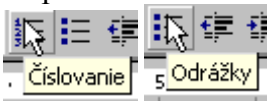


Tento text je zarovnaný doprava. Tento spôsob sa nepoužíva často.



Tento text je zarovnaný podľa okrajov. Tento spôsob sa často používa pri písaní odborných dokumentov, akými sú eseje a projekty.

Vynechaná je ikona zarovnania textu centrovaním, ktorá sa často používa u nadpisov alebo obrázkov, je použitá aj u tohto odstavca.

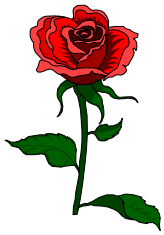
Číslovanie a odrážky
sú dôležité pre tvorbu zoznamu. Pri dlhších zoznamoch sa potom pri presunu položky nemusíme starať o prečíslovanie a odsek u danej položky je vhodne posunutý doprava. Používame ikony 

, výsledok ich aplikácie je vidno na zoznamoch dole.

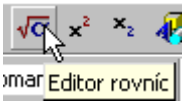
1. Algoritmus je inicializovaný náhodnou generáciou populácie
2. Každé riešenie z populácie je ohodnotené fitness.
3. Náhodne sa vyberie z populácie riešenie potom rulety.

- Algoritmus je inicializovaný náhodnou generáciou populácie
- Každé riešenie z populácie je ohodnotené fitness.
- Náhodne sa vyberie z populácie riešenie potom rulety.

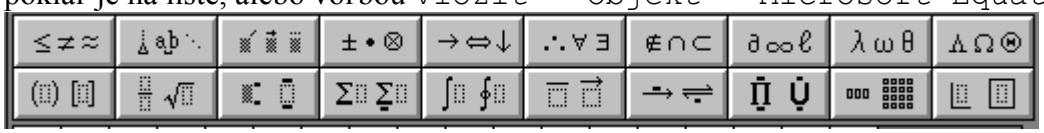
Vkladanie obrázka
Už hotový a uložený obrázok môžeme vložiť z hlavného menu: `Vložit - Obrázok`

a v kategórii rastliny je aj 

Pravým tlačítkom myši potom klikneme na obrázok a zvolíme `Formátovať objekt`, kde v paneli `Umiestnenie` môžeme vykliknúť `Plávať ponad text`, aby objekt bol vsadený v riadku spolu s textom (výšku jeho umiestnenia v riadku môžeme upraviť jeho vysvietením a `formátovaním Formát - Písmo - Medziznakové medzery - Umiestnenie - Zvýšené`, či `Znížené` a doplniť počet bodov), alebo v paneli `Obtekanie` môžeme zvoliť `Spôsob obtekania`, jedna z volieb nám napríklad dovolí, aby text išiel cez obrázok.

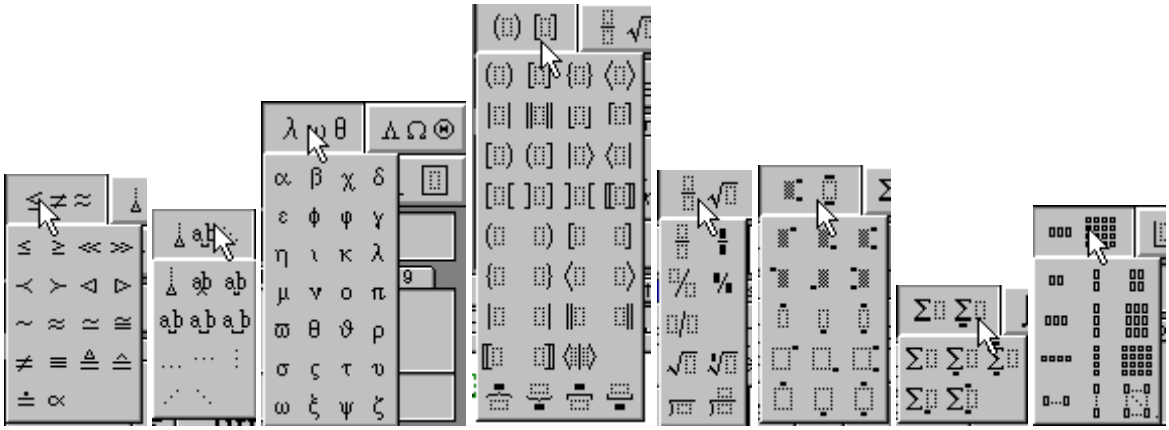
Equation – editor formúl
Pri prezentovaní výsledkov často sa vyskytne situácia, že musíte použiť aj matematický vzorec, MS Word obsahuje technické prostriedky (Equation) pre editáciu formúl, môžeme tak vytvoriť napríklad Einsteinovu formulu pre energiu $E = \sqrt{\frac{m^0}{1-\frac{v^2}{c^2}}}\,c^2 \Rightarrow E = m^0 c^2$ alebo Laplaceov integrál $\int_0^\infty e^{-x^2} dx = \frac{\sqrt{\pi}}{2}$. Editor formúl môžeme naštartovať ikonou 

, pokiaľ je na lište, alebo voľbou `Vložiť - Objekt - Microsoft Equation 3.0`.



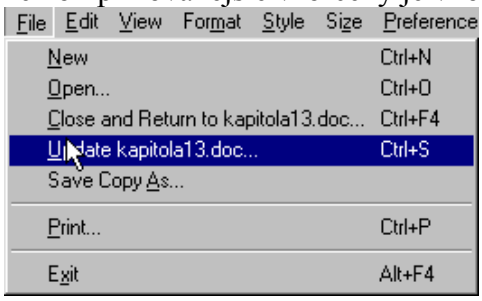
Základné typy templátov

Dolu uvedené panely sa rozvinú po poklepnutí na daný vzor na lište, treba kliknúť na vybraný z nich, a v prípade napríklad zlomku kliknúť do horného alebo prázdneho obdĺžniku  , aby sme mohli zapísať čitateľ alebo menovateľ. Podobne je to u horných alebo dolných  indexov.



nerovnice medzery grécke písmena zátvorky zlomky indexy sumy matice

Po napísaní vzorčeka je potrebné ísť na voľbu `File - Update` a `File - Close and return to` ..., čím sa uloží vzorček do textu a vyskočí z Editora rovníc. Je treba zdôrazniť, že komplikovanejšie vzorčeky je vhodné písať všetky v jednej rovnici.



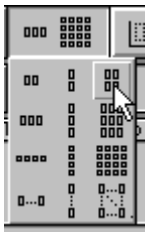
Keď už máme napríklad kvadratickú rovnicu $ax^2+bx+c=0$ prepísať na normalizovaný tvar $x^2+px+q=0$, najprv dvakrát klikneme na tento vzorček, čím sa dostaneme do editoru rovníc, vzorček potom uložíme a vyskočíme z editoru rovníc ako v predošlom prípade.

Formulu pre korene kanonickej kvadratickej rovnice $ax^2+bx+c=0$ by sme vyrobili následnou postupnosť krokov: $x_{12}= \Rightarrow x_{12}=\frac{\square}{\square} \Rightarrow x_{12}=\frac{-b}{\square} \Rightarrow x_{12}=\frac{-b\pm\sqrt{\square}}{\square} \Rightarrow x_{12}=\frac{-b\pm\sqrt{b^2-4ac}}{\square} \Rightarrow x_{12}=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$. Pri výrobe formule pre korene normalizovanej kvadratickej rovnice $x^2+px+q=0$ by postupnosť krokov vyzerala nasledovne: $x_{12}= \Rightarrow x_{12}=\frac{\square}{\square} \Rightarrow x_{12}=-\frac{p}{2} \Rightarrow x_{12}=-\frac{p}{2}\pm \Rightarrow x_{12}=-\frac{p}{2}\pm\sqrt{\square} \Rightarrow$

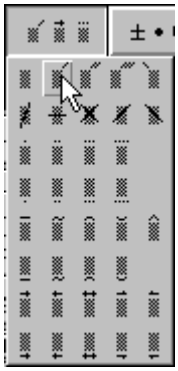
$x_{12} = -\frac{p}{2} \pm \sqrt{(\ )} \quad \Rightarrow \quad x_{12} = -\frac{p}{2} \pm \sqrt{\left(\frac{\square}{\square}\right)} \quad \Rightarrow \quad x_{12} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)} \quad \Rightarrow \quad x_{12} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2} \quad \Rightarrow$

$x_{12} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$.

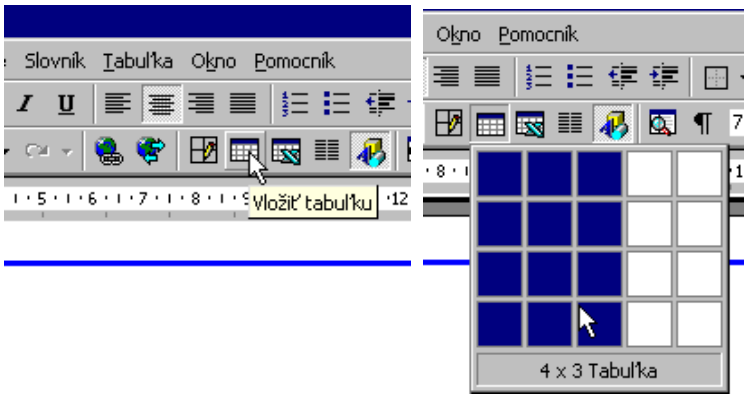
Štvorcová matica A by sa vyrábala nasledovne: $A = \ \Rightarrow A = \left(\square\right) \Rightarrow A = \begin{pmatrix} \square & \square \\ \square & \square \end{pmatrix} \Rightarrow$

$A = \begin{pmatrix} 1 & 0 \\ -1 & 2 \end{pmatrix}$, kde od prázdneho zelene orámovaného štvorčeku v zátvorke k matici v zátvorke sa prejde kliknutím na voľbu  , prvky matice sa doplnia po kliknutí do jednotlivých prázdnych štvorčekov.

Ako ďalší príklad poslúži tvorba parciálnej diferenciálnej rovnica difúzie $\frac{\partial c}{\partial t} = D\frac{\partial^2 c}{\partial x^2}$ (pre $t \geq 0$ a $0 \leq x \leq l$) pri postupnosti krokov $\frac{\partial c}{\partial t} = \ \Rightarrow \frac{\partial c}{\partial t} = D\frac{\partial^2 c}{\partial^2 x}$, pričom využijeme pri vkladaní znaku delta do rovnice $\frac{\partial c}{\partial t} = D\frac{\partial^{\square}}{\square}$ panel znakov 

Pomocou editoru rovníc môžeme generovať rôzne kombinácie písmen a znakov, ako sú A', A''', $\overline{A}$, $\vec{A}$, $\tilde{A}$, $\underline{A}$, $\overline{\underline{A}}$, a to pomocou panelu 

Konštrukcia tabuľky
sa najjednoduchšie robí cez ponukovú lištu, kde si po kliknutí na ikonu "vložiť tabuľku" vysvietením žiadaného počtu okienok zvolíme, koľko bude mať tabuľka riadkov a stĺpcov.

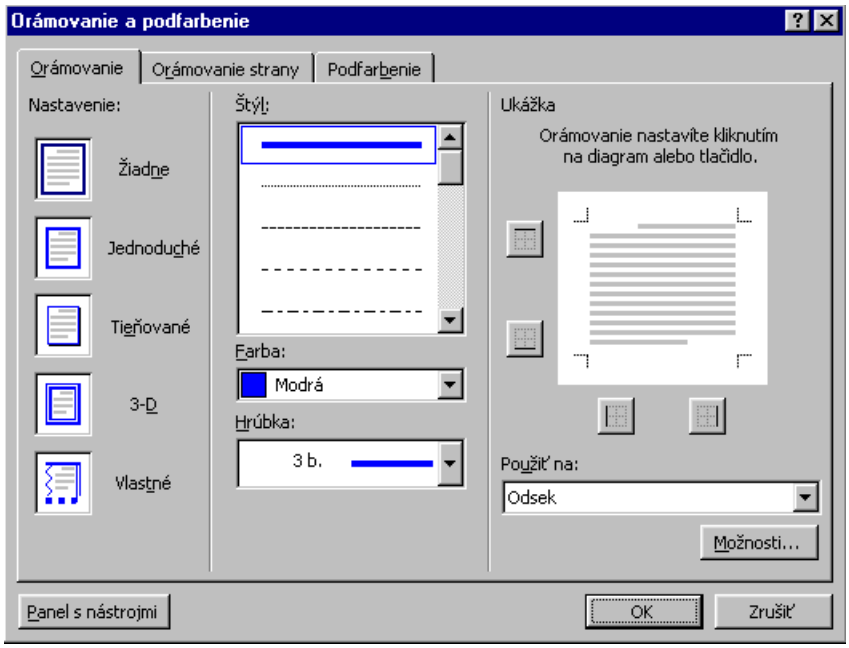


Tabuľka sa potom natiahne na celú šírku strany a výška jej buniek bude štandardná.

Takú tabuľku potom môžeme vyplniť dátami, pričom výška riadkov sa automaticky upravuje podľa potreby.

meno	rok narodenia	váha
Janko Hraško	1985	48
Ferko Hnilička	1967	95
Milan Parenica	1975	72

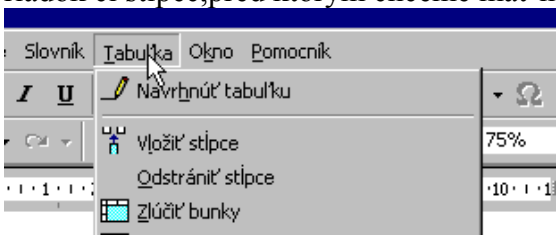
Tabuľku vysvietime a potom v položke `Formát` vyberieme položku `Orámovanie a podfarbenie`



Takto môžeme vyrobiť obrysy tabuľky

meno	rok narodenia	váha
Janko Hraško	1985	48
Ferko Hnilička	1967	95
Milan Parenica	1975	72

Zavedenie nového riadku/stĺpca v tabuľke je tiež jednoduché, vysvietime v tabuľke kurzorom riadok či stĺpec,pred ktorým chceme mať nový riadok alebo stĺpec a použijeme voľbu



Tak získame napr. nasledujúcu tabuľku, u ktorej sme ale naviac museli pomocou voľby `Tabuľka - Výška a šírka bunky` zmeniť šírku stĺpcov, aby sa tabuľka vošla na šírku stránky. Pridanie riadku sa dá urobiť aj oveľa jednoduchšie, nastavíte kurzor na koniec riadku, za ktorým sa má nový riadok pridať a stlačíte Enter.

