

I. Matematické základy informatiky

Update: 11. května 2018

Obsah

1	Konečné automaty, regulární výrazy, uzávěrové vlastnosti třídy regulárních jazyků.	2
2	Bezkontextové gramatiky a jazyky. Zásobníkové automaty, jejich vztah k bezkontextovým gramatikám.	6
3	Matematické modely algoritmů - Turingovy stroje a stroje RAM. Složitost algoritmu, asymptotické odhady. Algoritmicky nerozhodnutelné problémy.	7
4	Třídy složitosti problémů. Třída PTIME a NPTIME, NP-úplné problémy.	14
5	Jazyk predikátové logiky prvního řádu. Práce s kvantifikátory a ekvivalentní transformace formulí.	18
6	Pojem relace, operace s relacemi, vlastnosti relací. Typy binárních relací. Relace ekvivalence a relace uspořádání.	21
7	Pojem operace a obecný pojem algebra. Algebry s jednou a dvěma binárními operacemi.	24
8	FCA – formální kontext, formální koncept, konceptuální svazy. Asociační pravidla, hledání často se opakujících množin položek.	26
9	Metrické a topologické prostory – metriky a podobnosti.	32
10	Shlukování.	33
11	Náhodná veličina. Základní typy náhodných veličin. Funkce určující rozdělení náhodných veličin.	34
12	Vybraná rozdělení diskrétní a spojitě náhodné veličiny - binomické, hypergeometrické, negativně binomické, Poissonovo, exponenciální, Weibullovo, normální rozdělení.	35
13	Popisná statistika. Číselné charakteristiky a vizualizace kategoriálních a kvantitativních proměnných.	36
14	Metody statistické indukce. Intervalové odhady. Princip testování hypotéz.	37

1 Konečné automaty, regulární výrazy, uzávěrové vlastnosti třídy regulárních jazyků.

1.1 Konečné automaty (KA)

Konečný automat (KA) tvoří množina stavů, vstupní abeceda, přechodová funkce, počáteční a koncové stavy. Můžeme jej znázornit jako **tabulku**, **graf** či **strom**.

Konečné automaty se dělí na **deterministické** a **nedeterministické**. Deterministický konečný automat má pouze jeden počáteční stav a přechodová funkce vrací jeden stav. Zatímco nedeterministický KA může mít více počátečních stavů a přechodová funkce vrací množinu stavů.

- **Slovo** přijaté automatem je taková sekvence symbolů (ze vstupní abecedy), pro kterou automat skončí v koncovém stavu.
- **Regulární jazyk** je takový jazyk (množina slov) který lze popsat konečným automatem.

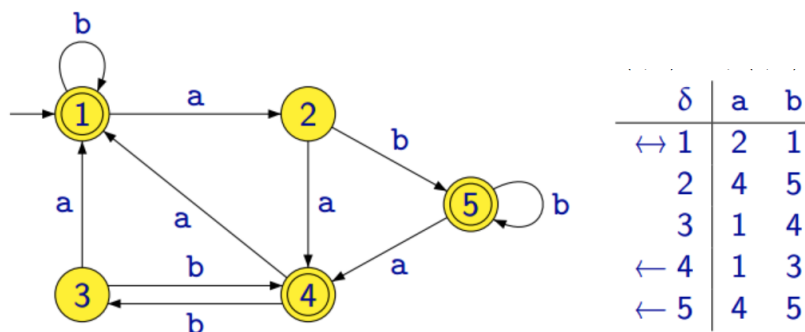
1.1.1 Deterministický konečný automat (DKA)

Skládá se ze **stavů** a **přechodů**. Jeden ze stavů je označen jako **počáteční stav** a některé jsou označeny jako **přijímací**. Je definován jako uspořádaná pětice $(Q, \Sigma, \delta, q_0, F)$, kde:

- Q je konečná neprázdná množina **stavů**.
- Σ (*sigma*) je konečná neprázdná množina vstupních symbolů, tzv. **vstupní abeceda**.
- δ (*delta*) je **přechodová funkce**, $\delta : Q \times \Sigma \rightarrow Q$.
- q_0 je **počáteční stav**, $q_0 \in Q$.
- F je neprázdná množina **koncových** neboli **přijímajících stavů**, $F \subseteq Q$.

Příklad

- $Q = \{1, 2, 3, 4, 5\}$, $\Sigma = \{a, b\}$, $F = \{1, 4, 5\}$
- $\delta(1, a) = 2$; $\delta(1, b) = 1$; $\delta(3, a) = 1$; $\delta(3, b) = 4$; $\delta(2, a) = 4$; $\delta(2, b) = 5$; $\delta(4, a) = 1$; $\delta(4, b) = 3$; $\delta(5, a) = 4$; $\delta(5, b) = 5$



1.1.2 (Zobecněný) Nedeterministický konečný automat ((Z)NKA)

Formálně je NKA definován jako pětice $A = (Q, \Sigma, \delta, I, F)$, s tím rozdílem, že oproti deterministickému KA má **více počátečních stavů** a **přechodová funkce vrací množinu stavů**. V případě ZNKA zde existují navíc **nulové epsilon** (ϵ) přechody:

- δ je přechodová funkce, vrací množinu stavů, $\delta : Q \times \Sigma \rightarrow P(Q)$, v případě **ZNKA** $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow P(Q)$.
- I je konečná množina počátečních stavů, $I \in Q$.



Na rozdíl od deterministického automatu:

- Může z jednoho stavu vést **libovolný počet přechodů** označených stejným symbolem (i **nulové** ϵ v případě ZNKA).
- Není zde nutné, aby z každého stavu vystupovaly všechny symboly, které do něj vstoupily \rightarrow **nemusí ošetřovat všechny varianty**, pouze odhadne, kterou cestou půjde.
- Nedeterministický automat přijímá dané slovo, jestliže **existuje alespoň jeden jeho výpočet**, který vede k přijetí tohoto slova.
- V automatu může být **víc než jeden počáteční stav**.
- Lze ho **převést na deterministický** (formou tabulky). Při převodu automatu, který má n stavů může mít výsledný nedeterministický až 2^n stavů.

1.1.3 Normovaný tvar

Začnu v počátečním stavu a procházím navštívené stavy a vytvářím tabulku. Každý KA má **právě 1** normovaný tvar. Také lze tímto způsobem zjistit, zda jsou automaty **ekvivalentní**.

1.2 Regulární výrazy

Regulární výraz je **řetězec popisující celou množinu řetězců**, konkrétně **regulární jazyk**. Regulární výrazy také můžeme chápat jako jednoduchý způsob, jak **popsat konečný automat** umožňující generovat všechna možná slova patřící do daného jazyka.

V regulárních výrazech využíváme znaky **abecedy** a symboly pro **sjednocení**, **zřetězení** a **iterace** regulárních výrazů. Za regulární výraz se považuje i samotný znak abecedy (např. a) stejně jako **prázdné slovo** ϵ a **prázdný jazyk** \emptyset .

1.2.1 Definice regulárních výrazů

Regulární výrazy popisují jazyky nad abecedou $A = \Sigma : \emptyset, \epsilon, a$ (kde $a \in \Sigma$) jsou regulární výrazy:

- \emptyset označuje **prázdný jazyk**,
- ϵ označuje jazyk $\{\epsilon\}$,
- a označuje jazyk $\{a\}$.

Dále, jestliže α, β jsou regulární výrazy, pak i $(\alpha + \beta)$, $(\alpha \cdot \beta)$, (α^*) jsou regulární výrazy, kde:

- $(\alpha + \beta)$ označuje **sjednocení** jazyků označených α a β ,
- $(\alpha \cdot \beta)$ označuje **zřetězení** jazyků označených α a β ,
- (α^*) označuje **iteraci** jazyka označeného α .

Neexistují žádné další regulární výrazy než ty definované podle předchozích dvou bodů.

Příklady

Ve všech případech je $\Sigma = \{0, 1\}$:

- **01** (0 a 1) ... jazyk tvořený jedním slovem 01,
- **0+1** (0 nebo 1) ... jazyk tvořený dvěma slovy 0 a 1,
- **(01)*** ... jazyk tvořený slovy $\epsilon, 01, 0101, 010101, \dots$,
- **(0+1)*** ... jazyk tvořený všemi slovy nad abecedou $\{0, 1\}$,
- **(01)*111(01)*** ... jazyk tvořený všemi slovy obsahující podслово 111, předcházení i následované libovolným počtem slov 01,
- **(0+1)*00+(01)*111(01)*** ... jazyk tvořený všemi slovy, která buď končí 00 nebo obsahují podслово 111 předcházené i následované libovolným počtem slov 01,
- **(0+1)*1(0+1)*** ... jazyk tvořený všemi slovy obsahujícími alespoň jeden symbol 1,
- **0*(10*10*)*** ... jazyk tvořený všemi slovy obsahujícími sudý počet symbolů 1.

