

Okruhy jsou aktualizovány a opravovány v příslušných souborech pro samostatné okruhy (složka ODT). Opravy **nebudou** propagovány do tohoto jednotného souboru, který sloužil pro prvotní sepisování. Tento jednotný soubor a jednotlivá PDF budou aktualizovány nárazově po větších úpravách.

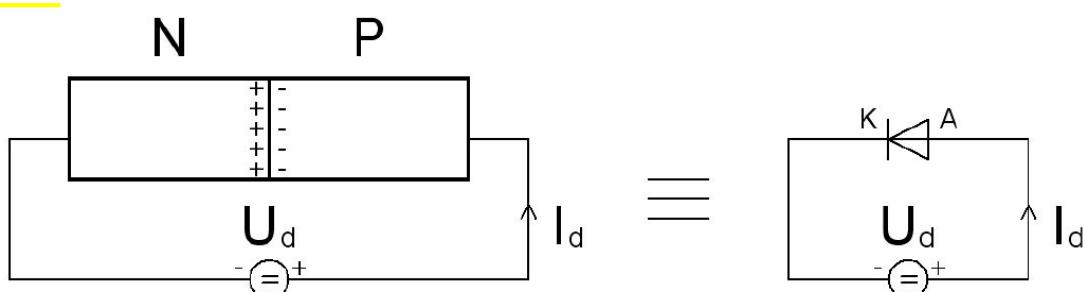
Návrhy na změny pište nejlépe přímo k samostatným souborům pro příslušné okruhy.

Pokud si někdo chce všechno vytisknout, sjednoťte si všechny samostatné ODT soubory a tiskněte až to. Veškeré aktualizace jsou prováděny do nich včetně upravení pozice obrázků, které jinak na tisku budou nečitelné.

[Složka se všemi materiály \(5d7QTv\\$jYPk8E8mJ22jYaktualizované soubory jsou v složce ODT\)](#)

# 1. Princip činnosti polovodičových prvků (dioda, bipolární a unipolární tranzistor ve spínacím režimu, realizace logických členů NAND a NOR v technologii CMOS).

- elektrony tečou od - k +, proud značíme, že teče od + k -.
- Základem jsou polokovy, hlavně **křemík**.
- Existují 2 typy polovodičových materiálů založených na příměsích:
  - **N** (negativní) - Při vazbě atomů vzniká v polokovu **volný elektron**, který je schopný vést proud - elektronová vodivost. Dopování **fosforem**.
    - majoritní: elektrony
    - minoritní: díry
  - **P** (pozitivní) - Při vazbě vzniká "díra" (**místo kde chybí elektron**) pro vedení proudu. Dopování **borem**.
    - majoritní: díry
    - minoritní: elektrony
- Oba typy polovodičových materiálů **nemají na venek elektrický náboj** (uvnitř materiálu je stejný počet protonů jako elektronů), pokud se spojí na mikroskopické úrovni, dojde k přesunu části (maličko) elektronů z **N** do **P**. V **P** vznikne záporný náboj a v **N** kladný náboj.
- V místě dotyku vzniká **potenciálová bariéra** (~0.7 V u křemíku).
- Přechod v propustném směru (anoda = kladná, katoda = záporná)

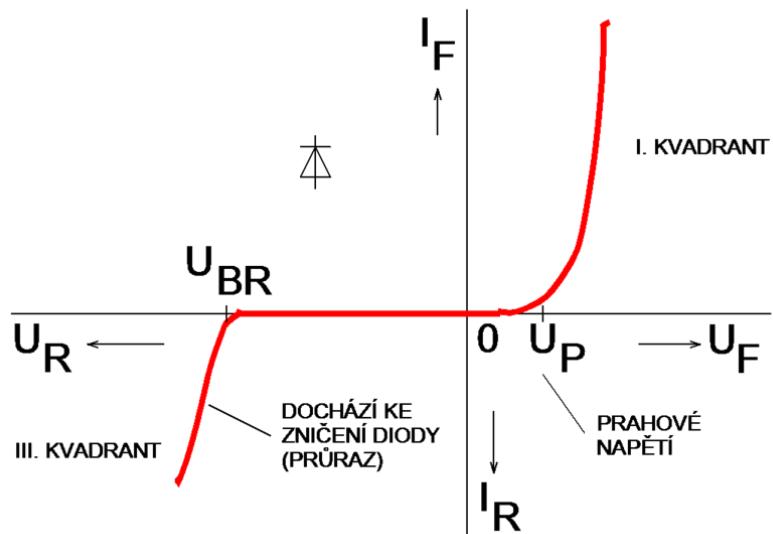


## Polovodičová dioda

obsahuje PN přechod, propouští proud *pouze v jednom směru*. Ideální dioda by propouštěla proud pouze v jednom směru, reálná dioda propouští v jednom lépe než v druhém. V závěrném směru prakticky nepropouští proud až do tzv. průrazného napětí, poté začne procházející proud strmě narůstat, což obvykle vede na zničení

diody (přehřátí a spálení). Pokud k tomu není dioda určená (zenerovy diody), ale i ty lze spálit příliš velkým proudem, musí být omezen prvky v obvodu.

- **Přechodová charakteristika diody - (obr.)**

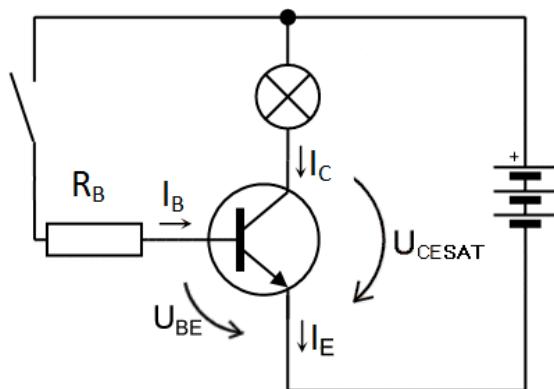


- Existuje více druhů diod (usměrňovací, spínací, fotodiody, luminiscenční...)

## Tranzistor

Elektronická součástka, která má schopnost zesilovat proud.

**Tranzistor jako spínač (spínací režim)** - Dokáže spínat velké proudy a pracuje velmi rychle. Pokud do báze teče proud (stačí malý) tak začne mezi kolektorem a emitorem a tedy i spotřebičem procházet velký proud. Malým proudem spínáme proud velký.

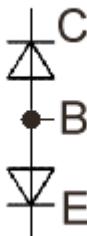
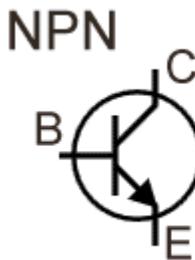


## Bipolární tranzistor BJT (Bipolar Junction Transistor)

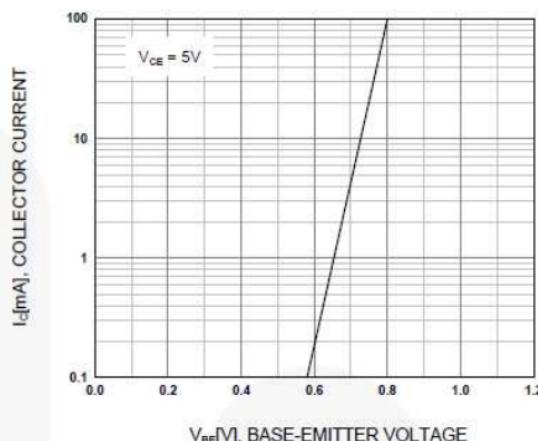
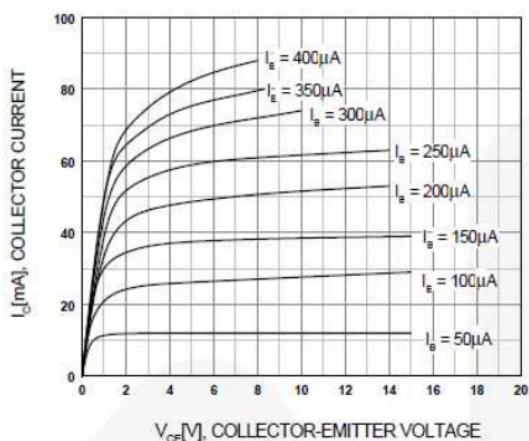
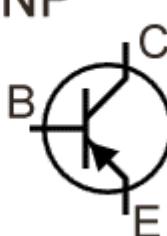
Přenos proudu uskutečňují oba nosiče jak N tak P. Jedná se o **2 PN** přechody vedle sebe - **PNP** nebo **NPN** tranzistor.

- **E - Emitor** je hodně nadopovaný, **B - Báze** je málo nadopovaná a velmi tenká, **C - Kolektor** je středně nadopovaný

- **Princip NPN** - emitter je připojen na katodu (elektrodu uvolňující elektrony) a společně s bází tvoří diodu v propustném směru. Pokud je na bázi přivedeno napětí o  $\sim 0.7V$  větší než na emitoru, začne touto diodou procházet proud. Díky tomu, že emitter je daleko více dopovan než báze, dostává se do báze mnohonásobně větší množství elektronů, než je zde děr. Protože báze je navíc tenká, jsou tyto elektrony přitahovány kladným elektrickým nábojem vznikajícím na kolektoru, který je připojen na anodu, a tranzistorem tak prochází proud. Pokud není na bázi stejně napětí jako na emitoru, nedojde k překonání **potenciálové bariéry** a celým tranzistorem proud neteče.  
<https://youtu.be/DXvAlwMAxiA>
- **Princip PNP** - emitter musí být připojen na anodu, kolektor na katodu a napětí mezi bází a emitorem musí být  $\sim 0.7V$  (mezi emitorem a bází  $\sim 0.7V$ ), jinak je princip obdobný s tím, že uvažujeme pohyb děr..



**PNP**

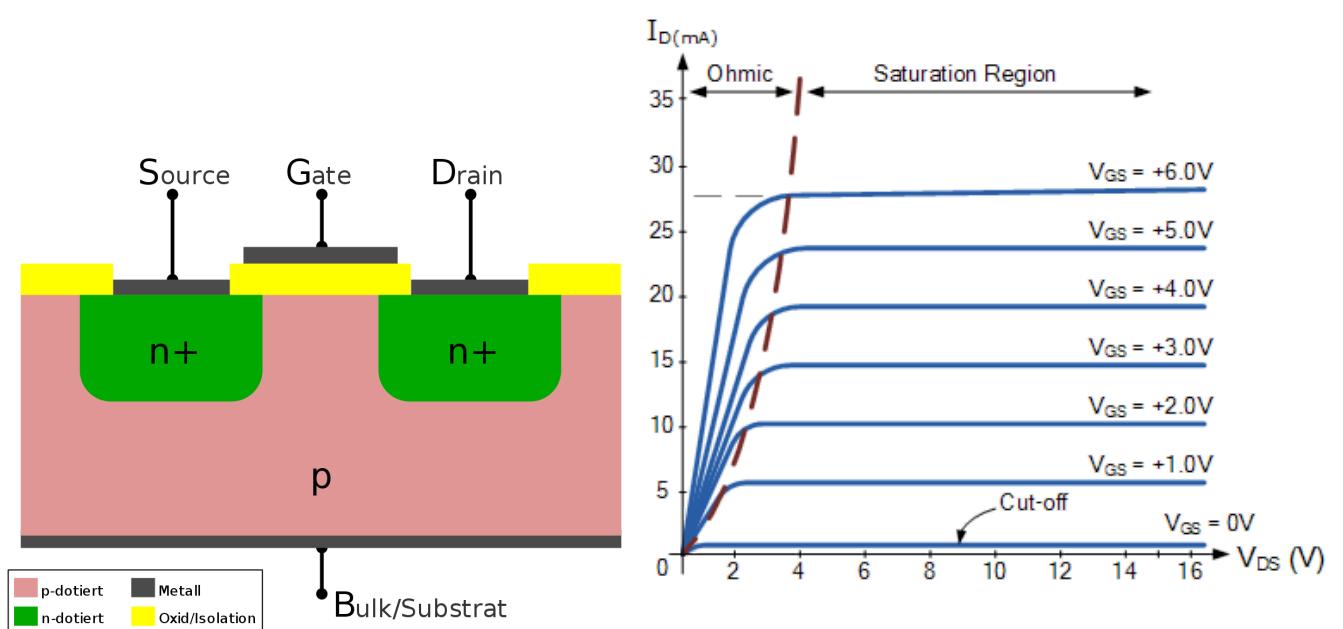


## Unipolární tranzistor MOSFET (Metal Oxide Semiconductor Field Effect Transistor)

Přenos proudu uskutečňuje pouze jeden nosič - **N-channel** nebo **P-channel**, dále jsou děleny na **enhancement** (nevedou proud a musí se zapnout) a **depletion** (vedou proud a musí se vypnout). Jsou řízeny napětím.

- **Gate** - řídící elektroda - přes vrstvu dielektrika (oxid - izolace) připojena k substrátu, **Source** - zdrojová elektroda - silně dopovaná, **Drain** - výstupní elektroda - silně dopovaná, **Body/Substrate** - málo dopován.

- **Princip NPN enhancement** - na Source je přivedeno záporné napětí (stejné jako na Bulk/Body, aby tranzistor nefungoval jako dioda) a na Drain je přivedeno kladné napětí (Drain a Source lze zaměnit). Aby tekl ze Source do Drain proud, musí být na Gate dovedeno kladné napětí (řádově menší, než mezi Source a Drain). Kolem Gate tak vznikne kladný náboj a začne odpuzovat elektrony ze Substrate/Body (díry se přesunují dolů) a zároveň přitahovat elektrony ze Source. Začne se vytvářet postupně od Source k Drain kanál (trojúhelníkový tvar). Při dostatečném napětí na Gate je elektrické pole natolik velké, že kanál propojí Source s Drain a začne téct proud (s vyšším Gate napětím se dále zvyšuje). Zvyšování napětí mezi Source a Drain způsobní i zvyšování **potenciálové bariéry** kolem Drain, což způsobí

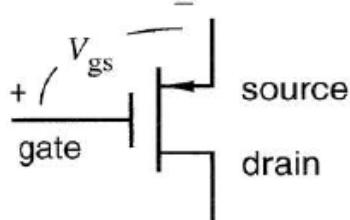


saturační fázi (Saturation Region - nezvyšuje se prakticky protékající proud).

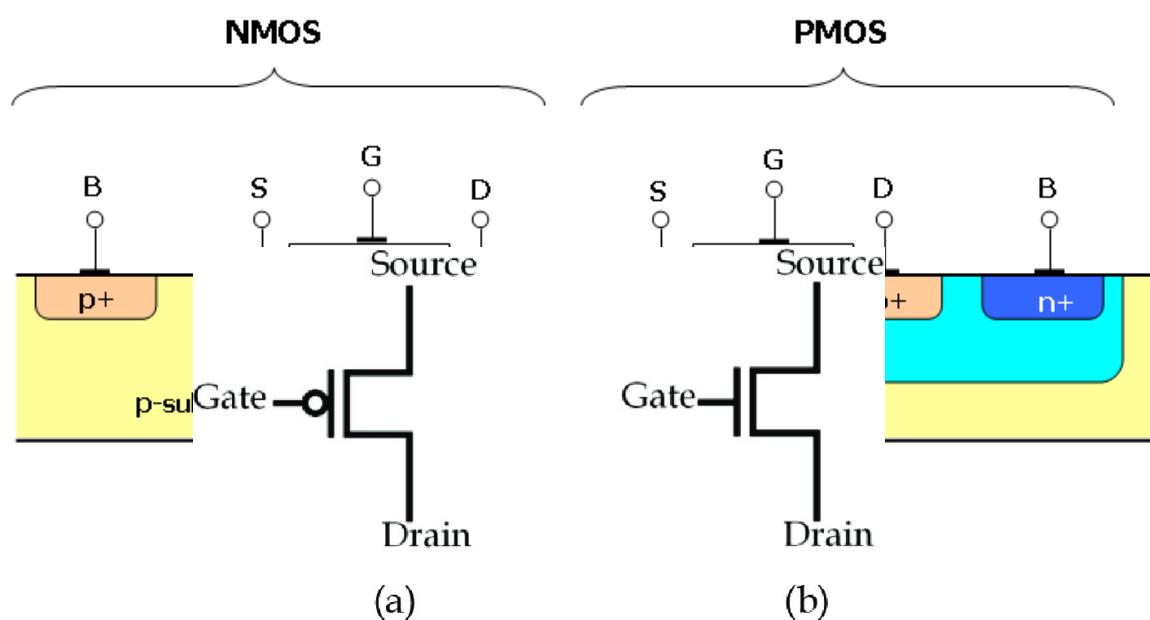
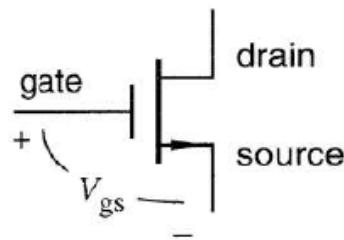
➡ How Does a MOSFET Work?

- **CMOS** - Typ MOSFET, využívá se pro výrobu logických integrovaných obvodů. **Nízká spotřeba a odolnost proti šumu.** Dělí se na (a) **PMOS** a (b) **NMOS**.

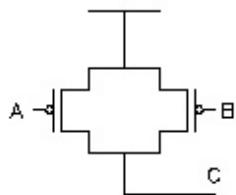
- **PMOS** - záporné (**low - 0**) napětí na gate sepne tranzistor - **děrová vodivost**. Source a Body/Substrate se zapojují na **kladný** pól zdroje, Drain se zapojuje na **záporný** pól zdroje. Source a Drain jsou typu **P**, Substrate je typu **N**. Na Source je menší napětí než na drain.



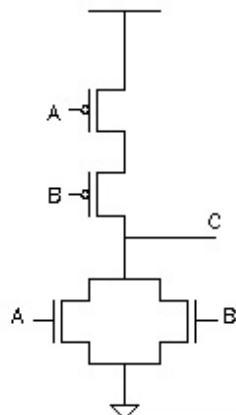
- **NMOS** - kladné (**high - 1**) napětí na gate sepne tranzistor - **elektronová vodivost**. Source a Body/Substrate se zapojují na **záporný** pól zdroje, Drain se zapojuje na **kladný** pól zdroje. Source a Drain jsou typu **N**, Substrate je typu **P**. Na Source je větší napětí než na Drain.



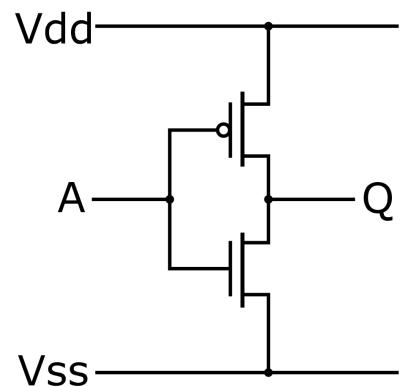
- NAND, NOR A INVERTOR (Technologie CMOS)



NAND



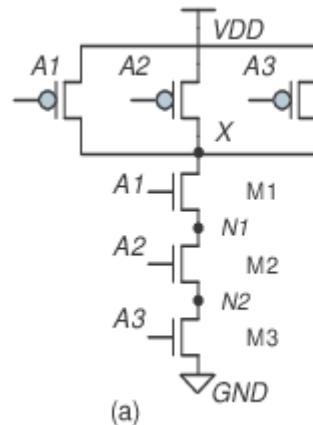
NOR



Z hradla **NAND** lze záměnou **PMOS** za **NMOS** a naopak vytvořit **OR**, z hradla **NOR** lze záměnou **PMOS** za **NMOS** a naopak vytvořit **AND**.

### NAND (Shefferova funkce) logika

$$\begin{aligned}\neg p &\equiv \neg(p \wedge p) \\ p \wedge q &\equiv \neg(\neg(p \wedge q)) \equiv \neg(\neg(p \wedge q) \wedge \neg(p \wedge q)) \\ p \vee q &\equiv \neg(\neg p \wedge \neg q) \equiv \neg(\neg(p \wedge p) \wedge \neg(q \wedge q)) \\ p \rightarrow q &\equiv \neg p \vee q \equiv \neg(p \wedge \neg q) \equiv \neg(p \wedge \neg(q \wedge q))\end{aligned}$$



### NOR (Peirceova funkce) logika

$$\begin{aligned}\neg p &\equiv \neg(p \vee p) \\ p \wedge q &\equiv \neg(\neg p \vee \neg q) \equiv \neg(\neg(p \vee p) \vee \neg(q \vee q)) \\ p \vee q &\equiv \neg(\neg(p \vee q)) \equiv \neg(\neg(p \vee q) \vee \neg(p \vee q)) \\ p \rightarrow q &\equiv \neg p \vee q \equiv \neg(p \vee p) \vee q \equiv \neg(\neg(p \vee p) \vee q) \vee \neg(\neg(p \vee p) \vee q)\end{aligned}$$

### Odkazy:

- Jak funguje tranzistor - [Transistors, How do they work ?](#)
- VA charakteristika - [https://upload.wikimedia.org/wikipedia/commons/4/43/Threshold\\_formulation\\_no\\_watermark.gif](https://upload.wikimedia.org/wikipedia/commons/4/43/Threshold_formulation_no_watermark.gif)
- [Working of Transistor as a Switch - NPN and PNP Transistors](#)
- <http://old.spsemoh.cz/vyuka/zel/tranzistory-bip.htm>
- [Transistor Logic NOT Gate - Inverter](#)

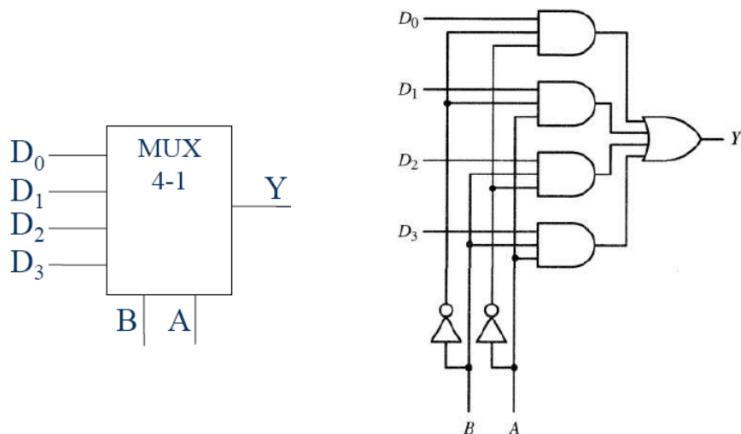
- Prakticky polovodiče a tranzistory - kapitola 7  
[https://knihy.nic.cz/files/edice/hradla\\_volty\\_jednocipy.pdf](https://knihy.nic.cz/files/edice/hradla_volty_jednocipy.pdf)

## 2. Kombinační logické obvody (multiplexor, demultiplexor, kodér, dekodér, binární sčítací).

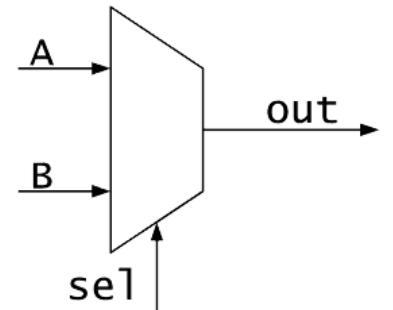
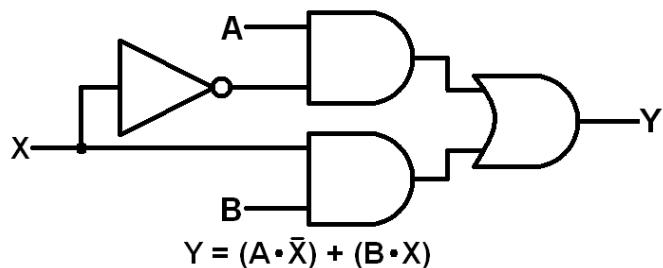
- Kombinační obvod je takový, kde jsou **hodnoty výstupních hodnot závislé pouze na kombinaci hodnot na vstupu**. Tedy nemají paměť.

### Multiplexor

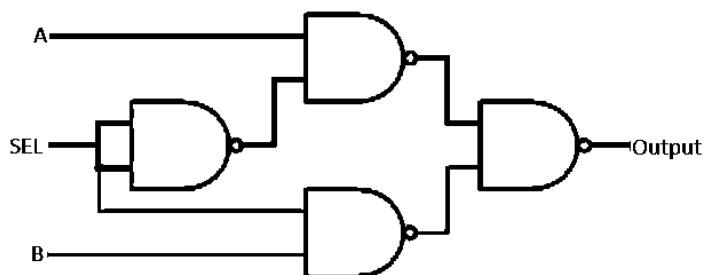
Kombinací **N řídících vstupů** vybírá na **výstup hodnotu jednoho** z až  $2^N$  vstupů.



2-1 multiplexor pomocí NOT, AND, OR

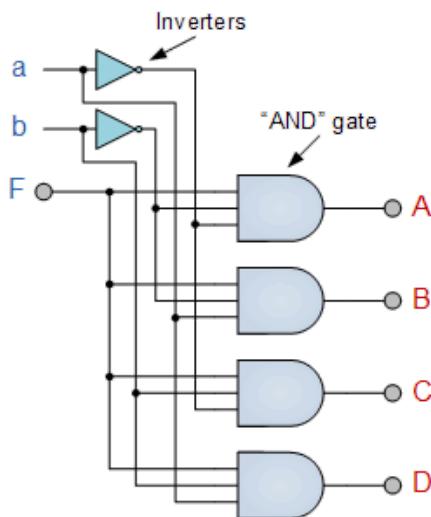


2-1 multiplexor pomocí NAND hradel

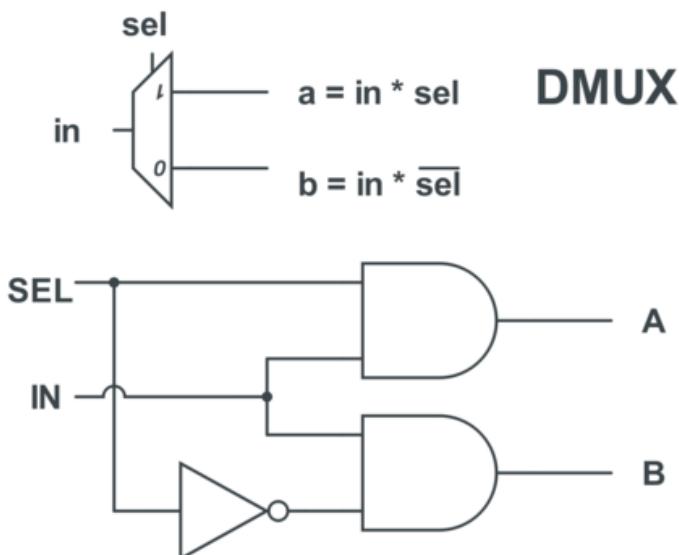


## Demultiplexor

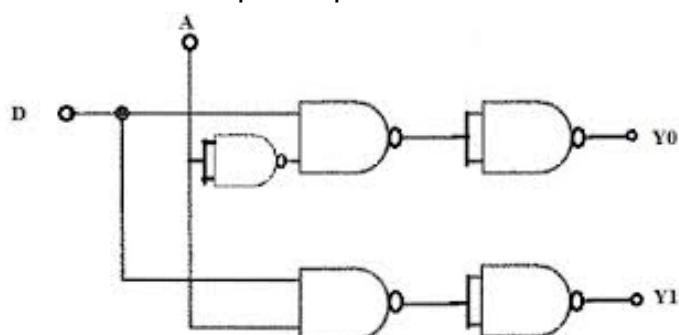
Kombinací **N řídících vstupů** vysílá vstup na jeden z obvykle  $2^N$  výstupů.



1-2 demultiplexor pomocí **NOT** a **AND** hradel



1-2 demultiplexor pomocí **NAND** hradel

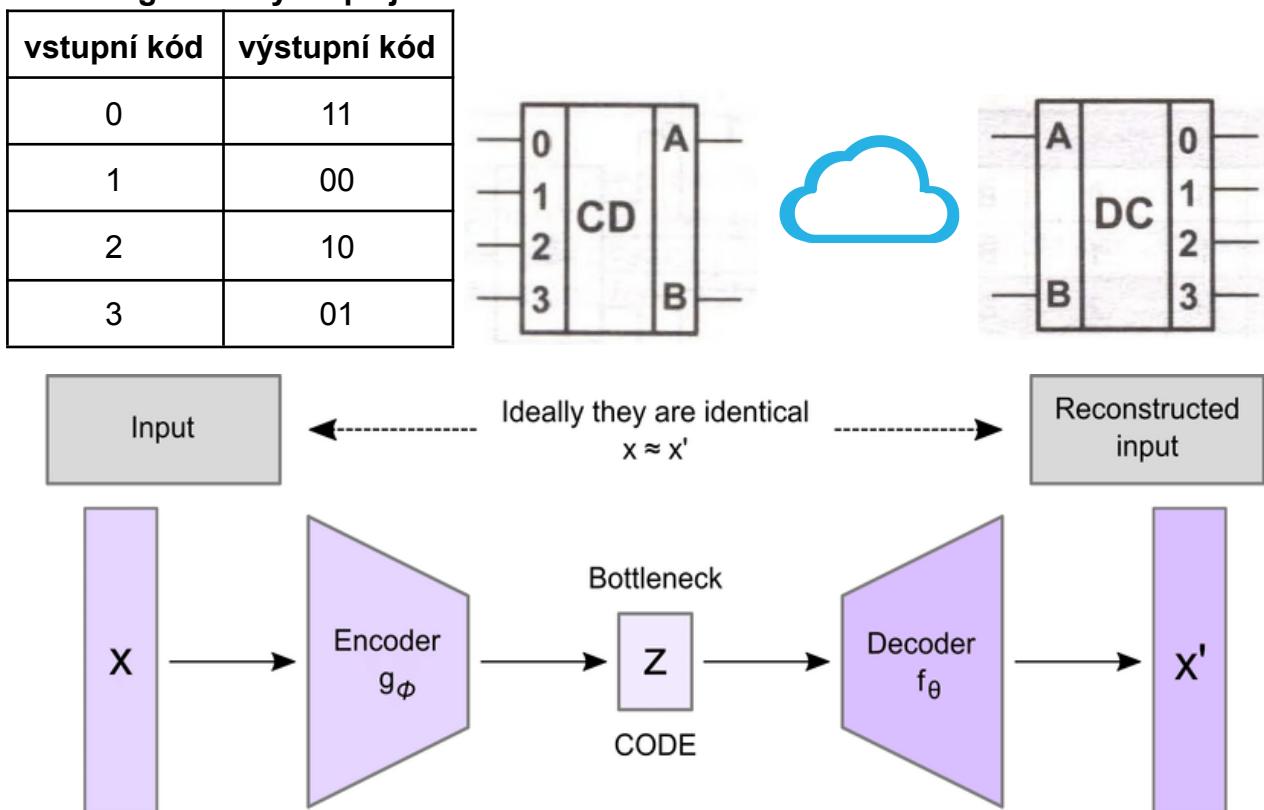


## Kodér a dekodér

**Kodér** kóduje (**kombinuje**, převádí) informaci z **N** vstupů na **K** výstupů na základě **kombinační tabulky**. Obecně platí, že **N > K** (jinak jde spíš o dekodér) a **2^K >= N** (jinak nelze zaručit jednoznačné kódování).

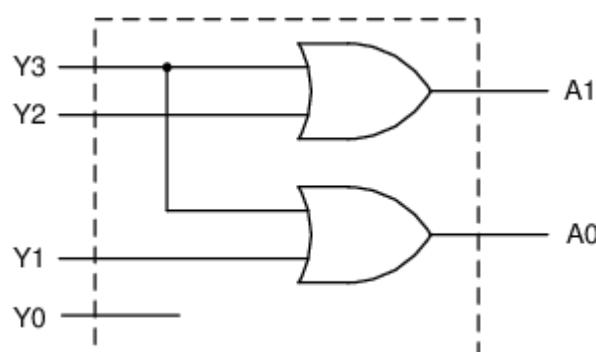
**Dekodér** je kombinační logický obvod **komplementární** ke kodéru. Dekóduje (**kombinuje**, převádí) informaci z **N** vstupů na **K** výstupů. Obecně platí, že **N < K**.

- To znamená, že zapojení kodéru a za ním dekodéru se **stejnou kombinační tabulkou**, která je **jednoznačná** (pro každý vstup generuje unikátní výstup), se bude chovat, jako by tam nebyl ani jeden. Může jít například o kodér ze **7-segmentového displeje na BCD** (nepoužívá se...) a dekodér z **BCD** na **7-segmentový displej**.



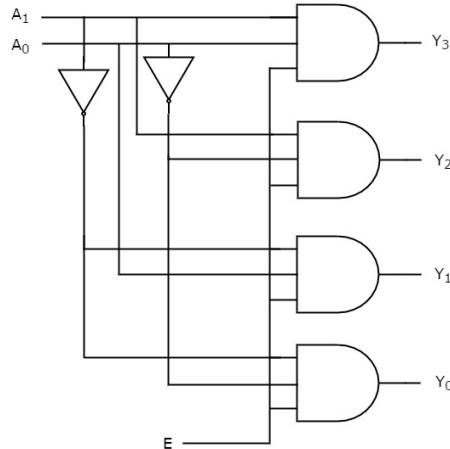
## Binární kodér (Encoder)

Umožňuje na **N** výstupů zakódovat až **2^N** vstupů. Musí ale platit, že vždy je **aktivní pouze jeden** ze vstupů. Pomocí kodéru lze například převádět stisknutou klávesu (vstup) na její binární hodnotu. Klávesa 0 je zapojena na první vstup, klávesa 1 na druhý atd. Na výstupu je poté 0b0000 pro klávesu 0, 0b0001 pro klávesu 1, 0b0010 pro klávesu 2, ...



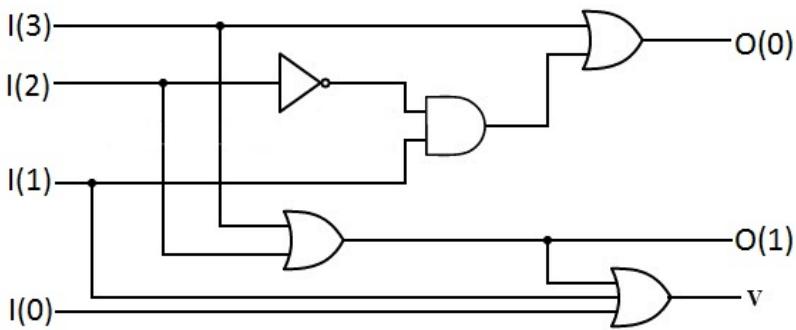
## Binární dekodér (Decoder)

Dekóduje **N vstupů** na  $2^N$  **výstupů**, narozdíl od kodéru mohou být vstupy libovolně aktivní. Pro daný vstup je ale aktivní vždy **pouze jeden výstup**. Dekodér lze například použít pro adresaci zařízení na sběrnici (binární adresa umožňuje aktivovat jedno z připojených zařízení). Dekodér lze implementovat pomocí demultiplexoru, vstup se zapojí na log. 1 a na **sel** se zapojí vstupy.



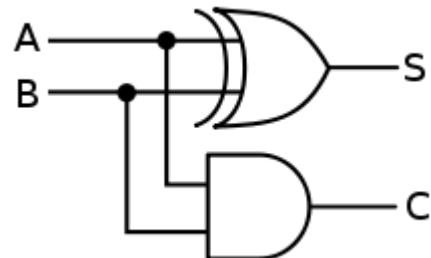
## Prioritní kodér

Narozdíl od běžného binárního kodéru umožňuje mít na **vstupu více aktivních vodičů**. V tom případě na výstup kóduje hodnotu odpovídající tomu s **největší prioritou**. Využívá se například k řízení obsluhy přerušení. Na obrázku má I(3) **největší** prioritu a I(0) nejmenší.



## Binární sčítačka

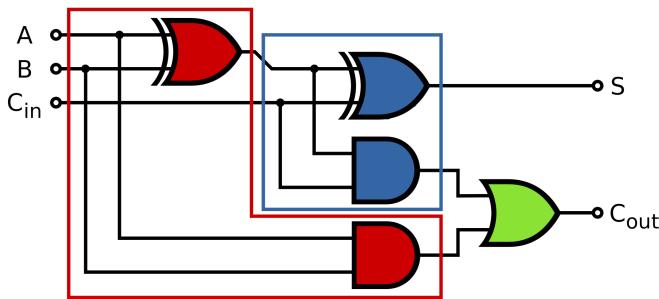
- **Poloviční sčítačka (half adder)** - realizuje sčítání dvou jednobitových čísel. Výstupem je jednobitový součet (**S**) a jednobitový příznak přenosu do vyššího řádu (**C**). Poloviční sčítačka ale sama **nedokáže zpracovat přenos z nižšího řádu**.



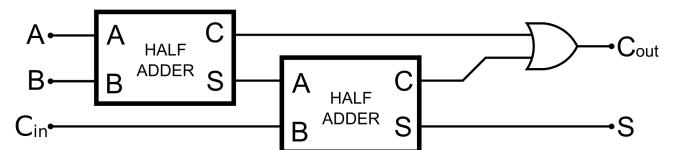
4 až 2 Prioritní Encoder

I <sub>3</sub>	I <sub>2</sub>	I <sub>1</sub>	I <sub>0</sub>	O <sub>1</sub>	O <sub>0</sub>	PROTI
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

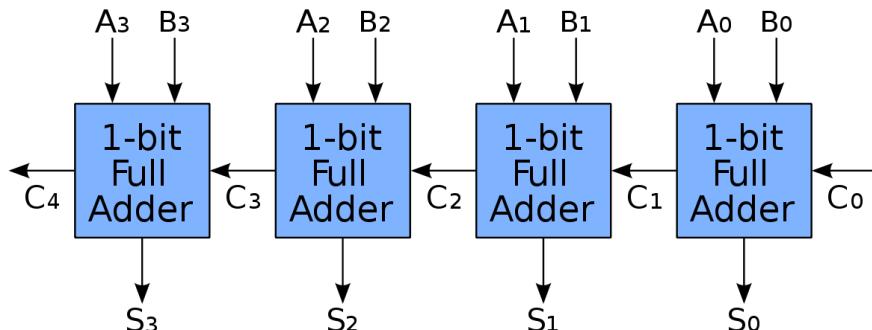
- **Úplná sčítačka (full adder)** - narozdíl od poloviční sčítačky umožňuje zpracovat i přenos z nižšího řádu. Vstupem jsou tedy sčítance **A**, **B** a přenos z nižšího řádu **C**. Výstupem je součet **S** a přenos do vyššího řádu **C**.



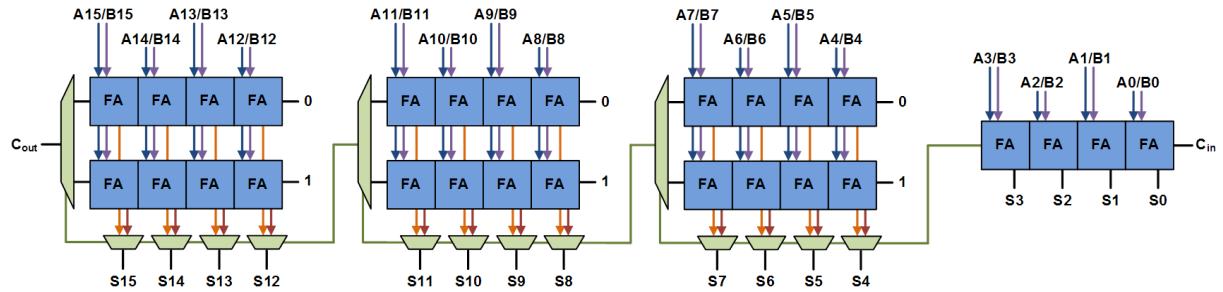
- **Sčítačka s přenosem (Ripple carry adder)** - Vzniká propojením více úplných sčítaček, tak aby se mohl šířit carry bit. **Pseudoparalelní**, přenos carry se **postupně šíří sčítačkou**. Poměrně pomalá kvůli čekání na carry bit. Zpoždění každého carry bitu je rovno zpoždění **3 log. členů**, zpoždění



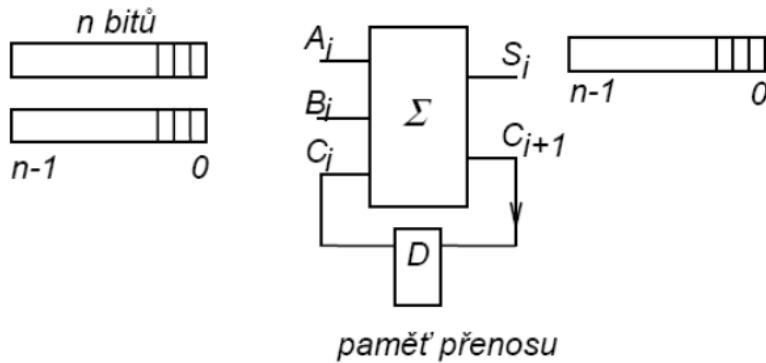
sečtení je rovno zpoždění **2 log. členů**. Z toho plyne, že zpoždění **S\_i = 3\*(i-1) + 2 = 3\*i - 1** (i-1 protože na 1. carry bit se nečeká)



- RCA lze urychlit **sčítačkou s výběrem přenosu (carry-select adder)**  
- předpočítáním si výsledků pro carry = 0 i carry = 1; 4 bitovými sčítačkami a zvolení těch správných pomocí multiplexoru na základě výsledků z předchozích sčítaček. Zpoždění této sčítačky je rovno zpoždění 4 bitové RCA sčítačky =  $4 \cdot 3 - 1 = 11$  log. členů a zpoždění 3 multiplexorů =  $3 \cdot 3 = 9$  log. členů. Celkově tedy **20 log. členů**.



- libovolně dlouhá binární čísla lze také sečíst jednou **úplnou binární sčítáčkou** za využití **posuvných registrů** (pro sčítance a součet; délka registrů určuje maximální délku binárního čísla) a **klopného obvodu typu D**, který je použit pro zapamatování si carry bitu (přenosu) **nejedná se kombinační obvod**.



- Sčítáčka s predikcí přenosu (CLA - Carry Lookahead Adder)** - Rychlejší jak **RCA**, výpočet opravdu probíhá paralelně, teoreticky lze sečíst dvě jakkoliv dlouhá bitová čísla s konstantním zpožděním  $S_i = 4 \text{ log. členů}$  (a se zpožděním carry bitu  $C_i = 3 \text{ log. členů}$ ). Reálně je délka sčítaných čísel omezena schopností implementovat **více vstupů hradla s ekvivalentním zpožděním jako dvojvstupů**, počtem hradel sčítáčky a s tím souvisejícím počtem spojů. Pro velká čísla přestává být tudíž praktická (náročná na výrobu, vysoká cena atd.) a je potřeba výpočet rozdělit.
  - 16-bit adder using 4-bit CLA
    - z tabulky vstupů do CLA lze odvodit carry bit  $C_{i+1} = A_i \text{ and } B_i \text{ or } (A_i \text{ xor } B_i) \text{ and } C_i = G_i \text{ or } P_i \text{ and } C_i$  ( $G$  - generate,  $P$  - propagate).

For the example provided, the logic for the generate ( $G$ ) and propagate ( $P$ ) values are given below.

$$C_1 = G_0 + P_0 \cdot C_0,$$

$$C_2 = G_1 + P_1 \cdot C_1,$$

$$C_3 = G_2 + P_2 \cdot C_2,$$

$$C_4 = G_3 + P_3 \cdot C_3.$$

Substituting  $C_1$  into  $C_2$ , then  $C_2$  into  $C_3$ , then  $C_3$  into  $C_4$  yields the following expanded equations:

$$C_1 = G_0 + P_0 \cdot C_0,$$

$$C_2 = G_1 + G_0 \cdot P_1 + C_0 \cdot P_0 \cdot P_1,$$

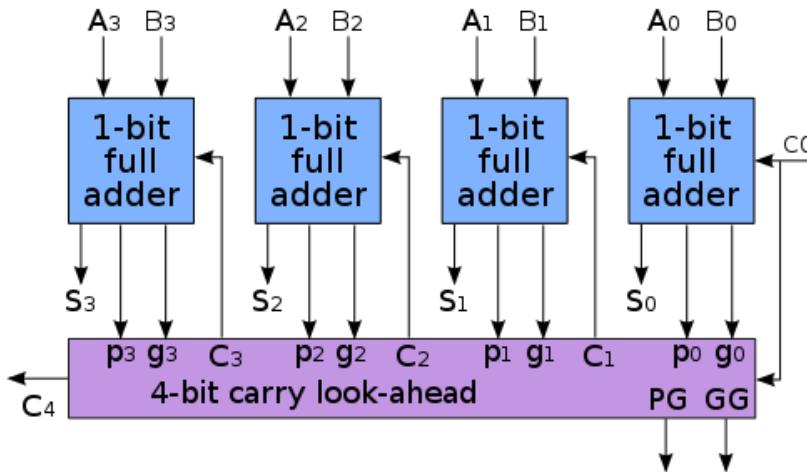
$$C_3 = G_2 + G_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 + C_0 \cdot P_0 \cdot P_1 \cdot P_2,$$

$$C_4 = G_3 + G_2 \cdot P_3 + G_1 \cdot P_2 \cdot P_3 + G_0 \cdot P_1 \cdot P_2 \cdot P_3 + C_0 \cdot P_0 \cdot P_1 \cdot P_2 \cdot P_3.$$

To determine whether a bit pair will generate a carry, the following logic works:

Z těchto rovnic lze učit, že výpočet  $C_i$  bude mít zpoždění **3 log. členů**, v případě, že máme více vstupů hradla (1. - A xor B, 2. P and C, 3. or všech vstupů).

- výpočet  $S_i$  bude probíhat jako  $((A_i \text{ xor } B_i) \text{ xor } C_i)$ , což znamená zpoždění dalšího log. členu a celkově tedy **4. log. členů**.



- **Vícebitové sčítačky s použitím 4 bitových (případně více) CLA sčítaček:**
  - zřetězení CLA sčítaček ve stylu RCA sčítačky, např. 16 bitová sčítačka pomocí 4 CLA sčítaček bude mít zpoždění  **$C_{16} = 4 * 3 = 12 \text{ log. členů}$**  a zpoždění  **$S_{15} = 3 * 3 + 4 = 13 \text{ log. členů}$**  (3x carry + poslední výpočet).
  - spojením více CLA sčítaček pomocí **LCU (lookahead carry unit)**, které dokáží opět s **konstantní rychlostí** spočítat přenosy mezi jednotlivými CLA sčítačkami.  **$C_4, C_8, C_{12}$  a  $C_{16}$**  jsou dle rovnice níže vypočteny se zpožděním **5 log. členů** a protože **G** a **P** již jsou vypočteny, zabere výpočet  **$S_{i4}$  až  $S_{i15}$**  už jen další **3 log. členy** a zpoždění tak bude celkem **8 log. členů**.

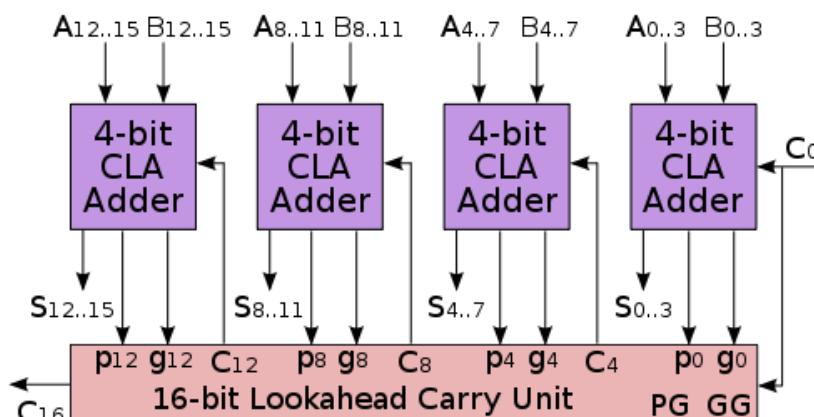
$$PG = P_0 \cdot P_1 \cdot P_2 \cdot P_3,$$

$$GG = G_3 + G_2 \cdot P_3 + G_1 \cdot P_3 \cdot P_2 + G_0 \cdot P_3 \cdot P_2 \cdot P_1.$$

They can then be used to create a carry-out for that particular 4-bit group:

$$CG = GG + PG \cdot C_{in}.$$

- obdobně jde vytvořit pomocí čtyř 16 bitových **LCU** sčítaček 64 bitovou sčítačku s dalším zanořením **LCU**.



# 3. Sekvenční logické obvody (klopné obvody, čítače, registry, stavové automaty – reprezentace a implementace)

**Sekvenční logický obvod (SO)** - Výstup obvodu závisí **nejen na vstupu**, ale také na **minulých vstupech**. Skládají se z **kombinační** části a **paměťové** části. Paměť v sekvenčním obvodu uchovává **vnitřní (současný) stav**.

- **Asynchronní SO** - Změna vstupní hodnoty hned ovlivní výstup. Reagují okamžitě.
- **Synchronní SO** - Řízený **hodinovým signálem**. Až při přechodu hodinového signálu se vstup projeví na výstupu. Reagují v periodických intervalech.
  - **Úrovňové SO** - SO sleduje vstupy po celou dobu hodinového signálu a průběžně na ně reaguje.
  - **Hranové (derivační) SO** - Reaguje na vstupy jen při přechodu hrany (náběžné nebo sestupné).

## Klopný obvod

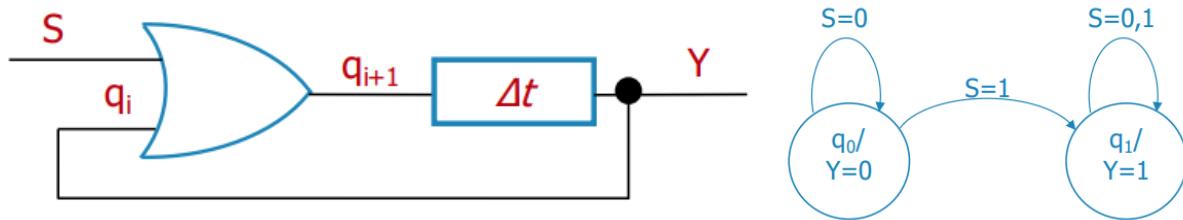
Klopné obvody jsou nejjednodušší sekvenční obvody. Jedná se o obvody, které **skokově** (neuvažujeme zpoždění změny úrovně hradla) přechází mezi **několika diskrétními stavy**.

Klopné obvodu dělíme na:

- **Astabilní** - nemají žádný stabilní stav - neustále kmitají (**osculují**).
  - např. pro generování obdélníkového signálu, hodinového signálu
- **Monostabilní** - mají **jeden stabilní stav**, jedná se o sekvenční obvod, který při spuštění **generuje puls**. Lze si jej představit jako misku a do ní hozený míček.
  - časovače
- **Bistabilní (flip-flop)** - nejpoužívanější typ klopných obvodů, má dva stabilní stavy, mezi kterými lze přepínat (**0 a 1**)
  - je základním stavebním prvkem rychlých **volatileních** pamětí (registry, paměti operandů, čítače, ...)
- **Schmittovy** - slouží k **úpravě tvaru impulzů**. Jeho základní vlastností je **hystereze**. Jeho výstup je závislý nejen na hodnotě vstupu, ale i na jeho **původním stavu**. Hystereze zabraňuje vzniku **zákmitů** výstupního signálu v okolí **střední** úrovně spínání.

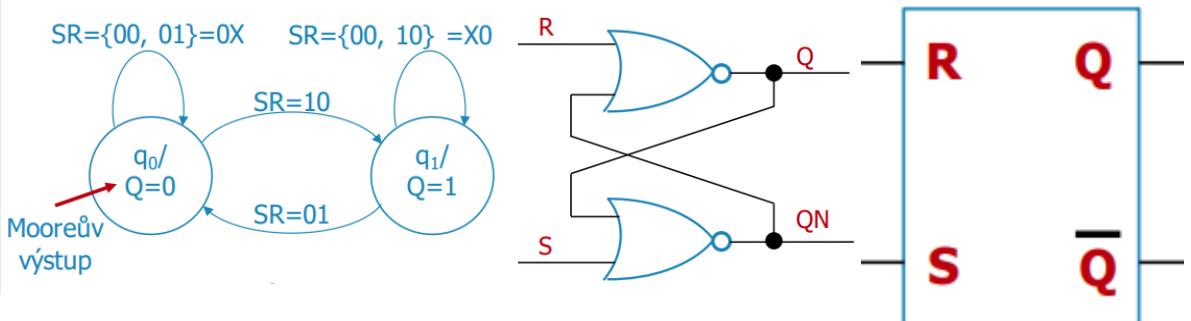
## Bistabilní klopné obvody

- **SET klopný obvod** - velmi jednoduchý, ale prakticky nepoužitelný. Nelze překlopit zpět ze stavu v **log. 1**. Produkuje **Moorův výstup**.

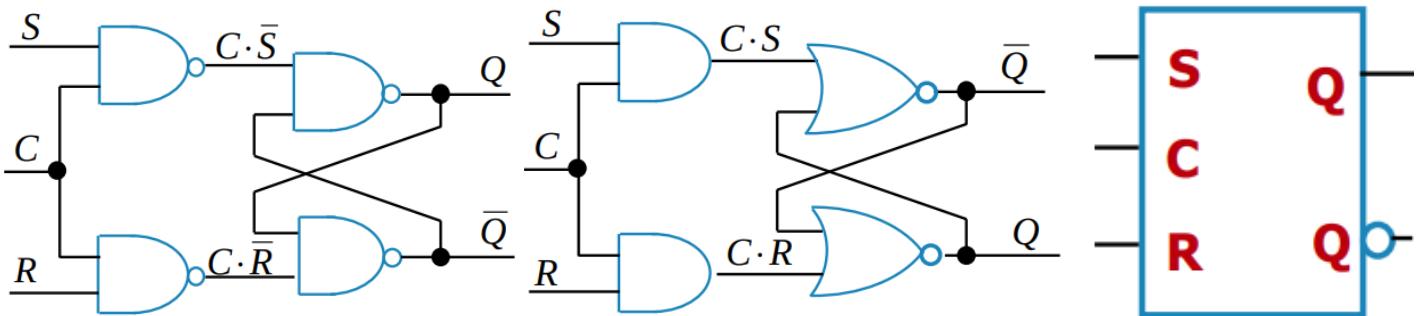


- **R-S klopný obvod (reset-set)**
  - Přivedení **log. 1** na vstup **R** (reset) nastaví na výstupu **log. 0**.
  - Přivedení **log. 1** na vstup **S** (set) nastaví na výstup **log. 1**.
  - Současné přivedení **log. 1** na **S** i **R** je **zakázaná** kombinace, dříve **nešlo určit** v této situaci hodnotu na výstupu, dnes u hradel **NOR** bude na obou **log. 0** (negovaný výstup bude stejný jako přímý). Lze tomu zabránit prioritou jednoho ze vstupů.

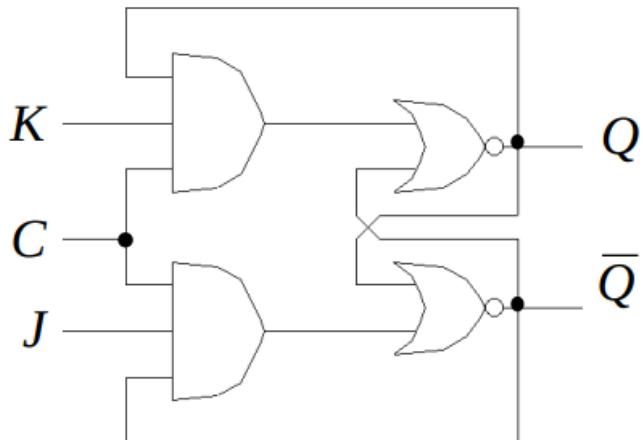
R	S	Q <sub>n+1</sub>
0	0	Q <sub>n</sub>
0	1	1
1	0	0
1	1	X



- **R-S s povolovacím vstupem** - Povolovací vstup **C** (Control, Enable), který povoluje činnost KO – obvod lze **nastavit** či **nulovat**, jen pokud je vstup **C** aktivní (aktivní může být v log. 1 i log. 0, záleží na návrhu).

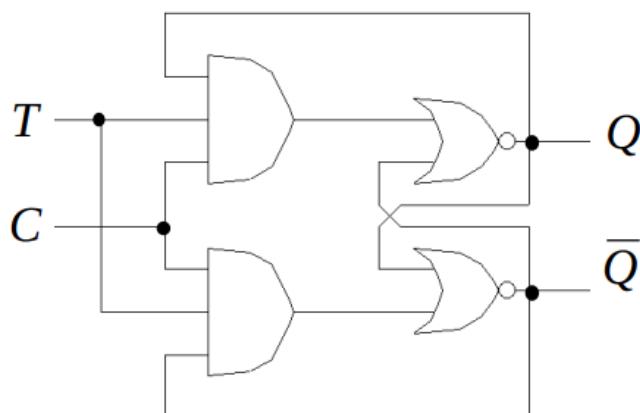


- **J-K klopný obvod (K-J: K odpovídá R, J odpovídá S)** - Zavádí zpětnou vazbu, která eliminuje zakázanou kombinaci. V případě, že jsou oba vstupy v **log. 1**, dochází ke **změně aktuální hodnoty na výstupu** (překlop - toggle).



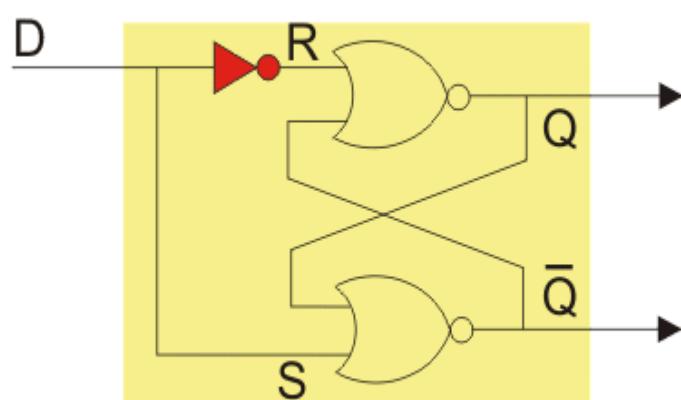
C	J	K	Q	$Q_{i+1}$	Činnost
0	X	X	0	0	Pamatuj (Hold)
0	X	X	1	1	
1	0	0	0	0	
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	1	Překlop (Toggle)
1	1	1	1	0	

- **T (toggle) klopný obvod** - Jedná se o J-K klopný obvod, který na oba vstupy posílá jednu hodnotu **T**. Pokud je **T** v **log. 0**, na výstupu zůstává aktuální hodnota, pokud je na vstupu **log. 1**, dochází k překlopení aktuální hodnoty na výstupu. Držení **log 1.** na vstupu **T** (a případně vstupu **C**) způsobí oscilování, změny log. 1 a log. 0.

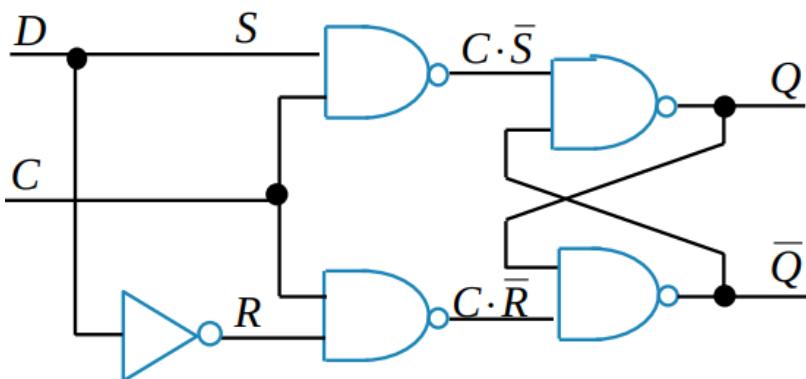


C	T	Q	$Q_{i+1}$	Činnost
0	0	0	0	Pamatuj (Hold)
0	1	1	1	
1	0	0	0	
1	0	1	1	
1	1	0	1	Překlop (Toggle)
1	1	1	0	

- **D (data) klopný obvod** - Jedná se R-S klopný obvod, který posílá na vstup **S** hodnotu **D** a na vstup **R** její negaci **not D**. Nemůže takto vzniknout nepovolený stav (**R** a **S** nikdy nebude současně v **log. 1**). Na výstup posílá hodnotu na vstupu (vytváří tedy zpoždění).



- **D klopný obvod s povolovacím vstupem** - Narození od jednoduchého **D** klopného obvodu, zde dochází ke **změně hodnoty** na výstupu, **jen** pokud je aktivní vstup **C** (pokud je C v **log. 0**, na výstupu setrvává nastavená hodnota).

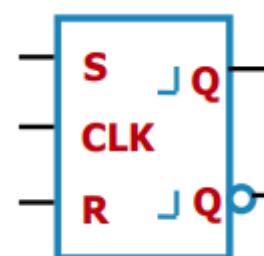
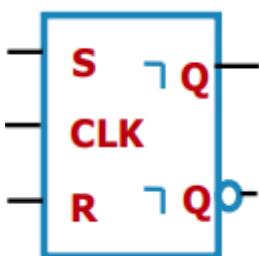


C	D	Q	$Q_{i+1}$	Činnost
0	X	0	0	Pamatuj (Hold)
0	X	1	1	
1	0	0	0	Ulož 0 (Store 0)
1	0	1	0	
1	1	0	1	Ulož 1 (Store 1)
1	1	1	1	

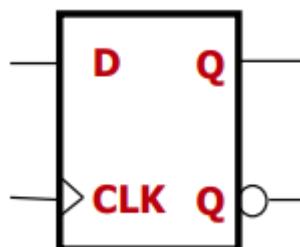
## Dvoufázové bistabilní klopné obvody

Využívají se v **synchronních obvodech**, které jsou řízeny hodinami **CLK**. Mohou být typu **Master-Slave** nebo **Derivační**.

- U **Master-Slave** dochází ke změně hodnoty na výstupu **buď** když je **CLK** v **log. 0, nebo v log. 1** ].
- Informace ze vstupů se při **CLK=0** zapisuje do **Master R-S KO** a následně se při **CLK=1** přepisuje do **Slave R-S KO**. Případně opačně.
- různé typy: **RS, JK, T, D**.



- **Derivační** mění hodnotu na výstupu při změně úrovně CLK, tj. buď s **nástupnou** hranou nebo se **sesupnou** hranou. (> nástupná, náběžná; < sesupná, doběžná)



## Čítače

Jedná se o speciální automaty, které reagují na vstupní impulsy (např. nástupná hrana) přechodem ze stavu do stavu (**přičítají** nebo **odečítají**).

- **Asynchronní čítač** - Neobsahuje synchronizační hodiny, čítání (přechod ze stavu do stavu) je řízeno pouze vstupem.
- **Synchronní čítač** - Je řízen hodinami, k přechodu ze stavu do stavu dochází např při náběžné hraně hodin.

Mohou mít řídící vstupy, které umožňují:

- **UP/DOWN** - specifikovat **směr čítání** (vzestupná/sestupná posloupnost stavů)
- **CE (clock enable)** - pro **CE=0 si pamatuje** aktuální stav, pro **CE=1 čítá**
- **P Enable** - čítač čítá pouze když je v **log. 1**, nemá vliv na **RCO**.
- **T Enable** - čítač čítá pouze když je v **log. 1**

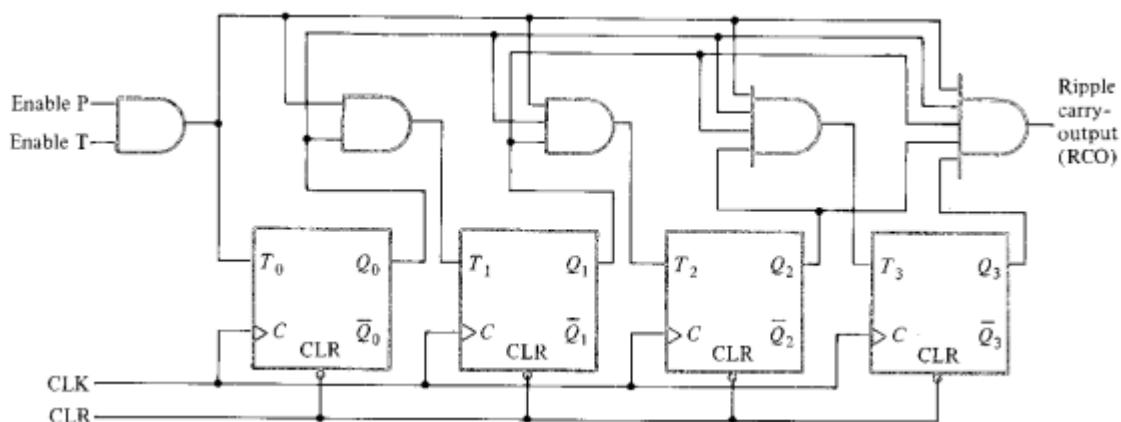
Mají výstup indikující přetečení (podtečení) čítače:

- **RCO** nebo **TC**

Prodloužení doby **čítání/dělícího poměru/vzniku přetečení** se implementuje kaskádovým zapojením čítačů. Čítače jsou zapojeny **vstupem CE** na **výstup RCO (TC)** předchozího čítače. (tři čítače MOD 10 za sebou přetékají jednou za  $10 \times 10 \times 10 = 1000$  impulsů)

## Čítače řešené T klopným obvodem

- 4 bitový čítač
- při přechodu z **15 do 0** dochází k přetečení (RCO v log. 1)
- aktuální stav čítače (binární hodnotu) lze získat z výstupů **Q0 až Q3**



## Čítač v Grayově kódu

- Grayův kód kóduje po sobě jdoucí čísla tak, že je mezi nimi vždy změna pouze v jediném bitu.
- redukce počtu operací při změně čísla o jedničku
- binární reprezentace (b) na Grayův kód (g):
  - MSB zůstává stejný

- pro každý bit:  $g_i = b_{i+1} \text{ xor } b_i$
- 1110 bin == 1001 Gray**

## Up/Down čítač

Postup: 3-Bit & 4-bit Up/Down Synchronous Counter

- určení tabulky ... Q2 Q1 Q0 (podle počtu bitů) aktuálního stavu - **present state logic**

$Q_c$	$\theta_B$	$\theta_A$
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

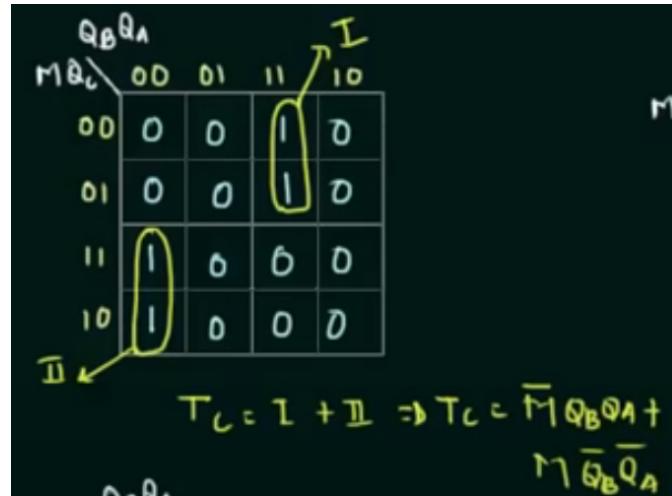
- určení tabulek následujícího stavu (Up - obrázek/Down) - **next state logic**

$Q_c$	$\theta_B$	$\theta_A$	$Q_c^+$	$Q_B^+$	$Q_A^+$
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

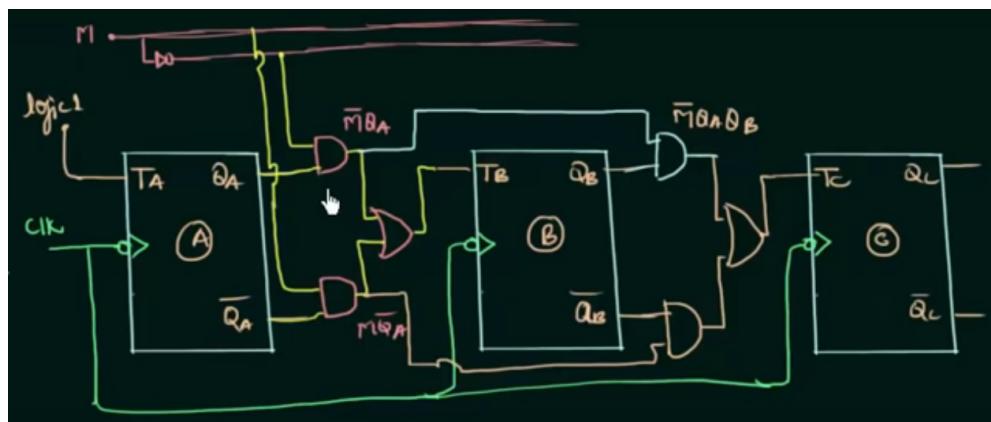
- určení log. úrovní vstupů klopných obvodů (budou se lišit u R-S, J-K, T - na obrázku, D) pro jednotlivé stavy

$Q_c$	$\theta_B$	$\theta_A$	$Q_c^+$	$Q_B^+$	$Q_A^+$	$T_c$	$T_B$	$T_A$
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	0	0	0	1
1	1	1	0	0	1	0	0	1

- zjednodušení logiky vstupů klopných obvodů pomocí **Karnaughových** map, pro každý KO jedna mapa

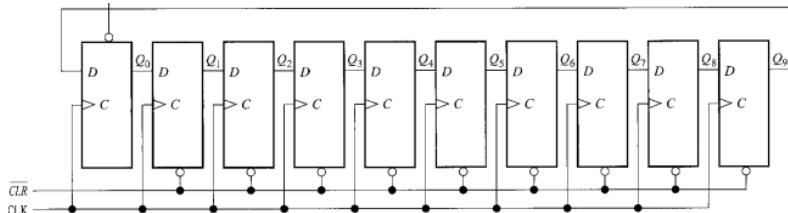


- implementace obvodu na základě určené logiky přechodů s použitím zvoleného klopného obvodu (na obrázku T)



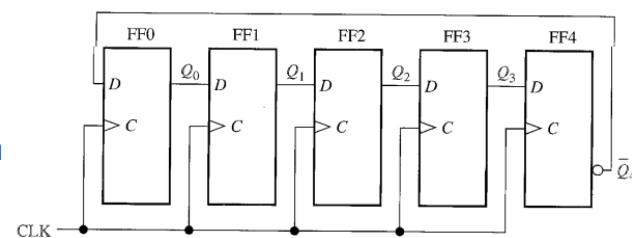
## Čítač one-hot (straight counter)

**V log. 1** je pouze jeden **D** klopný obvod, výstup kaskády je přiveden na vstup.



## Čítač v Johnsonově kódu

Opět čítá pomocí **D** klopných obvodů, **negovaný** výstup kaskády je přiveden na vstup.



## Registry

Umožňují ukládat informaci o určitém počtu bitů. Pro realizaci se obvykle používá klopný obvod typu **D**. KO jsou připojeny ke stejnemu zdroji hodinového signálu. Mohou mít asynchronní **CLR** pro nulování, povolení činnosti **CE**. Mají vektor vstupu např. **D31-D0** a vektor výstupu např. **Q31-Q0**, lze je zapisovat jednotlivě, ale obvykle se tak neděje. Reprezentují např. 32-bit číslo, které vstupuje do **ALU** jako **operand**. Dále se používají při konstrukci automatů pro uchování **vnitřního stavu**.

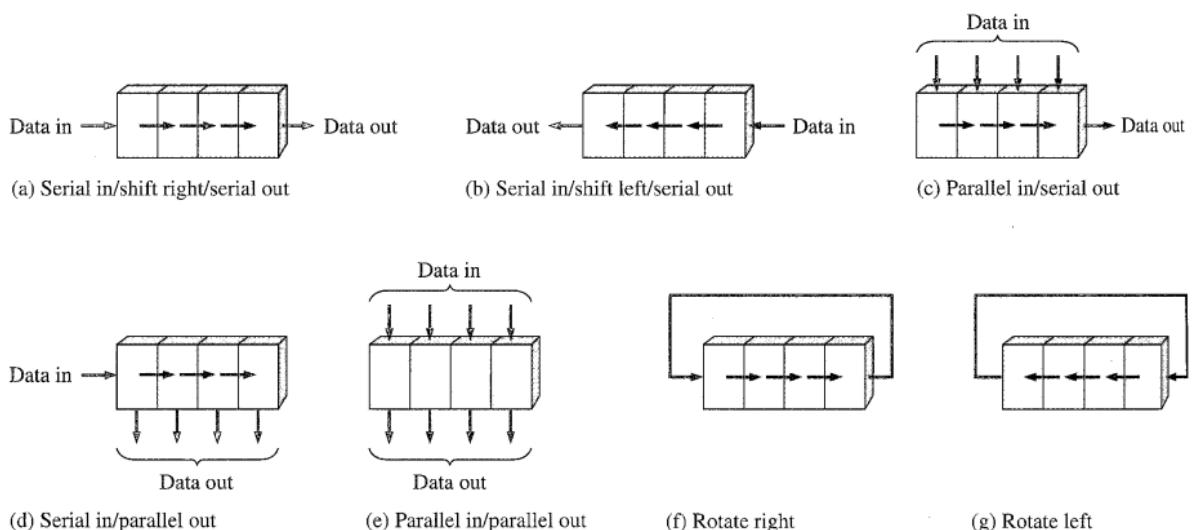
### Paměťový registr (latch)

Paměť pro několik bitů.

### Posuvný registr (shift)

Synchronní obvod (mají stejný CLK) sestaven z klopných obvodů zapojených do série. Posouvání uložených bitů od jeden bit vlevo nebo vpravo při každém hodinovém impulsu. Existuje mnoho druhů, lze je použít např:

- Master-Slave SPI** - MISO, MOSI,
- zápis dat pro odeslání přes **UART**,
- čtení obdržených dat z **UART**,
- ukládání hodnot operandů,
- sekvenční sčítačka**

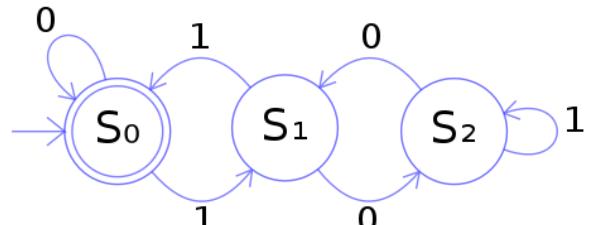


## Stavové automaty

**Stavový automat** - Konečný automat (KA) je šestice:

$M = (S, \Sigma, \Lambda, T, G, s_0)$ , kde:

- $S$  je konečná neprázdná množina stavů (vnitřní abeceda)
- $\Sigma$  je konečná vstupní abeceda
- $\Lambda$  je konečná výstupní abeceda
- $T$  je přechodová funkce ( $T : S \times \Sigma \rightarrow S$ )



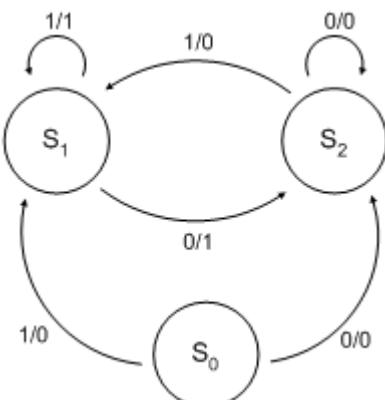
- $G$  je výstupní funkce ( $G : S \times \Sigma \rightarrow \Lambda$  pro Mealyho automat;  $G : S \rightarrow \Lambda$  pro Mooreův automat)
- $s \in S$  je počáteční stav

V elektronice ho tvoří 3 části:

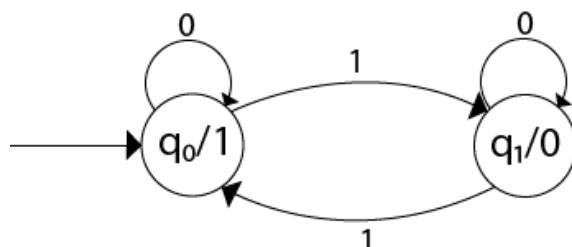
- **Next-state logika** (přechodová funkce) - Kombinační logická síť KLS. Na základě **současného stavu** (paměť) a **vstupu** generuje hodnotu následujícího stavu automatu.
- **Paměť** - Pro **zapamatování si současného stavu**.
- **Výstupy** - Mealy, Moore, kombinace.

Dle výstupní funkce je dělíme na:

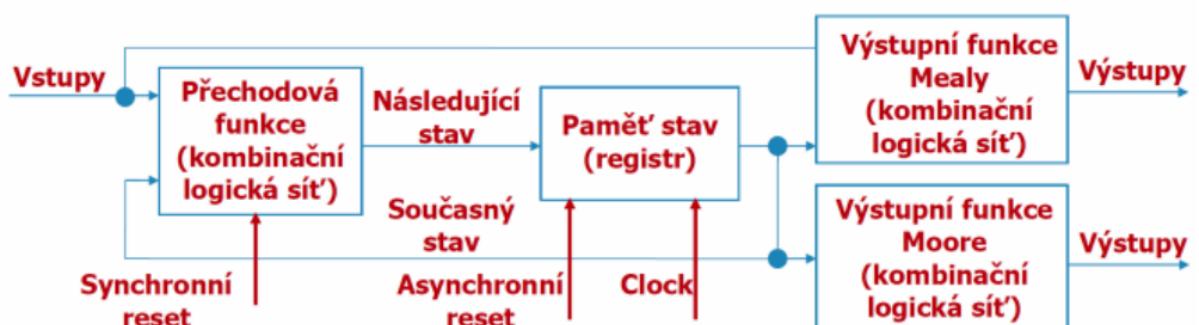
- **Mealyho automat** - Výstup je funkcí jak aktuálního stavu, tak vstupu  
 $Y = f(X, Z)$ .
  - značení přechodu: **vstup/výstup**



- **Mooreův automat** - Výstup je funkcií aktuálního stavu automatu -  $Y = f(Z)$ . Tedy změna na výstupu se projeví až v následujícím stavu.
  - značení stavů: **jméno stavu/výstup**



- **Moore/Mealy automat** - Kombinace mealyho a mooreova automatu.