Reg. číslo	meno	rok narodenia	váha
12985	Janko Hraško	1985	48
76345	Ferko Hnilička	1967	95
99213	Milan Parenica	1975	72

Keď potom vysvietime povedzme posledné dva stĺpce, môžeme ich voľbou `Tabuľka - Odstrániť stĺpce` odstrániť z tabuľky

Reg. číslo	meno	rok narodenia	váha
12985	Janko Hraško	1985	48
76345	Ferko Hnilička	1967	95
99213	Milan Parenica	1975	72

dostávame potom

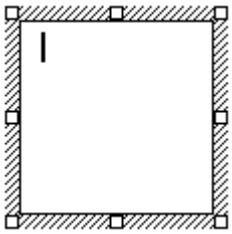
Reg. číslo	meno
12985	Janko Hraško
76345	Ferko Hnilička
99213	Milan Parenica

Tabuľka sa dá aj ľahko usporiadať, povedzme podľa roku narodenia, stačí vysvietiť stĺpec s rokom narodenia, použiť z hlavnej lišty príkaz `Tabuľka - Zoradiť`, a stačí ponechať nastavené voľby `Zoradiť podľa Stlpec 3, Typ: Číslo: Vzostupne` a stlačiť `OK`. Dostaneme

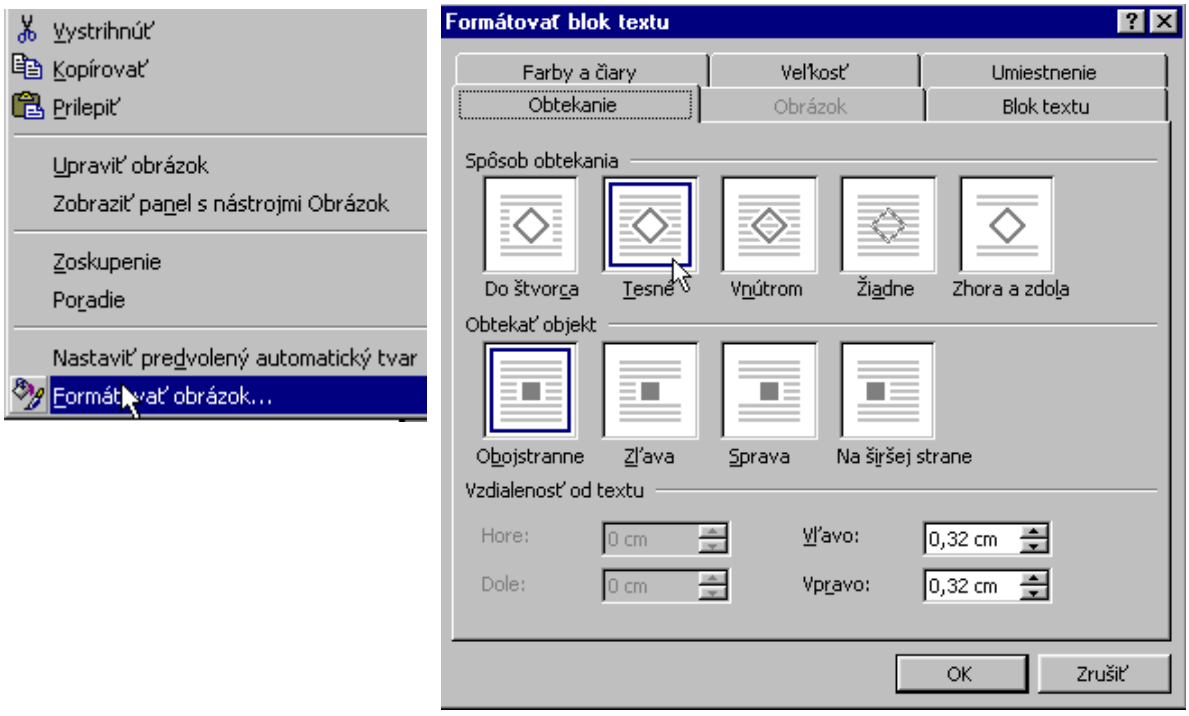
Reg. číslo	meno	rok narodenia	váha
76345	Ferko Hnilička	1967	95
99213	Milan Parenica	1975	72
12985	Janko Hraško	1985	48

Plávajúci blok

potrebujeme napríklad vtedy, keď máme obrázok aj s popisom vnoriť do textu, aby text bol aj vľavo alebo napravo okolo obrázku. V tom prípade postupujeme voľbou `Vložiť` Blok textu, a klikneme do textu, objaví sa štvorec, do ktorého môžeme začať písať text, ktorý môžeme potiahnutím za malé štvorčeky v rohoch a v stredoch strán zväčšovať či zmenšovať a kliknutím a potiahnutím za okraj štvorca ho posunujeme na strane dokumentu.



Už hotový inde vytvorený obrázok potom môžeme skopírovať Ctrl C a normálne "prilepiť" pomocou Ctrl V dovnútra bloku. Kliknutím pravým tklačítkom myši na okraj bloku a voľbou `Formátovať blok textu` a zmenou v panelu `Farby a čiary` môžeme odstrániť obrysovú čiaru bloku, a a zmenou v panelu `Obtekanie - Spôsob obtekania - Tesné` dosiahneme vnorenie bloku do textu..



Hore vidíme rôzne spôsoby obtekania bloku.

Úlohy

Úloha 13.1.

Nadpis

Vytvorte dokument, ktorý bude formátovaný podobným spôsobom ako toto zadanie. Nadpis má byť červenou farbou, typom písma Times New Roman veľkosti 20 a centrovaný, pomocou vysvietenia nadpisu a príkazov z lišty `Formát - Odsek` nastaviť veľkosť medzery za odstavcom na 20 bodov. (Pozor, napr. príkazy v českej verzii Wordu sa líšia názvami!) Základný text Vášho cvičenia môže kopírovať návod na tomto liste, alebo môžete vytvoriť ľubovoľný iný text podľa Vášho uváženia. Základný text (teda tento) má byť typom písma Times New Roman veľkosti 12 a zarovnaný na okraje, a má vo vnútri obsahovať aj slová vo fonte `Courier New` veľkosti 12.

Úloha 13.2. Ľubovoľný text v MS Worde si uložte pod názvom vášho priezviska v adresári Začiatočníci, otvorte si aj druhý dokument pomocou Súbor - Nový, obsah tohto dokumentu doňho prekopírujte a zobrazte si obidva dokumenty zaraz tak, aby jeden zapĺňal ľavú polovicu obrazovky a druhý pravú polovicu.

Úloha 13.3. Zobrazte svoj dokument rozdelený na dve časti (MS Word je spustený iba raz), v jednej časti ukážte začiatok dokumentu, v druhej koniec dokumentu, použite príkaz z hlavnej lišty `Okno – Rozdeliť`. Druhú časť zobrazte zväčšenú na 200% a so zobrazením všetkých "neviditeľných" znakov (napr. znakov odstavca - pozrite si ikony na štandardnom paneli nástrojov).

Úloha 13.4. Pre časť textu použite riadkovanie 1,5 (pozri `Formát – Odsek`) a zarovnanie doprava, čo je vidno na tejto vete. Zároveň si pomocou príkazov, ktoré nájdete vo `Formát - Odsek` nastavte v `Špeciálne – Prvý riadok` tak, aby prvý riadok začínal presne 3 centimetre od ľavého okraja stĺpca (teda predchádzajúceho textu).

Úloha 13.5. Vložte do textu symboly a použite horné a dolné indexy tak, aby ste vytvorili Ψ^{i}_{j} , pričom pokiaľ nemáte odpovedajúce ikonky vloženia symbolu na lište, použite `Vložiť – Symbol`, a pokiaľ nemáte formátovanie indexov na lište, vysvieťte si "i" resp. "j" a použite vhodnú voľbu vo `Formát – Písmo`

Úloha 13.6. Pomocou `Formát – Písmo` vyrobte písmo s iskriacim textom (animáciou).

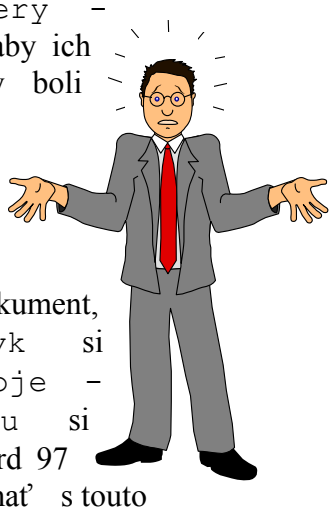
Úloha 13.7.

Ďalej si časť textu vysvieťte a choďte na `Formát - Orámovanie a podfarbenie`, a naformátujte si orámovanie čiarou tlstou 3 body a podfarbenie červenou.

Úloha 13.8. Naformátujte okraje textu tak, aby boli 1,5 cm od horného okraja a 2 cm od bočných a 3 cm od dolného okraja strany, vložte číslo strany a naformátujte ho tak, aby bolo začínalo od 3 strany, bolo 2 cm od spodného okraja, jeho veľkosť bola 16 a typ fontu Arial (po vložení čísla strany si ho vysvietime a naformátujeme podobne ako u bežného textu, pre formátovanie už existujúceho čísla strany je potrebné naň dvakrát kliknúť).

Úloha 13.9. Vložte si nasledujúce symboly z fontu Windings do textu 

a zväčšite si ich na veľkosť 36 a pomocou `Formát - Písmo - Medziznakové medzery - Umiestnenie` si ich posuňte dole, tak, aby ich stred súhlasil so stredom riadku, nie aby boli zarovnané na dolný okraj.



Úloha 13.10. Vložte obrázok zo súboru CODELAT.WMF z adresára popular do textu tak, aby mal tesné obtiekanie.

Úloha 13.11. Vyberte si celý dokument, pomocou `Nástroje - Jazyk` si nastavte Slovenčinu a v `Nástroje - Možnosti - Kontrola pravopisu` si nastavte automatickú kontrolu (Word 97 v kombinácii s Windows XP môže mať s touto kontrolou problémy riešiteľné iba správcom systému). Keď je niektoré slovo teraz červene podtrhnuté, kliknite naňho pravým tlačítkom myši a vyberte si z kontextovej ponuky správny tvar slova, alebo inú akciu pre dané slovo.

Úloha 13.12. Celý dokument si uložte pod názvom vášho priezviska v adresári Začiatočníci, otvorte si aj druhý dokument pomocou Súbor - Nový, obsah tohto dokumentu doňho prekopírujte a zobrazte si obidva dokumenty zaraz pomocou Príkazu `Okno - Usporiadať všetko`. Posledné odstavce si automaticky očíslujte a prispôsobte si pomocou ponuky v hlavnej lište `Formát` štýl číslovania na (a) (b) (c) a zarážku na 0,8 cm.

Úloha 13.13. Skopírujte si z internetu do Vášho nového dokumentu v MS Wordu akýkoľvek dlhší text (na niekoľko strán) a skúste si pomocou `Úpravy - Nahradiť` vyhľadať všetky výskyty jednej medzery a vymeňte ich za dve medzery za sebou.

Úloha 13.14. Skopírujte si z internetu do Vášho nového dokumentu v MS Wordu text z ľubovoľnej WWW strany s aspoň jednou stranou textu, všetky výskyty medzier vymeňte za znak ukončenia odstavca a zoraďte všetky takto získané slová (môžu sa opakovať) podľa abecedy. Použite usporiadanie z voľby `Tabuľka`, funguje aj na bežný text.

Úloha 13.15. Vymeňte výskyt akýchkoľvek číslic za červené **XXX**. Použite `Viac možností - Špeciálne` na určenie ľubovoľnej číslice a po vysvietení XXX `Formát - Písmo` na formátovanie červenej farby.

Úloha 13.16. Vložte prerušenie sekcie (`Vložiť`) približne doprostred dokumentu a strany v druhej sekcii nastavte "naležato" (`Súbor - Nastaviť strany`).

Úloha 13.17. Zistite počet znakov v dokumente (`Nástroje - Počet slov`).

Úloha 13.18. Pomocou príkazov `Nástroje - Prispôsobiť - Príkazy` si doplňte na lištu s príkazovými ikonkami ikonku na prečiarkovanie slov.

Úloha 13.19. Vložte si do dokumentu tabuľku s dvoma stĺpcami a troma riadkami, pridajte ešte jeden riadok nastavením sa tesne na koniec tabuľky a stlačením `Enter`, zmeňte veľkosť jednotlivých buniek na 1x3 cm, bunky v prvom riadku zlúčte do jednej bunky (po vysvietení buniek `Tabuľka - Zlúčiť bunky`), pomocou `Orámovanie a podfarbenie` nastavte orámovanie

nasledujúco (tlsté 3 body, tenké 1 bod), vložte do tabuľky číselné hodnoty, a do spodného riadku vložte vzorce (`Tabuľka - Vzorec`), ktoré spočítajú súčet hodnôt v prvom stĺpci a priemer v druhom stĺpci. Tabuľku posuňte pomocou "pravítka" (zobrazuje sa voľbou v `Zobraziť`) približne doprostred riadku.

Moja tabuľka	
10	30
20	40
30	23,33

Pozor, ako vidíte z priemeru, vzorec =AVERAGE(ABOVE) zobral hodnotu aj bunke s nadpisom Moja tabuľka, a keďže tam nie je číslo, počítal ju ako nulu, preto vyšiel nezmyseľný priemer! Dá sa obísť použitím vzorca =AVERAGE(b2;b3), kde b znamená druhý stĺpec a 2 a 3 sú čísla riadkov.

