I. Matematické základy informatiky

Update: 9. května 2018

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

1.1 Konečné automaty (KA)

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

Konečné automaty se dělí na **determistické** a **nedetermistické**. 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.

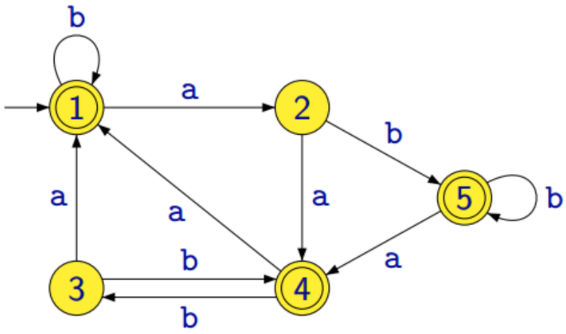
1.1.1 Deterministický konečný automat (DKA)

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**, $\delta : Q \times \Sigma \to Q$.
- q_0 je **počáteční stav**, $q_0 \in Q$.
- F je neprázdná množina **koncových** neboli **přijímajících stavů**, $F \subseteq Q$.

Příklad

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

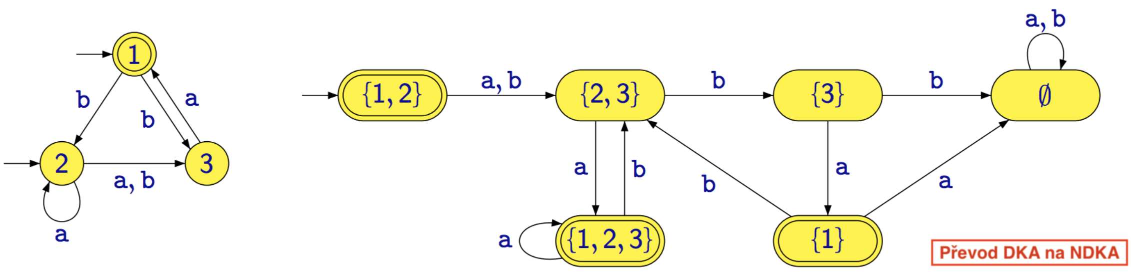


	δ	a	b
↔	1	2	1
	2	4	5
	3	1	4
←	4	1	3
←	5	4	5

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, $\delta : Q \times \Sigma \to P(Q)$, v případě **ZNKA** $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to P(Q)$.
- 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 $\to$ **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 $(\alpha + \beta)$, $(\alpha \cdot \beta)$, $(\alpha *)$ jsou regulární výrazy, kde:

- $(\alpha + \beta)$ označuje **sjednocení** jazyků označených α a β,
- $(\alpha \cdot \beta)$ označuje **zřetězení** jazyků označených α a β,
- $(\alpha *)$ 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 $\epsilon, 01, 0101, 010101, \ldots$,
- **(0+1)*** ... jazyk tvořený všemi slovy nad abecedou $\{0, 1\}$,
- **(01)*111(01)*** ... jazyk tvořený všemi slovy obsahující podslovo 111, předcházení i následované libovolným počtem slov 01,
- **(0+1)*00+(01)*111(01)*** ... jazyk tvořený všemi slovy, která buď končí 00 nebo obsahují podslovo 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.

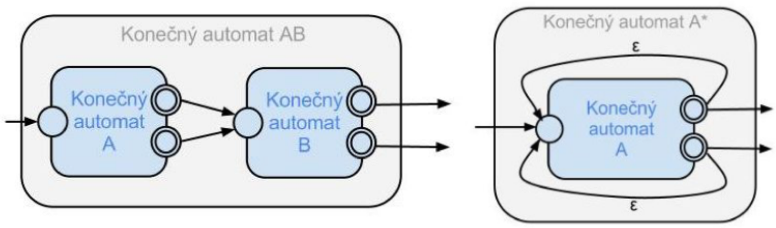
1.3 Uzávěrové vlastnosti třídy regulárních jazyků