**Odkazy:**

- SO: [SEKVENČNÍ LOGICKÉ OBVODY](#)
- Mealyho automat: [Mealyho automat](#)
- Mooreův automat: [Mooreův stroj](#)

# 4. Hierarchie paměti v počítači (typy a principy pamětí, princip lokality, organizace rychlé vyrovnávací paměti).

**Paměť** - Uchovává data, které je zde možné zapsat a později zase získat (přečíst).

## Dělení pamětí

**Hierarchie** - Hierarchie existuje z důvodu co nejlepšího **poměru cena/výkon**. Stejně tak je potřeba jak **volatilní**, tak **nevolutilní** paměti. Obecně platí, že čím **blíže** procesoru paměť je, tím je **rychlejší**, ale také **dražší** (náročnější na výrobu) a má **menší kapacitu**.

- **Primární/vnitřní**
  - uvnitř procesoru: **registry, cache** (RVP, obvykle několik úrovní),
  - mimo procesor: **RAM**
- **Sekundární/hlavní** (uvnitř PC) - **HDD, SSD**.
- **Terciární/vnější** (mimo PC) - **CD, DVD, Flash disk**.

**Fyzikální princip** - opět výhody a nevýhody cena/volatilita/rychlosť:

- polovodičové - bipolární, unipolární, unipolární s floating gate - **SSD**
- magnetické - **diskety, HDD, pásky**,
- optické - DVD, CD, Blue Ray,
- magnetoopické
- molekulární

## Stálosti obsahu:

- **Volatilní** - Po odpojení paměťové buňky od el. energie se data ztratí (RAM),
- **Nevolatilní** - Data setrvávají i po odpojení paměti z el. sítě.

## Doba uchování informace:

- **Statická** - Drží data libovolně dlouho (dokud jsou připojeny k el. energii).  
**Velká plocha na čipu** (klopné obvody), **rychlá, dražá, malá kapacita** - používá se pro registry a RVP
- **Dynamická** - Data je třeba po určité době (u DRAM v řádu milisekund) obnovovat, jinak se ztratí. **Pomalejší, levnější** (oproti statické), **menší plocha na čipu** (tvořena tranzistorem a kondenzátorem) - používá se pro DRAM paměti.

## Přístup k datům:

- **Libovolný** (RAM - Random Access Memory) - přístupová doba **nezávisí** na umístění paměťové položky.
- **Sériová** (SAM - Serial Access Memory) - přístupová doba **závisí** na umístění v paměti, respektive na aktuální pozici čtecí hlavy (pásky)

- **Smíšený** - kombinace výše popsaných přístupů. Např. HDD s více záznamovými povrchy. Výběr povrchu je libovolný, natočení hlavy je sekvenční.

#### Možnost přístupu/měnitelnost:

- **ROM (Read Only Memory)** - Pouze čtení
- **PROM (Programmable ROM)** - Lze **jednou** zapsat data a poté pouze číst
- **EPROM (Erasable PROM)** - Pro vymazání je potřeba použít externí proces, například pomocí **UV záření**.
- **EEPROM (Electrically EPROM)** - Lze **vymazat elektronicky**
- **RWM (Read Write Memory)** - Lze číst i zapsat do ní (SRAM, DRAM)

## Princip lokality

Aplikace pracuje obvykle pouze s **malou částí paměťového prostoru** (data programu nebo samotný kód programu).

- **Časová lokalita** - Pokud procesor používá nějakou položku v paměti, je vysoká pravděpodobnost, že ji bude používat **znovu**. To znamená je vhodné položku uložit, co nejblíže k procesoru.
- **Prostorová lokalita** - Pokud procesor pracuje s nějakou položkou v paměti, potom položky, které jsou umístěny v paměti v **blízkosti této položky**, budu s vysokou pravděpodobností **také použity** (aspoň u dobré napsaného kódu). To znamená, že je vhodné okolní položky uložit co nejblíže procesoru, aby toto očekávané čtení/zápis mohlo proběhnout rychleji.

## Parametry pamětí

- **Kapacita** - Udává množství dat, která paměť dokáže uložit. Udává se ve tvaru **respektujícím organizaci paměti**, tedy jako součin počtu paměťových míst a délky paměťového místa, tj. **N x n** bitů, např. **16K x 1 bit**
- **Přístupová doba** - doba od **zahájení čtení** (tj. udáním adresy paměťového místa a povetu R) po **získání obsahu** paměťového místa.
- **Přenosová rychlosť** - Počet **bitů/bytů**, které paměť přenese za **jednotku času**, např. 1 Gb/s.
- **Chybovost** - Počet chyb za určitý čas.
- **Poruchovost** - Střední doba mezi poruchami (MTBF - mean time between failure).

## Rychlá vyrovnávací paměť (RVP)

Anglicky **cache** - SRAM blízko procesoru, pomáhá rychlejšímu čtení/zápisu dat, než jaké by bylo použitím pouze operační paměti (RAM - pomalejší DRAM). V procesorech často rozdělena do několika vrstev - L1, L2 pro každé jádro a L3

sdílená mezi jádry. Hierarchické členění pamětí se snaží eliminovat rozdíl mezi rychlostí procesoru a rychlostí operačních pamětí.

Čtení/zápis	Počet cyklů
Registr	1 (i méně při pipelining)
RVP L1	cca 3
RVP L2	cca 14
Hlavní paměť (RAM)	cca 240

RVP je rozdělena do **bloků** o konstantní velikosti, která ideálně odpovídá velikosti bloků v hlavní paměti (RAM), což umožňuje jednodušší přesun dat (po blocích) z RAM do RVP. V RAM je ale paměťových bloků **řádově více** než v RVP, takže musí existovat mechanismy zajišťující, aby v RVP byly **potřebné bloky**. Tyto mechanismy vybírají bloky, které budou do RVP nově přidány a bloky, které v důsledku budou odstraněny. Musí pracovat s velmi **vysokou pravděpodobností úspěchu**, jinak by paměťová hierarchie zpomalovala přístup k datům.

- **Hit Rate** - udává pravděpodobnost nalezení dat v RVP, v praxi **95-99%**.
- **Miss Rate** ( $1 - \text{Miss Rate}$ ) - pravděpodobnost, že data **nejsou** v cache nalezena. V tomto případě je nutné potřebný blok s daty načíst z hlavní paměti, což obnáší uvolnění místa v RVP, vyhledání bloku v RAM a přenos dat. Tato doba se označuje jako ztrátová doba (**miss penalty**).
- **Přístupová doba** - doba potřebná k nalezení bloku v RVP, blok dat musí být v RVP.
- **Ztrátová doba** - doba, za kterou se musí popřípadě uvolnit místo v cache, nalézt data v hlavní paměti a nahrát je do cache.
- Doba čtení:
  - **Hit** = přístupová doba
  - **Miss** = přístupová doba + ztrátová doba (tato doba je delší, než v případě **nepoužití** RVP, musí k ní docházet minimálně)

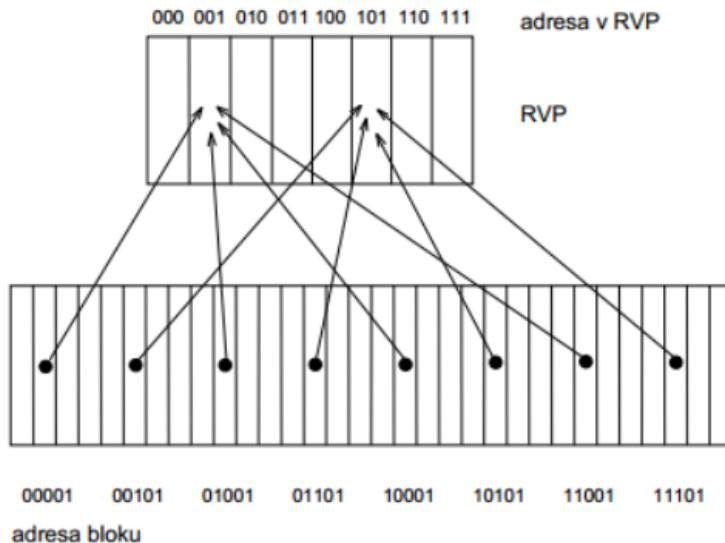
## Organizace RVP

Je dána zvoleným mapováním. Účel organizace RVP je především co nejvíce snížit **Miss Rate**. Organizace určuje jak jsou bloky z hlavní paměti přiřazovány/mapovány do RVP.

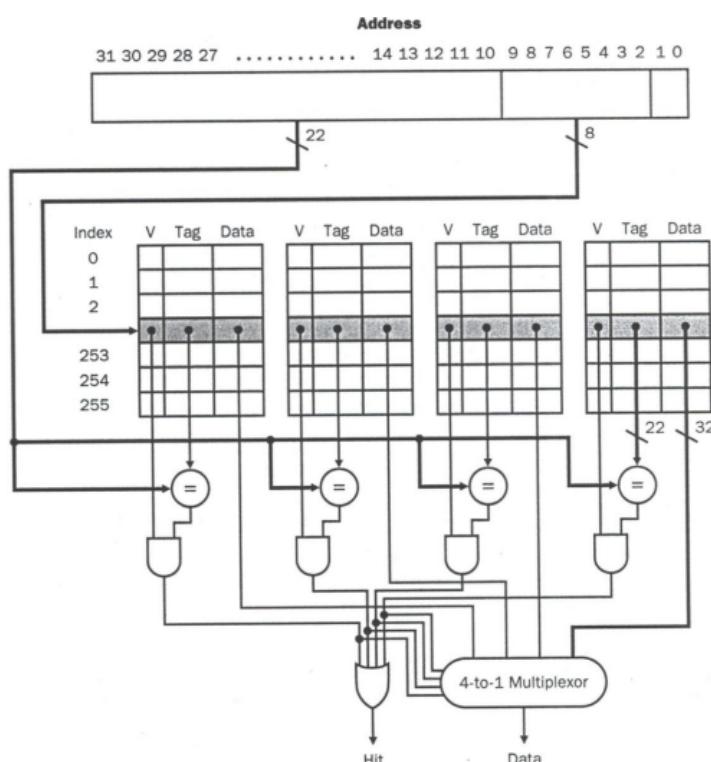
- **Přímé mapování** - Do RVP se mapují bloky na základě adres. Protože adresový prostor RVP je několikanásobně **menší**, než adresový prostor hlavní paměti, **maskují** se u adres hlavní paměti **horní bity**, tak aby délka adresy odpovídala délce adresy v RVP. To znamená, že bloky s adresou se stejnými spodními se mapují na jeden blok v RVP a **nemohou** tam tak být umístěny **současně**. Jedná se o jednoduchý koncept, který ale může

způsobovat velký počet výpadků (procesor současně pracuje s dvěma bloky, které mají stejnou adresu v RVP).

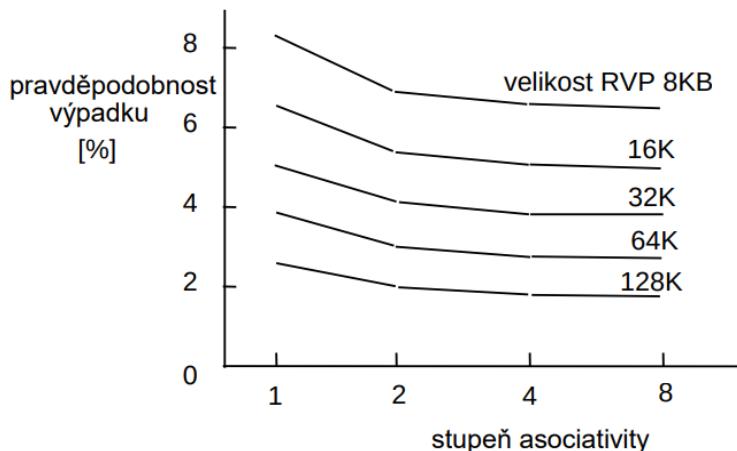
- např. pro RVP s  $2^{10}$  bloky a pamětí s  $2^{30}$  bloky bude pravděpodobnost při náhodném výběru bloku  $2^{10}/2^{30} = 1/2^{20}$  zaokrouhleně 1/1 000 000, tedy **0.0001%**.
- díky **časové a prostorové lokalitě** není ale přístup do paměti náhodný, proto je v praxi pravděpodobnost **Hit** tohoto řešení výrazně vyšší, okolo **90%**.



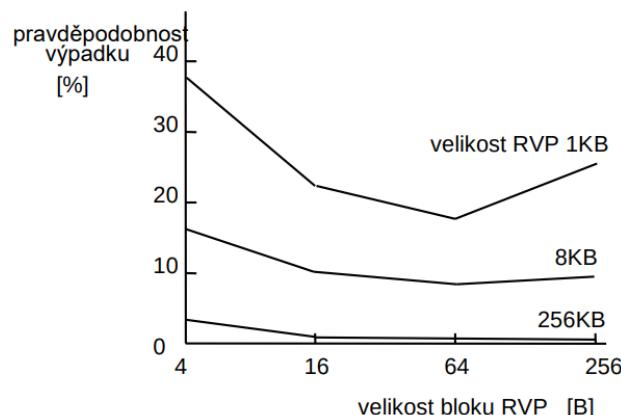
- **Více cestné mapování** - umožňuje do RVP uložit současně více bloků, které mají stejné spodní byty adresy (ukazatel). Adresový prostor RVP je tak rozdělen do více tabulek (2 pro dvoucestné, 4 pro čtyřcestné, ...), u záznamů v těchto tabulkách navíc musí být uložena zbylá část (maskovaná) adresy - **příznak**, aby se mohl vybrat ten správný záznam pro čtení/zápis. To znamená, že více cestná RVP o stejné **kapacitě dat** jako jednocestná, bude vyžadovat větší celkovou kapacitu a samozřejmě logiku (**komparátor**), která bude provádět **výběr správného záznamu**.



- **Plně asociativní mapování** - K této úrovni lze dospět zvyšováním stupně asociativity, až je tento stupeň roven počtu ukládaných záznamů v RVP (tj. nepoužívá se už adresa bloků RVP). Poté může být libovolný záznam uložen kdekoli v RVP a s ním **celá** jeho adresa (do RAM) jako příznak. Tento způsob uložení ale implikuje to, že musí být zajištěno **souběžné porovnání** adres všech uložených bloků v RVP, aby bylo možné **rychle** najít ten požadovaný. To je ale technologicky náročné (pokud vůbec možné pro velké počty bloků), drahé a v praxi **nepoužitelné**.



Pravděpodobnost výpadku lze samozřejmě nejlépe snížit **zvýšením kapacity RVP**, ale také lze snížit určením vhodné **velikosti bloku**.



U vícecestných RVP se musí také řešit problém výběru oběti, pokud je RVP pro daný ukazatel (adresu) plná (všechny bloky jsou označeny jako **validní**). Používají se metody jako náhodný výběr, LRU, MRU, FIFO atd. Obecně tyto metody vyžadují další logiku (obvody).

- **Skupinově asociativní mapování** - Kompromis mezi přímým a plně asociativním mapováním. Paměť je rozdělena do skupin, kde každá skupina obsahuje stejný počet bloků. Adresa skupiny je vyhledávána přímo, blok v ní pak asociativně.
- **Sektorové mapování** - Hlavní paměť je rozdělena do sektorů, které mají několik bloků. Cache je také rozdělena do sektorů o několika blocích. Sektor hlavní paměti může být uložen do libovolného sektoru v cache, ale bloky musí být v sektoru shodné. Při přenosu bloku do cache jsou tedy staré bloky označeny za neplatné (příznakovým bitem) a je tam blok vložen.

## Konzistence dat v RVP

Při zápisu dat do bloku RVP (nejblíž CPU), ztratí bloku nižších RVP a hlavní paměti platnost a **nesmí** se již použít, musí se nějak označit. Problém je zejména zjevný u více jádrových procesorů (např. každé jádro má vlastní L1 a L2 cache, sdílí L3).

Zápis do L1 jednoho jádra musí případně nevalidovat blok v L1 druhého jádra, které si jej musí znova načíst (problém souběžného přístupu řeší programátor).

Metody zaručení konzistence:

- **Write-Through** (přímý zápis) - Při přímém zápisu do RVP se zapisuje okamžitě i do bloku v hlavní paměti. Tento princip je jednoduchý na realizaci, ale při častém zápisu je pomalý (musí se často čekat na pomalou RAM). Lze poté rozdělit:
  - **Write-Through with Write Allocate** - Write-through, pokud ale nejsou zapisovaná data v cache dojde k jejich načtení.
  - **Write-Through with no Write Allocate** - Nedoje k načtení zapisovaných dat do cache.
- **Write-Back** (zpětný zápis) - K úpravě (zapsání) bloku do nižší úrovně paměti (hlavní paměti) dochází až když je blok z RVP **odstraněn** (odsunut). Neefektivní protože k zápisu dochází **vždy**, i když nedošlo ke změně (musí se čekat). Proto se bloky označují příznakem změny (**dirty bit**) a zápis pak probíhá následovně:
  - **Flagged Write Back** - Zápis do paměti se uskuteční až při uvolnění bloku z cache, ale **pouze** u těch bloků, které byly modifikovány (mají **nastavený dirty bit**).
  - **Flagged Register Write Back** - Opět jsou ukládány pouze modifikované bloky, ale ne přímo, jsou prvně zapsány do pomocného rychlého bloku - zápis je odložen a je možné tedy číst bez čekání. **Nejrychlejší**, ale **nejnáročnější** (nejdražší) způsob.
- **Write-Buffer** (zápis s mezipamětí) - rozšiřuje metodu **Write-Back** o to, že při vybrání oběti (bloku, který bude odsunut z RVP s nastaveným dirty item), se data bloku přesunou do mezipaměti pro zápis, ze které budou zapsány, až nebudou požadavky na čtení z hlavní paměti. Nemusí se tak čekat na zápis dat, která jsou odsouvána z RVP, předtím, než budou nahrazena novým blokem.

# 5. Vestavěné systémy (mikrokontrolér, periferie, rozhraní, převodníky).

Evropské strategické iniciativy definuje **embedded systems** jako kombinaci hardwaru a softwaru, jejímž smyslem je řídit externí proces, zařízení nebo systém.

**Vestavěný (Vestavný/Embedded) systém** je jednoúčelový počítač, který je zabudován do zařízení, které ovládá. Na rozdíl od univerzálních počítačů jsou vestavěné systémy **specializované** (nikoliv součástkami) a mají **předem určenou činnost**, kterou řídí. Např. výdej nápojů v automatu, volba pracího režimu a samotná činnost praní v pračce, automobil tesla atd. Snaží se o:

- **Autonomie** - Funkčnost bez lidského zásahu.
- **Reaktivnost** - Odezva na podnět v reálném čase.
- **Kritičnost** - Vliv odchylek na normální chování na bezpečné plnění úlohy.

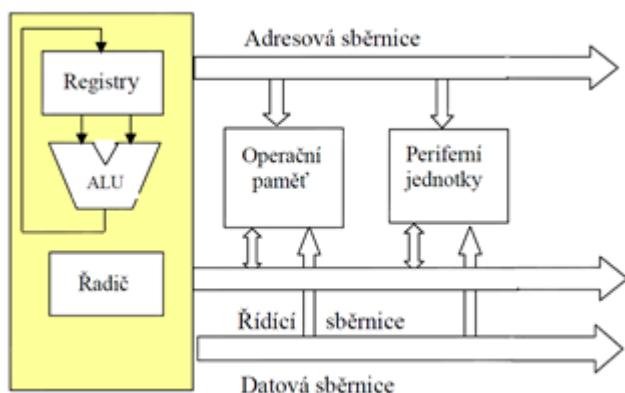
## Vlastnosti vestavných systémů:

- Omezená množina aplikačních systémů (vykonává pouze jeden úkol).
  - Často pouze jeden program na celý život.
- Často zpracovává fyzikálních veličin.
- Měly by být spolehlivé, bezpečné, zabezpečené a efektivní.
- Hlavní interakce nemusí být s člověkem.
- Ideálně by člověk neměl ani přemýšlet nad tím, že pracuje s počítačem.

I když je většinou funkce vestavěného systému **specializovaná**, nechceme pro každý systém vyvíjet **nový** integrovaný obvod např. s automatem, který by popisoval jeho činnost. Bylo by to časově a hlavně **finančně** náročné, navíc by poté nešlo provádět případné **změny** (i když v některých extrémních případech ano - optimalizace). Proto je dnes základem téměř každého vestavěného systému **mikrokontrolér** a na něj jsou napojeny dle potřeby periferní zařízení (senzory, display, reproduktor, ...), které jsou opět vyráběny ve velkých počtech. Samozřejmě existuje nespočet mikrokontrolérů, které se liší především cenou, výkonem, spotřebou, pamětí, a je na návrháři vestavěného systému, aby vybral pro účel navrhovaného systému vhodný. Stejně tak to platí o různá periferní zařízení.

## Mikrokontrolér (MCU)

Integrovaný obvod implementující kompletní počítač (ALU, paměť, síťové rozhraní, ..), který lze naprogramovat např. v jazyce C (nemusí být moc výkonný). Navíc od běžných procesorů často mikrokontroléry obsahují např. AD převodník, DA



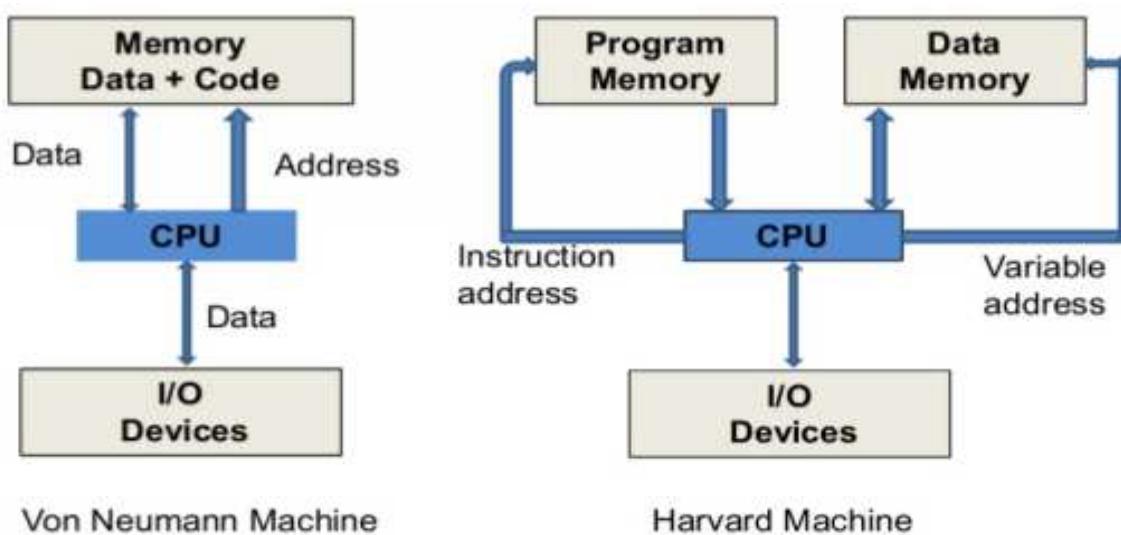
převodník, programovatelné časovače atd. Jsou hlavní komponentou vestavných systémů. Dle architektury je dělíme na:

### Von Neumannova architektura MCU

- **Společná paměť pro data i program,**
- **flexibilnější** pro měnící se aplikace,
- umožňuje **samomodifikující** se kód,
- instrukce (program) i operandy (data) se musí z paměti vybírat **postupně**, sběrnice je úzkým hrdlem. Je **pomalejší** než Harvardská.
- Jednodušší implementace, není třeba rozlišovat čtení dat od čtení programu.

### Harvardská architektura MCU

- **Paměť programu a dat jsou odděleny** (dvojí adresový prostor a dvě paměti),
- kód může být uložen v paměti jiného typu (FLASH) než data (RAM),
- Umožňuje použít **rozdílnou velikost buňky** paměti – např. 8 bitů (1 byte) pro data a 18 bitů pro instrukci (podle délky instrukce, tak aby jedno čtení načetlo vždy celou instrukci),
- Umožňuje v jednom taktu získat instrukci a **zároveň** její operandy - rychlejší než Von Neumann.



### Instrukční sada CISC (Complex Instruction Set Computing)

- **Složitá instrukční sada.**
- Mnoho typů instrukcí s mnoha variantami a mnoha adresovacími režimy (jedna instrukce může kombinovat více elementárních operací)
- Typické memory-to-memory instrukce (výběr operandů a uložení výsledků je součástí instrukce).
- instrukce nemusí mít stejnou délku - náročnější na dekódování
- provedení instrukce trvá více taktů
- nepoužívá mnoho registrů, někdy pouze **střadač**.

- Na čipu dominuje logika pro implementaci instrukcí.
- procesory Intel.

## Instrukční sada RISC (Reduced Instruction Set Computer)

- **Redukovaná instrukční sada.**
- Oproti CISC malé množství instrukcí, které provádí pouze **elementární** operace, časté instrukce jsou LOAD a STORE
- všechny instrukce jsou stejně dlouhé (stejný počet bitů v paměti) a typicky trvají **jeden** takt
- využívá velké množství registrů
- na čipu dominují registry (paměť) oproti obvodům pro vykonávání instrukcí
- procesory s touto architekturou mají menší spotřebu, dnes začínají převládat
- Apple Silicon

## Struktura MCU

- **Procesor**
- **Operační paměť** - RAM
- **Paměť programu** - EEPROM, PROM, ROM, FLASH, ...
- **Oscilátor** - Zdroj hodinového signálu (RC/Krystal)
- **Vstupně výstupní rozhraní (GPIO - general purpose input output)** - AD/DA převodník, paralelní/sériové porty, Ethernet... Obvykle jeden port lze využít k více účelům, aby nemělo MCU zbytečně velký počet výstupů.

## Periferie MCU

- **Časovač** - Měření času, např. RTC (real time clock), **TODO**
- **Hodiny** - přesnější zdroj hodinového signálu (**krystal**)
- **Čítač** - Viz otázka 3
- **Watchdog** - Stará se o to aby nedošlo k "zaseknutí" MCU při chybě. Pokud mu nepřijde pravidelné upozornění, tak resetuje celý MCU.
- **FPGA** - Programovatelné hradlové pole
- **Řadič přerušení** - klávesnice
- **senzory**

## Sériové rozhraní

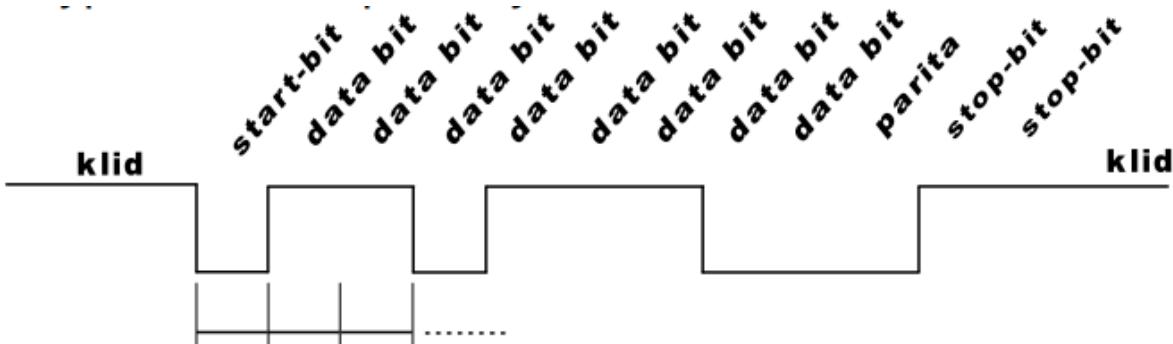
Slouží pro komunikaci s MCU. Komunikace probíhá po jednotlivých bitech (**sekvenčně**). Bity jsou přenášeny po jediném vodič jeden za druhým. Je třeba jednoznačně určit, v kterém okamžiku je na datovém vodiči hodnota kterého bitu.

- **Synchronní přenos:** Spolu s daty je přenášen i hodinový signál, který určuje, kdy lze číst další bit. (nutné jsou minimálně 2 vodiče)
- **Asynchronní přenos:** hodinový signál není přenášen, přijímač i vysílač si jej generují sami (musí mít smluvný **baud rate**). Je nutné zajistit dostatečnou přesnost hodinového signálu a jeho synchronizaci.
  - baud rate (Bd)

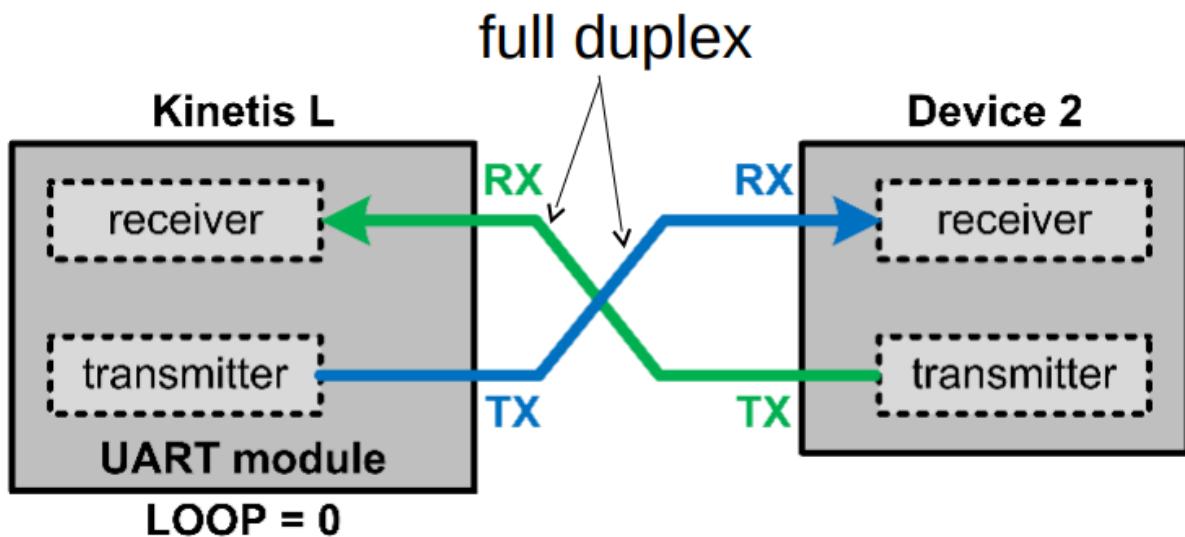
- počet symbolů (bitů, bytů, případně jiných přenášených prvků) přenesených za sekundu
- u sériové komunikace například 1 baud = 1 bit/s

## UART (Universal asynchronous receiver-transmitter)

Zařízení umožňující asynchronní komunikaci. Synchronizace je řešena pomocí **start bitu**, který je na začátku každého přenášeného rámce. Jedná se o přechod z klidové úrovni **log. 1** do **log. 0**. Přijímač podle něj **synchronizuje** hodiny, **za** tímto bitem již následují bity datové (8 bitů). Po datových bytech může následovat **paritní bit** a **jeden** nebo **dva stop bity** (vrátí linku opět do klidové úrovni **log. 1**). Přenos je tedy **start bit - LSB - ... - MSB - (paritní bit) - stop bit - (stop bit)**. Příklad na obrázku přenáší hodnotu **0x3B = 0b00111011**.



Komunikace u UART je **většinou plně duplexní (full-duplex)**, může být ale i **half-duplex** nebo pouze **simplex** (pokud přijímající stanice neobsahuje vysílač).



## SPI (Serial Peripheral Interface)

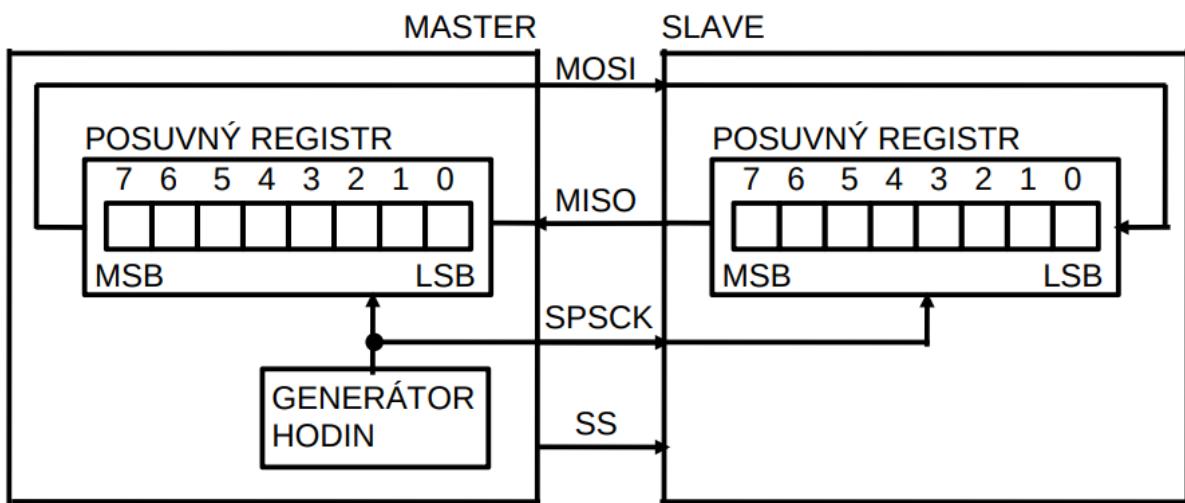
Jedná se **synchrone sériové** rozhraní typu **Master-Slave**, které je **plně duplexní**. V každém okamžiku **vždy** probíhá přenos **oběma** směry. Levná varianta pro připojování periferních zařízení, lze pomocí SPI realizovat ale i sběrnici.

- **Master:** na SPI sběrnici je vždy **jediný**. Je **zdrojem** hodinového signálu, **iniciuje** a **řídí** komunikaci. Obvykle se jedná o **MCU**.

- **Slave**: ke sběrnici může být připojeno **více** zařízení typu slave, **Master určuje** se kterým v daný okamžik komunikuje.

Vodiče, po kterých probíhá přenos, nebo jej řídí, jsou následující:

- **MISO** (Master In Slave Out) - vodič, po kterém jsou přenášena data od **Slave (zapisuje)** k **Master (čte)**.
- **MOSI** (Master Out Slave In) - vodič, po kterém jsou přenášena data of **Master (zapisuje)** k **Slave (čte)**
- **SPSCK** (SPI Serial Clock) - vodič po kterém je přenášen hodinový signál. Na zařízení Master je to vždy **výstup** a na zařízení Slave **vstup**.
- **SS** (Slave Select) - vodič, který vybírá Slave pro komunikaci. Na zařízení Master je **výstupem**. V **log. 1** je neaktivní (stejně jako UART), v **log. 0** je **aktivní**. V log. 0 musí být po **celou** dobu přenosu. V daném okamžiku může Master takto aktivovat **maximálně jeden** modul **Slave**, se kterým chce komunikovat. Zařízení, které mají SS v **log. 1** nezasahují do probíhající komunikace.



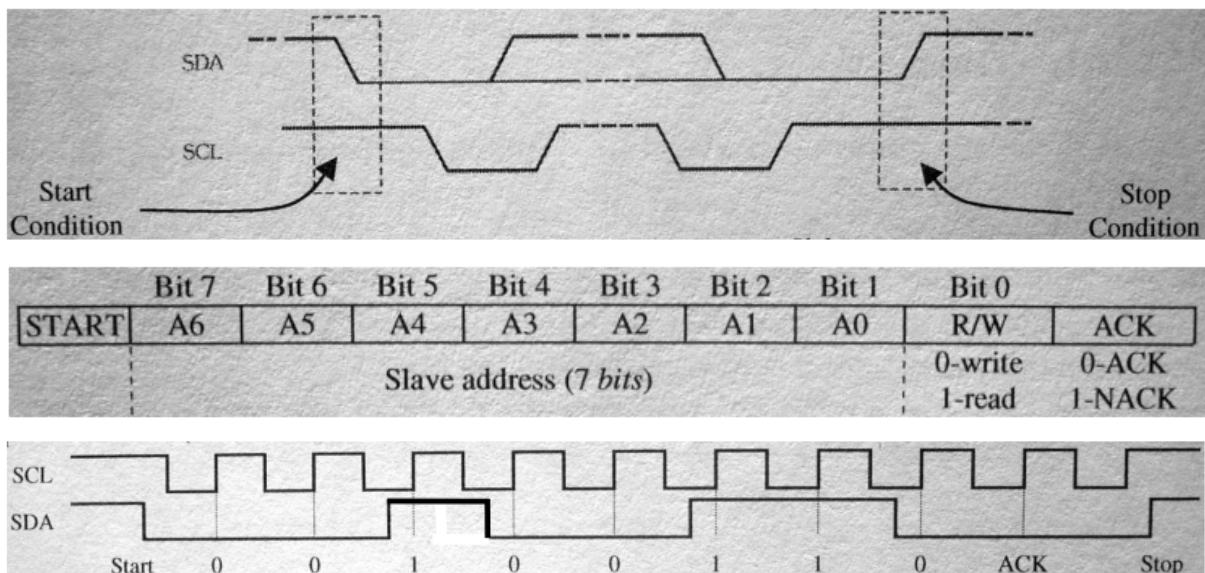
## I2C

Jedná se o **synchronní sériové** rozhraní, které funguje na principu **Master-Slave** komunikující způsobem **half-duplex**. Rozhraní tvoří dva vodiče: **SDA (Serial Data** - vodič po kterém jsou přenášena data) a **SCL (Serial Clock** - vodič nesoucí synchronizační signál - hodiny). Oba vodiče jsou zapojeny přes **pull up** rezistory k **Vdd** (log. 1), to znamená v klidu jsou oba vodiče v **log. 1**. Přenos dat probíhá následovně:

- **Start condition** (zahájení komunikace): komunikaci vždy zahajuje Master (obvykle MCU) při **SCL v log. 1** změnou úrovně **SDA** z **log. 1** na **log. 0** (**1->0**).
- **Komunikace**: Data jsou **vzorkována** z vodiče **SDA** s **náběžnou** hranou na vodiči **SCL**, změna úrovně na vodiči **SDA** musí být tedy prováděna, když je **SCL v log. 0**.

- **Stop condition** (ukončení komunikace): Master na vodiči **SDA** generuje hranu z **log. 0** do **log. 1 (0->1)**. **SCL** musí být v **log. 1**, jinak by šlo pouze o změnu dat.

Standardně se v datovém rámci přenáší **1 byte** užitečných dat (**po bitech**), každý rámec je zakončen s **ACK** (potvrzení Slave, že data obdržel). Volbu komunikujícího zařízení (Slave) Master realizuje pomocí **adresy** na začátku komunikace v **1. adresovém rámci**. Tato adresa má **7 bitů** (na **I2C sběrnici** tak může být připojeno až **128** zařízení Slave - Master je vždy **jeden**), **osmý** bit určuje, jestli půjde o **čtení** nebo **zápis**.



## Paralelní rozhraní

Slouží pro komunikaci komponentů MCU. Je současně posíláno více bitů po více vodičích. Tento způsob může být rychlejší, ale musí se brát ohled na interference paralelních vodičů (magnetická indukce).

<https://electronics.stackexchange.com/questions/21675/what-should-i-know-about-interference-between-wires-in-a-multi-conductor-cable>

## Převodníky

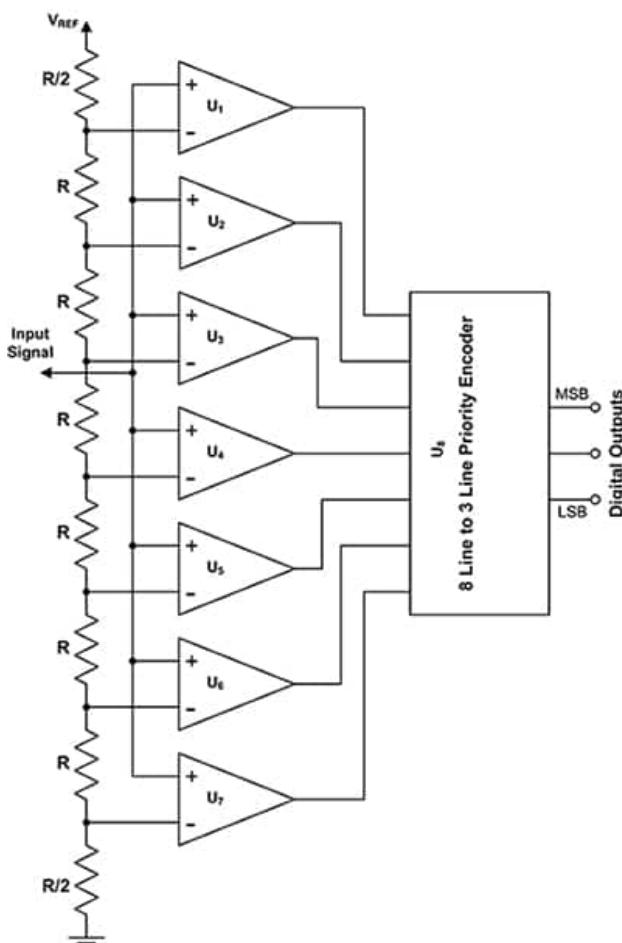
Umožňují komunikaci se světem, který je analogový. Slouží většinou k získávání informací z různých senzorů, které převádí měřené veličiny na napětí a toto napětí se poté převádí pomocí **AD** převodníku na binární hodnotu, kterou dokáže interpretovat MCU. U **DA** převodníku pak MCU pomocí binární hodnoty může vysílat potřebné napětí na výstupu, které např. nastavuje teplotou.

### AD převodník (analog-to-digital converter)

Převádí **analogový** (spojitý) signál na **digitální** (diskrétní - **binární číslo**, které dokáže interpretovat **MCU**). Naměřené hodnoty MCU ale dostává v rozsahu **0 až**

$2^N-1$  a musí je převést na odpovídající hodnotu napětí. Existuje více druhů AD převodníků, dva základní jsou:

- **Flash ADC** (direct-conversion ADC): Jedná se pouze o **kombinační** obvod, převod probíhá **paralelně** s **konstantní** rychlostí. **N** bitový převodník je tvořen  **$2^N - 1$  komparátory**, které jsou připojeny k **napěťovému žebříku** na vstupu **minus** (-), který dělí **referenční napětí**, a na vstup **plus** (+) je přivedeno **měřené napětí**. Výsledky komparátorů jsou připojeny k **prioritnímu kodéru**, na jehož výstupu je výsledek převodu. Jeho výhodou je **rychlosť** (převod probíhá v reálném čase kontinuálně pouze s nepatrným zpožděním), nevýhodou je cena a složitá realizace (velký počet **komparátorů**, nutnost mít velmi **přesné odpory**).
  - Pokud je  $V_{in}$  **poloviční** jako  $V_{ref}$ , bude v **log. 1** spodní polovina výstupů z komparátorů.
  - Pokud je  $V_{in}$  **nulové**, nebude v **log. 1** žádný komparátor.
  - Pokud je  $V_{in}$  **rovné nebo větší**  $V_{ref}$ , budou v **log. 1** všechny výstupy komparátorů.

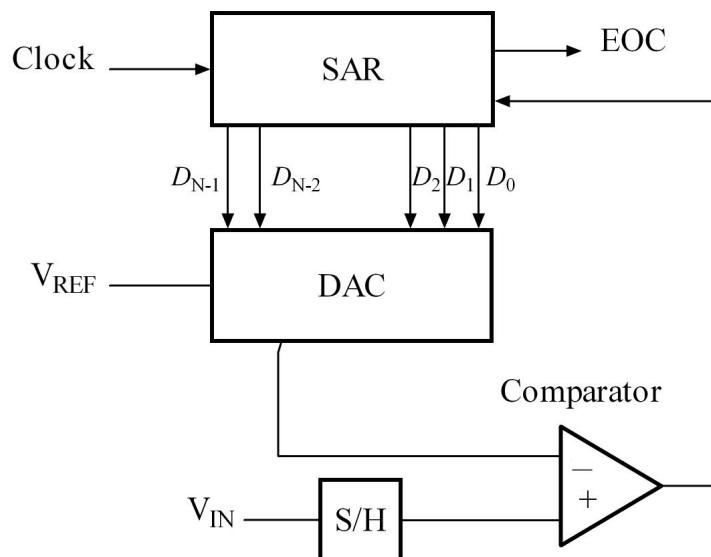


- **Aproximační ADC**: Jedná se o **sekvenční** obvod, měřené napětí musí být **navzorkováno** a poté až probíhá převod. Funguje na principu binárního vyhledávání (**půlení intervalů**) správné hodnoty napětí, pracuje ale s lineární

časovou složitostí **O(N)**, kde **N** je počet převáděných bitů (musí se zkontrolovat každý). Je tvořen:

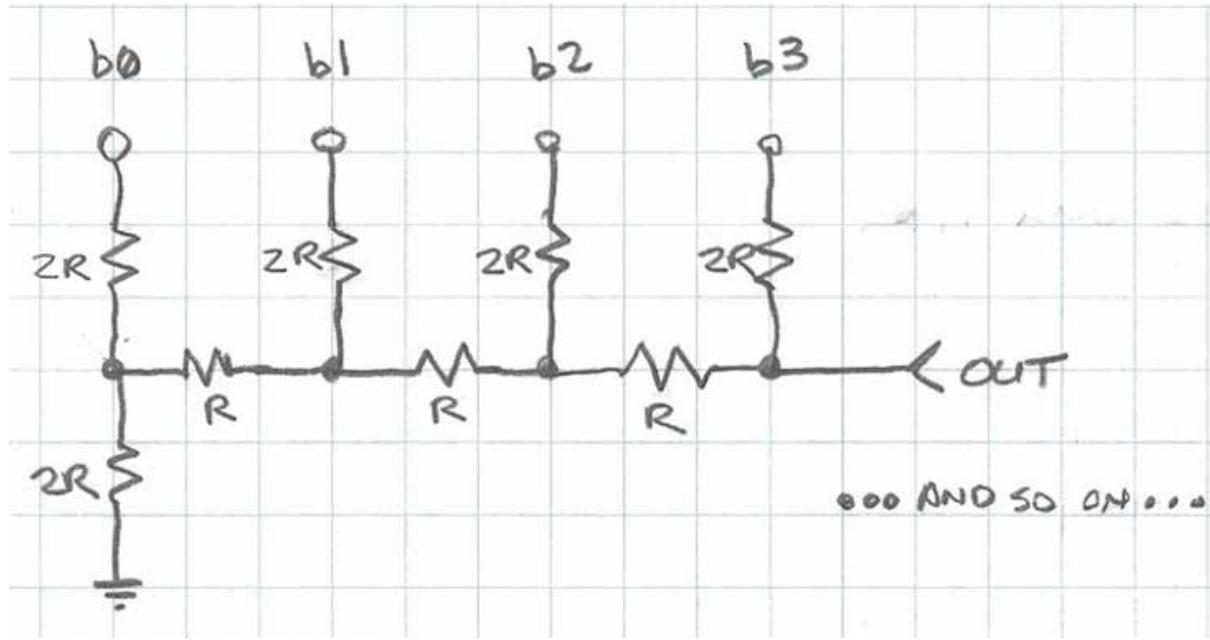
- **Sample and hold obvod**: navzorkuje napětí na začátku převodu a poté jej drží beze změny až do jeho konce,
- **DA převodník**: převádí aktuální **odhad** napětí z binární hodnoty na napětí,
- **Komparátor napětí**: porovnává **odhad** napětí naměřeného napětí s **měřeným** napětím,
- **SAR** (successive approximation register): obsahuje aktuální odhad naměřeného napětí v binární podobě. Na začátku je tato hodnota rovna polovině rozsahu (pouze **MSB** je v **log. 1**).

Převod probíhá tak, že **MSB** v **SAR** je nastaven na **jedna** a zbylé bity jsou **vynulovány**. Tato bin hodnota je převedena na napětí pomocí **DAC** a **komparátorem** porovnána s měřeným napětím. Pokud je odhad napětí **menší** než měřené napětí, zůstane **MSB v log. 1**, jinak je změněn na **log. 0**. V obou případech je nastaven druhý nejvíce signifikantní bit na **log. 1** a proces se opakuje. Teď už se ale po porovnání případně mění tento bit a na **log. 1** se nastavuje další.



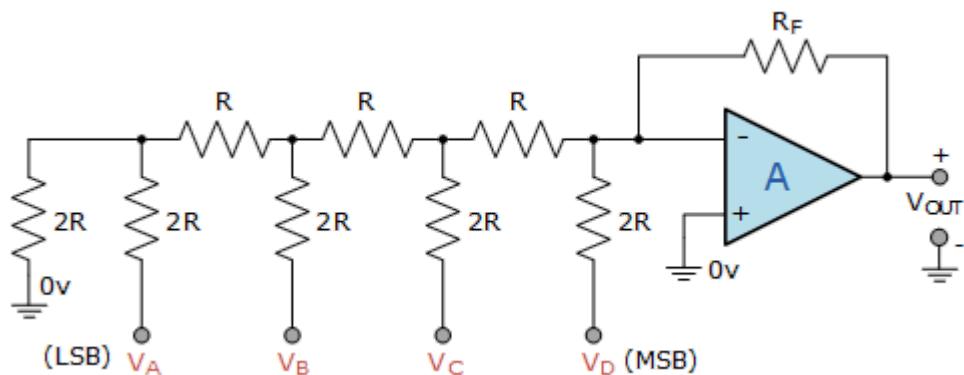
## DA převodník (digital-to-analog converter)

Převádí **digitální** signál (binární číslo) na **analogový** (spojitý). Většina DAC je tvořena **R-2R odporovým žebříkem** a případně ...



**b0** je **LSB**, **b3** je **MSB**. Celkový odpor obvodu v místě **out** je **R**. Paralelně zapojené dva **2R** rezistory pod **b0**, mají odpor roven součtu jejich **převrácených** hodnot, tedy **R**, což znamená, že pod **b1**, vzniknou také paralelně zapojené dva **2R** ( $2R$  a  $2 \times 1R$  v sérii) rezistory atd. Napětí z **b0** bude po cestě k **out** rozděleno **čtyřikrát**, tedy doputoje tam  $1/2^4 = 1/16$ , z **b1** bude rozděleno **třikrát** na  $1/8$ , z **b2** **dvakrát** na  $1/4$  a z **b3** jednou na  $1/2$  DAC Methods R2R Ladder . Na **out** takhle může být přivedeno napětí v rozmezí **0** až **Vcc**, které je na vstupech **b0-b3**, po šestnáctinových skocích. Pokud chceme převádět na jiná napětí, musí být **out** zapojeno ještě na **operační zesilovač**.

How OpAmps Work - The Learning Circuit



### Odkazy:

- Seriové rozhraní: [Sériová rozhraní u mikrokontrolerů – informatika](#)

# 6. Principy řízení a připojování periferních zařízení (přerušení, programová obsluha, přímý přístup do paměti, sběrnice).

Přístup k periferním zařízením řídí řadič periferních zařízení, jeho funkce jsou:

- komunikace s CPU,
- vyvolání přerušení,
- řízení a časování operací periferních zařízení,
- realizace vyrovnávací paměti,
- detekce a reportování poruch.

Řadič periferních zařízení obsahuje registry, kterými procesor se zařízením komunikuje - **datový register**, **řídící register**, **stavový registr**. Logicky lze periferní zařízení připojit dvěma způsoby:

- **mapovaný IO**: registry pro ovládání zařízení jsou **namapovány** na adresy **hlavní paměti**, periferní zařízení (registry) a paměť **sdílí stejný adresový prostor** a CPU tak může operace s periferním zařízením provádět stejně jako operace s pamětí (čtení a zápis). Řešení využívá adresový dekodér.
- **izolovaný IO**: hlavní paměť a periferní zařízení mají **různé adresové prostory**. Operace s periferními zařízeními provádí procesor pomocí speciálních instrukcí, např. IN, OUT nebo READ, WRITE.

Fyzicky lze periferních zařízení obecně připojit dvěma způsoby:

- **Point-to-point linky**: Mezi **každými dvěma** zařízeními (většinou je jeden ze dvojice **CPU**), které chtějí komunikovat je nutné vést k tomu vyhrazené vodiče.
  - **výhody**: kratší vodiče - vyšší kmitočty a **rychlosť** přenosu, komunikace 1 ku 1, současně může komunikovat **více párů** zařízeních - **velká propustnost**.
  - **nevýhody**: velký **počet** vodičů, **cena**, složitost, zařízení musí mít adekvátní počet vstupů/výstupů.
- **Sběrnice**: **velký** počet zařízení sdílí poměrně **malý** počet vodičů (připojují se na jeden centrální). Jedná se např. o Universal Serial Bus (**USB**), Peripheral Component Interconnect (**PCI**), PCI-Express, I2C
  - **výhody**: **nižší** počet vodičů, **levné**, lze dobře **škálovat**, zařízení stačí **jeden vstup/výstup** pro komunikaci s více jinými zařízeními.
  - **nevýhody**: delší vodiče - nižší kmitočty - **pomalejší** přenos, současně může komunikovat pouze **jeden pár** zařízení (respektive pouze jedno zařízení může v daný okamžik **vysílat**, přijímat mohou všechny)- **nižší propustnost**, složitější komunikace - **výběr příjemce**.

## Přerušení

přerušení, respektive **obsluha přerušení**, je mechanismus, kterým lze obsloužit **asynchronně** vznikající události mimo procesor - z **periferních zařízení**, např. úder do klávesnice (přerušení mohou vznikat i **uvnitř** procesoru, např. **dělení 0**, umožnuje tak OS řešit tyto události, případně mohou být vyvolána i programově).

Průběh obsluhy přerušení:

1. Periferní zařízení vyvolá přerušení (elektrický impuls) na řadič přerušení.
2. Řadič přerušení identifikuje a upozorní procesor.
3. Jakmile CPU je ve stavu, kdy může přerušení obsloužit (dokončí instrukci), požádá řadič o aktuálně **nejprioritnější nemaskované** přerušení a započne jeho obsluhu.
4. CPU uloží **stav** aktuálně běžícího procesu na **zásobník**.
5. Na základě vektoru přerušení vybere **obslužnou rutinu** (její adresu v paměti), vyhledá ji a spustí.
6. Po dokončení obslužné rutiny **obnoví stav** přerušeného programu a pokračuje v jeho vykonávání.

**Priorita** přerušení udává, v jakém **pořadí** budou přerušení zpracována, pokud jich současně vznikne více. Existují maskovatelná přerušení, která procesor ignoruje.

## Programová obsluha (polling)

Jedná se o nenáročný a neefektivní způsob obsluhy periferních zařízení. Běžící program se **periodicky dotazuje** ve smyčce na periferní zařízení, jestli nedošlo ke změně jeho stavu (stisk klávesy). Běžící program tak zbytečně plýtvá výpočetním výkonem a elektrickou energií, protože změny stavu zařízení vznikají v poměru k testování na tyto změny málo často.

## Přímý přístup do paměti (DMA - Direct Memory Access)

Koncept přerušení je nevýhodný, pokud je třeba přenášet **větší objemy dat**. Při obsluze přerušení se procesor **podílí na datových přenosech**, což ho zatěžuje. Přímý přístup do paměti umožňuje přenášet data z periferního zařízení do paměti nebo z paměti do paměti bez zatížení procesoru. Ten tak může provádět výpočet běžícího programu. Proces přímého přístupu do paměti probíhá následovně.

1. CPU předá **řadiči DMA** (specializovaný obvod, který obstarává přímý přístup do paměti) **adresu** v paměti, **adresu** periferního zařízení, **druh přenosu** (čtení/zápis) a **počet** přenášených **slov**.
2. Řadič DMA provádí **přesun** dat **bez** zásahu CPU, který může vykonávat další kód. Procesor je **omezen** na využití sběrnice.
3. Po **dokončení** přenosu dat (případně při chybě) **řadič DMA** zasílá procesoru **přerušení**, čímž ho o této skutečnosti informuje.

Rozšířením řadiče DMA je koncept IO procesoru, což je co-procesor, řídící IO periferních zařízení.

## Sběrnice

Jedná se o propojovací soustava umožňující komunikaci mezi **více** než jedním **zdrojem** dat a **přijímači** dat. Zajišťuje přenos dat a řídících povelů mezi nimi a definuje:

- **komunikační rozhraní**: soustava vodičů, jejich význam a elektrické charakteristiky.
- **komunikační protokol**: přesně **určuje**, jak bude **komunikace** mezi dvěma zařízeními **probíhat**, např. definuje **pořadí změn hodnot signálů** (viz start a stop condition u I2C)
- **transakce** (cyklus): **posloupnost** kroků, která **realizuje** danou funkci, např přečtení dat z adresy.

## Způsoby komunikace

popis viz minulá otázka.

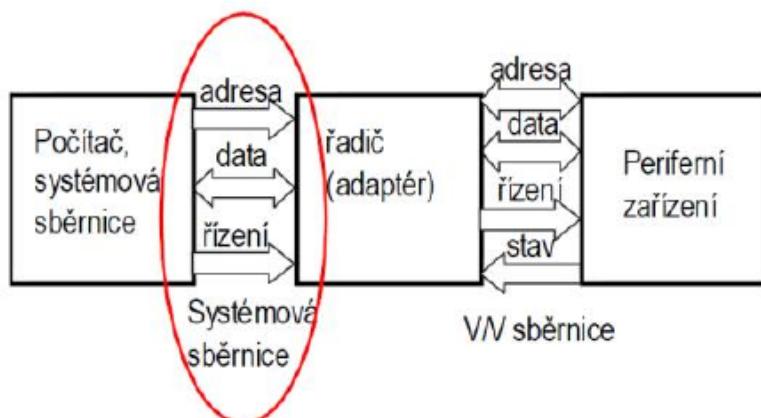
- **synchronní - asynchronní**,
- **sériová - paralelní**,
- **simplex - half duplex, full duplex**.
- **jednostranně** (bez handshake) - **oboustranně** (potvrzení přenosu - handshake)

## Druhy sběrnic

- **Sériová/Paralelní sběrnice** - viz okruh 5.
- **Interní/Externí sběrnice** - Pro propojení interních nebo externích periferií.
- **Nesdílená (Dedikovaná) sběrnice** - Pro každý **signál (data, adresa, řízení)** je samostatný vodič. Připojení pouze jednoho zařízení.
- **Sdílená sběrnice** - Všechny signály se posílají po **společné** sadě vodičů. Pro rozlišení o jaký signál se jedná se používají **identifikační signály (adresy)**.

## Komunikační kanály

- **Adresa**
- **Data**
- **Řízení**

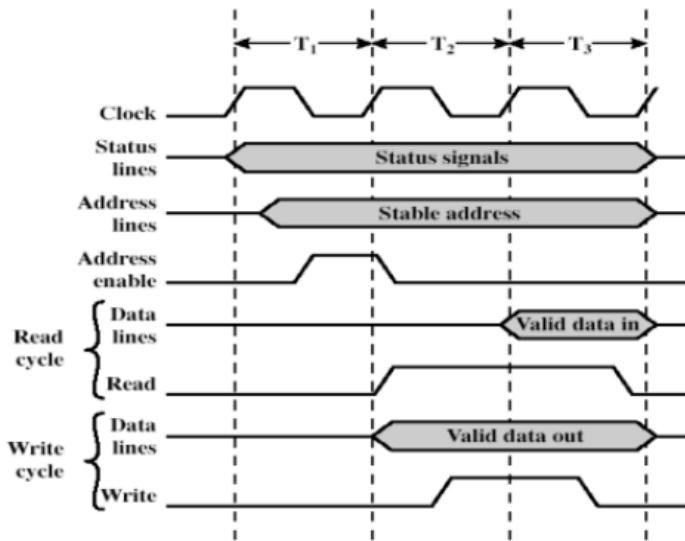


## Parametry

- **Rychlosť sběrnice** - určuje ji **šířka sběrnice, technologie** (paralelní/sériová), **kmitočet synchronizačních hodin**.
- **Šířka pásma** - **šířka sběrnice \* rychlosť sběrnice**

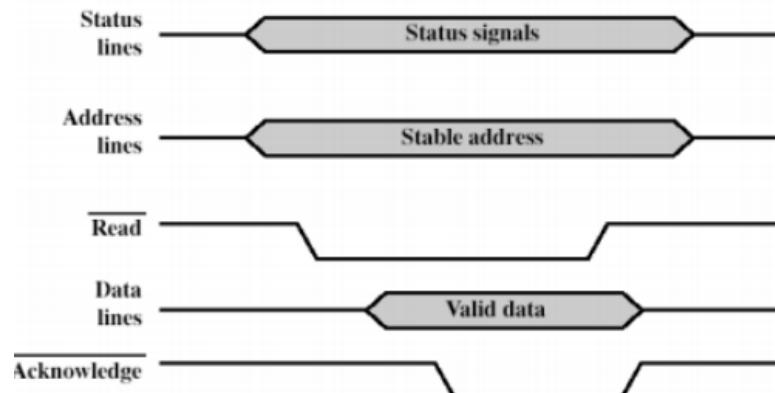
## Synchronní sběrnice

Komunikace řízena hodinovým signálem, který je rozveden do všech zařízení.



## Asynchronní sběrnice

Generování signálů je **vázáno na výskyt událostí**, obvykle zařízení mají signály definující začátek a konec čtení/zápisu (**MSYN** a **SSYN**).

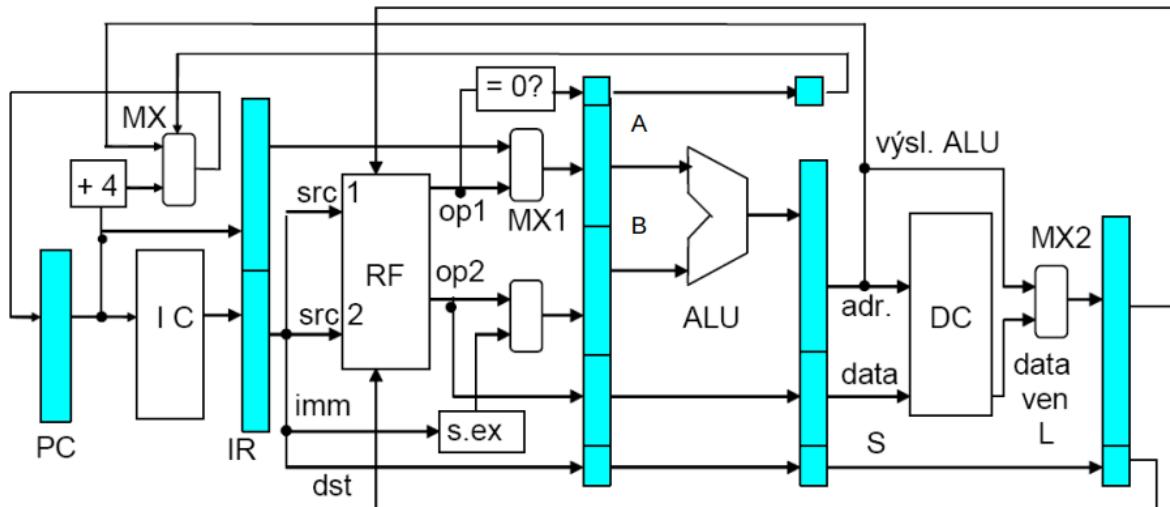


## Rozhodování o přidělení sběrnice

Pokud současně žádá o přidělení sběrnice více zařízení, musí být nějak zajištěno, aby na ni vysílalo vždy pouze jedno. Pro zajištění se používají přístupy:

- **centrální řízení** - decentralizované řízení,
- **prioritní přístup** - spravedlivé přidělování sběrnice - **cyklické přidělování** - **náhodné**.

# 7. Princip činnosti počítače (řetězené zpracování instrukcí, RISC, CISC).



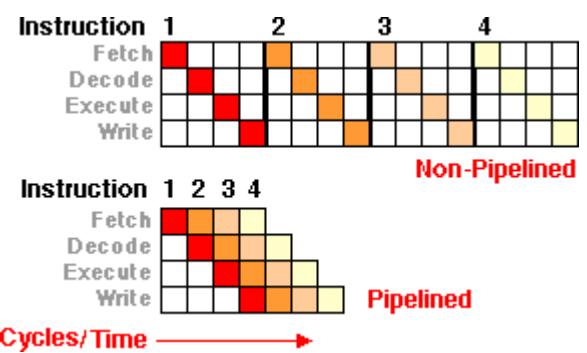
## Zřetězené zpracování instrukcí (pipelining)

Procesor je velmi složitý obvod a provedení instrukce se typicky děje v **různých** jeho částech. Proto jsou CPU rozděleny na **stupně**, každý stupeň může **současně** řešit jinou část instrukce. To znamená, že lze současně provádět výpočet více instrukcí, každou v **jiném stavu dokončení**. Mezi stupni CPU přechází instrukce s **taktem** hodin, ideálně lze díky **řetězenému zpracování (pipelining)** neustále využívat **všechny** stupně CPU a po **každém taktu** dokončit jednu instrukci (**CPI = 1 cycles per instruction**).

Basic five-stage pipeline		1	2	3	4	5	6	7
Instr. No.	Clock cycle	IF	ID	EX	MEM	WB		
1								
2								
3								
4								
5								

(IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back).

In the fourth clock cycle (the green column), the earliest instruction is in MEM stage, and the latest instruction has not yet entered the pipeline.



**Stavy provádění**, ve kterých se instrukce může nacházet vypadají:

- **Fetch (F)** - načtení instrukce,
- **Decode (D)** - dekódování instrukce,
- **Execute (E)** - vykonání instrukce,
- **Memory Access (M)** - čtení z paměti,
- **Write Back (W)** - zápis do paměti.

Zrychlení zřetězené linky

$$\frac{N*k*t}{(k+N-1)*(t+d)}$$

kde:

- **N** je počet instrukcí,
- **k** je počet stupňů zřetězené linky,
- **t** je zpoždění stupně,
- **d** je zpoždění registru (mezipaměti - klopný obvod)

## Konflikty (hazardy)

Situace, které je nutné při řetězení instrukcí řešit, jinak by mohl být výsledek výpočtu nesprávný. Často konflikty způsobují **zpomalení linky (stall)** linka musí čekat na **dokončení** některé instrukce, někdy stačí, když překladač změní pořadí **nezávislých** instrukcí. Konflikty dělíme na:

- **Strukturální** – obvodová struktura neumožňuje současně provedení určitých akcí, např. současně čtení dvou hodnot z paměti.
- **Datové** – jsou zapotřebí data z **předcházející** instrukce, která ještě není dokončena (např. u dvou následujících instrukcí využívající stejný register). Instrukce ADD R1, R2, R3 a po ní následující SUB R4, R5, R1 pracují s registrem R1, druhá instrukce by pracovala již s **neplatnou** hodnotou, pokud by problém nebyl řešen. Hloupé řešení je pozastavit linku a počkat na dokončení první instrukce. Chytré ale dražší řešení je doplnit CPU o datové cesty, které umožní předávat mezivýsledky (**forwarding** nebo **bypassing**) mezi jednotlivými stupni CPU. Né vždy to je ale možné, např. u dvojice instrukcí LOAD R1, addr a ADD R3, R1, R2. obsah R1 je získán až ve na konci fáze Memory Access první instrukce (konec 4. taktu), instrukce ADD jej potřebuje ale už na začátku fáze Execute ve svém 3. taktu (celkově začátek 4. taktu), linka tak musí **čekat**. Datové hazardy dělíme na:
  - **RAW – Read After Write**: 2. instrukce se pokouší číst zdroj před tím, než do něj zapsala 1. instrukce.
  - **WAW – Write After Write**: 2. instrukce se pokouší zapsat operand dřív, než je proveden zápis 1. instrukcí.
  - **WAR – Write After Read**: 2. instrukce se pokouší zapsat operand dřív, než je 1. instrukcí přečten.

- **Řídící** – skoková instrukce **mění** obsah PC (program counter, ukazatel na následující instrukci), nebo jiné. U nepodmíněných skoků procesor nezná adresu instrukce následující po skoku, nemůže tak načítat (fáze Fetch) další instrukce a linka musí být pozastavena. Pozastavení lze ale předejít v případech:
  - V programu se vyskytují jiné nezávislé instrukce, kterými lze vyplnit prostor mezi **získáním** adresy a skokem, jde o **zpožděný skok**. Změnu pořadí instrukcí provádí překladač, musí zajistit, aby nedošlo ke změně výsledku.
  - Zavést specializované obvody (**prediktor skoku a paměť cílové adresy skoku - BTB**), které odhadnou, zda **provést/neprovést** skok a adresu skoku. Dle odhadu se začnou načítat instrukce dle adresy z **BTB (branch target buffer** - cache, kterou CPU postupně naplňuje a aktualizuje) a linka se v případě správného odhadu **nepozastaví**. Pokud byl odhad **špatný**, je provádění těchto instrukcí předčasně ukončeno a začíná se s prováděním těch **správných**, to ale způsobí **zpomalení** linky. U moderních procesorů, které mají **delší linky** (10-20 stupňů), je velký požadavek na kvalitní prediktory. Bylo zjištěno, že se provede většinou kolem **80%** skoků. Predikce může být **statická** - určí ji překladač nebo **dynamická**, určuje ji procesor (speciální HW obvod). Nejjednodušší dynamická predikce odhaduje skok na základě toho, jak byl vykonán v předcházejícím běhu.

## Ortogonalní instrukční sada

Instrukce fungují se všemi adresovacími mody (instrukce není nucená například mít jako první operand akumulátor).

## RISC (Reduced Instruction Set Computer)

- **Redukovaná instrukční sada.**
- Oproti CISC má menší množství instrukcí s **jedním** adresovacím režimem - **register**, načítat data do/z registrů umožňují **pouze** instrukce **LOAD a STORE**.
- všechny instrukce jsou stejně dlouhé (**stejný počet bitů** v paměti).
- Na zřetězené lince lze ideálně zajistit dokončení instrukce **každý takt CPI = 1**. To znamená, že každá instrukce musí trvat **stejný počet taktů**.
- Využívá velké množství **registrov**.
- Na čipu **dominují registry** (paměť) oproti obvodům pro dekódování a řízení instrukcí.
- obvykle mají **menší** spotřebu energie,
- Apple Silicon.

## CISC (Complex Instruction Set Computing)

- **Složitá instrukční sada** - Hlavní myšlenkou procesorů CISC je to, že k provádění operací **načítání, vyhodnocení a ukládání** lze použít jedinou instrukci (**memory-to-memory, mem-to-reg, reg-to-mem, reg-to-reg**).
- Mnoho **typů instrukcí** s mnoha variantami a mnoha **adresovacími režimy**.
- Instrukce nemusí mít stejnou délku v bitech - náročnější na dekódování.
- Dle složitosti instrukce navíc mohou trvat různou dobu (počet taktů), hůře se řeší zřetězené zpracování a **CPI > 1**.
- Komplexní instrukce mohou být na rozdíl od RISC efektivnější, ale složitější na použití.
- Nepoužívá mnoho registrů, někdy pouze **střadač**.
- Na čipu dominuje logika pro **dekódování a řízení** instrukcí před registry.
- Intel x86.

# 8. Minimalizace logických výrazů (algebraické metody, Karnaughova mapa, Quine Mccluskey)

Minimalizaci logických výrazů provádíme z důvodu:

- menší, jednodušší logický výraz,
- menší počet logických hradel,
- menší počet spojů hradel,
- rychlejší výpočet,
- levnější výroba,
- méně integrovaných obvodů,
- delší doba bez poruch,
- jednodušší servis,
- větší spolehlivost při praktickém využívání.

## Metody minimalizace

- **Algebraické** - Aplikace **axiomů Booleovy algebry**. Náročné pro velké příklady.
- **Grafické** - Vennovy diagramy, **Karnaughova mapa**, jednotková krychle...
- **Algoritmické** - **Quine-McCluskey**.

## Algebraická minimalizace

Algebraickou minimalizaci provádíme aplikací **axiomů Booleovy algebry**, které jsou:

- **uzavřenost**:  $(a+b) \in B$ ,  $(a \cdot b) \in B$  ( $a, b \in B$ )
- **identita**:  $a+0=a$ ,  $a \cdot 1=a$  - z hlediska implementace není není tuto opraci provádět,
- **komutativita**:  $a+b=b+a$ ,  $a \cdot b=b \cdot a$  - z hlediska implementace umožňuje volbu zapojení do hradel,
- **distributivita**:  $a+b \cdot c = (a+b) \cdot (a+c)$ ,  $a \cdot (b+c)=a \cdot b+a \cdot c$  - kratší výrazy jsou z hlediska implementace výhodnější,
- **komplementárnost**:  $a \cdot a'=0$ ,  $a+a'=1$  - z hlediska implementace lze předem určit výsledek
- v množině  $B$  existují alespoň dva různé prvky

## Teorémy

Jsou odvozeny na základě **axiomů Booleovy algebry** a definují další užitečné vlastnosti využívané při **minimalizaci** logických výrazů, jsou to:

- **jedinečnost 0 a 1**,

- idempotence:  $a+a=a$ ,  $a\cdot a=a$ ,
- agresivita 1 a 0:  $a+1=1$ ,  $a\cdot 1=1$ ,
- absorpce:  $a+a\cdot b=a$ ,  $a\cdot(a+b)=a$ ,
- De Morganovy zákony:  $(a+b)'=a'\cdot b'$ ,  $(a\cdot b)'=a'+b'$ ,
- asociativita:  $(a+b)+c=a+(b+c)$ ,  $(a\cdot b)\cdot c=a\cdot(b\cdot c)$ ,
- dvojitá negace:  $a''=a$ ,
- absorpce negace:  $a+a'\cdot b=a+b$ ,  $a\cdot(a'+b)=a\cdot b$ .

## Normální formy

- **úplná normální disjunktní forma ÚNDF** (SoP: Sum of Products): Výraz je sepsán jako **suma součinů**. Z pravdivostní tabulky ji získáme tak, že vytvoříme **součiny (AND) vstupních** proměnných v řádcích, kde má výstupní funkce hodnotu **log. 1** tzv. **mintermy**. Všechny tyto mintermy pak **sečteme** (OR). Každá proměnná v součinu je zapsána tak, že pokud nabývá hodnoty **log. 0**, pak ji píšeme s **negací**, pokud **log. 1**, pak píšeme **bez negace**. U zkráceného zápisu jsou uvedeny **stavové indexy**, pro které nabývá funkce hodnoty **log. 1**.

### Nezkrácená ÚNDF

$$F(x, y, z) = \bar{x} \cdot \bar{y} \cdot \bar{z} + \bar{x} \cdot y \cdot \bar{z} + x \cdot \bar{y} \cdot \bar{z} + x \cdot y \cdot \bar{z}$$

### Varianty zkráceného zápisu ÚNDF

$$F(x, y, z) = \vee(0,2,4,6) = 1(0,2,4,6) = \Sigma m(0,2,4,6)$$

- **úplná normální konjunktní forma ÚNKF** (PoS: Product of Sums): Výraz je sepsán jako **součin sum**. Z pravdivostní tabulky ji získáme tak, že vytvoříme ze **součtů** vstupních proměnných v řádcích, kde má výstupní funkce hodnotu **log. 0** tzv. **maxtermů**, a všechny tyto **maxtermy** pak **vynásobíme**. Každá proměnná v součtu je zapsána tak, že pokud nabývá hodnoty **log. 0**, pak ji píšeme **bez negace**, pokud **log. 1**, pak píšeme s **negací**. U zkráceného

### Nezkrácená ÚNKF

$$F(x, y, z) = (x + y + \bar{z}) \cdot (x + \bar{y} + \bar{z}) \cdot (\bar{x} + y + \bar{z}) \cdot (\bar{x} + \bar{y} + \bar{z})$$

### Varianty zkráceného zápisu ÚNKF

$$F(x, y, z) = \wedge(1,3,5,7) = \&(1,3,5,7) = \Pi M(1,3,5,7)$$

<i>s</i>	<i>c</i>	<i>b</i>	<i>a</i>	<i>f</i>	<i>minterm</i>	<i>maxterm</i>
0	0	0	0	0	-	$a + b + c$
1	0	0	1	0	-	$\bar{a} + b + c$
2	0	1	0	0	-	$a + \bar{b} + c$
3	0	1	1	1	$ab\bar{c}$	
4	1	0	0	0	-	$a + b + \bar{c}$
5	1	0	1	1	$a\bar{b}c$	
6	1	1	0	1	$\bar{a}bc$	
7	1	1	1	1	$abc$	

zápisu jsou uvedeny **stavové indexy**, pro které nabývá funkce hodnoty **log.**

**0.**

- **ÚNDF:**  $a \cdot b \cdot c' + a \cdot b' \cdot c + a' \cdot b \cdot c + a \cdot b \cdot c'$
- **ÚNKF:**  $(a+b+c) \cdot (a'+b+c) \cdot (a+b'+c) \cdot (a+b+c')$
- **minimální normální disjunktní forma MNDF:** Minimální možné řešení **termu** (uspořádaná skupina proměnných a operátorů) v normální disjunktní formě. Již nelze eliminovat žádnou proměnnou z termu.
- **minimální normální konjunktní forma MNKF:** Minimální možné řešení **termu** v normální konjunktní formě. Již nelze eliminovat žádnou proměnnou z termu.