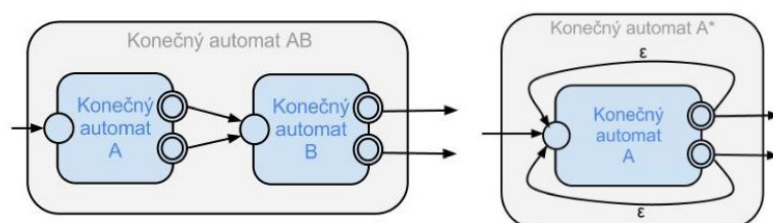
1.3 Uzávěrové vlastnosti třídy regulárních jazyků

Uzavřenost množiny nad operací znamená, že výsledek operace s libovolnými prvky z množiny bude opět spadat do dané množiny. Třidu regulárních jazyků značíme **REG**. Regulární výrazy (tedy i KA) jsou uzavřené vůči operacím:

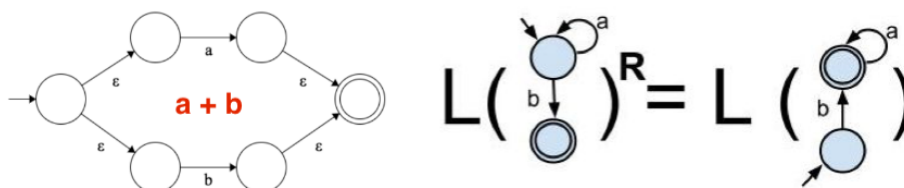
- **Sjednocení, průnik, doplněk** – je-li $L_1, L_2 \in \text{REG}$, pak také $L_1 \cup L_2, L_1 \cap L_2, L_1'$ jsou v REG.
- **Zřetězení, iterace** – je-li $L_1, L_2 \in \text{REG}$, pak také $L_1 \cdot L_2, L_1^*$ jsou v REG.
- **Zrcadlový obraz** – je-li $L \in \text{REG}$, pak také L^R jsou v REG.

1.3.1 Operace sjednocení, zřetězení, iterace a zrcadlový obraz u KA

- **Iterace** – spojíme koncové stavy jednoho KA s počátečními druhého KA ϵ přechodem. Na obrázku generuje automat A^* jazyk $L(A^*) = L(A)^*$, který je iterací jazyku generovaného modrého automatu A .
- **Zřetězení** – spojíme koncové stavy jednoho s počátečními stavy druhého. Na obrázku generuje konečný automat AB jazyk $L(AB) = L(A) \cdot L(B)$.



- **Sjednocení** – $L(A + B) = L(A) + L(B)$ získáme tak, že vytvoříme **nový počáteční stav**, ze kterého vedeme ϵ přechody do počátečních stavů obou automatů. Poté obdobě z koncových stavů obou automatů vedeme ϵ přechody do **nového koncového**.
- **Zrcadlový obraz** – pustíme automat pozpátku, celý jej převrátíme. **Přehodíme orientaci všech přechodů**, z počátečních stavů uděláme koncové a naopak.



- **Doplněk** – u DKA provedeme prohození označení přijímajících a ostatních stavů, u NKA je nejprve nutné provést převod na DKA.

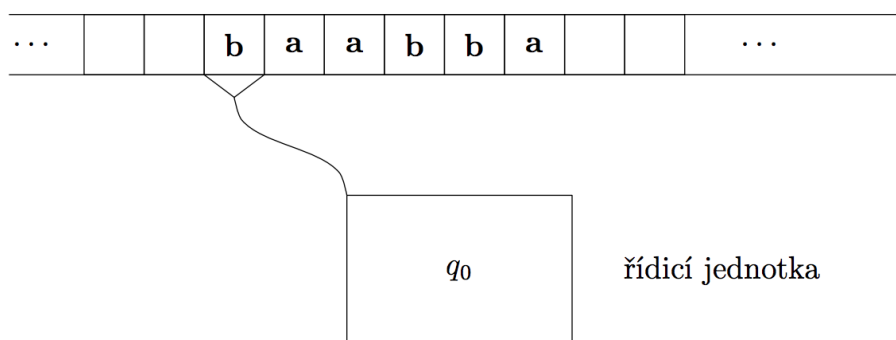
- 2 Bezkontextové gramatiky a jazyky. Zásobníkové automaty, jejich vztah k bezkontextovým gramatikám.

3 Matematické modely algoritmů - Turingovy stroje a stroje RAM. Složitost algoritmu, asymptotické odhady. Algoritmicky nerozhodnutelné problémy.

Ve snaze **popsat jakýkoliv algoritmus** si vymysleli matematici Turingovy a RAM stroje. Jde o dva různé přístupy (modely) univerzálních počítačů/programovacích jazyků. Jinými slovy těmito stroji lze **definovat a provést libovolný algoritmus**.

Historicky prvním „univerzálním programovacím jazykem“ byl Turingův stroj. Byl popsán dříve, ještě před rozmachem počítačů, proto se od reálného počítače (programování) podstatně liší, na rozdíl od RAM stroje. Turingův stroj například pracuje s **celou abecedou** zatímco RAM (podobně jako počítač) s **čísly**.

3.1 Turingův stroj (TS)



Turingův stroj je podobný konečnému automatu, ale má **oboustranně nekonečnou pásku** (je na ni zapsáno vstupní slovo), místo symbolu ϵ pro prázdné znaky se používá \square , **hlava** je **čtecí i zapisovací** a pohybuje se po pásce v **obou směrech**.

3.1.1 Formální definice TS

Turingův stroj, je definován jako šestice $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, kde:

- Q je konečná neprázdná množina **stavů**.
- Σ je konečná neprázdná množina **vstupních symbolů** (vstupní abeceda).
- Γ je konečná neprázdná množina **páskových symbolů**, kde $\Sigma \subseteq \Gamma$ a $\Gamma - \Sigma$ je (pří-
nejmenším) speciální znak \square (prázdný znak [Blank]).
- δ je přechodová funkce, $\delta : (Q - F) \times \Gamma \rightarrow Q \times \Gamma \times \{-1, 0, +1\}$.
- q_0 je **počáteční stav**, $q_0 \in Q$.
- F je množina **koncových stavů**, $F \subseteq Q$.

3.1.2 Definice instrukcí (pravidel) v TS

Podobně jako ZA lze konkrétní Turingův stroj zadat seznamem instrukcí. Tyto instrukce jsou opět dány přechodovou funkcí, význam instrukce: $(q, a) \rightarrow (q', a', m)$ je tento:

(akt. stav $[q]$, znak na pásce $[a]$) \rightarrow (**nový stav** $[q']$, **nový znak** $[a']$, **posun** $[\{-1; 0; +1\}]$)

3.1.3 Příklad

Tento příklad invertuje slovo, které je uvedené na úvodním obrázku u TS:

$$\begin{array}{ll} Q = \{q_1, q_2\} & (q_1, a) \rightarrow (q_1, b, +) \\ \Sigma = \{a, b, c, \square\} & (q_1, b) \rightarrow (q_1, a, +) \\ q_0 = q_1 & (q_1, \square) \rightarrow (q_2, \square, 0) \\ F = \{q_2\} & \end{array}$$

3.1.4 Modifikace TS

- **N-páskový TS** – čte a zapisuje do **více pásek** najednou, jediná změna je v přechodové funkci: $\delta : Q \times \Gamma^n \rightarrow Q \times (\Gamma \times \{L, R, N\})^n$.
- **N-hlavový TS** – má více čtecích hlav než klasický TS, každá hlava zapisuje/čte a pohybuje se **nezávisle na ostatních**.
- **Nedeterministický TS** – umožňuje výběr z více možností, pro jednu konfiguraci můžeme definovat **více pravidel**.

3.1.5 Základní pojmy

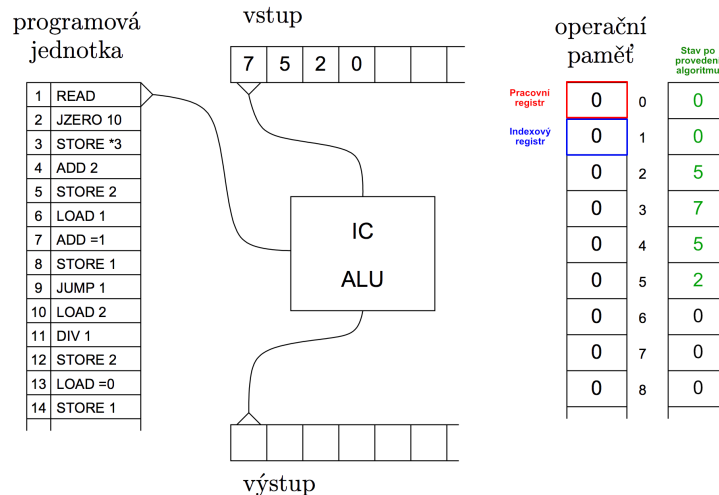
- **Turingovsky úplný** – stroj (počítač, programovací jazyk, úloha, ...), která má stejnou výpočetní sílu jako TS. Lze v něm **odsimulovat** libovolný jiný TS zadaný na vstupu.
- **Church-Turingova teze** – říká, že jakýkoliv výpočet lze úspěšně uskutečnit algoritmem běžícím na počítači, tedy „ke každému algoritmu existuje ekvivalentní TS“.

3.2 Model RAM (Random Access Machine)

RAM stroje již vycházejí ze skutečných počítačů, dá se tedy říct, že se jedná o jednoduchou abstrakci reálného procesoru s jeho strojovým kódem pracujícím s lineárně uspořádanou pamětí. Tento model slouží zejména k analýze algoritmů z hlediska (**paměťové, časové**) **složitosti**. Skládá se z těchto částí:

1. **Programová jednotka** – uchovává program, tvořený konečnou posloupností instrukcí.
2. **Neomezená pracovní paměť** – neomezená lineárně uspořádaná paměť, tvořená buňkami, do který lze zapisovat/číst celá čísla (\mathbb{Z}), adresovaná přirozenými čísly (\mathbb{N}) (0 = **pracovní** registr, 1 = **indexový** registr).

3. **Vstupní a výstupní páska** – lze na ně sekvenčně zapisovat/číst celá čísla (\mathbb{Z}).
4. **Centrální jednotka** – obsahuje programový register ukazující, která instrukce má být provedena. Ta se provede a programový registr se příslušně změní (zvýší o 1, o více v případě skoku).



Výše uvedený program vypočítá **aritmetický průměr**, který následně uloží do buňky paměti pod indexem č. **2**. Výsledek po dělení je roven 4,666 a po zaokrouhlení 5.