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

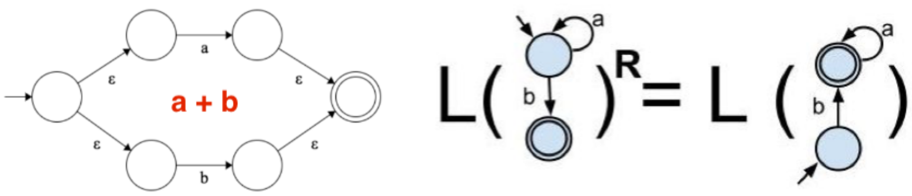
- **Sjednocení**, **průnik**, **doplněk** – je-li $L_1, L_2 \in \text{REG}$, pak také $L_1 \cup L_2, L_1 \cap L_2, L_1'$ jsou v REG.
- **Zřetězení**, **iterace** – je-li $L_1, L_2 \in \text{REG}$, pak také $L_1 \cdot L_2, L_1^*$ jsou v REG.
- **Zrcadlový obraz** – je-li $L \in \text{REG}$, pak také L^R 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 příjmajících a ostatních stavů, u NKA je nejprve nutné provézt převod na DKA.

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

2.1 Bezkontextové gramatiky (BG)

Bezkontextová gramatika definuje **bezkontextový jazyk**. Je tvořena **neterminály** (proměnné), **terminály** (konstanty) a **pravidly**, které každému neterminálu definují přepisovací pravidla. Jeden neterminál označíme jako **startovní**, kde začínáme a podle pravidel je dál přepisujeme na výrazy složené z terminálu a neterminálu. Jakmile už není co přepisovat, výraz obsahuje už jen neterminály, získali jsme **slovo**.

- Je **uzavřená** vůči operacím **sjednocení**, **zřetězení**, **iteraci** a **zrcadlový obraz**.
- Ke každé bezkontextové gramatice existuje **ekvivalentní zásobníkový automat**.

2.1.1 Formální definice BG

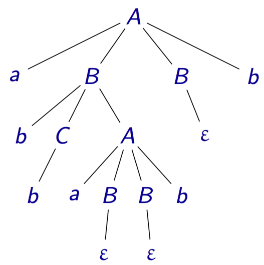
Bezkontextová gramatika je definována jako uspořádaná čtveřice $G = (\Pi, \Sigma, S, P)$, kde:

- Π (*velké pí*) je konečná množina **neterminálních** symbolů (neterminálů).
- Σ je konečná množina **terminálních** symbolů (terminálů), $\Pi \cap \Sigma = \emptyset$.
- S je **počáteční neterminál**, $S \in \Sigma$.
- P je konečná množina **přepisovacích pravidel**, $P \subseteq \Pi \times (\Pi \cup \Sigma)^*$.

2.1.2 Základní pojmy

- **Bezkontextový jazyk** – formální jazyk, který je akceptovaný nějakým zásobníkovým automatem.
- **Derivace slova** – jedno konkrétní odvození slova pomocí gramatiky, tedy záznam postupných přepisů od startovního neterminálu po konečné slovo. Derivace se podle postupu při přepisování dělí na:
  - **levou** – přepisujeme nejprve levé neterminály,
  - **pravou** – přepisujeme nejprve pravé neterminály.
- **Derivačni strom** – grafické znázornění derivace slova stromem. Pro všechny možné derivace (levou, pravou, moji) by měl derivační strom být **stejný**. Není-li tomu tak jedná se o **nejednoznačnou gramatiku**, což je nežádoucí jev.
  - **Špatně** $= \mathrm{A} \to \mathrm{A} \mid \epsilon$ (lze generovat až N způsoby), **Správně** $= \mathrm{A} \to \epsilon$
- **Chomského normální forma** – gramatika může obsahovat pouze pravidla typu: **A** $\to$ **BC** nebo **A** $\to$ **a** nebo **S** $\to \epsilon$ (pokud gramatika generuje pouze prázdný řetězec).
- **Nevypouštějící gramatika** – neobsahuje ϵ (*epsilon*) přechody.