## Karnaughova mapy

Karnaughovy mapy vytváříme s pomocí **Grayova** (sousední hodnoty se liší o jedený bit) kódu tak, aby se každé políčko se všemi sousedními lišilo pouze o **jediný** bit (přehození 3. sloupce a 3. řádku se 4.). Hodnota proměnné má být **1** v polích, kde je její odpovídající **bit 1**, v ostatních polích má být v **0**. V polích pod **podtržením** má proměnná **nabývat 1, v ostatních 0**.

		z				
		00	01	11	10	
		00	1	3	2	
		01	5	7	6	
		11	12	13	15	14
		10	8	9	11	10
w	x	y				

## Minimalizace Karnaughovy mapy v disjunktivní formě

**Jedničky** (popř. x) se sdružují do **skupin**, které jsou **mocniny 2** - lze i přes okraje a rohy. Pokud daná skupina zasahuje do **bitů 0 i 1** dané proměnné, proměnná **nebude** ve výsledném termu. Pokud daná skupina zasahuje pouze do **bitů 1** dané proměnné, **bude** proměnná ve výsledném termu **přímo**. Pokud daná skupina zasahuje pouze do **bitů 0**, **bude** proměnná ve výsledném termu **negovaně**.

		z				
		00	01	11	10	
		00	1	3	2	
		01	1	1	0	
		11	0	0	1	
x	x	y				

$$F(x, v, z) = \bar{x} \cdot z + x \cdot v \cdot \bar{z}$$

		z					
		00	01	11	10		
		w·x	y·z	00	01	11	10
w	00	0	1	3	2		
	01	1	0	X	1		
	11	X	1	0	0		
	10	1	1	0	0		
				x			
				y			

$$F(w, x, y, z) = \bar{w} \cdot y + w \cdot \bar{y} + \bar{w} \cdot x \cdot \bar{z}$$

### Minimalizace Karnaughovy mapy v konjunktivní formě

**Nuly** (popř. x) se sdružují do skupin, které jsou **mocniny 2** - lze i přes okraje a rohy. Pokud daná skupina zasahuje do **bitů 0 i 1** dané proměnné, proměnná **nebude** ve výsledném termu. Pokud daná skupina zasahuje pouze do **bitů 1** dané proměnné, **bude** proměnná ve výsledném termu **negovaně**. Pokud daná skupina zasahuje pouze do **bitů 0**, **bude** proměnná ve výsledném termu **přímo**.

		z					
		00	01	11	10		
		w·x	y·z	00	01	11	10
x	0	1		0	0		1
	1	1		0	0		1
				v			

$$F(x, y, z) = z'$$

		z					
		00	01	11	10		
		w·x	y·z	00	01	11	10
w	00	0	1	3	1		
	01	1	0	X			
	11	X	1	0	X		
	10	1	1	0	0		
				x			
				y			

$$F(w, x, y, z) = (w + x + y) \cdot (w + x' + z') \cdot (w' + y')$$

## Quine Mccluskey

Tabulární minimalizační metoda. Vhodná pro funkce s více proměnnými. Postup pro **ÚNDF (SOP)** ➔ Quine-McCluskey Minimization Technique (Tabular Method) :

- Seřaď do **skupin** jednotlivé implikanty v pořadí dle **počtu jedniček** v jejich binárních reprezentaci.

$$\sum_m (0, 1, 3, 7, 8, 9, 11, 15)$$

Step: -1

Group	Minterm	Bin. Rep.			
		A	B	C	D
0	$m_0$	0	0	0	0
1	$m_1$	0	0	0	1
	$m_3$	1	0	0	0
2	$m_7$	0	1	1	1
	$m_9$	1	0	0	1
3	$m_{11}$	0	1	1	1
	$m_{15}$	1	1	1	1

- Eliminuj proměnné, jejichž implikanty se liší v sousedních skupinách **jediným** bitem, místo tohoto bitu piš **pomlčku**. Pokud implikant nemá sousedné termury a nelze u něj eliminovat žádný bit, jedná se o zkrácený implikant.

### Step 2

Group	Matched pairs	Bin. Rep.			
		A	B	C	D
0	$m_0 - m_1$	0	0	0	-
	$m_0 - m_8$	-	0	0	0
1	$m_1 - m_3$	0	0	-	1
	$m_1 - m_9$	-	0	0	1
	$m_6 - m_9$	1	0	0	-
2	$m_3 - m_7$	0	-	1	1
	$m_3 - m_n$	-	0	1	1
	$m_9 - m_{11}$	1	0	-	1
3	$m_7 - m_{15}$	-	1	1	1
	$m_{11} - m_{15}$	1	-	1	1

3. Pokud byly eliminovány některé implikanty, pokračuj bodem 1 s takto eliminovanými implikanty.

### Step 3

Group	$M = P$	B.R.			
		A	B	C	D
0	$m_0 - m_1 - m_6 - m_9$	-	0	0	-
	$m_0 - m_8 - m_1 - m_9$	-	0	0	-
1	$m_1 - m_3 - m_9 - m_{11}$	-	0	-	1
	$m_1 - m_9 - m_3 - m_{11}$	-	0	-	1
2	$m_3 - m_7 - m_n - m_{15}$	-	-	1	1
	$m_3 - m_{11} - m_7 - m_{15}$	-	-	1	1

4. Hledej minimální řešení pokrytí dané funkce pomocí mřížky implikantů.

a. zapsat zkrácené implikanty do mřížky,

P.I.	Minterma zahrnující	0	1	3	7	8	9	11	15
$\bar{B}\bar{C}$	0, 1, 8, 9	X	X			X	X		
$\bar{B}D$	1, 3, 9, 11		X	X			X	X	
$CD$	3, 7, 11, 15			X	X			X	X

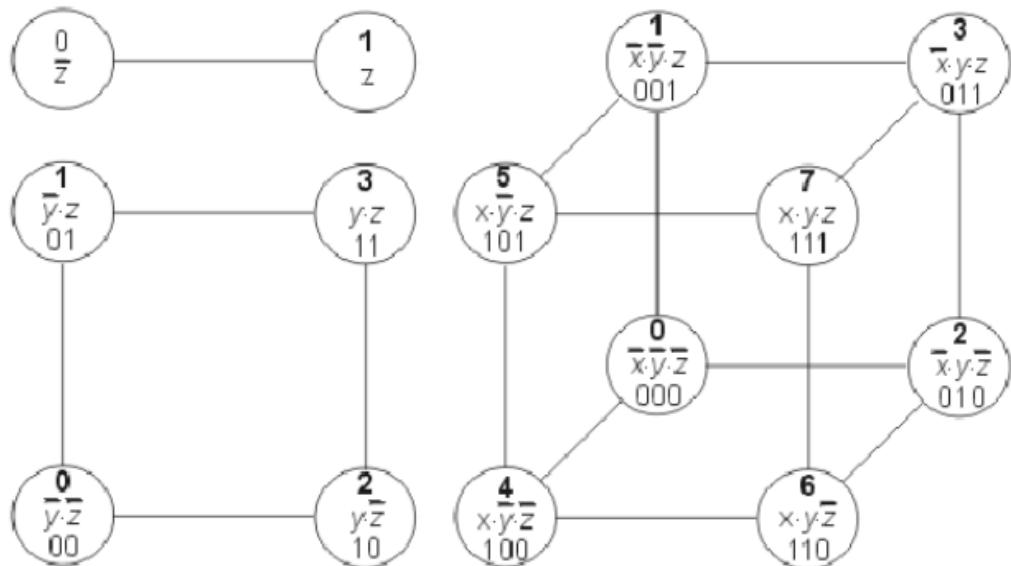
b. vybrat sloupce, kde je pouze jeden křížek,

P.I.	Minterma zahrnující	0	1	3	7	8	9	11	15
$\bar{B}\bar{C}$	0, 1, 8, 9	(X)	X			(X)	X		
$\bar{B}D$	1, 3, 9, 11		X	X			X	X	
$CD$	3, 7, 11, 15			X	(X)			X	(X)

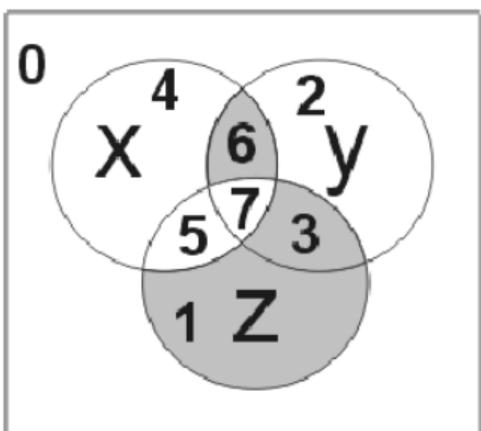
c. zapsat logický výraz na základě řádků, ve kterých se zakroužkované křížky.

$$Y = \bar{B}\bar{C} + CD$$

**N-rozměrná jednotková krychle** - Model pro 3-4 proměnné, velmi názorný.



**Vennovy diagramy** - graficky zobrazují boolovskou sousednost.



Obrázek 4 - Příklad funkce

**Odkazy:**

- [Disjunktivní a konjunktivní normální forma](#)

# 9. Reprezentace čísel a základní dvojkové aritmetické operace v počítači (doplňkové kódy, sčítání, odčítání, násobení, pevná a plovoucí řádová čárka, standard IEEE 754).

Čísla jsou v počítačích reprezentována hodnotami jednotlivých bitů - **binárně**, tedy ve **dvojkové soustavě**. Narozdíl od reálného světa je rozsah čísel omezen **počtem bitů**, které jsou pro číslo vyhrazeny. Dnes jsou obvykle používány tři číselné rozsahy, které mohou být **znaménkové i bez znaménka**, jsou to:

- **short int** - 16 bitů,
- **integer** - 32 bitů,
- **long int** - 64 bitů.

## Reprezentace celých čísel se znaménkem

Reprezentace znaménka vždy sníží rozsah **absolutní** hodnoty čísla na **polovinu**, nicméně umožňuje reprezentovat **záporná čísla a celkový počet** čísel se tak **nezmění**. Reprezentace znaménka je realizována následovně:

- **Přímý kód - První bit** je rezervován pro znaménko (**1 pro záporná, 0 pro kladná**). Existují zde však **2 nuly (+0 a -0)**. Použití například v IEEE754. (10000001 = -1). Kód **komplikuje** algoritmy pro **sčítání a odčítání**, nutnost testovat na znaménko. Dalším problémem je existence **dvojí nuly**.
- **Aditivní kód (kód s posunutou nulou)** - Číslo je posunuté o nějakou danou kontantu (Např. -127 = 00000000; 128 = 11111111; 0 = 01111111; -1 = 01111110...). **Nevýhodou** je, že reprezentace kladných znaménkových čísel se **liší** od neznaménkových, a při **násobení** se musí odečítat určená konstanta. Sčítání lze provádět normálně.
- Jedničkový doplněk (inverzní kód) - Záporná hodnota se získá **znegováním** všech bitů kladného čísla (MSB musí být u **kladných** vždy **0** a u **záporných** vždu **1**). Např. 0101 = 5, 1010 = -5, 0000 = 0, 1111 = -0. Kód **komplikuje** algoritmy pro **sčítání a odčítání** (jiné než u čísel bez znaménka). Dalším problémem je existence **dvojí nuly**.
- **Dvojkový doplněk** (doplňkový kód) - Kladné číslo je reprezentováno normálně, záporné však je **inverzí** kladného čísla a **přičtením 1**. **Nejpoužívanější** formát. Pouze jedna reprezentace 0. (11111111 = -1). Sčítání a odčítání lze realizovat **stejně** jako s neznaménkovými čísly, u násobení je nutné **replikovat MSB na dvojnásobný** rozsah čísla.

## Reprezentace desetinných čísel

Umožňují reprezentovat reálná čísla s omezenou přesností.

### Pevná řádová čárka

Pro zobrazení kladných reálných čísel. **Přesný počet** bitů je rezervováno pro **celou** část a **zbytek** pro **desetinnou** část čísla. Dnes se skoro nevyužívá. Např. 5 číslic pro celou část a 3 číslice pro desetinnou, **00101001 = 5.125**.

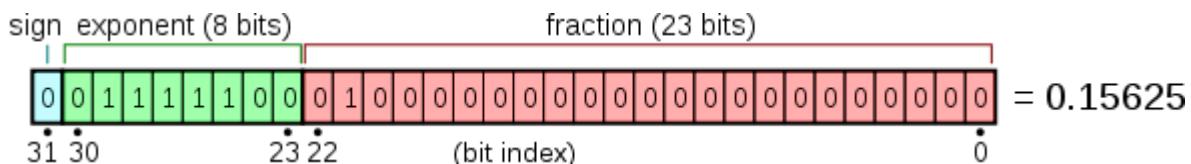
### Plovoucí řádová čárka (Floating point) - IEEE 754

Číslo je reprezentováno:

- **znaménkovým bitem (S)** - MSB - 1 znamená záporný, 0 kladný,
- **mantisou (M)** - v přímém kódu, obsahuje **implicitní 1**,
- **základem (B)** - je implicitní, rovna číslu **2**,
- **exponentem (E)** - v kódu posunuté nuly, což umožňuje vyjádřit záporný exponent.

**IEEE 754** definuje uložení čísla v plovoucí řadové čárce jako MSB je znaménkový bit, následují bity exponentu a poté až bity mantisy. Definovány jsou dvě reprezentace lišící se přesností a rozsahem čísla.

- **Single precision** (float - 32 bitů): Exponent má rozsah **8** bitů a je posunut o **-127** (polovinu rozsahu), mantisa je vyjádřena na **23** bitech, MSB je znaménkový. **Přesnost** je **7** dekadických čísel, rozsah zobrazení je  $\langle -2^{127}, 2^{127} \rangle$ , rozlišitelnost **normalizovaných** čísel je  **$2^{-127}$**  a **nenormalizovaných**  **$2^{-150}$** .



podmínka	hodnota	poznámka
$E = 1$ až $254$	$X = (-1)^s \times 2^{E-127} \times (1 + Q)$	základní formát
$E = 0, Q \neq 0$	$X = (-1)^s \times 2^{-126} \times Q$	denormalizovaná čísla
$E = 0, Q = 0, s = 0$	$X = 0$	kladná nula
$E = 0, Q = 0, s = 1$	$X = 0$	záporná nula
$E = 255, Q = 0, s = 0$	$X = +\infty$	kladné nekonečno (výsledek byl příliš vysoký)
$E = 255, Q = 0, s = 1$	$X = -\infty$	záporné nekonečno (výsledek byl příliš nízký)
$E = 255, Q > 0$	$X = \text{NaN}$	není číslo

- **Double precision** (double - 64 bitů): Exponent má rozsah **11** bitů a je posunut o **-1023** (polovinu rozsahu), mantisa je vyjádřena na **52** bitech, MSB

je znaménkový. **Přesnost** je **16** dekadických čísel, rozsah zobrazení je  $\langle -2^{1023}, 2^{1023} \rangle$ , rozlišitelnost **normalizovaných** čísel je  $2^{-1023}$  a **nenormalizovaných**  $2^{-1075}$ .

63 62	52 51	0
S	Exponent e	f <sub>51</sub> f <sub>50</sub> ... Mantisa f ... f <sub>0</sub>

**Explicitní jednička** - V mantise čísla je nutné explicitně mít jedničku.

**Implicitní jednička** - umožňuje zvýšit rozsah reprezentovaného čísla o jeden bit, v mantise se jednička neuvádí. Až na **denormalizované** čísla se s ní pracuje implicitně - **1.011 -> 011**.

### Chyba zobrazení

Jelikož se číslo skládá ze **součtu zlomků** mocnin 2 (např.  $2^{-2} = \frac{1}{4}$ ) **není** možné na konečném počtu bitů (registru) **reprezentovat** všechna čísla a tak dochází k **nepřesnostem**. Např. číslo **0.1** nelze přesně vyjádřit, což je obvyklá hodnota ve finančnictví a je nutné pro tyto aplikace používat jiné kódování čísel. Používá se **BCD (Binary Coded Decimal) kód**

- Každá dekadická číslice je zobrazena v jednom niblu (4 bitech bytu). Znaménko může být v prvním bytu.  $01000010 = 42$ .

Další nepřesnosti vznikají při sčítání/odčítání čísel, která jsou od sebe **řádově** vzdálená, mají **výrazně** rozdílný exponent (menší číslo může být úplně ignorováno).

### Matematické operace

#### Sčítání

- **Celočíselné** sčítání se provádí stejně jako v 10 soustavě. V případě sčítání v procesoru může dojít k **přetečení** (součet se nevleze do podporovaného rozsahu).

Zobrazitelný součet:

00101100	desítkově: 44
+ 01011010	+90
<hr/>	134

Nezobrazitelný součet:

10110100	desítkově: 180
+ 10110100	+180
<hr/>	# 360
1   01101000 = 104 <sub>10</sub>	

- **Floating point** sčítání není **asociativní** vlivem **zaokrouhlovacích chyb**. V **IEEE 754** se u sčítání musí nejdříve převést obě čísla na **stejný exponent**. Vždy se převádí číslo **s menším exponentem na větší**. (Nezapomenout na **implicitní 1**)  HOW TO: Adding IEEE-754 Floating Point Numbers

$$X = 1.000111 \times 2^5 \quad Y = 1.01001 \times 2^{11}$$

$$\begin{array}{r} 111 \\ 1000111 \times 2^5 \\ + 0.1010010 \times 2^5 \\ \hline 1.1100001 \times 2^5 \end{array}$$

### Odečítání

V CPU lze odčítání realizovat **bitovou negací** jednoho ze sčítanců a použít sčítání s nastavením **C0** na 1. Stejně tak je výhodné provádět ručně, pro člověka je to přirozenější.

Zobrazitelný rozdíl:

$$\begin{array}{r} 01011010 \\ - 00101100 \\ \hline 00101110 \end{array} \quad \text{desítkově: } 90 - 44 = 46$$

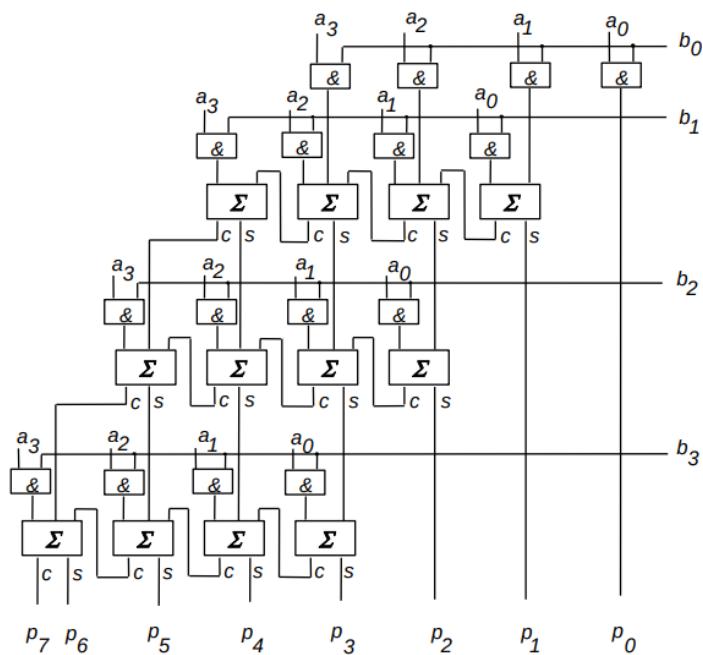
Nezobrazitelný rozdíl:

$$\begin{array}{r} 1 | 00101100 \\ - 01011010 \\ \hline 11010010 = 210_{10} \quad \# \quad - 46 \end{array} \quad \text{desítkově: } 44 - 90$$

### Násobení

- Celočíselné:** Výsledek může zabírat až **dvojnásobek** bitů. Při násobení čísel v doplňkovém kódu musí být nejdřív u obou čísel **rozšířen MSB** značící

$$\begin{array}{r} 0100 \\ * 0111 \\ \hline 0100 \\ 0100 \\ 0100 \\ \hline 11100 \end{array}$$



znaménko na **dvojnásobnou** přesnost původního čísla ve dvojkovém doplňkovém kódu.

- **IEEE 754:**

- znaménko výsledku se určí jako **XOR** znamének **činitelů**,
- **exponent** se určí součtem exponentů činitelů,
- **mantisa** se určí celočíselným násobením činitelů doplněných o **implicitní 1**.

$$X = -0.3_{10} = 0.01\overline{0011}_2 = 1.\overline{0011}_2 \times 2^{-2}_{10}$$

1	01111101	00110011001100110011010
---	----------	-------------------------

$$Y = 500.25_{10} = 11110100.01_2 = 1.111010001_2 \times 2^{8}_{10}$$

0	10000111	11110100010000000000000
---	----------	-------------------------

$$X \cdot Y = (X_s \cdot Y_s) \cdot 2^{X_E + Y_E} = (X_s \cdot Y_s) \cdot 2^6$$

$$X_E = -2 \quad X_s = 1.00110011001100110011010$$

$$Y_E = 8 \quad Y_s = 1.111010001$$

### Celočíselné dělení

V počítači velmi náročná operace prováděná mnoha různými metodami. Znaménkové dělení se může provádět s absolutní hodnotou a až poté dojde k doplnění znaménka. Postupy dělení:

- triviálně na principu **dělení pod sebou**,

Dekadicky	55 : 5 = 11	
Binárně	1 1 0 1 1 1 : 1 0 1 = 1 0 1 1	
	1 1 0 ↑	Za základ vezmeme 110, 110 > 101, výsledek = 1
	1 1 ↑	110 - 101 = 1, přidáme 1, 11 < 101, výsledek = 0
	1 1 1 ↑	Opišeme 11 a přidáme 1, 111 > 101, výsledek = 1
	1 0 1 ↑	111 - 101 = 10, přidáme 1, 101 = 101, výsledek = 1
	0	101 - 101 = 0, zbytek = 0

- **bez restaurace nezáporného zbytku**
  - začne se odečtením dělitele od dělence,

- b. na základě MSB se určí výsledek **i-tého** bitu: pro **MSB=1** je výsledek **0** - záporné číslo, pro **MSB=0** je výsledek **1** - kladné číslo,
- c. dělitel se posune o 1 bit doleva (násobení 2) a **MSB se zahodí**,
- d. pokud je výsledek i-tého bitu **0**, **přičte** se dělitel, v **opačném** případě se dělitel **odečte**.
- e. pokud nebylo dosaženo maximálního posuvu, pokračuje se bodem **b**),
- f. na konci může být nutné provést korekci zbyteku

$$\begin{array}{r}
 30 = 00011110, \quad 7 = 0111, \quad -7 = 1001 \\
 00011110 \\
 +1001 \qquad \qquad \qquad -d \\
 \hline
 10101110 \qquad <0 \Rightarrow c_4 = 0 \\
 0101110x \qquad \text{posuv} < - \\
 +0111 \qquad \qquad \qquad +d \\
 \hline
 1100110x \qquad <0 \Rightarrow c_3 = 0 \\
 100110xx \qquad \text{posuv} \\
 +0111 \qquad \qquad \qquad +d \\
 \hline
 000010xx \qquad >0 \Rightarrow c_2 = 1 \\
 00010xxx \qquad \text{posuv} \\
 +1001 \qquad \qquad \qquad -d \\
 \hline
 10100xxx \qquad <0 \Rightarrow c_1 = 0 \\
 0100xxxx \qquad \text{posuv} \\
 +0111 \qquad \qquad \qquad +d \\
 \hline
 1011xxxx \qquad <0 \Rightarrow c_0 = 0 \\
 +0111 \qquad \qquad \qquad +d \text{ (korekce na} \\
 \hline
 0010xxxx \qquad \qquad \qquad \text{zbytek 2}
 \end{array}$$

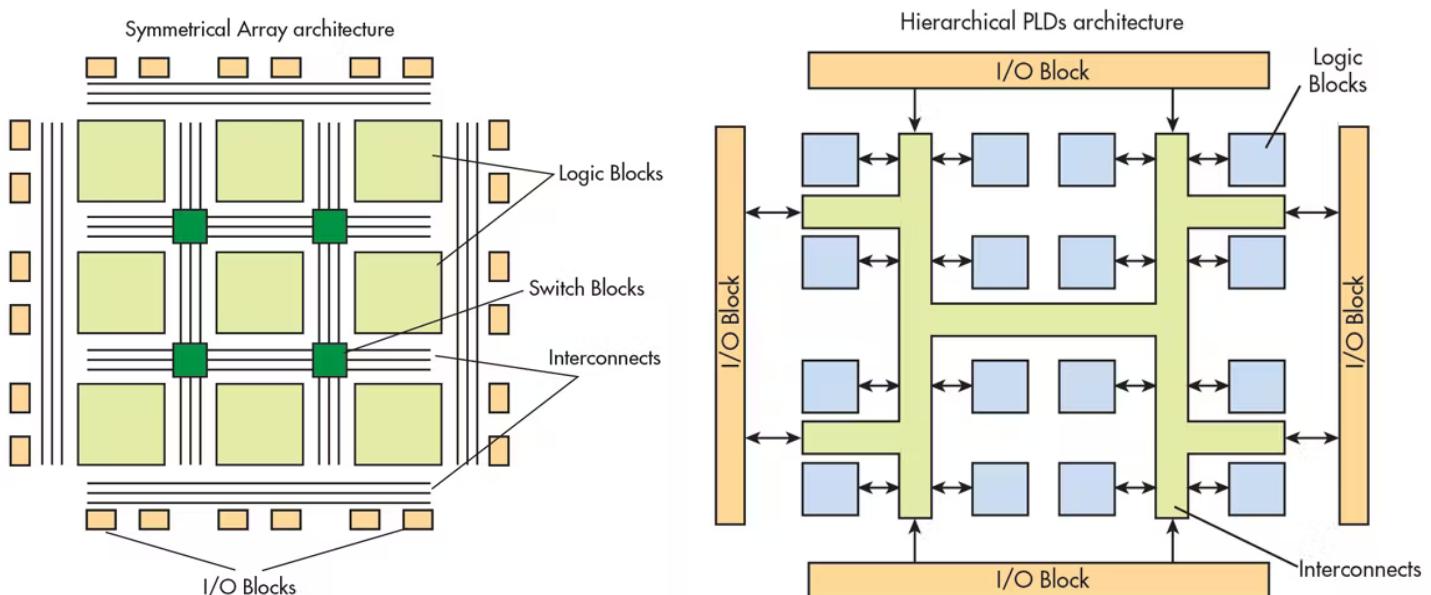
- algoritmem SRT

### Odkazy:

- [How to Convert a Number from Decimal to IEEE 754 Floating Point Representation](#)
  - [Video](#) - HOW TO: Convert Decimal to IEEE-754 Single-Precision Binary
- [HOW TO: Adding IEEE-754 Floating Point Numbers](#)

# 10. Technologie FPGA (vnitřní struktura, LUT), kroky návrhu aplikací využívajících FPGA a základy syntetizovatelného popisu hardware (strukturní a behaviorální popis obvodů).

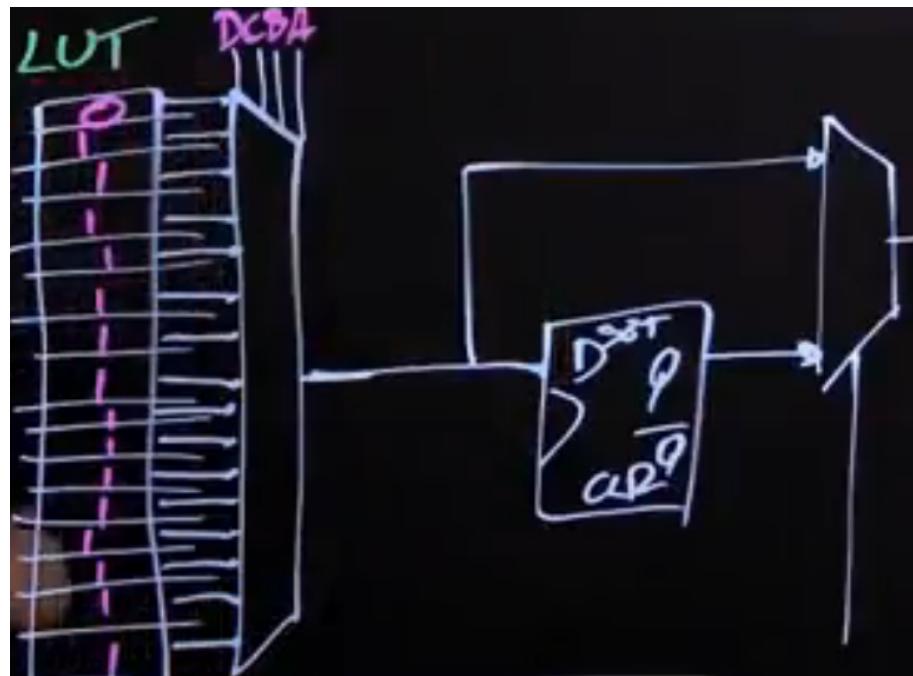
## FPGA (Field Programmable Gate Array)



FPGA je programovatelné hradlové pole, které je kompromisem mezi HW řešením - ASIC (Application specific integrated circuit - nejrychlejší, nejúspornější) a pouze SW řešení (největší flexibilita). Je tvořeno z:

- **Configurable Logic Blocks (CLBs)** — umožňují naprogramovat logické funkce realizované hardwarově. Jsou tvořeny součástkami, které dohromady v nejjednodušším případě mohou tvořit jeden **CLB** nebo **slice**. **CLB** může být tvořeno více **slice**:
  - Look Up Table (**LUT**):  $2^N$  bitový register obsahující výsledky kombinační logiky **N** proměnných. Jedná se volatilní paměť, tzn. konfigurace jednotlivých LUT musí být uložena v nevolatilní paměti (FLASH), ze které jsou LUT při spuštění nakonfigurovány.

- **$2^N-1$  multiplexor:** **N** určuje počet vstupních proměnných. Pomocí kterého se **vybírá** hodnota z **LUT** na výstup na základě hodnot **vstupních proměnných**.
- klopný obvod (typ **D**): vytváří paměť a umožňuje tvorbu sekvenčních obvodů.
- **2-1 multiplexor:** vybírá mezi výstupem z LUT (získaného pomocí  $2^N-1$  multiplexoru) nebo výstupem z klopného obvodu. Tento multiplexor tak určuje, jestli půjde o sekvenční nebo kombinační logiku.

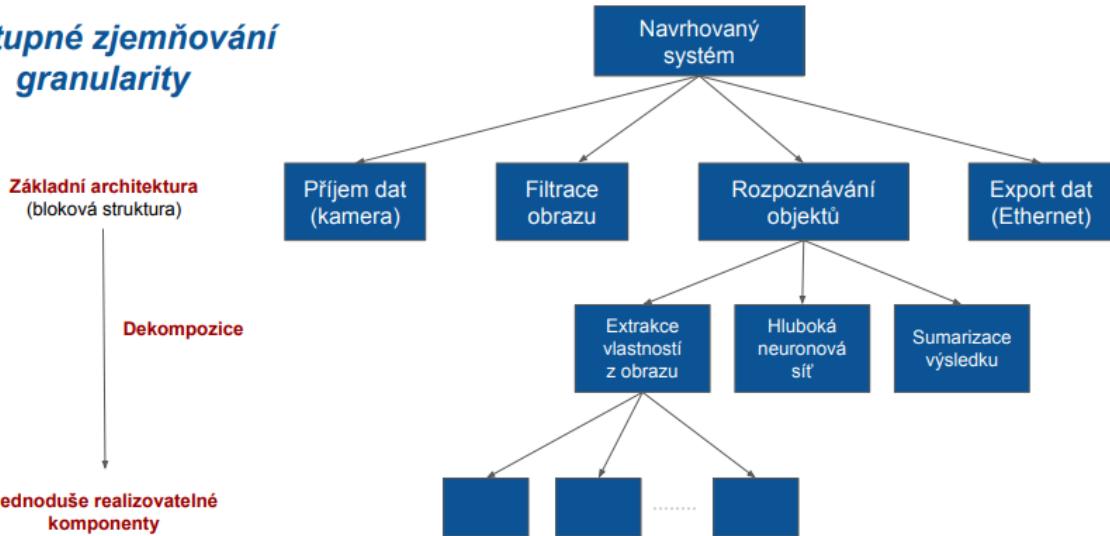


- případně další komponenty...
- **Programmable Interconnects** — programovatelné **propojení** mezi **CLBs**, jedná se o spoje vedené horizontálně a vertikálně. V místech křížení lze naprogramovat logiku propojování (**switch**).
- **Programmable I/O Blocks (IOBs)** — slouží k propojování **konfigurovatelných logických bloků** s externími součástkami pomocí propojovacích **pinů** FPGA čipu.
- **Vestavěné bloky** — paměti, sčítáčky, násobičky, ethernet, ...

## Kroky návrhu aplikací využívajících FPGA

Pro návrh aplikací využívající FPGA se používá systém návrhu **shora dolů (Top-Down)**. Začíná z celkového popisu problému a končí návrhem s úplnou mírou detailů jeho částí - **dekompozice** a začlenění do celku. Na každé úrovni abstrakce je potřeba **verifikovat správnou** funkci navrhovaného systému. Dekompozicí vzniknou **jednoduše realizovatelné komponenty** (sčítáčka, posuvný registr, násobička, čítač, ...), které lze popsát ve VHDL nebo Verilog. Např. zpracování obrazu:

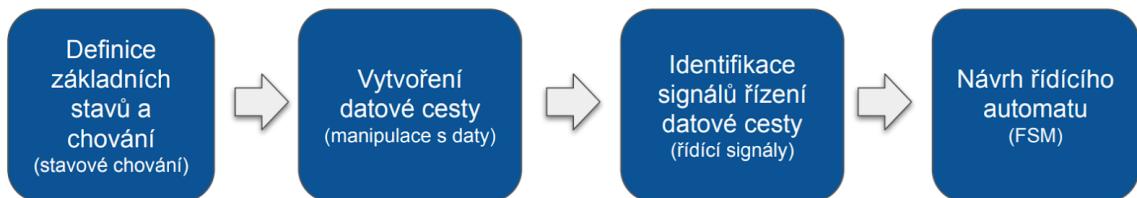
## Postupné zjemňování granularity



## Metodologie návrhu na úrovni meziregistrových přesunů (RT)

Návrh je vhodné rozdělit do několika kroků:

- **Popis obvodu na úrovni základního stavového automatu** - automat popisuje základní stavy obvodu, ale neřeší konkrétní nastavení řídících, vstupních nebo výstupních signálů.
- **Vytvoření datové cesty (datapath)** - datová cesta provádí manipulaci s daty, a to na základě řízení z obecného automatu.
- **Identifikace signálů pro řízení datové cesty** - definice signálů, které jsou potřeba pro napojení datové cesty na řídící blok.
- **Návrh řídícího automatu** - převedení obecného automatu na Mealyho nebo Moorův automat, který řídí datovou cestu



## Strukturální, behaviorální a dataflow popis ve VHDL

V praxi se typy popisů často kombinují, jedna logika lze současně popsat strukturálně, behaviorálně i pomocí dataflow. Pomocí syntézy (něco jako komplikace) jsou tyto popisy převáděny na konkrétní obvodová řešení pro danou technologii (FPGA, ASIC). VHDL umožnuje také pouze simulovat dané obvody.

### Strukturní popis

Při strukturním popisu **definujeme entitu** (např. poloviční sčítáčku), **propojením** vstupů a výstupů a **komponent** (AND a XOR hradla), které realizují logiku

komponenty. Samotné komponenty mohou být předtím popsány jako entity na nižší úrovni abstrakce. Strukturální popis umožňuje rozdělit komplexní systém na méně komplexní části, ty mohou být vyvíjeny a ověřovány samotně. Návrhář má u tohoto typu popisu kontrolu (pokud komponenty postupně popisuje od úrovně hradel nebo i níže) nad tím, jak bude výsledný obvod vypadat (z jakých bude prvků - hradel).

```

1. library ieee;
2. use ieee.std_logic_1164.all;
3.
4. entity half_adder is
5.   port (a, b: in std_logic;
6.         sum, carry_out: out std_logic);
7. end half_adder;
8.
9. architecture structure of half_adder is      -- Entity declaration for half adder
10.
11. component xor_gate
12.   port (i1, i2: in std_logic;
13.         o1: out std_logic);
14. end component;
15.
16. component and_gate
17.   port (i1, i2: in std_logic;
18.         o1: out std_logic);
19. end component;
20.
21. begin
22.   u1: xor_gate port map (i1 => a, i2 => b, o1 => sum);
23.   u2: and_gate port map (i1 => a, i2 => b, o1 => carry_out);
-- We can also use Positional Association
--     => u1: xor_gate port map (a, b, sum);
--     => u2: and_gate port map (a, b, carry_out);
24. end structure;
25.
26.
27.
```

## Behaviorální popis

Popisuje chování systému/funkce/bloku algoritmicky. Jedná se o nejvíce abstraktní popis. Z behaviorálního popisu není přímo jasné, jaká bude implementace na úrovni hradel (HW realizace). Behaviorální popis prvku obsahuje jeden nebo více procesů, které se dějí paralelně. Popis uvnitř procesu se odehrává sekvenčně. V procesu se nastavují výstupy na základě hodnot vstupů.

```

1. library ieee;
2. use ieee.std_logic_1164.all;
3.
4. entity half_adder is
5.   port (a, b: in std_logic;
6.         sum, carry_out: out std_logic);
7. end half_adder;
8.
9. architecture behavior of half_adder is
10. begin
11.   ha: process (a, b)
12.   begin
13.     if a = '1' then
14.       sum <= not b;
15.       carry_out <= b;
16.     else
17.       sum <= b;
18.       carry_out <= '0';
19.     end if;
20.   end process ha;
21.
22. end behavior;
```

## Dataflow popis

Pomocí DataFlow popisu se modelují systémy na základě toho, jak jimi putují data. Tento způsob popisu využívá odpovídající implementace logických bran. DataFlow popis popisuje systém/prvek jedním nebo více přiřazeními paralelních signálů.

```
1. library ieee;
2. use ieee.std_logic_1164.all;
3.
4. entity half_adder is
5. port (a, b: in std_logic;
6.        sum, carry_out: out std_logic);
7. end half_adder;
8.
9. architecture dataflow of half_adder is
10. begin
11.    sum <= a xor b;
12.    carry_out <= a and b;
13. end dataflow;
```

## Logická syntéza

Automatická transformace mezi různými úrovněmi popisu. Transformace na jemnější popis s cílem vylepšit parametry zadané uživatelem: rychlosť, spotřeba, rozměry.

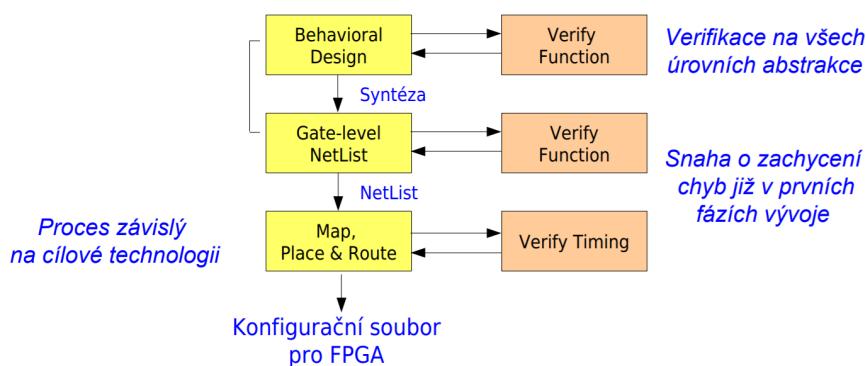
### Behaviorální syntéza

Z behaviorálního popisu algoritmu je vytvořena reprezentace na úrovni struktury (sčítáčka, posuvný registr, paměť, řídicí logika)

- Z HDL popisu na úrovni RT (meziregistrových přenosů) je vytvořen NetList prvků cílové technologie

### Logická syntéza

## | Design Flow pro technologii FPGA



Rozpoznání prvků cílové technologie a jejich mapování do FPGA. Výsledkem procesu je konfigurační soubor pro FPGA.

### Automatická syntéza:

- vstup: popis obvodu v HDL, knihovna prvků cílové technologie, constraints (např. spotřeba, velikost na čipu, čas...)

výstup: optimalizovaný NetList na úrovni prvků cílové technologie

# 11. 2D vektorová grafika: metody rasterizace úseček a polygonů, reprezentace objektů pomocí Bézierovy křivky

## Vektorová grafika

Zpracovávané a zobrazované informace popisujeme a **ukladáme analyticky** (spojitě) ve formě skupiny **vektorových entit** (**úsečky**, **kružnice**, **křivky**, **polygony**, atd.)

- Přesnost popisu je teoreticky neomezená.
- Vlastnosti uložených objektů (obrázků) lze kdykoliv jednoduše měnit.
- SVG

## Rastrová grafika

Zpracovávané a zobrazované informace popisujeme a **ukladáme diskrétně** ve formě **rastrové matice** (2D/3D) po **pixezech**.

- Nelze jednoduše měnit vlastnosti uložených objektů.
- Daná neměnná přesnost (rozlišení), nelze jednoduše měnit.
- PNG, JPG

## Rasterizace

Proces převodu vektorové reprezentace dat na rastrovou formu.

## Úsečka

Základní vektorová entita definovaná 2 body (konce). Všechny zmíněné algoritmy pracují **správně** pouze v **1. kvadrantu** a jen pokud úsečka **roste** rychleji ve směru osy X (má max sklon **45 stupňů**), ve směru **osy Y** také musí **růst**.

- **Obecný tvar** -  $Ax + By + C = 0$ , kde  $A = (y_2 - y_1)$ ,  $B = (x_2 - x_1)$ , **POZOR** vektor  $[A, B]$  je **normálovým vektorem** přímky.
- **Parametrický tvar** -  $x = x_1 + tA$ ,  $y = y_1 + tB$ , kde  $t \in [0, 1]$  a  $A = (x_2 - x_1)$ ,  $B = (y_2 - y_1)$ , vektor  $[A, B]$  je **směrovým vektorem** přímky.
- **Směrnicový tvar** -  $y = k*x + q$ , kde  $k = (y_2 - y_1) / (x_2 - x_1)$

## Algoritmus DDA (Digital differential analyzer)

Výpočet je ve **floating** point aritmetice. Postup je následující:

- Vykresluje úsečku po pixelech od bodu **P1** k bodu **P2**.

- V ose X postupujeme s přírůstkem **dx = 1**.
- V ose Y je přírůstek dán **velikostí směrnice** ( $k = (y2-y1)/(x2-x1)$ ) úsečky.
- Souřadnice Y se **zaokrouhuje** na **nejbližší celé číslo** - (přičít 0.5 před konverzí na int).

```
LineDDA(int x1, int y1, int x2, int y2)
{
    double k = (y2-y1) / (x2-x1);
    double y = y1;

    for (int x = x1; x <= x2; x++)
    {
        draw_pixel( x, round(y));
        y += k;
    }
}
```

### DDA s fixed-point aritmetikou

Používá bitový posun a eliminuje tak nutnost používat floating point, jinak pracuje na stejném principu jako floating point DDA.

```
#define FRAC_BITS 8

LineDDAFixed(int x1, int y1, int x2, int y2)
{
    int y = y1 << FRAC_BITS;
    int k = ((y2-y1) << FRAC_BITS) / (x2-x1);

    for (int x = x1; x <= x2; x++)
    {
        draw_pixel( x, y >> FRAC_BITS);
        y += k;
    }
}
```

### Bresenhamův algoritmus (midpoint algoritmus)

Algoritmus používá celočíselnou aritmetiku. Je jednodušší pro HW implementaci.

Dnes je tudíž **nejpoužívanější**. Postup:

- Vykresluje úsečku po pixelech od bodu **P1** k bodu **P2**.
- V ose X postupujeme s přírůstkem **dx = 1**.
- O posunu v ose Y rozhodujeme podle znamenka tzv. **prediktoru**.

## Rozhodování a výpočet chyby vykreslování $E$

$$E_i + \frac{\Delta y}{\Delta x} \begin{cases} < 0.5 & (x_i + 1, y_i) \\ \geq 0.5 & (x_i + 1, y_i + 1) \end{cases} \quad E_{i+1} = E_i + \frac{\Delta y}{\Delta x}$$

$$E_{i+1} = E_i + \frac{\Delta y}{\Delta x} - 1$$

Převod porovnání s 0.5 na test znaménka

- Nerovnice násobíme  $2\Delta x$

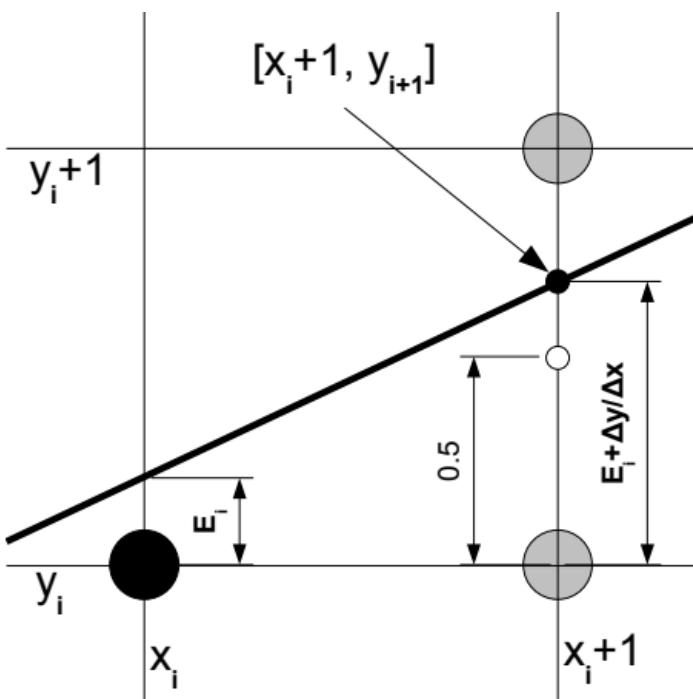
$$2\Delta x E_i + 2\Delta y - \Delta x \begin{cases} < 0 & E_{i+1} = E_i + 2\Delta y \\ \geq 0 & E_{i+1} = E_i + 2\Delta y - 2\Delta x \end{cases}$$

- Rozhodovací člen nazveme **prediktorem**  $P_i$

$$P_i = 2\Delta x E_i + 2\Delta y - \Delta x \begin{cases} < 0 & P_{i+1} = P_i + 2\Delta y \\ \geq 0 & P_{i+1} = P_i + 2\Delta y - 2\Delta x \end{cases}$$

## Počáteční hodnota predikce ( $E_0 = 0$ )

$$P_0 = 2\Delta y - \Delta x$$



```
LineBres(int x1, int y1, int x2, int y2)
{
    int dx = x2-x1, dy = y2-y1;
    int P = 2*dy - dx;
    int P1 = 2*dy, P2 = P1 - 2*dx;
    int y = y1;

    for (int x = x1; x <= x2; x++)
    {
        draw_pixel( x, y);
        if (P >= 0)
            { P += P2; y++; }
        else
            P += P1;
    }
}
```

## Kružnice

Definována souřadnicí **středu** a **poloměrem**:

$$(x - s_1)^2 + (y - s_2)^2 = r^2$$

Výpočty se provádí pro 1/4 kružnice, zbylá část se dopočítá díky **symetrii**. Algoritmy jsou odvozeny pro kružnici se **středem v počátku [0, 0]**.

Vykreslení po bodech (plyne z rovnice kružnice)

Nejjednodušší pro pochopení a implementaci, ale náročné pro HW zpracování. Pracuje se s **desetinnými** čísly (floating point). Vykresluje se oktant  $[x, y]$  na obrázky. **cx** je **x-ová** souřadnice **středu** a **cy** je **y-ová** souřadnice **středu**. Algoritmus vykresluje ve směru hodinových ručiček následovně:

- Jdeme po pixelu od bodu  $[0, R]$ , dokud není  $x = y$ .
- V **ose X** postupujeme s přírůstkem  **$dx = 1$** .
- Pozici v **ose Y** vypočteme po každé změně **X** podle vztahu  $y = \sqrt{r^2 - x^2}$
- Souřadnice **Y** se **zaokrouhuje** na **nejbližší celé** číslo - (přičítat **0.5** před konverzí na **int**).

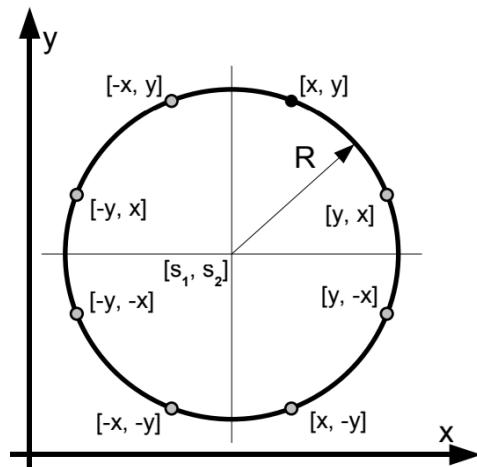
CircleByPoints(int cx, int cy, int R)

```
{
    int x = 0, y = R;
```

```
while (x <= y)
```

```
{
    draw_pixel_circle(x + cx, y + cy);
    x++;
    y = round(sqrt(R*R - x*x));
}
```

```
}
```

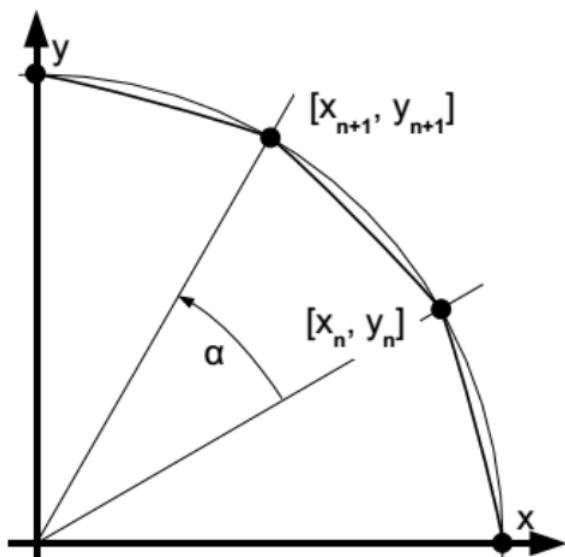


Vykreslení jako n-úhelník

Použitím DDA algoritmu se kružnice vykreslí jako by to byl **n-úhelník** (s větším poloměrem je potřeba větší  $n$ ) pomocí **úseček**. Využívá k tomu matici otočení

$$x' = x \cos \alpha - y \sin \alpha$$

$$y' = x \sin \alpha + y \cos \alpha.$$



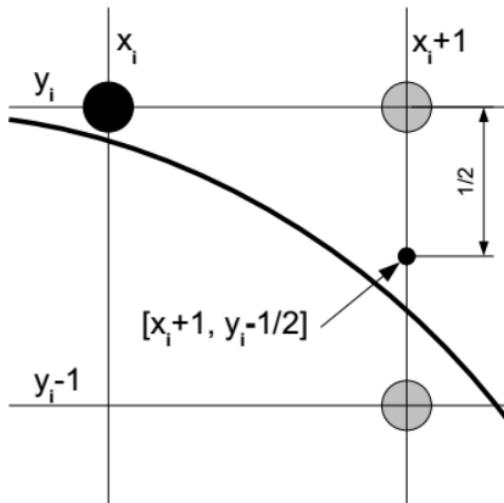
CircleDDA(int R, int N)

```
{
    double cosa = cos(2*PI/N);
    double sina = sin(2*PI/N);
    int x1 = R, y1 = 0, x2, y2;

    for (int i = 0; i < N; i++)
    {
        x2 = x1*cosa - y1*sina;
        y2 = x1*sina + y1*cosa;
        draw_line(x1, y1, x2, y2);
        x1 = x2;
        y1 = y2;
    }
}
```

## Midpoint algoritmus

Bresenham ale pro kružnici. Pracuje se s celými čísly, snadná implementace.



```
CircleMid(int s1, int s2, int R)
{ int x = 0, y = R;
  int P = 1-R, X2 = 3, Y2 = 2*R-2;

  while (x < y)
  {
    draw_pixel_circle(x, y);

    if (P >= 0)
      { P += -Y2; Y2 -= 2; y--; }

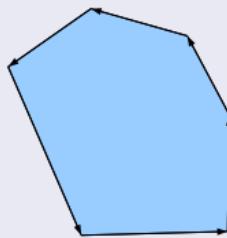
    P += X2;
    X2 += 2;
    x++;
  }
}
```

## Polygon (vyplňování uzavřených oblastí)

Proces nalezení a označení (obarvení) všech vnitřních bodů dané oblasti. Vstupem je ohrazení oblasti, výstupem je rastrový popis vyplněné oblasti.

### Konvexní (vypouklé, vyduté)

Pro libovolné dva body oblasti platí, že jejich spojnice je součástí oblasti, neprotíná její hranice.



### Konkávní (nekonvexní, prohnuté, duté)

Oblast, která není konvexní.



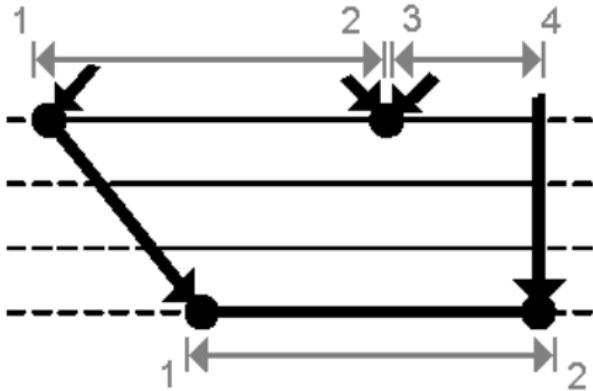
Druhy výplní: barvou (solid), šrafou (hatch), texturou, gradientem

### Řádkové vyplňování (Scanline Fill)

Základní algoritmus pro vyplňování obecných mnohoúhelníků. Vstupem je **orientovaný seznam vrcholů**, výstupem vyrasterizovaný polygon (**mnohoúhelník**). Princip rasterizace je následující:

- Vyhledávají se **maximální a minimální** hodnoty souřadnic **X** a **Y** těchto bodů, čímž se **ohraničí oblast**, ve které se polygon nachází.
- Oblast se prochází **po řádcích** (na konci řádku se **resetuje** čítač hran).
- Počítají se průsečíky s hranami. Pokud je aktuální **počet** průsečíků **lichý**, řádek se **obarvuje**.

Problémy jsou s **lokálními extrémy** a **vodorovnými hranami**. Vodorovné hrany se **přeskakují**, problém lokálních extrému lze řešit tak, že jsou všechny hrany **zkráceny** ve směru osy Y z obou stran o **1 pixel**, nebo že je **zaapočítán** v jednom bodě **průsečík obou hran**.

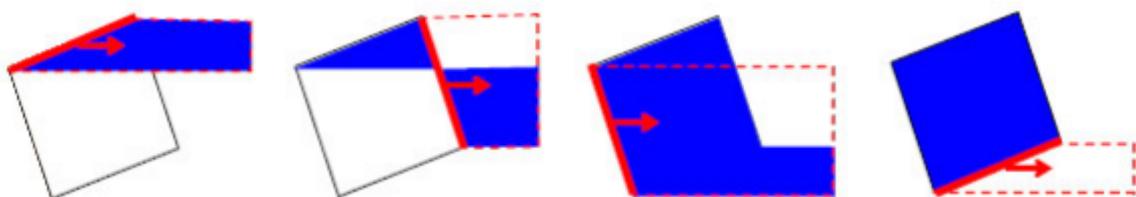


### Inverzní řádkové vyplňování

Vstupem je seznam hran nebo orientovaný seznam vrcholů. Není nutné na každém řádku testovat hrany polygonu. Postup je následující.

- Vyhledávají se **maximální a minimální** hodnoty souřadnic X a Y těchto bodů, čímž se **ohraničí oblast**, ve které se polygon nachází.
- Pro každou hranu se získá souřadnice **Ymin** a **Ymax**.
- Pro každou hranu se hledají **průsečíky s jednotlivými řádky** (X-ové souřadnice), pixely **napravo** od průsečíku se invertují (0 černá, 1 bílá).
- Překreslení obrysu.

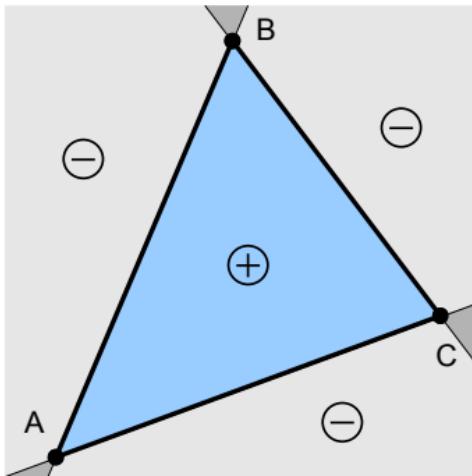
Po provedení postupu pro všechny hrany bude vybarven **pouze vnitřek** polygonu. Problém může být v zasažení oblasti mimo polygon. Umožňuje pouze binární obraz.



### Pinedův algoritmus (J. Pineda, 1988)

Pracuje pouze s **konvexními** mnohoúhelníky - trojúhelníky (ty jsou **vždy** konvexní). Umožňuje snadnou realizaci v HW. Vyplňovaná oblast je popsána seznamem hran a vyplňování probíhá:

- **Rozdělení oblasti** každou hranou na **poloroviny (kladnou a zápornou)**.
- **Vybarveny** jsou body (pixely) oblasti, které leží v **kladné polorovině** všech hran.



## Hranová funkce $E_i(x, y)$

Vekt. součin vektoru  $\vec{h}_i$  hrany a vektoru  $\vec{b}_{Pi}$  z počátku hrany k testovanému bodu  $P$ .

$$\begin{aligned}\vec{h}_i &= (x_{i1} - x_{i0}, y_{i1} - y_{i0}) = (\Delta x_i, \Delta y_i) \\ \vec{b}_{Pi} &= (x - x_{i0}, y - y_{i0}) \\ E_i(x, y) &= \vec{h}_i \times \vec{b}_{Pi} \\ E_i(x, y) &= (x - x_{i0})\Delta y_i - (y - y_{i0})\Delta x_i\end{aligned}$$

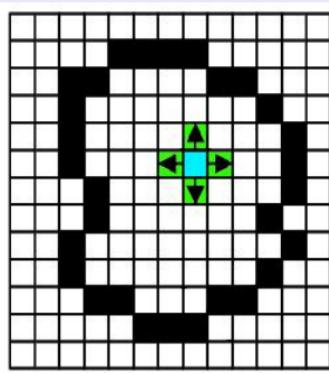
## Semínkové vyplňování

Vyplňovaná oblast musí být definovaná **spojitou hranicí** z pixelů **požadované barvy**. Obravují se pouze pixely s barvou pozadí následovně:

- Semínko uvnitř oblasti **šíříme** na sousedy v okolí (**obarvování** sousedních pixelů).
- Obarvené pixely se **rekurzivně** stávají semínky (problém rekurze - může selhat, lze využít frontu a rekurzi odstranit).

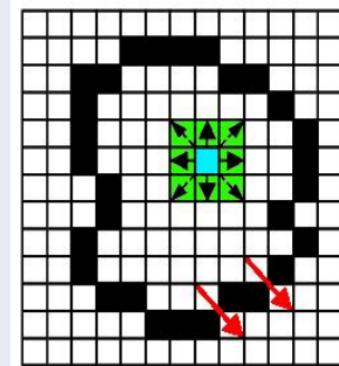
používají se dva druhy okolí **4-okolí** a **8-okolí**, 8-okolí vyžaduje lépe definovanou hranici.

### 4-okolí



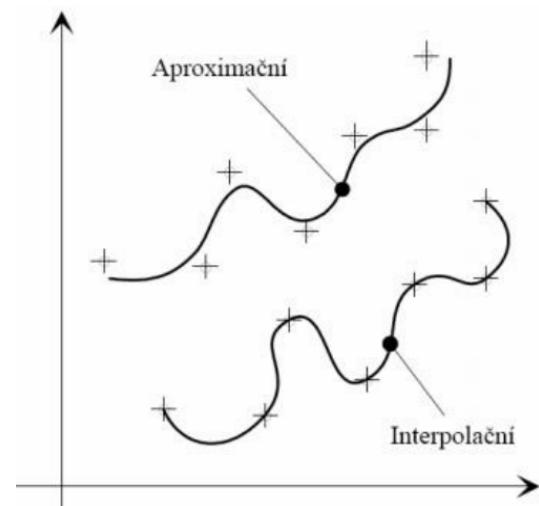
### 8-okolí

Vyžaduje "silnější" hranice.



## Křivky

- **Interpolaci** - Křivka prochází stanovenými (řídícími) body.
- **Aproximaci** - Křivka nemusí procházet řídícími body - Beziérové křivky.



## Beziérový křivky

- approximační křivka (2D grafika, fonty)
- polynomiální křivka využívající Bernsteinových polynomů,
- křivka stupně  $n$  určena  $n+1$  body,
- prochází koncovými body.

**Rekurentní definice s použitím Bernsteinových polynomů.** Používá se pro fonty.

$$Q(t) = \sum_{i=0}^n P_i \cdot B_i^n(t); \quad i = 0, 1, \dots, n$$

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}; \quad t \in [0, 1]$$

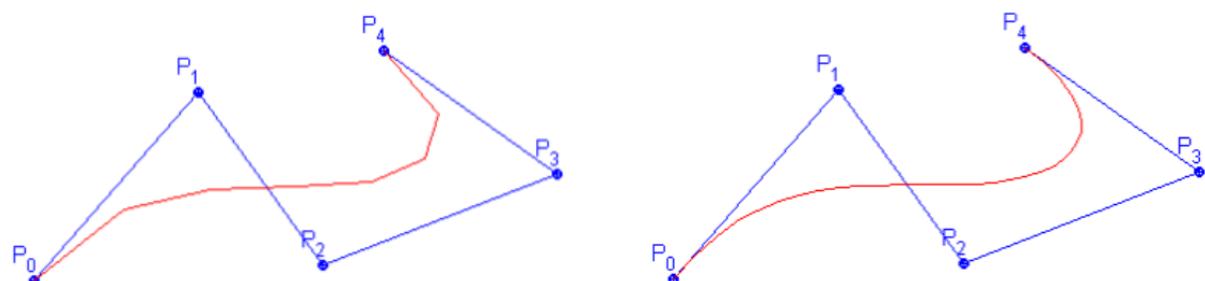
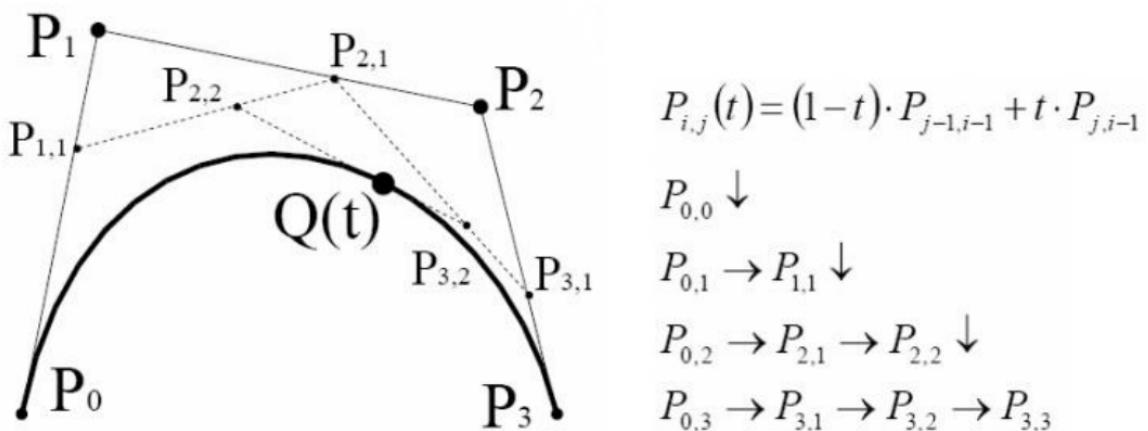
$B_0(t) = (1-t)^3$
$B_1(t) = 3t(1-t)^2$
$B_2(t) = 3t^2(1-t)$
$B_3(t) = t^3$

$$P(t) = P_0 B_0(t) + P_1 B_1(t) + P_2 B_2(t) + P_3 B_3(t) = \sum_{i=0}^3 P_i B_i(t)$$

## Algoritmus de Casteljau

**Rekurzivní** algoritmus pro vykreslování **Beziérových křivek** (plyne z rekurentní definice Bernsteinových polynomů). Postup:

- Zvolí se dostatečně jemný krok, se kterým se mění hodnota  $t$  ( $t$  náleží  $[0, 1]$ ).
- Úseky polynomů se dělí v poměru  $t$  a  $t-1$ , v místě dělení vznikne **nový bod** pro další dělení. **Snižuje** se tak s každým krokem **stupeň polynomu**, až na konci zbyde pouze jeden **bod**.
- Vzniklé body se **spojí úsečkami** (tvar křivky závisí na použitém kroku).

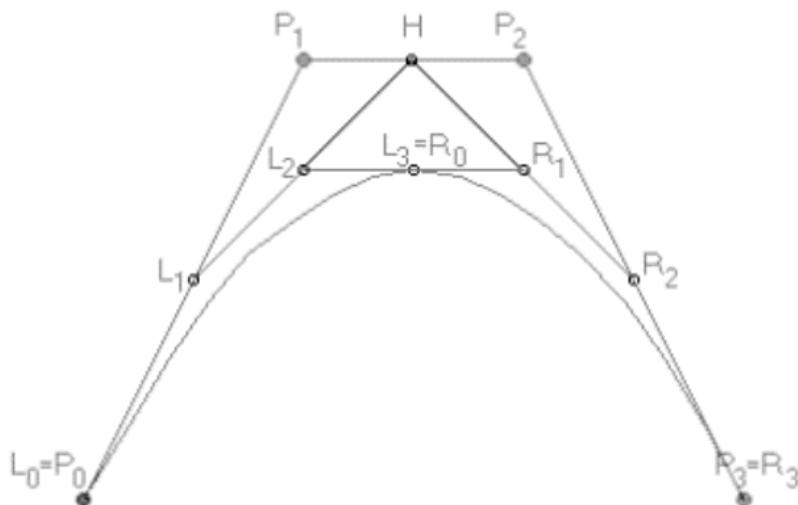


## Bézierovy kubiky

Beziérova křivka **3. stupně** popsaná Bernsteinovými polynomem 3. stupně.

Segment je popsaný **4 řídícími body**.

- **Divide and Conquer - De Casteljau** a Bézierovy kubiky. Rekurzivní dělení na **2 podkřivky**, dostatečně rovná křivka se dál neděli a vykreslí se.



## Navazování segmentů Beziérových kubik

Vyžaduje **totožný koncový body** a **stejné tečné vektory** v navazujícím bodě.

**Koncový bod je středem úsečky mezi předposledním bodem první křivky a druhým bodem druhé křivky.**

### Tečné vektory v koncových bodech

$$P'(\vec{0}) = 3(P_1 - P_0)$$

$$P'(\vec{1}) = 3(P_3 - P_2)$$

### Podmínky spojitosti $C^1$

- Totožnost koncových bodů.
- Shodné tečné vektory – koncový bod úseku  $Q_i$  je středem úsečky předposledního bodu  $Q_i$  a druhého bodu  $Q_{i+1}$ .

### Odkazy:

- [Modelování křivek](#)
- [Animace Bézierova křivka](#)
- [De Casteljau animace \(video\)](#)
- [De Casteljau's Algorithm and Bézier Curves](#)

# 12. Transformace a zobrazení 3D polygonálních modelů, principy programovatelného vykreslovacího řetězce.

## Geometrická transformace

změna pozice **vrcholů** objektů v aktuálním souřadnicovém systému nebo změna souřadnicového systému

### Lineární transformace

Lineární transformace **zachovává lineární kombinaci** (vektory je možné aplikovat zároveň nebo po jednom a výsledek bude vždy stejný). Pro **libovolné** dva vektory a **skalár** platí:

$$\begin{aligned} f(\vec{x_1} + \vec{x_2}) &= f(\vec{x_1}) + f(\vec{x_2}), \\ f(\alpha \vec{x_1}) &= \alpha f(\vec{x_1}). \end{aligned}$$

Mezi lineární transformace patří **měřítko** (zvětšení, zmenšení), **rotace** a **zkosení**.

### Afnní transformace

Afnní transformace zachovává **kolinearitu a dělící poměr** (body ležící na přímce budou ležet na přímce - v jednom bodě; i po zobrazení). Všechny základní geometrické operace (**měřítko Sc**, **rotace R**, **zkosení Sh** i **posunutí T**) jsou afnní. Lze ji vyjádřit jako lineární transformaci následovanou posunem.

### Homogenní souřadnice (váha)

Ke standardním **kartézským** souřadnicím (x, y ve 2D a x, y, z ve 3D) je přidána jedna navíc - souřadnice **w** (váha bodu, u **afnních** je **w = 1**). Umožňují pracovat se **všemi** druhy základních **transformací** jednotně pomocí **maticového zápisu**.

Maticový zápis umožnuje provádět jednoduché skládání transformací.

- **Posunutí** (translace): Ve **3D** stačí rozšířit o řádek a sloupec s **dz**, resp. **-dz**.

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ 1 \end{pmatrix} = \begin{pmatrix} v_x + t_x \\ v_y + t_y \\ v_z + t_z \\ 1 \end{pmatrix}$$

### Maticový zápis transformace

$$[x', y', 1] = [x, y, 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ d_x & d_y & 1 \end{bmatrix}$$
$$P' = P \cdot T$$

### Maticový zápis inverzní transformace

$$[x', y', 1] = [x, y, 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -d_x & -d_y & 1 \end{bmatrix}$$
$$P' = P \cdot T^{-1}$$

- **Změna měřítka** ve 3D (scale):  $P' = P \cdot S$ , ve 3D stačí rozšířit o řádek s  $S_z$ , resp.  $1/S_z$ .

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad S^{-1} = \begin{bmatrix} 1/S_x & 0 & 0 \\ 0 & 1/S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- **Zkosení**: 1 matice pro zkosení ve 2D, 3 matice pro zkosení ve směrech YZ, XZ a XY ve 3D.

$$S_H = \begin{bmatrix} 1 & S_{hy} & 0 \\ S_{hx} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad S_H^{-1} = \begin{bmatrix} 1 & -S_{hy} & 0 \\ -S_{hx} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S_{HYZ} = \begin{bmatrix} 1 & S_{hy} & S_{hz} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad S_{HXZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ S_{hx} & 1 & S_{hz} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$S_{HXY} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ S_{hx} & S_{hy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Rotace kolem počátku souřadného systému**: Ve 3D 3 matice

$$[x', y', 1] = [x, y, 1] \cdot \begin{bmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} R = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R^{-1} = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P' = P \cdot R$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha & 0 \\ 0 & -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y = \begin{bmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 & 0 \\ -\sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Rotace ve 3D kolem obecné osy:** dáná směrovým vektorem  $\mathbf{v}$  a bodem umístění  $\mathbf{P}$ , je třeba rozdělit na posloupnost transformací:
  - 1. Posunutí osy rotace do počátku souřadného systému
  - 2. Sklopení posunuté osy do jedné ze souřadných rovin
  - 3. otočení sklopené osy do jedné ze souřadných os (např. do X)
  - 4. Provedení požadované rotace o úhel  $\omega$  kolem příslušné osy (zde X)
  - 5. Vrácení osy rotace do původní polohy

$$M = T \cdot R_X \cdot R_Z \cdot R_{X(\omega)} \cdot R_Z^{-1} \cdot R_X^{-1} \cdot T^{-1}$$

Zkráceně musíme nejdřív obecnou osu dostat do pozice jedné z os souřadného systému, provést otočení a vrátit osu do původního místa.

## Skládání transformací

U skládání transformací závisí na jejich pořadí, např. **posunutí, rotace, zvětšení**.

- **zápis bodu v řádku:**  $\mathbf{P}' = \mathbf{P} \cdot T \cdot R \cdot S$ , matici M souhrnné transformace získáme jako  $M = T \cdot R \cdot S$  a  $\mathbf{P}' = \mathbf{P} \cdot M$ .
- **zápis bodu ve sloupci:**  $\mathbf{P}' = S \cdot R \cdot T \cdot \mathbf{P}$ , matici M souhrnné transformace získáme jako  $M = S \cdot R \cdot T$  a  $\mathbf{P}' = M \cdot \mathbf{P}$ .

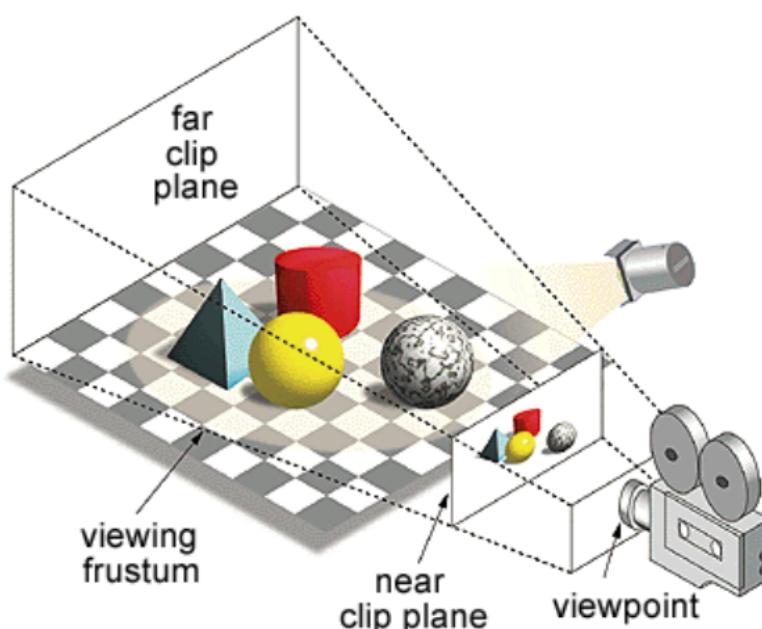
V obou případech musí být první transformace v pořadí **nejblíž transformovanému bodu**.

## zobrazení 3D polygonálních modelů

Zobrazení **3D** objektů provádíme většinou na **2D** obrazovku **projekcí** (transformací z 3D do 2D, to znamená ztrátu dat), obrazovka je tedy **průmětna**. Promítání provádíme pomocí paprsků, tzv. **projekčních paprsků**. V počítačové grafice se většinou rasterizují všechny objekty **pomocí trojúhelníku** (konvexnost, lze dobře akcelerovat v HW).

## Prvky 3D scény

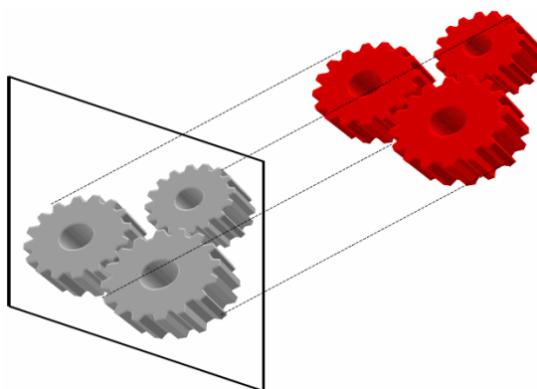
- **near clip plane:** určuje minimální hloubku zobrazovaných objektů,



- **far clip plane**: udává maximální hloubku zobrazovaných objektů,
- **viewing frustum**: prostor v modelovaném prostředí, který se objeví na obrazovce,
- **viewpoint**: místo, ze kterého scénu pozorujeme (kamera)
- **světlo**: bodové nebo plošné, umístění rozhoduje o viditelnosti objektů (barvy, stíny, ...),
- **modelované objekty**.

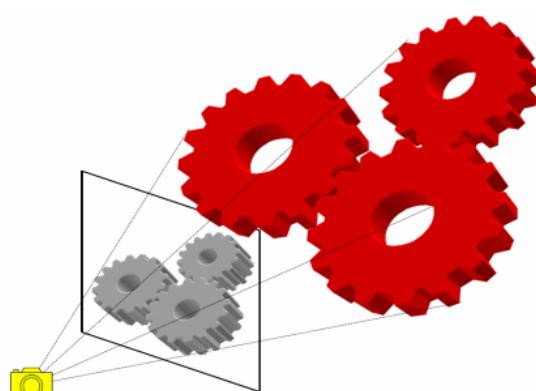
### Paralelní projekce (**rovnoběžná**, ortografická)

- **zachovává rovnoběžnost hran**
- **vzdálenost** průmětny **neovlivňuje velikost** průmětu (při změně vzdálenosti objektu se nemění jeho velikost)
- **kolmé promítání** - paprsky jsou **kolmé na průmětnu**
- použití: technické **CAD** aplikace, výkresová dokumentace



### Perspektivní projekce (**středová**)

- **nezachovává rovnoběžnost** hran
- použití ve **hrách, VR**, většinou
- **vzdálenost** průmětny od objektu **ovlivňuje velikost** průmětu (se vzdáleností objektu se objekt zmenšuje)
- nelineární středová projekce: paprsky **vycházejí z 1 bodu - středu projekce**
- pro jednodušší manipulaci se **kamera** obvykle **zafixuje** do počátku souřadného systému a **hýbe** se (pomocí transformačních matic) se **scénou**.
  - **Geometrický princip projekce**: z vrcholů trojúhelníku se do středu projekce (tj. střed souřadného systému, kam je umístěna i kamera) vrhnou paprsky (u perspektivní projekce se všechny paprsky sbíhají do středu projekce). V tom místě, kde paprsek protnul projekční rovinu se promítne bod, ze kterého paprsek původně vyšel.