3.2.1 Instrukce a typy operandů RAM

Typ	Hodnota operandu
$= i$	přímo číslo udané zápisem i
i	číslo obsažené v buňce s adresou i
$*i$	číslo v buňce s adresou $i + j$, kde j je aktuální obsah indexového registru

Zápis	Význam
READ	do pracovního registru (PR) se načte vstup a hlava se posune doprava
WRITE	na výstup se zapíše hodnota PR
LOAD op	do PR se načte hodnota dána operátorem op
STORE op	hodnota PR se uloží na do registru daného operátorem op
ADD op	k hodnotě PR se přičte hodnota daná operátorem op
SUB op	od hodnoty v PR se odečte hodnota daná operátorem op
MUL op	PR se vynásobí hodnotou danou operátorem op
DIV op	PR se vydělí hodnotou danou operátorem op
JUMP $návěští$	provede se skok na instrukci danou $návěštím$
JZERO $návěští$	pokud je hodnota v PR rovna 0 , provede se skok na $návěští$
JGTZ $návěští$	pokud je hodnota v PR větší než 0 , provede se skok na $návěští$
HALT	korektní ukončení programu

3.3 Složitost algoritmů

Abychom mohli **porovnávat** různé algoritmy řešící stejný problém, zavádí se pojem složitost algoritmu. Složitost je jinak řečeno **náročnost algoritmu** – čím menší složitost tím je algoritmus lepší. Přičemž nás může zajímat složitost z pohledu **času**, či **paměti**:

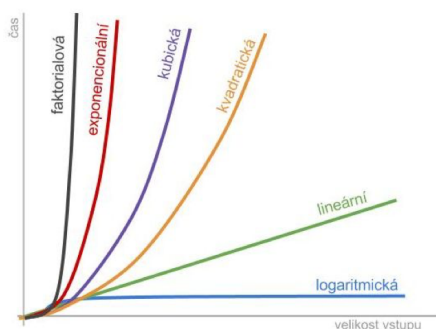
- **Časová složitost** – sleduje jak závisí **doba** výpočtu alg. na množství vstupních dat.
- **Prostorová složitost** – sleduje jak závisí **množství použité paměti** výpočtu alg. na množství vstupních dat.

Jelikož konkrétní čísla (čas, bity) se liší v **závislosti vstupních datech**, množství zpracovávaných dat a použitým programovacím jazyku, neudává se složitost číslu, nýbrž **funkcí závislou na velikosti vstupních dat**. Tato funkce se získá počítáním proběhlých instrukcí algoritmu sestaveném v univerzálním RAM stroji. A počítá se s nejhorším možným případem vstupu. To je důležité například u třídících algoritmů, kde hraje velkou roli to, jak moc už je vstupní pole setříděné (vstupuje-li do algoritmu už setříděná posloupnost čísel, algoritmus skončí okamžitě, zatímco s opačně seřazenými čísly se bude trápit dlouho.

3.4 Asymptotická notace

Je **způsob klasifikace počítačových algoritmů**. Ve většině případů nemusíme znát přesný počet provedených instrukcí a spokojíme se pouze s odhadem toho, jak rychle tento počet narůstá se zvyšujícím se vstupem. Asymptotická notace nám umožní **zanedbat méně důležité detaily** a **odhadnout** přibližně, **jak rychle daná funkce roste**. V souvislosti s asymptotickými odhady složitosti se používají tyto zápisy:

- $f \in O(g)$ – f roste **nejvýše tak rychle** jako g (f je ohraničena g **shora**) $[\leq]$.
- $f \in o(g)$ – f roste **(striktně) pomaleji** než g (f je ohraničena g **shora ostře**) $[<]$.
- $f \in \Theta(g)$ – f roste **stejně rychle** jako g $[=]$.
- $f \in \omega(g)$ – f roste **(striktně) rychleji** než g (f je ohraničena g **zdola ostře**) $[>]$.
- $f \in \Omega(g)$ – f roste **rychleji** než g (f je ohraničena g **zdola**) $[\geq]$.



Seřazeno podle složitosti:

- $f(n) \in \Omega(\log n)$ – logaritmická funkce (složitost),
- $f(n) \in \Omega(n)$ – lineární funkce (složitost),
- $f(n) \in \Omega(n^2)$ – kvadratická funkce (složitost),
- $f(n) \in O(n^k)$ pro nějaké $k > 0$ – polynomiální,
- $f(n) \in \Omega(k^n)$ pro nějaké $k > 1$ – exponenciální.

3.4.1 Úskalí asymptotické notace

Při používání asymptotických odhadů časové složitosti je třeba si uvědomit některá úskalí:

- Asymptotické odhady se týkají pouze toho, **jak roste čas s rostoucí velikostí vstupu** → neříkají nic o **konkrétní době výpočtu**. V asymptotické notaci mohou být **skryty velké konstanty**.
- Algoritmus, který má lepší asymptotickou časovou složitost než nějaký jiný algoritmus, **může být ve skutečnosti rychlejší** až pro nějaké hodně velké vstupy.
- Většinou analyzujeme složitost v **nejhorším případě**. Pro některé algoritmy může být doba výpočtu v nejhorším případě mnohem větší než doba výpočtu na „typických“ instancích (typicky Quicksort → nejhorší: $O(n^2)$, průměrná: $O(n \log n)$).

3.5 Algoritmicky nerozhodnutelné problémy

Rozhodovací problém je rozhodnutelný (řešitelný) pokud pro libovolný vstup z množiny vstupů, skončí algoritmus svůj výpočet a vydá správný výstup (tedy jestliže **existuje Turingův stroj, který jej řeší**).

Pokud nalezneme takový vstup, pro který všechny dosavadní algoritmy nejsou schopny nalézt výstup, můžeme tento problém označit za **nerozhodnutelný**. Speciální případ jsou **doplňkové problémy**, které vracejí přesně opačné výsledky než původní problém.

3.5.1 Definice problému

Problém je určen **trojicí** (IN, OUT, p) , kde:

- IN je množina (přípustných) **vstupů**,
- OUT je množina **výstupů**,
- $p : IN \rightarrow OUT$ je **funkce** přiřazující každému vstupu odpovídající výstup.

3.5.2 Ano/Ne problémy

Jsou to problémy, jejichž **výstupní množina obsahuje dva prvky** $OUT = \{\text{ano}, \text{ne}\}$. Na ano/ne problémy se dají převést ostatní problémy nepotřebujeme-li znát přesný výsledek:

- Nepotřebuji najít v poli nejmenší číslo, stačí mi vědět **zda pole obsahuje číslo menší než nula**.
- Nepotřebuji znát nejkratší cestu grafem, stačí mi najít cestu, **která je kratší než 8**.

3.5.3 Riceova věta

Tato věta ukazuje nerozhodnutelnost celé třídy problémů, její znění je následující „*Každá netriviální vstupně/výstupní (I/O) vlastnost programů je nerozhodnutelná*“.

- Vlastnost X je **vstupně/výstupní** právě tehdy, když každé dva programy se stejnou I/O tabulkou buď oba vlastnost X mají nebo ji oba nemají.
- Připomeňme tedy ještě, že vlastnost V je **triviální**, když ji mají buď všechny programy nebo ji nemá žádný program; taková vlastnost je podle definice také **vstupně/výstupní**.

Problém	1	2	3
Je triviální?	A	N	N
Je I/O?	A	A	N
Je nerozhodnutelný	N	A	N

3.5.4 Částečná rozhodnutelnost

Částečně rozhodnutelný problém, je takový problém, pro který jsme v případě vstupů, u nichž očekáváme odpověď ANO, **schopni vrátit odpověď ANO**, a v případě NE vrátit buď NE nebo \perp (program se nezastaví a nejsme schopni zjistit, zda by odpověď byla opravdu NE).

3.5.5 Převeditelnost mezi nerozhodnutelnými problémy

Důkaz neřešitelnosti lze provést skrze jiné, **už dokázané**, problémy. Řekneme, že problém P_1 je převeditelný na problém P_2 (značíme $P_1 \rightsquigarrow P_2$), jestliže alg., který k instanci I_1 problému P_1 sestrojí instanci I_2 problému P_2 tak, **že odpověď P_1, I_1 je stejná jako P_2, I_2** . Např.: DHP je převeditelný na HP. Z toho vyplývá, že pokud P_1 je nerozhodnutelný tak i P_2 je **nerozhodnutelný**.

3.5.6 Optimalizační problémy

Optimalizační problémy **hledají nejlepší řešení** v množině různých řešení. Příkladem je například: hledání nejkratší cesty, nejmenší kostry, apod.

3.5.7 Příklady nerozhodnutelných problémů

1. Hledání nejkratší cesty v grafu

- **VSTUP**: Orientovaný graf $G = (V, E)$ a dvojice vrcholů $u, v \in V$.
- **VÝSTUP**: Nejkratší cesta z u do v .

2. Hledání minimální kostry v grafu

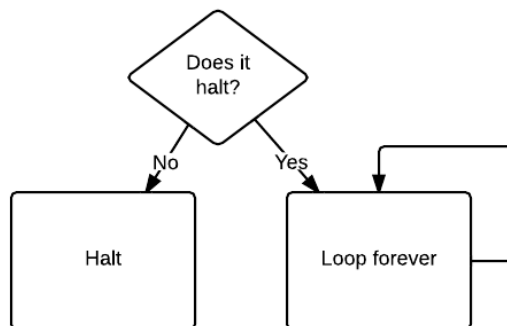
- **VSTUP**: Neorientovaný souvislý graf $G = (V_G, E_G)$ s ohodnocenými hranami.
- **VÝSTUP**: Souvislý graf $H = (V_H, E_H)$, kde $V_H = V_G$ a $E_H \subseteq E_G$, který má součet hodnot všech hran minimální.