Úloha 13.20. Pomocou Editora rovníc vytvorte rovnicu

$$D(\mathrm{A}_G) = d(\mathrm{A}_F, \mathrm{A}_G) = \sum_{\alpha \in A} |F(\alpha) - G(\alpha)|$$

Úloha 13.21. Pomocou Editora rovníc vytvorte rovnicu

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

Úloha 13.22. Pomocou Editora rovníc vytvorte rovnicu

$$\lim_{\xi \to \infty} f(\mathbf{x}) = \begin{cases} f_{max}\left(\mathbf{x} = \mathbf{x}_{opt}\right) \\ f_{min}\left(\mathbf{x} \neq \mathbf{x}_{opt}\right) \end{cases}$$

Úloha 13.23. Pomocou Editora rovníc vytvorte rovnicu

$$y = \begin{cases} 1 & \left(\text{ak} \quad \sum_{i=1}^{n} w_i x_i + \sum_{\substack{i,j=1 \\ (i \leq j)}}^{n} w_{ij} x_i x_j + \ldots - \vartheta \geq 0 \right) \\ 0 & \left(\text{ak} \quad \sum_{i=1}^{n} w_i x_i + \sum_{\substack{i,j=1 \\ (i \leq j)}}^{n} w_{ij} x_i x_j + \ldots - \vartheta < 0 \right) \end{cases}$$

Úloha 13.24. Pomocou Editora rovníc vytvorte rovnicu

$$A_4 = \left(\frac{k_2 k_3 + k_4 k_5}{k_1 k_3 + k_5^2} N_{c,tot}, \frac{k_1 k_4 - k_2 k_5}{k_1 k_3 + k_5^2} N_{c,tot} \right), P = \left\{ \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p \right\} \subseteq \{0,1,\#\}^n , ,$$

Úloha 13.25. Pomocou Editora rovníc vytvorte rovnicu

$$G_P(\boldsymbol{x}_P; Q) = \frac{\omega}{|U(Q)|} \sum_{\boldsymbol{x}_Q \in U(Q)} g_P(\boldsymbol{x}_P, \boldsymbol{x}_Q)$$

Úloha 13.26. Pomocou Editora rovníc vytvorte rovnicu

$$\mathbf{K} = \begin{pmatrix} 0.1 & 10^{-3} & 0 & 0 \\ 10^{-7} & 0.55 & 10^{-7} & 0 \\ \alpha^{x} & 10^{-11} & 0.8 & 10^{-11} \\ 0 & 0 & 10^{-7} & 1.0 \end{pmatrix}$$

Úloha 13.27. Pomocou Editora rovníc vytvorte rovnicu

$$\underbrace{x_i(t) = \frac{\sum_{j=1}^{n} q_{ij} x_i(0) e^{\lambda_i t}}{\sum_{m,j=1}^{n} q_{mj} x_j(0) e^{\lambda_j t}}}_{vzorček\ \ 1}$$

Úloha 13.28. Pomocou Editora rovníc vytvorte nasledujúcu tabuľku s koreňami kvadratickej rovnice, s veľkosťou písma 16 bodov.

#	p	q	korene
1	-3	2	$x_1=1, x_2=2$
2	1	1	$x_{1,2} = \left(-1 \pm i\sqrt{3}\right)/2$
3	-2	1	$x_{1,2} = 1$

Úloha 13.29. Voľbou `Nástroje – Možnosti` zmeňte počet posledne-krát používaných súborov ukázaných v položke `Súbor` na 9.

Úloha 13.30. Voľbou `Nástroje – Možnosti` zmeňte si nastavenie, aby ste mali automatické ukladanie súborov po 10 minútach.

Úloha 13.31. Voľbou `Nástroje – Možnosti` zmeňte si umiestnenie súborov, aby bol Word nastavený pri otváraní alebo ukladaní súborov na Váš vlastný adresár.

Úloha 13.32. Vložte do textu poznámku pod čiarou[1] (pozri dole na stránke, použie `Vložiť`).

Úloha 13.33. Upravte MS Word tak, aby zakaždým, keď napíšete samostatne stojace a , bolo toto písmeno nahradené výrazom Ha Ha Ha!!!, veľkosti 24, červené, s obrysom a iskriacim textom (animáciou). Použite `Nástroje – Automatické opravy- Formátovaný text`, zmienený text Ha Ha Ha!!! najprv vytvorte a naformátujte v bežnom texte a až potom premiestnite do poľa `za:`. V prípade potreby použite Pomocník.

Úloha 13.34. Použite `Súbor – Tlačiť` na vytlačenie ľubovoľného Vášho súboru vo formátu postskriptovskej tlačiarne do svojho adresára, vo Windows zmeňte príponu súboru z prn na ps a pozrite sa na súbor pomocou programu GhostView.

[1] Toto je poznámka pod čiarou, ktorej umiestnenie tu dole a očíslovanie bolo urobené automaticky

Úloha 13.35. Zadajte do Wordu hypertextový odkaz tak, že keď kliknete na text: Moj obľúbený vyhľadávač, otvorí sa Vám strana http://www.google.com v MS Internet Exploreri.

Úloha 13.36. Zmeňte pozadie textu na bledo modrú.

Úloha 13.37. Skúste vytvoriť blok textu ako je hore a naformátovať ho tak, aby bol umiestnený cez text, ktorý ste už napísali predtým.

Skúste písať cez my text

Úloha 13.38. Použite ponuku `WordArt` v `Kreslenie` aby ste napísali text nastojato napravo

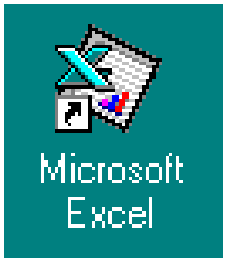
Úloha 13.39. Pomocou Wordovského kresliaceho prohramu vytvorte tučnú zaoblenú čiaru so šípkou na obidvoch koncoch, vyrobte jej kópiu a pomocou otáčania ju obráťte o 180^{o}.

Úloha 13.40. Skúste písať *našikmo* - použite ponuku `WordArt` v `Kreslenie`

Úloha 13.41. Vyrobte elipsu tak, aby pod ňou bolo vidno text, použite ponuku `Kreslenie`, potom nastavte obtiekanie pomocou formátovania kliknutím pravého tlačítka myši na elipsu

Skúste písať nastojato

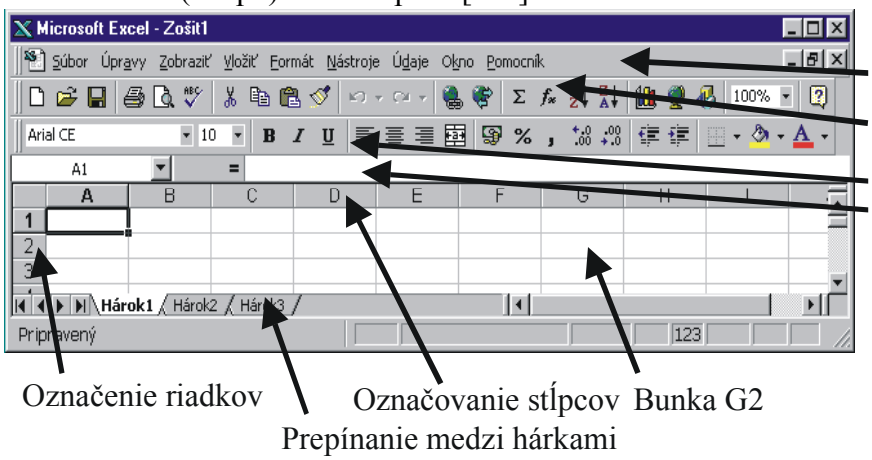
14 MS Excel 97



MS Excel 97 je tabuľkový procesor ("spreadsheet") a databázový programu s grafikou pre PC. Je to program na zaznamenávanie, analýzu a výpočet dát. Uľahčuje výpočet s opakovanými vzorci a automaticky prepočítava výsledky, keď zmeníte dáta. (Keď máte urobiť vo cvičeniach nejaký výpočet a vaše zadania sa líšia iba rôznymi dátami, stačí iba raz zadať vzorčeky, a potom už si iba každý študent v skupine zadá svoje dáta, výpočet výsledkov bude automatický, aj s automatickou zmenou prípadných výsledných grafov.) Umožňuje

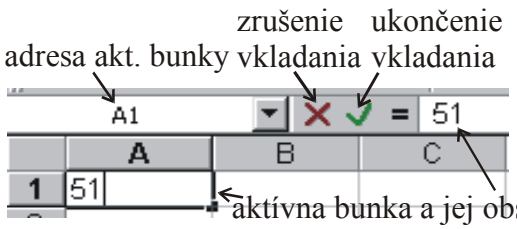
- ukladanie, premiestňovanie, **vypočítavanie a analýzu dát** vo forme čísel, textov a vzorcov,
- pohodlne usporadúvať, vyhľadávať a inak spracovávať dáta, používa štandardné **databázové operácie**
- bez námahy zobraziť dáta ako **grafy**
- automatizovať často potrebné úlohy príkazom **Makro** (programom, ktorý nemusíte programovať – ukážete mu postupnosť úkonov a on ich zopakuje)

Ďalej je uvedený iba stručný úvod do MS Excelu, predpokladá sa, že podrobnosti si pozriete v Pomocníkovi (Helpu) alebo napr. v [XX]. Úvodná obrazovka Excelu vyzerá nasledovne:



Naštartovanie programu

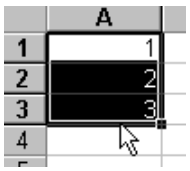
sa okrem kliknutí na ikonku Excelu dá urobiť tiež z menu `Štart - Programy - Microsoft Excel`. Excel automaticky naštartuje nový Zošit1 (Workbook1). Na otvorenie existujúceho excelovského súboru použite `File - Open`.



Keď začnete niečo písať na klávesnicu, písaný text alebo čísla sa vkladajú do **"aktívnej" bunky** označenej čiernym rámčekom. Zároveň sa ten istý text objavuje v lište na vzorce. Ukončiť vkladanie do danej bunky môžete stlačením klávesy `Enter` alebo kliknutím na zelené zaškrtnutie. Zrušiť vkladanie môžete kliknutím na červený krížik alebo klávesou `Esc`.

Z bunky na bunku sa pohybujeme
šípkami na klávesnici, alebo stlačením klávesy `Enter` pre pohyb dolu a `Tab` pre pohyb doprava; hore a doľava kombináciou týchto kláves s `Shift`.

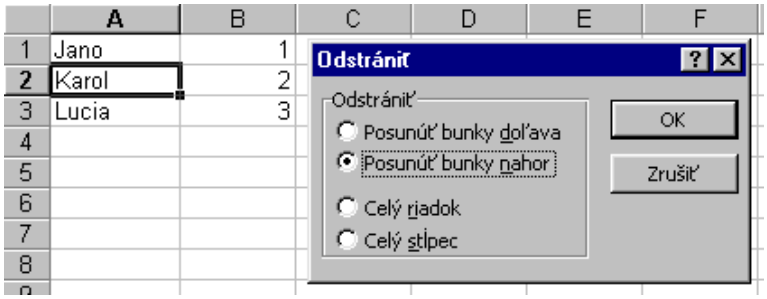
Vybrať viac buniek



sa dá napr. kliknutím myšou na bunku na začiatku alebo na konci želanej oblasti (prípadne v rohu želanej oblasti), držaním stlačeného tlačidla myši a potiahnutím na bunku na druhom konci želaného výberu (prípadne v opačnom rohu na uhlopriečke). Aktívna bunka pritom ostáva biela, ostatné sú čierne. Môžeme vybrať aj celý stĺpec kliknutím na jeho označenie, napr. A, celý riadok kliknutím na jeho označenie, napr. 1.

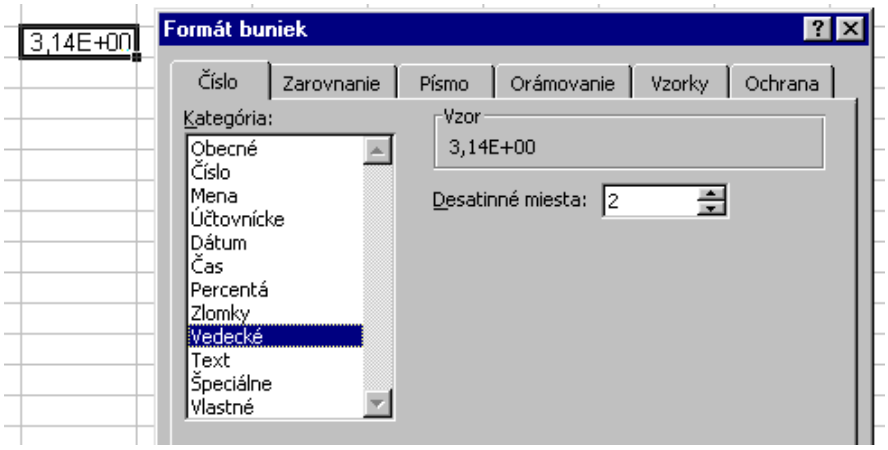
Kopírovať, presúvať alebo mazať zaraz
môžeme iba vybrané bunky. Okrem známych "horúcich kláves" `Ctrl C` na skopírovanie do pamäti, `Ctrl X` na skopírovanie do pamäti a vymazanie z buniek, `Ctrl V` na skopírovanie z pamäti do aktívnej bunky, môžeme presunúť vybraté bunky jednoducho ukázaním na ich čierny okraj a potiahnutím. Kopírujeme tak isto, iba so stlačenou klávesou `Ctrl`. Mažeme stlačením klávesy `Delete`.

Keď už nejaké dáta existujú a potrebujeme vložiť údaj doprostred nich, klikneme na cieľovú bunku (riadok, stĺpec) a ideme na voľby vložiť. Keď naopak potrebujeme údaj z prostriedku dát odstrániť, pomocou stlačenia `Delete` **iba vymažeme** obsah buniek.



Pokiaľ ale chceme automaticky posunúť dolné bunky alebo bunku naľavo na ich miesto, po vysvietení použijeme `Úpravy` - **`Odstrániť`**.