## RayTracing

Metoda pro realistické zobrazování. Funguje na principu zpětného sledování cest paprsku od kamery ke zdroji světla. Hledá se množství světla, které paprsek přináší. Paprsky dělíme na:

- **Primární:** Vychází z kamery (pixely obrazovky), mají společný počátek nebo jsou rovnoběžné
- **Stínové:** Z místa dopadu paprsku do každého světla, zajímá nás, jestli jej vidíme nebo nevidíme, podle toho následně spočítáme stín.
- **Sekundární:** Vznikají **odrazem a lomem** z míst **dopadů primárních** a sekundárních paprsků. Jsou chaotičtější než primární paprsky.

Výpočetně je RayTracing velmi náročný, lze optimalizovat například ohledem na to, že sekundární paprsky mají **menší vliv** na výsledný obraz, **intenzita světla** se snižuje se **čtvercem** vzdálenosti, **snížit rozlišení**.

## Radiozita

Metoda **globálního osvětlení scény**. Řeší **šíření energie**, objekty mohou být sekundárními zdroji světla (odraz).

- Vychází ze zákona zachování energie, vyžaduje energeticky uzavřenou scénu.
- Scéna musí být reprezentovaná **polygonálním modelem** (jiné modely - CSG, B-rep, Drátové modely).
- Vychází z dvousměrové distribuční funkce **BRDF**.
- Před vlastním výpočtem je třeba **polygony** ve scéně **rozdělit** na **malé plošky** a spočítat **konfigurační faktory** (**vliv** každé **plošky** na každou **jinou plošku** ve scéně).
- Nedokáže pracovat s průhlednými objekty.

## Principy programovatelného vykreslovacího řetězce (zobrazovací pipeline)

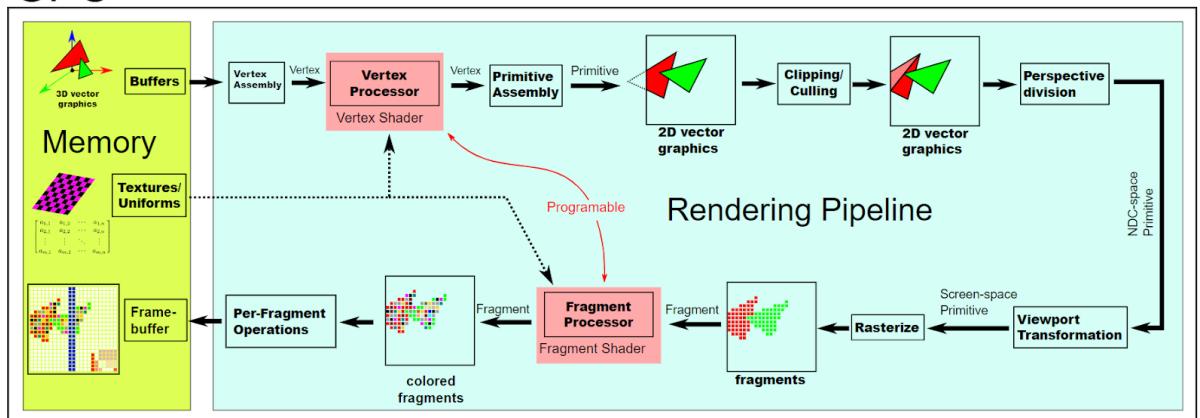
zobrazovací pipeline je tvořena těmito částmi:

1. **Vertex Assembly (Vertex Puller):** je zařízení na grafické kartě, které se stará o **sestavení vrcholů**. Vertex puller je tvořen **čtecími hlavami**, každá konstruuje jeden atribut vrcholu (pozice, normála, souřadnice textury, barva, ...). Čtecí hlavy se pohybují s krokem (**stride**) a mají nějaký posun (**offset**). Data čtou z pole bytů.
2. **Vertex Shader** (programovatelný): Provádí zpracování vrcholů z Vertex Assembly. Jedná se o násobení **modelovou maticí**, **view maticí** a **projekční maticí**. Vstupem jsou vrcholy v **model space**, výstupem vrcholy v **clip space**.
3. **Primitive Assembly:** je jednotka, která sestavuje trojúhelníky. **Čeká na 3** po sobě jdoucí **vrcholy** z vertex shaderu a **sestaví trojúhelník**. Lze na to také

nahlížet tak, že Primitive Assembly dostane příkaz vykreslit třeba 4 trojúhelníky. Jednotka tak spustí Vertex Shader 12x, který takto spustí 12x Vertex Assembly.

4. **Clipping/Culling**: provádí **ořez prostoru** na view frustum. Trojúhelníky na **hranicích** musí ořezat (může vést na rozdělení na 2) a zahazuje trojúhelníky, které nejsou vůbec vidět (odvrácené, překryté).
5. **Perspective Division**: provádí převod z homogenních souřadnic na kartézských na základě hloubky trojúhelníků (dělením - co je daleko se zmenší více, co je blízko se zmenší méně).
6. **Viewport Transformation**: Převádí souřadnice z NDC (normalized device coordinates, **-1, +1**) na rozlišení okna, aby se mohla provést rasterizace.
7. **Rasterization**: rasterizuje připravené trojúhelníky a produkuje **fragmenty** (čtvercové úlomky trojúhelníku - **pixely**, které se nakonec **zapíší** do **framebufferu**).
8. **Fragment Shader (programovatelný)**: **obravuje fragmenty** uvnitř trojúhelníka (střed pixelu musí ležet uvnitř). Pro konkrétní fragment (pixel trojúhelníku) se spočítají **barycentrické souřadnice** a použijí se pro **interpolaci** všech atributů z **vrcholů**.
9. **Per-Fragment Operations**: jedná se o dvě operace **hloubkový test** a **blending**. Hloubkový test se stará o **zahazování fragmentů**, které jsou **hlouběji** než to, co už se **vyrasterizovalo**, naopak pokud je **hloubka** nového fragmentu **menší**, je jeho **barva a hloubka zapsána do framebufferu**. **Blending** místo přepsání barvy ve framebufferu je míchá. Existuje mnoho způsobů realizace, např. pomocí **průhlednosti (alpha blending)**. Obsah framebufferu je poté zobrazen.

## GPU



# 13. Principy grafických uživatelských rozhraní (komunikační kanály, módy komunikace, systémy řízené událostmi, standardní prvky rozhraní).

Komunikaci člověka se strojem lze definovat jako **obousměrnou výměnu informací** mezi člověkem a strojem. Tok informací od počítače k člověku lze uskutečnit pomocí **periferních** výstupních zařízení generujících výstupy, na které mohou reagovat **smysly člověka**. Tok údaj od člověka k počítači lze naopak uskutečnit pomocí vstupních periferních zařízení, která mohou **reagovat na podněty** generované člověkem.

## Komunikační kanály

Komunikační kanály jsou způsoby **komunikace člověka a stroje** pomocí periferních zařízení - **monitor, klávesnice, tiskárna, myš, kamera, ...** Jsou **založené na lidských smyslech**.

### Komunikační kanály od stroje k člověku

- **Obraz (zrak)** - Nejvýhodnější pro přenos informace. **Nejvyšší informační propustnost**.
- **Zvuk (sluch)** - Vhodný pro přenos menšího počtu informací. Nižší propustnost a "sériový" přístup (je obtížné poslouchat 2 věci zároveň). Zvuk na sebe však může **lépe upozorňovat** - varovné signály.
- **Hmat** - Využívá se pro komunikaci **nevidomých** a stroje (braillovo písмо). Běžní uživatelé se nejčastěji setkávají s tímto druhem komunikace na mobilních zařízeních, formou **vibrací** (náhrada za pocit stisknutí tlačítka) a také na ovladačích her (např. signalizace překážky, **silový odpór** u závodního volantu).
- **Čich, chuť** - V současnosti **nejsou** pro komunikaci **použitelné**. Problémy s umělým syntezováním chuti a vůně v reálném čase.

### Komunikační kanály od člověka ke stroji

- **Pohyb (Hmat)** - Nejobvyklejší prostředek - **klávesnice**, tlačítka, přepínače, ... a **mechanické polohovací zařízení** - **myš**, páčky. (Dnes nejpoužívanější, ale neznamená to, že je nejlepší, pouze je obvykle nejjednodušší na implementaci - např. diktování textu může být pro většinu uživatelů příjemnější, než psaní na klávesnici).

- **Zvuk (Řeč)** - Rychle se vyvíjí, dnes už prakticky **použitelný** způsob ovládání, zejména v anglickém jazyce (asistenti na mobilu, speech to text apod.).
- **Problém** s rozpoznáváním řeči a náročnost zpracování, **soukromí**. Není vhodné pro sdělování citlivých informací (hesla, číslo OP., ...)
- **Obraz (gesta)** - Sdělování informací **gesty, pohyby a mimikou v obličeji** (filtry na Instagramu). Lze využít ve **virtuální realitě** a hrách (Just dance, Wii games...), gesta ruky pro přeskočení písničky na Android 10 (Google Pixel). Rozpoznávání osob a objektů - pro bezpečnost nebo klasifikaci.

## Módy komunikace

Nejzákladnějším dělením obrazové komunikace člověka s počítačem je dělení podle aktivity uživatele:

- **Aktivní komunikace** - Uživatel **řídí** činnost počítače - činnost počítače záleží na **vůli uživatele**. Například panel nástrojů v grafickém editoru, unixový terminál...
- **Pasivní komunikace** - Uživatel **odpovídá** na dotazy počítače. **Dialogová/modální okna**.

Je zřejmé, že komunikace s počítačem se **nebude trvale** odehrávat jen **pasivně** nebo jen **aktivně**, a že je vhodné oba způsoby **kombinovat**. **Aktivní** komunikaci je nutné používat tam, kde uživatel **musí rozhodnout**, jakou činnost má počítač vykonávat a kdy není možné rozhodnutí obsluhy předvídat. **Pasivní** komunikace je naopak vhodná tehdy, je-li příští **činnost** počítače **známá**, a obsluha jen zadává údaje nebo rozhoduje mezi několik málo možnými variantami, které lze přesně určit.

Vlastnost	Aktivní komunikace	Pasivní komunikace
Efektivní práce	? dle aplikace	- často neefektivní
Učení	- obtížné	+ snadné
Tvůrčí práce	+ snadná	- obtížná
Možnost chyby	- velká	+ malá

**Módy komunikace** - Stav ve kterém počítač reaguje jedinečným způsobem na vstup od uživatele (změny módů - změny chování počítače na vstupy, na klávesnici **Caps Lock, Insert**, grafický editor a levé tlačítko myši může kreslit, mazat, přesouvat objekty). Obecně čím méně modů tím lípe. Typické módy (např. Vim - i pro editaci):

- Zadání příkazu v příkazovém jazyce
- Odpověď na dotaz (potvrzení akce - např. smazání).
- Editace textu
- Reakce na chybu

## Přímá manipulace (Drag and Drop, Look and Feel)

Umožňuje interakci mezi uživatelem a objekty zobrazenými na obrazovce - podporuje **přirozené chování** uživatele. **Zjednodušuje ovládání** počítače pro neškolené uživatele a zlepšuje jejich dojem při práci s PC. Zjednoduší nároky na zkušeného uživatele, který nemusí vynakládat úsilí např. pro přesun souboru, ale provede jej intuitivně pomocí **Drag and Drop**. **Look and Feel** - uživatelské rozhraní se dá používat **intuitivním způsobem** vycházejícím z podobnosti s prací s předměty v běžném životě.

## Systémy řízené událostmi

Systémy, které se zabývají **detekcí, zpracováním a reakcí** na události. Tok programu je řízen různými událostmi (vstup z periferií - **zmáčknutí klávesy, pohyb myši**). Program naslouchá na tyto akce, a nějak reaguje (např. v JS pomocí listeners: onclick, onhover, onkeypress). U hardware je toto prováděno pomocí **přerušení**. Tento přístup je velmi využívaný. Programy, které takto pracují, běží v nekonečné smyčce (z té vystoupí po události signalizující ukončení) a čekají na příchod události (HW přerušení) nebo sami kontrolují její vznik (Polling), nebo probíhá formou zasílání zpráv. Detekuje se o jakou se jedná událost a spustí se její obsluha, po dokončení obsluhy se program dostává zpět do stavu, ve kterém čeká na další událost.

- QT: signal and slots,
- callbacks

## Standardní prvky rozhraní

Standardní prvky GUI obvykle označujeme pod zkratkou **WIMP**:

- **Window - okno**: reprezentují spuštěné programy:
  - **primární**: hlavní okno aplikace, obsahuje menu, více nezávislých funkcí,
  - **sekundární**: okno pro zpracování vedlejších, rozšiřujících či doplňkových funkcí, pevný rozměr, nemá min/maximalizaci, menší než primární okno. Může být nazývané jako **modální**, protože **mění mód/režim** chování aplikace/systému, mění pracovní postup. Může vyžadovat interakci uživatele, než vrátí řízení rodičovskému oknu (např. dialogová okna). Slouží k upozornění uživatele a blokování aplikace/systému pro získání klíčových informací (uživatel musí dokončit práci v modálním okně)
    - **modální** - dialog v rámci aplikace,
    - **systémově modální** - Mají prioritu nad aplikacemi (např. nedostatečná práva pro provedení akce, **potvrzení instalace**).
    - **nemodální** - Neprioritní. Uživatel může mít okno otevřené a stále pracovat s aplikací (výběr barvy štětce v malování například).

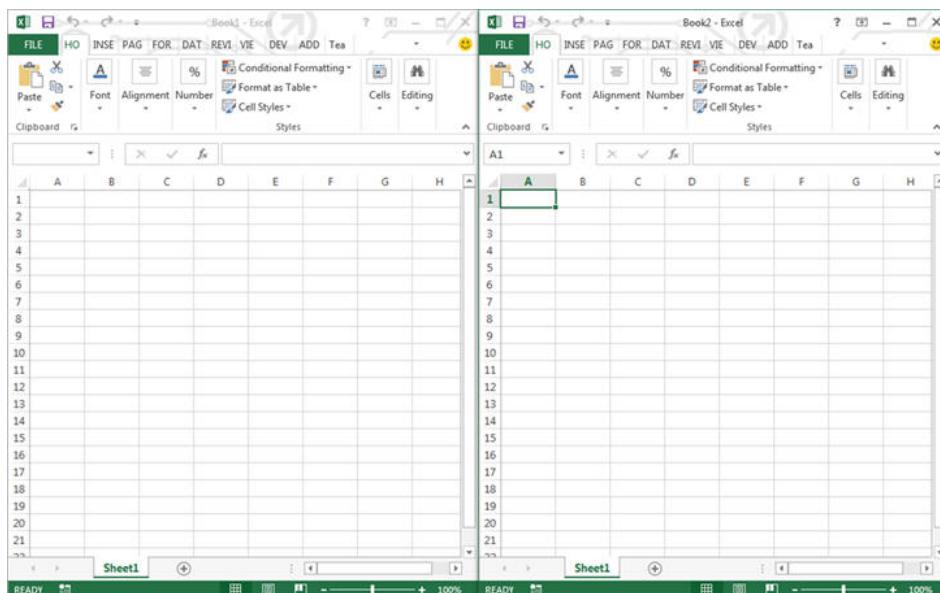
- **Icon - ikona:** reprezentují zkratky sloužící k provedení určité činnosti (například spuštění programu).
- **Menu:** textové nebo z ikon složené **nabídky**, ze kterých je možné jednu vybrat a provést tak určitou akci.
- **Pointer - ukazatel:** pohybující se grafický symbol reprezentující pohyb fyzického zařízení (myš), pomocí něhož uživatel vybírá ikony, položky v menu nebo data.

## Další prvku UI

- tlačítka,
- přepínače,
- popisky,
- checkboxy,
- seznamy,
- slidery,
- výběr souboru,
- vstupní pole pro text

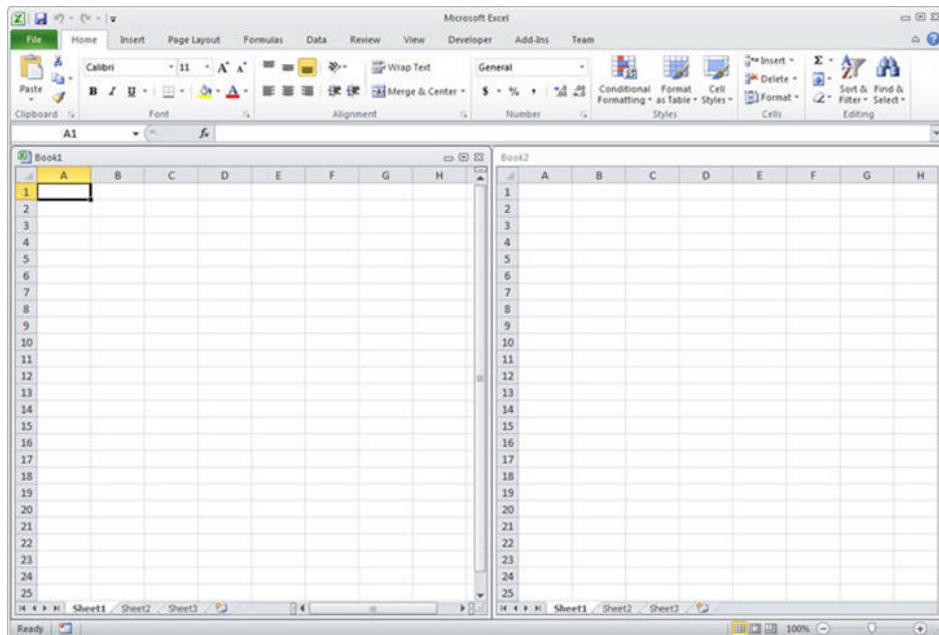
## Režim oken aplikace

- **Single-Document Interface (SDI):** Jedno primární okno, několik sekundárních. Jednoznačný vztah okna s objektem. Přehledné, srozumitelné. **Každé okno má své menu.** Několik instancí aplikace v liště OS.  
(Excel 2013)

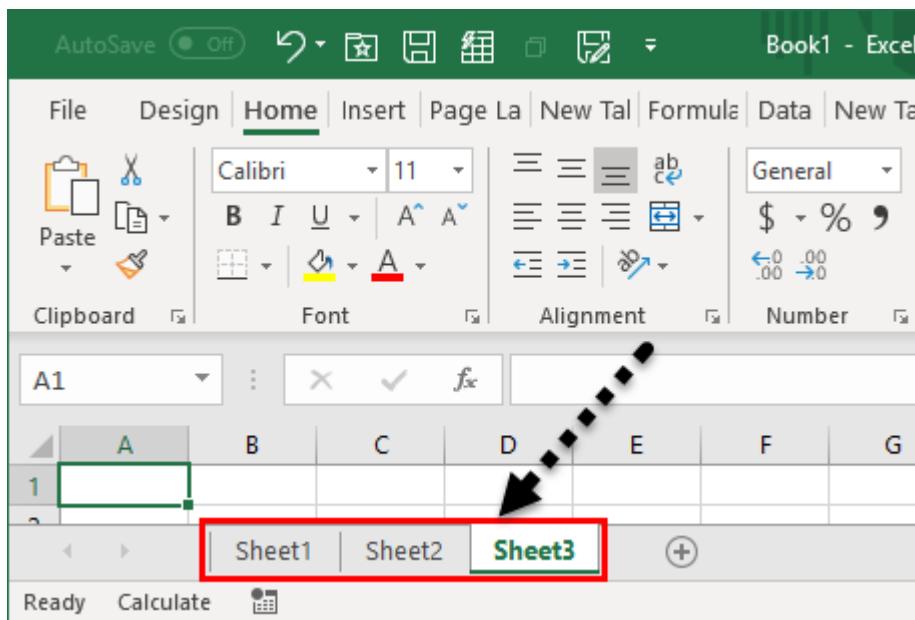


- **Multiple-Document Interface (MDI):** Jedna aplikace obsahuje více oken. Práce s jedním objektem z více pohledů (otevřený ve více oknech) nebo práce s více objekty současně (možnost při práci s jedním nahlížet na druhý).

Menší přehlednost a obtížnější zvládnutí.  
(Excel 2010)



- **Tabbed Document Interface (TDI)** - prohlížeče, editory zdrojových kódů, ...  
Dnes často používané. Řízené SDI - SDI doplněné o řídící okno, které obsahuje menu a seznam otevřených objektů (záložek). Kooperativní SDI - některé funkce mohou ovlivnit obsah i jiných oken.  
(jednotlivé dokumenty v záložkách)



Odkazy:

[https://www.fit.vutbr.cz/study/courses/ITU/private/lectures/zaklady/itu-zaklady\\_GUI\\_a\\_historie.pdf](https://www.fit.vutbr.cz/study/courses/ITU/private/lectures/zaklady/itu-zaklady_GUI_a_historie.pdf)

# 14. Spektrální analýza spojитých a diskrétních signálů.

**Signál** je záměrný fyzikální jev, nesoucí informaci o nějaké aktuální události. Jedná se o časově či prostorově proměnnou prostředí (**fyzikální veličina**).

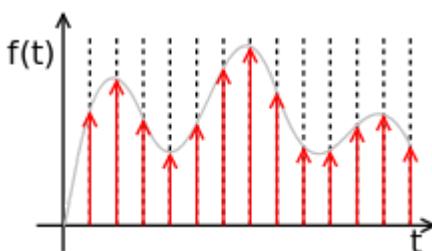
## Spojitý (analogový)

Vyskytují se v **reálném světě** (například zvukové vlny), mají pro každý časový okamžik určitou hodnotu, což tvoří souvislou (spojitou) křivku. Lze je zapsat funkcí. Značíme je  $x(t)$  a hodnoty spojitého signálu jsou z množiny **reálných čísel**.

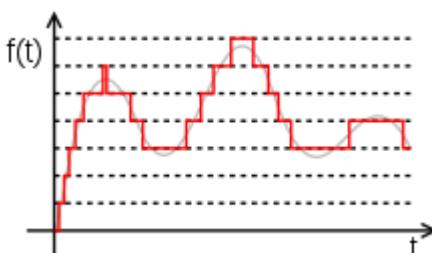
## Diskrétní

Jeho okamžitá hodnota se nemění spojité s časem ale **skokově**. Lze je ukládat a zpracovávat na **číslicových** počítačích. Značíme je  $x[t]$  a hodnoty diskrétních signálů jsou pouze z množiny **celých čísel** a navíc nabývají některé z **konečného počtu hodnot**. Pro získání diskrétního signálu **vzorkujeme** signály kolem nás, které jsou spojité. Následně signály **kvantujeme** - zaokrouhlujeme na čísla z oboru hodnot signálu (respektive na čísla, která dokáže reprezentovat v PC).

- **Vzorkování:** Periodicky zaznamenáváme hodnotu spojitého signálu. PerIODA zaznamenávání spojitého signálu musí být aspoň dvakrát menší, než nejmenší perioda některé ze složek spojitého signálu  
**(Nyquistův–Shannonův vzorkovací teorém**, jinak nelze signál rekonstruovat).



- **Kvantování:** diskretizace oboru hodnot signálu (převod z reálných čísel na celá). Je to obecně proces ztrátový a nevratný.



## Deterministické signály

Pro **každý spojitý čas** či **diskrétní čas** přesně víme, jakou **hodnotu** bude signál **mít**. Lze je definovat funkcí.

## Náhodné signály

Nelze je popsat rovnicí. Pro část ( $t$ ) nebo [ $n$ ] **nikdy** přesně **nevíme**, jaká bude jejich **hodnota**. Popisují se pomocí parametrů jako **střední hodnota** nebo **rozptyl**.

## Periodické

Průběh signálu se opakuje s určitou periodou.  $s(t+T) = s(t)$ .

- **Harmonické** - Nejjednodušejí definované periodické signaly. Mají tvar
$$x(t) = A \cdot \cos(\omega t + \varphi_0),$$
  - $A$  - Amplituda. Maximální hodnota periodické funkce.
  - $\omega$  - úhlový nebo kruhový kmitočet [rad/s] pro spojitý a [rad] pro diskrétní.
  - $\varphi$  - počáteční fáze [rad]. Posunutí funkce po ose x.

## Spektrální analýza

Spektrální analýzou signálů zjišťujeme jejich **frekvenční charakteristiku** (zajímají nás):

- **kde** jsou frekvenční komponenty signálu (na jakých frekvencích se signál nachází)
- **kolik** je na které frekvenci signálu (amplituda)
- jak je na které frekvenci signál **posunutý** (fázový posun).

**Rozklad** signálu na **sinusovky** a **cosinusovky**. Převod signálu z časové oblasti do frekvenční oblasti. Výsledkem spektrální analyzy je **spektrum**.

## Fourierova řada Fourier Series

Každý periodický signál lze vyjádřit jako součet sinusovek a cosinusovek.

**Frekvence** jednotlivých sinusovek a cosinusovek jsou **celočíselným násobkem** frekvence **původního** signálu. **Fázi** a **amplitudu** každé harmonické složky (sinusovky a cosinusovky), lze získat **furiérovým rozkladem** (výpočet koeficientů  $a_n$  a  $b_n$ ). Fourierova řada tedy **umožňuje popsat původní periodický signál součtem sinusovek a kosinusovek**.

$$f(t) = a_0 + \sum_{n=1}^N a_n \cos n\omega_0 t + \sum_{n=1}^N b_n \sin n\omega_0 t$$

$$\begin{aligned}
 a_0 &= \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) dx, & a_0 &= \frac{2}{T} \int_0^T f(t) dt, \\
 a_n &= \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos nx dx \quad (n = 1, 2, 3, \dots) & a_k &= \frac{2}{T} \int_0^T f(t) \cos(k\omega t) dt, \quad k \geq 1, \\
 b_n &= \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin nx dx \quad (n = 1, 2, 3, \dots) & b_k &= \frac{2}{T} \int_0^T f(t) \sin(k\omega t) dt, \quad k \geq 1.
 \end{aligned}$$

$$\cos(\omega) + i \sin(\omega) = \left( \cos\left(\frac{\omega}{n}\right) + i \sin\left(\frac{\omega}{n}\right) \right)^n$$

Euler now applies the limit  $n \rightarrow \infty$ :

$$\cos(\omega) + i \sin(\omega) = \lim_{n \rightarrow \infty} \left( \cos\left(\frac{\omega}{n}\right) + i \sin\left(\frac{\omega}{n}\right) \right)^n$$

using small angle approximations  $\cos(x) \approx 1$  and  $\sin(x) \approx x$ :

$$\cos(\omega) + i \sin(\omega) = \lim_{n \rightarrow \infty} \left( 1 + \frac{i\omega}{n} \right)^n = e^{i\omega}.$$

In the last line he applied the limit representation  $e^x = \lim_{n \rightarrow \infty} (1 + \frac{x}{n})^n$ .

Fourierova řada může být také zapsána jako součet komplexních exponenciál.

- **Vstup:** Spojitý periodický signál (funkce).
- **Výstup:** Koeficienty určující **amplitudy a fáze** komplexních exponenciál na **násobcích základní frekvence - diskrétní hodnoty**. Z **koeficientů** můžeme následně původní signál **znovu sestavit** pomocí vzorce pro **FŘ**.

Důležité je, že **FŘ** reprezentuje pořád ten **samý signál**, my ale z toho původního potřebujeme znát právě jeho **koeficienty**, které nám o něm dávají informace.

$$x(t) = \sum_{k=-\infty}^{+\infty} c_k e^{jk\omega_1 t} \quad \boxed{c_k = \frac{1}{T_1} \int_{T_1} x(t) e^{-jk\omega_1 t} dt}$$

Vzorec pro výpočet fourierovy řady

- vzorec pro výpočet koeficientů Fourierovy řady

### Diskrétní Fourierova řada - Discrete Fourier Series

**Diskrétní Fourierova řada** je tvořena **konečným počtem** součtů sinusovek a kosinusovek. Jedná se o aproximaci FŘ. **Signál musí** být stále **periodický**. Umožňuje nám provádět výpočty v praxi, protože data na počítačích nejsou **nikdy spojité** a navíc nedokážeme provést nekonečné množství součtů.

- Vstup: **Diskrétní periodický** signál s periodou **N**.
- Výstup: Koeficienty určující **amplitudy a fáze** komplexních exponenciál na **násobcích základní frekvence**, které se **periodicky** opakují s periodou **N**.

The periodic sequence  $x$  can be represented

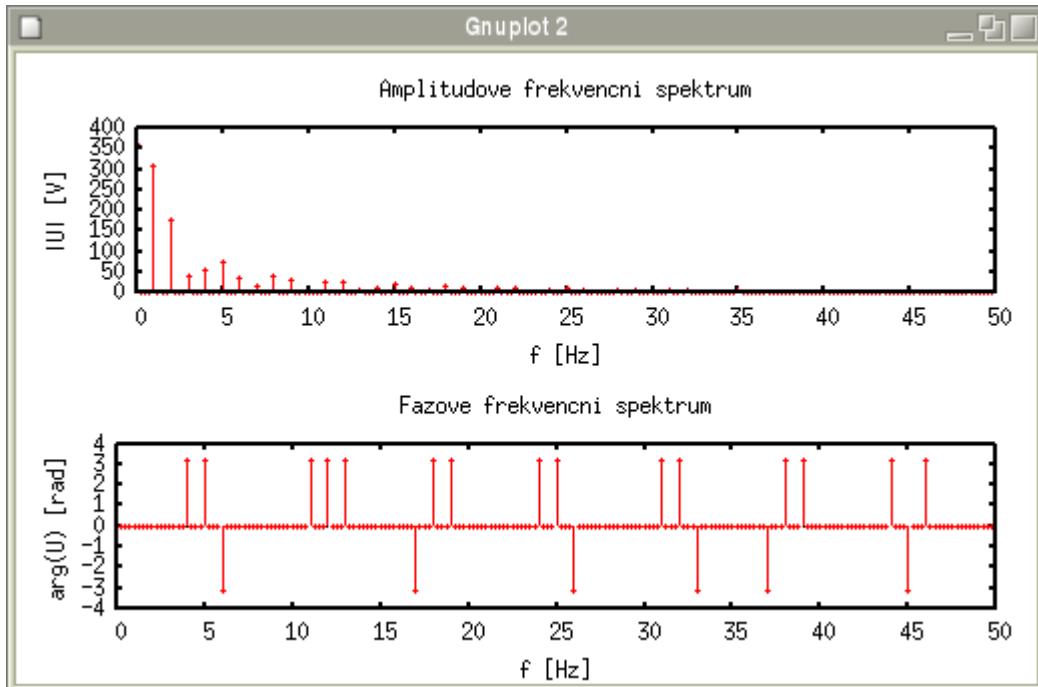
as a sum of  $N$  complex exponentials with frequencies  $k\frac{2\pi}{N}$ , where  $k = 0, 1, \dots, N-1$ :

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{jk\frac{2\pi}{N}n} \quad (1)$$

DFS coefficients are obtained as:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-jk\frac{2\pi}{N}n}.$$

for all times  $n$ , where  $X[k] \in \mathbb{C}$  is the  $k^{\text{th}}$  DFS coefficient corresponding to the complex exponential sequence  $\{e^{jk\frac{2\pi}{N}n}\}$ .

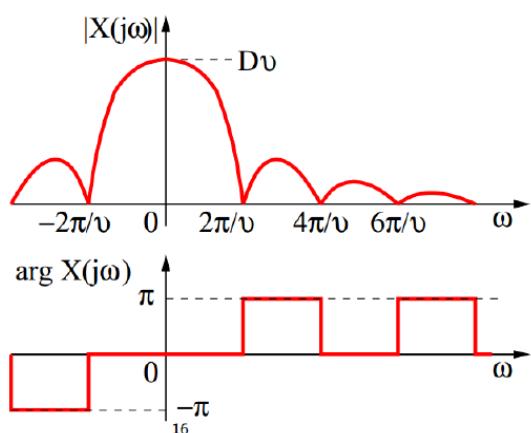
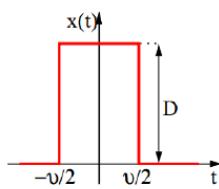


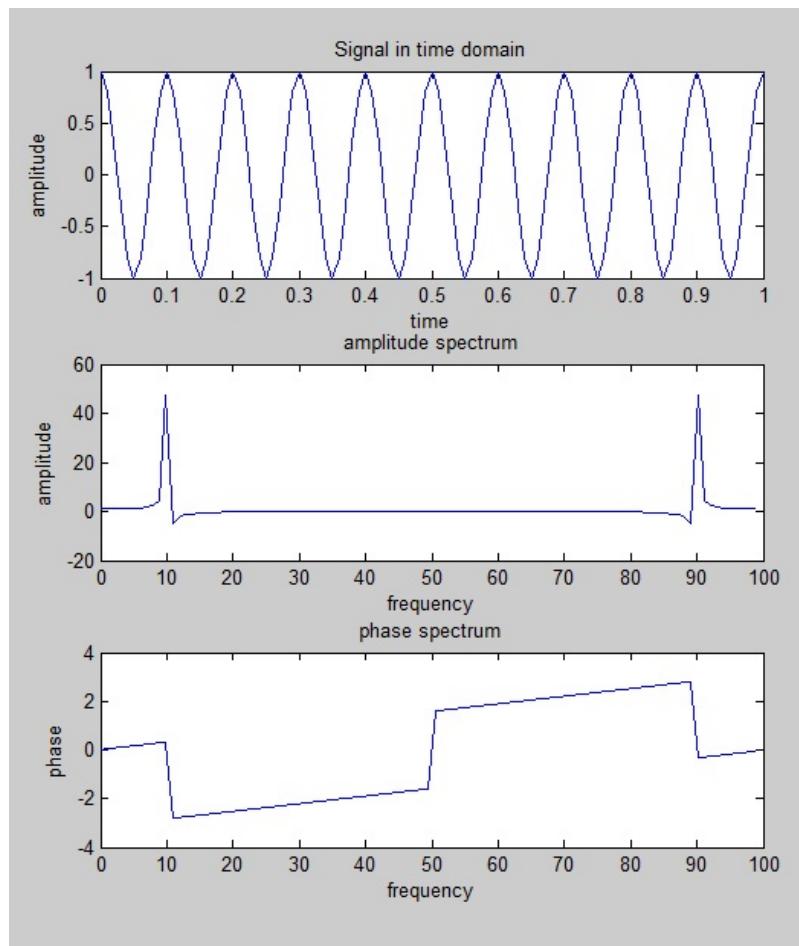
## Fourierova transformace - Fourier Transform

Zobecnění FŘ pro neperiodické signály. Slouží pro **převod** (transformaci) signálů z prostorové nebo časové oblasti do oblasti **frekvenční**. Umožňuje dekomponovat signál na **jednotlivé frekvence**, které ho tvoří.

- **Vstup** - Obecný spojitý signál (**nemusí** být periodický).
- **Výstup** - je řada komplexních čísel, z nichž každé odpovídá frekvenci, amplitudě a fázi ve výsledném **frekvenčním spektru**. (někdy lze využít pro získání koeficientů FŘ <https://lpsa.swarthmore.edu/Fourier/Xforms/FXFS.html>, pokud obsahuje **jednu periodu signálu a jinak 0** - lze ztotožnit se získáváním koeficientů FŘ)

$$S(\omega) = \int_{-\infty}^{\infty} s(t) e^{-i\omega t} dt$$





## Fourierova transformace s diskrétním časem - Discrete Time Fourier Transform

Zobecňuje DFŘ pro neperiodické signály. Slouží pro převod (transformaci) signálů z prostorové nebo časové oblasti do oblasti frekvenční.

- **Vstup:** Nekonečno vzorků obecného diskrétního signálu (nemusí být periodický).
- **Výstup:** Spojitá periodická funkce ve frekvenční oblasti, což není na PC použitelné. Diskrétní je pouze v případě periodického signálu.

$$X_{2\pi}(\omega) = \sum_{n=-\infty}^{\infty} x[n] e^{-i\omega n}.$$

## Diskrétní Fourierova transformace - Discrete Fourier Transform

Slouží pro převod (transformaci) signálů z prostorové nebo časové oblasti do oblasti frekvenční. Vzorkuje výstup DTFT (fourierova transformace s diskrétním časem), který je spojitý. Pracuje s kvadratickou O(n^2) časovou složitostí.

- **Vstup:** N vzorků obecného neperiodického signálu (DFT bere těchto N vzorků, jako by se periodicky opakovaly)

- Výstup: **N komplexních koeficientů** (komplexních exponenciál) které určují **amplitudy a fáze frekvence**. Pro PC tedy ideální situace.

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N} kn} \\ &= \sum_{n=0}^{N-1} x_n \cdot \left[ \cos\left(\frac{2\pi}{N} kn\right) - i \cdot \sin\left(\frac{2\pi}{N} kn\right) \right], \end{aligned}$$

## Fast Fourier Transform

modifikace DFT, která je rychlejší, pracuje s **linearitickou O(n\*logn)** časovou složitostí. Lze počítat pouze pro **počty vzorků**, které jsou rovny **mocnině 2** ( $2^n$ ). Jedná se dnes o jeden z **nejpoužívanějších algoritmů vůbec**. Např.:

- **Komprese** JPEG, MP3, ...: Fungují na principu výpočtu FFT a následně **odstranění frekvencí**, které jsou zastoupeny **minimálně** - což je většina, třeba i **99%**. Jedná se o ztrátovou kompresi (nelze prakticky rozetnat), ale **nezbytnou**.
- **derivace**
- **filtrace dat**: pomocí FFT nalezneme frekvence, které tvoří rušení a odstraníme je.
- **konvoluce**: pomocí FFT se přivedou signály **do frekvenční domény**, **vynásobí se a přivedou se zpět** pomocí IFFT (inverzní FFT). Běžná konvoluce je v **O(n^2)**, konvoluce pomocí FFT v **O(n\*logn)**.

## Pomůcka

platí že **periodicitu v časové doméně** způsobuje **diskrétnost ve frekvenční doméně** a **diskrétnost v časové doméně** způsobuje **periodicitu ve frekvenční doméně**.

FŘ je **periodická spojitá** (v čase) → její koeficienty jsou **diskrétní neperiodické** (ve frekvenci),

DFR je **periodická diskrétní** (v čase) → její koeficienty jsou **diskrétní periodické** (ve frekvenci), perioda u bou je **stejná**.

FT není **periodická spojitá** (v čase) → koeficienty jsou vyjádřeny **spojitou neperiodickou funkcí** (ve frekvenci),

DTFT není **periodická diskrétní** (v čase) → koeficienty jsou vyjádřeny **spojitou periodickou funkcí** (ve frekvenci)

DTF je **pseudo periodická diskrétní** (v čase) → koeficienty jsou **diskrétní a periodické** (ve frekvenci), ale vždy je jich **stejný počet jako v čase** (takže délka jedné periody - původního signálu).

Signály **pomalu** měnící se v čase budou mít **úzké spektrum** ve frekvenci, signály **rychle** měnící se v čase budou mít **široké spektrum** ve frekvenci

## Užitečné signály

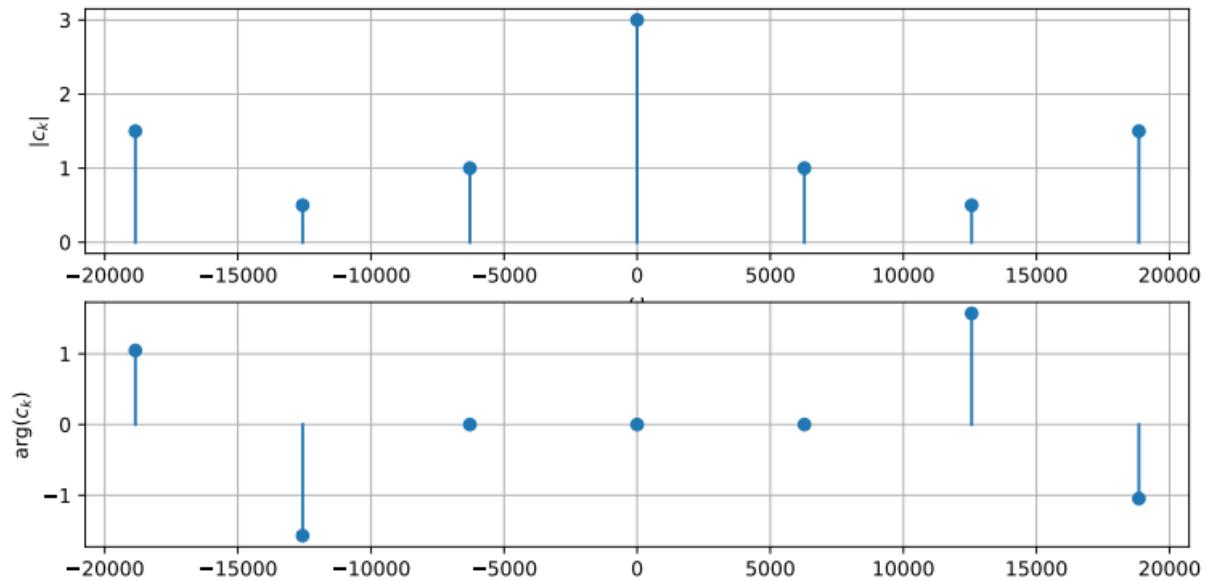
- několik cosinusovek:

$$x(t) = 3 + 2 \cos(2000\pi t) + 1 \cos(4000\pi t + \frac{\pi}{2}) + 3 \cos(6000\pi t - \frac{\pi}{3})$$

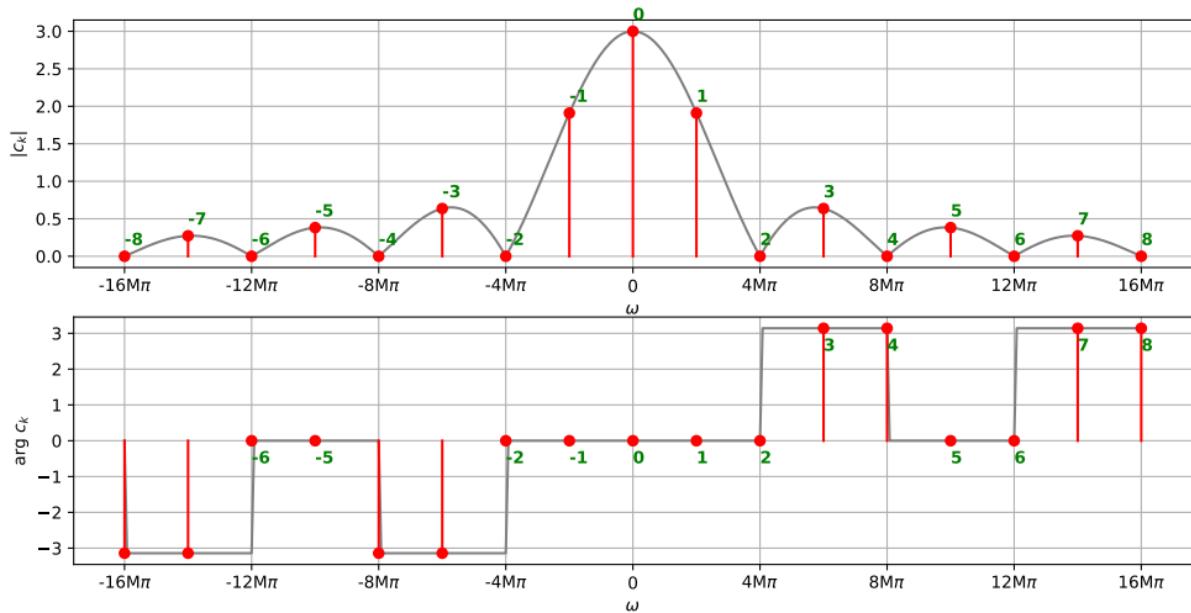
koefficienty získáme z převodního vzorečku (**půlka amplitudy a stejná, resp. opačná fáze**)

$$c_k = \frac{C_k}{2} e^{j\phi_k}, \quad c_{-k} = \frac{C_k}{2} e^{-j\phi_k}.$$

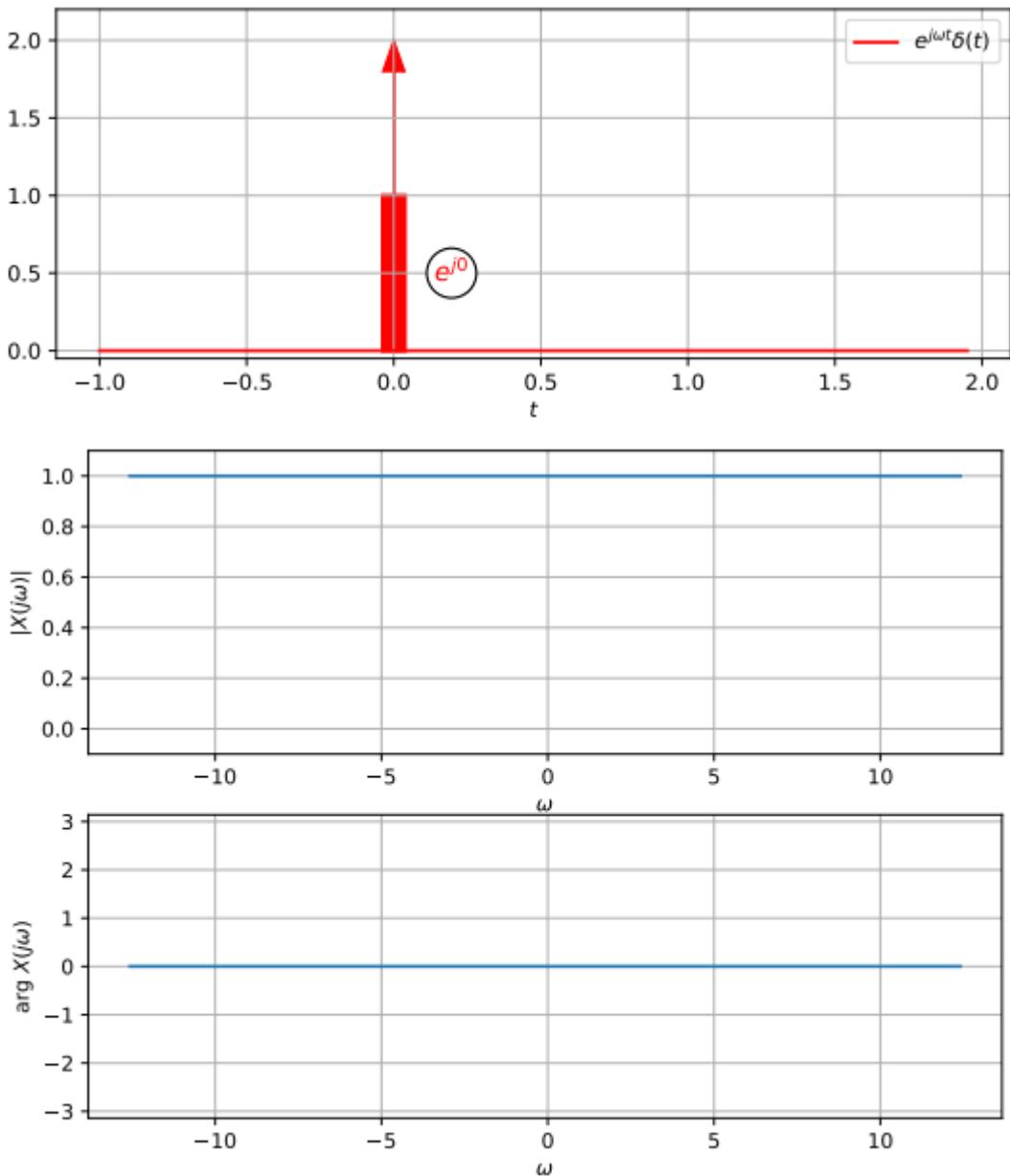
$$c_0 = 3, \quad c_1 = 1, \quad c_{-1} = 1, \quad c_2 = \frac{1}{2} e^{j\frac{\pi}{2}}, \quad c_{-2} = \frac{1}{2} e^{-j\frac{\pi}{2}}, \quad c_3 = 1.5 e^{-j\frac{\pi}{3}}, \quad c_{-3} = 1.5 e^{j\frac{\pi}{3}}.$$



- **obdelník:** Velikost impulsů bude  $D = 6$ , základní perioda  $T_1 = 1 \mu s$ , šířka impulsu  $\vartheta = 0.5 \mu s$ .  $\omega = 2\pi/\vartheta = 2\pi/(0.5 \times 10^{-6}) = 4\pi \times 10^{-6} = 4M\pi$ .  $\omega = 4M\pi$ .



- **Diracův impuls**  $\delta(t)$ : “za 0 sekund dokáže vyskočit do nekonečna a zase se vrátit zpět”



### Frekvence

- **Normální nenormovaná frekvence** -  $f$  [Hz]
- **Normální normovaná frekvence** -  $f / Fs$  [-]
- **Kruhová nenormovaná frekvence** -  $\omega = 2\pi f$  [rad/s]
- **Kruhová normovaná frekvence** -  $(2\pi f) / Fs$  [rad]

### video:

- ▶ But what is the Fourier Transform? A visual introduction.
- ▶ How are the Fourier Series, Fourier Transform, DTFT, DFT, FFT, LT and ZT Rel...
- ▶ The Discrete Fourier Transform (DFT)
- ▶ Image Compression and the FFT

3 Applications of the (Fast) Fourier Transform (ft. Michael Kapralov)

# 15. Číslicové filtry (diferenční rovnice, impulsní odezva, přenosová funkce, frekvenční charakteristika)

## Diferenční rovnice

Diferenční rovnice představují **rovnice o neznámé posloupnosti a jejích differencích**. Jedná se o **diferenciální rovnice**, které jsou v **diskrétním čase**. Obecná diferenční rovnice vypadá následovně:

$$y[n] = \sum_{k=0}^Q b_k x[n-k] - \sum_{k=1}^P a_k y[n-k], \quad (1)$$

kde  $x[n-k]$  jsou aktuální a zpožděné verze vstupu a  $y[n-k]$  jsou zpožděné verze výstupu.

diferenční rovnice stejně jako diferenciální vyžadují počáteční podmínky. Pomocí diferenčních rovnic můžeme popisovat LTI (Linear time-invariant) systémy.

- **linearita**: musí platit aditivita a scaling nebo homogenita:

$$ax_1[n] + bx_2[n] \rightarrow ay_1[n] + by_2[n]$$

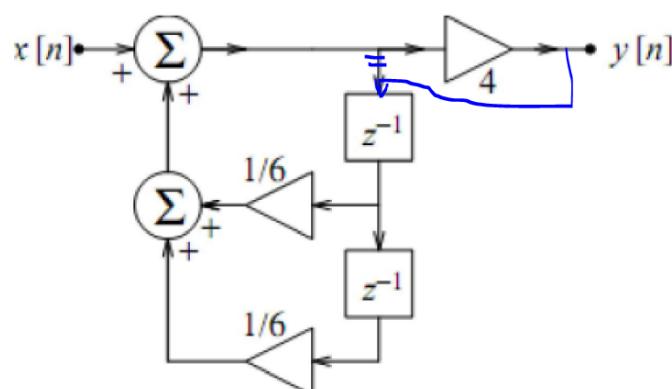
- **časová invariantnost**: systém **nemění své chování v čase**. Pokud systém zareagoval na signál  $\mathbf{x}$  výstupem  $\mathbf{y}$ , zareaguje na stejný signál  $\mathbf{x}$  za deset sekund stejným signálem  $\mathbf{y}$ .

Příklad popisu systému:

- aktuální výstup  $y[n]$  je dán součtem minulého vstupu  $1/2y[n-1]$ , aktuálního vstupu  $1/4x[n]$  a minulého vstupu  $1/4x[n-1]$ .

$$y[n] = \frac{1}{2}y[n-1] + \frac{1}{4}x[n] + \frac{1}{4}x[n-1]$$

- schéma jiného systému daného **diferenční rovnicí**  $y[n] = 4*(x[n] + 1/6y[n-1] + 1/6y[n-2])$



Příklady systémů, které se chovají jako dolní propust (propouští malé frekvence).

- průměr posledních 6 vstupů:

$$y[n] = \frac{1}{6} \{x[n] + x[n-1] + \dots + x[n-5]\}$$

- 'nabalující se vstup':

$$\underline{y[n] = 0.95 y[n-1] + 0.05 x[n]}$$

Příklady systémů, které se chovají jako horní propust (propouští velké frekvence).

- rozdíl posledních 6 vstupů:

$$y[n] = \frac{1}{6} \{x[n] - x[n-1] + x[n-2] - \dots - x[n-5]\}$$

- 'nabalující se vstup':

$$\underline{y[n] = -0.95 y[n-1] + 0.05 x[n]}$$

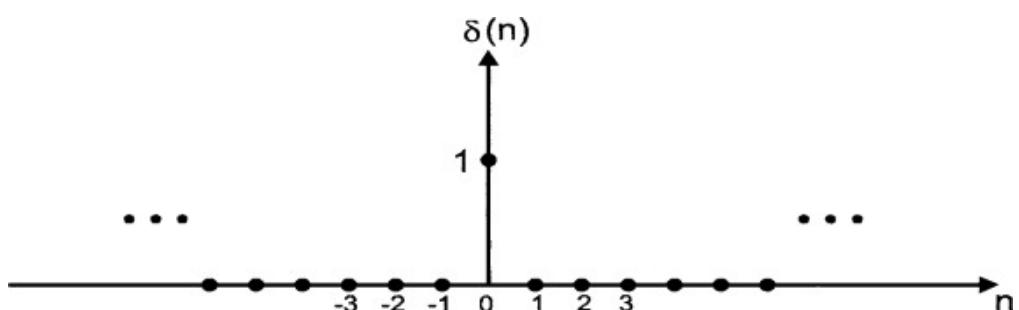
**videa:**

▶ Difference Equation Descriptions for Systems

## Impulsní odezva

Zkoumá **odezvu** systému na impuls. **Diskrétní impuls** je definován jako

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}$$

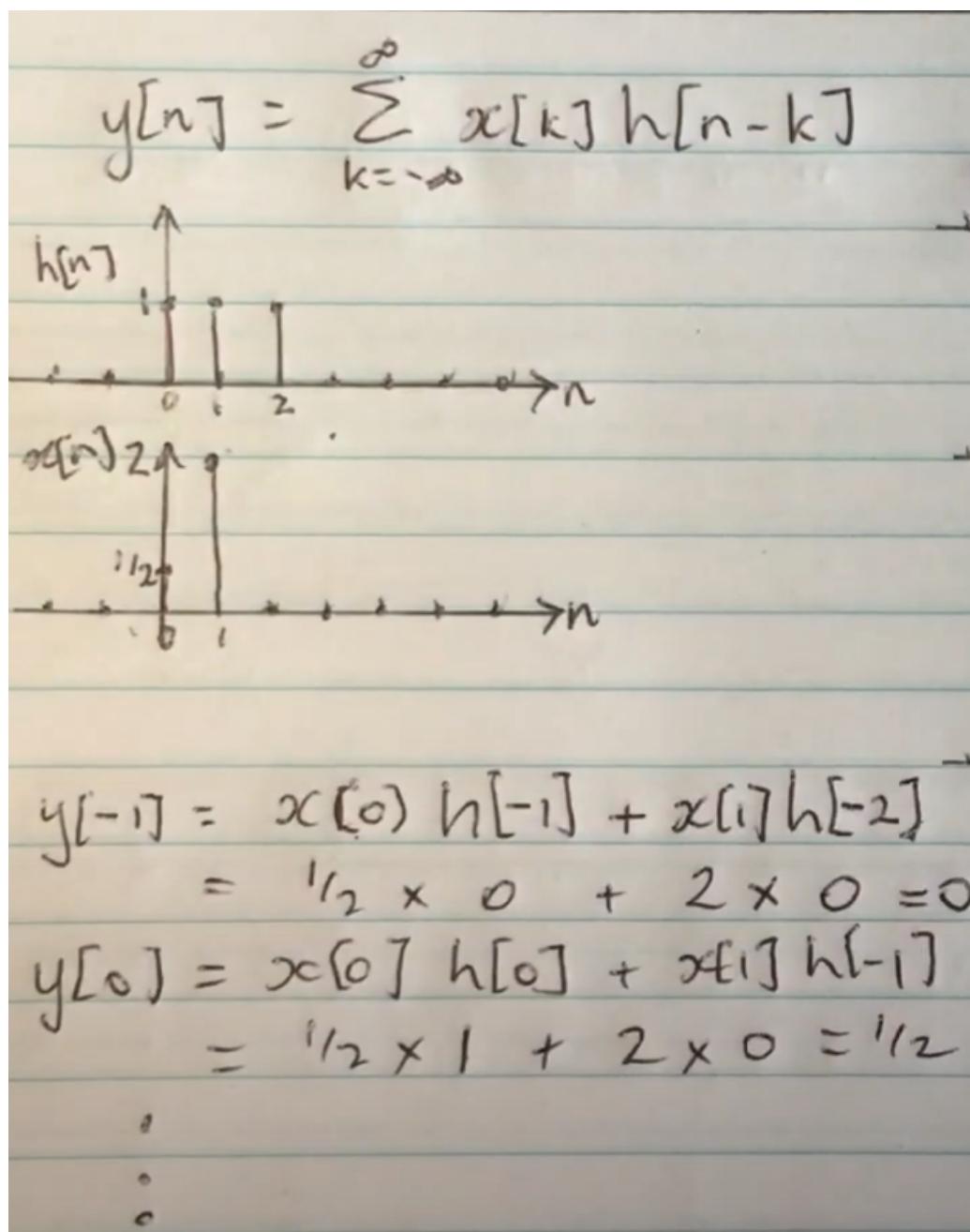


Jakýkoliv diskrétní signál může být reprezentován jako **vážený součet posunutých** diskrétních **impulsů** v čase. Impulsní odezva **definuje LTI** systémy a používá k tomu **konvoluci**. Impulsní odezva definuje systém, protože impuls obsahuje všechny frekvence (**ve frekvenční doméně je roven 1 pro všechny f**).

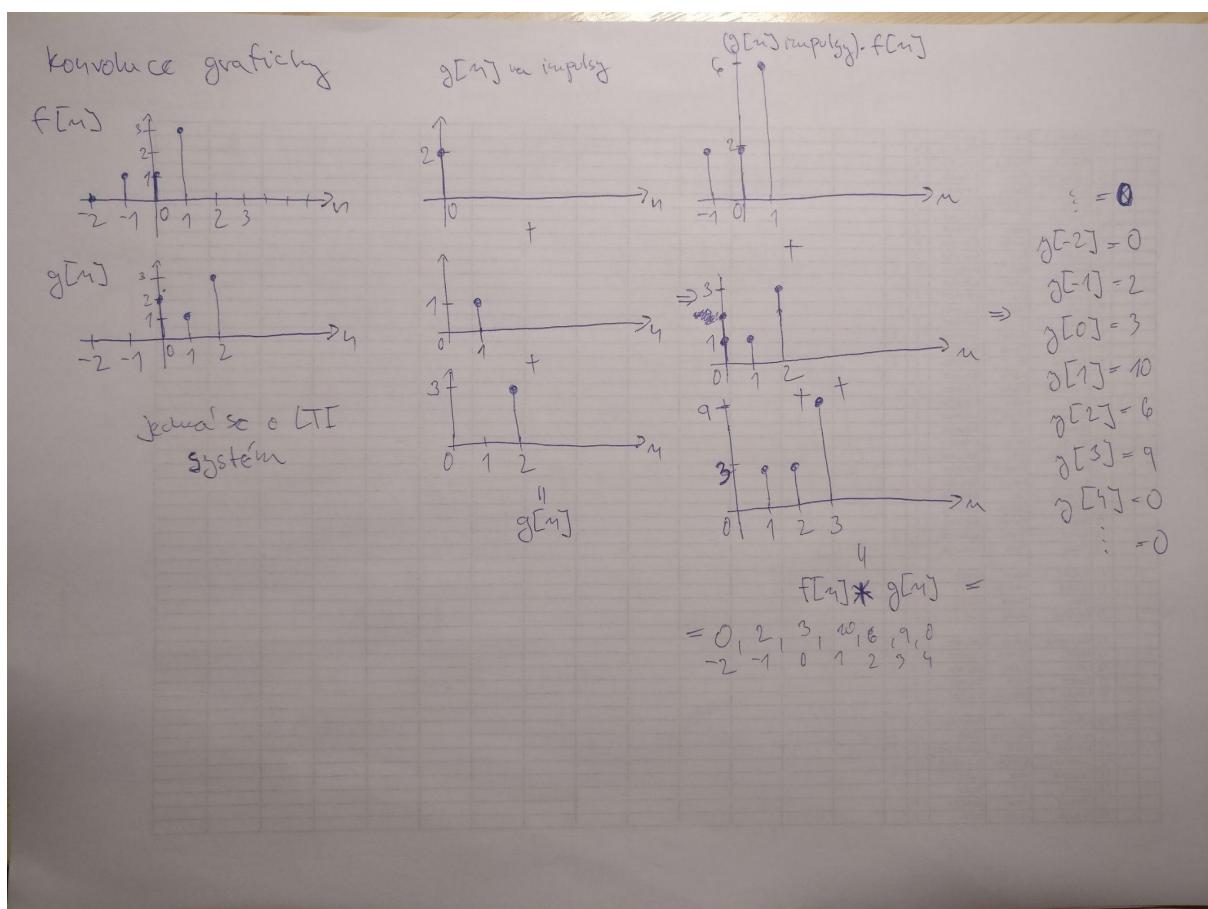
## Konvoluce

Základní operace pracující se dvěma signály je definovaná vzorcem

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m], (f * g)[n] = \sum_{m=-\infty}^{\infty} f[n-m]g[m].$$



## konvoluce graficky Discrete Time Convolution Example :

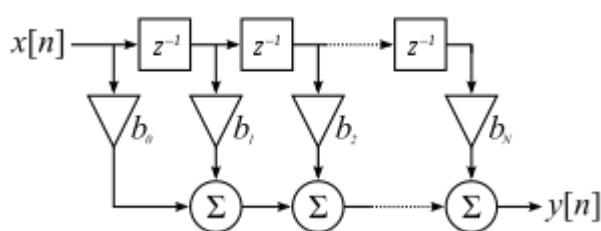


konvoluci lze ve **frekvenční doméně** provést pouze **násobením**.

FIR - Finite Impulse Response, IIR - Infinite Impulse Response

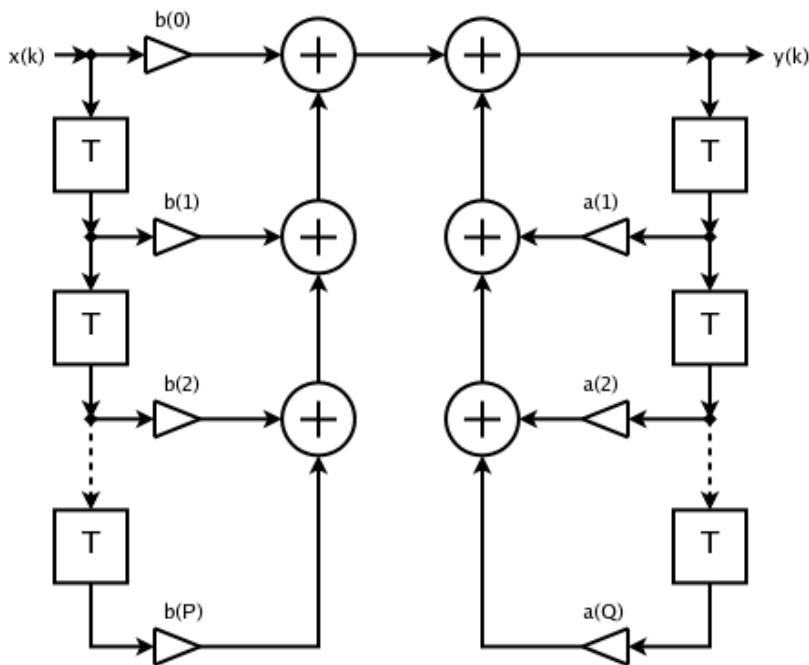
- **FIR:** impulsní odezva nebo odezva konečného vstupu má **konečnou délku**.  
Tyto filtry **nesmí** mít ani jednu **zpětnou smyčku**.

$$y[n] = \sum_{k=0}^n b_k x[n-k]$$



- **IIR:** impulsní odezva a odezva konečného vstupu má **nekonečnou délku**.  
Tyto filtry **musí** mít alespoň jednu **zpětnou smyčku**.

$$y[n] = \sum_{k=0}^m b_k x[n-k] - \sum_{k=1}^N a_k y[n-k]$$



## Přenosová funkce

Přenosová funkce definuje LTI systém. Pokud chceme na nějaký signál aplikovat filter, tak tímto filtrem **vynásobíme** (konvoluce v časové doméně) ve frekvenční doméně (např. filtrem, který propouští pouze spodní frekvence).  $X(z) * H(z) = Y(z) \rightarrow$  Přenosová funkce je definována jako **poměr** (podíl) **výstupu** -  $Y(z)$  ku **vstupu** -  $X(z)$  ve **frekvenční oblasti**, když jsou **zanedbány počáteční podmínky**. Jinak řečeno při **nulových** počátečních podmínkách, jde o podíl **Fourierovy transformace** výstupu a vstupu, respektive **Z transformace** u signálů, pro které nelze provést **FT** (signály, které **nejsou** v +- nekonečnu **0**, mají nekonečnou energii, Z transformace je násobí tak, aby **byly** v +- nekonečnu **0**). Přenosovou funkci lze také vyjádřit jako **FT impulsní odezvy** u FIR systémů (výstup je v nekonečnu 0), nebo jako **ZT**

➡ Z Transform Region of Convergence Explained IIR systémů (v nekonečnu nejsou 0). FT/ZT impulsní odezvy z toho důvodu, že  $X(z)$  - vstup **je 1** (FT jednotkového impulu).

$$H(z) = \frac{Y(z)}{X(z)}$$

The bilateral or two-sided Z-transform of a discrete-time signal  $x[n]$

$$X(z) = \mathcal{Z}\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n]z^{-n} \quad (\text{Eq.1})$$

Considering that in most IIR filter designs coefficient  $a_0$  is 1

$$H(z) = \frac{\sum_{i=0}^P b_i z^{-i}}{1 + \sum_{j=1}^Q a_j z^{-j}}$$

## Z-Transform Example #1

Find  $X(z)$  for  $x[k] = \begin{cases} 1 & k=-1 \\ 3 & k=0 \\ 2 & k=1 \\ -1 & k=2 \\ 0 & \text{else} \end{cases}$

⇒ Finite length signal. No convergence "worries".

$$X(z) = \sum_{k=-\infty}^{\infty} x[k] z^{-k} = \sum_{k=-1}^2 x[k] z^{-k} = (1)z^{(-1)} + (3)z^0 + (2)z^{-1} + (-1)z^{-2}$$

$$= z + 3 + 2z^{-1} - z^{-2}$$

Find DTFT of  $x[n]$ .  $\Rightarrow X(\Omega) = e^{j\Omega} + 3 + 2e^{-j\Omega} - e^{-2j\Omega}$

### Region of Convergence

- ▶ Z Transform Region of Convergence Explained
- ▶ Laplace Transform Region of Convergence Explained

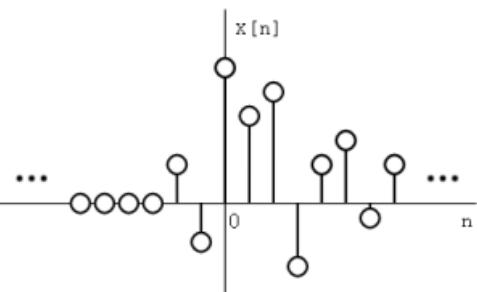


Figure 12.6.2: A right-sided sequence.

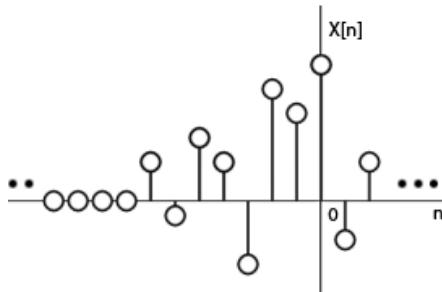


Figure 12.6.4: A left-sided sequence.

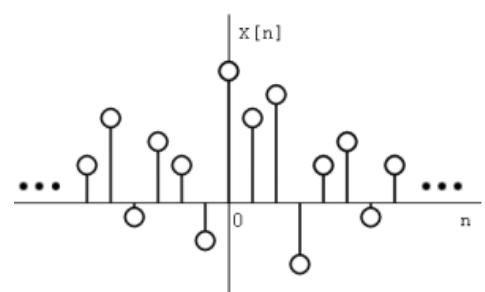


Figure 12.6.6: A two-sided sequence.

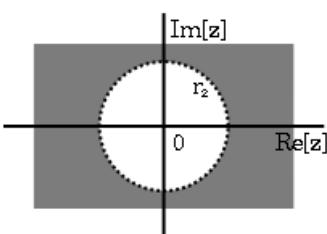
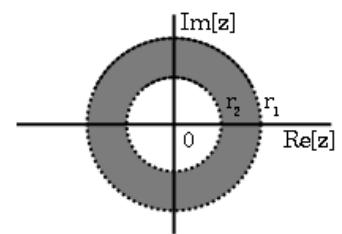
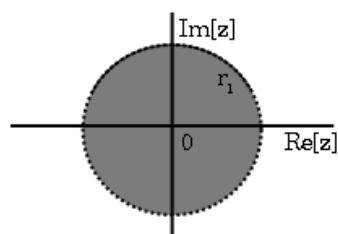
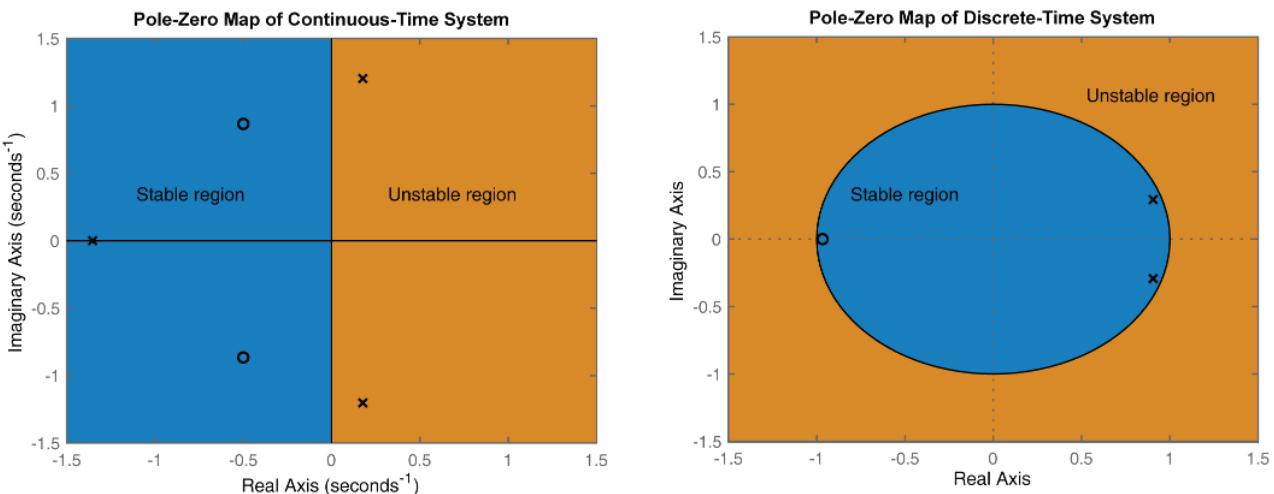


Figure 12.6.3: The ROC of a right-sided sequence. 12.6.5: The ROC of a left-sided sequence. 12.6.7: The ROC of a two-sided sequence.



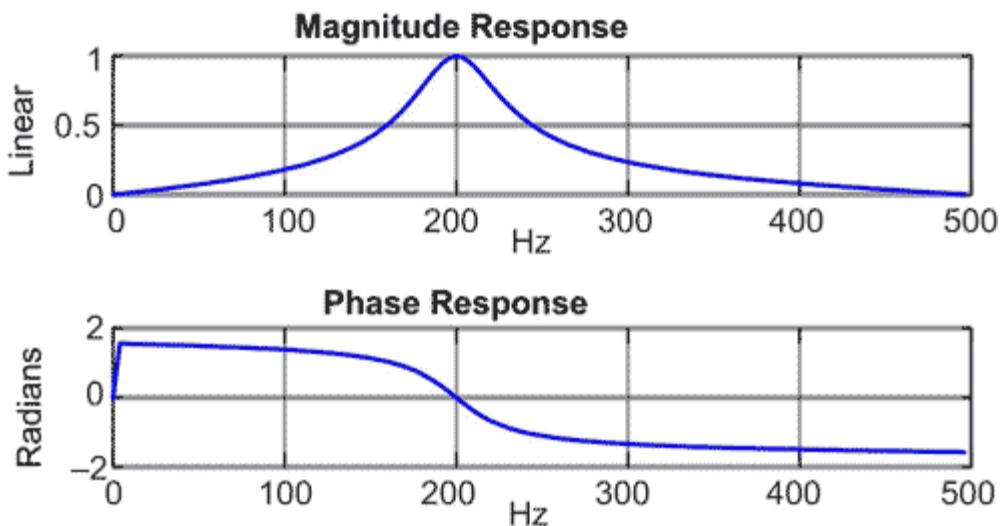
### Stabilita

- ▶ How do Poles and Zeros affect the Laplace Transform and the Fourier Transform...
- ▶ Frequency Response Magnitude and Poles and Zeros



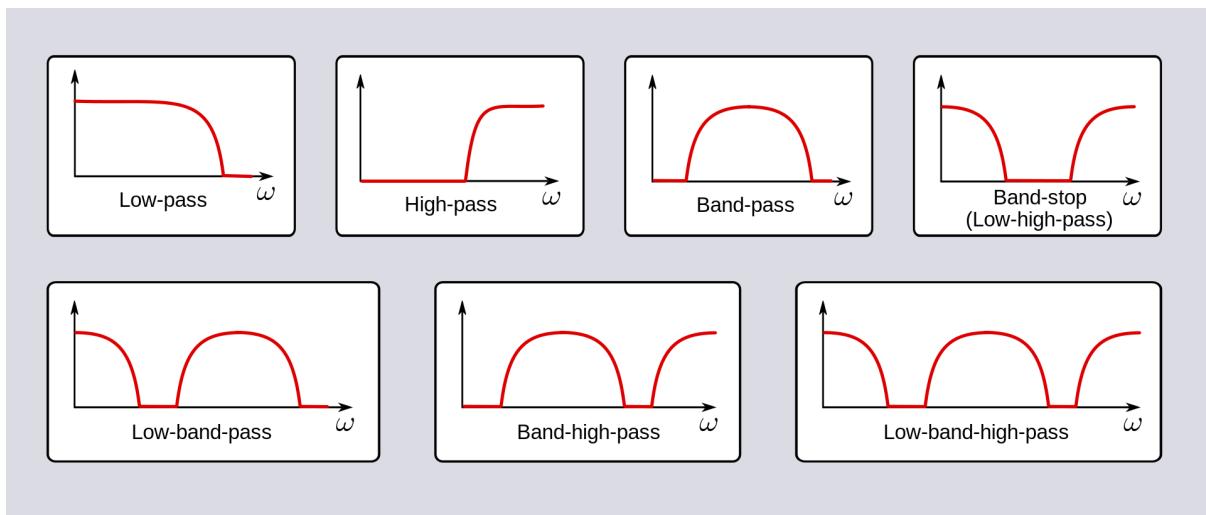
## Frekvenční charakteristika

**Frekvenční charakteristika** popisuje/definuje systém **ve frekvenční doméně**, stejně jako **impulsní odezva** jej charakterizuje v **časové doméně**. Graficky ji vynášíme ve dvou grafech jako **amplitudovou  $|H(\omega)|$**  (jak zeslabuje/zesiluje) a **fázovou charakteristikou  $\arg(H(\omega))$**  (jak posouvá).



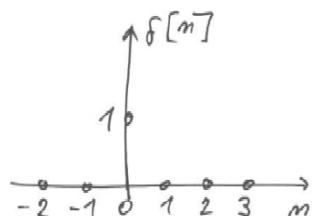
## Dělení frekvenčních charakteristik

- **Dolní propust** - filtr, který propouští nízké frekvence.
- **Horní propust** - filtr, který propouští vysoké frekvence.
- **Pásmová propust** - filtr, který propouští signál **jen určitých** frekvencí.
- **Pásmová zádrž** - filtr, který **nepropouští** signál **určitých** frekvencí.

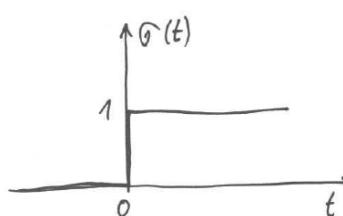


## Komplexní čísla

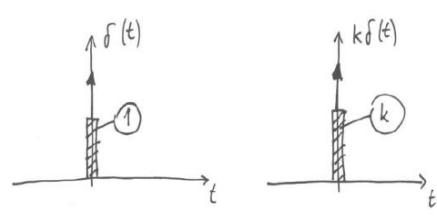
Jednotkový puls



Jednotkový skok



Diracův impuls



- Možné transformace signálu

- **Otočení časové osy** -  $s(-t)$
- **Zpozdění** -  $s(t-x)$ , kde  $x > 0$
- **Předběhnutí** -  $s(t+x)$ , kde  $x > 0$
- **Kontrakce** -  $s(m*t)$ , kde  $m > 1$
- **Dilatace** -  $s(t/m)$ , kde  $m > 1$

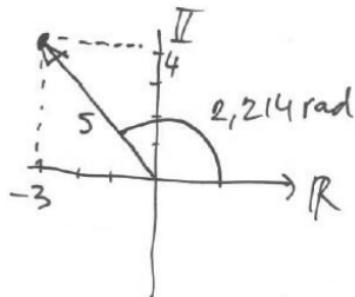
- **Komplexní číslo** - Složené z reálné části (osa x) a imaginární části (osa y), kde jednotka je odmocnina z -1, za kterou se substituuje  $i$  nebo  $j$ . Komplexní číslo je možné zapsat jako sumu těchto 2 hodnot (42 + 3i) nebo vektorem:  $z = |z|(\cos\phi + i\sin\phi)$ , kde  $|z|$  je délka vektoru a  $\phi$  je úhel s kladnou osou x.

$$|z| = \sqrt{(a^2 + b^2)}; \varphi = \tan^{-1} \frac{b}{a}$$

## Příklad 1

$$z = -3 + j4, \quad r = \sqrt{3^2 + 4^2} = 5, \quad \phi = \tan^{-1} \frac{4}{-3} = -0.92$$

což je špatně!. Správný úhel je  $\phi = \pi - 0.92 = 2.214$  rad.