3. Eq-CFG (Ekvivalence bezkontextových gramatik)

- **VSTUP:** Dvě bezkontextové gramatiky $G1, G2$.
- **OTÁZKA:** Platí $L(G1) = L(G2)$? Generují obě gramatiky stejný jazyk?

4. HP (Problém zastavení [Halting Problem])

- **VSTUP:** Turingův stroj M a jeho vstup w .
- **OTÁZKA:** Zastaví se M na w (tzn. je výpočet stroje M pro vstupní slovo w konečný)?



4 Třídy složitosti problémů. Třída PTIME a NPTIME, NP-úplné problémy.

4.1 PTIME (PSPACE)

Třída všech problémů, které lze řešit algoritmy s **polynomiální časovou (prostorovou) složitostí**, tj. s časovou složitostí $O(n^k)$, kde k je nějaká konstanta.

Tuto třídu problému považujeme za zvládnutelnou. Je **robustní** – mezi jednotlivými výpočetními modely (RAM, TS) existují vzájemné polynomiální simulace, nezáleží tedy jakým modelem budeme algoritmus simulovat, vždy bude patřit do třídy PTIME.

4.1.1 Problémy pařící do třídy PTIME

- Třídění a vyhledávání.
- Nejkratší cesta v grafu a minimální kostra grafu.
- Ekvivalence deterministických konečných automatů.
- Přijatelnost slova bezkontextovou gramatikou.

1. Výběr aktivit

- **VSTUP:** Množina aktivit s časovými intervaly, kdy je lze vykonávat.
- **VÝSTUP:** Největší možný počet kompatibilních aktivit (aktivit, které se nekryjí).

2. Optimalizace násobení řetězce matic

- **VSTUP:** Posloupnost matic.
- **VÝSTUP:** Plně uzávorkovaný součin.

3. LCS - problém nejdelší společné posloupnosti

- **VSTUP:** Dvě posloupnosti v, w v nějaké abecedě Σ .
- **VÝSTUP:** Nejdelší společná podposloupnost posloupností v, w .

4.2 NPTIME

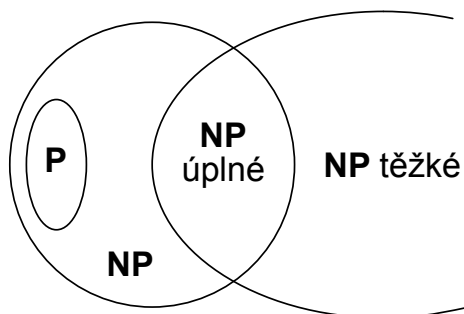
Třída všech rozhodovacích problémů (ano/ne), které jsou rozhodovány **nedeterministickými algoritmy s polynomiální časovou složitostí**. Tedy problémy, u kterých je možné pro daný vstup vždy ověřit zda je odpověď ANO, v případě NE tomu tak být nemusí.

Při odpovědi „ANO” je výsledek vždy správný (našel se alespoň jeden výpočet pro který je odpověď ANO) zatímco odpověď „NE” nemusí být vždy pravdivá a to z důvodu, že by na výstupu muselo být vždy NE (algoritmus by musel otestovat všechny možnosti). Pointou nedeterministických algoritmů je to že **náhodně nastřelují nějaké řešení a ověřují jejich správnost**.

4.3 Třída NP-úplných problémů

NP-úplné problémy jsou takové problémy, na které jsou **polynomiálně redukovatelné všechny ostatní problémy z třídy NP**. To znamená, že třídu NP-úplných úloh tvoří v jistém smyslu ty **nejtěžší úlohy** z NP.

Pokud by byl nalezen deterministický polynomiální algoritmus pro nějakou NP-úplnou úlohu, znamenalo by to, že všechny nedeterministicky polynomiální problémy jsou řešitelné v polynomiálním čase (tedy že $NP = P$). Otázka, zda nějaký takový algoritmus existuje, zatím nebyla rozhodnuta, předpokládá se však, že $NP \neq P$ (je však zřejmé, že $P \subseteq NP$).



4.3.1 NP-těžký problém

Problém P nazveme NP-těžkým, pokud pro libovolný problém A ze třídy NP platí, že A je **polynominálně převeditelné** (redukovatelné) na problém P , tedy pokud platí: $\forall P \in \text{NPTIME} : P \triangleright Q$.

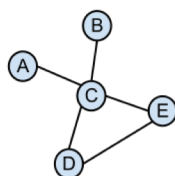
4.3.2 NP-úplný problém

Třída nejtěžších NPTIME problémů. Problém P je NP-úplný, pokud patří do třídy **NP** a je **NP-těžký**.

4.3.3 NP-úplné problémy

1. IS (problém nezávislé množiny)

- **VSTUP:** Neorientovaný graf G (o n vrcholech), číslo k ($k \leq n$).
- **OTÁZKA:** Existuje v G nezávislá množina velikosti k (tj. množina k vrcholů, z nichž žádné dva nejsou spojeny hranou)?



Program vybere náhodně k vrcholů z grafu a ověří zdali nejsou některé spojeny hranou. Když mezi nimi hranu **nenajde**, vrátí odpověď „ANO, v grafu existuje

nezávislá množina o velikosti k ". Když hranu mezi zvolenou množinou **najde**, vrátí odpověď „NE”. Přičemž odpověď ne **může být chybná**.

Třeba pro graf ABCDE na obrázku níže a $k = 3$, vybere algoritmus vrcholy $\{A, B, C\}$, které závislé jsou a vrátí chybnou odpověď „NE”. Když to ale algoritmus provede vícekrát, pro různé vrcholy, **pravděpodobnost správnosti odpovědi se zvyšuje**. A o tom to je. Nepotřebujeme znát 100% správnou odpověď, ale chceme se dočkat alespoň nějaké odpovědi.

Výstup pro $k > 3$: NE.

Výstup pro $k = 3$: ANO (ABD, ABE).

Výstup pro $k = 2$: ANO (AB, AD, AE, BD, BE).

2. Isomorfismus grafů

- **VSTUP**: Dva neorientované grafy G a H .
- **OTÁZKA**: Jsou grafy G a H izomorfní?

3. CG (Barvení grafu)

- **VSTUP**: Neorientovaný graf G a číslo k .
- **OTÁZKA**: Je možné graf G obarvit k barvami (tj. existuje přiřazení barev vrcholům tak, aby žádné dva sousední vrcholy nebyly obarveny stejnou barvou)?

4. SAT (problém splnitelnosti booleovských formulí)

- **VSTUP**: Booleovská formule v konjunktivní normální formě.
- **OTÁZKA**: Je daná formule splnitelná (tj. existuje pravdivostní ohodnocení proměnných, při kterém je formule pravdivá)?

5. 3-SAT (problém SAT s omezením na 3 literály)

- **VSTUP**: Formule v konjunktivní normální formě, kde každá klauzule obsahuje právě 3 literály.
- **OTÁZKA**: Je formule splnitelná?

6. HK (problém hamiltonovské kružnice)/HC (problém hamiltonovského cyklu)

- **VSTUP**: Neorientovaný graf G /Orientovaný graf G .
- **OTÁZKA**: Existuje v G hamiltonovská kružnice (uzavřená cesta, procházející každým vrcholem právě jednou)?

7. Subset-Sum

- **VSTUP**: Množina přirozených čísel $M = x_1, x_2, \dots, x_n$ a přirozené číslo s .

- **OTÁZKA:** Existuje podmnožina množiny M , pro niž součet jejích prvků je roven s ?

8. Problém obchodního cestujícího (TSP) ANO/NE verze

- **VSTUP:** Neorientovaný graf G s hranami ohodnocenými přirozenými čísly a číslo k .
- **OTÁZKA:** lze objet k měst a nejet víc než danou vzdálenost?

9. Vrcholové pokrytí (vertex cover)

- **VSTUP:** Neorientovaný graf G a přirozené číslo k .
- **OTÁZKA:** Existuje v grafu G množina vrcholů velikosti k taková, že každá hrana má alespoň jeden svůj vrchol v této množině?

5 Jazyk predikátové logiky prvního řádu. Práce s kvantifikátory a ekvivalentní transformace formulí.

Predikátová logika (PL) pracuje s primitivními formullemi (**predikáty**) vypovídajícími o **vlastnostech** a **vztazích** mezi **předměty** jistého **univerza** (individui). Je rozšířením výrokové logiky. Na rozdíl od výrokové logiky si všímá i struktury vět samotných a obsahuje predikáty a kvantifikátory. Pouze jen malá část úsudku může být formalizována pomocí výrokové logiky:

Všechny opice mají rády banány

Judy je opice

Judy má ráda banány

Z hlediska VL jsou to jednoduché výroky p, q, r a z p, q nevyplývá r .

5.1 Predikátová logika 1. řádu

Predikátová logika umožňuje uvažování nad **vlastnostmi**, jež jsou **sdíleny mnoha objekty**, díky použití **proměnných** a **kvantifikátorů**. V predikátové logice by byl výše uvedený úsudek formalizován takto:

Každé individuum, je-li **Opice** pak má rádo **Banány**.

Judy je individuum s vlastností být **Opice**.

Judy je individuum s vlastností mít rádo **Banány**.

$\forall x(O(x) \rightarrow B(x)); O(J) \neq B(J)$, kde x je **individuová proměnná**; O, B **predikátové symboly** a J **funkční symbol**.

Poznámka Pokud bychom chtěli formalizovat úsudky, které navíc vypovídají i o vlastnostech vlastností a vztahů a o vztazích mezi vlastnostmi a vztahy, museli bychom použít predikátovou logiku druhého řádu a vyššího. Tou se ale nebudeme zabývat.