Naformátovanie buniek
sa ľahko robí ich výberom a voľbou `Formát` - `Bunky`

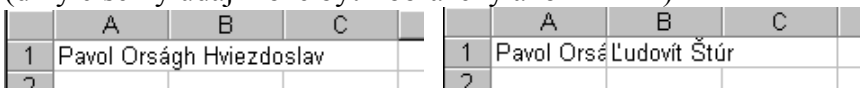


Zadávanie čísel
vyžaduje vedieť, **či máte implicitne desatinnú čiarku, alebo bodku**, v prípade použitia nesprávneho oddeľovača je dané číslo považované za text, čo sa pozná podľa toho, že je namiesto doprava automaticky zarovnané v bunke doľava. Keď máte smolu, tak vám to ešte z

1.2 vytvorí 1.II, čo je dátum. Problém je v tom, že toto dátum je považované za číslo, a to nie za číslo 1,2 ,ale za počet dní, ktorý prvého februára tohto roku uplynul od počiatku roku 1900. Napríklad pri sčitovaní Excel potom robí "zaujímavé" veci ☺.

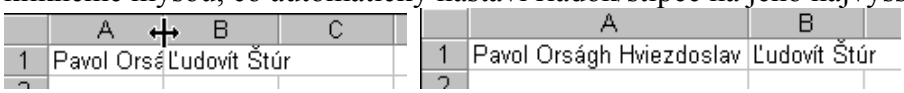
Dlhé meno alebo číslo

pri zadávaní do bunky svojím zobrazením presiahne do vedľajších buniek. Keď ale do vedľajšej bunky začneme dávať nové dáta, dáta v pôvodnej bunke ostanú, ale nie sú vidno (dlhý číselný údaj môže byť zobrazený ako ######)



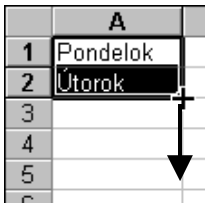
Rozšíriť riadok alebo stĺpec

môžeme tak, že umiestnime kurzor medzi písmena označujúce stĺpce alebo čísla označujúce riadky, až sa objaví dvojitá šípka, kurzor alebo potiahneme (ručná úprava) alebo dvakrát klikneme myšou, čo automaticky nastaví riadok/stĺpec na jeho najvyššiu/najširšiu hodnotu.

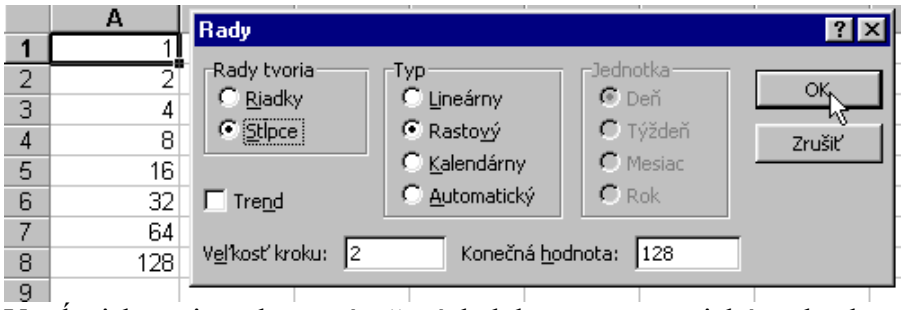


Vytváranie série dát

– čísla, mená dní a mesiacov vytvoríme tak, že prvé dve položky napíšeme do susedných buniek a potiahneme myšou dole alebo doprava po nastavení kurzora na pravý dolný roh prvých dvoch hodnôt série (pri výberu iba jednej hodnoty robíte iba kópie tejto hodnoty), keď



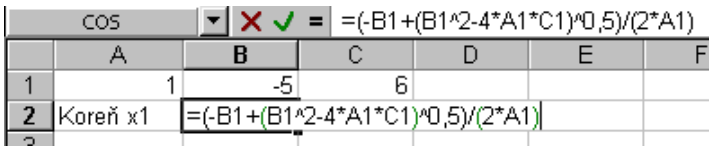
krížik ťaháme dolu, napravo sa objavujú nové hodnoty zo žltým podfarbením. Ďalej môžeme vytvoriť sériu automatickým vyplnením: vyplníte prvú hodnotu, volíte `Úpravy – Vyplniť – Rady...` (Rastový je geometrický, krok je kvocient).



V stĺpci hore je zobrazený už výsledok pre geometrický rad s kvocientom 2. Help získate kliknutím pravým tlačidlom myši na typ radu.

Zadávanie vzorcov

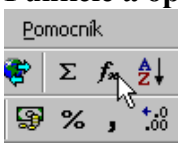
Každý vzorec začína znakom **=** nasledovaným odkazom na bunku (napr. A1), hodnotou, alebo funkciu. Vzorec vyjadrujúci koreň kvadratickej rovnice $x_1 = \frac{b + \sqrt{b^2 - 4ac}}{2a}$ môže vyzerať ako **`=(-B1+(B1^2-4*A1*C1)^0,5)/(2*A1)`**, teda COS v ľavom hornom rohu ukazuje na predošlú použitú funkciu, s daným vzorcom nemá nič spoločné. Keď stlačíme



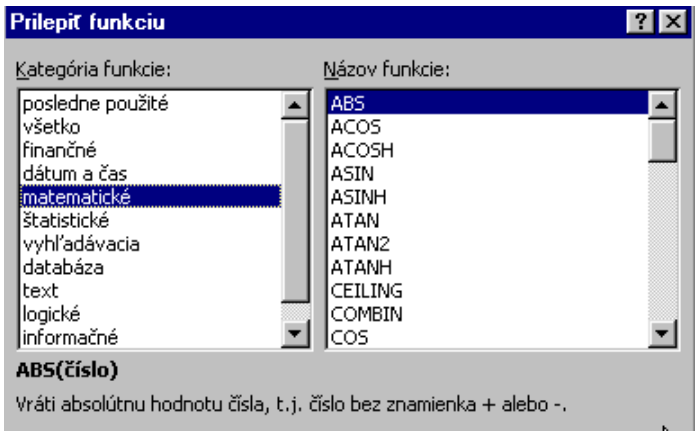
Enter, namiesto odkazov na bunky sa dosadia hodnoty v nich uložené, vzorec sa vyhodnotí a objaví sa výsledok, teda 3. Ako vidno,

násobenie je *, delenie /, umocňovanie ^. Keď nahradíte čísla v hornom riadku inými hodnotami, vzorec sa automaticky prepočíta.

Funkcie a operátory



sa zadávajú do aktuálnej bunky napríklad kliknutím na ikonku funkcie, nasledovanú výberom funkcie na "prilepenie". Existuje veľa funkcií, všetky musia byť nasledované guľatými zátvorkami. Do zátvoriek sa zadávajú premenné alebo odkazy na premenné, alebo napr. u funkcie `PI()`, ktorá vráti Ludolfovo číslo, ostávajú zátvorky prázdne.



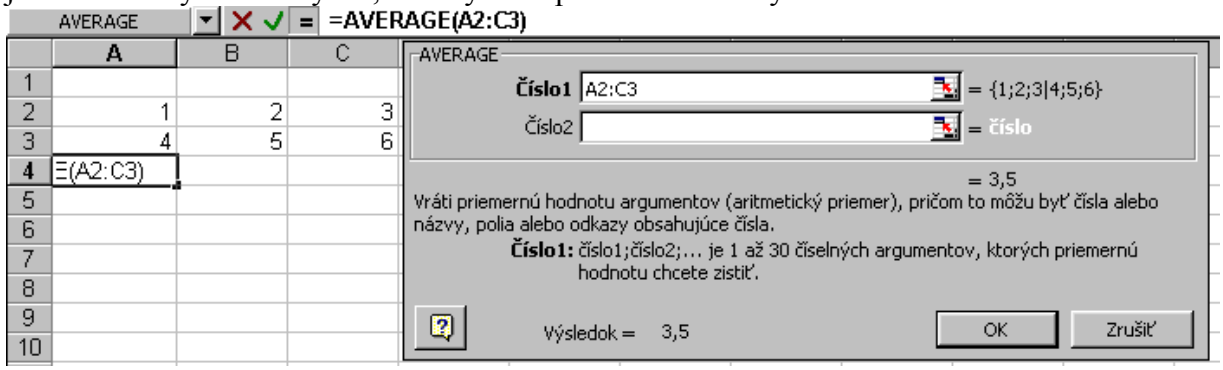
Užitočné numerické funkcie sú:
ABS(číslo) vráti absolútnu hodnotu čísla
AVERAGE(pole) vráti priemernú hodnotu argumentov (aritmetický priemer)
COS(číslo) argument je číslo v radiánoch, s predponou A je inverzná funkcia, s príponou H hyperbolická
DEGREES(číslo) konvertuje radiány na stupne
EXP(číslo) vráti e (základ prirodzeného logaritmu) umocnené na zadané číslo
IF(podmienka; výsledok pri splnení podmienky; výsledok pri nesplnení podmienky) vráti výsledok na základe splnenia či nesplnenia podmienky
FACT(číslo) vráti faktoriál čísla
LOG(číslo; základ) vráti logaritmus pri danom základe, LN() prirodzený logaritmus, LOG10() dekadický
MAX(pole) vráti najvyššiu hodnotu z množiny hodnôt. Ignoruje logické hodnoty a text.
MDETERM(pole) vráti determinant matice poľa s rovnakým počtom riadkov a stĺpcov
MIN(pole) vráti najnižšiu hodnotu z množiny hodnôt. Ignoruje logické hodnoty a text.
MINVERSE(pole) vráti inverznú maticu z matice poľa s rovnakým počtom riadkov a stĺpcov; musí ale byť vysvietené pole požadovaného rozmeru a po zadaní umiestnenia vstupného poľa treba stlačiť kombináciu troch klávesov Ctrl+Shift+Enter, nie OK, lebo pri stlačení tlačidla OK sa zobrazí iba prvá hodnota matice
MMULT(pole1;pole2) vráti maticu so súčinom polí, ktorá obsahuje rovnaký počet riadkov ako pole1 a rovnaký počet stĺpcov ako pole2; musí ale byť vysvietené pole požadovaného rozmeru a po zadaní umiestnenia vstupných polí treba stlačiť kombináciu troch klávesov Ctrl+Shift+Enter, nie OK, lebo pri stlačení tlačidla OK sa zobrazí iba prvá hodnota matice
MOD(delenec; deliteľ) vráti zvyšok po delení čísla
PI() vráti hodnotu π s presnosťou na 15 číslic
POWER(číslo;mocnina) umocní číslo na zadanú mocninu
RADIAN(číslo) konvertuje stupne na radiány
RAND()vráti náhodné číslo z intervalu (0,1), nevyžaduje argument, zmení sa pri každom

prepočítaní listu
ROUND(číslo; počet číslic) zaokrúhli číslo k najbližšej hodnote na „počet číslic“ za desatinnou čiarkou
ROUNDDOWN(číslo; počet číslic) zaokrúhli číslo smerom nadol k nule na „počet číslic“ za desatinnou čiarkou
ROUNDUP(číslo; počet číslic) zaokrúhli číslo smerom nahor od nuly na „počet číslic“ za desatinnou čiarkou
SIN(číslo) argument je číslo v radiánoch, s predponou A je inverzná funkcia, s príponou H hyperbolická
SQRT(číslo) vráti druhú odmocninu čísla
SUM(pole) spočíta všetky čísla v rozsahu buniek
TAN(číslo) argument je číslo v radiánoch, s predponou A je inverzná funkcia, s príponou H hyperbolická.

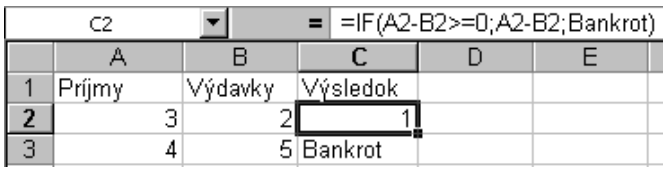
Odkazy na bunky

Pri oddeľovaní jednotlivých argumentov u funkcií používame bodkočiarku, dvojbodkou medzi odkazmi na dve bunky vkladáme aj všetky bunky medzi týmito dvoma, napr. A2:C3 je to isté ako A2;A3;B2;B3;C2;C3.

Namiesto explicitného zadania odkazov je možné po výberu funkcie tieto bunky jednoducho vysvietiť myšou, odkazy sa doplnia automaticky.

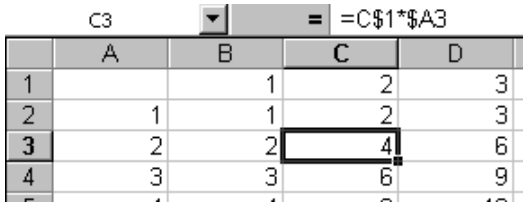


Funkcie nemusí ani nič počítať, môžu napr. vypísať text pri splnení podmienky (vzorec bol zapísaný iba do C2, do C3 bol skopírovaný). Pri kopírovaní vzorcov s odkazmi sa automaticky mení aj odkazy, teda keď máme v bunke C2 vzorec `=IF(A2-B2>=0;A2-B2;"Bankrot")` a tento vzorec skopírujeme do bunky C3, odkazy na bunky A2 a B2 sa zmení na A3 a B3, teda v C3 bude vzorec `=IF(A3-B3>=0;A3-B3;"Bankrot")`. Keď by sme tento vzorec skopírovali nie do bunky C3, ale do bunky D2, odkazy na bunky A2 a B2 sa zmení na B2 a C2, teda v D2 bude vzorec `=IF(B2-C2>=0;B2-C2;"Bankrot")`.

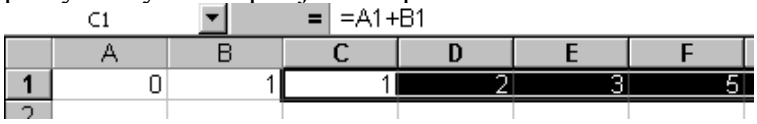


Keď ale nechceme, aby sa pri kopírovaní odkaz na nejakú bunku zmenil, zadáme na ňu absolútny odkaz, ktorý vyrobíme pridaním dolárových znakov pred písmeno stĺpca a číslo riadku, teda napr. namiesto A1 bude **`$A$1`**. To je **absolútny odkaz**, ktorý sa pri kopírovaní vzorca už nemení. Odkaz bez dolárov **`A1`** sa volá **relatívny odkaz**. Existuje ešte ale aj **zmiešaný odkaz**, kde je dolárový znak iba pred písmenom alebo iba pred číslom, teda napr. **`$A1`** alebo **`A$1`.** Odkaz **`$A1`** ostáva bezo zmeny pri kopírovaní doprava, mení sa iba pri kopírovaní nadol. Odkaz **`A$1`** sa mení pri kopírovaní doprava, mení sa iba pri kopírovaní nadol.