- **Exponenciální tvar komplexního čísla** -  $z = r^* e^{j\phi}$
- **Komplexně sdružené číslo** - pro číslo  $z = a + bi = re^{j\phi}$  existuje komplexně sdružené číslo  $\bar{z} = a - bi = re^{-j\phi}$ . Vznikne změnou znaménka imaginární části čísla. Např.  $\overline{3 - 2i} = (3 - 2i)^* = 3 + 2i$ . Součtem čísla a jeho komplexně sdruženého čísla vznikne reálné číslo. Funkce  $y = e^{jx}$  je komplexní exponenciála součtem s komplexně sdruženou získáme kosinusovku.

$$\cos \phi = \frac{e^{j\phi} + e^{-j\phi}}{2} \quad \sin \phi = \frac{e^{j\phi} - e^{-j\phi}}{2j}$$

- **Obecná kosinusovka**

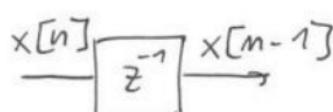
$$C_1 \cos(\omega_1 t) = \frac{C_1}{2} e^{j\omega_1 t} + \frac{C_1}{2} e^{-j\omega_1 t}$$

## Filtr

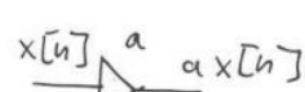
**Filtr** je lineárně časově invariantní (LTI) systém, který upravuje signál **ve frekvenční oblasti**. Je charakteristický **impulsní odezvou**. Výstup se získá konvolucí. Lze implementovat pomocí diferenční rovnice.

- **Základní bloky filtrů**

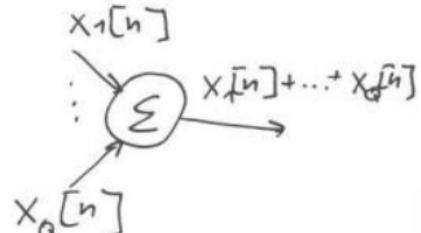
### Zpožd'ovací článek



### násobička



### sčítáčka



## 15. Číslicové filtry (diferenční rovnice, impulsní odezva, přenosová funkce, frekvenční charakteristika) Ta otazka to celkem presne rika. Chci vedet:

jake jsou zakladni bloky cislicoveho filtru: zpozdeni, nasobeni, soucet.

- jak vypada schema obecnego IIR filtru s nerekurzivni (koeficienty b) a rekurzivni casti (koeficienty a).
- co z nej ustrihnut, aby se z nej stal FIR nebo ciste IIR filtr.
- jak schema zapsat diferencni rovnici.
- jak ji prevest na obrazovou formu (tedy jak ji z-transformovat): pomucka: konstanty zustanou konstanty, vsechna  $x[n]$  se prepisou na  $X(z)$ ,  $y[n]$  na  $Y(z)$  a kdyz je nekde zpozdeni o neco, musi se vyjadrit pomocí  $z^{\{-neco\}}$
- jak z toho udelat prenosovou funkci (reseni: chcete dostat podil  $Y(z)/X(z)$ , vyjde Vam podil polynomu  $B(z)/A(z)$ , kde citatel zavisi na vstupni casti, jmenovatel na vystupni).
- jak z toho udelat kmitoctovou charakteristiku: nahradit z za  $e^{\{j \omega\}}$ , kde omega je norm. kruh. frekvence... mimochedom (viz minula otazka), ted uz by Vam melo byt jasne, proc je frekv. charakteristika cislic. filtru periodicka s  $F_s$
- velice zhruba (bez rovnic) vedet, jak se frekv. charakteristika da spocitat nebo odhadnout rozlozenim citatele i jmenovatele na nuly a poly.

**upozorneni:** toto si pod otazkami predstavuju ji jako garant a ucitel ISS, u statnic ale muzete potkat dalsi lidi, kteři temto vecem dobre rozumi (celkem kdokoliv z recove a graficke skupiny, Fucik, Sekanina, a mnozi dalsi) a ti mohou mit lehce odlišnou interpretaci. Naucenim vyse uvedeneho ale rozhodne neprohlopitez.

# 16. Množiny, relace a zobrazení.

## Množina

Matematická struktura **neopakujících** se objektů (prvků množiny), chápáných jako celek. Množina je jednoznačně určena svými prvky, ale nezáleží na jejich pořadí. Množiny mohou být prázdné, konečné a nekonečné.

## Nekonečné množiny

- **Spočetná** - Taková množina, která je vzájemně **jednoznačná** (bijektivní) zobrazení s některou **podmnožinou přirozených čísel**.
- **Nespočetná** - Množina, kterou nelze jednoznačně vzájemně zobrazit na žádnou podmnožinu přirozených čísel (např. **reálná čísla**).

## Popis množiny a značení

Množiny obvykle značíme velkými písmeny a jejich prvky písmeny malými. Je-li prvek **a** a prvkem množiny **B**, píšeme:  $a \in B$ . Způsoby zadávání:

- **Výčtem prvků**:  $A = \{42; 1337; 94; 0\}$ ,
- **Predikátem**:  $B = \{2^k \mid k \in \mathbb{N}\}$ ,
- **Intervalem**:  $(a; b)$ ,  $(a; b]$ ,  $[a; b)$ ,  $[a; b]$  -  $a < b$ ,
- **Dobře známé množiny**:  $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}, \emptyset$ .

## Potenční množina

Potenční množina množiny **X**  $P(X)$  je taková množina, která obsahuje všechny podmnožiny množiny **X**. Pokud je množina **X** konečná a její mohutnost  $|X| = n$ , pak je mohutnost potenční množiny  $P(X)$  rovna  $|P(X)| = 2^n$ . Např.:

$$\begin{aligned} A &= \{1, 2, 3\} \\ P(A) &= \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\} \\ &\quad (\text{pozn. množina samotná je také podmnožinou sama sebe}). \end{aligned}$$

## Prázdná množina ( $\emptyset$ nebo $\{\}$ )

Množina, která neobsahuje žádné prvky.

## Uzavřenosť množiny na operaci

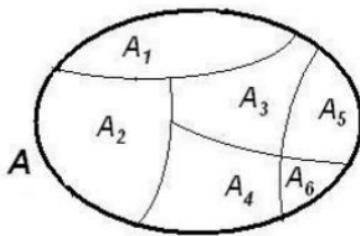
Podmnožina  $M \subset A$  je **uzavřená** vůči algebraické operaci pokud tato operace **vrátí** hodnotu z **M**. Tedy Provedením operace s **uzavřenou množinou k této operaci** získáme opět prvek **z této množiny**.

## Podmnožina

Množina **B** je podmnožinou množiny **A**, pokud platí, že všechny prvky množiny B se **nachází** v množině **A**. Pak je možné množinu **A** označit také za **nadmnožinu**. Stav "bytí podmnožinou" se také nazývá **inkluze**.

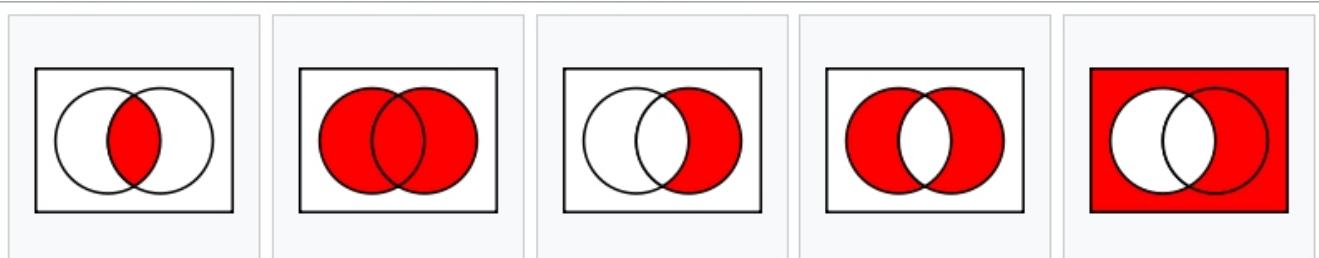
## Rozklad množiny

Rozkladem množin vznikne **systém množin** (množina množin), kde sjednocením jejích prvků získáme původní rozloženou množinu a zároveň pokud pro libovolné dva prvky rozkladu platí  $X \cap Y \neq \emptyset$ , pak  $X = Y$ .



## Operace s množinami

- **Průnik:** průnik dvou nebo více množin označuje taková množina, která obsahuje pouze ty prvky, které se nalézají ve všech těchto množinách.
- **Sjednocení:** sjednocení dvou nebo více množin označuje taková množina, která obsahuje každý prvek, který se nachází alespoň v jedné ze sjednocovaných množin, a žádné další prvky.
- **Rozdíl:** rozdíl dvou množin označuje taková množina, která obsahuje každý prvek, který se nachází v první z množin, ale nenachází se ve druhé z nich, a žádné další prvky.
- **Symetrická differenze** (symetrický rozdíl): symetrická differenze nebo symetrický rozdíl dvou množin označuje taková množina, která obsahuje všechny prvky z obou množin, které nejsou v jejich průniku. Značení:
  - $A \Delta B$ ,  $A \setminus B$ ,  $A \ominus B$ .
- **Doplňek:** doplněk množiny A nebo **komplement** množiny A označuje množina  $A^c$  všech prvků, které nejsou v A a přitom v nějaké jiné (předem dané) množině jsou obsaženy (na obrázku v U). Aby bylo možné doplněk definovat, je třeba znát množinu, vzhledem ke které se doplněk počítá.



Průnik dvou množin  
 $A \cap B$

Sjednocení dvou množin  
 $A \cup B$

Rozdíl množin A (vlevo) a B (vpravo)  
 $A^c \cap B = B \setminus A$

Symetrická differenze dvou množin  
 $A \Delta B$

Doplňek A v U  
 $A^c = U \setminus A$

## Vlastnosti operací

- **Komutativnost** - Vlastnost binární operace, která říká, že nezáleží na pořadí argumentů -  $A \cap B = B \cap A$ . U komutativní operace můžeme **zaměnit pořadí** hodnot a **nezměníme tím výsledek**.
- **Asociativita** - Vlastnost **binární operace**, která říká, že **nezáleží** na tom, jak použijeme **závorky** u výrazu, kde je více operandů. Nezáleží, v jakém pořadí budeme tedy tento výraz počítat -  $(A \cap B) \cap C = A \cap (B \cap C)$ .
- **Distributivita** - Vlastnost **binární operace vůči jiné binární operaci**, říkající, že můžeme tuto **operaci distribuovat** přes jinou operaci -  $x^*(y + z) = (x * y) + (x * z)$ . Nebo  $(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$ .

## Relace

Jako relaci nebo n-ární relaci nazveme v matematice libovolný vztah mezi skupinou prvků jedné nebo více množin. Podle **arity** dělíme relace na **unární, binární, ternární a nární**.

Relace mezi množinami  $A_1, A_2, \dots, A_k$ , pro  $k \in \mathbb{N}$ , je libovolná podmnožina kartézského součinu  $R \subseteq A_1 \times A_2 \times \dots \times A_k$ . Pokud  $A_1 = A_2 = \dots = A_k = A$ , hovoříme o **k-ární relaci R na A**.

### Unární relací

V relaci se nachází jeden prvek, např.:

- relace: **Být kladné číslo**. Ptáme se, je číslo kladné?
- relace: **Být pravdivý výrok**. Ptáme se, je výrok pravdivý?

### Binární relace

Binární relace jsou relace, do kterých vstupují **dva** prvky množiny (množiny mohou být různé). Binární relace  $R$  mezi množinami  $A, B$  je libovolná podmnožina  $R$  **kartézského součinu** množin  $A, B$ . Označení:  $[x, y] \in R$  nebo  $(xRy)$ . Pokud  $A = B$ , hovoříme o binární relaci na množině  $A$ . Množina  $A$  se nazývá **definičním oborem** relace  $R$  a množina  $B$  **oborem hodnot** relace  $R$ .

- **Symetrická**: pokud platí  $(pRq)$ , tak platí i  $(qRp)$ . Relace je symetrická pokud pro  $p$  a  $q$  z  $X$  platí, že  $p$  je v relaci s  $q$  a  $q$  je současně v relaci s  $p$ .
  - např. relace  $R =$  ‘je sourozenec’ - **Prázdná relace je symetrická**, tzn. platí to i když někdo nemá sourozence.

$$\forall a, b \in X : [a, b] \in R \implies [b, a] \in R$$

- **Antisymetrická** (slabě antisymetrická): Relace, ve které nenastává že by  $p$  bylo v relaci s  $q$  a zároveň  $q$  v relaci s  $p$ . Pokud tak nastane,  $p$  se rovná  $q$ . Jedná se např. o  $R =$  ‘je menší nebo rovno’ ((neostré) uspořádání). **Nejedná se o opak symetrie**. Relace může být **současně symetrická i asymetrická**, např. rovnost.

$$(\forall a, b \in X)(aRb \wedge bRa \Rightarrow a = b)$$

- **Asymetrická** (Silně antisymetrická): relace, která je **současně antisymetrická a ireflexivní**. Jedná se např. o  $R = \text{'je menší než'}$  (ostrá nerovnost). Jediná relace, která je **současně symetrická a asymetrická**, je **prázdná** relace.

$$(\forall a, b \in X)(aRb \Rightarrow \neg(bRa))$$

- **Reflexivní**: pokud pro všechna  $x$  patřící do  $X$  platí **( $xRx$ )**, jinak řečeno, pokud je každý prvek **v relaci sam se sebou**. **Prázdná** relace nad **prázdnou** množinou **je reflexivní**, ale **prázdná** relace nad **naprázdnou** relací **není reflexivní**.

- např.  $R = \text{'je stejný'}$ ,  $R = \text{'je větší nebo rovno'}$ ,  $R = \text{'je podmnožinou'}$

$$\forall a \in X : [a, a] \in R$$

- **Tranzitivní** - Pokud **( $pRq$ )** a současně **( $qRr$ )**, pak platí **( $pRr$ )**. Pokud pro každé  $p, q, r$  z množiny  $X$  platí, že pokud je  $p$  v relaci s  $q$  a  $q$  v relaci s  $r$ , tak je i  $p$  v relaci s  $r$ . **Prázdná relace je tranzitivní**.
  - např.  $R = \text{'je sourozenec'}$ ,  $R = \text{'je vyšší'}$
  - relace  $R = \text{'je kamarád'}$  **není tranzitivní**.

$$\forall a, b, c \in X : (([a, b] \in R \wedge [b, c] \in R) \implies [a, c] \in R)$$

## Známé binární relace

- **Ekvivalence**: relace, která je současně **reflexivní, symetrická a tranzitivní**. Např. **rovnoběžnost, rovnost, podobnost trojúhelníků**.
- **Uspořádání**: relace, která je současně **reflexivní, (slabě) antisymetrická a tranzitivní**. Např.  $R = \text{'větší nebo rovno než'}$  nebo  $R = \text{'je podmnožinou'}$
- Ostré uspořádání: relace, která je současně **ireflexivní, asymetrická a tranzitivní**. Např.  $R = \text{'větší než'}$  nebo  $R = \text{'je vlastní podmnožinou'}$  (podmnožina množiny, která ji není rovna).

## Inverzní relace

Inverzní binární relace je množina uspořádaných párů, která je přesnou inverzí (obrácením pořadí) množiny uspořádaných párů původní binární relace. Např.: pro  $R = \{(1, a), (2, b), (3, c)\}$  je inverzní relace  $R^{-1} = \{(a, 1), (b, 2), (c, 3)\}$

## Uzávěry relací

- **Tranzitivní**: Warshall's Algorithm (Finding the Transitive Closure) Např. pro  $R = \{(2, 1), (2, 3), (3, 1), (3, 4), (4, 1), (4, 3)\}$  on set  $A = \{1, 2, 3, 4\}$   
je tranzitivní uzávěr  $R^+ = \{(2, 1), (2, 3), (2, 4), (3, 1), (3, 3), (3, 4), (4, 1), (4, 3), (4, 4)\}$

- **Reflexivní:** reflexivní uzávěr binární relace  $\mathbf{R}$  na množině  $\mathbf{X}$  je nejmenší reflexivní relace  $\mathbf{S}$  na množině  $\mathbf{X}$  obsahující relaci  $\mathbf{R}$ .

$$S = R \cup \{(x, x) : x \in X\}$$

Např.:  $X = \{1, 2, 3\}$ ,  $R = \{(1, 1), (2, 3)\}$ ,  $S = \{(1, 1), (2, 2), (3, 3), (2, 3)\}$

- **Symetrický:** symetrický uzávěr binární relace  $\mathbf{R}$  na množině  $\mathbf{X}$  je nejmenší symetrická relace  $\mathbf{S}$  na množině  $\mathbf{X}$  obsahující relaci  $\mathbf{R}$ .

$$S = R \cup \{(y, x) : (x, y) \in R\}.$$

Např.:  $X = \{1, 2, 3\}$ ,  $R = \{(1, 1), (2, 1), (2, 3)\}$ ,  $S = \{(1, 1), (1, 2), (2, 1), (2, 3), (3, 2)\}$

## Ternární relace

Ternární relace jsou relace, do kterých vstupují **tři** prvky množiny (množiny mohou být různé). Ternární relace  $\mathbf{R}$  mezi množinami  $\mathbf{A}, \mathbf{B}, \mathbf{C}$  je libovolná podmnožina  $\mathbf{R}$  **kartézského součinu** množin  $\mathbf{A}, \mathbf{B}, \mathbf{C}$ . Označení:  $[x, y, z] \in \mathbf{R}$  nebo  $\mathbf{R}(x, y, z)$ . Pokud  $\mathbf{A} = \mathbf{B} = \mathbf{C}$ , hovoříme o ternární relaci na množině  $\mathbf{A}$ .

- např.  $R = \text{'je mezi'}$

## Algebra

Množina, na které jsou definované nějaké **operace** a daná množina je vzhledem k těmto operacím **uzavřená**, tzn. že výsledkem operace nad prvky této množiny je vždy také prvek této množiny. Např. sčítání na množině přirozených čísel.

## Kongruence

Kongruence je ekvivalence (reflexivní, symetrická a tranzitivní relace) na algebře (množina uzavřená vůči operacím). Jedná se např. o množinu **zbytkových tříd**.

## Kartézský součin

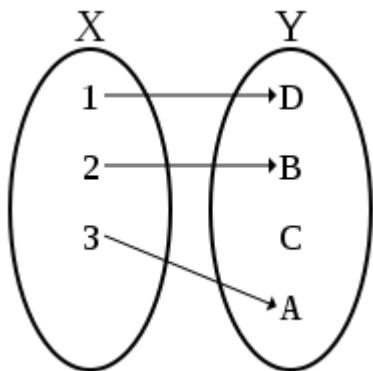
Kartézským součinem množin  $\mathbf{X}$  a  $\mathbf{Y}$  vznikne množina všech **uspořádaných dvojic**, ve kterých je první položka prvkem množiny  $\mathbf{X}$  a druhá prvkem množiny  $\mathbf{Y}$ .

$$X \times Y = \{(x, y) : x \in X \wedge y \in Y\}$$

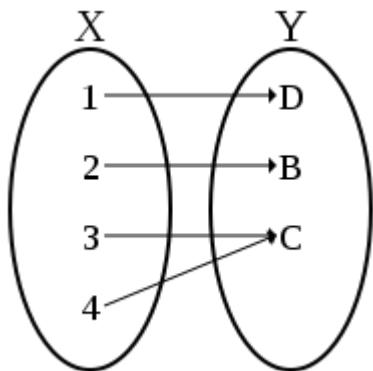
## Zobrazení

Předpis, kterým se prvkům množiny  $\mathbf{X}$  přiřazuje **nejvýše jeden** prvek množiny  $\mathbf{Y}$ . Tedy zobrazení z množiny  $\mathbf{X}$  do množiny  $\mathbf{Y}$ . Prvky  $\mathbf{X}$  se nazývají **vzory** a prvky  $\mathbf{Y}$  se nazývají **obrazy**. Podobně jako u funkce musí každému prvku z  $\mathbf{X}$  být přiřazena nanejvýš jedna hodnota. **Zobrazení je speciálním případem binární relace, u které má každý vzor nejvýše jeden obraz.**

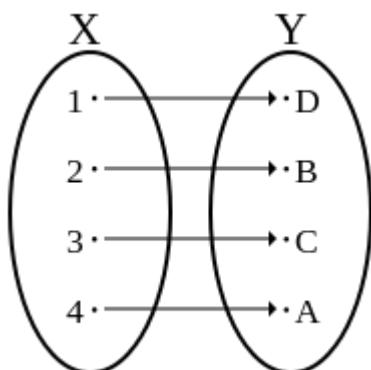
- **Injectivní (prosté) zobrazení:** Každý prvek z Y má namapován nejvíce 1 prvek z X.



- **Surjektivní zobrazení (zobrazení na):** Zobrazuje na celou cílovou množinu (nezůstane volný prvek, tedy každý prvek Y má namapovaný alespoň 1 prvek z X).



- **Bijektivní (vzájemně jednoznačné) zobrazení:** Zároveň injectivní a surjektivní zobrazení.



	surjective	non-surjective
injective		
non-injective		
surjective-only		
general		

## Inverzní zobrazení

Inverzní zobrazení k nějakému zobrazení  $f: A \rightarrow B$  přiřazuje prvkům z množiny  $B$  prvky množiny  $A$ , tedy **obrazům** zobrazení  $f$  jejich **vzory**. Zobrazení  $f$  musí být **prosté** (injektivní).

## Supremum a maximum

Nechť  $A$  je neprázdná **shora ohraničená** množina (množina, která nejde do nekonečna) reálných čísel. Číslo  $\sup(A)$  se nazývá **supremum** množiny  $A$ , jestliže je **nejmenší horní závorou** množiny  $A$ .  $\sup(A)$  musí být **větší nebo rovno** všem prvkům množiny  $A$ , nemusí být ale maximem. Např.: interval  $X = (2, 3)$  má **supremum 3** ( $\sup(X) = 3$ ), ale nemá maximum, protože se jedná o otevřený interval.  $Y = (2, 3)$  má **supremum 3**, které je současně **maximem**.

## Infimum a minimum

Nechť  $A$  je neprázdná **zdola ohraničená** množina (množina, která nezačíná v mínus nekonečnu) reálných čísel. Číslo  $\inf(A)$  se nazývá **infimum** množiny  $A$ , jestliže je **největší dolní závorou** množiny  $A$ .  $\inf(A)$  musí být **menší nebo rovno** všem prvkům množiny  $A$ , nemusí být ale maximem. Např.: interval  $X = (2, 3)$  má **infimum 2** ( $\inf(X) = 2$ ), ale nemá minimum, protože se jedná o otevřený interval.  $Y = <2, 3)$  má **infimum 2**, které je současně **minimem**.

## Svaz

Svaz je **uspořádaná množina**, která je doplněna o vlastnost, že pro **každé dva prvky** z daného svazu musí existovat **supremum a infimum**, které také naleží danému svazu.

## Grupa

Grupou nazýváme množinu  $G$  spolu s binární operací (v tomto případě značenou  $+$ ). Musí platit:

- Operace  $+$  musí být na množině  $G$  **uzavřená**, tj. musí vracet prvek z množiny  $G$ .
- Operace  $+$  musí být na množině  $G$  **asociativní**.
- **Existence neutrálního prvku**: Existuje prvek  $e \in G$  (neutrální prvek) takový, že pro všechny ostatní prvky  $a \in G$  platí:  $a + e = e + a = a$ .
- **Existence inverzních prvků**: Pro každý prvek  $a \in G$  existuje prvek  $b \in G$  takový, že  $a + b = b + a = e$ , kde  $e \in G$  je neutrální prvek.

# 17. Diferenciální a integrální počet funkcí jedné a více proměnných.

## Diferenciální počet

Matematická disciplína, která zkoumá **změny funkčních hodnot** v závislosti na změně nezávislé proměnné.

### Spojitost funkce

Spojitost funkce je její důležitá vlastnost pro určování limit a derivací. Spojitá funkce je funkce, jejíž hodnoty se **mění plynule**, tedy při dostatečně malé změně hodnoty  $x$  se hodnota  $f(x)$  změní libovolně málo. Intuitivní (ne zcela přesná) představa spojité funkce spočívá ve funkci, jejíž graf lze nakreslit jedním tahem, aniž by se tužka zvedla z papíru. Spojitost definujeme pomocí limity následovně:

- Funkce je v bodě  $x_0$  definována ( $x_0$  patří do definičního oboru).
- V bodě  $x_0$  existuje limita funkce a je rovna právě funkční hodnotě v tomto bodě.

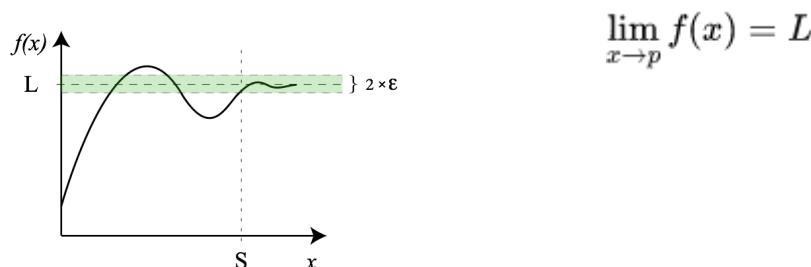
$$\lim_{x \rightarrow x_0} f(x) = f(x_0)$$

Pokud funkce není v bodě spojitá, může tato nespojitost být dvojího druhu:

- **prvního druhu:** Limity **zleva a zprava se nerovnají**, ale jsou **vlastní**.
- **druhého druhu:** Funkce má alespoň jednu **nevlastní jednostrannou limitu** nebo pokud alespoň jedna **limita neexistuje**.

### Limita

Limita je matematická konstrukce vyjadřující, že se hodnoty zadané funkce blíží **libovolně blízko** k nějakému bodu. Právě tento bod je pak označován jako limita. Funkce  $f(x)$  má v bodě  $p$  limitu  $L$ , jestliž k **libovolně zvolenému okolí** bodu  $L$  **existuje okolí** bodu  $p$  tak, že pro všechna reálná  $x \neq p$  tohoto okolí **náleží hodnoty**  $f(x)$  zvolenému okolí **bodu L**. Značíme jako



Funkce má v bodě **maximálně jednu limitu**. Funkce  $f(x)$  je spojitá v bodě  $p$ , právě když pro limitu v bodě  $p$ , platí  $f(x) = f(p)$ . Druhy limit:

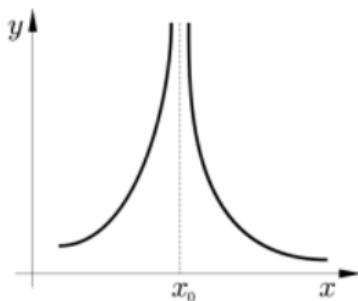
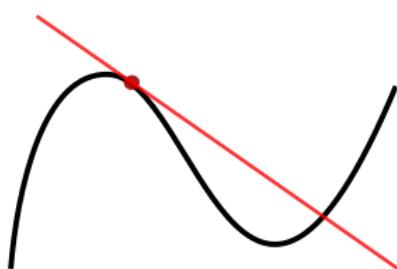
- **Vlastní** - Pokud hodnota neroste do +- nekonečna.

- **Nevlastní** - Limita roste do +- nekonečna.
- **Zprava** - Pokud se blížíme k bodu zprava na ose (z kladných čísel).
- **Zleva** - Pokud se blížíme k bodu zleva na ose (ze záporných čísel).

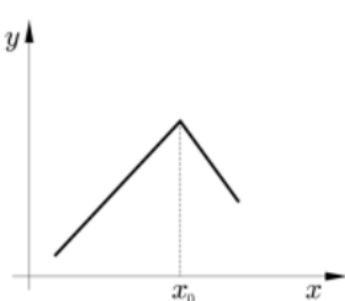
## Derivace

Derivace funkce je **změna** (růst či pokles) její hodnoty v poměru ke **změně** jejího **argumentu**, pro velmi malé změny argumentu. V případě dvourozměrného grafu funkce  $f(x)$  je **derivace této funkce** v libovolném bodě (pokud existuje) rovna **směrnici tečny tohoto grafu**. Například pokud funkce popisuje **dráhu tělesa v čase**, bude její derivace v určitém bodě udávat **okamžitou rychlosť**. Pokud popisuje **rychlosť**, bude **derivace udávat zrychlení**. Derivaci definujeme pomocí limity. Funkce  $f(x)$  je v bodě  $x$  **diferencovatelná** (má derivaci), pokud v tomto bodě derivaci. Pokud limita v bodě  $x$  neexistuje nebo je **nevlastní**, pak **není derivace** funkce  $f(x)$  v bodě  $x$  **definována**. Má-li funkce v daném bodě derivaci, pak je v tomto bodě i spojitá (naopak to **neplatí**).

$$f'(a) = \lim_{h \rightarrow 0} \frac{f(a+h) - f(a)}{h} = \lim_{x \rightarrow a} \frac{f(x) - f(a)}{x - a}$$



Derivace v  $x_0$  neexistuje  
a funkce je nespojitá.



Derivace v  $x_0$  neexistuje  
a funkce je spojitá.

### Pravidla derivování

$$[f + g]' = f' + g'$$

$$[f - g]' = f' - g'$$

$$[fg]' = f'g + fg'$$

součinové pravidlo

$$\left[\frac{f}{g}\right]' = \frac{f'g - fg'}{g^2}$$

podílové pravidlo

$$[g(f)]' = g'(f) \cdot f'$$

řetízkové pravidlo

Derivaci používáme k **vyšetření průběhu funkce**. První derivací určujeme, jestli funkce v daném bodě roste, nebo klesá. V bodech, kde je **první derivace nulová**, nebo **neexistuje** hledáme **lokální extrémy** (minima a maxima). **Druhou derivací** určujeme **konvexnost** (druhá derivace je kladná) a **konkávnost** (druhá derivace je záporná) funkcí. Body, ve kterých je druhá **derivace nulová nebo neexistuje**, jsou **inflexní body**. **Asymptoty grafu funkce** jsou přímky, ke které se graf funkce blíží v +/-nekonečnu

- **Bez směrnice**: podezřelé jsou hodnoty **vyloučené** z definičního oboru. Pro ověření je potřeba **spočítat limitu zleva a zprava** pro daný bod a pokud vyjdou **+nekonečno**, jedná se o **asymptotu bez směrnic**.
- **Se směrnicí**: Jedná se o asymptotu, která **není rovnoběžná s osou y**. Mohou existovat **maximálně 2** pro jednu funkci. ( $y = kx + q$ ). Výpočet:

$$\lim_{x \rightarrow \pm\infty} [f(x) - (ax + b)] = 0$$

$f$	$f'$	$\mathcal{D}(f)$	$\mathcal{D}(f')$	Pozn.
const.	0	$\mathbf{R}$	• (tj. jako $\mathcal{D}(f)$ )	
$x^n$	$nx^{n-1}$	$\mathbf{R}$	•	$n \in \mathbf{N}$
$x^a$	$ax^{a-1}$	$x > 0$	•	$a \in \mathbf{R}$
$e^x$	$e^x$	$\mathbf{R}$	•	
$a^x$	$a^x \ln a$	$\mathbf{R}$	•	$a > 0$
$\ln x$	$\frac{1}{x}$	$x > 0$	•	
$\log_a x$	$\frac{1}{x \ln a}$	$x > 0$	•	$a \in (0, 1) \cup (1, +\infty)$
$\sin x$	$\cos x$	$\mathbf{R}$	•	
$\cos x$	$-\sin x$	$\mathbf{R}$	•	
$\operatorname{tg} x$	$\frac{1}{\cos^2 x}$	$x \neq \frac{\pi}{2} + k\pi$	•	
$\operatorname{cotg} x$	$-\frac{1}{\sin^2 x}$	$x \neq k\pi$	•	
$\arcsin x$	$\frac{1}{\sqrt{1-x^2}}$	$(-1, 1)$	$(-1, 1)$	v $\pm 1$ : jen jednostranné derivace
$\arccos x$	$-\frac{1}{\sqrt{1-x^2}}$	$(-1, 1)$	$(-1, 1)$	v $\pm 1$ : jen jednostranné derivace
$\operatorname{arctg} x$	$\frac{1}{1+x^2}$	$\mathbf{R}$	•	
$\operatorname{arccotg} x$	$-\frac{1}{1+x^2}$	$\mathbf{R}$	•	
$\sinh x$	$\cosh x$	$\mathbf{R}$	•	
$\cosh x$	$\sinh x$	$\mathbf{R}$	•	
$\operatorname{tgh} x$	$1 - \operatorname{tgh}^2 x$	$\mathbf{R}$	•	
$\operatorname{cotgh} x$	$1 - \operatorname{cotgh}^2 x$	$x \neq 0$	•	
$\operatorname{argsinh} x$	$\frac{1}{\sqrt{x^2+1}}$	$\mathbf{R}$	•	
$\operatorname{argcosh} x$	$\frac{1}{\sqrt{x^2-1}}$	$x > 1$	•	
$\operatorname{argtgh} x$	$\frac{1}{1-x^2}$	$-1 < x < 1$	•	
$\operatorname{argcotgh} x$	$\frac{1}{1-x^2}$	$ x  > 1$	•	

## Integrální počet

Integrální počet je část matematiky, která se zabývá především **integrací**. Integrály se využívají pro hledání **ploch**, **objemů** a délek **křivek**. Integraci můžeme také považovat **za proces sčítání**, dokonce tak lze definovat **určitý integrál**.

$$\int_a^b f(x) dx = \lim_{\delta x \rightarrow 0} \sum_{x=a}^b f(x) \delta x$$

**Neurčitý integrál** definujeme pomocí derivace takto: Nechť  $I$  je otevřený interval,  $f(x)$  a  $F(x)$  funkce na něm definované, pokud  $F'(x) = f(x)$ , pak se funkce  $F(x)$  nazývá primitivní funkcí k funkci  $f(x)$ , nebo též **neurčitý integrál funkce  $f(x)$  na intervalu  $I$** .

$$\int f(x) dx = F(x) + c.$$

## Neurčitý integrál

Operace integrování je **opačná** k operaci derivování. Integrál ale **není inverzní** funkcí k derivaci, protože **derivováním ztrácíme informaci o konstantní** (stejnosměrné) složce derivované funkce. Abychom integrováním zderivované funkce získali funkci původní, musíme znát její hodnotu **alespoň v jednom bodě**. Pomocí této hodnoty můžeme **vypočítat konstantní (c)** složku.

## Určitý integrál

Slouží pro výpočet **povrchu, objemu** nebo **obvodu** geometrického útvaru (výpočet je omezen na určitou část). Aby byla funkce integrovatelná nemusí být **spojitá** na celém intervalu nebo **spojitá po částech** (měla **konečně** mnoho bodů nespojitosti).

$$\int_a^b f(x)dx = [F(x)]_a^b = F(b) - F(a) \quad \int_a^b f(x)dx = - \int_b^a f(x)dx.$$

## Postupy integrace

- **Metoda per partes:** využívá se pro integraci funkcí v součinovém tvaru dle vzorce (pro úspěšnou integraci velmi **záleží** na správném výběru **u a v**):

$$\int u \cdot v' dx = u \cdot v - \int u' \cdot v dx$$

Příklad integrace metodou per partes:

$$\begin{aligned} \int x \cdot \cos x dx & \quad \left[ \begin{array}{l} u = x, \quad v' = \cos x \\ u' = 1, \quad v = \sin x \end{array} \right] \quad \int x \cdot \cos x = x \cdot \sin x - \int 1 \cdot \sin x \\ & \quad \int x \cdot \cos x = x \cdot \sin x - \int 1 \cdot \sin x \\ & \quad = x \cdot \sin x - (-\cos x) \\ & \quad = x \cdot \sin x + \cos x + c \end{aligned}$$

- **Substituční metoda:** Během integrování nahrazujeme část integrované funkce za jinou, provedeme integraci a vrátíme funkci zpět do původního tvaru. U **určitého integrálu** je **nutné přepočítat meze integrace**.

$$\int f(\phi(t)) \cdot \phi'(t) dt = \int f(x) \cdot x' dx$$

Příklad integrace substituční metodou:

$$\int \sin 6t dt \quad \int \sin 6t dt = \left[ \begin{array}{l} x = 6t \\ dx = 6dt \end{array} \right] \int \sin x \frac{dx}{6} = \frac{1}{6} \cdot \int \sin x dx \quad \frac{1}{6} \cdot \int \sin x dx = \frac{1}{6} \cdot (-\cos x)$$

Příklad integrace určitého integrálu:

$$\int_0^{\pi/2} e^{\sin(x)} \cos(x) dx = \left| \begin{array}{l} y = \sin(x) \\ dy = \cos(x) dx \\ x = 0 \mapsto y = \sin(0) = 0 \\ x = \pi/2 \mapsto y = \sin(\pi/2) = 1 \end{array} \right| = \int_0^1 e^y dy = [e^y]_0^1 = e - 1.$$

$f$	$\int f \frac{dx}{x}$	Pozn.	Kde
$x^n$	$\frac{x^{n+1}}{n+1}$	$n \in \mathbf{Z}, n \neq -1$	$x \in \mathbf{R}$ pro $n \geq 0, x \in \mathbf{R} \setminus \{0\}$ pro $n < 0$
$x^a$	$\frac{x^{a+1}}{a+1}$	$a \in \mathbf{R} \setminus \mathbf{Z}$	$x \in (0, +\infty)$
$\frac{1}{x}$	$\ln x $		$x \in \mathbf{R} \setminus \{0\}$
$e^x$	$e^x$		$x \in \mathbf{R}$
$a^x$	$\frac{a^x}{\ln a}$	$a > 0, a \neq 1$	$x \in \mathbf{R}$
$\sin x$	$-\cos x$		$x \in \mathbf{R}$
$\cos x$	$\sin x$		$x \in \mathbf{R}$
$\operatorname{tg} x$	$-\ln \cos x $		$x \in \mathbf{R} \setminus \{\frac{\pi}{2} + k\pi, k \in \mathbf{Z}\}$
$\operatorname{cotg} x$	$\ln \sin x $		$x \in \mathbf{R} \setminus \{k\pi, k \in \mathbf{Z}\}$
$\frac{1}{\csc^2 x}$	$\operatorname{tg} x$		$x \in \mathbf{R} \setminus \{\frac{\pi}{2} + k\pi, k \in \mathbf{Z}\}$
$\frac{1}{\sin^2 x}$	$-\operatorname{cotg} x$		$x \in \mathbf{R} \setminus \{k\pi, k \in \mathbf{Z}\}$
$\arcsin x$	$x \arcsin x + \sqrt{1-x^2}$		$x \in (-1, 1)$
$\arccos x$	$x \arccos x - \sqrt{1-x^2}$		$x \in (-1, 1)$
$\operatorname{arctg} x$	$x \operatorname{arctg} x - \frac{1}{2} \ln(x^2 + 1)$		$x \in \mathbf{R}$
$\operatorname{arccotg} x$	$x \operatorname{arccotg} x + \frac{1}{2} \ln(x^2 + 1)$		$x \in \mathbf{R}$
$\frac{1}{\sqrt{1-x^2}}$	$\arcsin x$		$x \in (-1, 1)$
$\frac{1}{\sqrt{1-x^2}}$	$-\arccos x$		$x \in (-1, 1)$
$\frac{1}{1+x^2}$	$\operatorname{arctg} x$		$x \in \mathbf{R}$
$\frac{1}{1+x^2}$	$-\operatorname{arccotg} x$		$x \in \mathbf{R}$
$\sinh x$	$\cosh x$		$x \in \mathbf{R}$
$\cosh x$	$\sinh x$		$x \in \mathbf{R}$
$\operatorname{tgh} x$	$\ln(\cosh x)$		$x \in \mathbf{R}$
$\operatorname{cotgh} x$	$\ln \sinh x $		$x \in \mathbf{R} \setminus \{0\}$
$\frac{1}{\sqrt{x^2+1}}$	$\operatorname{argsinh} x$		$x \in \mathbf{R}$
$\frac{1}{\sqrt{x^2-1}}$	$\operatorname{sign} x \arg \cosh  x $		$ x  > 1$
$\frac{1}{1-x^2}$	$\arg \operatorname{tgh} x$	Pozor na def. obor!	$-1 < x < 1$
$\frac{1}{1-x^2}$	$\arg \operatorname{cotgh} x$	Pozor na def. obor!	$ x  > 1$

## Více proměnných

### Parciální derivace

Parciální derivace funkce o **více proměnných** je její derivace vzhledem k **jedné** z těchto proměnných, přičemž s **ostatními** proměnnými se zachází jako s **konstantami**. **Většinou** platí:

$$F''_{yx} = F''_{xy}$$

Pomocí parciálních derivací se např. určuje tečná rovina grafu dvou proměnných

$$z = f(x_0, y_0) + f'_x(x_0, y_0)(x - x_0) + f'_y(x_0, y_0)(y - y_0)$$

## Gradient (parciální derivace)

Jedná se o **vektor prvních parciálních derivací** dle **všech** proměnných funkce, který určuje směr **největšího růstu** funkce (respektive největšího poklesu, pokud jej vezmeme záporně). Délka vektoru gradientu je nárůst veličiny **f** na intervalu jednotkové délky.

$$\vec{\text{grad}} F = (F'_x; F'_y)$$

Umožňuje vypočítat **derivaci** funkce více proměnných ve **směru nějakého vektoru**, viz [19 - Gradient a jeho využití \(MAT - Diferenciální počet funkcí více proměnných\)](#). Směrová derivace se vypočítá jako skalární součin vektoru **u** (vektor udávající směr) a gradientu v bodě **A** (lze vyjádřit i obecně a poté dosadit libovolný bod).

$$s = \vec{\text{grad}} F_A \cdot \vec{u}_{\text{jed}}$$

Gradient je kolmý na křivky (u funkcí 2 proměnných) a plochy (u funkcí 3 proměnných) o **stejné funkční hodnotě - hladiny funkce**. Jedná se např. o **vrstevnice** na mapách.

V bodech, kde je gradient **nulový vektor**, se mohou nacházet **lokální extrémy** funkce více proměnných nebo **sedlové body**. Jedná se o stacionární body. Lokální **maximum** se v bodě nachází, pokud existuje i gradient druhé parciální derivace a jeho **hodnota je menší než 0**, respektive je zde lokální **minimum**, pokud je **hodnota větší než 0**.

$$\begin{aligned} d^2 f(A) < 0, \text{ funkce } f \text{ má v bodě } A \text{ ostré lokální maximum,} \\ d^2 f(A) > 0, \text{ funkce } f \text{ má v bodě } A \text{ ostré lokální minimum.} \end{aligned}$$

$$d^2 f(A) = \frac{\partial^2 f}{\partial x^2}(A) dx^2 + 2 \frac{\partial^2 f}{\partial x \partial y}(A) dx dy + \frac{\partial^2 f}{\partial y^2}(A) dy^2.$$

$$d^2 f(A) = (dx \ dy) \cdot \begin{pmatrix} \frac{\partial^2 f}{\partial x^2}(A) & \frac{\partial^2 f}{\partial x \partial y}(A) \\ \frac{\partial^2 f}{\partial x \partial y}(A) & \frac{\partial^2 f}{\partial y^2}(A) \end{pmatrix} \cdot \begin{pmatrix} dx \\ dy \end{pmatrix}$$

## Dvojný integrál

Jedná se o integrál funkce **dvou proměnných**. Používá se např. pro výpočet **objemu**, výpočet **hmotnosti nehomogenní plochy**, výpočet **povrchu**. Při výpočtu se snažíme **dvojný integrál** převést na **integrál dvojnásobný**, tj. **dva jednoduché integrály** a integraci provádět postupně (Fubiniova věta). Hranicí vnitřního integrálu budou tvořit funkce.

$$\iint_I f(x, y) dx dy = \int_a^b \left( \int_{f(x)}^{g(x)} f(x, y) dy \right) dx \quad \iint_I x y dx dy = \int_0^2 \left( \int_{x^2}^{2x} x y dy \right) dx$$

## Trojný integrál

Jedná se o integrál funkce **tří proměnných**. Používá se např. pro výpočet **objemu** tělesa ohrazeného třemi křivkami, výpočet jeho **hmotnosti**, na základě proměnlivé hustoty, **statické momenty**, aj.

$$V = \iiint_M 1 \, dx dy dz \quad m = \iiint_M \rho_{x,y,z} \, dx dy dz$$

Výpočet **trojnáho integrálu** opět provádíme jeho **převodem na trojnásobný integrál**.

$$\iiint_B f(x, y, z) dx dy dz = \int_a^b \left( \int_{\varphi(x)}^{\psi(x)} \left( \int_{\Phi(x,y)}^{\Psi(x,y)} f(x, y, z) dz \right) dy \right) dx.$$

## Příklady dvojného a trojnáho integrálu

- **dvojný integrál na čtverci:**  $M = \langle 0, 3 \rangle \times \langle 0, 1 \rangle$

$$\begin{aligned} \int_M \frac{x^2}{3+y^2} dA &= \int_0^3 \int_0^1 \frac{x^2}{3+y^2} dy dx = \int_0^3 \left[ \frac{x^2}{\sqrt{3}} \operatorname{arctg} \frac{y}{\sqrt{3}} \right]_{y=0}^{y=1} dx = \\ &= \frac{\pi\sqrt{3}}{18} \int_0^3 x^2 dx = \frac{\pi\sqrt{3}}{2}. \end{aligned}$$

- **dvojný integrál - Fubiniova věta:**

**Řešení:**  $M$  je ohraničená přímkou  $y = -x$  a parabolou  $y = x - x^2$  (Obr. 1).

Souřadnice průsečíků obou křivek získáme řešením soustavy dvou rovnic

$$\begin{aligned} y &= -x, \\ y &= x - x^2. \end{aligned}$$

Řešením této soustavy zjistíme, že křivky se protnou v bodech  $(0, 0)$  a  $(2, -2)$ . Funkce  $f(x, y) = xy$  je na  $M$  spojitá a je zřejmé, že pro libovolné  $x \in \langle 0, 2 \rangle$  je  $-x \leq y \leq x - x^2$ . Užitím Fubiniové věty pak dostaváme

$$\begin{aligned} \int_M xy dA &= \int_0^2 \int_{-x}^{x-x^2} xy dy dx = \int_0^2 \left[ \frac{1}{2}xy^2 \right]_{y=-x}^{y=x-x^2} dx = \\ &= \frac{1}{2} \int_0^2 (x(x-x^2)^2 - x^3) dx = -\frac{16}{15}. \end{aligned}$$

- **trojný integrál na čtverci:**

$$W = \langle 0, 1 \rangle \times \langle 1, 2 \rangle \times \langle 2, 3 \rangle$$

$$\begin{aligned} \int_W (2x - y + z) dV &= \int_0^1 \int_1^2 \int_2^3 (2x - y + z) dz dy dx = \\ &= \int_0^1 \int_1^2 \left[ 2xz - yz + \frac{z^2}{2} \right]_{z=2}^{z=3} dy dx = \int_0^1 \int_1^2 \left( 2x - y + \frac{5}{2} \right) dy dx = \\ &= \int_0^1 \left[ 2xy - \frac{y^2}{2} + \frac{5}{2}y \right]_{y=1}^{y=2} dx = \int_0^1 (2x + 1) dx = 2. \end{aligned}$$

- **trojný integrál - Fubiniova věta:**

Zapíšeme-li množinu  $M$  ve tvaru

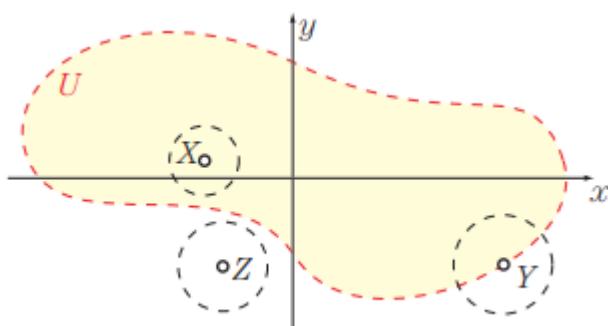
$$M = \{(x, y) \in \mathbb{R}^2 : 0 \leq x \leq 1 \wedge 0 \leq y \leq 1 - x\},$$

můžeme použitím Fubiniové věty pro dvojný integrál náš trojný integrál převést na trojnásobný integrál. Potom

$$\begin{aligned} \int_W \frac{1}{1+x+y} dV &= \int_0^1 \int_0^{1-x} \int_0^{1-x-y} \frac{1}{1+x+y} dz dy dx = \\ &= \int_0^1 \int_0^{1-x} \left[ \frac{z}{1+x+y} \right]_{z=0}^{z=1-x-y} dy dx = \int_0^1 \int_0^{1-x} \frac{1-x-y}{1+x+y} dy dx = \\ &= \int_0^1 [2 \ln(1+x+y) - y]_{y=0}^{y=1-x} dx = \int_0^1 (2 \ln 2 - 2 \ln(x+1) + x-1) dx = \\ &= \frac{3}{2} - 2 \ln 2. \end{aligned}$$

### Limity funkcí více proměnných

Limity funkcí více proměnných fungují na **podobném** principu jako limity funkce jedné proměnné. Limita funkce **existuje**, když hodnoty **libovolně malého okolí** bodu **spadají do ohraničeného pásu funkčních hodnot**. U funkce **dvou** proměnných bude okolím **kruh**, u tří proměnných **koule**. Limitu funkce více proměnných v bodě  $\mathbf{p} \in \mathbf{U}$  lze počítat pouze, pokud je tento bod **bodem hromadným** ( $X$  a  $Y$  na obr.) - bod jehož **každé prstencové** okolí má s množinou  $\mathbf{U}$  **neprázdný průnik**. Body, které toto nesplňují, jsou body **izolované** ( $Z$  na obr.) a v nich **limitu funkce více proměnných počítat nelze**.



- Pokud máme funkci spojitou v limitním bodě, pak lze hodnotu limity získat pouhým dosazením bodu do funkčního předpisu.

**Příklad 3.1.1.** Vypočtěte:

$$\lim_{(x,y) \rightarrow (1,2)} \frac{x^3y - xy^3 + 1}{(x-y)^2}.$$

*Řešení:* Do lomeného výrazu dosadíme bod  $(1, 2)$ :

$$\lim_{(x,y) \rightarrow (1,2)} \frac{x^3y - xy^3 + 1}{(x-y)^2} = \frac{2 - 2^3 + 1}{(-1)^2} = -5.$$

- **Otevřená množina** - Neobsahuje žádný bod ze své hranice.
- **Uzavřená množina** - Obsahuje všechny body své hranice.
- **Vnitřní bod** - Bod uvnitř množiny.
- **Hraniční bod** - Bod na hranici množiny.
- **Hranice množiny** - Množina všech hraničních bodů.

## Další info

### L'Hôpitalovo pravidlo

Umožňuje v některých případech vypočítat limitu podílu dvou funkcí. Říká, že limita podílu dvou funkcí, které splňují jisté předpoklady, je rovna limitě podílu derivací těchto funkcí.

$$\lim_{x \rightarrow a} \frac{f(x)}{g(x)} = \lim_{x \rightarrow a} \frac{f'(x)}{g'(x)}$$

### Taylorova řada

Mocninná řada, vyjádřená jako **suma derivací funkce v bodě**. Pokud se jedná o rozvoj v **okolí bodu 0**, mluvíme o **Maclaurinově řadě**. Taylorovy a Maclaurinovy se využívají k approximaci funkcí.

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \dots = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k$$

### Odkazy:

- [Neurcity integral](#)
- [Integrace substitucí](#)
- [Spojitost funkce](#)
- [Funkce více proměnných](#)
- [VZORCE PRO INTEGROVÁNÍ](#)
- Derivace: [Tabulky](#)

# 18. Číselné soustavy a převody mezi nimi.

**Číselné soustavy** vyjadřují způsob reprezentace čísel. Podle způsobu určení hodnoty čísla z dané reprezentace rozlišujeme dva hlavní druhy číselných soustav: poziční číselné soustavy a nepoziční číselné soustavy. **Dnes** se obvykle používají **soustavy poziční**. Zápis čísla dané soustavy je posloupností symbolů, které se nazývají číslice.

## Nepoziční číselná soustava

Číselné soustavy, ve kterých **není** hodnota číslice dána jejím umístěním v dané sekvenci číslic. Tyto způsoby zápisu čísel se **dnes** již téměř **nepoužívají** a jsou považovány za zastaralé.

### Římská číselná soustava (římské číslice)

Římská číslice	I	V	X	L	C	D	M
Hodnota	1	5	10	50	100	500	1000

Římané psali číslici **4 jako IIII**, **40 jako XXXX** atd. tento zápis opravdu umožňuje zápis **číslic na libovolné pozice**. Pravidlo pro odečtení, že 4 se zapisuje jako IV, 40 jako XL atd., se začalo používat až ve středověku, komplikuje ale umístění číslic (největší se tak psala vlevo, nejmenší vpravo). Např. číslo **78** se běžně zapisuje jako **LXXVIII**. Lze ale zapsat také např. jako XXVLIII, XXIIILV - uvažujeme pravidlo odečtení a libovolně, pokud jej neuvažujeme např. IXLXIVI, IIVILXL, ... Poziční zápis v římské číselné soustavě není možný, protože neexistuje symbol pro **nulu**.

- **Převod z římské do desítkové:** jednotlivé číslice vynásobíme jejich hodnotou a poté sečteme. Např. **LXVXI = 50 + 10 + 5 + 10 + 1 = 76**.
- **Převod z desítkové do římské:** musíme najít první číslici, jejíž hodnota je menší, než hodnota čísla v desítkové soustavě. Zapsat ji tolikrát, kolikrát se do čísla vejde, od čísla odečít zapsanou hodnotu a pokračovat další číslicí. Např. převod čísla **32**: L - nevleze se, X - vleze se (3x), V - nevleze se, I - vleze se (2x) → **XXXII**.

## Poziční číselné soustavy

Převládající způsob písemné reprezentace čísel. V tomto způsobu zápisu čísel je **hodnota každé číslice** dána její **pozici v sekvenci symbolů**. Všechny poziční soustavy **musí** mít symbol pro **nulu**. Celá část je oddělena od zlomkové speciálním znakem (zpravidla řádovou čárkou či tečkou). Nejběžnější poziční číselnou soustavou je soustava **desítková**. Pro zápis hodnoty čísla v libovolné soustavě můžeme použít polynomiální zápis.

$$(132)_{16} = 2 \cdot 16^0 + 3 \cdot 16^1 + 1 \cdot 16^2 = 2 + 48 + 256 = 306$$

$$A = a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + \dots + a_i \cdot 10^i + \dots + a_1 \cdot 10^1 + a_0 \cdot 10^0$$

## Základ (báze, radix) číselné soustavy

Číslo definující **maximální počet číslic**, které jsou v dané soustavě k **dispozici**.

- **dvojková** (binární, **r=2**) – přímá implementace v digitálních elektronických obvodech (použitím logických členů).
- **osmičková** (oktální, oktalová, **r=8**)
- **desítková** (decimální, dekadická, **r=10**) – nejpoužívanější v běžném životě
- **dvanáctková (r=12)** – dnes málo používaná, ale dodnes z ní zbyly názvy prvních dvou řádů – **tucet** a **veletucet**.
- **šestnáctková** (hexadecimální, **r=16**) – používá se v oblasti informatiky, pro číslice 10 až 15 se používají písmena **A** až **F**.
- **šedesátková (r=60)** – používá se k měření času pro zlomky hodiny; číslice se obvykle zapisují desítkovou soustavou jako 00 až 59 a řády se oddělují **dvoječkou**. Staré názvy prvních dvou řádů jsou **kopa** a **velekopa**.

## Převody mezi pozičními soustavami

Běžný postup při převodu čísel mezi dvěma číselnými soustavami je **převod přes desítkovou soustavu**. Pokud však **základ jedné soustavy** je **mocninou základu soustavy druhé**, lze postupovat i **přímo**.

### Substituční metoda převodu soustav

Lze použít pro libovolnou soustavu, pro člověka je však nejjednodušší pro převod do desítkové soustavy.

- Číslo se vyjádří polynomiálním zápisem v cílové soustavě (dekadická).
- Vypočítají se členy polynomu a sečtou.

$$\begin{aligned} & (10011,011)_2 \\ & = (1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3})_2 \\ & = (16 + 2 + 1 + 0,25 + 0,125)_{10} \\ & = (19,375)_{10} \end{aligned}$$

## Metoda dělení základem

Vhodná pro převod celých čísel.

$$(109)_{10} / 2 = 54 \text{ zb. } 1(\text{LSB})$$

$$(54)_{10} / 2 = 27 \text{ zb. } 0$$

$$(27)_{10} / 2 = 13 \text{ zb. } 1$$

$$(13)_{10} / 2 = 6 \text{ zb. } 1$$

$$(6)_{10} / 2 = 3 \text{ zb. } 0$$

$$(3)_{10} / 2 = 1 \text{ zb. } 1$$

$$(1)_{10} / 2 = 0 \text{ zb. } 1(\text{MSB})$$

$$(109)_{10} = (1101101)_2$$

Protože dělení je pro člověka náročná operace, lze použít způsob, kdy hledáme největší mocninu cílové soustavy, která je menší než převáděná hodnota, zjistíme, kolikrát se do převáděného čísla vejde a odečteme tento násobek od převedeného čísla - získanou číslici zapíšeme na správné pozici čísla v soustavě, do které převádíme. Postup opakujeme, dokud nevyčerpáme všechny pozice cílového čísla. (Vhodné zejména u **dvojkové** soustavy, ve které známe hodnoty jednotlivých mocnin 2)

## Metoda násobení základem

Vhodná pro **převod desetinných čísel**. Číslo se násobí základem soustavy, do které ho převádíme. Po každém kroku se sepíše celočíselná část. Končíme po **dosažení 0** nebo **požadované přesnosti**.

$$(0,6875)_{10} \cdot 2 = 1,375 = 1 + 0,375 = b_{-1} + 0,375 \text{ (MSB)}$$

$$(0,375)_{10} \cdot 2 = 0,75 = 0 + 0,75 = b_{-2} + 0,75$$

$$(0,75)_{10} \cdot 2 = 1,5 = 1 + 0,5 = b_{-3} + 0,5$$

$$(0,5)_{10} \cdot 2 = 1,0 = 1 + 0,0 = b_{-4} + 0,0 \text{ (LSB)}$$

$$(0,6875)_{10} = (0,1011)_2$$

## Substituční převod desetinných čísel

Postupuje se stejně jako s celými čísly, ale tentokrát je lepší čísla odečítat a jak se dostaneme na desetinnou čárku, použijí se **záporné mocniny**.

$$32,625 - 2^5 = 0,625 \quad (1)$$

$$0,625 - 2^4 = 0,625 \quad (0)$$

$$0,625 - 2^3 = 0,625 \quad (0)$$

$$0,625 - 2^2 = 0,625 \quad (0)$$

$$0,625 - 2^1 = 0,625 \quad (0)$$

$$0,625 - 2^0 = 0,625 \quad (0)$$

desetinná čárka

$$0,625 - 2^{-1} = 0,125 \quad (1)$$

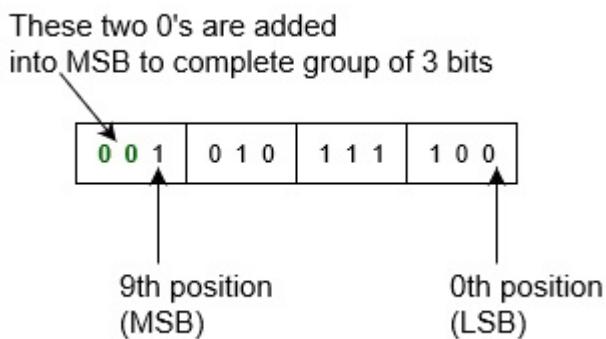
$$0,125 - 2^{-2} = 0,125 \quad (0)$$

$$0,125 - 2^{-3} = 0 \quad (1)$$

Výsledek bude tedy 100000,101

## Převod mezi soustavami s mocninným rozdílem základů

- **Převod mezi dvojkovou a osmičkovou soustavou:** na zapsání všech osmičkových číslic stačí přesně **3 bity**. Stačí číslo rozdělit na **trojice** (číslo se doplní na násobek 3 přidáním nul před číslo - nezmění hodnotu) a každou zapsat jako **osmičkovou číslici**, respektive z osmičkové číslice vytvářet **trojice binárních číslic**. Např. převod binárního čísla 1010111100:



$$(1010111100)_2 = (001|010|111|100)_2 = (1|2|7|4)_8 = (1274)_8$$
$$(736)_8 = (7|3|6)_8 = (111|011|110)_2 = (111011110)_2$$

- **Převod mezi dvojkovou a šestnáctkovou soustavou:** na zapsání všech šestnáctkových číslic stačí přesně **4 bity**. Stačí číslo rozdělit na **nibly** (čtveřice) a každý zapsat jako **šestnáctkovou číslici**, respektive z šestnáctkové číslice vytvářet **nibly** (čtveřice binárních číslic).  
 $(0001|0011|0010|1010)_2 = (1|3|2|A)_{16}$  $(8|A|F)_{16} = (1000|1010|1111)_2$

# 19. Výroková logika a predikátová logika. Syntaxe a sémantika výrokové logiky. Splnitelnost a platnost. Logická ekvivalence a logický důsledek.

Normální formy. Jazyk predikátové logiky prvního řádu. Syntaxe, termy a formule, volné a vázané proměnné.

## Výroková logika

Výroková logika tvoří formální odvozovací systém, ze syntaktických a odvozovacích pravidel. <https://www.fit.vutbr.cz/~lengal/idm-2021/vyrokova-logika.pdf>

### Syntax výrokové logiky

Syntax je dán abecedou a gramatikou:

- **abeceda:** Je tvořena spočetně nekonečnou množinu výrokových proměnných, logickými spojkami (logické symboly  $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ ) a logickými konstantami (**0** a **1**).  $X = \{x, y, z, \dots, x_1, x_2, \dots, 0, 1, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, (, )\}$
- **gramatika:** vyjadřuje pravidla, jak můžeme tvořit formule výrokové logiky.
  - Je-li  $x$  výroková proměnná, tj.  $x \in X$ , pak **x, 0 a 1 jsou formule**.
  - Jsou-li  $\phi$  a  $\psi$  formule, pak jsou formule i  $(\neg\phi)$ ,  $(\phi \wedge \psi)$ ,  $(\phi \vee \psi)$ ,  $(\phi \rightarrow \psi)$  a  $(\phi \leftrightarrow \psi)$ .
  - Formule výrokové logiky jsou právě všechny konečné řetězce získané pomocí předchozích dvou pravidel.

**Pozor**, výroková logika **neuvážuje** bez rozšíření **precedenci** jednotlivých **logických symbolů**.

### Sémantika výrokové logiky

Sémantika formule určuje její **význam** a určuje, jakou **pravdivostní hodnotu** formule nabude pro **jednotlivá ohodnocení proměnných**. Tato hodnota se často definuje pomocí pravdivostní tabulky. Pravdivostní tabulka pro dvě formule  $\phi$  a  $\psi$  říká, jaká bude výsledná hodnota formule získané aplikací dané **logické spojky** na  $\phi$  a  $\psi$ .

$\varphi$	$\psi$	$\neg\varphi$	$\varphi \vee \psi$	$\varphi \wedge \psi$	$\varphi \rightarrow \psi$	$\varphi \leftrightarrow \psi$
0	0	1	0	0	1	1
0	1	1	1	0	1	0
1	0	0	1	0	0	0
1	1	0	1	1	1	1

Splňuje, splnitelnost, platnost, nesplňuje, neplatnost, nesplnitelnost

V následujících bodech uvažujeme ohodnocení proměnných  $I : X \rightarrow \{0, 1\}$ .

- $I \models \varphi$  • **I splňuje  $\phi$**  -  $I \models \phi$ : zjišťujeme na základě daného ohodnocení proměnných  $I$  formule  $\phi$ . Pokud je pro toto ohodnocení hodnota  $\phi 1$ , je  $I$  modelem formule  $\phi$  a formuli splňuje.
- **splnitelná**: Formule je splnitelná, pokud existuje nějaké ohodnocení proměnných  $I$  takové, že  $I \models \phi$  (pro nějaké ohodnocení  $I$   $I$  splňuje  $\phi$ ). Z pravdivostní tabulky poznáme **splnitelnou formuli** tak, že se ve sloupci značící  $\phi$  vyskytuje **alespoň jednou hodnota 1**.
- $\models \varphi$  • **platná, tautologie,  $\models \phi$** : formule je platná (tautologie) pokud je splněna libovolným ohodnocením proměnných. Pomocí pravdivostní tabulky můžeme platnou formuli poznat tak, že v **posledním sloupci** (sloupec s  $\phi$ ) tabulky jsou **samé hodnoty 1**.
- $I \not\models \varphi$  • **I nesplňuje  $\phi$**  -  $I \not\models \phi$ : zjišťujeme na základě daného ohodnocení proměnných  $I$  formule  $\phi$ . Pokud je pro toto ohodnocení hodnota  $\phi 0$ , není  $I$  modelem formule  $\phi$  a formuli nesplňuje.
- $\not\models \varphi$  • **neplatná,  $\not\models \phi$** : formule  $\phi$  je neplatná pokud existuje **ohodnocení** proměnných, které je **nesplňuje**. Pomocí pravdivostní tabulky takovou formuli poznáme tak, že by v posledním sloupci (sloupec s  $\phi$ ) je **alespoň jedna hodnota 0**.
- **nesplnitelná, kontradikce**: formule je nesplnitelná, pokud **není splnitelná**, tj. ve sloupci s  $\phi$  pravdivostní tabulky jsou **samé 0**.

$$\varphi_1: (x \wedge y) \rightarrow x$$

$\varphi_1$  je *platná i splnitelná*

$$\varphi_2: (x \vee y) \rightarrow x$$

$\varphi_2$  je *neplatná, ale splnitelná*

$$\varphi_3: (x \vee y) \wedge \neg(x \vee y)$$

$\varphi_3$  je *neplatná a nesplnitelná*

$x$	$y$	$x \wedge y$	$(x \wedge y) \rightarrow x$	$x \vee y$	$(x \vee y) \rightarrow x$	$\neg(x \vee y)$	$(x \vee y) \wedge \neg(x \vee y)$
0	0	0	1	0	1	1	0
0	1	0	1	1	0	0	0
1	0	0	1	1	1	0	0
1	1	1	1	1	1	0	0

Logická ekvivalence a logický důsledek

- $\varphi \Leftrightarrow \psi$  • **logická ekvivalence**: Dvě formule  $\phi$  a  $\psi$  jsou (logicky) **ekvivalentní**, zapisováno  $\phi \Leftrightarrow \psi$ , pokud pro **všechna ohodnocení** proměnných  $I$  platí, že  $I$  je modelem (splňuje)  $\phi$  právě tehdy, když  $I$  je modelem (splňuje)  $\psi$ .
- **logický důsledek**: Formule  $\psi$  je **logickým důsledkem** formule  $\phi$ , zapisováno  $\phi \Rightarrow \psi$ , tehdy, když pro **každé ohodnocení** proměnných  $I$  platí, že je-li  $I$  modelem (splňuje)  $\phi$ , pak je  $I$  rovněž modelem (splňuje)  $\psi$ .

**Neplést si s logickými spojkami implikace a bikondicionál**, i když je význam podobný. Implikace a bikondicionál se používají uvnitř formulí, logický ekvivalence logický důsledek mezi formulemi.

## Normální formy

Jedná se o formule, které splňují jistá syntaktická omezení.

### Negační normální forma (NNF)

Formule je v NNF, pokud:

1. obsahuje jen následující logické spojky: **0, 1,  $\neg$ ,  $\wedge$ ,  $\vee$**  a
2. **negace  $\neg$**  se vyskytuje jen **před proměnnými**.

Jedná se o literály (proměnné nebo negace proměnných) spojené konjunkcemi a disjunkcemi. Formule v NNF:

$$\begin{aligned} & x \wedge \neg y, \\ & (x \vee y) \wedge (\neg z \vee (\neg x \wedge \neg y)) \\ & x. \end{aligned}$$

Postup převodu obecné formule na formulu v NNF:

1. **Přepíšeme** postupně všechny **bikondicionály**  $\leftrightarrow$  ve formuli **za implikace**.

$$x \leftrightarrow y \rightsquigarrow (x \rightarrow y) \wedge (y \rightarrow x)$$

2. **Přepíšeme** postupně všechny **implikace**  $\rightarrow$  ve formuli **za negaci a disjunkci**.

$$x \rightarrow y \rightsquigarrow \neg x \vee y$$

3. Pomocí **De Morganových** zákonů postupně **přesuneme negaci** co **nejhlouběji**.

$$\begin{aligned} \neg(x \wedge y) &\rightsquigarrow \neg x \vee \neg y \\ \neg(x \vee y) &\rightsquigarrow \neg x \wedge \neg y \end{aligned}$$

4. Kdykoliv to jde, **eliminujeme dvojitou negaci** (dvojitá negace v NNF není povolena).

$$\neg\neg x \rightsquigarrow x$$

### Disjunktivní normální forma (DNF)

Formule je v DNF (sum of products, SoP) v případě, že je zapsána jako disjunkce konjunkcí literálů (konjunkce jsou uvnitř závorek, disjunkce na nejvyšší úrovni). V případě DNF se **konjunkcí** literálů se říká **klauzule**. Používá se následující znáčení, kde **i,j** je literál:

$$\bigvee_i \bigwedge_j \ell_{i,j}$$

Příklady formulí v DNF:

$$\begin{aligned}
 & (x \wedge y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge z) \vee (x \wedge \neg y \wedge z), \\
 & (x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge \neg x_5) \vee (x_2 \wedge \neg x_3 \wedge x_5), \\
 & (x \wedge y) \vee \neg z, \\
 & x \wedge \neg y, \\
 & x \vee \neg y, \\
 & 1. \\
 & 0.
 \end{aligned}$$

Postup převodu obecné formule na formuli v DNF:

1. Převedeme obecnou formuli na formuli v NNF.
2. Formuli v NNF převedeme do tvaru, kde jsou **všechny konjunkce pod disjunkcemi** pomocí **distributivního zákona**.

$$x \wedge (y \vee z) \rightsquigarrow (x \wedge y) \vee (x \wedge z)$$

Konjunktivní normální forma (CNF)

Formule je v CNF (product of sums, PoS) v případě, že je zapsána jako konjunkce disjunkcí literálů (disjunkce jsou uvnitř závorek, konjunkce na nejvyšší úrovni). V případě CNF se disjunkcí literálů se říká **klauzule**. Používá se následující znáčení, kde **i,j** je literál:

$$\bigwedge_i \bigvee_j \ell_{i,j}$$

Příklady formulí v CNF:

$$\begin{aligned}
 & (x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (x \vee \neg y \vee z), \\
 & (x_1 \vee x_2 \vee x_3 \vee x_4 \vee \neg x_5) \wedge (x_2 \vee \neg x_3 \vee x_5), \\
 & (x \vee y) \wedge \neg z, \\
 & x \vee \neg y, \\
 & x \wedge \neg y, \\
 & 0. \\
 & 1.
 \end{aligned}$$

Postup převodu obecné formule na formuli v CNF:

1. Převedeme obecnou formuli na formuli v NNF.
2. Formuli v NNF převedeme do tvaru, kde jsou všechny **disjunkce pod konjunkcemi** pomocí **distributivního zákona**.

$$x \vee (y \wedge z) \rightsquigarrow (x \vee y) \wedge (x \vee z)$$

# Jazyk predikátové logiky prvního řádu

Oproti výrokové logice **poskytuje** predikátová logika mnohem bohatší vyjadřovací prostředky. <https://www.fit.vutbr.cz/~lengal/idm-2021/predikatova-logika.pdf>

## Abeceda predikátové logiky

1. **logické spojky:**  $\neg, \wedge, \vee, \rightarrow, \leftrightarrow,$
2. **proměnné:**  $x, y, z, \dots, x_1, x_2, \dots \in X$ , kde  $X$  je spočetně nekonečná množina proměnných
3. **kvantifikátory:**  $\exists, \forall,$
4. **závorky:**  $),(),$
5. **funkční symboly:**  $f_1, f_2, \dots \in F (+_{/2}, \sin_{/1}, e_{/0}),$
6. **predikátové symboly:**  $p_1, p_2, \dots (<_{/2}, \text{isnan}_{/1}),$
7. **predikátový symbol rovnosti:**  $=_{/2}.$

**Funkční a predikátové symboly** mají **aritu**, která říká, s kolika parametry (operandy) daný symbol pracuje.

## Gramatika predikátové logiky

- **termy:**
  - Je-li  $x$  proměnná ( $x \in X$ ), pak řetězec “ $x$ ” je **term**.
  - Je-li  $f$  funkční symbol s aritou  $n$  a  $t_1, \dots, t_n$  jsou **termy**, pak i řetězec „ $f(t_1, \dots, t_n)$ “ je term.
- Jazyk teorie grup se signaturou  $\langle \mathcal{F} = \{\cdot_{/2}, e_{/0}\}, \mathcal{P} = \emptyset \rangle$ . Příklady termů v tomto jazyce jsou následující:
  - „ $(z \cdot (x \cdot e)) \cdot y$ “,
  - „ $e$ “,
  - „ $x$ “ a
  - „ $e \cdot e$ “.
- **formule:**
  - Je-li  $p$  predikátový symbol s aritou  $n$  a  $t_1, \dots, t_n$  jsou **termy**, potom je řetězec „ $p(t_1, \dots, t_n)$ “ **formule** (toto platí i pro „vestavěný“ binární predikátový symbol  $=$ ). Formuli tohoto tvaru říkáme **atomická formule**.
  - Jsou-li  $\phi$  a  $\psi$  **formule**, pak jsou **formule** i řetězce „ $(\neg\phi)$ “, „ $(\phi \wedge \psi)$ “, „ $(\phi \vee \psi)$ “, „ $(\phi \rightarrow \psi)$ “ a „ $(\phi \leftrightarrow \psi)$ “.
  - Je-li  $\phi$  formule a  $x \in X$  proměnná, pak jsou formule i řetězce „ $(\exists x\phi)$ “ a „ $(\forall x\phi)$ “.

Jazyk teorie uspořádání se signaturou  $\langle \mathcal{F} = \emptyset, \mathcal{P} = \{\leq_{/2}\} \rangle$ :

- $\forall x(x \leq x)$
- $\forall x\forall y((x \leq y \wedge y \leq x) \rightarrow x = y)$
- $\forall x\forall y\forall z((x \leq y \wedge y \leq z) \rightarrow x \leq z)$
- $\forall x\forall y(x \leq y \vee y \leq x)$

## Syntax predikátové logiky

Syntax predikátové logiky tvoří jí abeceda a gramatika. Navíc funkční a predikátové symboly nejsou **pevně zafixovány**, ale lze je chápout jako *parametr* jazyka, který si volíme podle toho, co chceme v logice **vyjádřit**. Jedná se o signaturu jazyka predikátové logiky, která je dána jako dvojice **<množina funkčních symbolů, množina predikátových symbolů>**. Příklady jazyků predikátové ligiky.

1. Jazyk teorie uspořádání se signaturou  $\langle \mathcal{F} = \emptyset, \mathcal{P} = \{\leq_{/2}\} \rangle$ :

- $\forall x(x \leq x)$
- $\forall x\forall y((x \leq y \wedge y \leq x) \rightarrow x = y)$
- $\forall x\forall y\forall z((x \leq y \wedge y \leq z) \rightarrow x \leq z)$
- $\forall x\forall y(x \leq y \vee y \leq x)$

2. Jazyk teorie grup se signaturou  $\langle \mathcal{F} = \{\cdot_{/2}, e_{/0}\}, \mathcal{P} = \emptyset \rangle$ :

- $\forall x(x \cdot e = x \wedge e \cdot x = x)$
- $\forall x\exists y(x \cdot y = e \wedge y \cdot x = e)$

3. Jazyk teorie množin se signaturou  $\langle \mathcal{F} = \emptyset, \mathcal{P} = \{\in_{/2}\} \rangle$ :

- $\forall u(u \in x \rightarrow u \in y)$
- $\forall x\exists y\forall z(\forall u(u \in z \rightarrow u \in x) \rightarrow z \in y)$

4. Jazyk teorie polí se signaturou  $\langle \mathcal{F} = \{\text{read}_{/2}, \text{write}_{/3}\}, \mathcal{P} = \emptyset \rangle$ :

- $\forall x\forall y(\forall i(\text{read}(x, i) = \text{read}(y, i)) \rightarrow x = y)$

5. Jazyk elementární (tzv. Peanovy) aritmetiky se signaturou  $\langle \mathcal{F} = \{0_{/0}, S_{/1}, +_{/2}, \cdot_{/2}\}, \mathcal{P} = \emptyset \rangle$ :

- $\forall x\forall y\exists z(x + y = z)$
- $\forall x\forall y(x \cdot S(y) = x \cdot y + x)$
- $\exists x\forall y(\neg(x = S(y)))$

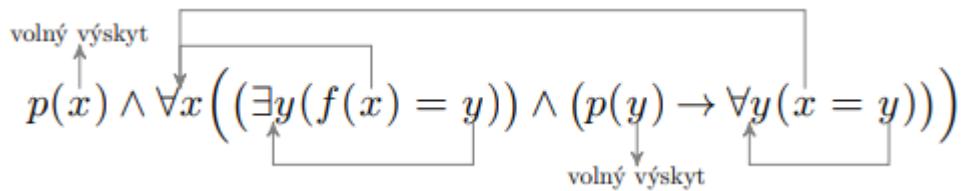
## Vázané proměnné

Výskyt proměnné je ve formuli vázaný, pokud se nachází v oboru platnosti kvantifikátoru  $\exists$  nebo  $\forall$ . Pokud je výskyt proměnné vázaný, pak je vázaný nejbližším kvantifikátorem nad sebou. Příklad oborů platnosti jednotlivých kvantifikátorů:

$$p(x) \wedge \forall x \left( (\exists y(f(x) = y)) \wedge (p(y) \rightarrow \forall y(x = y)) \right)$$

## Volné proměnné

Volné proměnné jsou takové, které **nejsou vázané žádným kvantifikátorem**. Z předchozí formule jsou to tyto:



Proměnná je ve formuli **volná**, pokud v ní **alespoň jeden volný výskyt**. Formuli s **volnými proměnnými** říkáme **výroková forma**, formuli **bez volných proměnných** říkáme **uzavřená formule** nebo také **výrok**.