5.1.1 Formální jazyk PL1 – Abeceda

Logické symboly:

- Individuové proměnné: x, y, z, \dots
- Logické spojky: \wedge konjunkce, \vee disjunkce, \rightarrow implikace, \leftrightarrow ekvivalence, \neg negace.
- Kvantifikační symboly: \forall, \exists .

Speciální symboly: (n-arita = počet argumentů)

- Predikátové: P^n, Q^n, \dots
- Funkční: f^n, g^n, h^n, \dots

Pomocné symboly: závorky a jiná interpunkční znaménka $(,), \dots$

5.1.2 Formální jazyk PL1 – Gramatika

Termy:

1. každý symbol proměnné x, y, \dots je **term**,
2. jsou-li t_1, \dots, t_n ($n \geq 0$) termy a je-li f n -ární **funkční symbol**, pak výraz $f(t_1, \dots, t_n)$ je term; pro $n = 0$ se jedná o **individuovou konstantu** (značíme a, b, c, \dots),
3. jen výrazy dle 1. a 2. jsou termy.

Atomické formule:

- je-li P n -ární predikátový symbol a jsou-li t_1, \dots, t_n termy, pak výraz $P(t_1, \dots, t_n)$ je **atomická formule** (na vstupu jsou pouze termy).

Formule:

- každá atomická formule je formule,
- je-li výraz A formule, pak $\neg A$ je formule,
- jsou-li výrazy A a B formule, pak výrazy $(A \wedge B), (A \vee B), (A \rightarrow B), (A \leftrightarrow B)$ jsou formule, je-li x proměnná a A formule, pak výrazy $\forall x A$ a $\exists x A$ jsou formule.

5.2 Převod z přirozeného jazyka do PL1

- \forall – „všichni“, „žádný“, „nikdo“, \dots
- \exists – „někdo“, „něco“, „někteří“, „existuje“, \dots

Větu musíme často ekvivalentně přeformulovat, pozor: v češtině **dvojitý zápor!**

- Žádný student není důchodce: $\forall x(S(x) \rightarrow \neg D(x))$.
- Ale, „všichni studenti nejsou důchodci“ čteme jako „ne všichni studenti jsou důchodci“:
 $\neg \forall x(S(x) \rightarrow D(x)) \leftrightarrow x(S(x) \rightarrow \neg D(x))$

Jako **pomůcka** k řešení může sloužit tato zásada:

- Po **všeobecném** kvantifikátoru následuje formule ve tvaru implikace: $\forall \dots \rightarrow$.
- Po **existenčním** kvantifikátoru formule ve tvaru konjunkce: $\exists \dots \wedge$.

5.2.1 Volné a vázané proměnné

$$\begin{array}{ccc} \forall x \exists y P(x, y, t) \wedge \neg \exists x Q(y, x) \\ \swarrow \quad \downarrow & & \downarrow \quad \searrow \\ \text{vázané, volné} & & \text{volné, vázané} \end{array}$$

5.3 Ekvivalentní úpravy

Při ekvivalentních úpravách se používají **de Morganovy** zákony v PL1: $\neg \forall x A \leftrightarrow \exists x \neg A$
 $\neg \exists x A \leftrightarrow \forall x \neg A$.

Příklady

- Není pravda, že všichni vodníci jsou zelení. \leftrightarrow Někteří vodníci nejsou zelení.
 $\neg \forall x (V(x) \rightarrow Z(x)) \leftrightarrow \exists x (V(x) \wedge \neg Z(x))$
- Není pravda, že někteří vodníci jsou zelení. \leftrightarrow Žádný vodník není zelený.
 $\neg \exists x (V(x) \wedge Z(x)) \leftrightarrow \forall x (V(x) \rightarrow \neg Z(x))$
- Everybody loves somebody sometimes.
 $\forall x \forall y \forall z L(x, y, z)$
- Marie má ráda pouze vítěze.
 $\forall x (R(m, x) \rightarrow V(x))$

5.4 Ekvivalentní transformace

- Aplikace negace: $\neg \forall x [V(x) \rightarrow Z(x)] \leftrightarrow \exists x [V(x) \wedge \neg Z(x)]$.
- **De morganovy zákony:** $\forall x [((P(x) \wedge Q(x)) \vee D(x))] \leftrightarrow \forall x [(P(x) \vee D(x)) \wedge (Q(x) \vee D(x))]$.
$$\neg(A \vee B) \iff (\neg A) \wedge (\neg B)$$
$$\neg(A \wedge B) \iff (\neg A) \vee (\neg B)$$
- Převod **implikace:** $\forall x (P(x) \rightarrow G(x)) \leftrightarrow \forall x (\neg P(x) \vee G(x))$.

Patří zde i část z převodu do **Skolemovy Klauzární formy**:

1. eliminace nadbytečných kvantifikátorů,
2. eliminace spojek $\rightarrow, \leftrightarrow$,
3. přesun negace dovnitř,
4. přejmenování proměnných,
5. přesun kvantifikátorů doprava,
6. přesun všeobecných kvantifikátorů doleva,
7. použití distributivních zákonů.

5.5 Sémantika v PL1

- Při substituci termů za proměnné ($A(x/t)$), je třeba dbát na nahrazování pouze **volných proměnných**.
- Při definici formule, je nutné vysvětlit co **znamenaají** jednotlivé predikátové symboly, termy atd.

6 Pojem relace, operace s relacemi, vlastnosti relací. Typy binárních relací. Relace ekvivalence a relace uspořádání.

6.1 Relace

- **N-ární relace** nad množinami A_1, \dots, A_n je libovolná podmnožina kartézského součinu $A_1 \times \dots \times A_n$ (tyto množiny jsou **nosiči** relace).
- **Kartézský součin** množin A a B , označovaný $A \times B$, je množina všech uspořádaných dvojic, kde první prvek z dvojice patří do množiny A a druhý do množiny B . Příklad: $\{a, b\} \times \{a, b, c\} = \{(a, a), (a, b), (a, c), (b, a), (b, b), (b, c)\}$.

6.1.1 Typy relací

- **homogenní** – jediný druh nosiče ($A \times A$),
- **heterogenní** – alespoň dva různé druhy nosiče ($A \times B$),
- **unární** ($n = 1$), **binární** ($n = 2$), **ternární** ($n = 3$), **n-ární** – podle arity,
- **triviální** – úplná ($\rho = A_1 \times A_n$), **prázdná** ($\rho = \emptyset$),
- **netriviální** – $\emptyset \subset \rho \subset A_1 \times \dots \times A_n$.

6.1.2 Vlastnosti relací

Binární relace $R \subseteq A \times A$ je:

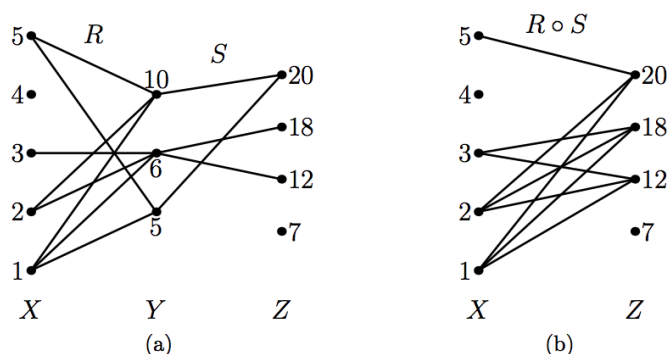
- **Reflexivní** – $\forall x \in A : (x, x) \in R$.
- **Ireflexivní** – $\forall x \in A : (x, x) \notin R$.
- **Symetrická** – $\forall x \in A : (x, y) \in R \Rightarrow (y, x) \in R$.
- **Asymetrická** – $\forall x \in A : (x, y) \in R \Rightarrow (y, x) \notin R$.
- **Antisymetrická** – $\forall x \in A : (x, y) \in R \wedge (y, x) \in R \Rightarrow x = y$.
- **Tranzitivní** – $\forall x \in A : (x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$.

Příklad

- Relace „=” na \mathbb{N} je reflexivní, symetrická, antisymetrická a tranzitivní, ale není ireflexivní ani asymetrická.
- Relace „ \leq ” na \mathbb{N} je reflexivní, antisymetrická a tranzitivní, ale není ireflexivní, symetrická ani asymetrická.
- Relace „ $<$ ” na \mathbb{N} je ireflexivní, asymetrická, antisymetrická a tranzitivní, ale není reflexivní ani symetrická.

6.2 Operace s relacemi

- **Průnik** – Prvek x náleží do průniku relací $R1 \cap R2$, pokud patří do množiny $R1(x \in R1)$ a zároveň do $R2(x \in R1)$.
- **Sjednocení** – Prvek x náleží do sjednocení relací $R1 \cup R2$, pokud patří do množiny $R1(x \in R1)$ nebo $R2(x \in R1)$.
- **Doplňěk** – Doplněk $R1'$ k relaci $R1$ rozumíme všechny prvky které nepatří do $R1$.
- **Inverze** – Relace $R^{-1} \subseteq B \times A$ je inverzní k relaci $R \subseteq A \times B$, pokud $xR^{-1}y \Leftrightarrow yRx$.
- **Skládání relací** – Výsledkem je množina dvojic, kde pokud existují dvojice $(a, b) \in R$ a $(b, c) \in S$, pak jejich složení $(a, c) \in R \circ S$.



Obrázek 1.2: (a) Relace R a S , (b) jejich složení.

6.3 Typy binárních relací

Mezi nejznámější typy binárních relací patří **ekvivalence** $(=)$ [Re, Sy, Tr], **uspořádání** $(<, >, \leq, \geq)$ [Re, An, Tr] a **tolerance** [Re, Sy].