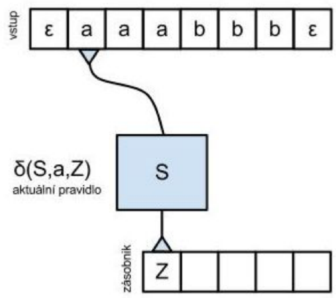
$A \rightarrow aBBb \mid AaA$
$B \rightarrow \varepsilon \mid bCA$
$C \rightarrow AB \mid a \mid b$

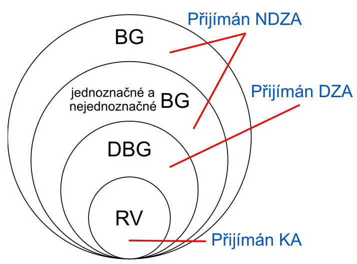


$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow$
$abbaBbb \Rightarrow abbabb$

2.2 Zásobníkové automaty (ZA)

Slouží k **rozpoznání bezkontextových jazyků**. S využitím zásobníků si může pamatovat kolik a jaké znaky přečetl, což je potřeba právě k rozpoznání bezkontextového jazyka. Zásobníkový automat je v podstatě konečný automat rozšířený o zásobník.





- ZA na základě **aktuálního znaku** na pásce, **prvního znaku v zásobníku** a **aktuálního stavu** změní svůj stav a **přepíše** znak v zásobníku podle daných pravidel.
- ZA **přijímá** dané slovo, jestliže skončí v konfiguraci (q, ϵ, ϵ), tedy když se přečte celé vstupní slovo a zásobník je **prázdný**.
- **Konfigurace** je dána: aktuálním stavem, obsahem pásky a obsahem zásobníku.
- **Deterministický** – nesmí se objevit **stejná konfigurace vícekrát**.

2.2.1 Formální definice zásobníkového automatu

Zásobníkový automat M je definován jako šestice $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$, kde:

- Q je konečná neprázdná množina **stavů**.
- Σ je konečná neprázdná množina **vstupních symbolů** (vstupní abeceda).
- Γ (*velká gamma*) je konečná neprázdná množina **zásobníkových symbolů**.
- δ je **přechodová funkce** (konečná množina instrukcí), $\delta : Q \times (\Sigma \cup \epsilon) \times \Gamma \text{ß} P_{\text{fin}}(Q \times \Gamma^*)$.
- q_0 je **počáteční stav**, $q_0 \in Q$.
- Z_0 je **počáteční zásobníkový symbol**, $Z_0 \in \Gamma$.

2.2.2 Definice instrukcí (pravidel) v ZA

Instrukce (sady instrukcí reprezentují přechodovou funkci δ) definují **chování automatu**:

$$(q, a, X) \rightarrow (q', \alpha), \text{ kde } a \in \Sigma. \tag{1}$$

Tato instrukce je aplikovatelná jen v situaci (neboli konfiguraci), kdy **řídicí jednotka** je ve stavu q, **čtecí hlava** na vstupní pásce čte symbol a a na vrcholu zásobníku je symbol X. Pokud je **instrukce aplikována**, vykoná se následující:

1. řídicí jednotka **přejde do stavu** q,
2. čtecí hlava na vstupní pásce se **posune o jedno políčko doprava**,
3. vrchní symbol v zásobníku se **odebere** (vymaže),
4. **na vrchol zásobníku se přidá** řetězec α tak, že jeho nejlevější symbol je aktuálním vrcholem zásobníku.

Pravidlo	Akce (Z = zásobník)	Význam
$\delta(q_1, a, X) \rightarrow (q_1, YX)$	**přidání** prvku do Z	na začátek zásobníku se vloží Y
$\delta(q_1, a, X) \rightarrow (q_1, Y)$	**přepsání** prvku v Z	první prvek zásobníku se přepíše na Y
$\delta(q_1, a, X) \rightarrow (q_1, \epsilon)$	**smazání** prvku ze Z	první prvek zásobníku se smaže neboli nahradí prázdným slovem ϵ
$\delta(q_1, a, X) \rightarrow (q_2, X)$	**změna stavu**	stav q_1 se změní na stav q_2
$\delta(q_1, a, X) \rightarrow \emptyset$	**pád** automatu	ukončení výpočtu, slovo nebylo přijato

2.3 Převod BG na zásobníkový automat

Využívá se tzv. metody shora-dolů, která obsahuje pouze **1 stav**:

1. pro všechny **neterminály** vypíšu pravidla typu: $(q, \epsilon, A) \rightarrow \{(q, B), (q, C)\}$,
2. všechny **terminály** přepíšu na pravidla typu: $(q, a, a) \rightarrow (q, \epsilon)$.