## Sémantika predikátové logiky

Sémantika predikátové logiky je podstatně **komplikovanější** než u výrokové logiky. V predikátové logice musíme proměnným **přiřazovat hodnoty** z nějakého univerza a musíme **interpretovat funkční a predikátové symboly** (provádět **realizaci** jazyka). Např. abychom určili, jestli formule platí, musíme znát:

$$\forall x (f(y) < x)$$

- jakou hodnotu bude mít proměnná **y**,
- jaké všechny prvky bude uvažovat **kvantifikátor „ $\forall x$ “**,
- jaká je **sémantika funkčního symbolu „ $f/1$ “** a
- jaká je **sémantika predikátového symbolu „ $</2$ “**.

Příklady různých realizací jazyka:

Uvažujme jazyk predikátové logiky se signaturou  $\langle \mathcal{F} = \{+/_2\}, \mathcal{P} = \emptyset \rangle$ . Následují příklady realizací tohoto jazyka:

1. Realizace  $I_1$  modelující sčítání přirozených čísel, kde
  - $D_{I_1} = \mathbb{N}$  a
  - $I_1(+)=+_N$  (tj. sčítání přirozených čísel:  $+_N = \{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 1, (0,2) \mapsto 2, \dots\}$ ).
2. Realizace  $I_2$  modelující sčítání tříprvkových vektorů reálných čísel, kde
  - $D_{I_2} = \mathbb{R}^3$  a
  - $I_2(+)=\{([x_1,y_1,z_1],[x_2,y_2,z_2]) \mapsto [x_1 +_{\mathbb{R}} x_2, y_1 +_{\mathbb{R}} y_2, z_1 +_{\mathbb{R}} z_2] \mid x_1, x_2, y_1, y_2, z_1, z_2 \in \mathbb{R}\}$ , kde  $+_{\mathbb{R}}$  je sčítání reálných čísel.
3. Realizace  $I_3$  modelující spojení (supremum) v Booleově algebře nad  $\{0,1\}$ .
  - $D_{I_3} = \{0,1\}$  a
  - $I_3(+)=\{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 1, (1,1) \mapsto 1\}$ .
4. Realizace  $I_4$  modelující konkatenaci řetězců v jazyce Python, kde
  - $D_{I_4} = \text{str}$  (datový typ pro řetězec v jazyce Python) a
  - $I_4(+)=\{("ab", "cd") \mapsto "abcd", ("foo", "bar") \mapsto "foobar", \dots\}$

# 20. Boolovy algebry.

**Booleova algebra** je algebraická struktura se dvěma binárními ( $\wedge$ ,  $\vee$ ) a jedním unárním operátorem ( $\neg$ ). Jedná se o **distributivní komplementární svaz**.

Booleova algebra je **šestice** ( $B$ ,  $\wedge$ ,  $\vee$ ,  $\neg$ ,  $0$ ,  $1$ ), kde:

- $B$  je neprázdná množina,
- $0 \in B$  a je nejmenší prvek,
- $1 \in B$  a je největší prvek,
- $\wedge$  je **binární operace průsek** (infimum),
- $\vee$  je **binární operace spojení** (supremum),
- $\neg$  je **unární operace doplňku**.

## Axiomy

- **Komutativita:** Binární operace u níž **nezáleží na pořadí prvků**.

$$x \vee y = y \vee x \quad x \wedge y = y \wedge x$$

- **Distributivita:** Binární operaci je možné distribuovat přes jinou operaci.

$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z) \quad x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$

Operaci **průsek** je možné **distribuovat přes** operaci **spojení** a operaci **spojení přes** operaci **průsek**.

- **Asociativita:** u operace nezáleží, jak jsou použity závorky u více operandů (někde nebývá jako axiom).

$$a \vee (b \vee c) = (a \vee b) \vee c \quad a \wedge (b \wedge c) = (a \wedge b) \wedge c$$

- **Neutralita 0 a 1:** **0** je neutrální prvek pro operaci **spojení** a **1** je neutrální prvek pro operaci **průsek**.

$$x \vee 0 = x \quad x \wedge 1 = x$$

- **Komplementárnost:** existence doplňku (komplimentu)

$$x \vee \neg x = 1 \quad x \wedge \neg x = 0$$

- **Absorpce:** (někde nebývá jako axiom)

$$a \vee (a \wedge b) = a \quad a \wedge (a \vee b) = a$$

## Vlastnosti

- **Agresivita nuly** - Při průseku jakéhokoliv prvku s 0, je výsledek 0.

$$x \wedge 0 = 0$$

- **Agresivita jedničky** - Při spojení jakéhokoliv prvku s 1, je výsledek 1.

$$x \vee 1 = 1$$

- **Idempotence** - Opakovaným použitím na nějaký vstup vznikne stejný výstup.

Tedy jakákoli binární operace prvku se sebou samým má za výsledek ten stejný prvek.

$$x \vee x = x$$

$$x \wedge x = x$$

- **Absorpce negace**  
 $x \vee (\neg x \wedge y) = x \vee y, x \wedge (\neg x \vee y) = x \wedge y$
- **Dvojitá negace**  
 $\neg(\neg x) = x$
- **0 a 1 jsou vzájemně komplementární**  
 $\neg 0 = 1, \neg 1 = 0$
- **De Morganovy zákony**  
 $\neg x \wedge \neg y = \neg(x \vee y), \neg x \vee \neg y = \neg(x \wedge y)$

## Příklady Booleových algeber

- **triviální algebry:**
  - **0 = 1** (obsahují pouze jeden prvek)
  - všechny operace dávají stejný výsledek
- **algebry výroků:**
  - **0 - false** (nepravda), **1 - true** (pravda),
  - $\wedge$  - konjunkce,  $\vee$  - disjunkce,  $\neg$  - negace
- **množinová algebra:**
  - **0 - prázdná množina** ( $\emptyset$ ), **1 - univerzum** ( $U$ ),
  - $\wedge$  - průnik,  $\vee$  - sjednocení,  $\neg$  - doplněk
- **algebry elektrických obvodů:**
  - **0 - low** (log. 0), **1 - high** (log. 1)
  - $\wedge$  - **AND**,  $\vee$  - **OR**,  $\neg$  - **NOT**

## Funkce

Na Booleově algebrách lze realizovat  **$2^2 \cdot 2^2 = 16$**  funkcí.

Funkce	Název funkce	Logický člen	Algebraický výraz
$f_0$	konstanta		0
$f_1$	logický součet	NEBO(OR,ODER)	$a + b$
$f_2$	implikace		$\bar{a} + b$
$f_3$	implikace		$a + \bar{b}$
$f_4$	Shefferova funkce	NAND	$\bar{a} + \bar{b} = \overline{ab}$
$f_5$	logický součin	A ( AND,UND)	$ab$
$f_6$	inhibice		$\bar{a}b$
$f_7$	inhibice		$a\bar{b}$
$f_8$	Pierceova funkce	NOR	$\bar{a} \cdot \bar{b} = \overline{a + b}$
$f_9$	identita		$a$
$f_{10}$	identita		$b$
$f_{11}$	ekvivalence		$ab + a\bar{b}$
$f_{12}$	neekvivalence	Exclusive OR	$\bar{a}b + a\bar{b}$
$f_{13}$	negace	NE (NOT,NICHT)	$\bar{a}$
$f_{14}$	negace	NE (NOT,NICHT)	$\bar{b}$
$f_{15}$	konstanta		1

- NOT - Negace (doplňek)

Funkce	$Y = \bar{A}$	
Značení		Pravdivostní tabulka
norma	symbol	
ANSI/MIL		
IEC		$\begin{array}{ c c } \hline X(A) & Y \\ \hline 0 & 1 \\ 1 & 0 \\ \hline \end{array}$
DIN		

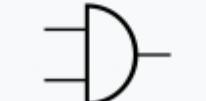
- Opakovač - Realizuje funkci identity. Lze použít jako buffer.

Funkce	$Y = A$	
Značení		Pravdivostní tabulka
norma	symbol	
ANSI/MIL		
IEC		$\begin{array}{ c c } \hline X(A) & Y \\ \hline 0 & 0 \\ 1 & 1 \\ \hline \end{array}$
DIN		

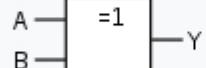
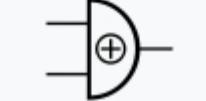
- OR - Disjunkce (spojení)

Funkce	$Y = A + B$	
Značení		Pravdivostní tabulka
norma	symbol	
ANSI/MIL		$\begin{array}{ c c c } \hline X_1(A) & X_2(B) & Y \\ \hline 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ \hline \end{array}$
IEC		
DIN		

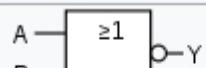
- **AND** - Konjunkce (průnik)

Funkce	$Y = A \cdot B$		
Značení		Pravdivostní tabulka	
norma	symbol	$X_1(A)$	$X_2(B)$
ANSI/MIL		0	0
IEC		0	1
DIN		1	0
		1	1

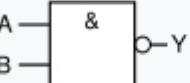
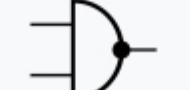
- **XOR** - negovaná ekvivalence - **Exkluzivní disjunkce**

Funkce	$Y = A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$		
Značení		Pravdivostní tabulka	
norma	symbol	$X_1(A)$	$X_2(B)$
ANSI/MIL		0	0
IEC		0	1
DIN		1	0
		1	1

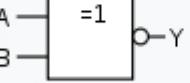
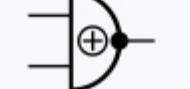
- **NOR** - Negovaná disjunkce - **Peirceova funkce**

Funkce	$Y = \overline{A + B} = \overline{A} \cdot \overline{B}$		
Značení		Pravdivostní tabulka	
norma	symbol	$X_1(A)$	$X_2(B)$
ANSI/MIL		0	1
IEC		0	0
DIN		1	0
		1	1

- **NAND** - Negovaná konjunkce - **Shefferova funkce**

Funkce	$\mathbf{Y} = \overline{\mathbf{A} \cdot \mathbf{B}} = \overline{\mathbf{A}} + \overline{\mathbf{B}}$	
Značení		Pravdivostní tabulka
norma	symbol	
ANSI/MIL		$X_1(A)$
IEC		$X_2(B)$
DIN		$Y$

- **XNOR** - negovaná exkluzivní disjunkce - **Ekvivalence**

Funkce	$\mathbf{Y} = \overline{\mathbf{A} \oplus \mathbf{B}} = \mathbf{A} \cdot \mathbf{B} + \overline{\mathbf{A}} \cdot \overline{\mathbf{B}}$	
Značení		Pravdivostní tabulka
norma	symbol	
ANSI/MIL		$X_1(A)$
IEC		$X_2(B)$
DIN		$Y$

### Shefferova funkce

Pomocí Shefferovy funkce lze vyjádřit všechny ostatní funkce Booleovy algebry, viz:

$$\neg p \equiv \neg(p \wedge p)$$

$$p \wedge q \equiv \neg(\neg(p \wedge q)) \equiv \neg(\neg(p \wedge q) \wedge \neg(p \wedge q))$$

$$p \vee q \equiv \neg(\neg p \wedge \neg q) \equiv \neg(\neg(p \wedge p) \wedge \neg(q \wedge q))$$

$$p \rightarrow q \equiv \neg p \vee q \equiv \neg(p \wedge \neg q) \equiv \neg(p \wedge \neg(p \wedge q))$$

### Peirceova funkce

Pomocí Piercovy funkce lze také vyjádřit všechny ostatní funkce Booleovy algebry, viz:

$$\neg p \equiv \neg(p \vee p)$$

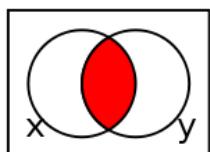
$$p \wedge q \equiv \neg(\neg p \vee \neg q) \equiv \neg(\neg(p \vee p) \vee \neg(q \vee q))$$

$$p \vee q \equiv \neg(\neg(p \vee q)) \equiv \neg(\neg(p \vee p) \vee \neg(q \vee q))$$

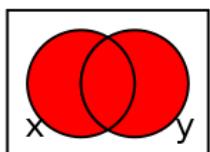
$$p \rightarrow q \equiv \neg p \vee q \equiv \neg(p \vee p) \vee q \equiv \neg(\neg(p \vee p) \vee q) \vee \neg(\neg(p \vee p) \vee q))$$

## Vennovy diagramy

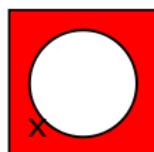
Grafické znázornění příslušnosti prvků do množin.



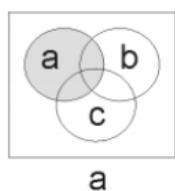
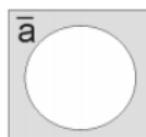
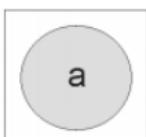
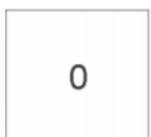
$$x \wedge y$$



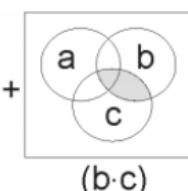
$$x \vee y$$



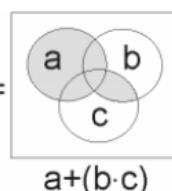
$$\neg x$$



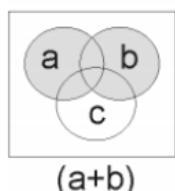
$$a$$



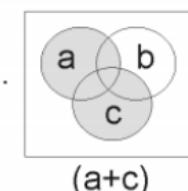
$$(b \cdot c)$$



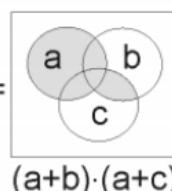
$$a + (b \cdot c)$$



$$(a+b)$$



$$(a+c)$$



$$(a+b) \cdot (a+c)$$

## Switching algebra

Jiný výraz pro booleovu algebru a vyjadřuje to, že v tomto systému (dvojkový systém) jsou prováděny všechny operace (**AND**, **OR**, **NOT**). Tedy použitím booleovy algebry děláme výpočty ve dvojkové soustavě.

# 21. Regulární jazyky a jejich modely (konečné automaty, regulární výrazy).

## Regulární jazyk

Regulární jazyky jsou **nejjednodušší formální jazyky** v rámci **Chomského hierarchie**. Nad abecedou  $\Sigma$  je lze zavést následovně:

- prázdný jazyk  $\emptyset$  je regulární.
- pro každé  $a, a \in \Sigma$ , jazyk  $\{ a \}$  je regulární.
- pokud  $A$  a  $B$  jsou regulární jazyky, jsou  $A \cup B$  (sjednocení),  $A \cdot B$  (konkatenace), a  $A^*$  (iterace, umožňuje jazyk prázdného řetězce  $\{ \epsilon \}$ ) také **regulární**.
- žádné **další** jazyky regulární **nejsou**.

Dále platí, že jazyk jazyk je regulární, pokud:

- Existuje **konečný automat** (deterministický i nedeterministický), který **akceptuje** právě všechna slova z tohoto jazyka (**akceptuje/přijímá** tento jazyk).
- Existuje **regulární výraz**, který tento jazyk značí.
- Existuje **regulární gramatika**, která jej **generuje**.

## Jazyk

Nechť  $\Sigma^*$  značí **množinu** všech **řetězců** nad  $\Sigma$ . Každá **podmnožina**  $L \subseteq \Sigma^*$  je **jazyk** nad  $\Sigma$ . Tedy podmnožina řetězců abecedy. **Počet** všech **slov** jazyka je jeho **kardinalita**.

- **Konečný a nekonečný** - Jazyk  $L$  je **konečný**, pokud  $L$  obsahuje **konečný počet řetězců**, jinak je nekonečný.
- **Operace nad jazyky**
  - **Sjednocení** - Stejně jako u množin
  - **Průnik** - Stejně jako u množin
  - **Rozdíl** - Stejně jako u množin
  - **Doplňek** - Stejně jako u množin
  - **Konkatenace** - Nechť  $L_1$  a  $L_2$  jsou dva **jazyky** nad  $\Sigma$ . Konkatenace jazyků  $L_1$  a  $L_2$  je definována jako  $L = L_1L_2 = \{xy: x \in L_1 \text{ a } y \in L_2\}$   $L = L_1L_2 = \{0, 1\}.\{2, 3\} = \{02, 03, 12, 13\}$ . **Pozor**  $L\{\epsilon\} = \{\epsilon\}L = L$ , ale  $\emptyset L = L\emptyset = \emptyset$ .
  - **Reverzace** - Nechť  $L$  je jazyk nad abecedou  $\Sigma$ . Reverzace jazyka  $L$ , **reverse(L)**, je definována:  $\text{reverse}(L) = \{\text{reverse}(x): x \in L\}$ . Jedná se o převrácení slov abecedy.  $\text{reverse}(\{02, 03, 12, 13\}) = \{20, 30, 21, 31\}$ .

## Abeceda

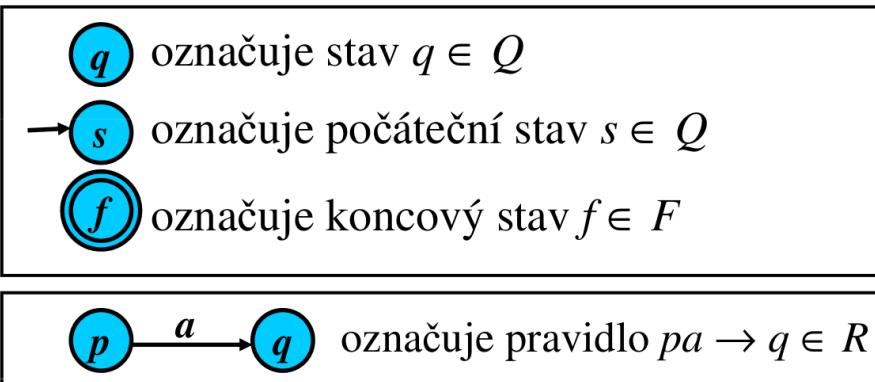
Konečná, neprázdná množina elementů, které nazýváme symboly.

## Konečný automat (KA)

Konečný automat (KA) je pětice  $M = (Q, \Sigma, R, s, F)$ , kde:

- $Q$  je konečná množina stavů,
- $\Sigma$  je vstupní abeceda,
- $R$  je konečná množina pravidel tvaru:  $pa \rightarrow q$ , kde  $p, q \in Q, a \in \Sigma \cup \{\epsilon\}$ ,
- $s \in Q$  je počáteční stav,
- $F \subseteq Q$  je množina koncových stavů.

Graficky KA značíme následovně:



## Přijímaný jazyk KA

Pokud s nějakou vstupní posloupností znaků se dostaneme až do **konečného stavu** automatu, pak automat tento **jazyk přijímá**.

## Ekvivalentní KA

Dva modely pro popis formálních jazyků (např. konečné automaty) jsou ekvivalentní, pokud specifikují **tentýž jazyk**.

## Typy konečných automatů

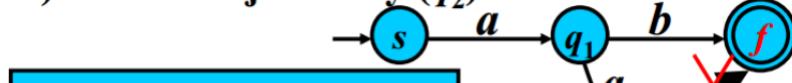
- **Nedeterministický**: Může obsahovat  $\epsilon$  přechody a existují stavy, ze kterých lze s načtením **stejného symbolu** přejít do **více stavů**.
- **Bez epsilon přechodů** - Neobsahuje  $\epsilon$  přechody.
- **Deterministický** - Neobsahuje  $\epsilon$  přechody a z každého **stavu** může přejít **maximálně** do **jednoho** dalšího s načtením **stejného znaku**.
- **Úplný deterministický** - Pro **každý znak** abecedy existuje **právě jeden** přechod v **každém stavu** (nepotřebné směřují do **false stavu**). **Nemůže se zaseknout**.
- **Dobře specifikovaný** - **Nemá nedostupné** stavy a má **maximálně jeden neukončující** stav (false stav). Pro každý konečný automat existuje ekvivalentní dobré specifikovaný konečný automat.

### 1) Nedostupné stavy ( $q_2$ ):



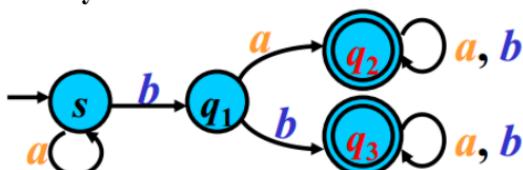
Neexistuje posloupnost přechodů z počátečního stavu do nedostupného stavu

### 2) Neukončující stavy ( $q_2$ ):



Neexistuje posloupnost přechodů z neukončujícího stavu do koncového stavu

- Minimální** - Pokud obsahuje pouze rozlišitelné stavy. Pro dobře specifikovaný KA existuje právě jeden minimální KA



- $s$  a  $q_1$  jsou **rozlišitelné**, protože např. pro  $w = a$ :

$$\begin{aligned} sa &\vdash s, s \notin F \\ q_1 a &\vdash q_2, q_2 \in F \end{aligned}$$

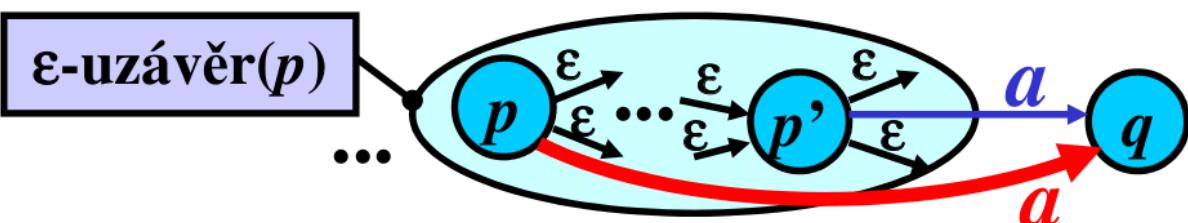
- $q_2$  a  $q_3$  jsou **nerozlišitelné**, protože pro každé  $w \in \Sigma^*$ :

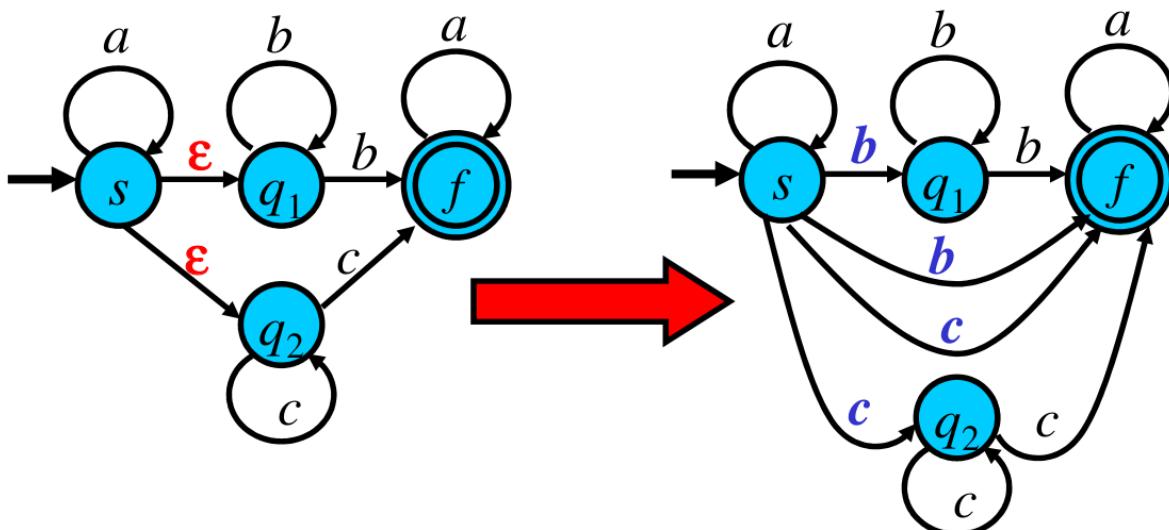
$$\begin{aligned} q_2 w &\vdash^* q_2, q_2 \in F \\ q_3 w &\vdash^* q_3, q_3 \in F \end{aligned}$$

## Determinizace KA

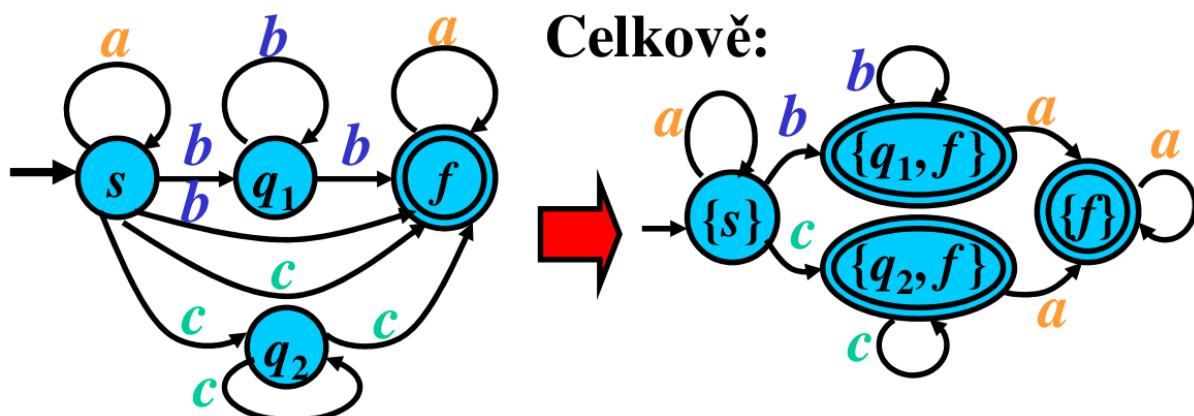
Převod nedeterministického KA na deterministický. Pokud má nedeterministický KA **n** stavů, má jeho deterministická varianta nejvíce  **$2^n$  stavů**. Každý nedeterministický automat má svoji deterministickou variantu. Postup:

1. Odstranění  $\epsilon$  přechodů pomocí  $\epsilon$  uzávěru - množina stavů, do kterých se můžeme dostat přečtením libovolného symbolu z abecedy.





2. Z počátečního stavu vytváříme nové stavě tak, že tyto stavě jsou množinou stavů, do kterých se můžeme dostat přečtením libovolného symbolu (prázdné stavě neuvažujeme). Z takto vzniklých nových stavů postupujeme dále obdobně (je nutné dát pozor, aby byly brány v potaz všechny výstupy vzniklé množiny stavů). Za nové koncové stavě označíme ty, které obsahují alespoň jeden původní koncový stav.



Shrnutí:

	KA	KA bez $\epsilon$ -přech	DKA	Úplný KA	DSKA
Počet všech pravidel tvaru $p \rightarrow q$ , kde $p, q \in Q$	0-n	0	0	0	0
Počet pravidel tvaru $pa \rightarrow q$ , pro libovolné $p \in Q$ a libovolné $a \in \Sigma$	0-n	0-n	0-1	1	1
Počet všech nedostupných stavů	0-n	0-n	0-n	0-n	0
Počet všech neukončujících stavů	0-n	0-n	0-n	0-n	0-1
Počet všech možných těchto automatů pro jeden regulární jazyk	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$

## Regulární výraz (RV)

Jedná se o výrazy s operátory “.”, “+”, “\*”, které značí v tomto pořadí **konkatenaci**, **sjednocení** a **iteraci**. Nechť  $\Sigma$  je abeceda. Regulární výrazy nad abecedou  $\Sigma$  a **jazyky**, které **značí**, jsou definovány následovně:

- $\emptyset$  je **RV** značící prázdnou množinu (**prázdný jazyk**),
- $\epsilon$  je **RV** značící jazyk  $\{\epsilon\}$ ,
- $a$ , kde  $a \in \Sigma$ , je **RV** značící jazyk  $\{a\}$ ,
- Nechť  $r$  a  $s$  jsou regulární výrazy značící po řadě jazyky  $L_r$  a  $L_s$ , potom:
  - $(r.s)$  je RV značící jazyk  $L = L_r L_s$ ,
  - $(r+s)$  je RV značící jazyk  $L = L_r \cup L_s$ ,
  - $(r^*)$  je RV značící jazyk  $L = L_r^*$ .

Závorky lze redukovat zavedením priority operátorů

**Priority:**  $*$  >  $.$  >  $+$

RV  $r.s$  může být zapsán jako  $rs$

RV  $rr^*$  nebo  $r^*r$  může být zapsán jako  $r^+$

Příklady RV:

$r_1 = ab + ba$	značí $L_1 = \{ab, ba\}$
$r_2 = a^+b^*$	značí $L_2 = \{a^n b^m : n \geq 1, m \geq 0\}$
$r_3 = ab(a + b)^*$	značí $L_3 = \{x : ab \text{ je prefix } x\}$
$r_4 = (a + b)^*ab(a + b)^*$	značí $L_4 = \{x : ab \text{ je podřetězec } x\}$

### Prázdné slovo ( $\epsilon$ )

Prázdný řetězec, který ale vyhovuje jazyku (je možné jej vygenerovat a přijmout).

### Řetězec

Jakákoliv posloupnost terminálních a neterminálních symbolů (znaků).

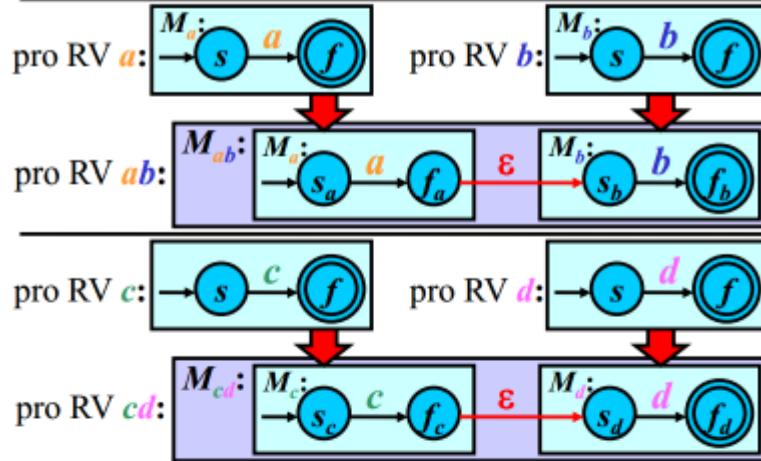
Nechť  $\Sigma$  je abeceda.

- $\epsilon$  je řetězec nad abecedou  $\Sigma$
- pokud  $x$  je řetězec nad  $\Sigma$  a  $s \in \Sigma$ , potom  $xs$  je **řetězec** nad abecedou  $\Sigma$

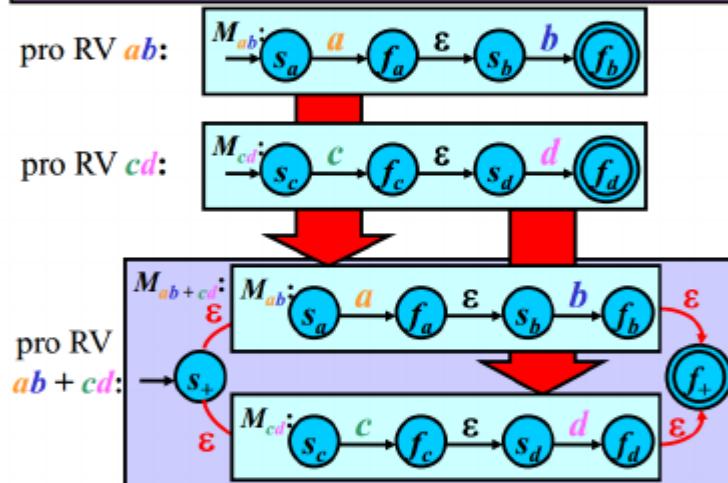
## Převod regulárního výrazu na konečný automat

### Převod z RV na KA: Příklad 1/3

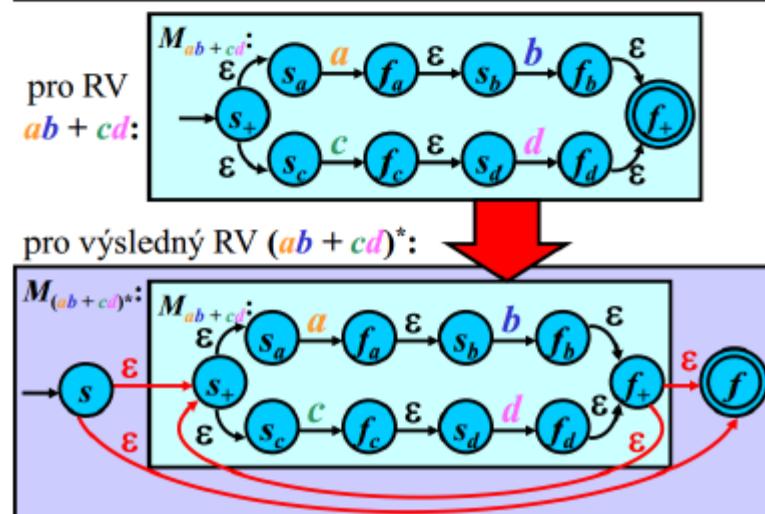
Převeďme RV  $r = ((ab) + (cd))^*$  na ekvivalentní KA  $M$



### Převod z RV na KA: Příklad 2/3



### Převod z RV na KA: Příklad 3/3



## Regulární gramatika

Každá **regulární gramatika popisuje regulární jazyk** Je to čtveřice: (**T, N, P, S**)

- T - Konečná množina terminálních symbolů,
- N - Konečná množina neterminálních symbolů,
- P - Konečná množina pravidel, ve tvaru:
  - $X \rightarrow wY$  (w je řetězec)
  - $X \rightarrow w$
  - $S \rightarrow \epsilon$ , poté se však nesmí S vyskytovat na pravé straně pravidla.
- S - Počáteční symbol.

Existují **pravé** ( $X \rightarrow wY$ ) a **levé** ( $X \rightarrow Yw$ ) **regulární gramatiky**, které jsou si **ekvivalentní**.

Příklady:

Příklad:  $G = (\{a, b\}, \{A, B, C\}, P, A)$

$P = \{$

- $A \rightarrow \epsilon \mid aB$
- $B \rightarrow bC$
- $C \rightarrow a$

$\}$

$A \rightarrow Bb$

$B \rightarrow \epsilon/Ba/Bb$

It derives the language that contains all the strings which end with b.  
i.e.  $L' = \{b, bb, ab, aab, bab, abb, bbb \dots\}$

## Pumping lemma

Používá se k dokázání, že jazyk **NENÍ REGULÁRNÍ** (používá se důkaz sporem), nemůže tedy dokázat, že jazyk je regulární. Nechť  $L$  je **RJ**. Pak existuje  $k \geq 1$  takové, že: pokud  $z \in L$  a  $|z| \geq k$ , pak existuje  $u, v, w: z = uvw$ ,

- $v \neq \epsilon$
- $|uv| \leq k$
- pro každé  $m \geq 0$ ,  $uv^mw \in L$

▶ Pumping Lemma (For Regular Languages) | Example 1

## Odkazy

- Definice z IFJ: [IFJ Teorie](#)
- [Minimalizace a kanonizace, nedeterministické konečné automaty a determinizace](#)

# 22. Bezkontextové jazyky a jejich modely (zásobníkové automaty, bezkontextové gramatiky).

## Bezkontextový jazyk (BkJ)

Jedná se o formální jazyk, pro který platí:

- Generuje jej **bezkontextová gramatika**,
- Přijímá (akceptuje) jej **zásobníkový automat**.

Nejznámějším bezkontextovým jazykem je jazyk  $L(G) = \{(a^n)(b^n) : n \geq 0\}$

## Bezkontextová gramatika

Slouží pro **generování BkJ**. Je to **čtverice  $G = (N, T, P, S)$** , kde:

- $N$  - abeceda **neterminálů**,
- $T$  - abeceda **terminálů**, přičemž  $N \cap T = \emptyset$ ,
- $P$  - konečná **množina pravidel** tvaru  $A \rightarrow x$ , kde  $A \in N$ ,  $x \in (N \cup T)^*$ ,
- $S$  - počáteční neterminál,  $S \in N$ .

$$G = (N, T, P, S), \text{ kde } N = \{S\}, T = \{a, b\},$$

$$P = \{1: S \rightarrow aSb, 2: S \rightarrow \epsilon\}$$

$$S \xrightarrow{[2]} \epsilon$$

$$S \xrightarrow{[1]} aSb \Rightarrow ab$$

$$S \xrightarrow{[1]} aSb \Rightarrow aaSbb \xrightarrow{[1]} aabb$$

⋮

**$L = \{a^n b^n : n \geq 0\}$  je bezkontextový jazyk.**

**Definice:** Nechť  $G = (N, T, P, S)$  je BKG.

*Jazyk generovaný BKG  $G$ ,  $L(G)$ , je definován:*

$$L(G) = \{w : w \in T^*, S \xrightarrow{*} w\}$$

## Derivační krok BKG

Jedná se o změnění řetězce použitím pravidla na neterminál. Pokud  $uAv \Rightarrow uxv$  v  $G$ , můžeme říct, že  $G$  provádí derivační krok z  $uAv$  do  $uxv$ .

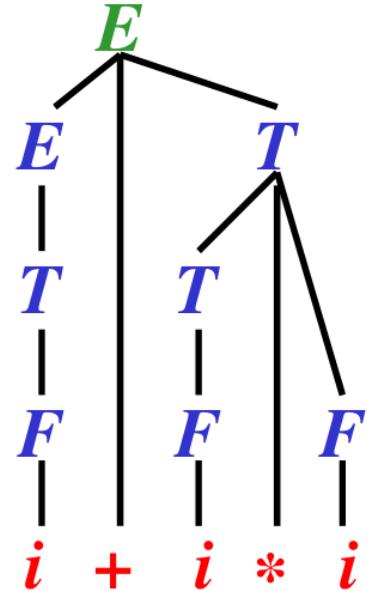
**Definice:** Nechť  $G = (N, T, P, S)$  je BKG. Nechť  $u, v \in (N \cup T)^*$  a  $p = A \rightarrow x \in P$ . Potom,  $uAv$  přímo derivuje  $uxv$  za použití  $p$  v  $G$ , zapsáno  $uAv \Rightarrow uxv [p]$  nebo zjednodušeně  $uAv \Rightarrow uxv$ .

$$G = (N, T, P, E), \text{ kde } N = \{E, F, T\}, T = \{i, +, *, (, )\},$$

$$P = \{ \begin{array}{lll} 1: E \rightarrow E + T, & 2: E \rightarrow T, & 3: T \rightarrow T * F, \\ 4: T \rightarrow F, & 5: F \rightarrow (E), & 6: F \rightarrow i \end{array} \}$$

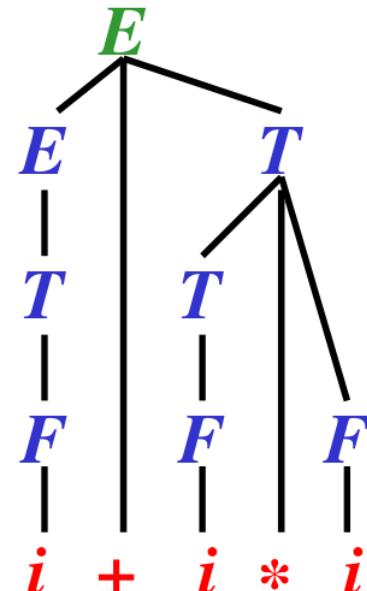
- **Nejlevější derivace:** Během nejlevějšího derivačního kroku je přepsán nejlevější neterminál.

$$\begin{aligned} E &\Rightarrow_{lm} \underline{E} + T & [1] \\ &\Rightarrow_{lm} \underline{T} + T & [2] \\ &\Rightarrow_{lm} \underline{F} + T & [4] \\ &\Rightarrow_{lm} i + \underline{T} & [6] \\ &\Rightarrow_{lm} i + \underline{T} * F & [3] \\ &\Rightarrow_{lm} i + \underline{F} * F & [4] \\ &\Rightarrow_{lm} i + i * \underline{F} & [6] \\ &\Rightarrow_{lm} i + i * i & [6] \end{aligned}$$



- **Nejpravější derivace:** Během nejpravějšího derivačního kroku je přepsán nejpravější neterminál.

$$\begin{aligned} E &\Rightarrow_{rm} E + \underline{T} & [1] \\ &\Rightarrow_{rm} E + T * \underline{F} & [3] \\ &\Rightarrow_{rm} E + T * \underline{i} & [6] \\ &\Rightarrow_{rm} E + \underline{F} * i & [4] \\ &\Rightarrow_{rm} E + i * i & [6] \\ &\Rightarrow_{rm} \underline{T} + i * i & [2] \\ &\Rightarrow_{rm} \underline{F} + i * i & [4] \\ &\Rightarrow_{rm} i + i * i & [6] \end{aligned}$$



$A \rightarrow x$  znamená, že  $A$  má být přepsáno na  $x$

## Zásobníkový automat

Konečný automat rozšířený o **zásobník**. Pro každou **BKG** existuje zásobníkový automat co ji přijímá. Zásobníkový automat (ZA) je sedmice:

$M = (Q, \Sigma, \Gamma, R, s, S, F)$ , kde:

- $Q$  je konečná množina stavů,
- $\Sigma$  je vstupní abeceda,
- $\Gamma$  je zásobníková abeceda,
- $R$  je konečná množina pravidel tvaru  $Apa \rightarrow wq$ , kde  $A \in \Gamma$ ;  $p, q \in Q$ ;  $a \in \Sigma \cup \{\epsilon\}$ ,  $w \in \Gamma^*$ , ( $R$  je konečná relace z  $\Gamma \times Q \times (\Sigma \cup \{\epsilon\})$  do  $\Gamma^* \times Q$ )
- $s \in Q$  je počáteční stav,
- $S \in \Gamma$  je počáteční symbol na zásobníku,
- $F \subseteq Q$  je množina koncových stavů.

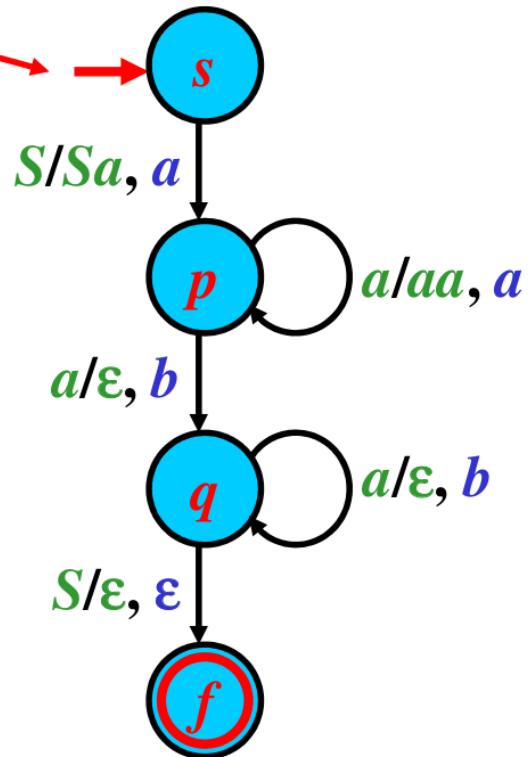
Interpretace pravidel:  $Apt \rightarrow wq \in R$  znamená, že pokud je aktuální stav  $p$ , aktuální symbol na vstupní pásce  $t$  a symbol na vrcholu zásobníku  $A$ , potom zásobníkový automat může přečíst  $t$  a na zásobníku nahradit  $A$  za  $w$  a přejít ze stavu  $p$  do  $q$ .



$$M = (Q, \Sigma, \Gamma, R, s, S, F)$$

kde:

- $Q = \{s, p, q, f\}$ ;
- $\Sigma = \{a, b\}$ ;
- $\Gamma = \{a, S\}$ ;
- $R = \{Ssa \rightarrow Sap, apa \rightarrow aap, apb \rightarrow q, aqb \rightarrow q, Sq \rightarrow f\}$
- $F = \{f\}$



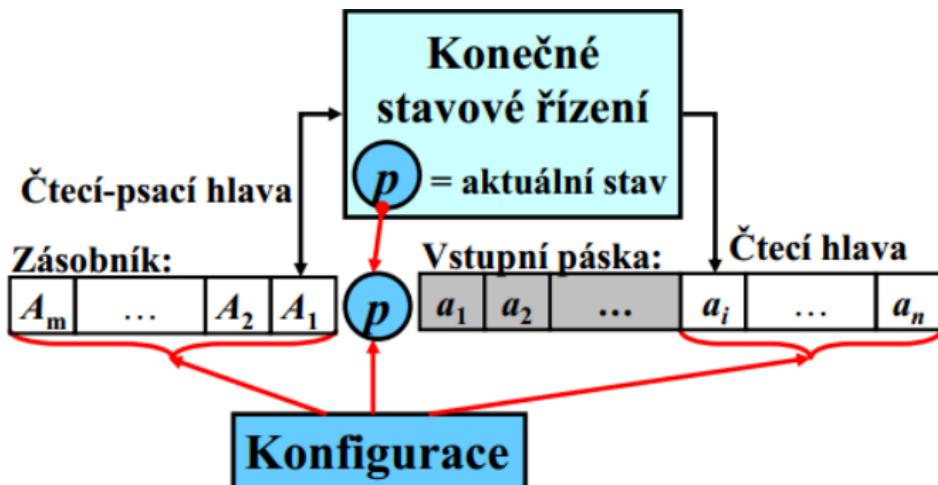
## Typy přijímaných jazyků ZA

- **Přechodem do koncového stavu:** ZA M přijímá jazyk L, pokud se čtením všech řetězců jazyka L dostane do koncového stavu.
- **Vyprázdněním zásobníků:** ZA M přijímá jazyk L, pokud se čtením všech řetězců jazyka L vyprázdní jeho zásobník.
- **Přechodem do koncového stavu a vyprázdněním zásobníku:** přijímá jazyk L, pokud se čtením všech řetězců jazyka L dostane do koncového stavu a současně se vyprázdní jeho zásobník.

Tyto tři typy ZA jsou si **ekvivalentní** a existují algoritmy pro **převody** mezi nimi.

## Konfigurace ZA

Konfigurace ZA M je řetězec  $\chi \in \Gamma^* Q \Sigma^*$ . Jedná se o **aktuální stav zásobníku**, **aktuálního stavu** a **vstupní pásky**, jejíž část **nebyla** ještě přečtena.

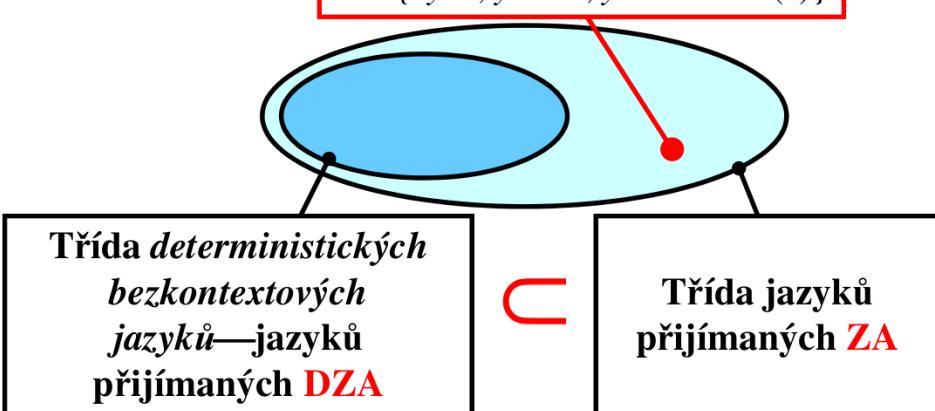


## Deterministický zásobníkový automat (DZA)

Může provést z každé konfigurace **maximálně jeden přechod**. M je deterministický ZA, pokud pro každé pravidlo tvaru  $Apa \rightarrow wq \in R$  platí, že množina  $R - \{Apa \rightarrow wq\}$  (množinový rozdíl) neobsahuje **žádné pravidlo** s levou stranou  $Apa$  nebo  $Ap$ , jinak řešeno **levá strana** pravidel je v množině  $R$  vždy **pouze jednou** - je unikátní. DZA je podmnožinou KA, DZA nemusí přijímat některé jazyky, která KA přijímá → KA je silnější než DKA.

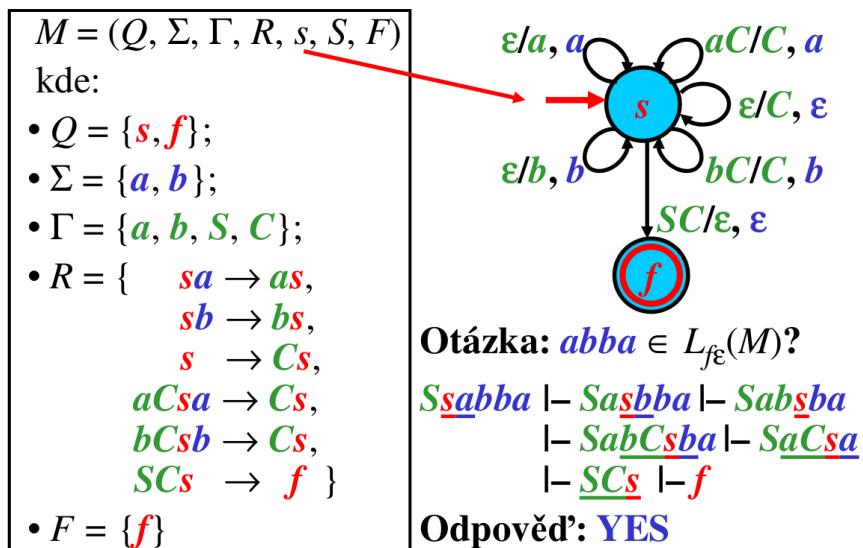
### Ilustrace:

$$L = \{xy: x, y \in \Sigma^*, y = reversal(x)\}$$

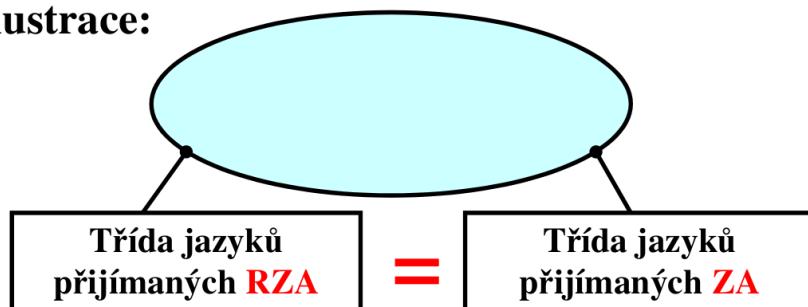


## Rozšířený zásobníkový automat (RZA)

Z vrcholu zásobníku v **RZA** lze číst celý řetězec (v ZA to byl pouze jeden symbol). Pravidla jsou tak definována takto: **R** je konečná množina pravidel tvaru:  $vpt \rightarrow wq$ , kde  $v, w \in \Gamma^*$ ;  $p, q \in Q$ ;  $t \in \Sigma \cup \{\epsilon\}$ . RZA je ekvivalentní s ZA (jsou stejně silné). Mohou existovat **RZA**, které přijímají jazyky přechodem do koncového stavu, vyprázdněním zásobníku nebo vyprázdněním zásobníku a současně přechodem do koncového stavu. Opět jsou tyto typy RZA ekvivalentní. Následuje příklad RZA.



**Illustrace:**



## RZA a ZA jako modely pro synt. analýzu

**RZA** nebo **ZA** mohou simulovat konstrukci derivačního stromu pro **BKG**. Používají se k tomu dva přístupy.

**1) Shora dolů**



**Z S směrem ke vstupnímu řetězci**

**2) Zdola nahoru**



**Ze vstupního řetězce směrem k S**

## Převod z BKG na RZA (pro SA zdola nahoru)

1. obrázek je pro SA zdola nahoru, 2. pro SA shora dolu.

$$Q := \{s, f\};$$

$$\Sigma := T;$$

$$\Gamma := N \cup T \cup \{\#\};$$

Konstrukce množiny  $R$ :

- **for each**  $a \in \Sigma$ : přidej  $sa \rightarrow as$  do  $R$ ;
- **for each**  $A \rightarrow x \in P$ : přidej  $xs \rightarrow As$  do  $R$ ;
- přidej  $\#Ss \rightarrow f$  do  $R$ ;

$$F := \{f\};$$

$$Q := \{s\};$$

$$\Sigma := T;$$

$$\Gamma := N \cup T;$$

Konstrukce množiny  $R$ :

- **for each**  $a \in \Sigma$ : přidej  $asa \rightarrow s$  do  $R$ ;
- **for each**  $A \rightarrow x \in P$ : přidej  $As \rightarrow ys$  do  $R$ , kde  $y = \text{reversal}(x)$ ;

$$F := \emptyset;$$

Např. (pořadí obrázků zůstává):

$$G = (N, T, P, S), \text{ kde:}$$

$$N = \{S\}, T = \{(, )\}, P = \{S \rightarrow (S), S \rightarrow ()\}$$

**Máme nalézt:** RZA  $M$ , pro který platí:  $L(G) = L(M)_f$

$$M = (Q, \Sigma, \Gamma, R, s, \#, F), \text{ kde:}$$

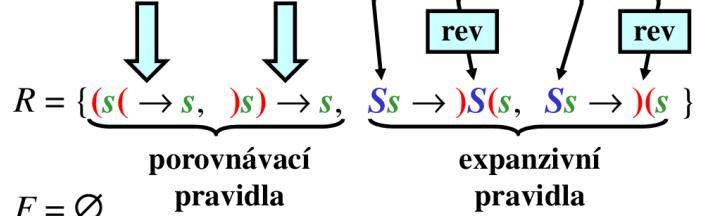
$$Q = \{s, f\}, \Sigma = T = \{(, )\}, \Gamma = \{(, ), S, \#\}, F = \{f\}$$

$$R = \{s( \rightarrow (s, s) \rightarrow )s, (S)s \rightarrow Ss, ()s \rightarrow Ss, \#Ss \rightarrow f\}$$

$$M = (Q, \Sigma, \Gamma, R, s, S, F) \text{ kde:}$$

$$Q = \{s\}; \quad \Sigma = T = \{(, )\}; \quad \Gamma = N \cup T = \{S, (, )\}$$

$$P = \{(")" \in T, ")" \in T, S \rightarrow (S) \in P, S \rightarrow () \in P\}$$



$$F = \emptyset$$

Modely pro syntaktickou analýzu zdola nahoru

**RZA M = (Q, Σ, Γ, R, s, #, F)** (# je počáteční symbol na zásobníku, s je počáteční stav)

1. **M** obsahuje **shiftovací pravidla**, která **přesouvají** vstupní symboly **na zásobník**. (tvorba: Pro každé  $t \in \Sigma$ : přidej  $st \rightarrow ts$  do  $R$ .)
2. **M** obsahuje **redukční pravidla**, která simulují **aplikaci gramatických pravidel** pozpátku. (tvorba: Pro každé  $A \rightarrow x \in P \vee G$ : přidej  $xs \rightarrow As$  do  $R$ )
3. **M** také obsahuje **speciální pravidlo**  $\#Ss \rightarrow f$ , pomocí kterého provede **M** přechod do **koncového stavu**.

Modely pro syntaktickou analýzu shora dolu

**ZA M = (Q, Σ, Γ, R, s, S, F)** (F = prázdná množina)

1. **M** obsahuje **porovnávací pravidla**, která **porovnají** symbol z vrcholu **zásobníku** a aktuální symbol ze **vstupní pásky**. (tvorba: pro každé  $a \in \Sigma$ : přidej  $asa \rightarrow s$  do  $R$ )
2. **M** obsahuje **expanzivní pravidla**, která **simulují gramatická pravidla**. (tvorba: pro každé  $A \rightarrow t_1\dots t_n \in P$  v  $G$ , přidej  $As \rightarrow t_n\dots t_1s$  do  $R$ ; = **reversal(t<sub>1</sub>\dots t<sub>n</sub>)**)

**odkazy:**

- Definice z IFJ: [IFJ Teorie](#)

# 23. Struktura překladače a charakteristika fází překladu (lexikální analýza, deterministická syntaktická analýza a generování kódu).

## Překladač

Překladač čte **zdrojový program** překládá jej na **cílový program**.

- **Vstup** - Text zdrojového kódu ve zdrojovém jazyce (obvykle vyšší programovací jazyk - **C++, Go, C#, Java**).
- **Výstup** - Cílový program napsaný v cílovém jazyce (obvykle binární kód nebo bytecode), který je **funkčně ekvivalentní** se zdrojovým programem.

## Fáze překladu

Jednotlivé části mohou často být spojené. Někdy se provádí více průchodů (definice funkce může být až za jejím voláním, optimalizace, ...)

1. **Lexikální analýza**: Lexikální analyzátor - **scanner**,
2. **Syntaktická analýza**: Syntaktický analyzátor - **parser**,
3. **Sémantická analýza**: Sémantický analyzátor,
4. **Generování vnitřního kódu**: Generátor vnitřního kódu,
5. **Optimalizace**: Optimalizátor,
6. **Generování cílového kódu**: Generátor cílového kódu.

## Lexikální analýza



- **Vstup** - Text ve zdrojovém jazyce.
- **Výstup** - Řetězec tokenů.

Provádí **rozpoznávání** a **klasifikaci lexémů**, reprezentuje lexémy pomocí **tokenů**. Odstraňuje prázdné znaky a komentáře a komunikuje s tabulkou symbolů. Zdrojový program je rozdělen na **lexémy** = logicky oddělené lexikální jednotky (**identifikátory**, **čísla**, **klíčová slova**, **operátory**, ...). Každý lexém je reprezentován **tokenem**, který může mít **atributy** (u čísla je to jeho hodnota, u proměnné její název, u řetězce řetězec, ...). Jednotlivé lexémy jazyků jsou navrženy tak, aby je specifikovaly

regulární výrazy a bylo je možné přijímat deterministickými konečnými automaty.

Na DKA je tak založena **implementace** lexikální analýzy - **scanneru**.

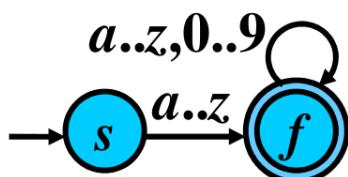
Implementujeme pomocí **switch** statement ve **while** cyklu (může být vnořené).

Klíčová slova a identifikátory se rozlišují podle **tabulky klíčových slov**. Informace (jméno, typ, konstantní hodnota, počet a typy parametrů v případě funkce, ...) o identifikátorech se uchovávají v **tabulce symbolů**, která má **zásobníkovou strukturu**, což umožňuje např. definovat globální a lokální proměnné.

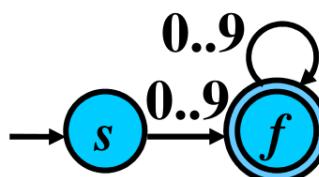
## 1) Rozpoznávání jednotlivých lexémů pomocí DKA:

### Příklad:

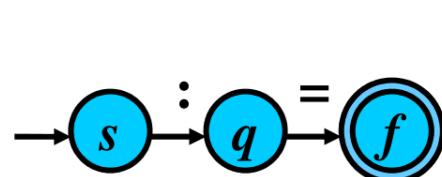
#### Identifikátor:



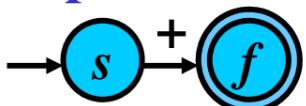
#### Celé číslo:



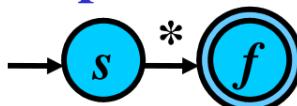
#### Přiřazení:



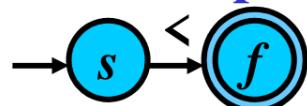
#### Operátor +:



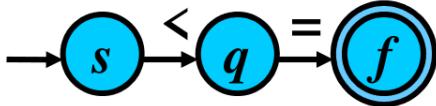
#### Operátor \*:



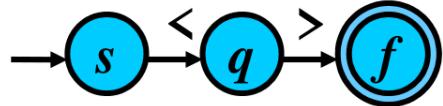
#### Relační op. <:



#### Relační op. <=:



#### Relační op. <>:

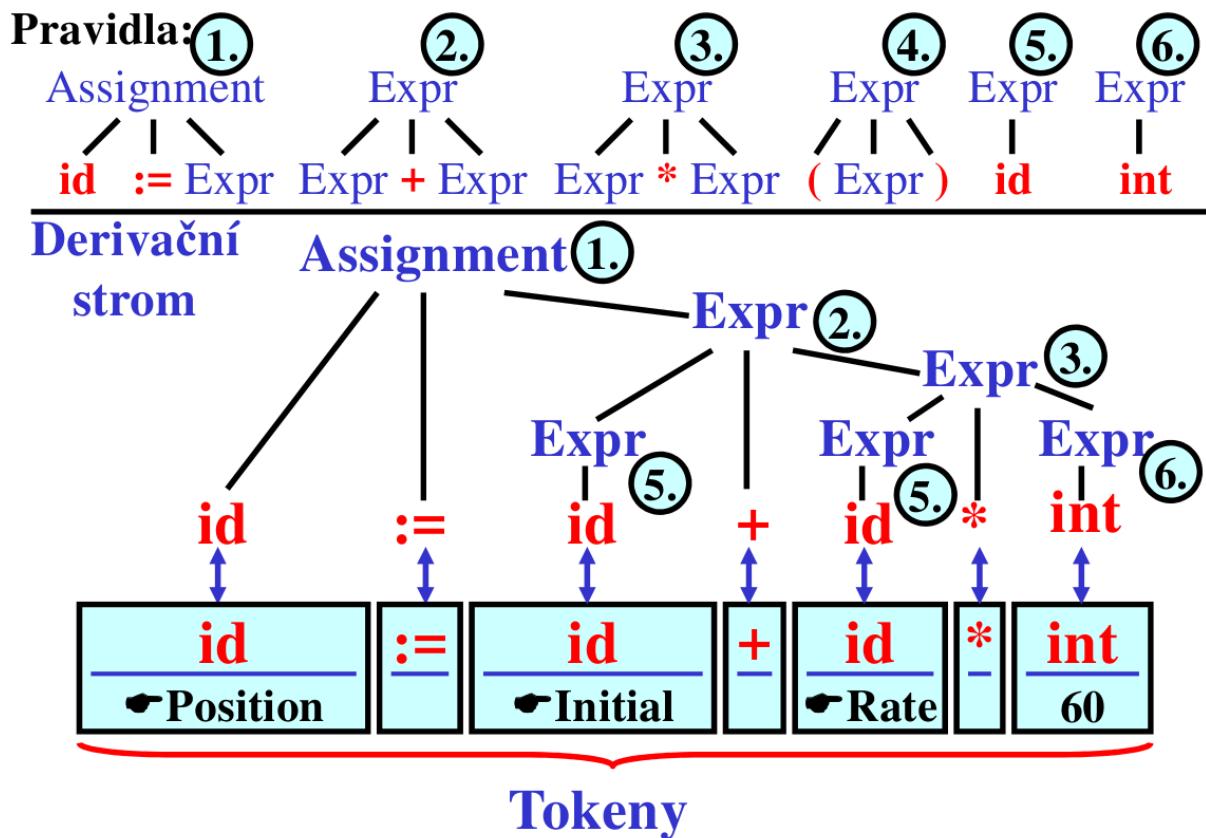


### Syntaktická analýza

- **Vstup** - Řetězec tokenů.
- **Výstup** - Simulace konstrukce derivačního stromu.

Syntaktický analyzátor (**parser**) kontroluje, zda **řetězec tokenů** reprezentuje **syntakticky správně** napsaný program. Program je **správný**, pokud je k danému řetězci tokenů nalezen **derivační strom**, jinak správný není. Simulace konstrukce **derivačního stromu** je založena na **gramatických pravidlech**. Používají se dva přístupy, a to **shora dolů** a **zdola nahoru**. Pro syntaktickou analýzu se používají **deterministické zásobníkové automaty** (terminály jsou **tokeny**), respektive podmnožiny **bezkontextových gramatik** - **LL gramatiky** a **LR gramatiky** (BKG jsou silnější než LL a LR gramatiky)

- **první L**: čtení zleva doprava,
- **druhé L**: levá derivace (leftmost derivation) - nahrazuje se **nejlevější neterminál**.
- **druhé R**: pravá derivace (rightmost derivation) - nahrazuje se **nejpravější neterminál**.



### Syntaktická analýza shora dolů

Syntaktická analýza shora dolů je založená na **LL gramatikách** a **LL tabulkách**. LL gramatika **bez  $\epsilon$**  pravidel je BKG  $G = (N, T, P, S)$ , pro kterou navíc pro každé  $t \in T, A \in N$  platí, že  $t \in T, A \in N$  a existuje maximálně jedno pravidlo  $A \rightarrow X_1X_2...X_n \in P$  takové, že:  $t \in \text{First}(X_1X_2...X_n)$ .

- **First(x)** je množina všech terminálů, kterými může začínat **řetězec derivovatelný** z  $x$ ,  $x \in (N \cup T)^*$ .

**Konstrukce LL-tabulky**

$\alpha$	...	$a$	...
...			
$A$	$\alpha(A, a)$		
...			

$\alpha(A, a) = A \rightarrow X_1X_2...X_n \in P$   
pokud  $a \in \text{First}(X_1)$ ; jinak  $\alpha(A, a)$  je prázdné  $\Rightarrow$  CHYBA

**Vytvořme: LL tabulku**

	id	int	$:$ =	...
$\langle \text{prog} \rangle$				
$\langle \text{st-list} \rangle$	2 $\leftarrow id \in \text{First}(\langle \text{stat} \rangle)$			
$\langle \text{stat} \rangle$	6 $\leftarrow id \in \text{First(id)}$			
$\langle \text{it-list} \rangle$				
$\langle \text{item} \rangle$	10 $\leftarrow id \in \text{First(id)}$			

Zbytek vytvoříme analogicky.

**Prav. r:  $A \rightarrow X_1X_2...X_n$**

	$\text{First}(X_1)$
1: $\langle \text{prog} \rangle \rightarrow \text{begin} ...$	{begin}
2: $\langle \text{st-list} \rangle \rightarrow \langle \text{stat} \rangle ...$	{id, write, read}
3: $\langle \text{st-list} \rangle \rightarrow \text{end}$	{end}
4: $\langle \text{stat} \rangle \rightarrow \text{read} ...$	{read}
5: $\langle \text{stat} \rangle \rightarrow \text{write} ...$	{write}
6: $\langle \text{stat} \rangle \rightarrow \text{id} ...$	{id}
7: $\langle \text{it-list} \rangle \rightarrow , ...$	{,}
8: $\langle \text{it-list} \rangle \rightarrow )$	{)}
9: $\langle \text{item} \rangle \rightarrow \text{int}$	{int}
10: $\langle \text{item} \rangle \rightarrow \text{id}$	{id}

LL-gramatiky s  **$\epsilon$ -pravidly** odstraní levé rekurze, ale vyžadují zavést další množiny **Empty**, **Follow** a **Predict**.

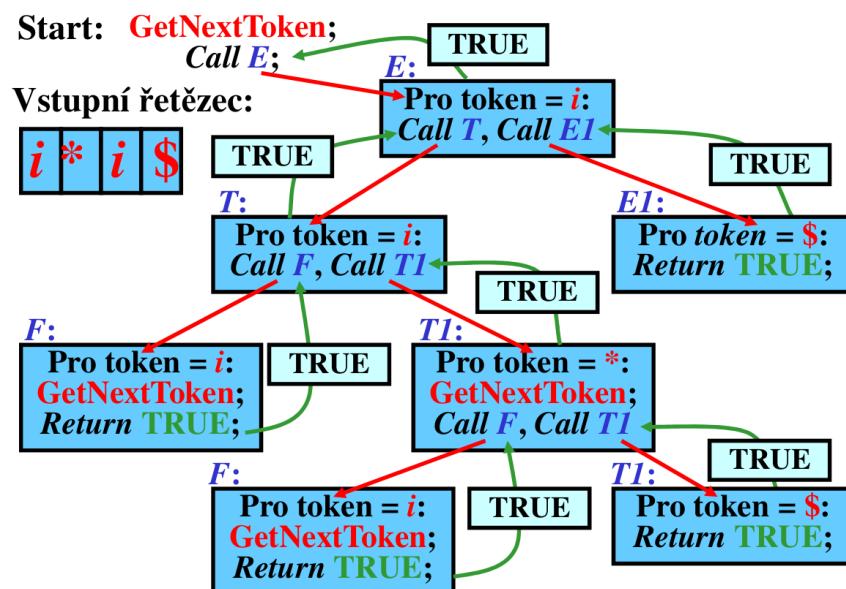
- **Empty:** **Empty(x)** je množina, která obsahuje jediný prvek  $\epsilon$  ( $\text{Empty}(x) = \{\epsilon\}$ ), pokud **x** derivuje  $\epsilon$ , jinak je prázdná ( $\text{Empty}(x) = \emptyset$ ).
- **Follow:** **Follow(A)** je množina všech **terminálů**, které se mohou vyskytovat vpravo od **A** (**A** je neterminál) ve větné formě.
- **Predict:** **Predict(A → x)** je množina všech **terminálů**, které mohou být aktuálně nejlevěji vygenerovány, pokud pro libovolnou větnou formu použijeme pravidlo **A → x**.

	<i>i</i>	+	*	(	)	\$
<i>E</i>	1			1		
<i>E'</i>		2			3	3
<i>T</i>	4			4		
<i>T'</i>		6	5		6	6
<i>F</i>	8			7		

$$\begin{array}{ll}
 1: E \rightarrow TE' & 5: T' \rightarrow *FT \\
 2: E' \rightarrow +TE' & 6: T' \rightarrow \epsilon \\
 3: E' \rightarrow \epsilon & 7: F \rightarrow (E) \\
 4: T \rightarrow FT' & 8: F \rightarrow i
 \end{array}$$

### Implementace LL analyzátoru

- **Rekurzivní sestup:** každý **neterminál** je reprezentován **procedurou/funkcí**, která řídí jeho syntaktickou analýzu a může tak **rekurzivně** volat procedury jiných neterminálů dle pravidel. Např. pro **pravidlo E → TF** volá procedura pro neterminál **E** nejdříve proceduru pro neterminál **T** a poté (pokud je úspěšná) volá proceduru pro neterminál **F** a když obě uspějí, vrací také úspěch.



- **Prediktivní syntaktická analýza:** Využívá syntaktický analyzátor se **zásobníkem**, který je řízený **LL tabulkou**. **Pravá strana** pravidel se ukládá na zásobník **obráceně - reversal** a provádí se vždy syntaktická analýza neterminálu na **vrcholu** zásobníku. Např. pro každý neterminál může být

definováno pole procedur/funkcí, které provádí pravidlo při načtení symbolu. Pokud pro daný symbol není, dochází k chybě.

## Syntaktická analýza zdola nahorů

Provádí Pravý rozbor = reverzovaná posloupnost pravidel, která je použita v **nejpravější derivaci pro vstupní řetězec**. Analyzátory pracující zdola nahoru dělíme na **precedenční syntaktické analyzátory** (nejslabší, ale jednoduché na implementaci) a **LR syntaktické analyzátory** (nejsilnější, ale složité pro implementaci, jsou silnější než LL analyzátory, protože jdou zdola - mohou existovat "nedeterministická pravidla").

## Precedenční syntaktický analyzátor

- Nesmí existovat více pravidel se **stejnou pravou stranou**.
- Gramatika nesmí obsahovat  **$\epsilon$ -pravidla**.

Pro syntaktickou analýzu využívá **precedenční tabulku**, která je dána **asociativitou** a **precedencí operátorů**. Používá se zejména k SA matematických výrazů (přiřazení do proměnných).

	+	*	(	)	i	\$	
+	>	<	<	>	<	>	
*	>	>	<	>	<	>	
(	<	<	<	=	<		
)	>	>	>	>	>		
i	>	>	>	>	>		
\$	<	<	<		<		

**Vstupní řetězec:  $i + i * i \$$**

Pushdown	Op	Vstup	Rule
\$	<	$i+i*i\$$	
$\$ < i$	>	$+i*i\$$	4: $E \rightarrow i$
$\$ E$	<	$+i*i\$$	
$\$ < E +$	<	$i*i\$$	
$\$ < E + < i$	>	$*i\$$	4: $E \rightarrow i$
$\$ < E + E$	<	$*i\$$	
$\$ < E + < E *$	<	$i\$$	
$\$ < E + < E * < i$	>	\$	4: $E \rightarrow i$
$\$ < E + < E * E$	>	\$	2: $E \rightarrow E * E$
$\$ < E + E$	>	\$	1: $E \rightarrow E + E$
$\$ E$	>	\$	

Úspěch  
 Pravý rozbor: 44421

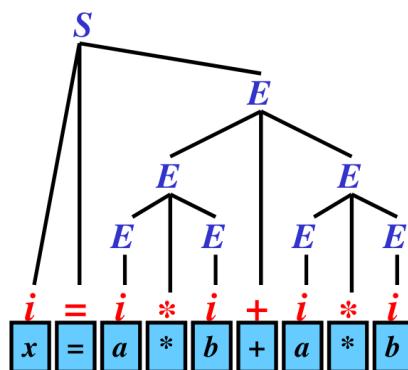
## Sémantická analýza

- **Vstup** - Simulace konstrukce derivačního stromu.
- **Výstup** - Abstraktní syntaktický strom.

Kontroluje sémantické aspekty programu, tj. provádí **kontrolu typů** a případně **implicitní konverze** (int na double), kontroluje **deklarace funkcí a proměnných**, kontrola **dělení 0**, kontrola **nepoužitých proměnných**, kontrola **pravdivosti logických výrazů** atd.

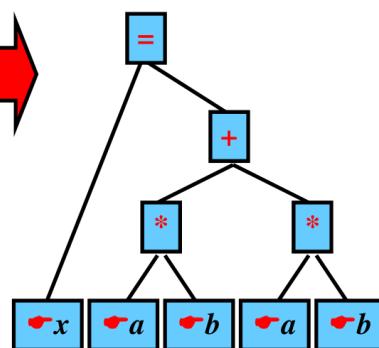
- derivační strom pro

$$x = a^*b + a^*b:$$



- ASS pro

$$x = a^*b + a^*b:$$



### Syntaxí řízený překlad

Syntaktický analyzátor (parser) řídí:

- Provádění sémantických akcí
- Generování abstraktního syntaktického stromu

### Generátor vnitřního kódu

- **Vstup** - Abstraktní syntaktický strom.
- **Výstup** - Vnitřní kód.

Generuje **vnitřní kód** - vnitřní reprezentaci programu (většinou **tříadresný**), ten je **jednotný**, lehce se **překládá** do cílového kódu a lehce se **optimalizuje**. Generování vnitřního kódu může být prováděno rekursivně na základě abstraktního syntaktického stromu. Syntaktický analyzátor, který pracuje metodou **zdola nahorů**, může generovat tříadresný kód **přímo** bez tvorby ASS. Přímé generování 3AK je založeno na **postfixové notaci**.

### Tříadresný kód

- Instrukce v tříadresném kódu (3AK) má tvar:

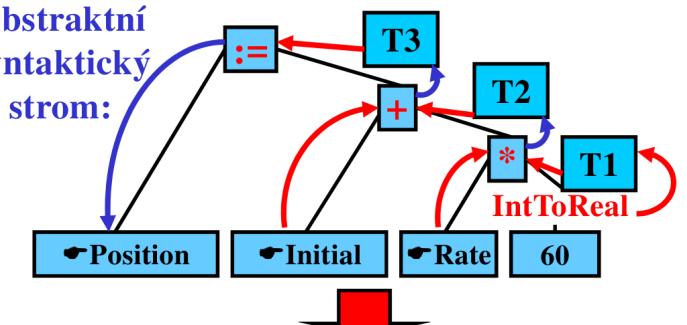
(**o**, **‐a**, **‐b**, **‐r**)

- **o** – operátor ( $+$ ,  $-$ ,  $*$ , ...)
- **a** – operand 1 ( $\neg a$  = adresa **a**)
- **b** – operand 2 ( $\neg b$  = adresa **b**)
- **r** – výsledek ( $\neg r$  = adresa **r**)

### Příklady:

$(:=, a, , c) \dots c := a$   
 $(+, a, b, c) \dots c := a + b$   
 $(not, a, , b) \dots b := not(a)$   
 $(goto, , , L1) \dots goto L1$   
 $(goto, a, , L1) \dots \text{if } a = \text{true} \text{ then goto } L1$   
 $(lab, L1, , ) \dots \underline{\text{label }} L1:$

Abstraktní  
syntaktický  
strom:



### Vnitřní kód:

```

T1 := IntToReal(60)
T2 := -Rate * T1
T3 := -Initial + T2
Position := T3
  
```

## Optimalizátor

- **Vstup** - Vnitřní kód.
- **Výstup** - Optimalizovaný vnitřní kód.