Načo sa zmiešané odkazy dajú použiť? Ako príklad si predstavme, že máme vygenerovať malú násobilku. Najprv si vygenerujeme čísla v prvom riadku a prvom stĺpci, pričom prvok A1 necháme prázdny. Potom do B2 vložíme vzorec `=B$1*$A2` a skopírujeme ho do celej tabuľky. Vidíme, že keď sa posunieme napr. na C3, tak sa vzorec zmení na `=C$1*$A3` ,čo je presne to čo sme chceli, odkazovať sa iba na prvý riadok a na prvý stĺpec.



Relatívne odkazy sa využijú napríklad u výpočtu Fibonacciho čísel (pozri kapitolu 2). Tie sú definované ako fib(0) = 0, fib(1) = 1, fib(n) = fib(n-1) + fib(n-2) pre väčšie n, teda "sčítaj posledné dve, aby si dostal ďalšie". Ako Exelom vyriešiť problém, koľko je fib(12) ? Do bunky A1 dáme 0, do bunky B1 dáme 1, do bunky C1 dáme =A1+B1 a potiahnutím za pravý dolný roh kopírujeme doprava.

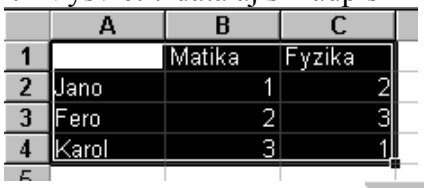


Odkazy buniek môžu viesť aj na iné hárky, dokonca aj na iné súbory, napr. odkaz `'[Ročník I.xls]Skupina I'!$B$2:$B$4;'` odkazuje na bunky **B2-B4** v hárku pomenovanom **Skupina I** a v súbore **Ročník I.xls**.

Vytváranie grafov

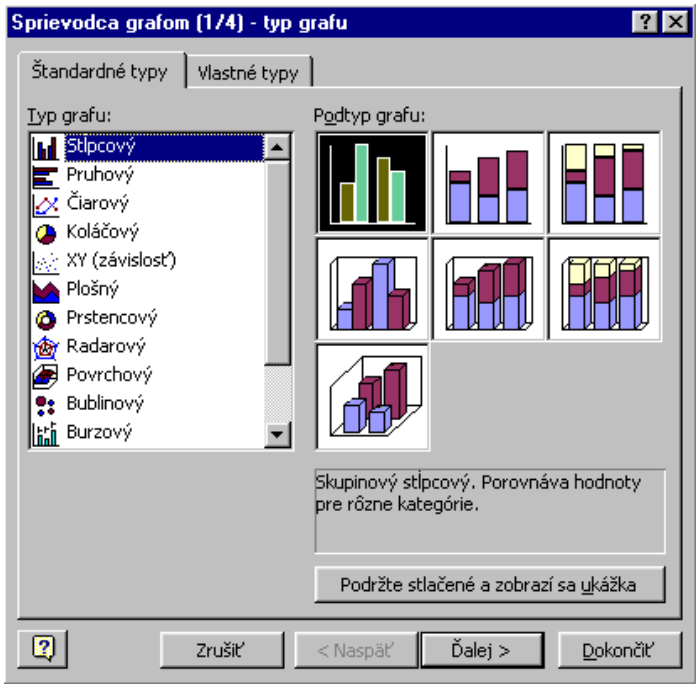
Najjednoduchším postupom sa dá vyrobiť graf troma krokmi:

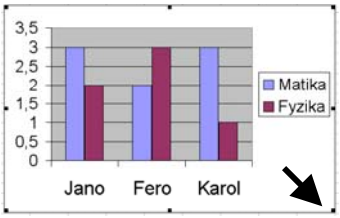
1. Vysvietiť dáta aj s nadpismi



2. Kliknúť na ikonku grafu 

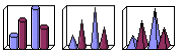
3. Kliknúť dole vpravo na tlačidlo `Dokončiť`



Výsledok je nasledujúci stĺpcový graf 

Kliknutím a potiahnutím presúvame graf, potiahnutím za čierny štvorček v rohu či v prostriedku strany zväčšujeme.
Zmenou dát sa mení aj graf!
Formátovanie robíme kliknutím pravým tlačidlom myši.

Ďalšie typy grafov a ich použitie:

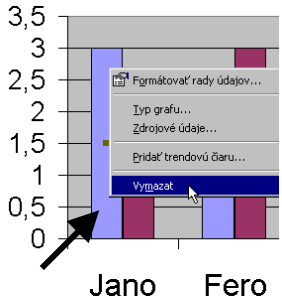
-  `pruhové grafy`, pri dlhých menovkách u x
-  `koláčové grafy` pomer časti z celku, v ekonomike, pri voľbách
-  `prstencové` ako koláčové grafy, iba pre viac sád hodnôt
-  `plošné grafy` hodnoty viac zložiek $y_1, y_2, ..., y_n$ sú poskladané na seba, napr. pre trend predaja jednotlivých produktov, vidíme aj celkový trend
-  `paprskové grafy` v sociometrii na zobrazenie vzťahu ľudí,
-  `bublinové grafy` nahradzujú 3-D povrchové grafy pre niekoľko málo hodnôt. Priemer krúžkov vyjadruje hodnotu tretej premennej.
-  `burzové grafy` pre sady miním, maxím, a priemerov
-  `valcové, kužeľové, ihlanové grafy`

atď. - aj vlastné typy
Popri každom z týchto grafov si môžeme vybrať kliknutím z viacerých podtypov zobrazených napravo.

Najobľúbenejšie typy grafov sú

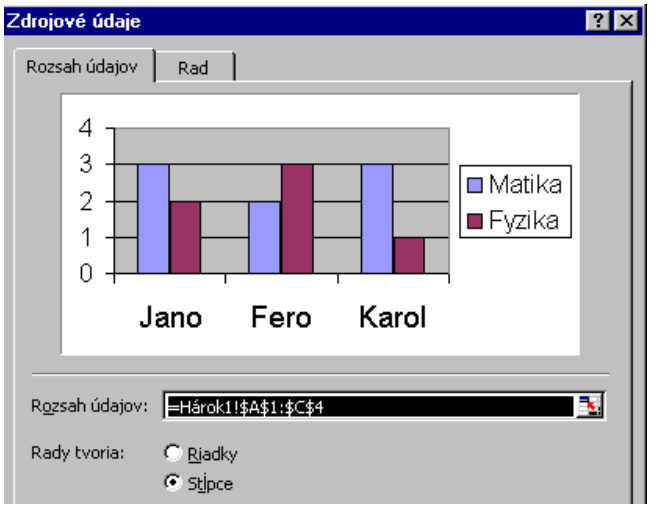
-  `Čiarový`  `XY (závislosť)`  `Povrchový`

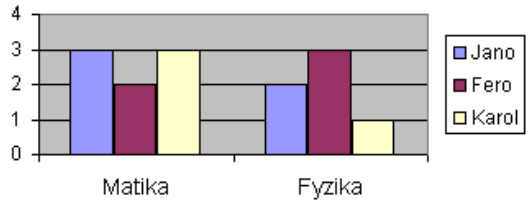
Odstraňovanie kriviek
Čo robiť, keď sa chceme zbaviť predmetu matiky v hore uvedenom grafe známok z predmetov?

- Vyberieme iba dáta, čo chceme v grafe, matiku nevysvietime (pri ďalších výberoch držíme stlačené `Ctrl`), potom klikneme 
- Alebo klikneme v grafe pravým tlačidlom myši na jeden zo sady nechcených údajov a klikneme `Vymazat` 

Riadky za stĺpce?

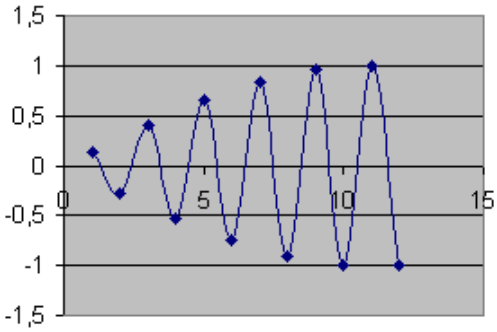
Keď chceme, aby na hore použitom príklade grafu predmetov boli na vodorovnej osi výsledky zoskupené podľa predmetov a nie podľa žiakov, pravým tlačidlom myši klikneme na graf, ďalej klikneme na `Zdrojové údaje` a na voľbu `Riadky`. Výsledkom je potom graf, kde predmety Matika a Fyzika sú ako x-ové súradnice a známky študentov sú farebne odlíšené:



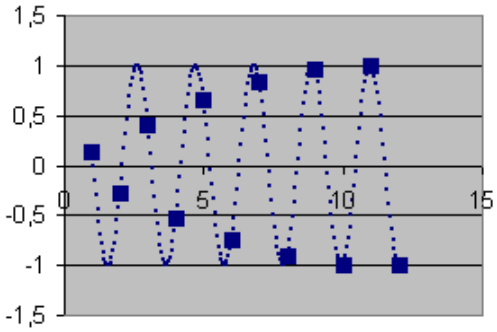


`XY (závislosť) s údajovými bodmi spojenými hladkými čiarami`

V tomto type grafu **pozor na rozlíšenie!** Napr. $y = \sin 3x$ pre x od 1 po 12, krok 1 a krok 0,05 dostávame v prvom prípade nezmysel, až v druhom prípade s podrobnejším rozlíšením dostávame skutočný priebeh (štvorčekmi sú odlíšené body, na základe ktorých program "odhadnul" priebeh prvého grafu):



Pekne vyzerajúci nezmysel



Skutočný priebeh funkcie sin 3x

Lineárna regresia a regresný koeficient

Pre vytvorený graf `XY (závislosť)` môžeme v Excelu jednoducho získať aj často používané údaje ako je hodnota regresného koeficientu alebo môžeme aj automaticky preložiť regresnú priamku. Jednoduch klikneme na voľnú bielu plochu v grafe. Na hlavnej lište sa objaví voľba `Graf`. Z nej ideme na `Pridať trendovú čiaru` - `Typ trendu a regresie` `Lineárny`. Vo vedľajšom paneli `Možnosti` zaškrtnúť `Zobraziť v grafe rovnicu regresie` a `Zobraziť v grafe rovnicu spoľahlivosti R na druhú`

Automatické preloženie odpovedajúcich kriviek spolu so získaním neznámych koeficientov môžeme urobiť aj pre iné typy základných závislostí ako sú trendy

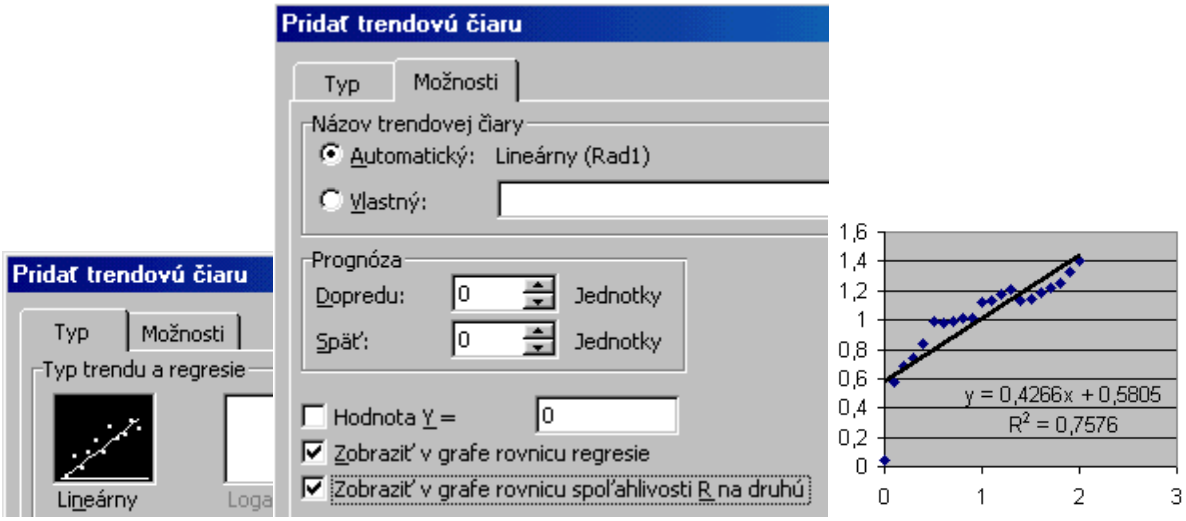
`Logaritmický` $y=a\ln(x)+b$

`Polynomický` $y=a+\ldots+fx^5$

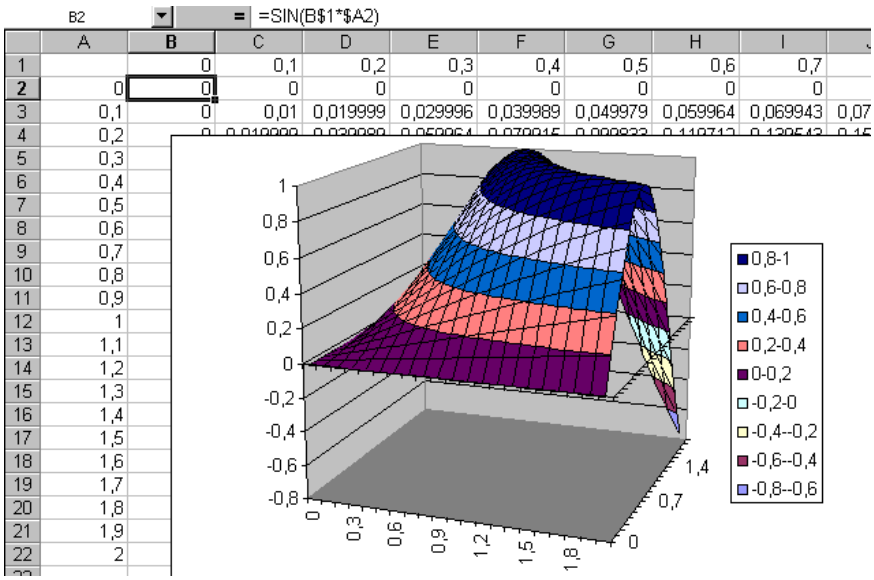
`Mocninový` $y=ax^b$

`Exponenciálny` $y=ae^{bx}$

`Kĺzavý priemer` pre hladkú krivku bez regresnej rovnice.