6.3.1 Ekvivalence [Re, Sy, Tr]

Relace ekvivalence představuje jakési zjemnění relace rovnosti. Vždy můžeme rozhodnout, že jsou dva prvky množiny stejné, tj. že $a = a$. Ale někdy se nám hodí zjistit, zda jsou si dva prvky **pouze podobné**, ne nutně stejné. Neboli zda mají stejnou nějakou zásadní vlastnost. Například dvě knihy můžeme považovat za podobné, pokud mají stejný žánr nebo **pomocí ekvivalence**: dvě knihy jsou ekvivalentní pokud mají stejný žánr.

- Binární relace na množině X je ekvivalentní, pokud je R na A : **reflexivní**, **symetrická** a **tranzitivní**.
- **Třída ekvivalence** prvku a je množina všech prvků ekvivalentních s daným prvkem a .
- **Průnik** dvou ekvivalencí je zase ekvivalence.
- **Sjednocení** dvou ekvivalencí nemusí znamenat, že výsledek bude ekvivalentní.

6.3.2 Uspořádání [Re, An, Tr]

- Binární relace na množině X je **neostrým uspořádáním**, pokud je R na A : **reflexivní**, **antisymetrická** a **tranzitivní**.
- Binární relace na množině X je **ostrým uspořádáním**, pokud je R na A : **ireflexivní**, **antisymetrická** a **tranzitivní**.
- Uspořádání je **úplné** pokud neexistují neporovnatelné prvky.

7 Pojem operace a obecný pojem algebra. Algebry s jednou a dvěma binárními operacemi.

7.1 Algebra

Algebra je naukou o **algebraických strukturách**, tedy **množinách**, na nichž jsou zavedeny nějaké **operace**. Slouží pro popis objektů reálného světa a operací prováděných s těmito objekty. Příklady algeber:

- $(\mathbb{N}, \{+^2\})$ - sčítání nad množinou přirozených čísel,
- $(2^M, \{\cup, \cap\})$ - množina všech podmnožin M s operací průnik a sjednocení.

7.1.1 Definice

Každý objekt algebry je reprezentován **datovým nosičem** (množina popisující data, se kterými pracujeme). A **operacemi** – nejjednoduššími transformacemi, které nad daty můžeme realizovat. **Algebraická struktura** je definována jako (A, \circ) , kde:

- A – nosič algebry (množina objektů – čísel, proměnných, ...),
- \circ – množina operací nad nosičem X .

7.2 Operace

Operace na množině A je definována jako zobrazení

$$f : A^n \rightarrow A, \quad (1)$$

tedy zobrazení, které každé n -tici prvků množiny A , jednoznačně přiřazuje prvek z množiny A . Číslo n nazýváme **arita operace** a podle něj operace označujeme jako **nulární** ($n = 0$), **unární** ($n = 1$), **binární** ($n = 2$), **ternární** ($n = 3$).

7.3 Algebraické struktury s jednou binární operací

Definována jako (A, \circ) s jedním nosičem (A) a jednou homogenní binární operací (\circ). Nejprve je nutné zmínit **vlastnosti binárních operací**:

- **asociativita**: $a * (b * c) = (a * b) * c$,
- **komutativita**: $a * b = b * a$.

Kromě již zmíněné asociativity a komutativity algebraické struktury také zavádí existenci:

- **Jednotkového prvku**: e takové, že $\forall x \in X : x \circ e = e \circ x = x$. Tedy prvek, který nezmění výsledek (1 u násobení, 0 u sčítání).
- **Inverzního prvku**: \bar{x} takové, že $\forall x \in X : x \circ \bar{x} = \bar{x} \circ x = e$. Tedy prvek, který převede výsledek na jednotkový prvek.

neutrálního či **inverzního prvku**, a další charakteristiky.

7.3.1 Klasifikace algebraických struktur

Všechny níže uvedené klasifikace algebraické struktury (A, \circ) zahrnují i ty co jsou pod nimi. Tedy pokud je nějaká algebraická struktura (AS) Monoid, je i Pologrupa a Grupoid.

1. **Grupoid** – **uzavřenost** (univerzalita) na nosiči (po výpočtu je výsledek stále v množině A).
2. **Pologrupa** – splňuje vlastnost **asociativity**.
3. **Monoid** – existence **jednotkového prvku**.
4. **Grupa** – existence **inverzního prvku**.
5. **Abelova grupa** – splňuje vlastnost **komutativity** (symetrická podle diagonály).

Kongruence – označuje ekvivalenci na algebře, která je slučitelná se všemi operacemi na algebře.

7.3.2 Morfismy

- **Homomorfismus** – zobrazení, které převádí jednu algebraickou strukturu na jinou:
 $f(a_1 \cdot a_2) \rightarrow f(a_1) \circ f(a_2)$.
- **Izomorfismus** – bijektivní homomorfismus.
- **Epimorfismus** – surjektivní homomorfismus.
- **Monomorfismus** – injektivní homomorfismus.
- **Endomorfismus** – homomorfismus z objektu do sebe sama (stejná množina).
- **Automorfismus** – endomorfismus, který je izomorfní.

7.4 Okruhy (Algebraické struktury s dvěma binárními operacemi)

Okruh je algebraický systém $(A, +, \cdot)$ se dvěma základními binárními operacemi, kde první $(A, +)$ je **abelova grupa** a druhá (A, \cdot) je alespoň **pologrupa**. Podobně jako u předchozí AS, i zde se zavádí nový pojem:

- **Existence dělitele nuly** – říká, že ve struktuře existují 2 nenulové prvky, pro něž platí $a \circ b = 0$.

U všech typů okruhů musí být splněna podmínka první struktury, která musí být **abelova grupa**, a druhá musí být:

- **Okruh** - uzavřená (U), asociativní (A) [pologrupa].
- **Unitární okruh** - U, A, existence jednotkového prvku (J) [monoid].
- **Obor Integrity** - U, A, J [monoid] + **nesmí** obsahovat dělitele nuly.
- **Těleso** - U, A, J a existence inverzního prvku (I) [grupa] + **nesmí** obsahovat dělitele nuly.
- **Pole** - U, A, J, I a komutativita [grupa] + **nesmí** obsahovat dělitele nuly.

8 FCA – formální kontext, formální koncept, konceptuální svazy. Asociační pravidla, hledání často se opakujících množin položek.

8.1 Formální konceptuální analýza (FCA)

Metoda analýzy tabulkových dat (objektů a jejich vlastností), umožňuje jiný pohled na data (využívá se např. u data miningu). **Vstupem** pro FCA jsou **tabulková data**, která jsou uspořádána následovně: **objekty** (řádky) a **atributy** (sloupce). Tyto tabulková data vytváří tzv. **kontexty**.

	červené	bílé
jablko	×	
zelí	×	×

8.2 Formální Kontext

Formální kontext K obsahuje objekty z množiny O a atributy z množiny A . Vztahy mezi objekty a atributy jsou charakterizovány binární relací R . Obecně se pro popis kontextu používá výraz:

$$K = (O, A, I). \quad (2)$$

Takto vymezený formální kontext je dobře zobrazitelný tabulkou, ve které jsou **řádky** obsazeny **objekty**, **sloupce atributy** a incidenční data ($I \subseteq O \times A$) vyjadřují relaci R (I je relace incidence).

8.2.1 Galoisovy konexe

Umožňují přecházet z množiny objektů na jejich společné atributy (\uparrow **intent**) a naopak (\downarrow **extent**).

- **Intent** $\uparrow - 2^X \rightarrow 2^Y; A \subseteq X; A^\uparrow = \{y \in Y; \forall x \in A(x, y) \in I\}$ [*z objektu na atributy*].
- **Extent** $\downarrow - 2^Y \rightarrow 2^X; B \subseteq Y; B^\downarrow = \{x \in X; \forall y \in B(x, y) \in I\}$ [*z atributů na objekt*].

Příklad

$$\begin{aligned} A_1 &= \{\text{jablka, zelí}\}, A_1^\uparrow = \{\text{červené}\} & A_2 &= \{\text{zelí}\}, A_2^\uparrow = \{\text{červené, bílé}\} \\ B_1 &= \{\text{červené, bílé}\}, B_1^\downarrow = \{\text{zelí}\} & B_2 &= \{\text{červené}\}, B_2^\downarrow = \{\text{zelí, jablko}\} \end{aligned}$$

8.3 Formální koncept

Formální koncept je dvojice (A, B) , kde A je množina objektů, a B je množina atributů, které jsou **společné pro všechny objekty** z množiny A . Koncept (A, B) je tedy dán jako: $(A, B) \Leftrightarrow A = B^\downarrow \wedge B = A^\uparrow$, kde $A \subseteq X$, $B \subseteq Y$, kde X a Y jsou objekty a atributy výše uvedené tabulky.

8.3.1 Uzávěrový operátor $\uparrow\downarrow$

Jak již značí $A^{\uparrow\downarrow} = C(A)$ vypovídá, k určení uzávěru atributů množiny A se nejprve provede **intent** a poté **extent**. Uzávěr má tyto vlastnosti:

1. **Idempotence** – $C(C(A)) = C(A)$.
2. **Extensionalita** – $A \subseteq C(A)$.
3. **Monotonie** – $A_1 \subseteq A_2 \Rightarrow C(A_1) \subseteq C(A_2)$.

8.4 Konceptuální svazy

Uspořádaná množina konceptů tvoří tzv. konceptuální svaz. Ten lze graficky znázornit **Hasseovým diagramem** – každý vrchol grafu reprezentuje **jeden koncept**. Koncepty K_i a K_j jsou v grafu spojeny, pokud $K_i \leq K_j$, přičemž K_j je umístěn výše než K_i a neexistuje takové K_k , že $K_i \leq K_k \leq K_j$.