Snaží se o optimalizace vnitřního kódu. V rámci **globální** optimalizace odstraňuje mrtvý (nedosažitelný kód) a v rámci **lokální** optimalizace optimalizuje kód v bloku. Lokální optimalizace zahrnují např. **šíření konstanty, redukci logických výrazů, šíření kopírováním, rozbalení cyklu, výrazové invarianty v cyklu** (výpočet, který se provádí stejně při každém průchodu cyklem) atd. Je možné optimalizovat na **rychlosť** nebo na **velikost** výsledného programu. Překladač ale optimalizátor mít vůbec **NEMUSÍ** a výsledek programu bude funkčně **stejný**.

## Generátor cílového kódu

- **Vstup** - Optimalizovaný vnitřní kód (pouze vnitřní kód).
- **Výstup** - Cílový program.

Převádí vnitřní kód na cílový program, který je zapsán v cílovém jazyce. Obvykle je to **bytecode, assembler** nebo **strojový kód**.

## Slepé generování

Pro každou **3AK instrukci** existuje procedura, která generuje příslušný cílový kód.

- **výhody**: jednoduché pro implementaci,
- **nevýhody**: 3AK instrukce je **mimo kontext** ostatních a může tak docházet k **přebytečným** načítáním a ukládáním proměnných do/z registrů.

## Kontextové generování

Udržuje si přehled mezi jednotlivými 3AK instrukcemi. Pracuje na principu, jestliže je **hodnota** proměnné **v registru** a bude „**brzy**“ použita, **ponech** ji v registru.

Proměnné se dělí na **živé** (live - budou ještě v bloku použity) a **mrtvé** (dead - již jejich hodnoty nebudou použity, mohou být ale přepsány a poté **dále používány**). U živých proměnných se ještě uchovává **řádek následujícího použití**. Pro označení stavů proměnných (živá/mrtvá) se používá **zpětný algoritmus** 3AK instrukce se čtou od **konce bloku směrem k jeho začátku**.

# 24. Numerické metody (přímé a iterační metody pro řešení soustav lineárních rovnic, numerické řešení obyčejných diferenciálních rovnic).

## Matice

Slouží pro zjednodušený zápis soustavy rovnic. Využívají se pro řešení lineárních rovnic.

- **Čtvercová matice** - Stejný počet řádků a sloupců.

$$A = \begin{pmatrix} 0 & 1 & 5 \\ 8 & 5 & 23 \\ 47 & 154 & 2 \end{pmatrix}$$

- **Nulová matice** - Všechny prvky jsou 0

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

- **Jednotková matice** - Čtvercová matice, která má na hlavní diagonále (úhlopříčka z levého horního rohu) jedničky a všude jinde 0.

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- **Schodová matice** - Každý následující řádek má na začátku více nul než předchozí řádek.

$$A_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & \pi \\ 0 & 0 & 1 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad A_3 = \begin{pmatrix} 1 & 1 & 1 & 1 & 8 \\ 0 & 0 & 0 & 5 & 1 \\ 0 & 0 & 0 & 0 & 5 \end{pmatrix}$$

- **Transponovaná matice** - Zamění se řádky a sloupce (prvek co byl na 1;2 bude na 2;1)

$$\begin{pmatrix} 0 & 1 & 5 \\ 8 & 5 & 23 \\ 47 & 154 & 2 \end{pmatrix}^T = \begin{pmatrix} 0 & 8 & 47 \\ 1 & 5 & 154 \\ 5 & 23 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}^T = \begin{pmatrix} 3 & 6 \\ 4 & 7 \\ 5 & 8 \end{pmatrix}$$

- **Symetrická matice** - Taková, která je stejná i po transponování.

$$A = \begin{pmatrix} 9 & 3 & 4 \\ 3 & 7 & 0 \\ 4 & 0 & 2 \end{pmatrix}$$

- **Antisymetrická matice** - Podobné jako symetrická ale prvky na druhé straně jsou obrácené ( $A = -A^T$ ). Hlavní diagonála tedy musí být 0.

$$A = \begin{pmatrix} 0 & -3 & -4 \\ 3 & 0 & 5 \\ 4 & -5 & 0 \end{pmatrix}$$

- **Diagonální matice** - Všude jinde než na hlavní diagonále jsou 0 (na hlavní diagonále být můžou ale nemusí).

$$A_1 = \begin{pmatrix} 9 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 2 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

- **REGULÁRNÍ matice** - Její determinant je **nenulový**.
- **Diagonálně dominantní matice** - Pokud je absolutní hodnota každého prvku na diagonále větší nebo rovna součtu absolutních hodnot zbylých prvků ve sloupci nebo řádku (řádkově/sloupcově).

$$A = \begin{bmatrix} 3 & -2 & 1 \\ 1 & -3 & 2 \\ -1 & 2 & 4 \end{bmatrix}$$

Je diagonálně dominantní, protože

$$|a_{11}| \geq |a_{12}| + |a_{13}| \text{ od té doby } |+3| \geq |-2| + |+1|$$

$$|a_{22}| \geq |a_{21}| + |a_{23}| \text{ od té doby } |-3| \geq |+1| + |+2|$$

$$|a_{33}| \geq |a_{31}| + |a_{32}| \text{ od } . |+4| \geq |-1| + |+2|$$

- **Pozitivně definitní matice** - Čtvercová matice, u které platí

$$\mathbf{x} \neq 0 \implies \mathbf{x}^T \mathbf{M} \mathbf{x} > 0$$

## Determinant

Udává orientovaný obsah, respektive objem u 3. řádkové matice.

- **Křížové pravidlo** - Slouží pro výpočet determinantu **2x2 matice**.

$$\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad \det \mathbf{A} = ad - bc$$

- **Sarrusovo pravidlo** - Slouží pro výpočet determinantu **3x3 matice**.

$$\left| \begin{array}{ccc|cc} a_{11} & a_{12} & a_{13} & a_{11} & a_{12} \\ a_{21} & a_{22} & a_{23} & a_{21} & a_{22} \\ a_{31} & a_{32} & a_{33} & a_{31} & a_{32} \end{array} \right|$$

$$\det \mathbf{A} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33}$$

- **Determinant NxN matice** - Rozloží se na menší matice. Obsahuje-li matice **nulový řádek**, je její **determinant nulový**. Obsahuje-li matice **dva stejné řádky**, je její **determinant nulový**. Vznikla-li matice B z matice A **výměnou řádků**, pak  $|B| = -|A|$ . Vznikla-li matice B z matice A **vynásobením** jednoho jejího **řádku** konstantou  $c \in \mathbb{R}$ , platí  $|B| = c|A|$ .

$a_{11}$	$a_{12}$	$a_{13}$	$\dots$
$a_{21}$	$a_{22}$	$a_{23}$	$\dots$
$a_{31}$	$a_{32}$	$a_{33}$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\ddots$

Pozor na střídání znamének u kofaktorů:

$$A_{ij} = (-1)^{i+j} M_{ij}$$

Znaménka jsou rozmištěna jako na šachovnici.

Vypočtěte determinant matice A pomocí Laplaceova rozvoje.

$$A = \begin{pmatrix} 8 & 3 & 0 & 4 \\ 6 & 7 & 0 & 5 \\ -1 & 0 & 3 & 0 \\ 0 & 2 & 0 & 1 \end{pmatrix}$$

Nejvhodnější je použít rozvoj podle třetího sloupce:

$$\begin{aligned} |A| &= 0 \cdot (-1)^{1+3} \begin{vmatrix} 6 & 7 & 5 \\ -1 & 0 & 0 \\ 0 & 2 & 1 \end{vmatrix} + 0 \cdot (-1)^{2+3} \begin{vmatrix} 8 & 3 & 4 \\ -1 & 0 & 0 \\ 0 & 2 & 1 \end{vmatrix} + \\ &+ 3 \cdot (-1)^{3+3} \begin{vmatrix} 8 & 3 & 4 \\ 6 & 7 & 5 \\ 0 & 2 & 1 \end{vmatrix} + 0 \cdot (-1)^{4+3} \begin{vmatrix} 8 & 3 & 4 \\ -1 & 0 & 0 \\ 6 & 7 & 5 \end{vmatrix} = 3 \cdot 6 = 18 \end{aligned}$$

## Přímé metody pro řešení algebraických rovnic

Vedou k řešení soustavy po **konečném počtu kroků**. Toto řešení by bylo **přesné** kdybychom se nedopouštěli zaokrouhlovacích chyb.

### Cramerovo pravidlo

Vhodné pro velmi malé soustavy rovnic. Je-li matice soustavy **regulární ( $D \neq 0$ )**. Při výpočtu i-tého determinantu vždy nahrazujeme i-tý sloupec sloupcem s výsledky.

$$x_1 = \frac{D_1}{D}, \quad x_2 = \frac{D_2}{D}, \quad x_n = \frac{D_n}{D},$$

$$\begin{aligned} x_1 + 2x_2 - x_3 &= 1 \\ -2x_1 + x_2 - 3x_3 &= 2 \\ 2x_2 - x_3 &= -2 \end{aligned}$$

$$D = \begin{vmatrix} 1 & 2 & -1 \\ -2 & 1 & -3 \\ 0 & 2 & -1 \end{vmatrix} = -1 + 4 + 6 - 4 = 5$$

Úkolem je řešit soustavu rovnic

$$\begin{aligned} x + y &= 3 \\ x - 2y &= 1 \end{aligned}$$

$$D_1 = \begin{vmatrix} 1 & 2 & -1 \\ 2 & 1 & -3 \\ -2 & 2 & -1 \end{vmatrix} = -1 - 4 + 12 - 2 + 6 + 4 = 15$$

Determinant matice soustavy je

$$\det \mathbf{A} = \begin{vmatrix} 1 & 1 \\ 1 & -2 \end{vmatrix} = -3$$

Poněvadž je  $\det \mathbf{A} \neq 0$ , lze použít Cramerovo pravidlo.

Dále určíme

$$\det \mathbf{A}_1 = \begin{vmatrix} 3 & 1 \\ 1 & -2 \end{vmatrix} = -7$$

$$\det \mathbf{A}_2 = \begin{vmatrix} 1 & 3 \\ 1 & 1 \end{vmatrix} = -2$$

$$D_2 = \begin{vmatrix} 1 & 1 & -1 \\ -2 & 2 & -3 \\ 0 & -2 & -1 \end{vmatrix} = -2 - 4 - 6 - 2 = -14$$

$$D_3 = \begin{vmatrix} 1 & 2 & 1 \\ -2 & 1 & 2 \\ 0 & 2 & -2 \end{vmatrix} = -2 - 4 - 4 - 8 = -18$$

Řešení má tedy tvar

$$\begin{aligned} x &= \frac{\det \mathbf{A}_1}{\det \mathbf{A}} = \frac{-7}{-3} = \frac{7}{3} \\ y &= \frac{\det \mathbf{A}_2}{\det \mathbf{A}} = \frac{-2}{-3} = \frac{2}{3} \end{aligned}$$

## Gaussova eliminační metoda

Základem je úprava matice soustavy na **schodovitý tvar** - prohazování řádků, násobením a dělením nenulovým číslem a **přičítáním/odečítáním násobků jednotlivých řádků** k jiným. Pomineme-li zaokrouhlovací chyby, metoda poskytuje přesný výsledek, ale je poměrně náročná pro výpočet, konkrétně je třeba provést  $n^{3/3}$  (složitost v počtu provedených aritmetických operací je tedy **kubická**) aritmetických operací.

$$\begin{array}{c}
 \left( \begin{array}{ccc|c} 2 & 3 & 7 & 47 \\ 3 & 8 & 1 & 50 \\ 0 & 3 & 3 & 27 \end{array} \right) \xrightarrow{\text{x3}} \left( \begin{array}{ccc|c} 6 & 9 & 21 & 141 \\ 6 & 16 & 2 & 100 \\ 0 & 3 & 3 & 27 \end{array} \right) \xrightarrow{-} \\
 \sim \left( \begin{array}{ccc|c} 6 & 9 & 21 & 141 \\ 0 & 7 & -19 & -41 \\ 0 & 3 & 3 & 27 \end{array} \right) \xrightarrow{\text{x3}} \left( \begin{array}{ccc|c} 6 & 9 & 21 & 141 \\ 0 & 21 & -57 & -123 \\ 0 & 21 & 21 & 189 \end{array} \right) \xrightarrow{-} \\
 \sim \left( \begin{array}{ccc|c} 6 & 9 & 21 & 141 \\ 0 & 21 & -57 & -123 \\ 0 & 0 & 78 & 312 \end{array} \right) \\
 \hline
 \end{array}$$

$$\begin{array}{rcl}
 6x_1 + 9x_2 + 21x_3 = 141 \\
 21x_2 - 57x_3 = -123 \\
 78x_3 = 312
 \end{array} \rightarrow \boxed{\begin{array}{l} x_3 = 4 \\ x_2 = 5 \\ x_1 = 2 \end{array}}$$

## LU rozklad

### ► LU decomposition - An Example

Pomocí nalezeného LU rozkladu matice A najděte řešení soustavy

$$\begin{array}{rcl}
 2x_1 - 3x_2 + x_3 = 5 \\
 -3x_1 + 5x_2 + 2x_3 = -4 \\
 x_1 + 2x_2 - x_3 = 1
 \end{array}$$

Při Lower-Upper rozkladu se začne s jednotkovou maticí, která násobí původní matici, provádí se úpravy jako při GEM a zapisují se do jednotkové matice viz příklad (pozor na **znaménka**)

$$\begin{array}{l}
 \left( \begin{array}{ccc} 2 & -3 & 1 \\ -3 & 5 & 2 \\ 1 & 2 & -1 \end{array} \right) \sim \left( \begin{array}{ccc} 2 & -3 & 1 \\ 0 & 0,5 & 3,5 \\ 0 & 3,5 & -1,5 \end{array} \right) \xrightarrow{\text{II}+1,5\text{I}} \quad L = \left( \begin{array}{ccc} 1 & ? & ? \\ -1,5 & ? & ? \\ +0,5 & ? & ? \end{array} \right) \\
 \sim \left( \begin{array}{ccc} 2 & -3 & 1 \\ 0 & 0,5 & 3,5 \\ 0 & 0 & -26 \end{array} \right) \xrightarrow{\text{III}-7\text{II}} \quad L = \left( \begin{array}{ccc} 1 & 0 & ? \\ -1,5 & 1 & ? \\ +0,5 & 7 & ? \end{array} \right) \quad L = \left( \begin{array}{ccc} 1 & 0 & 0 \\ -1,5 & 1 & 0 \\ 0,5 & 7 & 1 \end{array} \right), \quad U = \left( \begin{array}{ccc} 2 & -3 & 1 \\ 0 & 0,5 & 3,5 \\ 0 & 0 & -26 \end{array} \right)
 \end{array}$$

Následně je nutné pomocí matice **L** upravit vektor pravé strany rovnice a na základě tohoto vektoru vypočítat řešení rovnice pomocí matice **U**.

$$\begin{array}{lcl} y_1 & = & 5 \\ -1,5y_1 + y_2 & = & -4 \\ 0,5y_1 + 7y_2 + y_3 & = & 1 \end{array} \Rightarrow \begin{array}{lcl} y_1 & = & 5 \\ y_2 & = & 3,5 \\ y_3 & = & -26 \end{array}$$

$$\begin{array}{lcl} 2x_1 - 3x_2 + x_3 & = & 5 \\ 0,5x_2 + 3,5x_3 & = & 3,5 \\ -26x_3 & = & -26 \end{array} \Rightarrow \begin{array}{lcl} x_1 & = & 2 \\ x_2 & = & 0 \\ x_3 & = & 1 \end{array}$$

## Iterační metody

Narozdíl od přímých **nevedou k přesnému výsledku** po konečném předem daném počtu kroků. Zvolíme počáteční approximaci řešení a tu v každém kroku **zlepšujeme**. K řešení se **přibližujeme postupně** a až je dostatečně přesný výpočet ukončíme (**výsledek** je tedy **přibližný**, k přesnému bychom se dostali s nekonečným počtem operací).

### Jacobiho metoda

#### ► The Jacobi Method

Konverguje pokud je matice soustavy rovnic **ostře řádkově diagonálně dominantní** (součet absolutních hodnot řádku je **větší**, než hodnota na diagonále) nebo **ostře sloupcově diagonálně dominantní** (součet absolutních hodnot sloupce je **větší**, než hodnota na diagonále). Pokud podmínky nejsou splněny, může konvergovat, ale nemusí.

$$15x_1 - x_2 + 2x_3 = 30$$

$$2x_1 - 10x_2 + x_3 = 23$$

$$x_1 + 3x_2 + 18x_3 = -22$$

- Matice soustavy je diagonálně dominantní, protože platí:  
 $|15| > |1| + |2|, |10| > |2| + |1|, |18| > |1| + |3|$

- Proto je konvergence metody vždy zaručena. Vypíšeme iterační vztahy:

$$x_1^{(r+1)} = \frac{1}{15}(30 + x_2^{(r)} - 2x_3^{(r)})$$

$$x_2^{(r+1)} = -\frac{1}{10}(23 - 2x_1^{(r)} - x_3^{(r)})$$

$$x_3^{(r+1)} = \frac{1}{18}(-22 - x_1^{(r)} - 3x_2^{(r)})$$

- Jako počáteční approximaci zvolíme  $x = (0,0,0)^T$ . Postupné získávané approximace řešení budeme zapisovat do tabulky:

r	$x_1^{(r)}$	$x_2^{(r)}$	$x_3^{(r)}$
0	0	0	0
1	2	-2,3	-1,2222
2	2,0096	-2,0222	-0,9500
3	1,9918	-1,9930	-0,9968
4	2,0000	-2,0013	-1,0007

## Gauss-Seidelova metoda

Metoda je podobná Jacobiho metodě. Liší se v tom, že v každém kroku používá již **novou hodnotu proměnné** (pokud je známa) a ne tu z minulé iterace. Konverguje pro **ostře řádkově nebo sloupcově dominantní matici** a navíc nebo pokud je matici **pozitivně definitní** (lze zjistit např., že všechny její subdeterminanty z levého horního rohu musí být **větší než 0** Checking if a Matrix is Positive Definite nebo lze zjistit **pivot testem** - převodem matice na horní trojúhelníkovou matici a porovnáním hodnot na diagonále s 0, pokud jsou **větší než 0**, je matici pozitivně definitní Positive Definite Matrices and Minima (pro větší nebo rovno 0 jsou matice semidefinitní)).

$$\begin{aligned}x_1^{(r+1)} &= \frac{a_{14} - a_{12}x_2^{(r)} - a_{13}x_3^{(r)}}{a_{11}} \\x_2^{(r+1)} &= \frac{a_{24} - a_{21}x_1^{(r+1)} - a_{23}x_3^{(r)}}{a_{22}} \\x_3^{(r+1)} &= \frac{a_{34} - a_{31}x_1^{(r+1)} - a_{32}x_2^{(r+1)}}{a_{33}}\end{aligned}$$

$$\begin{aligned}x_1^{k+1} &= \frac{1}{2} \left( x_2^k + \frac{1}{3} \right) \\x_2^{k+1} &= \frac{1}{2} \left( x_1^{k+1} + x_3^k + 1 \right) \\x_3^{k+1} &= \frac{1}{2} \left( x_2^{k+1} - \frac{1}{3} \right)\end{aligned}$$

Průběh výpočtu je zaznamenán v následující tabulce.

$k$	$x_1^k$	$x_2^k$	$x_3^k$
0	0	0	0
1	0.1667	0.5833	0.1250
2	0.4583	0.7917	0.2290
:	:	:	:
14	0.6666	1.0000	0.3333
15	0.6666	1.0000	0.3333

## Numerické řešení obyčejných diferenciálních rovnic

Rovnice, kde jako proměnné vystupují **derivace funkcí**. U některých lze nalézt přesné analytické řešení, ale většinou **nelze nebo je to velmi obtížně**. Naštěstí lze řešení diferenciální rovnice velmi **dobře approximovat** za použitím některé **numerické metody**, které jsou založené na **iteračním řešení** těchto rovnic.

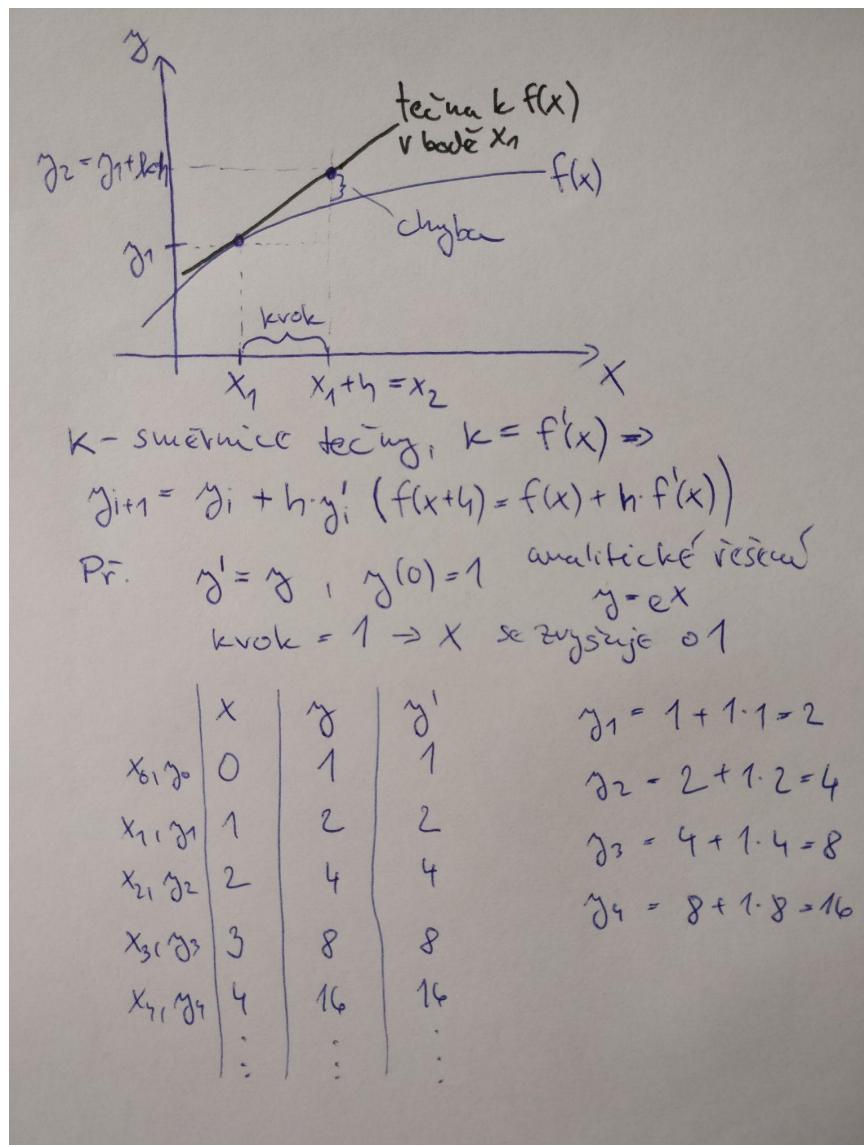
Numerické metody dělíme na:

- **jednokrokové**: vychází pouze z **aktuálního stavu** (aktuální ohodnocení proměnných), např. **Eulerova metoda**, metody **Runge-Kutta (RK2, RK4, RK8)**.
- **vícekrokové**: využívají historii stavů (ohodnocení proměnných), používají **hodnoty zapamatované z předchozích kroků**. Mohou být rychlejší než jednokrokové, ale obvykle mají **problém se startem** (pro prvních  $n$  iterací se použije **jednokroková metoda** → nevhodné pro nespojitě funkce). Jedná se např. o **metodu Adams-Bashforth**. (existují i samostartující metody)
- **prediktor-korektor**: Nejprve se vypočítá odhad nového  $y_{n+1}$ . V tomto bodě je vypočtena derivace  $f_{n+1}$ , která je následně použita pro výpočet přesnější aproximace  $y_{n+1}$ .
- **explicitní**: výsledek v každé iteraci získáme **dosazením do vzorce**.
- **implicitní**: vyžaduje řešení **algebraických rovnic** v každé iteraci.

### Eulerova metoda

Nejjednodušší jednokroková metoda. Založená na principu **směrnice tečny** v bodě, která je daná **1. derivací** funkce v tomto bodě. Směrnice udává, o kolik se **zvětší/zmenší** hodnota  $y$  při změně  $x$  (definuje tak přímku).

$$y' = f(x, y), \quad y(x_0) = y_0 \quad y_{i+1} = y_i + h \cdot f(x_i, y_i), \quad i = 0, 1, 2, \dots$$



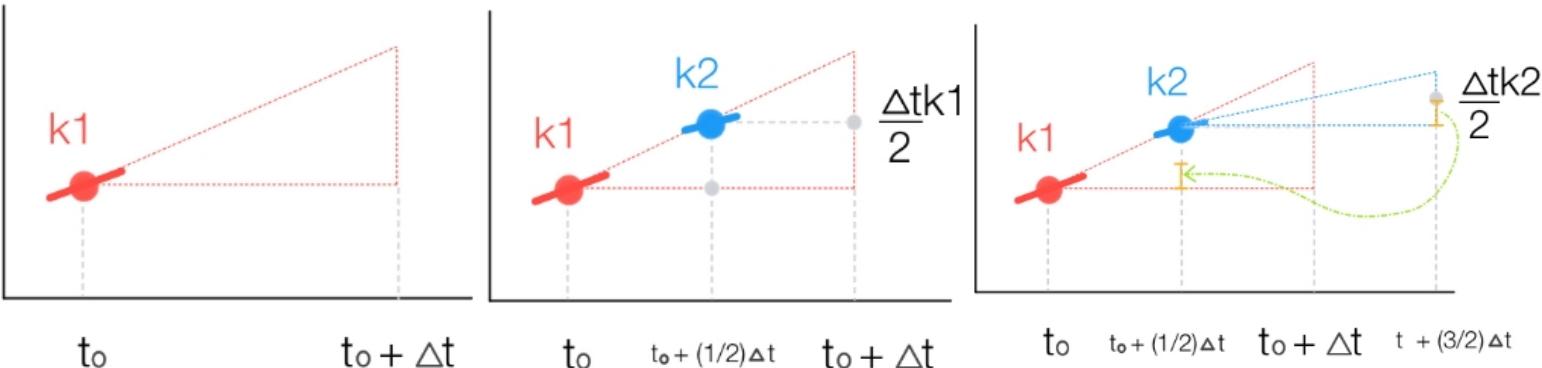
Př. 2:  $y' = x - y$ ;  $y(0) = 1$ ;  $h=0,2$  na  $<0; 0,6>$

$$\begin{aligned}y_1 &= y_0 + h \cdot f(x_0, y_0) = y_0 + h \cdot (x_0 - y_0) = 1 + 0,2 \cdot (0 - 1) = \underline{0,8} \\y_2 &= y_1 + h \cdot f(x_1, y_1) = y_1 + h \cdot (x_1 - y_1) = 0,8 + 0,2 \cdot (0,2 - 0,8) = \underline{0,68} \\y_3 &= y_2 + h \cdot f(x_2, y_2) = y_2 + h \cdot (x_2 - y_2) = 0,68 + 0,2 \cdot (0,4 - 0,68) = \underline{0,624}\end{aligned}$$

## Runge-Kutta

Jednokroková metoda. Vylepšuje eulerovu metodu. Existují RK metody různých řádů (RK1 - eulerova metoda, RK2, RK4, RK8). Koeficienty u těchto metod jsou vypočteny tak, aby metoda řádu **b** odpovídala **Taylorovu polynomu** funkce  $y(t)$  stejného řádu. Pro výpočty se nejčastěji používá metoda **Runge-Kutta 4. řádu**. Pro RK metodu **k-tého** řádu se počítá **k** částečných odhadů (druhý odhad je závislý na prvním, třetí na druhém, ...) a výsledný odhad se poté vypočte jako krok vynásobený jejich váženým průměrem.

$$y_{n+1} = y_n + h \left( \frac{1}{2}k_1 + \frac{1}{2}k_2 \right) \quad \begin{aligned}k_1 &= f(t_n, y_n), \\k_2 &= f(t_n + h, y_n + hk_1).\end{aligned}$$

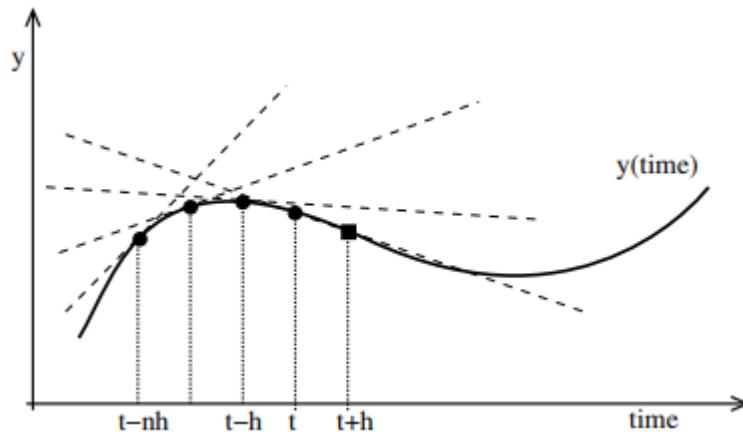


## Adams-Bashforth

**Vícekroková metoda**, pamatuje si výsledky předchozích kroků. Např. dvojkrokový a čtyřkrokový:

$$y_{n+2} = y_{n+1} + \frac{3}{2}hf(t_{n+1}, y_{n+1}) - \frac{1}{2}hf(t_n, y_n).$$

$$y_{n+1} = y_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3})$$



### Adams-Basforth-Moulton

Metoda typu **prediktor-korektor**, zpřesňují výsledek použitím prvotního odhadu pro výpočet výsledného odhadu.

$$y_{n+1} = y_n + \frac{h}{24}(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2})$$

### Tuhé systémy

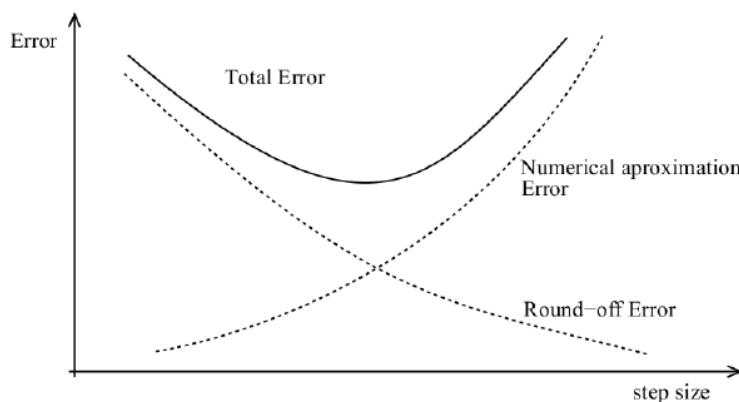
Problematické pro řešení pomocí běžných numerických metod (RK). Vyskytují se zde velmi rozdílné časové konstanty. Zkrácení kroku často nelze (zaokrouhlovací chyby, malá efektivita). Je nutné použít speciální metody.

$$y'' + 101y' + 100y = 0$$

### Chyby numerických metod

Při každé aproximaci se musí počítat s chybou výsledku.

- **Lokální chyba** - Vzniká v každém kroku - **zaokrouhlovací** nebo **aproximační**.
- **Akumulované chyby** - Sesbírané chyby po **celou dobu výpočtu**.



Hledání ideální délky kroku.

# 25. Teorie grafů. Pojem grafu, základní pojmy, isomorfismus grafů, souvislost. Grafové algoritmy pro hledání nejkratší cesty a minimální kostry.

Neformálně se graf skládá z vrcholů a hran, které tyto vrcholy spojují. Jedná se o speciální případy binárních relací. **Formálně** je graf (jednoduchy neorient.) uspořádaná dvojice  $\mathbf{G} = (\mathbf{V}, \mathbf{E})$

- $\mathbf{V}$  je množina vrcholů,
- $\mathbf{E}$  je množina hran – množina vybraných **douuprvkových podmnožin množiny vrcholů** (znázorňují spojení mezi vrcholy).

Hranu mezi vrcholy  $u$  a  $v$  píšeme jako  $\{u, v\}$ , nebo zkráceně  $uv$ .

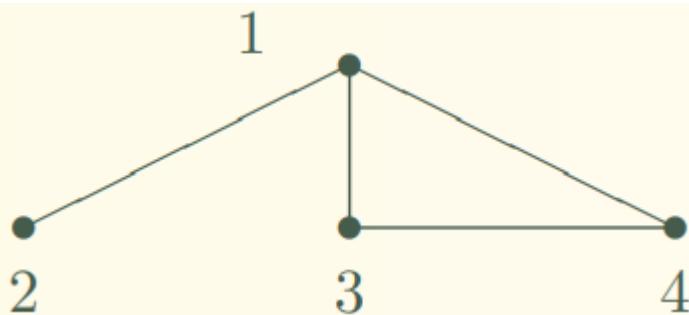
- **sousední vrcholy:** vrcholy spojené hranou.

Hrana  $uv$  vychází z vrcholů  $u$  a  $v$ .

Množiny grafu  $G$  odkazujeme:

- $\mathbf{V}(G)$ : množina **vrcholů**,
- $\mathbf{E}(G)$ : množina **hran**.

Grafy zadáváme **neformálně** graficky



nebo formálně výčtem vrcholů a hran

$$V = \{1, 2, 3, 4\}, \quad E = \left\{ \{1, 2\}, \{1, 3\}, \{1, 4\}, \{3, 4\} \right\}$$

Stupeň vrcholu

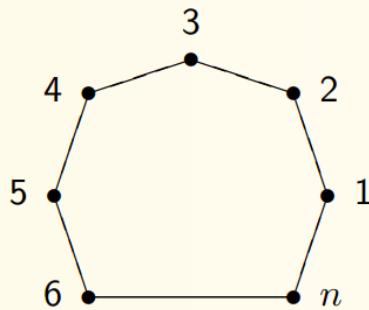
Stupeň vrcholu  $x$  v grafu  $\mathbf{G}$  rozumíme počet hran vycházejících (hrna vychází z obou konců současně) z vrcholu  $x$ . Stupeň vrcholu  $x$  v grafu  $G$  značíme  $d_G(x)$ . Graf je:

- **d-regulární:** všechny jeho vrcholy mají stejný stupeň  $d$ .

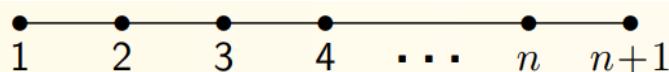
**Nejvyšší stupeň** grafu  $G$  značíme  $\Delta(G)$ . **Nejnižší stupeň** grafu  $G$  značíme  $\delta(G)$ .

## Typy grafů

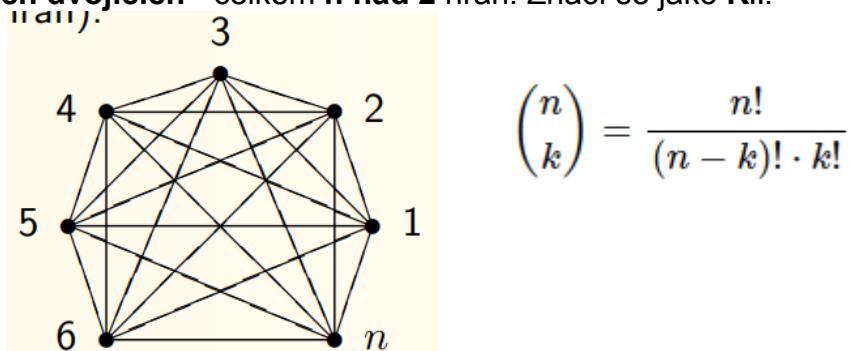
- **Kružnice:** Kružnice délky  $n$  má  $n \geq 3$  různých vrcholů spojených do jednoho cyklu  $n$  hranami. Značí se jako  $C_n$ .



- **Cesta:** Cesta délky  $n \geq 0$  má  $n+1$  různých vrcholů spojených za sebou  $n$  hranami. Značí se jako  $P_n$ . (zádné vrcholy ani hrany se neopakují)

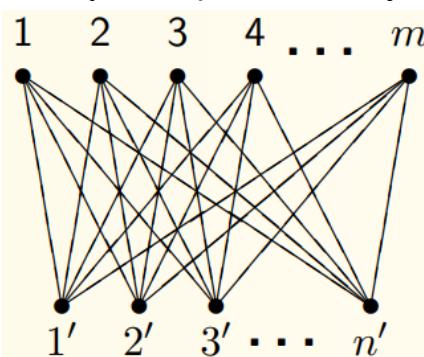


- **Úplný graf:** Úplný graf na  $n \geq 1$  vrcholech má  $n$  různých vrcholů spojených po všech dvojicích - celkem  $n$  nad 2 hran. Značí se jako  $K_n$ .

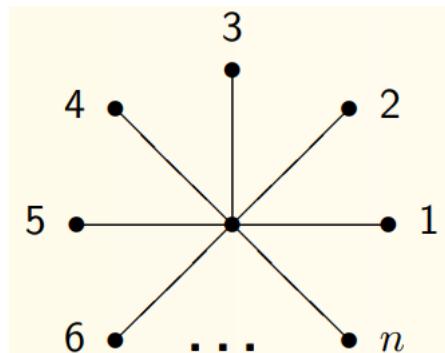


$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!}$$

- **Úplný bipartitní graf:** Úplný bipartitní graf na  $m \geq 1$  a  $n \geq 1$  vrcholech má  $m+n$  vrcholů ve dvou skupinách (partitách), přičemž hranami jsou spojeny všechny  $m \cdot n$  dvojice z různých skupin. Značí se jako  $K_{m,n}$ .



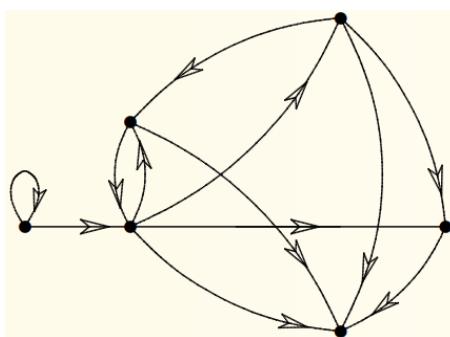
- **Hvězda:** Hvězda s  $n \geq 1$  rameny je zvláštní název pro úplný bipartitní graf. Značí se jako  $K_{1,n}$ .



- **Sled:** je v grafu **posloupnost vrcholů** taková, že mezi každými dvěma po sobě jdoucími vrcholy je hrana. **Hrany i vrcholy** se mohou na rozdíl od kružnice **opakovat**. Sled je procházka po hranách grafu z **u** do **v**, která může obsahovat cykly.
- **Tah:** je sled v grafu, ve kterém se **neopakují hrany** (vrcholy se opakovat mohou). **Uzavřený tah** je tah, který končí ve vrcholu, ve kterém **začal** (jinak je neuzavřený). Graf **G** lze nakreslit **jedním uzavřeným tahem právě**, když **G** je **souvislý** a všechny jeho vrcholy jsou **sudého stupně**.

## Orientované grafy

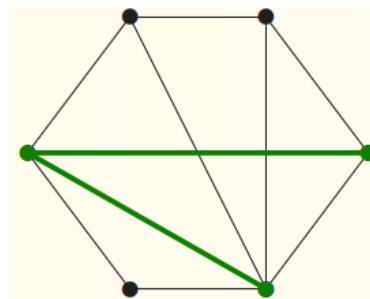
V orientovaných grafech má každá hrana jistý směr. Formálně mají orientované grafy množinu orientovaných hran **A**, která je dána  $\mathbf{A} \subseteq \mathbf{V(G)} \times \mathbf{V(G)}$ .



## Podgraf

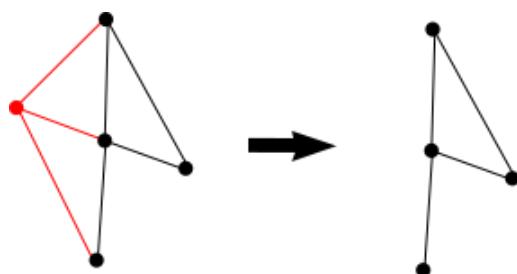
Podgrafem grafu **G** rozumíme libovolný graf **H** pro který platí:

- množina vrcholů grafu **H** je **podmnožinou** vrcholů grafu **G**:  $\mathbf{V(H)} \subseteq \mathbf{V(G)}$ .
- za hrany má **libovolnou podmnožinu** hran grafu **G**, které ale mají oba vrcholy ve **V(H)**.



## Indukovaný podgraf

Indukovaným podgrafem je podgraf  $\mathbf{H} \subseteq \mathbf{G}$  takový, který obsahuje **všechny hrany grafu G** mezi dvojicemi vrcholů z  $\mathbf{V(H)}$ . Jinak řečeno graf **H** vznikne smazáním části vrcholů grafu **G** a **pouze** hran, které vycházely z těchto vrcholů.



## Izomorfismus

Izomorfismus grafů **G** a **H** je **bijektivní** zobrazení  $f: V(G) \rightarrow V(H)$ , pro které každá dvojice  $u, v \in V(G)$  je spojená hranou v grafu **G** **právě, když** je dvojice  $f(u), f(v) \in V(H)$  spojená hranou v grafu **H**. Příklad pro grafy **G** a **G'**:

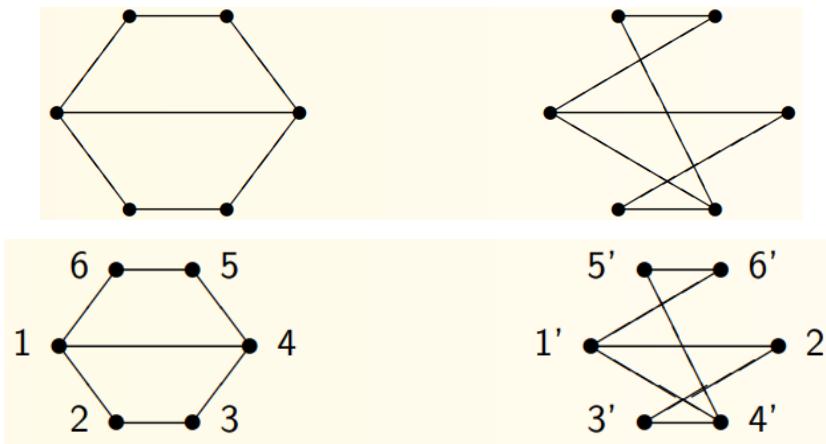
$$\exists F: V(G) \rightarrow V(G'): \{x, y\} \in E(G) \Leftrightarrow \{f(x), f(y)\} \in E(G')$$

Pro izomorfní grafy **G** a **H** platí (může to platit ale i pro neizomorfní grafy):

- **G** a **H** mají stejný počet vrcholů,
- **G** a **H** mají stejný počet hran,
- zobrazení **f** zobrazuje na sebe vrcholy **stejných stupňů**, tzn. mají stejné počty vrcholů o stejných stupních.

Postup hledání izomorfismu (pokud nějaký bod neplatí, grafy nejsou izomorfní):

1. ověříme **stejný počet vrcholů** u obou grafů,
2. ověříme **stejný počet hran** u obou grafů,
3. vytvoříme posloupnosti stupňů vrcholů pro každý graf (seřazeny od nejmenšího po největší) a ověříme, že jsou stejné.
4. zkoušíme všechny **přípustné možnosti** zobrazení izomorfismu.



## Typy podgrafů grafu G

- **Kružnice v G**, je podgraf  $H \subseteq G$ , který je **izomorfní** nějaké **kružnici**. (kružnici délky 3 říkáme trojúhelník).
- **Indukovaná kružnice v G**, je **indukovaný** podgraf  $H \subseteq G$ , který je **izomorfní** nějaké **kružnici**.
- **Cesta v G**, je podgraf  $H \subseteq G$ , který je **izomorfní** nějaké **cestě**.
- **Klika v G**, je podgraf  $H \subseteq G$ , který je **izomorfní** nějakému **úplnému grafu** (graf jehož všechny vrcholy jsou spojeny hranou se všemi zbylými).
- **Nezávislá množina X v G**, je podmnožina vrcholů  $X \subseteq V(G)$ , mezi kterými nevedou v **G** žádné hrany (přímo tyto vrcholy spojuje hrana, spojení přes více hran existovat může).

## Souvislost

Možnost se v grafu pohybovat z jakéhokoliv vrcholu do jakéhokoliv jiného vrcholu podél jeho hran. To znamená, že pro každé dva vrcholy  $u, v \in V(G)$  existuje sled z vrcholu  $u$  do vrcholu  $v$ . U orientovaných grafů rozlišujeme:

- **slabá souvislost** — graf je slabě souvislý, pokud jeho symetrizace (odstranění směru hran) je souvislý graf.
- **silná souvislost** — graf je silně souvislý, pokud pro každé dva vrcholy  $u, v$  existuje cesta z  $u$  do  $v$  i z  $v$  do  $u$ .

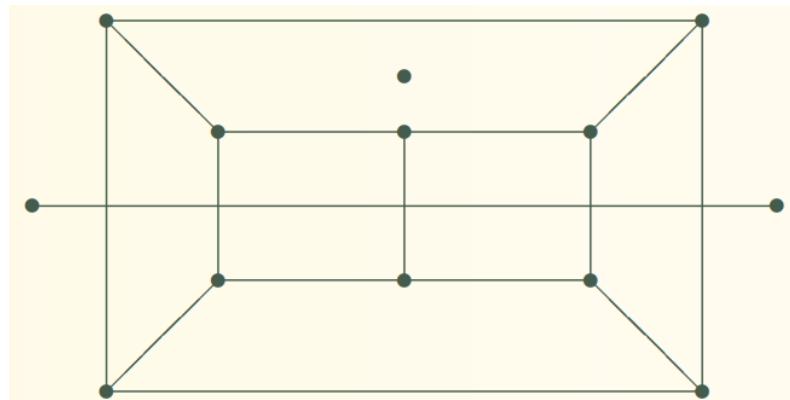
Relace  $\sim$  na množině vrcholů  $V(G)$  libovolného grafu  $G$ , je definována tak, že  $u, v \in V(G)$  jsou v relaci  $u \sim v$ , právě když v grafu  $G$  existuje **sled** začínající v  $u$  a končící ve  $v$ . Tato relace je:

- **reflexivní**: každý vrchol je spojen sám se sebou sledem délky 0.
- **symetrická**: sled z  $u$  do  $v$  lze obrátit na sled z  $v$  do  $u$  vždy u neorientovaného grafu.
- **tranzitivní**: dva sledy na sebe můžeme vždy **navázat v jeden**.

Relace  $\sim$  je tedy relací **ekvivalence**.

## Komponenty souvislosti

Jsou jednotlivé **třídy ekvivalence** grafu (graf je **souvislý**, pokud má **pouze jednu** komponentu souvislosti). Graf o střech komponentách:



## Stromy

Strom je jednoduchý **souvislý graf  $T$  bez kružnic**. Grafy bez kružnic lze také nazývat **acyklické**. **Les** je nesouvislý graf tvořený více stromy. Stromové grafy jsou zároveň kostrou grafu. Pro stromy platí:

- pokud mají více než jeden vrchol, existuje vrchol se **stupněm 1**,
- mezi každými dvěma vrcholy vede **právě** jedna cesta.

## Grafové algoritmy

Využívají nějakou paměť - zásobník, frontu, seřazené pole.

## Prohledávání do šířky - BFS

Algoritmus postupně prochází celé úrovně od počátečního vrcholu. Jako paměť využívá **frontu**. Pracuje následovně:

1. První vrchol se vloží do fronty,
2. Pokud není fronta prázdná, zpracuj vrchol na **začátku fronty**. Zpracování znamená umístění vrcholů, do kterých vede hrana ze zpracovávaného vrcholu, **do fronty**.

Algoritmus BFS lze použít pro zjištění nejkratší **vzdálenosti** mezi dvěma vrcholy spojeného **neváženého** grafu.

## Prohledávání do hloubky - DFS

Algoritmus prochází nejprve do co nejvzdálenější úrovně a poté se postupně vynořuje a zase co nejvíce zanořuje. Jako paměť využívá zásobník a pracuje následovně:

1. První vrchol se vloží na zásobník,
2. Pokud není zásobník prázdný, zpracuje se vrchol na **vrcholu zásobníku**. Zpracování znamená umístění vrcholů, do kterých vede hrana ze zpracovávaného vrcholu, na **zásobník**.

## Dijkstrův algoritmus

Algoritmus pro hledání **nejkratší cesty** (vzdálenosti) mezi dvěma **vrcholy u a v v kladně váženém grafu**. Jako úložiště používá **prioritní frontu**. Může pracovat s časovou komplexitou  $O((hrany+vrcholy)*\log(vrcholy))$  nebo  $O(vrcholy^2)$  při použití pole. Princip algoritmu:

1. Všechny vrcholy až na počáteční jsou ohodnoceny **nekonečnem** (neznáme do nich cestu), počáteční vrchol je ohodnocen **0** (cesta do tohoto vrcholu je nulová). Všechny vrcholy jsou označeny za nezpracované.
2. Z nezpracovaných vrcholů vybereme ten s **nejmenší hodnotou** (na začátku prioritní fronty - při 1. iteraci to bude počáteční vrchol - vrchol **u**). Pokud je tento vrchol hledaným vrcholem (vrchol **v**), ukončíme algoritmus, jinak **přepíšeme** vzdálenosti všech vrcholů (již zpracované ignorujeme), do kterých se můžeme z **vybraného vrcholu dostat hranou**, **přičtením ohodnocení** této hrany k hodnotě **zpracovávaného** vrcholu, pokud je tato hodnota **MENŠÍ** než aktuální ohodnocení. V tomto případě si také **zaznamenáme**, přes **jaký vrchol** vede tato nejkratší cestu.
3. Aktuálně zpracovávaný uzel označíme za zpracovaný a pokud jsou ještě nezpracované vrcholy, pokračujeme 2. bodem.
4. Zpětně **rekonstruujeme cestu** z cílového vrcholu k startovacímu vrcholu na základě **zapamatovaných** údajů v bodě 2.

▶ Dijkstra's Algorithm - Computerphile

## Jarníkův (Primův) algoritmus

Algoritmus, který hledá **minimální kostru ve váženém grafu** (stromový podgraf, který propojuje všechny vrcholy) - **minimum spanning tree**. Jako úložiště může použít např. prioritní frontu pro nenavštívené. Princip algoritmu:

1. Do úložiště ulož všechny vrcholy s ohodnocením **nekonečno** (ještě nejsou v kostře)
2. Vyber jakýkoliv uzel grafu, změň jeho ohodnocení na **0** a odstraň jej z úložiště.
3. U vrcholů (musí být v úložišti, tj. **ještě nenavštívené**), do kterých se lze z tohoto vrcholu dostat **aktualizuj** jejich **ohodnocení** hodnotou cesty k tomuto vrcholu, pokud je tato hodnota **menší než aktuální**, a zapiš vrchol, ze kterého tato **cesta vede**.
4. Vyber vrchol z úložiště, který má **nejmenší ohodnocení**, a odstraň jej z úložiště.
5. Pokud úložiště **není prázdné**, pokračuj s bodem 3.
6. Zpětně rekonstruuji použité hrany.

▶ Prim's algorithm in 2 minutes — Review and example

Minimum edge weight data structure	Time complexity (total)
adjacency matrix, searching	$O( V ^2)$
binary heap and adjacency list	$O(( V  +  E ) \log  V ) = O( E  \log  V )$
Fibonacci heap and adjacency list	$O( E  +  V  \log  V )$

## Kruskalův algoritmus

Algoritmus pro hledání minimální kostry ve váženém grafu. Pracuje na principu, že z grafu vybíráme vždy **hranu s nejmenším ohodnocením**, aby se **nevytvořila kružnice**, dokud nespojíme všechny vrcholy. Komplexita je  **$O(|E| * log(|V|))$** .

▶ Kruskal's Algorithm: Minimum Spanning Tree (MST)

# 26. Řešení úloh (prohledávání stavového prostoru, rozklad na podúlohy, metody hraní her).

## Stavový prostor

Graf, kde uzly představují jednotlivé stavy úlohy a hrany představují použité operátory. Je to dvojice (**S, O**), kde:

- S - množina stavů úlohy - **uzly** (vrcholy) grafu.
- O - množina operátorů - **hrany** grafu.

## Úloha

Dvojice (**S<sub>0</sub>, G**),  $S_0 \in S$  a  $G \subset S$  kde:

- $S_0$  - **počáteční stav**.
- G - **množina** všech konečných/**cílových** stavů.

## Hodnotící kritéria prohledávacích metod

- **Úplnost** - Algoritmus je úplný, když vždy najde řešení, pokud daná úloha řešení má. Každá úplná metoda pro **CSP** (Constraint Satisfaction Problem - záleží pouze na nalezení cílového stavu, posloupnost operátorů je irrelevantní) je **optimální**.
- **Optimálnost** - Pokud existuje více možných řešení, metoda najde to nejlepší. (optimální úloha je tedy **vždy úplná**).
- **Paměťová/Prostorová a časová složitost** - Např. lineární - **O(n)**, kvadratická **O(n^2)**, linearitmická **O(n\*logn)**, exponenciální **O(2^n)**, ...

## Metody řešení úloh prohledáváním stavového prostoru

- **Neinformované/Slepé metody**: Nevyužívají **žádné** informace, které by mohly **usnadnit** nalezení řešení. Používají se pouze pokud o řešené úloze skutečně žádné informace **nemáme**.
- **Informované metody** - Využívají nějaké informace (**heuristické funkce**) o řešené úloze, které mohou **usnadňovat** nebo **umožňovat** jejich řešení.
- **Metody lokálního prohledávání** - Místo **systematického** prohledávání stavového prostoru prohledávají pouze **okolí aktuálního stavu**. Vhodné pro řešení optimalizačních problémů. **Nemusí být úplné ani optimální**.

## Pojmy

- **Expanze uzlu** - Určení všech jeho bezprostředních následovníků - uzelů spojených hranou s expandovaným uzlem (postupná aplikace všech operátorů na uzel).
- **Ohodnocení uzlu** - Je dáno součtem **cen přechodů** (součet ohodnocení hran po cestě) z kořenového uzlu do tohoto uzlu.

## Slepé metody

### Prohledávání do šířky - Breadth First Search (BFS)

Úplná a optimální. Používá **frontu** pro ještě nezpracované uzly - **OPEN** a **seznam** pro již zpracované uzly - **CLOSED**.

1. Sestroj frontu OPEN a seznam CLOSED (pro variantu s CLOSED). Do OPEN vlož počáteční uzel.
2. Je-li fronta OPEN prázdná, pak úloha **nemá řešení**, jinak pokračuj.
3. Vyber z **čela** fronty OPEN uzel.
4. Je-li **cílovým** ukonči prohledávání jako **úspěšné a vrat' cestu**, jinak pokračuj.
5. Expanduj uzel (zde lze také testovat na to, jestli je uzel cílový), jeho následovníky co **nejsou** v OPEN ani v CLOSED, umísti do fronty OPEN, expandovaný uzel do CLOSED. Vrať se na bod 2.

Složitost je stejná časově i prostorově a závisí na **faktoru větvení - b** a hloubce **cílového uzlu - d**:

- **bez modifikace** v bodě 5:  $O(b^{d+1})$ ,
- **s modifikací** v bodě 5:  $O(b^d)$ .

### Prohledávání do hloubky - Depth First Search (DFS)

1. Sestroj **zásobník** OPEN a umísti do něj počáteční uzel.
2. Je-li OPEN **prázdný**, pak úloha **nemá řešení**, jinak pokračuj.
3. Vyber z vrcholu OPEN první uzel.
4. Je-li uzel uzlem **cílovým**, ukonči prohledávání jako **úspěšné a vrat' cestu**, jinak pokračuj.
5. Expanduj vybraný uzel a do OPEN vlož všechny jeho **bezprostřední** následníky (modifikace: bezprostředního následníka vlož na zásobník jen pokud tam **ještě není** a pokud **není předkem** právě expandovaného uzlu). Vrať se na bod 2.

Metoda bez modifikace **není úplná ani optimální** (do nekonečna se můžeme pohybovat v cyklu). Po modifikaci **je** metoda **úplná** ale stále **není optimální**.

- **bez modifikace**: časově i prostorově má nekonečnou časovou složitost,
- **s modifikací**: časová složitost je  $O(b^m)$  a prostorová složitost je  $O(m)$ , kde **b** je faktor větvení a **m** je počet **různých stavů**. U stromové struktury je prostorová složitost  $O(b^d)$ , kde **d** je hloubka.

## Prohledávání do omezené hloubky - Depth Limited Search (DLS)

Jedná se o metodu DFS, která prohledává pouze do omezené hloubky od počátečního uzlu. **Není úplná ani optimální.**

1. Sestroj **zásobník OPEN** a umístí do něj počáteční uzel.
2. Je-li OPEN **prázdné**, pak úloha **nemá řešení**. Jinak pokračuj.
3. Vyber z vrcholu OPEN první uzel.
4. Je-li vybraný uzel uzlem **cílovým**, ukonči prohledávání jako **úspěšné** a vrať cestu. Jinak pokračuj.
5. Je-li **hloubka** vybraného uzlu **menší** než zadaná **maximální hloubka**, tak tento uzel expanduj a do OPEN vlož všechny následovníky, kteří tam ještě nejsou. Vrať se na bod 2.

Složitost závisí na prohledávané hloubce, **b** - faktor větvení, **I** - prohledávaná hloubka:

- **časová:**  $O(b^*I)$ ,
- **prostorová:**  $O(b^I)$ .

## Prohledávání do omezené hloubky s postupným zanořováním - Iterative Deepening Search (IDS)

Metoda je **úplná i optimální**.

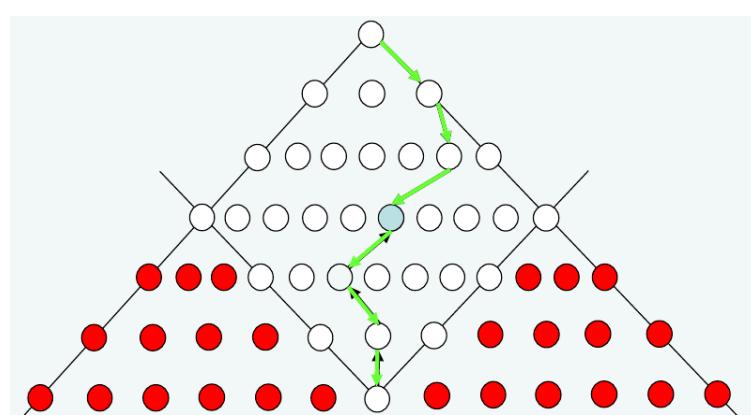
1. Nastav aktuální maximální hloubku na **1**.
2. Zavolej proceduru **DLS** s omezením na nynější hloubku.
3. Skončí-li **DLS** s úspěchem, ukonči prohledávání s úspěchem a vrať cestu.
4. Skončí-li **DLS** s neúspěchem, tak pokud v DLS **nebyl alespoň jeden uzel expandován z důvodu dosažení maximální hloubky**, inkrementuj maximální hloubku a vrať se na bod 2. Jinak ukonči prohledávání jako **neúspěšné**.

Složitost je dána maximální hloubkou, **b** - faktor větvení, **d** - maximální prohledávaná hloubka

- **časová:**  $O(b^*I)$ ,
- **prostorová:**  $O(b^d)$ .

## Obousměrné prohledávání - Bidirectional Search (BS)

Metoda je **úplná i optimální**. Metodu lze použít pouze na řešení úloh s **reverzibilními** operátory (např. **Loydova osmička**). Současně prohledává prostor **od počátečního stavu i od cílového stavu** a hledá uzel, ve kterém se obě prohledávání setkají.



Časová i prostorová složitost metody je stejná, **b** - faktor větvení, **d** - hloubka řešení:

- $O(2^b \cdot b^{d/2})$ , což odpovídá  $O(b^d \cdot b^{d/2})$ .

## Prohledávání do šířky s respektováním cen přechodů - Uniform Cost Search (UCS)

Metoda je **úplná** i **optimální**. Není vhodná pro úlohy, kde **optimální řešení** leží na **málo cestách s vysokou cenou** (prohledává se zbytečně mnoho cest s malými cenami). Jedná se o variantu Dijkstrova algoritmu, kde do **prioritní fronty** se místo všech uzlů vkládají pouze ty, ke kterým jsme již došli.

1. Sestroj dva **seznamy** OPEN (prioritní fronta - uzly určené k expanzi) a CLOSED (již expandované uzly - existuje verze i bez tohoto seznamu). Do open vlož počáteční uzel včetně jeho ohodnocení.
2. Je-li seznam OPEN **prázdný**, pak úloha **nemá řešení**. Jinak pokračuj.
3. Vyber ze seznamu OPEN uzel s **nejnižším ohodnocením**.
4. Je-li vybraný uzel uzlem **cílovým**, ukonči prohledávání jako **úspěšné** a vrát cestu. Jinak pokračuj.
5. Vybraný uzel expanduj a jeho následníky, kteří nejsou v CLOSED umísti do OPEN (**včetně jejich ohodnocení**). Expandovaný uzel vlož do CLOSED. Z uzlů, které se v OPEN vyskytují **vícekrát** ponech jenom ten s **nejnižším ohodnocením** (jinak řečeno aktualizuj ohodnocení, pokud je lepší). Vrať se na bod 2.

Časová i prostorová složitost je dána **cenou optimálního řešení  $C^*$**  a **minimálním přírůstkem ceny  $\Delta C_{min}$**  mezi dvěma uzly:

- $O(b^d \cdot (C^* / \Delta C_{min}))$

## Prohledávání se zpětným navracením - Backtracking Search

Metoda je vhodná i pro **CSP** (pokud aplikace operátoru vede na stav porušující omezující podmínky, pak je tento operátor považován za **neaplikovatelný** - bod 3 algoritmu), **je úplná**, ale **není optimální**. Backtracking je podobný DFS, ale místo expanze uzlu generuje pouze jediného následníka - nejprve jednoho a při návratech pak další.

- 1. Sestroj zásobník OPEN a umísti do něj počáteční uzel.
- 2. Je-li OPEN prázdný, pak úloha nemá řešení - ukonči prohledávání jako neúspěšné. Jinak pokračuj.
- 3. Jde-li na uzel na vrchu zásobníku aplikovat první/další operátor, tak tento operátor aplikuj a pokračuj bodem 4. Jinak odstraň testovaný uzel z vrcholu a vrať se na 2.
- 4. Je-li vygenerovaný uzel uzlem cílovým, ukonči prohledávání jako úspěšné a vrať cestu. Jinak ulož uzel na vrchol zásobníku (modifikace: uzel na zásobník ulož, pokud se tam ještě nenachází) a vrať se na 2.

Časová a prostorová složitost je stejná jako pro DFS:

- **bez modifikace**: časově i prostorově má nekonečnou časovou složitost,

- **s modifikací:** časová složitost je  $O(b^m)$  a prostorová složitost je  $O(m)$ , kde  $b$  je faktor větvení a  $m$  je počet **různých stavů**.

## Prohledávání s dopřednou kontrolou - Forward Checking Search

Vhodné i pro CSP. Metoda je **úplná i optimální**.

- ▶ forward checking, CSP heuristics .

Funguje na principu, že každé proměnné (uzlu) přiřadí množinu všech přípustných hodnot pro danou proměnnou (u barvení mapy to budou **všechny dostupné barvy**). U první proměnné vybereme některou z přípustných hodnot (např. modrou barvu) a aktualizujeme přípustné hodnoty u ostatních proměnných (proměnné, které na mapě sousedí s právěobarvenou proměnnou nemohou být modré). Vybereme další proměnnou a jí přiřadíme hodnotu (např. zelenou) a takhle postup opakujeme dokud nejsou pro všechny proměnné vybrány hodnoty. Může se ale stát, že u některé proměnné dojde k odstranění všech možných hodnot, které ji lze přiřadit (už nelze obarvit, aby nebylo porušeno pravidlo). Pak musíme u předešlých proměnných změnit jejich přiřazení (postupně se vynořovat a měnit). Pokud již nejde přiřazení nijak změnit, úloha nemá řešení.

### Heuristiky pro výběr proměnných:

- Proměnná s **nejmenším počtem přípustných hodnot**.
- Proměnná, která má **největší vliv na omezení zbývajících volných proměnných**.

### Heuristiky pro přiřazení hodnoty proměnné:

- Vyber hodnotu, která **vylučuje nejméně hodnot**, které mají společná omezení s vybranou proměnnou.

Složitost  $n$  - počet proměnných,  $m$  - počet přípustných hodnot:

- časová:  $O(m^n)$ ,
- prostorová:  $O(n)$ .

## Prohledávání s minimalizací konfliktů - Min-Conflict Search

Vhodné i pro CSP. **Neexistuje důkaz** o její **úplnosti ani optimálnosti** a tedy ani odhad časové složitosti.

1. Přiřaď každé proměnné  $x_i$  ( $i = 1, \dots, n$ ) **libovolnou** hodnotu z množiny **přípustných** hodnot. Nastav pomocné proměnné  $i$  (počítadlo proměnných) a  $j$  (počítadlo **správně** přiřazených proměnných) na hodnoty 1.
2. Spočítej počet **možných konfliktů** pro každou hodnotu proměnné  $x_i$ .
3. Pokud existuje pro jinou možnou hodnotu  $x_i$  počet konfliktů **menší nebo stejný** než pro její aktuální hodnotu, tak ji změň na tuto hodnotu a nastav hodnotu  $j$  na 1, jinak **inkrementuj**  $j$ .
4. Pokud  $j == n$  (všechny proměnné jsou přiřazeny nejlépe), pak bylo nalezeno optimální řešení. Jinak inkrementuj  $i$  a **pokračuj**.
5. Je-li hodnota  $i > n$ , pak  $i = 1$  a pokračuj bodem 2.

V paměti si udržuje pouze nynější stav a par proměnných, tedy prostorová náročnost je:  $O(1)$

## Informované metody

Používají heuristické funkce pro **odhad ceny** z aktuálního stavu do cíle a přičítají jej k aktuální ceně cesty.  $f(s_k)$  je ohodnocení k-tého stavu, které je dáno součtem:

- $g(s_k)$ : cena cesty od **počátečního stavu** ke **k-tému stavu**.
- $h(s_k)$ : je odhadovaná (pomocí **heuristické funkce**) cena cesty od **k-tého stavu** ke stavu **cílovému**.

### Prohledávání od nejlepšího - Best First Search (BestFS)

- Sestroj **seznam OPEN** (popř. CLOSED) a vlož do něj počáteční uzel včetně jeho ohodnocení.
- Je-li seznam OPEN prázdný, pak úloha nemá řešení - skončí s neúspěchem. Jinak pokračuj.
- Vyber z OPEN uzel s nejlepším ohodnocením.
- Je-li vybraný uzel uzlem cílovým - skončí s úspěchem a vrať cestu. Jinak pokračuj.
- Expanduj vybraný uzel. Všechny jeho následovníky co nejsou předky (využití CLOSED), umísti do OPEN včetně jejich ohodnocení. Z těch co jsou v OPEN **vícekrát** ponech pouze uzel s **nejlepším ohodnocením**. Vrať se na 2.

**Neinformovaná metoda UCS** je extrémním případem BestFS, u které je hodnota  $h(s_k)$  **vždy 0**.

### Chamtvé prohledávání - Greedy Search (GS)

Jedná se také o extrémní případ BestFS, kdy  $g(s_k) = 0$ . **Úplná** (pokud se používá seznam CLOSED), ale **není optimální**.

Časová a prostorová složitost je  $O(b^d)$ , kde **b** je faktor větvení a **d** je hloubka.

### A\* prohledávání/Search

Spadá pod BestFS. Je **úplná a optimální**. Heuristická funkce je **spodním odhadem skutečné ceny (nejlepší/nejlevnější odhad ceny)** cesty od ohodnoceného uzlu k cíli. Tato funkce nikdy **nesmí** odhadovanou cenu cesty **přecenit**, jinak algoritmus nebude fungovat správně,  $h = 0$  je přípustná vždy (ale pak je to metoda UCS).

Časová i prostorová náročnost závisí na heuristice, pohybuje se od:

- pro **h** blízké nule:  $O(b^d)$ ,
- pro **h** rovné skutečné ceně:  $O(d)$ ,

kde **b** je faktor větvení a **d** je hloubka cíle.

## Metody lokálního prohledávání

většinou nejsou úplné ani optimální, ale mohou být například rychlé.

## Hill Climbing

**Není úplná ani optimální.** Umí jít pouze jedním směrem a nedokáže se vracet. Pokud by měla hledat např. nejvyšší horu z údolí, je velmi pravděpodobné, že se zastaví na prvním kopci, protože neumožňuje klesání.

1. Vytvoř uzel **Current** a ulož do nej počáteční stav  $s_0$  spolu s jeho ohodnocením.
2. Expanduj **Current**, ohodnoť jeho bezprostřední následníky a vyber z nich **nejlépe ohodnoceného (Next)**.
3. Je-li ohodnocení **Current** lepší než ohodnocení **Next**, ukonči řešení a vrať **Current**. Jinak pokračuj.
4. Nahraď **Current** uzlem **Next**. Vrať se na 2.

Prostorová složitost je **O(1)** - používáme pouze 2 proměnné Current a Next. Časová složitost je **O(d)** - jdeme pouze jednou cestou.

## Simulated Annealing

**Není úplný ani optimální.** Pracuje jak s ohodnocením, tak s **náhodností**. Narozdíl od Hill Climbing dokáže opustit lokální extrémy. Je inspirována tuhnutím kovů. S postupně **klesající pravděpodobností** (tuhnutím kovu s klesající teplotou) umožňuje vybrat **hůře** ohodnocený uzel - na začátku algoritmu je poměrně pravděpodobné, že opustíme lokální extrém.

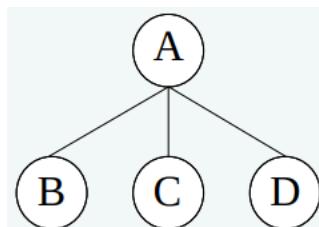
1. Vytvoř tabulku pro klesání "teploty" v závislosti na kroku výpočtu.
2. Vytvoř pracovní uzel Current a ulož do něj počáteční stav a jeho ohodnocení. Nastav krok výpočtu na 0 ( $k=0$ ).
3. Z tabulky zjistí aktuální teplotu  $T$ . Je-li  $T=0$ , ukonči řešení a vrať jako výsledek Current. Jinak pokračuj.
4. Expanduj Current a z jeho následovníků vyber náhodně jednoho z nich (Next).
5. Vypočítej rozdíl ohodnocení uzlů Current a Next  
 $\Delta E = \text{value}(\text{Current}) - \text{value}(\text{Next})$ .
6. Pokud  $\Delta E > 0$ , pak nahraď uzel Current uzlem Next. Jinak je zaměň s pravděpodobností  $p = e^{\Delta E/T}$ .
7. Inkrementuj  $k$  a vrať se na 3.

Časová složitost závisí na rychlosti klesání teploty (rychlosti klesání pravděpodobnosti, že bude vybrán hůře ohodnocený uzel). Prostorová složitost je **O(1)** - konstantní.

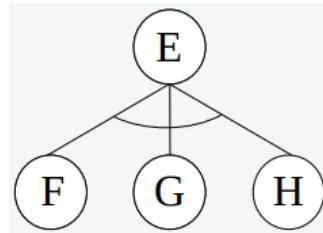
## Metody řešení úloh rozkladem na podproblémy (AND/OR grafy)

U těchto úloh uzly znamenají **problémy/podproblémy NIKOLIV** stavy. Problém lze rozložit dvěma způsoby:

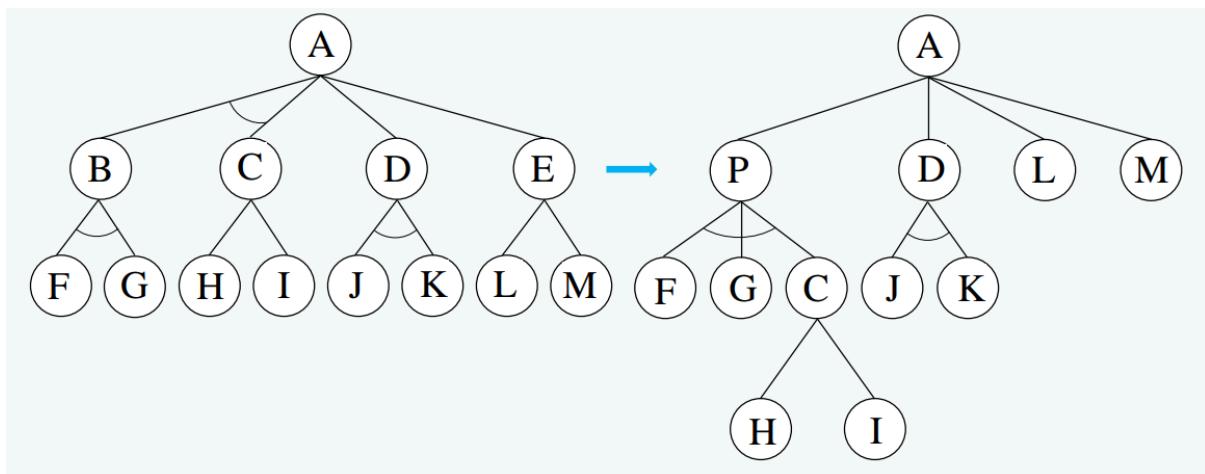
- **OR uzel:** Problém je řešitelný, pokud je **alespoň jeden** z podproblémů řešitelný.



- AND uzel: Problém je řešitelný, pokud jsou **řešitelné všechny** jeho podproblémy.



Obecný **AND/OR** graf (graf s uzly AND a OR) lze převést na graf, ve kterém jsou v každé **vrstvě** buď pouze **AND**, nebo **OR** uzly. Každou musí být možné převést ze stavového prostoru na **rozklad podproblémů**.



### Slepé AND/OR graf pro BFS, DFS

BFS a DFS prohledávání s tím, že řešitelnost uzlu je dána jeho typem (AND, OR, případně kombinací AND a OR) a řešení spočívá v dokázání, že je **řešitelný i počáteční uzel** (kořen). Toho se docílí **propagováním informace o řešitelnosti/neřešitelnosti** z nižších vrstev do vyšších a na tom je algoritmus založený.

1. Sestroj OPEN (**fronta** pro BFS, **zásobník** pro DFS) a prázdný **graf/strom G** a do obou ulož počáteční uzel (problém), který **nesmí** být elementárně řešitelný nebo neřešitelný.
2. Výjmi z OPEN uzel a označ jej X.
3. Expanduj X a všechny jeho následovníky připoj ke grafu G.
  - a. Pro všechny **řešitelné** následníky X přenes **informaci o jejich řešitelnosti jejich předchůdcům**. Je-li **řešitelný počáteční problém**, ukonči řešení jako **úspěšné**.
  - b. Pro všechny **neřešitelné** následníky X přenes **informaci o jejich neřešitelnosti jejich předchůdcům**. Není-li **řešitelný počáteční problém**, ukonči řešení jako **neúspěšné**.
  - c. Všechny ostatní následníky X ulož do OPEN.

4. Odstraňte z OPEN všechny uzly, které mají **vyřešení předchůdce** (nedává smysl je už řešit).
5. Je-li OPEN **prázdný** - skonči s **neúspěchem**. Jinak se vrať na 2.

## Slepý AO pro Backtracking

Podobný jako DFS, test uzlu na **řešitelnost/neřešitelnost** probíhá **před** generováním jeho následníka - vždy se generuje pouze **jeden následník**, který se vyhodnotí a až poté se **případně generuje další**.

1. Sestroj **graf G a zásobník OPEN** a do obou ulož počáteční uzel.
2. Je-li uzel na vršku OPEN řešitelný, pak:
  - a. Je-li tento uzel počáteční - skonči **úspěšně**.
  - b. Jinak přenes informaci o řešitelnosti na předchůdce a uzel z OPEN odstraň.
3. Je-li uzel na vršku neřešitelný, pak:
  - a. Je-li tento uzel uzlem počátečním, ukonči řešení jako **neúspěšné**.
  - b. Jinak přenes informaci o neřešitelnosti na předchůdce, a uzel odstraň z OPEN.
4. Není-li uzel na vršku OPEN řešitelný/neřešitelný, pak **generuj** jeho následníka, ulož ho do OPEN a připoj k G.
5. Vrať se na bod 2.

## Informovaný AO\* algoritmus

Algoritmus řeší, jestli je uzel řešitelný a také jaká je cena vyřešení uzlu. Může skončit neúspěšně, i když existuje řešení, pokud je převýšena jeho **maximální cena**.

Výpočet ceny pro uzly je (může být doplněno o heuristickou funkci):

- Ohodnocení **AND** je rovno **součtu** ohodnocení všech jeho následníků.
- Ohodnocení **OR** je rovno **nejmenšímu** z ohodnocení jeho následníků.