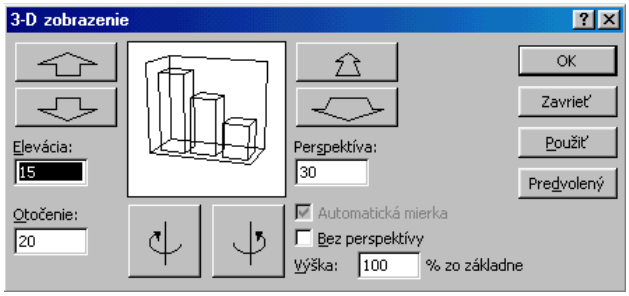


Trojdimenzionálne grafy a ich úprava

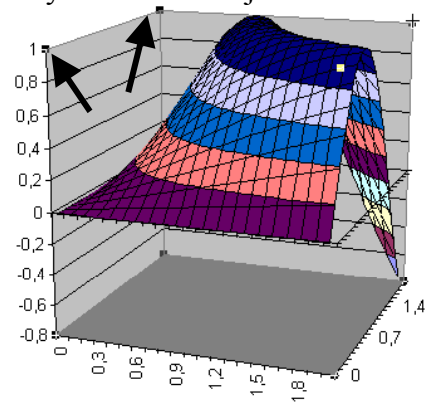


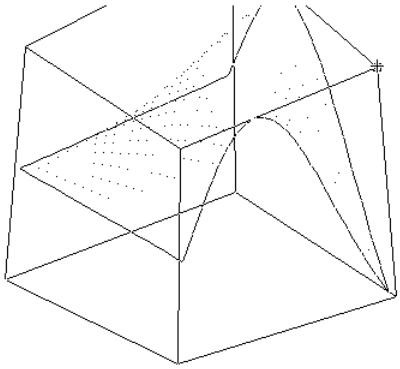
Pre získanie trojrozmerného grafu potrebujeme dvojrozmernú maticu, kde riadky predstavujú x-ovú súradnicu, stĺpce y-ovú súradnicu a hodnoty v bunkách z-ovú súradnicu. Príkladom je nasledujúci graf funkcie Sin($x \cdot y$). Keď máme maticu dát vytvorenú pomocou relatívnych odkazov, vysvietime maticu dát a postupujema na - `Sprievodca grafom` – `Povrchový (Surface)` – `Dokončiť`. Počet hodnôt u os a farebných odlíšení veľkostí *z*-tovej hodnoty sa mení automaticky so zmenou veľkosti obrázku.

Uhol pohľadu:
pre zmenu jeho nastavenia klikneme pravým tlačidlom myši na bielu plochu grafu a volíme `3-D zobrazenie`



Graf je tiež možné natáčať myšou ťahom za upravovacie body – keď klikneme na sivé pozadie plôch x-y,y-z, alebo x-z, v ôsmich rohoch kocky sa objavia čierne štvorčeky a na ktorýkoľvek z nich je možné kliknúť a potiahnuť. Pri otáčaní sa zobrazujú iba obrysy.



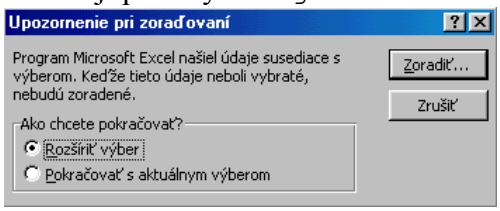


Triedenie dát

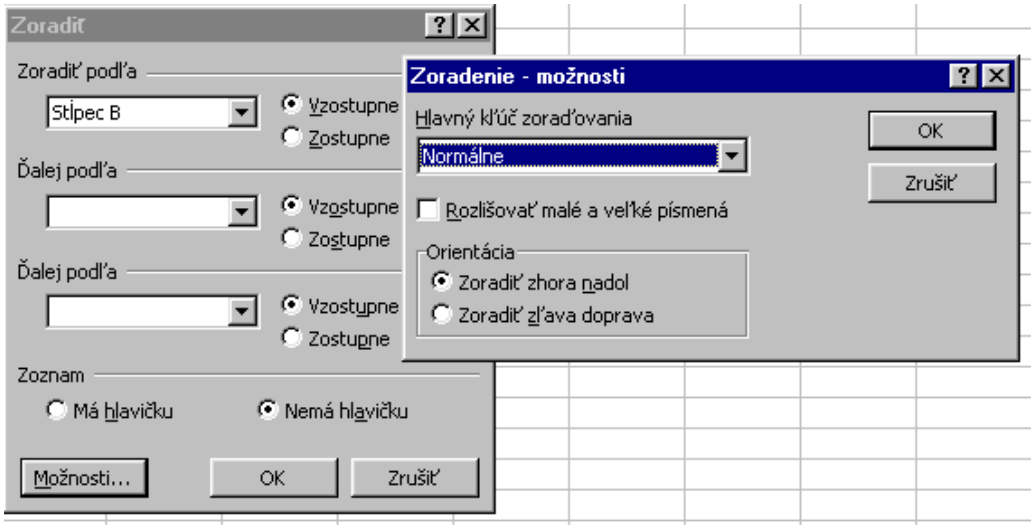
Klikneme na ktorúkoľvek bunku vo vnútri tabuľky a na ikonku  alebo  . Celá tabuľka sa potom usporiadá podľa dát v danom stĺpci.

Keď vysvietime celý stĺpec, usporiada iba vysvietený stĺpec, čo by nám v ďalej uvedenom príklade usporiadalo iba priezviská, ale krstné mená by ostali na pôvodných miestach, takže by priezviskám neodpovedali!

Keď použijeme z hlavnej ponuky `Údaje-Zoradiť` , budeme upozornení na vhodnosť rozšírenia výberu:

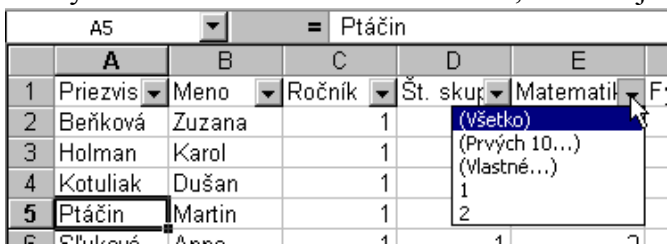


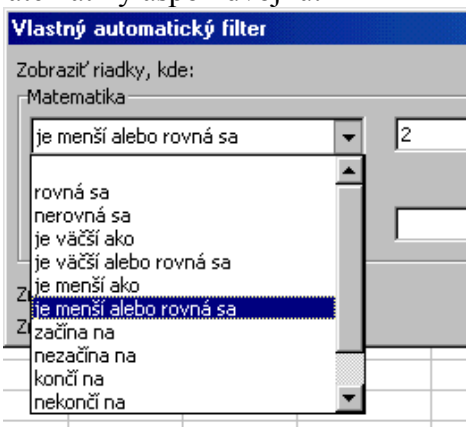
Ďalej si môžeme zvoliť, podľa ktorých kritérií sa má tabuľka zoraďovať



Filtrovanie dát

sa robí, keď chceme vidieť iba vybrané dáta. Dáta v Excelu ostanú, iba ich nebude vidno, ale kedykoľvek môžeme skrývanie nevhodných dát zrušiť. Pri filtrovaní postupujeme cez volby `Údaje-Filter-Automatický filter` - vedľa názvov stĺpcov sú šípky -voľby `Všetko (All), Prvých 10 (Top 10), Vlastné` ⇒ automatický filter (pre stĺpec až dve kritériá), vedľa kritéria hodnotu, s ktorou porovnávame. Napríklad môžeme vyberať z tabuľky ohodnotení krúžku iba študentov, ktorí majú z matematiky aspoň dvojku.





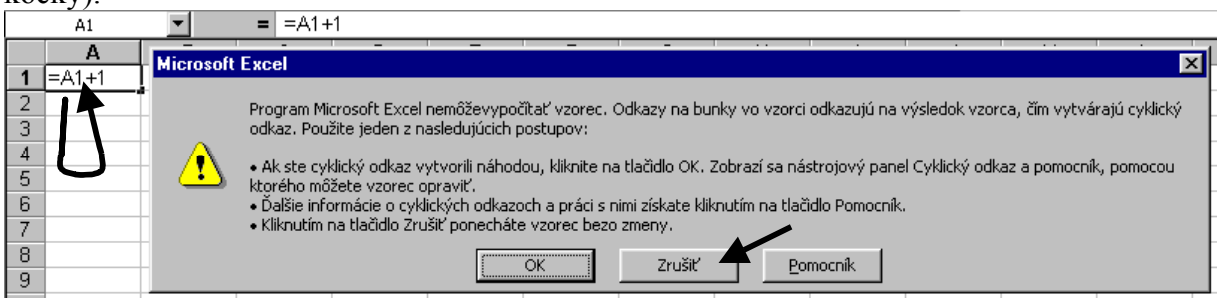
Keď chceme vidieť zasa celú tabuľku, v ponuke `Údaje-Filter-Automatický filter` zrušíme zaškrtnutie.

Pre filter podľa vlastných kritérií si dopredu napíšeme do buniek vzorce kritérií a použijeme voľbu `Údaje-Filter-Rozšírený filter.`

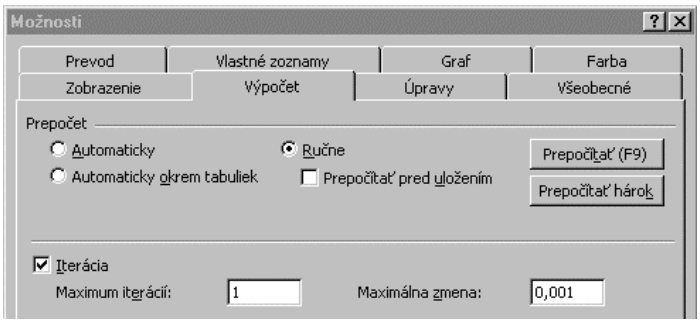
Cyklické výpočty

Keď použijeme bunku so vzorčekom, kde je odkaz na bunku samotnú, alebo keď použijeme v rôznych bunkách dva alebo viac vzorčekov, ktoré sa v "orientovanom cykle" odkazujú navzájom na seba, objaví sa varovné hlásenie. Treba stlačiť `Zrušiť`, aby sme mohli ďalej počítať.

Zásadou kvalitného výpočtu je použitať podmienku IF(splnenie podmienky dovoľujúcej výpočet; vzorec s cyklickým odkazom; počiatočná hodnota bunky). Keby sme napríklad v bunke A1 mali iba vzorček =A1+1, fungovalo by postupné pričítanie, ale nemohli by sme výpočet opakovať od začiatku (to môžeme napríklad v ďalšom príklade pri hre v kocky).



Potom ideme na voľbu z lišty `Nástroje-Možnosti-Výpočet` , nastavíme voľby ako v nasledujúcom paneli a stláčame `F9`



Hra v kocky proti počítaču alebo použitie náhodných čísel

Použitie náhodných čísel môže byť niekedy veľmi užitočné. Ako príklad uvedieme nastavenie. kedy sa môžeme staviť s počítačom o výsledok hry v kocky a zadávame výšku stávky, kedy aj počítač aj vy máte dva hody. Excel počíta aj Vašu celkovú výhru alebo prehru:

F2 = =IF(G2=1;IF(A2+B2>C2+D2;F2-E2;IF(A2+B2=C2+D2; F2;F2+E2));0)

	A	B	C	D	E	F	G	H
1	Počítač		Ja		Stávka	Celková výhra	Referenčná bunka	
2	3	3	4	6	3	15	1	

Do bunky `A1` napíšte Počítač, do bunky `C1` napíšte Ja, do bunky `E1` Stávka, do bunky `F1` Celková výhra, do bunky `H1` Referenčná bunka. Do buniek `A2:D2` vložte vzorec `=ROUNDDOWN(RAND()*6;)+1` ktorý počíta náhodné celé číslo od 1 po 6 ako výsledok hodu kocky. Do bunky `E2` vložte číslo odpovedajúce výši vašej stávky. Do bunky `F2` vložte vzorec `=IF(G2=1;IF(A2+B2>C2+D2;F2-E2;IF(A2+B2=C2+D2;F2;F2+E2));0)` do bunky `G2` vložte 0, stlačte `F9`, potom vložte 1 a stláčajte `F9` a prípadne meňte výšku stávky v bunke `E2`.

Hra Life

Nasleduje popis možnosti, ako nastaviť Excel tak, aby bol schopný hrať hru „Life“, uvedenú v kapitole 9.