Příklad

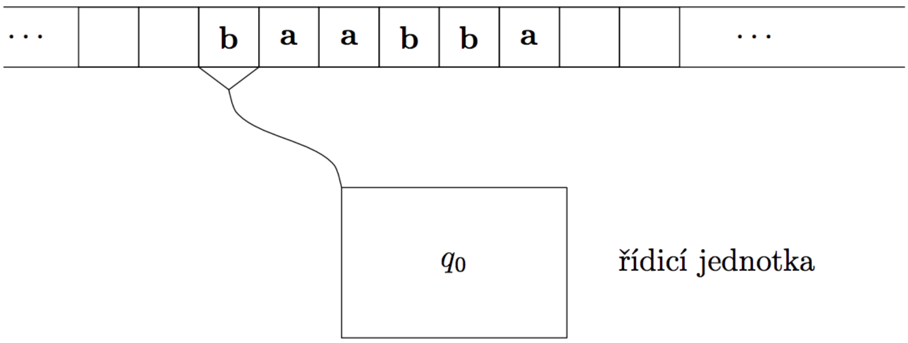
Vstupní gramatika:	**Instrukce, převedené dle výše uvedených pravidel:**
$S \rightarrow A\vert\ B$	$(Q, \epsilon, S) \rightarrow \{(q, A), (q, B)\}$
$A \rightarrow a$	$(Q, \epsilon, A) \rightarrow (q, a)$
$B \rightarrow (c)$	$(Q, \epsilon, B) \rightarrow (q, (c))$
$\Sigma = \{A, B, S\}$	$(Q, a, a) \rightarrow (q, \epsilon)$
$\Gamma = \{a, c, (,)\}$	$(Q, (, () \rightarrow (q, \epsilon)$
	$(Q, c, c) \rightarrow (q, \epsilon)$
	$(Q,),)) \rightarrow (q, \epsilon)$

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 $\Box$, **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 (přinejmenším) speciální znak $\Box$ (prázdný znak [Blank]).
- δ je přechodová funkce, $\delta : (Q - F) \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$.
- 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: $(q, a) \to (q', a', m)$ je tento:

(akt. stav $[q]$, znak na pásce $[a]$) $\to$ (**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:

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

3.1.4 Modifikace TS

- **N-páskový TS** – **čte** a **zapisuje** do **více pásek** najednou, jediná změna je v přechodové funkci: $\delta : Q \times \Gamma^n \to Q \times (\Gamma \times \{L, R, N\})^n$.
- **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

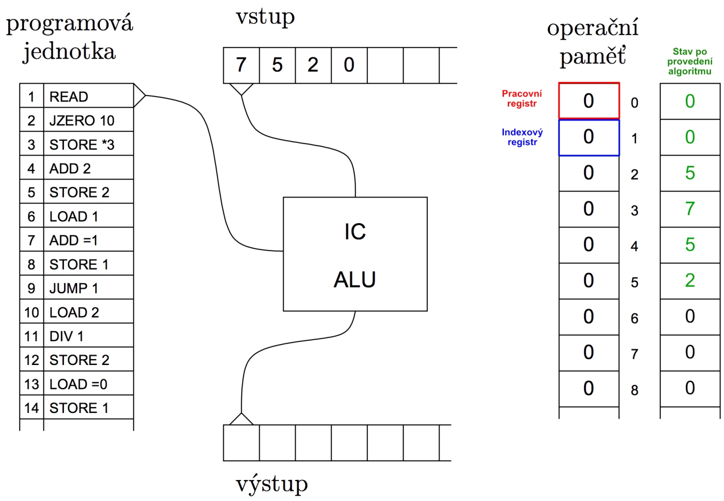
- **Turingovsky úplný** – stroj (počítač, programovací jazyk, úloha, ...), která má stejnou výpočetní sílu jako TS. Lze v něm **odsimulovat** libovolný jiný TS zadaný na vstupu.
- **Church-Turingova teze** – říká, že jakýkoliv výpočet lze úspěšně uskutečnit algoritmem běžícím na počítači, tedy „ke každému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

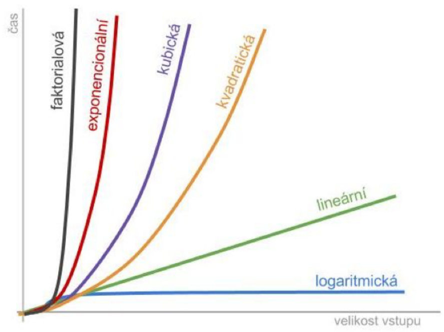
- **Časová složitost** – sleduje jak závisí **doba** výpočtu alg. na množství vstupních dat.
- **Prostorová složitost** – sleduje jak závisí **množství použité paměti** výpočtu alg. na množství vstupních dat.