8.4.1 Návod k vytvoření konceptuálního svazu

1. Vytvořím a postupně si zapíšu množinu všech extentů na jednotlivých attributech (v grafu zapisuji **zdola nahoru**).
2. Vytvořím jedinečné průniky extentů.
3. Nesmím zapomenout na zahrnutí průniku s prázdnou množinou $= \emptyset$.
4. Přidám extent zahrnující všechny objekty.
5. Pro odpovídající extenty vytvořím intenty (v grafu zapisuji **shora dolů**).

Příklad

	2 nohy	4 nohy	mléko	vlna
pes		×		
ovce		×	×	×
koza		×	×	
slepice	×			

$$e_7 = \{\text{pes, ovce, koza, slepice}\}$$

$$e_2 = \{4 \text{ nohy}\}^\downarrow = \{\text{pes, ovce, koza}\}$$

$$e_3 = \{\text{mléko}\}^\downarrow = \{\text{ovce, koza}\}$$

$$e_1 = \{2 \text{ nohy}\}^\downarrow = \{\text{slepice}\}$$

$$e_4 = \{\text{vlna}\}^\downarrow = \{\text{ovce}\}$$

$$e_5 = e_2 \cap e_3 = \{\text{pes}\}$$

$$e_6 = \emptyset$$

$$i_7 = e_7^\uparrow = \emptyset$$

$$i_2 = e_2^\uparrow = \{4 \text{ nohy}\}$$

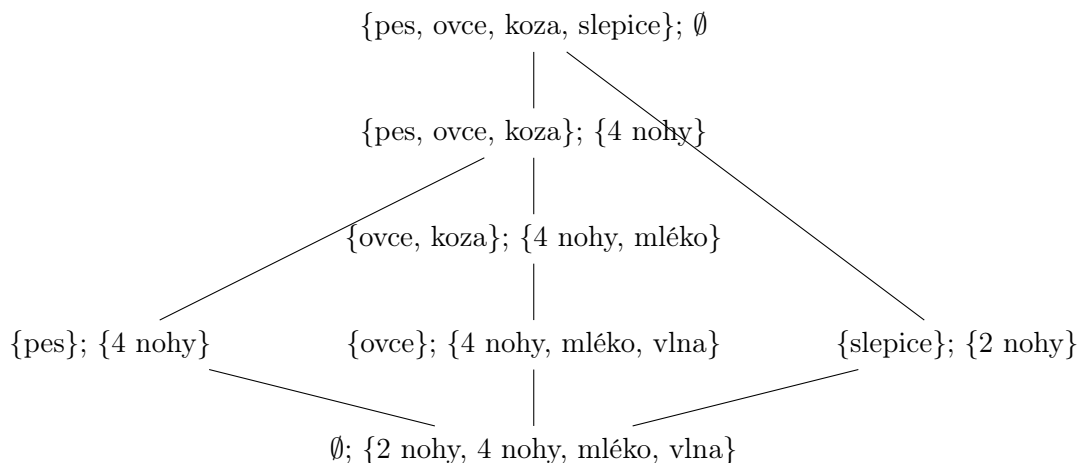
$$i_3 = e_3^\uparrow = \{4 \text{ nohy, mléko}\}$$

$$i_1 = e_1^\uparrow = \{2 \text{ nohy}\}$$

$$i_4 = e_4^\uparrow = \{4 \text{ nohy, mléko, vlna}\}$$

$$i_5 = e_5^\uparrow = \{4 \text{ nohy}\}$$

$$i_6 = e_6^\uparrow = \{2 \text{ nohy, 4 nohy, mléko, vlna}\}$$



8.5 Svazy (pro doplnění, asi není nutné úplně znát k FCA)

Svaz je algebra (L, \cap, \cup) s dvěma základními binárními operacemi $x \cap y$ **spojení (supré-mum)** ($\sup(x, y)$) a $x \cup y$ **průsek (infimum)** ($\inf(x, y)$), které mají následující vlastnosti:

1. **Univerzalita (jednoznačnost)** – $\forall x, y \exists z \ x \cap y = z \mid \forall x, y \exists z \ x \cup y = z$.
2. **Asociativita** – $x \cap (y \cap z) = (x \cap y) \cap z \mid x \cup (y \cup z) = (x \cup y) \cup z$.
3. **Komutativita** – $x \cap y = y \cap x \mid x \cup y = y \cup x$.
4. **Absorbce** – $x \cap (x \cup y) = x \mid x \cup (x \cap y) = x$.

8.5.1 Typy svazů

1. **Distributivní** – platí zde axiomy distributivity a neobsahuje ani **diamant** ani **pentagon**: $x \cup (x \cap z) = (x \cup y) \cap (x \cup z) \mid x \cap (x \cup z) = (x \cap y) \cup (x \cap z)$.



2. **Modulární** – slabší reprezentace distributivity, **nesmí** obsahovat **pentagon**, $a \geq c$:
 $a \cap (b \cup c) = (a \cap b) \cup c$.
3. **Komplementární** – platí zde, že pro každý prvek x existuje komplement x' , kdy:
 $x \cap x' = [\text{svazová } 0]$ a $x \cup x' = [\text{svazová } 1]$
4. **Boleaovský svaz** – komplementární \wedge distributivní

8.5.2 Vlastnosti svazů

Pro každé dva prvky v množině existuje **sup** a **inf**. **Úplný svaz** – nastane tehdy, zda pro **libovolné neprázdné podmnožiny** existuje **sup** a **inf**. U svazů můžeme dále získat tyto vlastnosti:

- **Minimum a maximum** – žádný není menší/větší než a (vrchol diagramu).
- **Nejmenší a největší** – pouze jeden nejmenší/největší prvek, pokud je jich více, neexistuje největší/nejmenší prvek.
- **Dolní ($L(A)$) a horní ($U(A)$) závora** – všechny prvky jsou \leq / \geq než A ,
- **Infimum** – nejmenší prvek dolní závory.
- **Supremum** – nejmenší prvek horní závory.

8.6 Asociační pravidla

Termín asociační pravidla široce zpopularizoval počátkem 90. let v souvislosti s analýzou nákupního košíku. Při této analýze se zjišťuje, jaké druhy zboží si současně kupují zákazníci v supermarketech (např. pivo a párek). **Jde tedy o hledání vzájemných vazeb (asociací) mezi různými položkami** sortimentu prodejny. Přitom není upřednostňován žádný speciální druh zboží jako závěr pravidla.

8.6.1 Základní charakteristika pravidel

U pravidel vytvořených z dat nás obvykle zajímá kolik příkladů splňuje **předpoklad** a kolik **závěr** pravidla, kolik příkladů splňuje předpoklad i závěr **současně**, kolik příkladů splňuje předpoklad a **nesplňuje** závěr. . . . Tedy, zajímá nás, jak pro pravidlo:

$$Ant \Rightarrow Suc, \quad \text{kde } Ant, Suc \subseteq I \text{ (položky)} \quad (3)$$

kde **Ant** (**předpoklad**, levá strana pravidla, **antecedent**) a **Suc** (**závěr**, pravá strana pravidla, **sukcedent**) jsou kombinace kategorií, pro něž příslušná **kontingenční tabulka** vypadá následovně:

	Suc	$\neg Suc$
Ant	a	b
$\neg Ant$	c	d

- $Ant \wedge Suc$ – **a** je počet objektů pokrytých současně předpokladem i závěrem,
- $Ant \wedge \neg Suc$ – **b** je počet objektů pokrytých předpokladem a nepokrytých závěrem,
- $\neg Ant \wedge Suc$ – **c** je počet příkladů nepokrytých předpokladem ale pokrytých závěrem,
- $\neg Ant \wedge \neg Suc$ – **d** je počet příkladů nepokrytých ani předpokladem ani závěrem.

8.6.2 Základní charakteristiky asociačních pravidel

- **Support (podpora)** – relativní četnost objektů splňující předpoklad i závěr, jinými slovy $\frac{\text{počet splňujících výstup}}{\text{počet položek}}$:

$$sup(Ant \Rightarrow Suc) = \frac{a}{a + b + c + d}, \in \langle 0; 1 \rangle. \quad (4)$$

- **Confidence (spolehlivost)** – podmíněná pravděpodobnost závěru pokud platí předpoklad, tedy $\frac{\text{podpora obou}}{\text{podpora závěru}}$:

$$\text{conf}(Ant \Rightarrow Suc) = \frac{\text{sup}(Ant \cup Suc)}{\text{sup}(Suc)}. \quad (5)$$

- Další: **pokrytí, zajímavost, závislost.**

8.7 Hledání často se opakujících množin položek

Frequent item set je množina, kde $\text{sup}(K) \geq \gamma$, máme tedy stanovenou **minimální podporu**. Pokud je např. $\gamma = 0,3$ pak je minimální podpora 30%.

8.7.1 Generování kombinací

Základem všech algoritmů pro hledání asociačních pravidel je **generování kombinací (konjunkcí) hodnot atributů**. Při generování vlastně procházíme (prohledáváme) prostor všech přípustných konjunkcí. Metod je několik:

- do hloubky,
- do šířky,
- heuristicky,

8.7.2 Algoritmus apriori

Jedná se o nejznámější algoritmus pro hledání asociačních pravidel. Jádrem algoritmu je **hledání často se opakujících množin** položek (frequent itemsets). Jedná se kombinace (konjunkce) kategorií, které dosahují předem zadané četnosti (**minimální podpory**) v datech.