1. Otvoríme si nový, štvrtý hárok pomocou `Vložiť-Pracovný hárok (Insert-Worksheet)` a do jeho bunky `A1` vložíme hodnotu `TRUE`.
2. V prvom, druhom a treťom hárku urobiť `Úpravy-Prejsť na-Odkaz: A1:AN25 – OK – Formát – StĺpEc – Šírka – Šírka stĺpca: 2 OK`, kliknúť na nejakú bunku, aby sa zrušilo vysvietenie `(Edit-Go to Reference: A1:AN25 – OK – Format – Column – Width – Column Width 2 OK)`
3. Do bunky `B2` v hárku 1 vložíme vzorec
   `=IF(Hárok4!$A$1;Hárok3!B2;IF(Hárok2!A1+Hárok2!B1+Hárok2!C1+Hárok2!A2+Hárok2!C2+Hárok2!A3+Hárok2!B3+Hárok1!C3=3;1;IF(Hárok2!A1+Hárok2!B1+Hárok2!C1+Hárok2!A2+Hárok2!C2+Hárok2!A3+Hárok2!B3+Hárok2!C3=2;Hárok1!B2;0)))`
   `(=IF(Sheet4!$A$1;Sheet3!B2;IF(Sheet2!A1+Sheet2!B1+Sheet2!C1+Sheet2!A2+Sheet2!C2+Sheet2!A3+Sheet2!B3+Sheet1!C3=3;1;IF(Sheet2!A1+Sheet2!B1+Sheet2!C1+Sheet2!A2+Sheet2!C2+Sheet2!A3+Sheet2!B3+Sheet2!C3=2;Sheet1!B2;0))) )`
   potiahnutím ho skopírujeme až do `AM2` a potom celý riadok potiahnutím skopírujeme až do `AM24`
4. Do bunky `A1` v hárku 2 vložíme vzorec `=Hárok1!A1 (=Sheet1!A1)` a potiahnutím ho skopírujeme až do `AN1` a potom celý riadok potiahnutím skopírujeme až do `AN25`

5. V hárku 1 použijeme Úpravy-Prejsť na-Odkaz: A1:AN25 – OK –Formát – Podmienené formátovanie (Edit-Go to Reference: A1:AN25 – OK – Format – Conditional Formatting) a zadáme Podmienka 1 hodnota bunky rovná sa, klikneme na Formát (Format), a zadáme Písmo Farba: (Font Color) a klikneme na šipku dole a na červenú, a Vzorky Farba: (Patterns Color:) klikneme na červenú. Potom ideme na Pridať> (Add>>) a zadáme Podmienka 2 hodnota bunky rovná sa 0 (Condition 2 Cell Value is equal to 0) klikneme na Formát (Format), a zadáme Písmo Farba: (Font Color) a klikneme na bielu, a Vzorky Farba: (Patterns Color:) klikneme na bielu.
6. Do buniek v hárku 3 náhodne napíšeme do buniek jednotky, napr. tak, aby v rozmedzí A2:AM24 tvorili zopár „kĺzakov" – viď obrázok vyššie.
7. Stlačíme F9. Nastavíme sa do hárku 4, zmeníme hodnotu v A1 z TRUE na FALSE, nastavíme sa do hárku 1 a stláčame F9 a pozorujeme, ako sa obrazovka mení.

Úlohy

Úloha 14.1. Vygenerujte v stĺpci A aritmetickú postupnosť 2,6,10,... až do čísla 500.

Úloha 14.2. Vygenerujte v stĺpci A geometrickú postupnosť 1,1.5,2.25 ... až do čísla menšieho alebo rovného 2000.

Úloha 14.3. Vytvorte rad, kde prvé dve čísla sú 1,1, a každé ďalšie je dvojnásobkom súčtu predchádzajúcich dvoch, až do 10^{10}.

Úloha 14.4. Vytvorte geometrický rad začínajúci 1,2,4, a zistite, koľko sa dá zobraziť jeho členov, dokiaľ veľkosť čísla neprekročí maximálnu hodnotu zobraziteľnú v Exceli.

Úloha 14.5. Zaplňte maticu s ľavým horným rohom v A1 a pravým dolným rohom v T200 číslom 7 v každej bunke.
Návod na riešenie:
Napíšte číslo 7 do bunky A1, vyberte pole A1:T200 (napríklad pomocou voľby z lišty hlavnej ponuky Úpravy - Prejsť na... Odkaz A1:T200) a použite voľbu z lišty hlavnej ponuky Úpravy - Vyplniť - Dolu a Úpravy - Vyplniť - Vpravo

Úloha 14.6. Vytvorte v prvom stĺpci sadu zlomkov 1/3,2/9,4/27 ... až dokiaľ dolný zlomok bude mať viac ako 4 cifry. Zlomky musia byť naformátované v tvare zlomku, nie ako desatinné číslo (pozrite sa no formát bunky, číslo).
Návod na riešenie:
Použite voľbu z lišty hlavnej ponuky Formát - Bunky - Číslo - Kategória: Vlastné, Typ: ????/????

Úloha 14.7. Vytvorte prvých 50 čísel radu, kde prvé tri čísla sú 1,1,1 a každé ďalšie je trojnásobkom súčtu predchádzajúcich troch

Úloha 14.8. Vytvorte rad 100 čísel $\sin(30^0)$, $\sin(\sin(30^0))$, $\sin(\sin(\sin(30^0)))$.(pozor na prevod stupne - radiány).

Úloha 14.9. Vytvorte dva rady čísel, prvý 1,2,3,...n, druhý vedľa neho $\sqrt{\frac{k^3}{5}}$, kde k je číslo z vedľajšieho radu.

Úloha 14.10. Vytvorte tabuľku hodnôt y/x, kde x aj y budú hodnoty od 10 po 20. Skopírujte časť tabuľky pre x a y od 11 do 19 do druhého hárku tak, že skopírujete iba hodnoty, nie vzorce.

Návod na riešenie:

Pozrite sa na prípravu tabuľky malej násobilky v úvodnom texte. Kopírovanie urobte pomocou vysvietenia buniek, potom použite `Ctrl C`, nastavte sa na miesto, kde chcete kopírovať a užite voľbu z lišty hlavnej ponuky `Úpravy - Prilepiť špeciálne - Hodnoty`

Úloha 14.11. Vytvorte v Exceli vzorček, ktorý z bodového ohodnotenia v bunke A1 automaticky vytvorí známku v bunke A2, kde 1-59 bodov je ohodnotené známkou 4, 60-65 je ohodnotené známkou 3, 66-80 bodov je ohodnotené známkou 2, a 81-100 bodov ohodnotené známkou je 1.

Návod na riešenie:

Použite trikrát do seba vnorenú funkciu IF podobne ako v prípade modelovania hry v kocky ku konci tejto kapitoly, automatické hodnotenie známky z bodov v bunke `A2` je

`=IF(A1<60;4; IF(A1<66;3;IF(A1<81;2;1))).`

Úloha 14.12. Vytvorte ohodnotenie 3 ľudí z dvoch predmetov v hárku 1 bodmi od 1 po 100(ľudia v stĺpci, predmety v riadku), a vypočítajte priemer z obidvoch predmetov dole vzorčekom. Rovnako to urobte v hárku 2 pre 4 ľudí, a v hárku 3 vyjadrite pomocou vzorčekov priemery hodnôt pre obidva hárky (nie priemerov!) z obidvoch predmetov zvlášť, a pre celkový priemer všetkých hodnôt. V prvom a druhom hárku vedľa ohodnotení bodmi uveďte aj ohodnotení známkami (automaticky pomocou funkcie využívajúcej do seba vnorené if, teda =if(podmienka1;výraz1;if(podmienka2;výraz2;výraz3))), kde 1-55 bodov je 3, 56-79 bodov je 2, a 80-100 bodov je 1.

Návod na riešenie:

Priemer zo sady hodnôt z rôznych hárkov sa počíta napr. ako

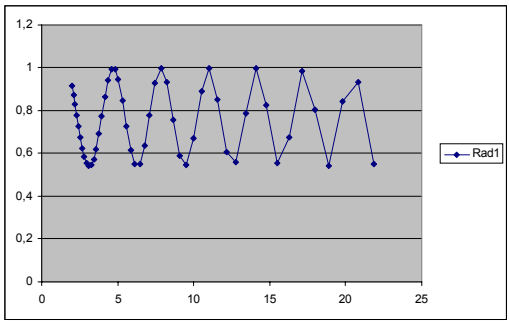
`=AVERAGE(Hárok1!B2:C4; Hárok2!B2:C5)`

a automatické hodnotenie známky z bodov urobte podobne ako v predchádzajúcej úlohe.

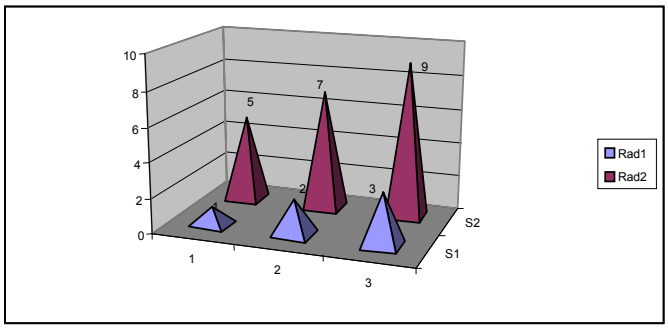
Úloha 14.13. Vytvorte v Exceli Heronov vzorček pre výpočet plochy trojuholníka s dĺžkami strán a,b,c, kde plocha $A=\sqrt{s(s-a)(s-b)(s-c)}$ a s je polovina obvodu trojuholníka. Dĺžky strán budú v bunkách A1, A2, a A3.

Úloha 14.14. Vložte si do Excelu funkciu AND a prečítajte si, v akom formáte sa používa. Na základe tejto informácie vytvorte v Exceli kontrolu, či čísla zadané v bunkách A1, A2, a A3 ako dĺžky strán môžu vôbec tvoriť trojuholník, teda vytvorte vzorček, ktorý pri splnení podmienok a>0, b>0, c>0, a+b>c, a+c>b, b+c>a vypíše "Je to trojuholník", inak vypíše "Nie je to trojuholník" (použite pre spojení podmienok funkciu AND)

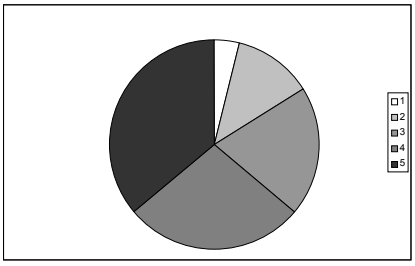
Úloha 14.15. Vytvorte v Exceli tabuľku, kde v prvom stĺpci bude sada 50 hodnôt geometrického radu s počiatkom v 2 a koeficientom 1,05, v druhom stĺpci budú hodnoty funkcie cos argumentov z prvého stĺpca a v treťom stĺpci budú hodnoty cos(cos(x)) argumentov x z prvého stĺpca. Vytvorte graf x-y závislosti hodnôt tretieho stĺpca na prvom stĺpci, keď body sú spojené úsečkami. Výsledok by mal vyzerať nasledovne:



Úloha 14.16. Vytvorte dole uvedený graf dvoch radov hodnôt 1,2,3 a 5,7, 9 s tým, že hodnoty budú automaticky uvedené u ihlanov



Úloha 14.17. Z hodnôt 1,2,3, ..., 10 v stĺpci A vytvorte koláčový graf z tým, že zoberiete do grafu každú druhú hodnotu a graf nebude mať farebné dieliky, ale rôzne druhy sivej



Úloha 14.18. Vytvorte graf zobrazujúci závislosť hodnôt funkcie $\frac{1}{(\log r)\sqrt{r}}$ na x podľa tabuľky

```
x 2   5  7  9 12 14
r 17 15 12  6  5  3
```

a zobrazte tiež priamku lineárnej závislosti, získanú metódou najmenších štvorcov, spolu s jej rovnicou a regresným koeficientom

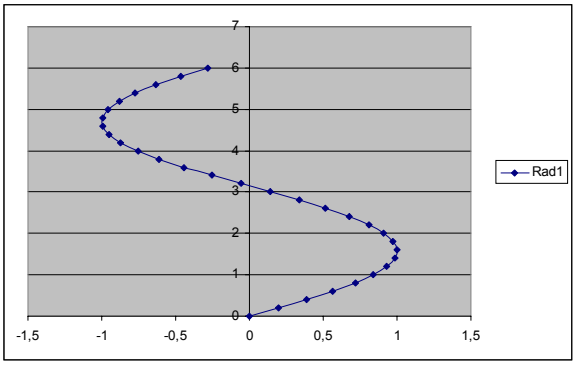
Úloha 14.19. Vytvorte v Exceli graf zobrazujúci závislosť hodnôt x na y podľa tabuľky

```
x 2   5  7 9 12 14
r 17 15 12 6  5  3
```

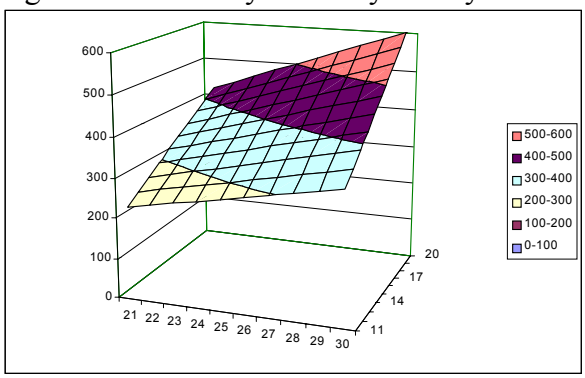
a zobrazte tiež priamku lineárnej závislosti, získanú metódou najmenších štvorcov, spolu s jej rovnicou a regresným koeficientom; rovnicu posuňte tak, aby sa neprekrývala s grafom.

Úloha 14.20. Vytvorte x-y závislosť zo sady hodnôt x: 1, 2, 3, 4, 5 a y: 1, 2,5 4 6 9. Do grafu pridajte trendovú čiaru a rovnicu pre preloženie hodnôt krivkou odpovedajúcou polynómu tretieho stupňa.

Úloha 14.21. Vytvorte rad 0 0,2 0,4 ... po 6, v stĺpci A, v stĺpci B vytvorte hodnoty SIN týchto uhlov a zobrazte v nasledujúcom grafe



Úloha 14.22. Vytvorte v Exceli pomocou použití kopírovania jedného vzorčeku tabuľku všetkých možných násobkov X×Y, kde X sú čísla od 11 do 20 v riadku a Y sú čísla od 21 do 30 v stĺpci. Zobrazte výsledok ako trojrozmerný graf, kde Y bude na "vodorovnej" osi ako na obrázku. Odstráňte sivé zobrazenie plôch X-Y, X-Z a Y-Z v grafu v Exceli. Výsledok by mal vyzerať nasledovne



Návod na riešenie:
Pri odstraňovaní sivej výplne plôch kliknite pravým tlačítkom myši na graf.

Úloha 14.23. Vytvorte v Exceli tabuľku hodnôt funkcie cos (x y), kde x ide v riadku od -0,5 do 0,5 po 0.05 a y ide v stĺpci od 0 do 3 po 0.2. Výslednú tabuľku zobrazte trojrozmerným povrchovým grafom, kde ale osa y musí byť zobrazená zhruba vodorovne a osa x v približne 45^0 uhle. Graf skopírujte do dokumentu MS Wordu
Návod na riešenie:
Pozrite sa na prípravu tabuľky malej násobilky v úvodnom texte.