Jelikož konkrétní čísla (čas, bity) se liší v **závislosti vstupních datech**, množství zpracovávaných dat a použitém programovacím jazyku, neudává se složitost čísly, 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 používjí tyto zapisy:

- $f \in O(f)$ – 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:

- $f(n) \in \Omega(logn)$ – 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.

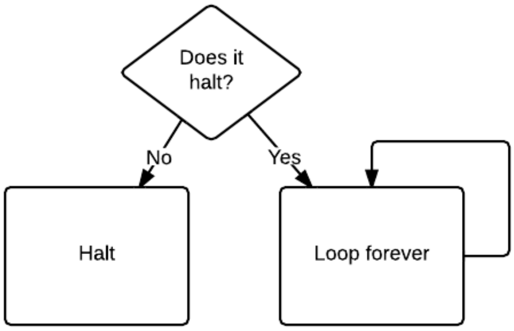
3.5.7 Příklady nerozhodnutelných problémů

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) složitosti**, tj. s časovou složitosti $O(n^k)$, kde k je nějaká konstanta.

Tuto třídu problému považujeme za zvládnutelnou. Je **robustní** – mezi jednotlivými výpočetními modely (RAM, TS) existují vzájemné polynominální 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

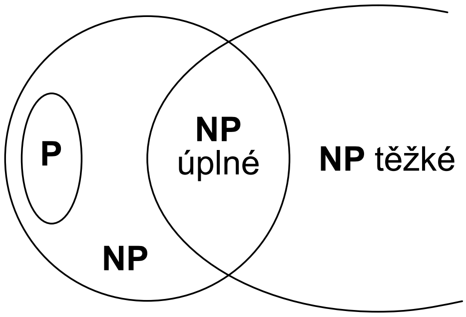
Třída všech rozhodovacích problémů (ano/ne), které jsou rozhodovány **nedeterministickými algoritmy s polynomiální časovou složitosti**. Tedy problémy, u kterých je možné pro daný vstup vždy ověřit zda je odpověď ANO, v případě NE tomu tak být nemusí.

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: $\forall P \in$ NPTIME : $P \triangleright Q$.

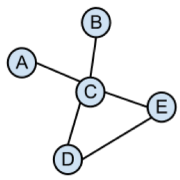
4.3.2 NP-úplný problém

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 laterály.
    - **Otázka**: Je formule splnitelná?
6. **HK (problém hamiltonovské kružnice)/HC (problém hamiltonovskho cyklu)**
    - **Vstup**: Neorientovaný graf G/Orientovaný grav G.
    - **Otázka**: Existuje v G hamiltonovská kružnice (uzavřená cesta, procházející každým vrcholem právě jednou)?
7. **Subset-Sum**
    - **Vstup**: Množina přirozených čísel $M = x_1, x_2, \ldots, x_n$ 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 neujet víc než danou vzdálenost?

9. **Vrcholové pokryti (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 teto množině?

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

Predikátová logika (PL) pracuje s primitivními formulemi (**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 **O**pice pak má rádo **B**anány.
Judy je individuum s vlastností být **O**pice.

Judy je individuum s vlastností mít rádo **B**anány.

$\forall x(O(x) \to B(x)); O(J) \neq B(J)$, 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: $x, y, z, \ldots$
- Logické spojky: $\wedge$ konjunkce, $\vee$ disjunkce, $\to$ implikace, $\leftrightarrow$ ekvivalence, $\neg$ negace.
- Kvantifikační symboly: $\forall, \exists$.