Algoritmus apriori

1. do L_1 přiřaď všechny kategorie, které dosahují alespoň požadované četnosti
2. polož $k=2$
3. dokud $L_{k-1} \neq \emptyset$
 - 3.1. pomocí funkce *apriori-gen* vygeneruj na základě L_{k-1} množinu kandidátů C_k
 - 3.2. do L_k zařaď ty kombinace z C_k , které dosáhly alespoň požadované četnosti
 - 3.3. zvětš počítadlo k

Funkce *apriori-gen*(L_{k-1})

1. pro všechny dvojice kombinací $Comb_p, Comb_q$ z L_{k-1}
 - 1.1. pokud $Comb_p$ a $Comb_q$ se shodují v $k-2$ kategoriích přidej $Comb_p \wedge Comb_q$ do C_k
2. pro každou kombinaci $Comb$ z C_k
 - 2.1. pokud některá z jejích podkombinací délky $k-1$ není obsažena v L_{k-1} odstraň $Comb$ z C_k

Při hledání kombinací délky k , které mají vysokou četnost se využívá toho, že **již známe kombinace délky $k-1$** . Při vytváření kombinace délky k spojujeme kombinace délky $k-1$.

Jde tedy o **generování kombinací „do šířky”**. Přitom pro vytvoření jedné kombinace délky k požadujeme, aby všechny její podkombinace délky $k - 1$ **splňovaly požadavek na četnosti**. Tedy např. ze tříčlenných kombinací $\{A_1A_2A_3, A_1A_2A_4, A_1A_3A_4, A_1A_3A_5, A_2A_3A_4\}$ dosahujících požadované četnosti vytvoříme **pouze jedinou čtyřčlennou** kombinaci $A_1A_2A_3A_4$. Kombinaci $A_1A_3A_4A_5$ sice lze vytvořit spojením $A_1A_3A_4$ a $A_1A_3A_5$, ale mezi tříčlennými kombinacemi chybí $A_1A_4A_5$ i $A_3A_4A_5$.

9 Metrické a topologické prostory – metriky a podobnosti.

10 Shlukování.

11 Náhodná veličina. Základní typy náhodných veličin. Funkce určující rozdělení náhodných veličin.

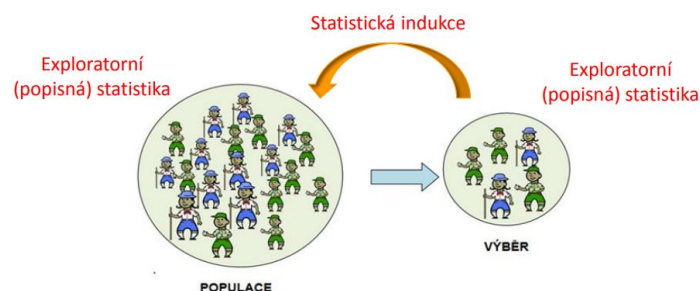
- 12 Vybraná rozdělení diskrétní a spojité náhodné veličiny - binomické, hypergeometrické, negativně binomické, Poissonovo, exponenciální, Weibullovo, normální rozdělení.

13 Popisná statistika. Číselné charakteristiky a vizualizace kategoriálních a kvantitativních proměnných.

14 Metody statistické indukce. Intervalové odhady. Princip testování hypotéz.

14.1 Statistická indukce

Statistická indukce je metoda, která dovoluje stanovit vlastnost celku (**základního souboru**) na základě pozorování jeho částí (**náhodného výběru**).



Základní soubor (populace)

- Je množina všech teoreticky možných objektů (např. jedinců) v uvažované situaci = statistický soubor, který je vymezen cílem výzkumu a pro který vyvozujeme závěry výzkumného šetření.
- Charakterizuje se **parametrem**, což je např. výška, váha, IQ, atp.
- Má konečný nebo nekonečný (hypotetický) **rozsah**, který je dán N (např.: $N = 150$ lidí, opic, rostlin,...).

Výběrový soubor (výběr)

- Je část populace vybrané na základě předem stanovených kritérií resp. pravidel (podmnožina základního souboru).
- O **náhodném výběru** uvažujeme, když splňuje dvě základní vlastnosti: pravděpodobnost zařazení do vzorku je pro všechny statistické jednotky populace nenulová a statistické jednotky jsou do vzorku vybrané nezávisle jedna od druhé.
- O **reprezentativním výběru** uvažujeme, když výběrový soubor dobře odráží strukturu celého zkoumaného souboru.

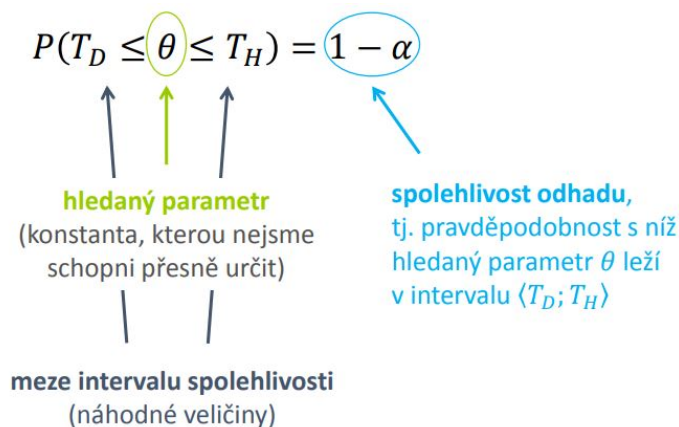
Principy statistického usuzování

1. Statistické usuzování znamená zobecňování z výběrových statistik na parametry rozdělení.
2. Abychom mohli provést statistické usuzování, musíme mít nějakou teorii, jež popisuje náhodné chování sledovaných proměnných.
3. Existují dva typy výběrových chyb: náhodné výběrové chyby a systematické chyby. Získáním náhodného výběru zmenšujeme systematickou chybu a získáváme podklad pro odhad náhodné výběrové chyby.

4. Výběrová rozdělení statistik jsou teoretická pravděpodobnostní rozdělení, která popisují vztah mezi výběrovou statistikou a populací.
5. Směrodatná odchylka výběrového rozdělení statistiky (odhad parametru) se nazývá směrodatná chyba. Odhaduje náhodnou výběrovou chybu vypočítané statistiky (odhadu parametru).
6. Jak roste velikost výběru, výběrová chyba a směrodatná chyba se zmenšují.
7. Směrodatná chyba se používá k získání intervalového odhadu parametrů i k testování hypotéz o parametrech rozdělení.

14.1.1 Intervalové odhady

- V praktických aplikacích často určujeme **odhad příslušného parametru** pomocí intervalového odhadu.
- Tento odhad je reprezentován intervalem $\langle t_D, t_H \rangle$, v němž hledaný parametr leží s předem určenou pravděpodobností (spolehlivostí), kterou označujeme $(1 - \alpha)$.
- neboli parametr populace aproximujeme intervalem, v němž s velkou pravděpodobností příslušný populační parametr leží.
- **Interval spolehlivosti** (konfidenční interval) pro parametr θ se spolehlivostí $1 - \alpha$, kde $\alpha \in \langle 0; 1 \rangle$, je taková dvojice statistik (T_D, T_H) , že $P(T_D \leq \theta \leq T_H) = 1 - \alpha$.



- **Intervalový odhad** t_D, t_H je jednou z realizací intervalu spolehlivosti.
- Požadavky na interval spolehlivosti:
 - Co **největší spolehlivost** odhadu.
 - Co **nejmenší šířka** intervalu spolehlivosti. (S rostoucí šířkou intervalového odhadu klesá významnost získané informace.)
- S rostoucí spolehlivostí se zvětšuje šířka intervalového odhadu a tím klesá významnost takto získané informace.

- S rostoucím rozsahem výběru se šířka intervalového odhadu snižuje.
- Typy intervalů spolehlivosti:

– **oboustranné**

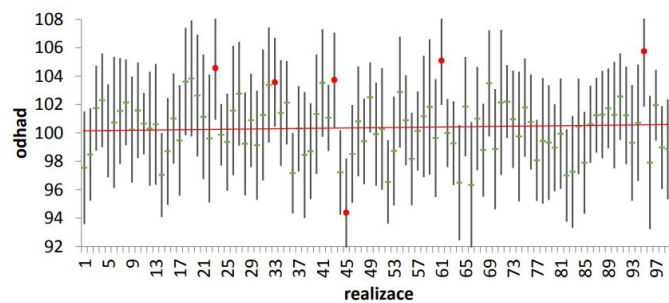
$$P(\theta < T_D) = P(\theta > T_H) = \frac{\alpha}{2}$$

Tyto dvě podmínky zaručují, že $P(T_D \leq \theta \leq T_H) = 1 - \alpha$

– **jednostranné** (odhadujeme-li například délku života nějakého zařízení, je pro nás důležitá pouze dolní mez)

* **levostranné** $P(\theta \geq T_D^*) = 1 - \alpha$

* **pravostranné** $P(\theta \leq T_H^*) = 1 - \alpha$



Obrázek 1: Co to znamená, že spolehlivost odhadu je $1 - \alpha$?

Simulace 100 intervalových odhadů střední hodnoty (spolehlivost 0,95) získaných na základě opakovaných výběrů o rozsahu 30 z populace se střední hodnotou 100. 6 intervalů ze 100 neobsahuje skutečnou střední hodnotu

14.2 Jak najít intervalový odhad parametru θ ?

Obecně:

1. Zvolíme vhodnou výběrovou charakteristiku $T(\mathbf{X})$, jejíž rozdělení známe.
- 2.

$$P\left(\frac{x_\alpha}{2} \leq T(\mathbf{X}) \leq x_{1-\frac{\alpha}{2}}\right) = 1 - \alpha,$$

$$P(T(\mathbf{X}) \leq x_{1-\alpha}) = 1 - \alpha,$$

$$P(T(\mathbf{X}) \geq x_\alpha) = 1 - \alpha.$$