Úloha 14.24. Vytvorte tabuľku hodnôt funkcie sin(x y), kde x ide v riadku od -1 do 1 po 0.1 a y ide v stĺpci od 0 do 4 po 0.2. Výslednú tabuľku zobrazte trojrozmerným povrchovým grafom, kde ale osa x musí byť zobrazená vodorovne a osa y v 45^0 uhle.
Návod na riešenie:
V riadku 1 do buniek `B1`, `C1`, ... dajte -1 -0.9 -0.8 ... 1
v stĺpci `A` do buniek `A2`, `A3`, ... dajte 0 0,2 0,4 ... 4
do bunky `B2` dajte vzorec `=SIN(B$1*$A2)` a vzorec skopírujte do buniek `A2:V22`.
Pri vysvietenom výberu buniek `A2:V22` stlačte ikonku `Sprievodca grafom`, vyberte `Povrchový`, `3D- povrchový` a dajte `Dokončiť`. Otáčanie grafu urobte podľa návodu uvedeného v úvodnom texte.

Úloha 14.25. V prvom stĺpci vytvorte rad 300 opakujúcich sa symbolov a,b,c,a,b,c ..., v druhom stĺpci rad 1,2,3 ...300 a v treťom stĺpci rad sin x, kde x sú hodnoty z druhého stĺpca. Celú tabuľku zoraďte tak, aby najprv boli uvedené riadky s prvým symbolom rovným a, potom s b a nakoniec s c, a v rámci a-čok, b-čok a c-čok aby riadky boli usporiadané zostupne podľa veľkosti hodnôt v treťom stĺpci.

Návod na riešenie:

Do bunky `A1` dajte a do `A2` dajte b, do `A3` dajte c, vysvieťte tieto 3 bunky a potiahnite za pravý dolný roh až do riadku 300. Do bunky `B1` dajte 1 do `B2` dajte 2, vysvieťte tieto 2 bunky a potiahnite za pravý dolný roh až do riadku 300. Do bunky `C1` dajte `=SIN(B1)`, vysvieťte bunku a potiahnite za pravý dolný roh až do riadku 300. Vysvieťte všetky hodnoty, choďte na ponuku `Údaje - Zoradiť`, dajte `Zoradiť podľa: stĺpca A` o `Vzostupne` a pod tým `Zoradiť podľa: stĺpca C` o `Zostupne`.

Úloha 14.26. Vytvorte rad čísel 1,2,...1000 a pomocou podmieneného formátovania (namiesto voľby "hodnota" použite "vzorec" a pravým tlačítkom myši - "Čo je to" - zistite, ako sa používa) vysvieťte všetky čísla deliteľné piatimi bezo zvyšku červenou farbou.

Úloha 14.27. Vytvorte pole čísel tvorené trojuholníkom, kde v stĺpci A sú samé hodnoty 1 a v bunkách v druhom a v ďalších riadkoch a v druhom a v ďalších stĺpcoch je vždy súčet bunky priamo hore a bunky vľavo hore (tzv. Pascalov trojuholník, aj keď posunutý doľava). Takýto trojuholník sa využíva napr. na kombinačné čísla teda $\binom{4}{2}=6$ znamená, že v štvrtom riadku a druhom stĺpci je číslo 6 (riadky a stĺpce indexujeme od nuly). Ďalej je tento trojuholník známy u napr. u koeficientov mocnín dvojčlenov, napr. $(a+b)^3=a^3+3a^2 b + 3 ab^2 + b^3$, kde 1 3 3 1 je v treťom riadku Pascalovho trojuholníka. V druhom stĺpci sú tiež tzv. trojuhelníkové čísla 1+2=3,1+2+3=6, 1+2+3+4=10 ...

Pomocou podmieneného formátovania hodnotu bunky medzi 100 a 2000 zobrazte s modrým pozadím, s červeným písmom a orámovane, a hodnoty nad 30000 zobrazte so žltým orámovaním.

Úloha 14.28. Vytvorte v Exceli vzorček, ktoré po stlačení klávesy F9 vygeneruje náhodné číslo od 1 do 10.

Návod na riešenie:

Pozrite sa na generovanie náhodného čísla u hry v kocky v úvodnom texte.

Úloha 14.29. Vytvorte vzorček odkazujúci sa na toto vygenerované náhodné číslo, ktorý pre čísla od 1 do 5 vypíše "Prehral som" a pre čísla od 6 do 10 "Vyhral som".

Úloha 14.30. Vytvorte v stĺpci A sadu hodnôt 1,2,3, ..., 10 pomocou takých vzorčekov, že keď stlačíte klávesu F9, všetky hodnoty sa zvýšia o jednotku, teda po prvom stlačení na 2, ..., 11.

Návod na riešenie:

Pozrite sa na cyklické vzorce v úvodnom texte.

Úloha 14.31. Zobrazte animáciu priebehu funkcie sin x, keď držíte stlačené F9. V druhom hárku v bunke `A1` nastavte hodnotu 1. V prvom hárku v prvom stĺpci nastavte v prvých dvadsiatich bunkách takú funkciu, že pokiaľ je bunka A1 v hárku 2 rovná 1, potom je v prvých dvadsaťjedna bunkách aritmetický rad začínajúci v 0 a pokračujúci po 0,1 až po 2. Pokiaľ je hodnota A1 v hárku 2 rovná 0, potom nastavte možnosti výpočtu a funkciu v prvej bunke tak, že vždy po stlačení F9 sa k momentálnej hodnote pripočíta 0,1. V prvej bunke je teda funkcia if, s cyklickým odkazom sama na seba (musíte povoliť cyklický výpočet a nastaviť

iteráciu na manuálne a na 1). V druhej a ďalších bunkách sa za základ výpočtu zoberú hodnoty bunky o riadok vyššie. V druhom stĺpci vytvorte hodnoty funkcie sin s argumentom z prvého stĺpca naľavo. Zobrazte aj X-Y závislosť týchto dvoch stĺpcov.

Návod na riešenie:

Do bunky `A1` v Hárku2 dajte 1, do bunky `A1` dajte `=IF(Hárok2!A1=1;0;A1+0,1)`, do bunky `A2` dajte `=A1+0,1`, vyberte bunku a kopírujte nadol. Do bunky `B1` dajte `=SIN(A1)`, vyberte bunku a kopírujte nadol. Vyberte všetky numerické hodnoty v obidvoch stĺpcoch, stlačte ikonku `Sprievodca grafom`, vyberte `XY závislosť` a dajte `Dokončiť`. Do bunky `A1` v Hárku2 dajte 0, do výstražného panelu kliknite `Zrušiť`, kliknite na `Nástroje - Možnosti - Výpočet`, dajte `Ručne`, vykliknite `Iterácia`, do `Maximum iterácií` napíšte 1 a stlačte `OK`. Stláčajte klávesu `F9` a pozerajte sa na animáciu. Pre návrat na pôvodné hodnoty do bunky `A1` v Hárku2 dajte 1, stlačte klávesu `F9` a do bunky `A1` v Hárku2 dajte znova 0.

Úloha 14.32. Urobte si zadanie vzorcov pre hru Life popísané v tejto kapitole a doplňte vzorce do okrajových buniek hárku 1, teda do A1:AN1, do A24:AN24 a do stĺpcov A a AN tak, aby okraje obdĺžnika boli spojené do tzv. toroidu (ľudovo pneumatiky), tak že vrchný riadok má za susedný riadok spodný a naopak, a stĺpec A je susedný so stĺpcom AN, teda aspoň čo sa prežívania buniek týka. V doteraz prezentovanom modeli útvary zvané „kĺzáky“ popísané v kapitole 9 spadnú cez okraj a zmiznú, u toroidu by sa mali objaviť na opačnej strane obrazovky.

Úloha 14.33. Vytvorte si v Exceli tabuľku s číslami v stĺpci 1,3,2, a so súčtom týchto čísel. Skopírujte túto tabuľku do dokumentu MSWordu troma rôznymi spôsobmi

- tak, aby sa hodnoty v tabuľke dali meniť, ale aby sa ich súčet nemenil automaticky ("neexcelovskú" tabuľku vo Worde)
- tak, aby sa hodnoty v tabuľke dali meniť a aby sa ich súčet menil automaticky
- aby sa hodnoty nedali meniť

Vytvorte v Exceli z prvých troch hodnôt sĺpcový graf a skopírujte ho do dokumentu MSWordu dvoma rôznymi spôsobmi

- tak, aby sa hodnoty v grafe dali meniť tým, že na nich kliknete myšou a upravíte zdrojové údaje, na základe ktorých bol graf vytvorený ale ktoré normálne v MSWorde nevidno
- tak, aby sa graf dal meniť iba ako obrázok, tzn. že hodnoty sa dajú upravovať iba ako grafické objekty.

Návod na riešenie:

Pri kopírovaní použite voľbu `Úpravy - Prilepiť špeciálne`

Úloha 14.34. Vyriešte sústavu rovníc pomocou použitia vstavaných funkcií EXCELu MINVERSE a MMULT popísaných dole.

1x +2y +3z=13
3x +4y +6z=27
5x +7y +9z=44

MINVERSE(pole) vráti inverznú maticu z matice po a s rovnakým po tom riadkov a stĺpcov; musí ale byť vysvietené pole požadovaného rozmeru a po zadaní umiestnenia vstupného po a treba stlačiť kombináciu troch klávesov Ctrl+Shift+Enter, nie OK, lebo pri stlačení tlačidla OK sa zobrazí iba prvá hodnota matice.

MMULT(pole1; pole2) vráti maticu so sú inom polí, ktorá obsahuje rovnaký počet riadkov ako pole1 a rovnaký počet stĺpcov ako pole2; musí ale by vysvietené pole požadovaného rozmeru a po zadaní umiestnenia vstupných polí treba stlačiť kombináciu troch klávesov Ctrl+Shift+Enter, nie OK, lebo pri stlačení tlačidla OK sa zobrazí iba prvá hodnota matice.

Literatúra

[1] Csontó, J., Palko, M.: *Umelý život*. Elfa Press, Košice, 2003.

[2] Csontó, J.: *Umelá inteligencia v príkladoch*. Vydavateľstvo TU, Košice, 1995.

[3] Gahér, F.: *Logika pre každého*. IRIS, Bratislava, 1998.

[4] Gáliková, S.: *Úvod do filozofie mysle*. Vydavateľstvo Honner, Martin, 2001.

[5] Haykin, S.: *Neural Networks. A Comprehnesive Foundation*. Prentice-Hall, Upper Saddle River, NJ, 1999.

[6] Hillis, D.: *Obrazce v kameni*. Kalligram, Bratislava, 2002.

[7] Holland, J. H.: *Hidden Order. How Adaptation Builds Complexity*. Perseus Book, Cambridge, MA., 1995.

[8] Hopcroft, J. E., Ullman, J. D.: *Formálne jazyky a automaty*. Alfa, Bratislava, 1978.

[9] Jirků, P., Vejnarová. J.: *Logika - Neformální úvod do formální logiky*. VŠE, Praha, 2000.

[10] Kelemen, J., Ftáčnik, M., Kalaš, I., Mikulecký, P.: *Základy umelej inteligencie*. Alfa, Bratislava, 1992.

[11] Kelemen, J.: *Myslenie, počítač ...* . Spektrum, Bratislava, 1989.

[12] Kelemen, J.: *Strojovia a agenty*. Archa, Bratislava, 1994.

[13] Kokles, M., Romanová, A.: *Informačný vek*. Sprint vfra, Bratislava, 2002.

[14] Kvasnička, V., Beňušková, Ľ., Farkaš, I., A. Kráľ, Pospíchal, J. a Tiňo, P.: *Úvod do teórie umelých neurónových sietí*. IRIS, Bratislava, 1997.

[15] Kvasnička, V., Pospíchal, J., Tiňo, P.: *Evolučné algoritmy*. Vydavateľstvo STU, Bratislava, 2000.

[16] Mařík, V., Štepánková, O., Lažanský, J. a kol.: *Umělá inteligence I-III*, Academia, Praha, 1993, 1997, 2000.

[17] McCulloch, W. S. and Pitts, W.: A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* **5** (1943) 115-133.

[18] Minsky, M. and Papert, S.: *Perceptrons. An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1969.

[19] Minsky, M. L.: *Computation. Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, NJ, 1967.

[20] Molnár, Ľ., Češka, M., Melichar, B.: *Gramatiky a jazyky*. Alfa, Bratislava, 1987.

[21] Návrat, P. a kol.: *Umelá inteligencia*, Vydavateľstvo STU, Bratislava, 2002.

[22] Popper, K. R.: Logika vědeckého bádání, Praha, OIKOYMENH, 1997.

[23] Pstružina, K.: *Svět poznávaní. K filozofickým základům kognitívní vědy*. Nakladatelství Olomouc, 1998.

[24] Russel, S., Norwig, P.: *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 1995.

[25] Rybár, J. a kol.: *Filozofia a kognitívne vedy*. IRIS, Bratislava, 2002.

[26] Rybár, J., Benušková, Ľ. a Kvasnička, V. (editori): *Kognitívne vedy*. Kalligram, Bratislava, 2002.

[27] Sinčák P. a Andrejková, G.: *Neurónové siete I a II. (Inžiniersky prístup)*. ELFA Press, Košice, 1996.

[28] Sochor, A.: *Klasická matematická logika*. Karolinum, Praha, 2001.

[29] Sternberg, R. J.: *Kognitívní psychologie*. Portál, Praha, 2002.

[30] Šefránek, J.: *Inteligencia ako výpočet*. IRIS, Bratislava, 2000.

[31] Šíma, J., Neruda, R.: *Teoretické otázky neurónových sítí*. MATFYZPRESS, Praha, 1996.

[32] Švejdar, V.: *.Logika: neúplnost, složitost a nutnost.* Academia, Praha, 2002.

[33] Töpfer, P.: Algoritmy a programovací techniky. Prometheus, Praha, 1995.

[34] Višňovský, E., Popper, M., Plichtová, J. (ed.): *Príbehy o hľadaní mysle*. Veda, Bratislava, 2001.

[35] I. Zelinka: *Umělá inteligence - hrozba nebo naděje*? BEN, Praha, 2003.

[36] Znám, S., Bukovský, Ľ., Hejný, M., Hvorecký, J., Riečen, B.: *Pohľad do dejín matematiky*. Alfa, Bratislava, 1986.