Speciální symboly: (n-arita = počet argumentů)

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

Pomocné symboly: závorky a jiná interpunkční znaménka (,), …

5.1.2 Formální jazyk PL1 – Gramatika

Termy:

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

Atomické formule:

- je-li P n-ární predikátový symbol a jsou-li $t_1, \ldots, t_n$ termy, pak výraz $P(t_1, \ldots, t_n)$ 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", ...
- $\exists$ – „někdo", „něco", „někteří", „existuje", ...

Větu musíme často ekvivalentně přeformulovat, pozor: v češtině **dvojí 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: $\forall \ldots \rightarrow$.
- Po **existenčním** kvantifikátoru formule ve tvaru konjunkce: $\exists \ldots \wedge$.

5.2.1 Volné a vázané proměnné

$$\forall x \exists y P(x, y, t) \wedge \neg\exists x Q(y, x)$$

vázané, volné　　　　volné, vázané

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: $\neg\forall x[V(x) \rightarrow Z(x)] \quad \leftrightarrow \quad \exists x[V(x) \wedge \neg Z(x)]$.
- **De morganovy** zákony: $\forall x[((P(x) \wedge Q(x)) \vee D(x)] \quad \leftrightarrow \quad \forall x[(P(x) \vee D(x)) \wedge (Q(x) \vee D(x))]$.

  $$\neg(A \vee B) \iff (\neg A) \wedge (\neg B)$$
  $$\neg(A \wedge B) \iff (\neg A) \vee (\neg B)$$

- Převod **implikace**: $\forall x(P(x) \rightarrow G(x)) \quad \leftrightarrow \quad \forall x(\neg P(x) \vee G(x))$.

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 **znamenají** 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 $A_1, \ldots, A_n$ je **libovolná podmnožina kartézského** součinu $A_1 \times \ldots \times A_n$ (tyto množiny jsou **nosičemi** 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** – $\emptyset \subset \rho \subset A_1 \times \ldots \times A_n$.

6.1.2 Vlastnosti relací

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

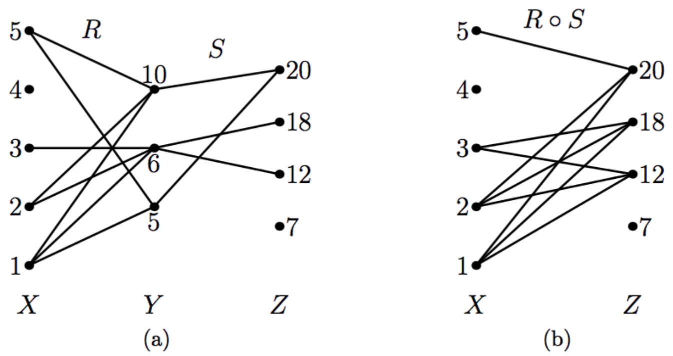
- **Reflexivní** – $\forall x \in A : (x, x) \in R$.
- **Ireflexivní** – $\forall x \in A : (x, x) \notin R$.
- **Symetrická** – $\forall x \in A : (x, y) \in R \Rightarrow (y, x) \in R$.
- **Asymetrická** – $\forall x \in A : (x, y) \in R \Rightarrow (y, x) \notin R$.
- **Antisymetrická** – $\forall x \in A : (x, y) \in R \wedge (y, x) \in R \Rightarrow x = y$.
- **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** $R2(x \in R1)$.
- **Doplněk** – Doplňkem $R1'$ 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 **ekvivalenece** $(=)$ [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 uspořádání**, pokud je R na A: **reflexivní**, **antisymetrická** a **tranzitivní**.
- Binární relace na množině X je **ostrým uspořádání**, 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

$$f : A^n \to A, \tag{2}$$

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**: $\overline{x}$ takové, že $\forall x \in X : x \circ \overline{x} = \overline{x} \circ x = e$. 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: $f(a_1 \cdot a_2) \to f(a_1) \circ f(a_2)$.
- **Izomorfismus** – bijektivní homomorfismus.
- **Epimorfismus** – subjektivní homomorfismus.
- **Monomorfismus** – injektivní homomorfismus.
- **Endomorfismus** – homomorfismus z objektu do sebe sama (stejná množina).
- **Automorfismus** – endorfismus, který je izomorfní.

7.4 Okruhy (Algebraické struktury s dvěma binárními operací)

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.

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. Okruhy pokývají předměty Teoretická informatika, Pravděpodobnost a statistika, Matematika pro zpracování znalostí