Algoritmus pracuje následovně:

1. Sestroj strukturu G (AO strom) a umísti do něj počáteční uzel (INIT) s ohodnocením. Stanov hondotu FUTILITY (maximální povolenou cenu).
2. Je-li INIT označen jako řešitelný (SOLVED), ukonči řešení jako úspěšné. Je-li  $\text{INIT} \geq \text{FUTILITY}$ , skonči s neúspěchem. Jinak pokračuj.
3. Procházej nejnadějnější podstrom až narazíš na neexpandovaný uzel (NODE).
4. Expanduj NODE, pokud nemá následníka, přiřaď mu hodnotu FUTILITY a přejdi na 7. Jinak pokračuj.
5. Představují-li někteří bezprostřední následníci elementární úlohy, označ je SOLVED (0 ohodnocení) u zbývajících jej vypočti.
6. Připoj následníky NODE ke stromu G a přenes informaci o jejich ohodnocení směrem k INIT.
7. V uzlech OR označ nejnadějnější podstomy.
8. Vrať se na bod 2.

graficky viz: <https://www.goeduhub.com/7063/implement-ao-search-algorithm>

## Metody hraní her

Cílem je určit tah hráče na tahu, který povede k vítězství, nebo pro který je pravděpodobnost jeho vítězství největší. Nejprve budeme uvažovat pouze hry, které hrají dva pravidelně se střídající hráči, které označíme symboly **A** a **B**. Každý z nich chce vyhrát a oba mají úplný přehled o aktuálním stavu hry. Pro tyto hry obecně platí:

- Hráč na tahu (označuje se vždy jako hráč **A**) zvítězí, vede-li k jeho vítězství alespoň jeden tah – problém **OR**.
- Hráč na tahu zvítězí, vedou-li po jeho tahu k jeho vítězství všechny možné tahy protihráče (hráče **B**) – problém **AND**.

## Rozdělení her dvou protihráčů

- **Jednoduché hry** - Lze prozkoumat všechny možné tahy (NIM). K řešení lze použít **AND/OR** algoritmy.
- **Složité hry** - Zkoumá se pouze několik následujících tahů (šachy).
- **Hry s neurčitostí** - (hod kostkou např.) Zkoumá se pouze několik následujících tahů s respektováním **neurčitosti** (hod kostkou - člověče). Pracuje se s očekávanými hodnotami.

## MinMax (Složité hry)

Každý stav je ohodnocen hodnotící funkcí (kladná příznivé pro hráče na tahu - **A**, vítězství/prohra má hodnotu +/nekonečno). Hráč **A** chce maximální hodnoty - v **OR** uzlu se bere **max** jeho potomků, **B** chce minimální - v **AND** uzlu se bere min jeho potomků. Řešením problému v daném stavu je určení **nejvhodnějšího tahu hráče A** (pro každý tah se musí řešit znova a znova). Metoda je **rekurzivní** a zahajuje se, když je **na tahu hráč A**. Vstupem jsou

- stavy hry X,
- maximální hloubka prohledávání.

Výstupem je

- **aktuální stav**,
- **tah**, který k tomuto stavu vede.

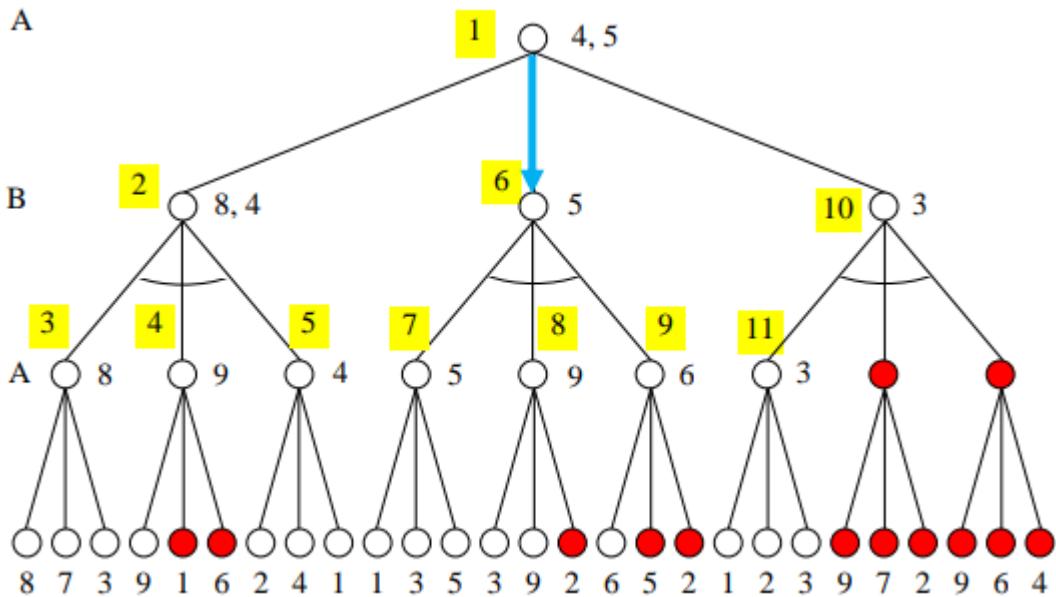
Algoritmus může vypadat následovně

1. Je-li uzel **X** listem vrací ohodnocení tohoto uzlu.
2. Je-li na tahu hráč **A**, tak postupně pro všechny jeho možné tahy volá **MinMax** pro hráče **B** a vrací **maximální** z navrácených hodnot a tah, který k tomuto vede.
3. Je-li na tahu hráč **B**, tak postupně pro všechny jeho možné tahy (bezprostřední následníky uzlu **X**, tj. stavy před tahem hráče **A**) volá proceduru **MiniMax** pro hráče **A** a vrací **minimální** z navrácených hodnot.

Popis příkladu na obrázku:

- žlutá čísla určují **pořadí** při vyhodnocování,
- nepodbarvená čísla určují **ohodnocení uzlů** a změny jejich ohodnocení,

- červené uzly jsou vyhodnocovány **zbytečně** (už je jasné, že si hráč tuto větev stejně nezvolí. V kroku 4 je jasné, že hráč **B** si **radši** vybere větev kroku 3 (ohodnocení  $8 < 9$ ), než větev 4, která má již po prvním potomkovi ohodnocení 9 a už může mít pouze vyšší).



## AlfaBeta (Složité hry)

### ▶ Step by Step: Alpha Beta Pruning

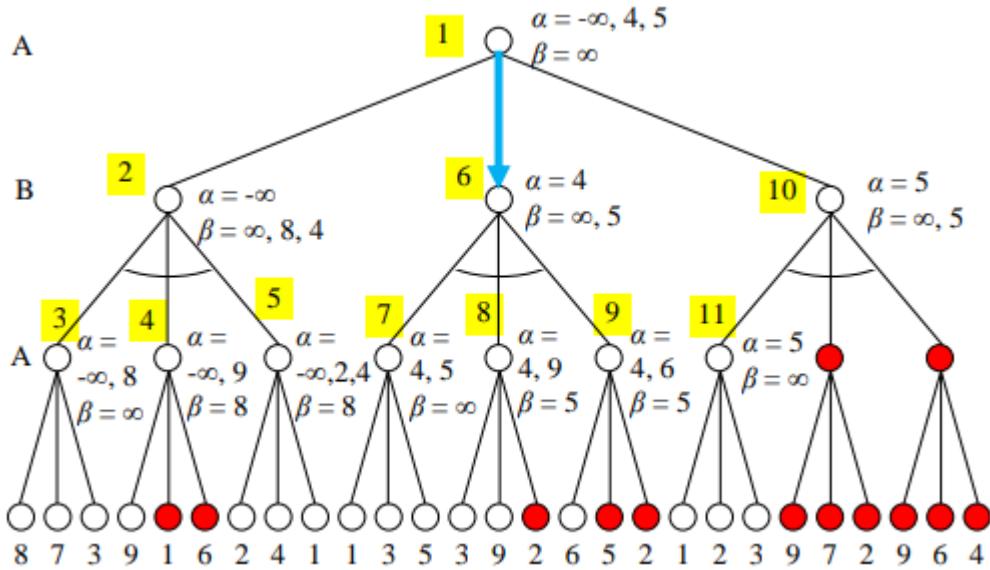
Zbytečnému vyšetřování uzlů lze zabránit pomocí alfa-beta řezů. **Alfa** řezy zabraňují zbytečnému vyšetřování tahů hráče **A**, **beta** řezy pak zbytečnému vyšetřování tahů hráče **B**. Procedura AlfaBeta vychází z procedury MiniMax, je rekurzivní, vstupními parametry jsou stejné a navíc přidává parametry  $\alpha$  a  $\beta$ , které jsou na začátku nastaveny na  $\alpha = -\infty$  a  $\beta = +\infty$  (tedy opačné hodnoty než hráči A a B požadují).

- Je-li uzel X listem, procedura vrací ohodnocení tohoto uzlu.
- Je-li **na tahu A**:
  - Dokud platí  $\alpha < \beta$ , tak postupně pro všechny tahy volá **AlfaBeta** s aktuálními hodnotami  $\alpha$  a  $\beta$  (po každém volání se  $\alpha$  mění). Po každém vyšetření tahu **nastaví  $\alpha$  na maximum** (postupně se zvětšuje z  $-\infty$ ) z aktuální a vrácené hodnoty.
  - Je-li  $\alpha \geq \beta$  (před každým dalším následník se **musí toto kontrolovat**), nebo nemá-li X žádného bezprostředního **následníka**, vrací aktuální hodnotu  $\alpha$  a tah, který vede k nejlepšímu stavu (v případě více stejných ten první nalezený).
- Je-li **na tahu B**:
  - Pokud platí  $\alpha < \beta$ , tak postupně pro tahy volá AlfaBeta s aktuálními  $\alpha$  a  $\beta$  (po každém volání se  $\beta$  mění). Po každém vyšetření nastaví  $\beta$  na **minimum** (postupně se zmenšuje z  $+\infty$ ) z aktuální a vrácené hodnoty.
  - Je-li  $\alpha \geq \beta$  nebo nemá-li X žádné následníky, vrací **aktuální  $\beta$** .

Popis příkladu na obrázku:

- žlutá čísla určují **pořadí** při vyhodnocování,

- nepodbarvená čísla určují **ohodnocení  $\alpha$  a  $\beta$**  a změny jejich ohodnocení,



## ExpectiMinMax (Hry s neurčitostí)

- ▶ Games: Q2. Expectiminimax
- ▶ AI U3 D5 Prababilistic Cut ExpectiMax AlphaBeta

Podobné MinMax, počítá se ale z pravděpodobnosti, že k nějakému stavu dojde. Metoda je rekuzivní a pracuje s **expectimin** pro hráče **B** (očekávané **minimum**) a **expectimax** pro hráče **A** (očekávané **maximum**). **Expectimin** a **expectimax** jsou spočteny jako **součet ohodnocení po všech možných výsledcích hodu kostky** (případně jiné pravděpodobnostní veličiny). Např. pokud má uzel 2 následovníky, první má ohodnocení 10, druhý má ohodnocení 5. Pravděpodobnost vybrání 1. následníka je ale pouze 0.25, druhého 0.75. celkové ohodnocení uzlu tak bude  **$0.25 \cdot 10 + 0.75 \cdot 5 = 6.25$** .

$$\text{expectimin}(C_i) = \sum_k P(h_k) * \min_j (D_{ikj})$$

$$\text{expectimax}(D_j) = \sum_k P(h_k) * \max_i (C_{jki})$$

V těchto vzorcích značí:

$C_i$	$i$ -tý možný tah hráče A po jeho hodu kostkou
$D_{ikj}$	ohodnocení stavu po $i$ -tém tahu hráče A, hodu $h_k$ a $j$ -tém tahu hráče B
$D_j$	$j$ -tý možný tah hráče B po jeho hodu kostkou
$C_{jki}$	ohodnocení stavu po $j$ -tém tahu hráče B, hodu $h_k$ a $i$ -tém tahu hráče A
$P(h_k)$	pravděpodobnost hodu $h_k$

- Je-li uzel **X** listem (konečným stavem hry, nebo uzlem v **maximální hloubce**) procedura vrací **ohodnocení tohoto uzlu**.

2. Je-li na tahu hráč **A**, tak postupně pro všechny jeho možné tahy (bezprostřední následníky uzlu X) volá proceduru **Expectiminimax** pro hráče **B**, vrací **maximální** hodnotu z navrácených hodnot **expectimin** a tah, který k nejlépe ohodnocenému bezprostřednímu následníkovi **vede** (tento **tah** má opět význam pouze u kořenového uzlu, kdy představuje nejvýhodnější reálný tah hráče **A**).
3. Je-li na tahu hráč **B**, tak postupně pro všechny jeho možné tahy (bezprostřední následníky uzlu **X**) volá proceduru **Expectiminimax** pro hráče **A** a vrací **minimální** hodnotu z navrácených hodnot **expectimax**.

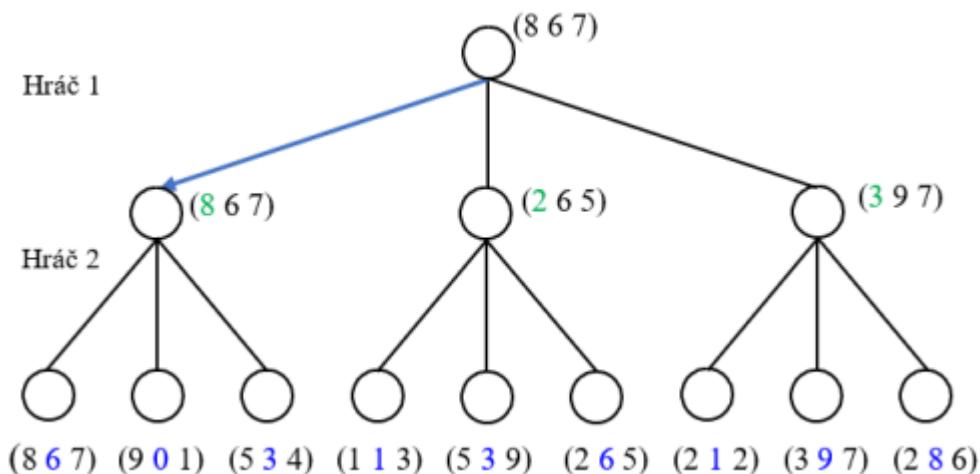
## Metody hraní her n hráčů

Hráči se pravidelně střídají, každý hráč chce vyhrát a všichni mají úplný přehled o aktuálním stavu hru.

Jednotlivé stavy hry jsou ohodnocovány **jedinou hodnotící funkcí** aplikovanou na **každého hráče zvlášť**, přičemž hodnota této funkce musí být **tím vyšší**, čím je daná pozice pro hodnoceného hráče **výhodnější**. U každého stavu je pak jeho hodnocení dánou **seznamem hodnot hodnotící funkce** pro jednotlivé hráče.

Například pro hru se třemi hráči ohodnocení stavu **(8 3 5)** znamená, že daný stav má ohodnocení **8 pro prvního** hráče, **3 pro druhého** hráče a **5 pro třetího** hráče.

Každý hráč si vybírá pro něj nejvýhodnější ohodnocení. Na obrázku jsou zeleně vyznačena ohodnocení pro hráče 1 a modře ohodnocení pro hráče 2.



## Procedura Max^n (hry více hráčů)

Jedná se o zobecnění procedury **MiniMax** pro **n** hráčů. Proceduru volá vždy hráč na tahu

- vstup: aktuální stav hry (uzel X)
- výstup: ohodnocení tohoto stavu

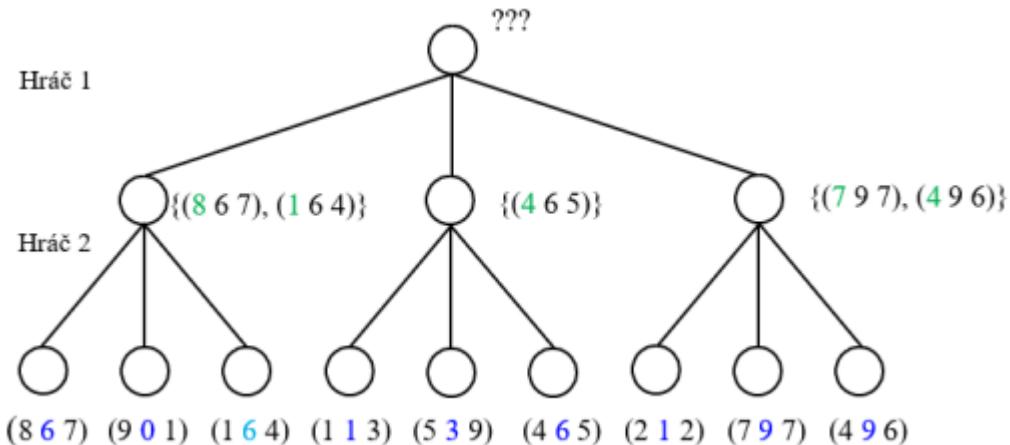
Princip algoritmu:

1. Je-li uzel X listem (konečným stavem hry nebo uzlem v maximální hloubce), tak procedura vrací ohodnocení tohoto uzlu ( $v_1, v_2, \dots, v_n$ ).

2. Jinak tato procedura pro aktuálního hráče  $i$  volá rekurzivně sama sebe na všechny své bezprostřední následníky (tj. stavy před tahem následujícího hráče  $j$  ( $j = i + 1$  a pokud  $j > n$ , pak  $j = 1$ ) a vrací ohodnocení, které je pro daného hráče, tj. hráče  $i$ , nejvýhodnější.

## Soft-Max<sup>n</sup>

Problémem procedury **Max<sup>n</sup>** je nejednoznačnost hodnocení uzlu (stavu hry) v případech, kdy stejné **maximální ohodnocení má několik** jeho bezprostředních následníků. Tento problém procedura Soft-Max<sup>n</sup> řeší, vracením množiny se stejnými ohodnoceními pro příslušného hráče, a tím umožňuje hráči na tahu informovanější rozhodování.



Pokud **hráči 1** stačí k vítězství **4** body, pak si tento hráč zřejmě vybere tah rovně dolů k uzlu **(4 6 5)**. V opačném případě může buď **riskovat**, tj. zvolit tah vlevo (hráč **2** si však může vybrat tah **(1 6 4)**, který je pro hráče 1 velmi nevýhodný - hodnota 1), nebo sázet raději na jistotu, tj. zvolit tah vpravo.

Sprehľadnenie algoritmov v tabuľke:

	Algoritms	Typ	Úplnosť	Optimálnosť	Využitie	Priestorová	Časová
1	BFS	Slepá	Áno	Áno	Fronta	$O(b^{d+1})$ (*)	$O(b^{d+1})$ (*)
2	BS	Slepá	Áno	Áno	Fronta	$O(2 \cdot b^{d/2}) \sim O(b^{d/2})$	$O(2^{d/2}) \sim O(b^{d/2})$
3	DFS	Slepá	Nie (*)	Nie	Zásobník	$O(\infty)$ (*)	$O(\infty)$ (*)
4	DLS	Slepá	Nie	Nie	Zásobník	$O(b \cdot l)$	$O(b^l)$
5	IDS	Slepá	Áno	Áno	Zásobník	$O(b \cdot d)$	$O(b^d)$
6	Backtracking	Slepá	Áno	Nie	Zásobník	$O(b \cdot m/b)$	$O(b^{m/b})$
7	Backtracking CSP	Slepá	Áno	Áno	Zásobník	$O(n)$	$O(m^n)$
8	UCS	Slepá	Áno	Áno	Seznam	$O(b^{C^b / \Delta c_{min}})$	$O(b^{C^b / \Delta c_{min}})$
9	A*	Informovaná	Áno	Áno	Seznam	$O(b^d) \sim O(d)$	$O(b^d) \sim O(d)$
10	GS	Informovaná	Áno (*)	Nie	Seznam	$O(b^d)$	$O(b^d)$
11	BestFS	Informovaná	Áno	Áno	Seznam		
12	Hill-climbing	Lokálne prehľadávanie					
13	Simulované žihání	Lokálne prehľadávanie					
14	AO – AND/OR	Lokálne prehľadávanie					

**GS (\*)** - Pokud se do seznamu OPEN ukládají všechni bezprostřední následníci expandovaného uzlu  $\Rightarrow$  i jeho předci(v bodu 5 se pak vypustí kontrola "kteří nejsou jeho předky") pak **GS není uplný**!

**DFS (\*)** - Modifikovaná metoda (s eliminací stejných stavů a předků v Open) je **úplná**, ale **není optimální**. Potom

Časová:  $O(b^{m/b})$   
Priestorová:  $O(b \cdot m/b)$

**BFS (\*)** - Metodu BFS lze modifikovat testováním cílového stavu již při vygenerování uzlu. Potom

Časová:  $O(b^d)$   
Priestorová:  $O(b^d)$

**Odkazy:**

- Tabulka metod ve větším:  
<https://cdn.discordapp.com/attachments/539908031157370900/577122492934520833/unknown.png>
- [Metody prohledávání](#)

# 27. Strojové učení (učení s učitelem, učení bez učitele, posilované učení).

Strojové učení (ML, Machine Learning) je schopnost inteligentního systému **měnit své znalosti** tak, aby příště vykonával stejnou nebo podobnou činnost **účinněji a efektivněji**.

## Učení s učitelem

Spočívá v tom, že pro každý krok učení je **známá požadovaná odezva** a systém je tak okamžitě informován o **aktuálním hodnocení jeho poslední akce**. Učení se provádí na tzv. **trénovací množině** příkladů **T**, kdy každý příklad je představován **množinou vstupních hodnot** (vstupním vektorem) a **množinou správných/požadovaných výstupních hodnot** (výstupním vektorem):

$$T = \{(\vec{i}_1, \vec{d}_1), (\vec{i}_2, \vec{d}_2), \dots, (\vec{i}_P, \vec{d}_P)\},$$

$\vec{i}_i$  ... i-tý vstupní vektor

$\vec{d}_i$  ... i-tý výstupní vektor

Často se **množina** dostupných dat **nepoužívá** k trénování **celá**, ale rozdělí se na dvě nebo tři podmnožiny. První (**trénovací, cca 80 % dat**) se použije k trénování, druhá (**testovací, cca 10 % až 20 % dat**) se použije k testování a třetí (**cca 10 % dat**) se někdy použije k doladění parametrů.

Metody učení s učitelem pracují s vektory, které nabývají symbolických hodnot (případné **číselné hodnoty** jsou rovněž chápány jako symbolické) a jsou založené na předpokladu, že každá hypotéza, která **vyhovuje dostatečně velké množině trénovacích příkladů**, bude vyhovovat i dalším, dosud **neznámým příkladům**.

## Tvorba rozhodovacích stromů (Decision Tree Building)

Rozhodovací stromy slouží ke klasifikaci objektů na základě hodnot jejich atributů/vlastností. Jsou vytvářeny ze známé množiny příkladů. Používají se při

Klient	Příjem	Dluh	Historie úvěrů	Ručení	Risk
1	< 15	vysoký	špatná	žádné	vysoký
2	15 – 35	vysoký	neznámá	žádné	vysoký
3	15 – 35	nízký	neznámá	žádné	přiměřený
4	< 15	nízký	neznámá	žádné	vysoký
5	> 35	nízký	neznámá	žádné	nízký
6	> 35	nízký	neznámá	adekvátní	nízký
7	< 15	nízký	špatná	žádné	vysoký
8	> 35	nízký	špatná	adekvátní	přiměřený
9	> 35	nízký	dobrá	žádné	nízký
10	> 35	vysoký	dobrá	adekvátní	nízký
11	< 15	vysoký	dobrá	žádné	vysoký
12	15 – 35	vysoký	dobrá	žádné	přiměřený
13	> 35	vysoký	dobrá	žádné	nízký
14	15 – 35	vysoký	špatná	žádné	vysoký

dolování dat z databází.

Příklad viz str 105:

<https://www.fit.vutbr.cz/study/courses/IZU/private/2022-opora-IZU.pdf>

## Algoritmus Decision Tree

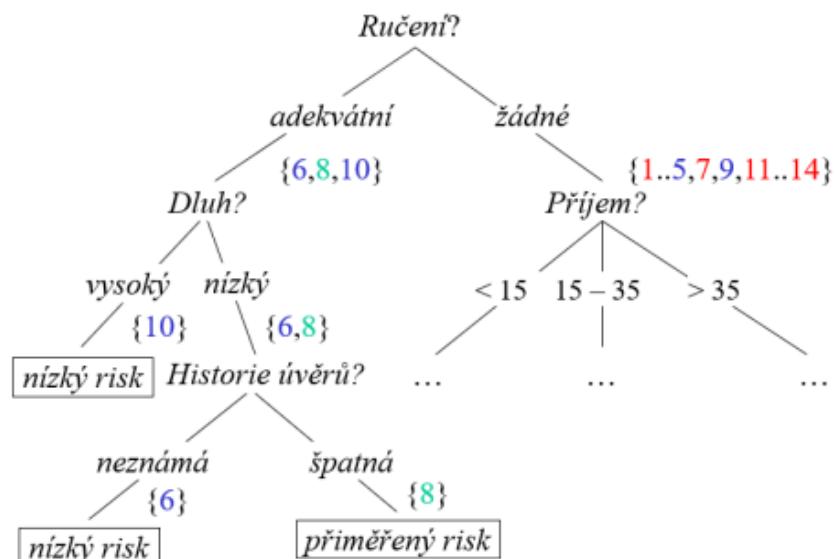
Jedná se o základní algoritmus pro budování **rozhodovacích stromů**. Má dva vstupní parametry:

- **množinu příkladů MP**: jednotlivé řádky tabulky viz výše,
- **množinu podmínkových atributů MA**.

Algoritmus funguje následovně:

1. Patří-li **všechny prvky** množiny příkladů **MP** do **stejné třídy**, vraťte **listový uzel** označený touto třídou, jinak pokračujte.
2. Je-li množina atributů **MA** prázdná, vraťte **listový uzel** označený **disjunkcí** všech tříd, do kterých patří prvky v množině příkladů **MP**, jinak pokračujte.
3. Vyberte atribut **A<sub>i</sub>**, odstraňte jej z množiny atributů **MA** a učiňte jej kořenem **aktuálního stromu**. Nechť **MA-i** je množina atributů **MA** bez atributu **A<sub>i</sub>**.
4. Pro každou hodnotu **H<sub>ji</sub>** vybraného atributu **A<sub>i</sub>**:
  - a. Vytvořte novou větev stromu označenou hodnotou **H<sub>ji</sub>**.
  - b. Volejte rekurzivně algoritmus s parametry **MP<sub>ji</sub>** a **MA-i**, kde množina **MP<sub>ji</sub>** je podmnožinou všech prvků množiny příkladů **MP**, které **mají** hodnotu **H<sub>ji</sub>** atributu **A<sub>i</sub>**.
  - c. Připojte vrácený podstrom/uzel k této věti.

Jinak řečeno, využívá **strom na základě rozhodovacích parametrů**. Hloubka stromu závisí na tom, jestli je možné nějaký rozhodovací atribut ignorovat, protože nehledě na jeho hodnotu jsou výsledky z trénovací množiny stejné.



Na základě trénovací množiny, viz tabulka výše, lze říct, že pokud má uživatel **adekvátní ručení** a **vysoký dluh** je **risk poskytnutí úvěru nízký** bez zohlednění jeho **příjmu a historie úvěru** (příjem a historie úvěru budou mít v realitě vliv při

rozhodnutí o riziku, ale trénovací množina je neobsahuje, takže se podle nich nemůže naučit a musí je ignorovat).

Algoritmus je jednoduchý, ale obsahuje zásadní problém, kterým je výběr atributu **A<sub>i</sub>** v bodě 3. **Nevhodné** výběry vedou k **hlubokým** a **neefektivním** vyhledávacím stromům, přestože **optimální strom** může být poměrně **jednoduchý**.

### Algoritmus ID3 (Induction of Decision Tree)

Jedná se o modifikaci algoritmu Decision Tree, který **řeší** problém při **výběru rozhodovacího atributu A<sub>i</sub>** v bodě 3. Výběr atributu není náhodný, ale je prováděn tak, aby byl **maximalizován informační zisk**, tj. aby byl **nejdříve** vybrán takový **atribut**, který co **nejvíce ovlivní výsledné rozhodnutí** (atribut, který ovlivňuje rozhodnutí úplně nejvíce je pak kořenem rozhodovacího stromu). Lze na to nahlížet také tak, že **množiny**, které vzniknou **rozdelením dle nějakého atributu** mají co **nejmenší míru entropie** (míru neuspořádanosti) tj. je v nich co nejvíce prvků spadajících pod stejný výsledek. Pokud jsou v množině 3 prvky, které spadají **všechny do jedné kategorie** výsledků (např. nízky risk), je míra **entropie** této množiny **nulová**. Pokud zde budou naopak 3 prvky spadající do **3 různých kategorií** (nízký, přiměřený a vysoký risk), je míra entropie této množiny **nejvyšší**. Vybráme tedy takový **atribut**, který rozdělí množinu na podmnožiny s **nejmenší entropií** a přinese tak největší informační zisk.

Informační zisk s respektováním hodnot **atributu Příjem (<15, 15-35, >35)** se vypočítá takto: **MP1 (<15) = {1, 4, 7, 11}**, **MP2 (15-35) = {2, 3, 12, 14}**, **MP3 (>35) = {5, 6, 8, 9, 10, 13}** (už jen podle rozdělení prvků do množin je zřejmé, že MP1 bude mít nulovou entropii, MP2 bude mít z této trojice největší entropii a entropie MP3 bude mezi).

$$E(MP_1) = -\frac{4}{4} \log_2 \left( \frac{4}{4} \right) = 0$$

$$E(MP_2) = -\frac{2}{4} \log_2 \left( \frac{2}{4} \right) - \frac{2}{4} \log_2 \left( \frac{2}{4} \right) = 1$$

$$E(MP_3) = -\frac{5}{6} \log_2 \left( \frac{5}{6} \right) - \frac{1}{6} \log_2 \left( \frac{1}{6} \right) = 0.650$$

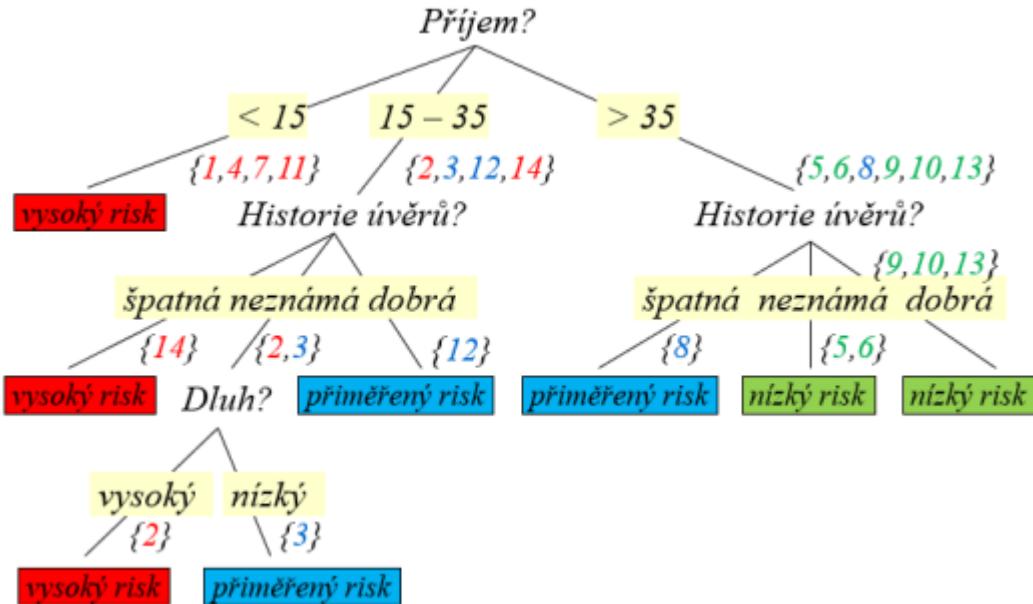
**Celková entropie** rozdelení podle atributu Příjem je pak spočítána jako **vážený průměr** (váha je dána počtem prvků v množině) entropií podmnožin.

$$E(MP, \text{Příjem}) = \sum_{i=1}^3 \frac{|MP_i|}{|MP|} E(MP_i) = \frac{4}{14} E(MP_1) + \frac{4}{14} E(MP_2) + \frac{6}{14} E(MP_3) =$$

$$= \frac{4}{14} \left( -\frac{4}{4} \log_2 \left( \frac{4}{4} \right) \right) + \frac{4}{14} \left( -\frac{2}{4} \log_2 \left( \frac{2}{4} \right) - \frac{2}{4} \log_2 \left( \frac{2}{4} \right) \right) +$$

$$+ \frac{6}{14} \left( -\frac{5}{6} \log_2 \left( \frac{5}{6} \right) - \frac{1}{6} \log_2 \left( \frac{1}{6} \right) \right) = 0.564$$

Výsledný prohledávací strom vytvořený pomocí ID3 bude vypadat následovně:



## Prohledávání prostoru verzí (Version Space Search)

Prohledávání prostoru verzí představuje soubor metod učení významných pojmu/hypotéz na základě **pozitivních** a **negativních** příkladů. Při učení se hledá takový **popis daného pojmu/objektu/hypotézy**, který **zahrnuje všechny pozitivní příklady a vylučuje všechny negativní příklady** z trénovací množiny příkladů. Trénovací množina **vždy** musí obsahovat **pozitivní i negativní** příklady. Příklad trénovací množiny pro identifikaci pojmu může být následující:

- |                                           |         |
|-------------------------------------------|---------|
| 1. objekt( <i>malá,červená,koule</i> )    | kladný  |
| 2. objekt( <i>malá,modrá,kvádr</i> )      | záporný |
| 3. objekt( <i>velká,červená,koule</i> )   | kladný  |
| 4. objekt( <i>velká,červená,krychle</i> ) | záporný |
| 5. objekt( <i>malá,modrá,koule</i> )      | kladný  |
| 6. objekt( <i>malá,bilá,kvádr</i> )       | záporný |

## Algoritmus Specific to general search

Metoda učení prohledáváním prostoru, která pracuje **od specifického k obecnějšímu**.

1. Vytvořte dvě prázdné množiny **S** (Specific) a **N** (Negative).
2. Uložte do množiny **S** první **kladný příklad**. Pro každý další příklad **p** z trénovací množiny:
  - a. je-li **p** **kladným příkladem**, pak pro každý pojem **s**  $\in S$ 
    - i. jestliže pojem **s** nelze unifikovat (**s** je obecnější než **p** a **p** spadá do množiny prvků, které lze pomocí **s** sestavit) s příkladem **p**, pak jej nahraťte jeho **nejvíce specifickým zobecněním** (tak aby už vyjádřit pomocí **s** šel), které lze unifikovat s příkladem **p**,
    - ii. odstraňte z **S** všechny pojmy **s**, které jsou **více obecné** než jiné pojmy v **S**,

- iii. odstraňte z **S** všechny pojmy **s**, které lze **unifikovat** s některým pojmem v **N**.
- b. je-li **p** **záporným příkladem**, pak
  - i. odstraňte z **S** všechny pojmy **s**, které lze unifikovat (lze pomocí nich vyjádřit záporný prvek **p**) s příkladem **p**,
  - ii. přidejte příklad **p** do **N**.

Jinak řečeno s **pozitivními příklady** se výsledné řešení stává **obecnější** - přidávají se **možnosti správných řešení**. Negativní příklady naopak množinu možných řešení redukují (odstraňují) ta řešení, která by správně identifikovala i špatný objekt). Průběh algoritmu na příkladu s míčem vypadá následovně, výsledkem je, že míč je **objekt(X,Y,koule)**:

<b>S = { }</b>	<b>N = { }</b>
1. <b>p = objekt(malá,červená,koule)</b>	<b>S = { objekt(malá,červená,koule) }</b>
2. <b>p = objekt(modrá,kvádr)</b>	<b>N = { objekt (modrá,kvádr) }</b>
3. <b>p = objekt(velká,červená,koule)</b>	<b>S = { objekt(X,červená,koule) }</b>
4. <b>p = objekt(velká,červená,krychle)</b>	<b>N = { objekt(modrá,kvádr),</b> <b>          objekt(velká,červená,krychle) }</b>
5. <b>p = objekt(modrá,koule)</b>	<b>S = { objekt (X,Y,koule) }</b>
6. <b>p = objekt(bilá,kvádr)</b>	<b>N = { objekt(modrá,kvádr),</b> <b>          objekt(velká,červená,krychle),</b> <b>          objekt(bilá,kvádr) }</b>

## Algoritmus General to Specific Search

Metoda učení prohledáváním prostoru, která pracuje **od obecného ke specifickému**.

1. Vytvořte dvě prázdné množiny **G** (General) a **P** (Positive) a uložte do **G** **nejobecnější** pojem (všechny parametry jsou vyjádřené proměnnou).
2. Pro každý další příklad **p** z trénovací množiny:
  - a. je-li **p** **záporným příkladem**, pak pro každý pojem **g ∈ G**:
    - i. jestliže pojem **g** lze unifikovat s příkladem **p**, pak jej nahraďte jeho **nejobecnější specializací**, kterou **nelze unifikovat** s příkladem **p**,
    - ii. odstraňte z **G** všechny pojmy **g**, které jsou **více specializované** než jiné pojmy v **G**,
    - iii. odstraňte z **G** všechny pojmy **g**, které nelze unifikovat s některým pojemem v **P**.
  - b. je-li **p** **kladným příkladem**, pak:
    - i. odstraňte z **G** všechny pojmy **g**, které nelze unifikovat s příkladem **p**,
    - ii. přidejte příklad **p** do **P**.

Jinak řečeno se **zápornými příklady** se z obecného řešení **stává konkrétnější**, protože je řešení **konkretizováno**, aby nevyhovovalo **záporným** řešením. **Kladné příklady** pak **odstraňují** ta řešení, pomocí kterých je **není možné vyjádřit** (unifikovat). Průběh algoritmu je na obrázku, výsledek je **objekt(X,Y,koule)**:

	$G = \{\text{objekt}(X,Y,Z)\}, P = \{\}$
1. $p = \text{objekt}(\text{malá},\text{červená},\text{koule})$	$P = \{\text{objekt}(\text{malá},\text{červená},\text{koule})\}$
2. $p = \text{objekt}(\text{malá},\text{modrá},\text{kvádr})$	$G = \{\text{objekt}(\text{velká},Y,Z),$ $\quad \text{objekt}(X,\text{červená},Z), \text{objekt}(X,\text{bilá},Z),$ $\quad \text{objekt}(X,Y,\text{koule}), \text{objekt}(X,Y,\text{krychle})\}$
	$\Rightarrow G = \{\text{objekt}(X,\text{červená},Z), \text{objekt}(X,Y,\text{koule})\}$
3. $p = \text{objekt}(\text{velká},\text{červená},\text{koule})$	$P = \{\text{objekt}(\text{malá},\text{červená},\text{koule}),$ $\quad \text{objekt}(\text{velká},\text{červená},\text{koule})\}$
4. $p = \text{objekt}(\text{velká},\text{červená},\text{krychle})$	$G = \{\text{objekt}(\text{malá},\text{červená},Z),$ $\quad \text{objekt}(X,\text{červená},\text{kvádr}),$ $\quad \text{objekt}(X,\text{červená},\text{koule}),$ $\quad \text{objekt}(X,Y,\text{koule})\}$
	$\Rightarrow G = \{\text{objekt}(X,Y,\text{koule})\}$
5. $p = \text{objekt}(\text{malá},\text{modrá},\text{koule})$	$P = \{\text{objekt}(\text{malá},\text{červená},\text{koule}),$ $\quad \text{objekt}(\text{velká},\text{červená},\text{koule}),$ $\quad \text{objekt}(\text{malá},\text{modrá},\text{koule})\}$
6. $p = \text{objekt}(\text{malá},\text{bilá},\text{kvádr})$	$G = \{\text{objekt}(X,Y,\text{koule})\}$

## Candidate eliminations

Spojuje postupy předchozích algoritmů.

Vytvořte dvě prázdné množiny **G** (General) a **S** (Specific) a uložte do **G** nejobecnější pojem.

Uložte do množiny **S** první **kladný příklad**.

Pro každý další příklad **p** z trénovací množiny:

je-li **p kladným příkladem** pak:

odstraňte z **G** všechny pojmy  $g \in G$ , které nelze unifikovat s příkladem

**p**,

pro každý pojem  $s \in S$ :

jestliže pojem **s nelze unifikovat** s příkladem **p**, pak jej nahraďte jeho **nejvíce specifickým zobecněním**, které lze unifikovat s příkladem **p**,

odstraňte z **S** všechny pojmy **s**, které jsou **více obecné** než jiné pojmy v **S**,

odstraňte z **S** všechny pojmy **s**, které nejsou **více specifické**, než některé pojmy v **G**.

je-li **p záporným příkladem**:

odstraňte z **S** všechny pojmy **s**, které lze unifikovat s příkladem **p**,

pro každý pojem  $g \in G$ :

jestliže pojem **g** lze unifikovat s příkladem **p**, pak jej nahraďte jeho **nejvíce zobecněnou specializací**, kterou **nelze unifikovat** s příkladem **p**,

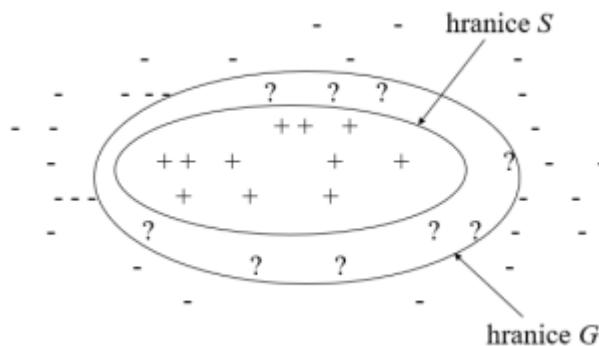
odstraňte z **G** všechny pojmy **g**, které jsou **více specifické** než jiné pojmy v **G**,

odstraňte z **G** všechny pojmy **g**, které nejsou více obecné než některé pojmy v **S**.

Jestliže **G = S** a obě množiny přitom **obsahují jediný pojem**, pak je výsledkem učení právě tento pojem. Příklad tohoto algoritmu, výsledek je **objekt(X,Y,koule)**:

	$G = \{\text{objekt}(X,Y,Z)\}, S = \{\}$
1. $p = \text{objekt(malá,červená,koule)}$	$S = \{\text{objekt}(malá,červená,koule)\}$
2. $p = \text{objekt(malá,modrá,kvádr)}$	$G = \{\text{objekt(velká,Y,Z)}, \text{objekt}(X,\text{červená},Z),$ $\text{objekt}(X,\text{bílá},Z), \text{objekt}(X,Y,\text{koule}),$ $\text{objekt}(X,Y,\text{krychle})\}$
	$\Rightarrow G = \{\text{objekt}(X,\text{červená},Z), \text{objekt}(X,Y,\text{koule})\}$
3. $p = \text{objekt(velká,červená,koule)}$	$S = \{\text{objekt}(X,\text{červená},\text{koule})\}$
4. $p = \text{objekt(velká,červená,krychle)}$	$G = \{\text{objekt}(malá,\text{červená},Z),$ $\text{objekt}(X,\text{červená},\text{kvádr}),$ $\text{objekt}(X,\text{červená},\text{koule}),$ $\text{objekt}(X,Y,\text{koule})\}$
	$\Rightarrow G = \{\text{objekt}(X,Y,\text{koule})\}$
5. $p = \text{objekt(malá,modrá,koule)}$	$S = \{\text{objekt}(X,Y,\text{koule})\}$
6. $p = \text{objekt(malá,bílá,kvádr)}$	$G = \{\text{objekt}(X,Y,\text{koule})\}$

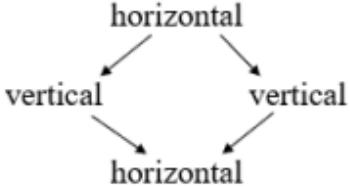
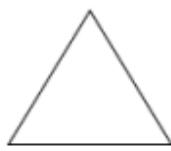
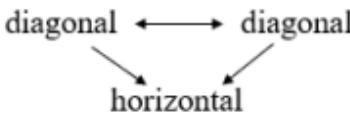
Význam a vztah množin **S** a **G** je na obrázku níže. Každý pojem, který **by byl obecnější** než nějaký pojem v **G**, **by zahrnoval některé negativní** příklady, každý pojem, který **by byl specifickější** než nějaký pojem v **S**, **by vylučoval některé pozitivní** příklady. Výsek s otazníky značí objekty, které nejsou v trénovací množině, ale na základě výsledků algoritmů **prohledávání prostoru verzí** je poté lze identifikovat, např.: **míč je kulatý objekt, na jehož barvě, ani velikosti nezáleží**.



## Rozpoznávání a klasifikace obrazů (Pattern Recognition and Classification):

Pro rozpoznávání obrazů existují různé algoritmy, které pracují s jedním z následujících popisů:

- **Příznakový** (popis vektory číselných příznaků): využívá **statických informací o příznacích** obrazů obsažených v množině trénovacích dat. Jde o tzv. **statické příznakové rozpoznávání**.
- **Strukturální/syntaktický** (popis strukturálními prvky, tzv. primitivy): využívá **vztahy mezi příznaky** obrazů rozpoznávaných objektů.

<u>příznakový popis</u>	<u>objekt</u>	<u>strukturální popis</u>
počet segmentů	4	
počet horizontálních segmentů	2	
počet vertikálních segmentů	2	
počet diagonálních segmentů	0	
vektor příznaků	(4,2,2,0)	
		
počet segmentů	3	
počet horizontálních segmentů	1	
počet vertikálních segmentů	0	
počet diagonálních segmentů	2	
vektor příznaků	(3,1,0,2)	
		

Zásadním předpokladem úspěšného rozpoznávání je **výběr relevantních příznaků**, resp. **primitiv**.

## Příznakové rozpoznávání

U příznakového rozpoznávání pracujeme s:

- n-rozměrnými **číselnými vektory** příznaků, které popisují rozpoznávané objekty,
- **množinou tříd**, do kterých rozpoznávané objekty chceme zařadit,
- **trénovací množinu**, která je tvořena **dvojicemi** n-rozměrný číselný **vektor** a k němu **třídou**, do které spadá.

Cílem je poté zařadit libovolný vektor příznaků do jedné z tříd - **klasifikovat** jej.

Metody **příznakového rozpoznávání/klasifikace** vycházejí z předpokladu, že obrazy objektů nebo jevů **stejných tříd** tvoří v **n rozměrném** obrazovém prostoru **shluky**. Tyto shluky mohou být od sebe **zřetelně oddělitelné (separable)**, nebo se mohou **prolínat** a pak jsou **neoddělitelné (inseparable)**. Systémy, které se na trénovací množině naučí obrazy rozpoznávat a poté **klasifikují nové** obrazy, se nazývají **klasifikátory**.

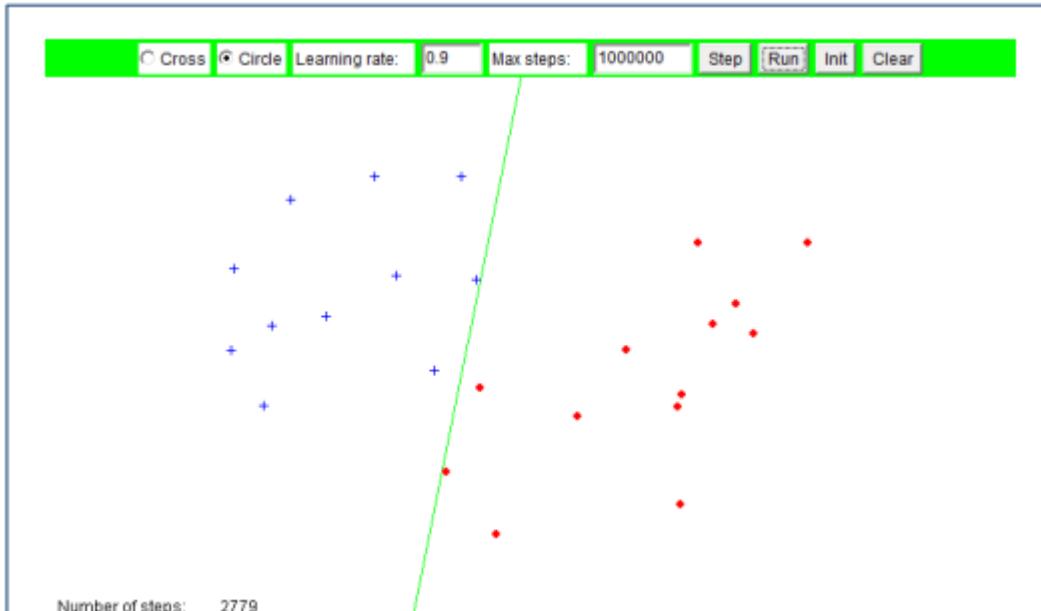
## Dichotomie

Jedná se o **klasifikaci** do **dvou tříd**. V praxi je **dichotomie** poměrně **častá**, obecně jde klasifikovat do **n** tříd, což spočívá v nalezení **křivek/ploch/k-rozměrných útvarů**, které obrazy jednotlivých tříd od sebe **oddělují**. Toho se zajišťuje pomocí **diskriminačních** funkcí (pro každou třídu jedna), které obraz ohodnocují. **Obraz** je pak **umístěn/klasifikován** do třídy, jejíž **diskriminační funkce** má **největší** hodnotu (je nutné pro každý obraz vypočítat hodnoty všech diskriminačních funkcí). U dichotomie stačí jedna diskriminační funkce (respektive **2**, které od sebe **odečítáme**) a obrazy objektů klasifikujeme na základě **znaménka** do dvou tříd (kladná a záporná).

Algoritmus určení **lineárního klasifikátoru** pro dichotomii (diskriminační funkce):

1. Vynulujte vektor vah.
2. Nastavte indikátor změny **modif = false**.
3. Pro každý **vektor** z trénovací množiny, který **není správně klasifikován** (diskriminační funkce vrací jiné znaménko, než je v trénovacím příkladě) **upravte vektor vah** (kterým je vektor klasifikován) a nastavte **modif = true**.
4. Pokud došlo k úpravě vektoru vah (modif = true), tak se vraťte na bod 2.

Výsledkem bude rozdělení prostoru dle následujícího obrázku pro 2D.



Jeden klasifikátor klasifikující do **R tříd** může být nahrazen **R klasifikátory** pro dichotomii **naučených** na klasifikaci patří/nepatří do třídy  $r$ ,  $r \in \langle 1, R \rangle$ .

### Etalony

Jedná se o **těžiště shluků obrazů** jednotlivých tříd. Obraz je klasifikován do třídy, jejímuž **etalonu má nejbližše** (nejmenší vzdálenost v prostoru). Tato klasifikace však opět pracuje na principu klasifikace podle diskriminačních funkcí.

### Rozpoznávání obrazů reprezentovaných neoddělitelnými třídami obrazů

V případě **neoddělitelných tříd** obrazů již nelze rozhodnout, že **obraz x** patří do třídy  $r$ , ale lze konstatovat, že obraz  $x$  patří do třídy  $r$  s pravděpodobností  $P(r|x)$ . Úkolem trénovacích algoritmů je nalézt takový klasifikátor, který bude klasifikovat obrazy s co největší pravděpodobností, respektive minimalizovat ztrátu, která vzniká klasifikací do nesprávné třídy. Využívá se k tomu ztrátová matice, ztráta pro **správnou klasifikaci obrazu je 0** a pro **chybnou 1**. Na základě ztrát se matice zjednoduší na obrázek vpravo.

$$\lambda = \begin{bmatrix} \lambda(1,1) & \lambda(1,2) & \dots & \lambda(1,R) \\ \lambda(2,1) & \lambda(2,2) & \dots & \lambda(2,R) \\ \vdots & \vdots & \vdots & \vdots \\ \lambda(R,1) & \lambda(R,2) & \dots & \lambda(R,R) \end{bmatrix} \quad \lambda = \begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & \dots & 0 \end{bmatrix}$$

## Strukturální rozpoznávání

Jedná se o rozpoznávání obrazů na základě jeho popisu pomocí **primitiv**. Existují dva základní přístupy ke strukturálnímu rozpoznávání obrazů:

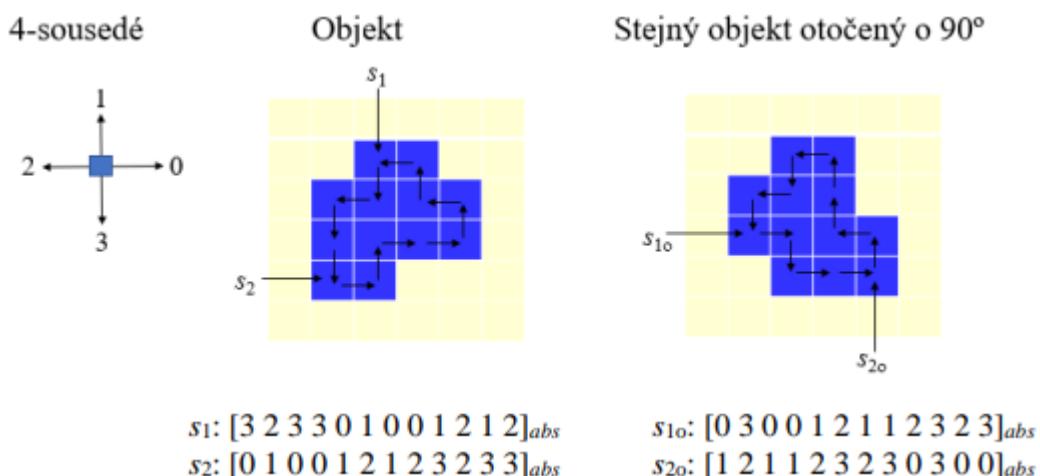
- rozpoznávání obrazů pomocí **gramatik** - **syntaktické rozpoznávání** (syntactic recognition)
- rozpoznávání obrazů **porovnáním se vzory** uloženými v databázi (template matching) - **neučili jsme se**.

Syntaktické rozpoznávání klasifikuje obrazy do **R** tříd pomocí **syntaktické analýzy**. Obrazy, tj. **řetězcové popisy objektů** nebo jevů, jsou přitom chápány jako **slova** a množina primitiv jako množina **terminálních symbolů**. Pokud pak gramatika generuje **všechna slova/obrazy**, která reprezentují obrazy třídy **r**, a negeneruje **žádné slovo**, která reprezentuje obraz **jiné třídy** (jinak řečeno všechny gramatiky generují vzájemně **různá slova**), lze klasifikaci převést na problém určení gramatiky, která jako jediná ze všech **R** gramatik generuje rozpoznávané slovo/obraz.

Na **úspěšnost** strukturálního/syntaktického rozpoznávání má velký vliv **výběr primitiv** (malá množina primitiv má malou vyjadřovací sílu, velká množina může být nezvládnutelná při učení).

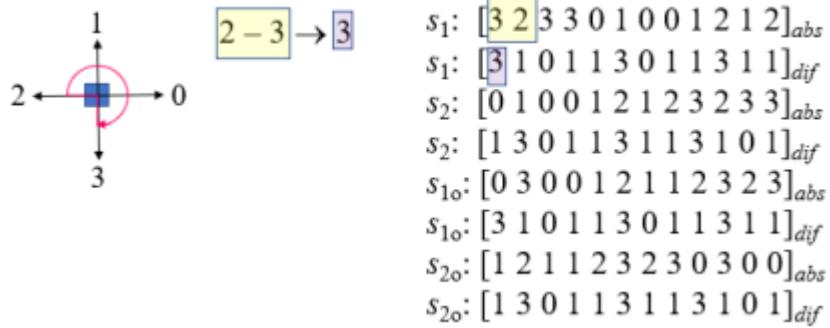
### Freemanův řetězcový kód

Používá se ke strukturálnímu popisu obrysu objektu (obrazu). Za primitiva popisující obrys objektu považujeme **směry sousedů**. **Problémy** při popisu objektů tímto způsobem je jejich **natočení** a **startovací pozice** popisu (popis je invariantní vůči posuvu), což jej činí prakticky nepoužitelným.



Problémy lze eliminovat následovně:

- **natočení**: Řešením je popis pomocí relativního (diferenciálního) kódu, který místo primitiv používá **rozdílů/diferencí natočení** mezi dvěma sousedními primitivy (následující – aktuální). Pro určení těchto rozdílů se postupuje v **opačném směru** (tj. po směru hodinových ručiček, overflow  $2-3 = 3$ ,  $1-3 = 2$ , ... pokud je **první číslo menší**, je nutné k výsledku **přičít 4**). Na obrázku lze po provedení postupu vidět, že **diferenční kódy** pro  $s_1$  a  $s_{1o}$  jsou stejně jako jsou stejné diferenční kódy  $s_2$  a  $s_{2o}$ .



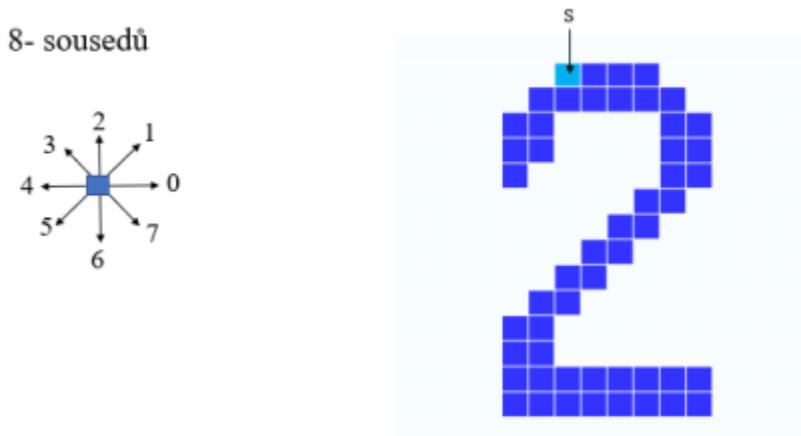
- **startovací pozice:** řešení spočívá v tom, že řetězec **relativního** (diferenciálního) **kódu** rotujeme tak, aby řetězec číslic po převodu na číslo bylo **číslo největší** (tj. zleva obsahovalo největší číslice). Tomuto číslu se říká **číslo tvaru - shape number**.

$s_1 = s_{1o} = [3 \text{ } 1 \text{ } 0 \text{ } 1 \text{ } 1 \text{ } 3 \text{ } 0 \text{ } 1 \text{ } 1 \text{ } 3 \text{ } 1 \text{ } 1]_{dif}$   $s_2 = s_{2o} = [1 \text{ } 3 \text{ } 0 \text{ } 1 \text{ } 1 \text{ } 3 \text{ } 1 \text{ } 1 \text{ } 3 \text{ } 1 \text{ } 0 \text{ } 1]_{dif}$

*shape number:*  $[3 \text{ } 1 \text{ } 1 \text{ } 3 \text{ } 1 \text{ } 0 \text{ } 1 \text{ } 1 \text{ } 3 \text{ } 0 \text{ } 1 \text{ } 1]$ .

**Číslo tvaru** (shape number) je **invariantní** vůči **posunutí, natočení** objektu (pro 4 sousedy pouze **po 90°**) a **startovacímu bodu** popisu, je však **závislé na velikosti objektu**.

Freemanův řetězcový kód existuje i pro 8 sousedů, viz obrázek:



Řetězcový kód pro 8 sousedů:

$[5566121000066655555666000000024444432111112233444]_{abs} =$   
 $[010317700060070000100200000220000777000010101001]_{dif}$

Číslo tvaru:

$[7770000010101001010317700060070000100200000220000]$

Číslo tvaru  $[3 \text{ } 1 \text{ } 1 \text{ } 3 \text{ } 1 \text{ } 0 \text{ } 1 \text{ } 1 \text{ } 3 \text{ } 0 \text{ } 1 \text{ } 1]$  popisující objekt výše by generovala například **gramatika** s těmito přepisovacími pravidly:

- $S \rightarrow 3N$ ,
- $N \rightarrow 0N$ ,
- $N \rightarrow 1N$ ,
- $N \rightarrow 3N$ ,

- $N \rightarrow 1$

Tato gramatika by však **generovala i další řetězce**, které by daný objekt **nepopisovaly**, a proto k rozpoznávání by byla zřejmě **nepoužitelná**. Nalezení **relevantních** přepisovacích pravidel (tj. určení správné gramatiky) je mnohem **náročnější** než učení příznakových klasifikátorů nebo trénování neuronových sítí a dnes se tak moc nepoužívá.

## Učení bez učitele

Učení bez učitele spočívá v **hledání podobností** mezi příklady **trénovací množiny** a v zařazování příkladů s **podobnými charakteristikami do skupin**. Umělý systém přitom **nedostává žádnou informaci** o správnosti klasifikace a jedinou informací, kterou má, resp. může mít je **počet skupin**, do kterých má příklady z trénovací množiny zařazovat/klasifikovat.

Příklady trénovací množiny jsou téměř vždy představované **číselnými vektory** příznaků klasifikovaných objektů či dějů. Metody **učení bez učitele** jsou pak založeny na předpokladu, že tyto vektory, resp. body, které specifikují v příslušném **n rozměrném** obrazovém prostoru **shluky**. Tyto **shluky** mohou představovat velmi **rozmanité n rozměrné útvary**, například v několika souřadnicích může jít o zcela kompaktní shluky, zatímco v jiných souřadnicích mohou být značně rozptýlené.

### On-Policy

TODO

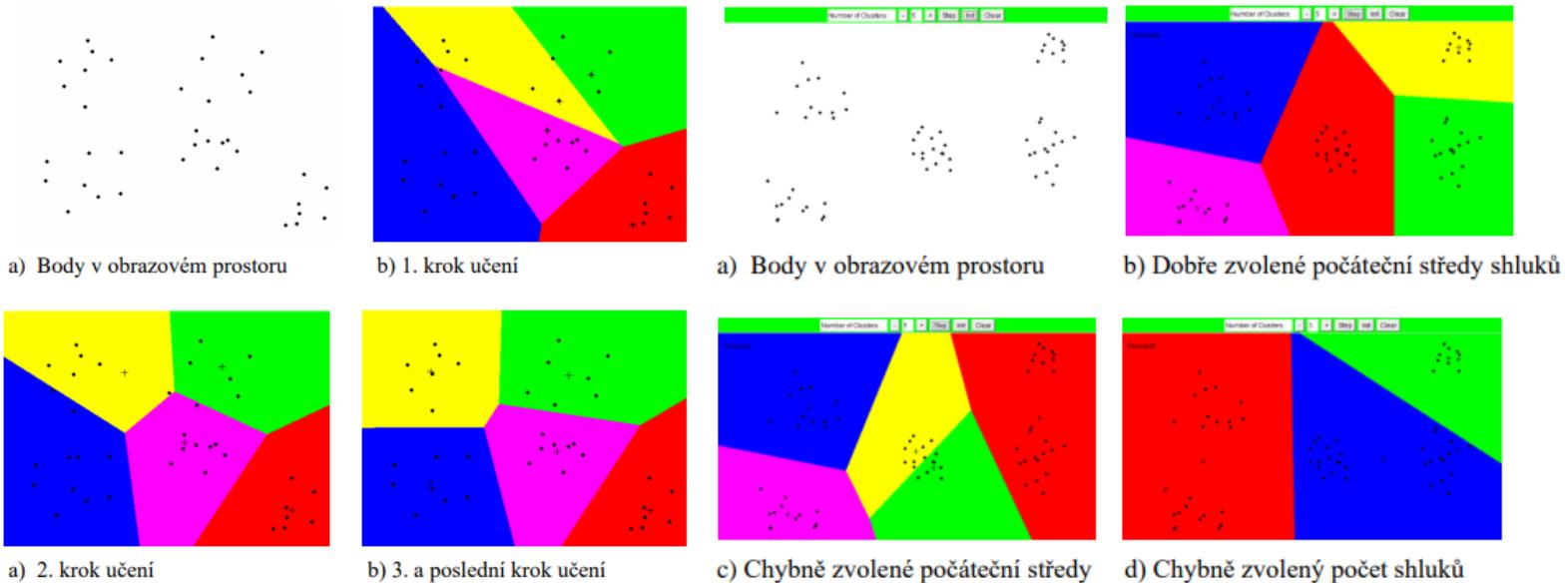
### Off-Policy

TODO

### k-means clustering

Algoritmus **klasifikuje** příklady (číselné vektory) z trénovací množiny do předem daného počtu **k** shluků. Algoritmus **zařazuje** vstupní vektor do toho **shluku**, k jehož **středu/těžišti má nejkratší vzdálenost**. Vstupem algoritmu je počet shluků **k** (musí být menší než počet vektorů) a průběh učení je následující:

1. Náhodně určí **k** rozdílných vektorů (často je vybere z trénovací množiny), které považuje za **středy (těžiště)** shluků.
2. Zařadí všechny vektory trénovací množiny do **příslušných shluků**, dle použité **metriky** (nejčastěji se používá asi **Euklidovská** vzdálenost - délka úsečku mezi body, ale lze použít např. **Hammingovu** vzdálenost - je nejmenší počet pozic, na kterých se řetězce stejně délky daného kódu liší)
3. **Přeypočítá středy** všech shluků.
4. Pokud se pozice ani jednoho středu nezměnila (tj. vektory byly opakovaně zařazeny do stejných shluků), algoritmus končí. Jinak pokračuje bodem 2.



## Posilované učení

Posilované učení se od učení s učitelem odlišuje tím, že systém **ohodnocuje** své akce na základě **penalizací** či **odměn** získaných v **koncových stavech** a na základě **svých hodnocení** stavů/akcí získaných **vlastními** předchozími **zkušenostmi**.

### Policy-only learning

Princip tohoto algoritmu je založený na tom, že z **každého uzlu** grafu vedou **maximálně 2** (koncové 1) **cesty** (hrany). každý uzel obsahuje **schránku s kameny** (analogie pravděpodobnosti) dvojí barvy - **bílé** a **černé** (pokud je počet bílých kamenů větší, je pravděpodobnější výběr cesty vlevo, naopak je to u většího počtu

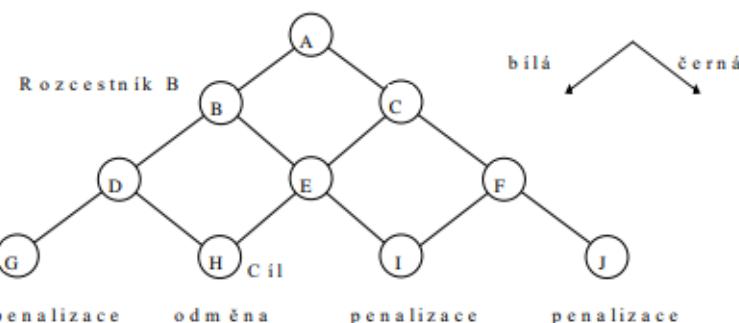
$$P_{vlevo} = \frac{N_{bílé}}{N_{bílé} + N_{černé}}, \quad P_{vp право} = \frac{N_{černé}}{N_{bílé} + N_{černé}}$$

černých kamenů).

Na začátku učení obsahuje schránka každého rozcestníku **stejný a dostatečný počet** černých a bílých kamenů. Učení pak probíhá na tomto principu, že se provádí náhodné procházky (náhodnost je dána pravděpodobností výběru cesty v uzlech).

Pokud výsledná cesta:

- **skončí v cílovém stavu** (na obrázku uzel H), je do schránek na této cestě **přidán** kámen odpovídající barvy podle výběru směru - **odměna**.  
**Pravděpodobnost** výběru této cesty **se zvyšuje**.
- **skončí mimo cílový stav** (na obrázku uzly G, I, J), je ze schránek **odebrán** kámen odpovídající barvy podle výběru směru - **penalizace**.  
**Pravděpodobnost** výběru této cesty **se snižuje**.



Po ukončení učení vybírá umělý systém cestu **pouze porovnáním počtu kamenů**, tj. je-li ve schránce rozcestníku **více bílých kamenů**, pokračuje **levou cestou jinak** pokračuje cestou **pravou**.

Počet cest z uzlu lze jednoduše **rozšířit** přidáním více kamenů **různých barev** (rozdělení počáteční pravděpodobnosti mezi více cest).

Metoda může mít 2 zásadní **problémy**:

- V případě obecného grafu se může **vracet do již dříve vyšetřovaných uzelů** (například v bludišti), může se teoreticky při učení zacyklit.
- Všechny akce provedené na jedné **cestě se hodnotí stejnými vahami**, přestože správná ohodnocení mohou být značně rozdílná (Např. u řešení, které končí v cílovém stavu, je vložen do každé schránky kámen podporující tuto cestu, i když tato cesta nemusí být výhodná - zbytečně dlouhá atd., lepší by bylo, kdyby odměna/penalizace měla vždy stejně velkou hodnotu, která by byla rozdělena mezi uzly na cestě).

### Metoda TD learning (On-Policy)

Metoda je založená na **náhodných procházkách**, během kterých ohodnocuje jednotlivé stavy  $s$ , **stav  $s$  je vždy bezprostředním předchůdcem stavu  $s'$** . Dochází tedy k **šíření hodnot** ze stavů s **odměnou/penalizací**. Šíření ohodnocení je dáno následujícím vzorcem:

$$\text{ohodnocení}(s) = \text{ohodnocení}(s) + \alpha * (r(s') + \gamma * \text{ohodnocení}(s') - \text{ohodnocení}(s)),$$

kde:

- $\alpha$ : koeficient **učení**,
- $\gamma$ : koeficient určující **vliv** ohodnocení stavu  $s'$  na ohodnocení předcházejícího stavu  $s$ ,
- $r(s')$ : odměnu (reward) za dosažení stavu  $s'$ ,
- **ohodnocení(s)**: ohodnocení (utility) stavu  $s$  při použití dané **strategie pohybu**.

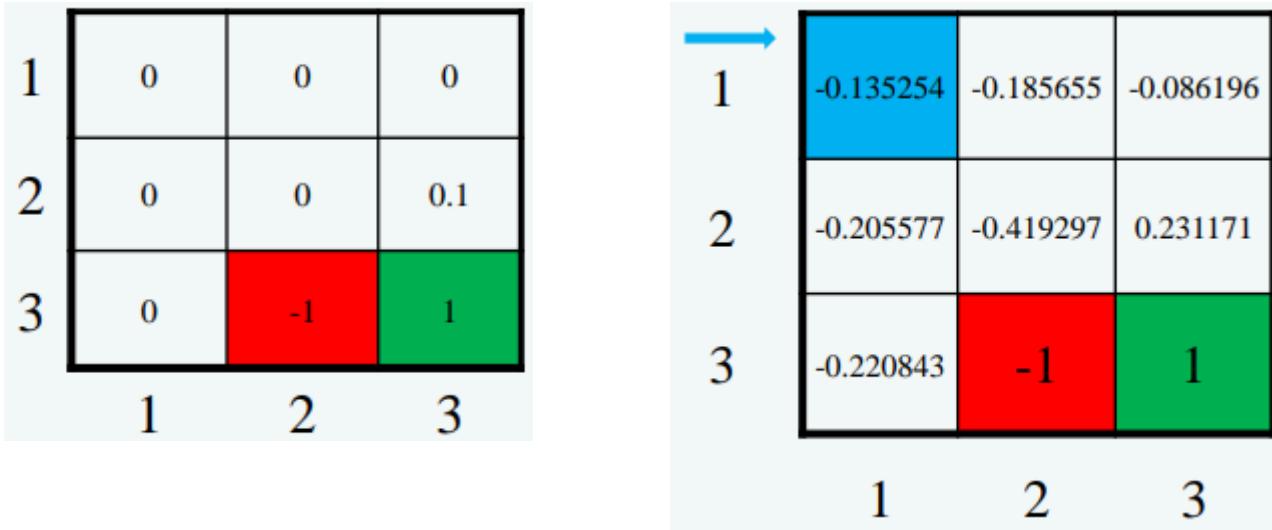
Pro přecházení mezi stavy lze použít různé strategie:

- **Pravděpodobnosti** přechodů mezi sousedními stavy jsou **stejné** a do sousedních stavů přechází **zcela náhodně (random policy)**.
- Pravděpodobnosti jsou **různé** (mohou být například dány **aktuálními hodnotami sousedních stavů**, tj. čím je **vyšší ohodnocení** sousedního stavu, tím je **vyšší pravděpodobnost** přechodu do tohoto stavu).
- Pravděpodobnost jednoho přechodu je **jedničková** a pravděpodobnosti ostatních **nulové**, což je **extrémní případ** předchozího stavu (**greedy policy**).
- **Kombinace předchozích** případů, kdy s pravděpodobností danou parametrem  $\epsilon$  se použije random policy a s pravděpodobností ( $1 - \epsilon$ ) se použije **greedy policy** ( $\epsilon$ -greedy policy).

Princip učení:

- Zvolte hodnoty koeficientů  $\alpha$  a  $\gamma$  ( $0 < \alpha \leq 1$ ;  $0 < \gamma \leq 1$ ) a **vynulujte** ohodnocení všech stavů. Dále **vynulujte počítadlo** procházek  $p$  a nastavte jejich **maximální** počet na **pmax**. Nastavte **start** → **s**.
- Generujte nový stav **s'** s použitím některé strategie.
- Přepracujte novou hodnotu stavu s pomocí vztahu pomocí vzorce výše.
- Je-li stav **s'** cílovým stavem, pak inkrementuj počet procházek a **start** → **s**, jinak **s' → s**.
- Je-li  $p < p_{\text{max}}$ , pak se vraťte na bod 2.

Příklad, vlevo po 1. kroku učení, vpravo po naučení:



**Po naučení** se již přechází z libovolného necílového stavu do jeho sousedního stavu, který má **nejvyšší hodnotu** (ale musí být **stejnou nebo lepší** než hodnota aktuálního stavu - to může způsobit problém uváznutí, z nějaké stavu nemusí být dosažitelný žádný jiný stav, lze řešit snížením koeficientu učení  $\alpha$ )

### Q learning (Off policy)

Metoda je podobná metodě **TD learning**. Tato metoda **místo ohodnocení stavů ohodnocí akce v těchto stavech** a k tomuto ohodnocení používá vztah:

$$Q(s, a) = Q(s, a) + \alpha * (r(s') + \gamma * \max Q(s', a') - Q(s, a)),$$

kde:

- $\alpha$ : koeficient **učení**,
- $\gamma$ : koeficient určující **vliv** ohodnocení stavu  $s'$  na ohodnocení předcházejícího stavu  $s$ ,
- $r(s')$ : odměnu (reward) za dosažení stavu  $s$ ,
- $Q(s, a)$ : označuje ohodnocení **akce a** provedené **ve stavu s**.
- $\max Q(s', a')$ : označuje **maximální** hodnotu z **ohodnocení** všech **akcí a'**, které je možné provést ve **stavu s'**.

Metoda je aktivní metodou, tj. bez předem **dané strategie** výběru stavu  $s'$ . Princip učení:

- Zvolte hodnoty koeficientů  $\alpha$  a  $\gamma$  ( $0 < \alpha \leq 1$ ;  $0 < \gamma \leq 1$ ) a **vynulujte** ohodnocení  $Q(s, a)$  všech **akcí a** ve všech **stavech s**. Dále vynulujte **počítadlo**

**procházeck**  $p = 0$  a nastavte jejich maximální počet **pmax**. Nastavte **start** → **s**.

2. Vyberte akci **a**, která povede k přechodu ze stavu **s** do stavu **s'**.
3. Vypočítejte novou hodnotu vybrané **akce a** ve **stavu s** pomocí vztahu výše.
4. Je-li **stav s'** cílovým stavem, pak inkrementuj počítadlo procházek a nastav **start** → **s**, jinak nastav **s' → s**.
5. Je-li  $p < \text{pmax}$ , pak se **vrat' te** na bod 2.

Ohodnocení akcí může vypadat následovně:

	1	0.12	0.22	0.41	0.18	
	0.12		0.36		0.59	
2	0.25		0.27		0.20	
	0.31	-0.1	0.45	-0.22		
	-0.26		-0.57		0.77	
3	0					
	1	-1	-1	1	3	

Po naučení se přechází již na základě získaného ohodnocení akcí (v tomto případě přechodů)

### SARSA (On-Policy)

Metoda sarsa je přístup, který také ohodnocuje akce, ale k jejich výběru používá nějakou **strategii  $\pi$** , tzn. místo hledání maxima z možných následujících akcí se používá přímo akce  $Q(s', a')$  vybraná také **strategií  $\pi$** . Jde tedy o postup **s, a, r', s', a'** (SARSA). Hlavní funkce pro aktualizaci Q-hodnoty závisí na **aktuálním stavu s**, **akci a**, kterou agent zvolí, **odměně r**, kterou agent dostane za volbu této akce **a**, **stavu s'**, do kterého agent vstoupí po provedení akce **a** a nakonec další akci **a'**, kterou agent zvolí ve svém **novém stavu**.

# 28. Principy modelování a simulace systémů (systémy, modely, simulace, algoritmy řízení simulace).

## Systémy

Systém je soubor elementárních částí (prvků systémů), které mají mezi sebou určité vazby. Můžeme je dělit na:

- **Reálné systémy**
- **Nereálné systémy** - Fiktivní, ještě neexistující.
- **Spojité** - Všechny prvky mají **spojité chování**.
- **Diskrétní** - Všechny prvky mají **diskrétní chování**.
- **Kombinované** - Obsahuje **spojité i diskrétní prvky**.
- **Deterministické** - Všechny prvky jsou **deterministické**.
- **Nedeterministické** - Alespoň jeden prvek s **nedeterministickým chováním**.

Formálně se jedná o dvojici  $S = (U, R)$ , kde:

- $U = \{u_1, u_2, \dots, u_n\}$  je univerzum - konečná **množina prvků** systému. Každý prvek je navíc tvořen dvojicí  $(X, Y)$ ,  $u = (X, Y)$ , kde:
  - $X$  je množina všech **vstupních proměnných**.
  - $Y$  je množina všech **výstupních proměnných**.
- $R$  - Množina všech **propojení** (relací), která je podmnožinou **kartézských součinů vstupů a výstupů** jednotlivých prvků.

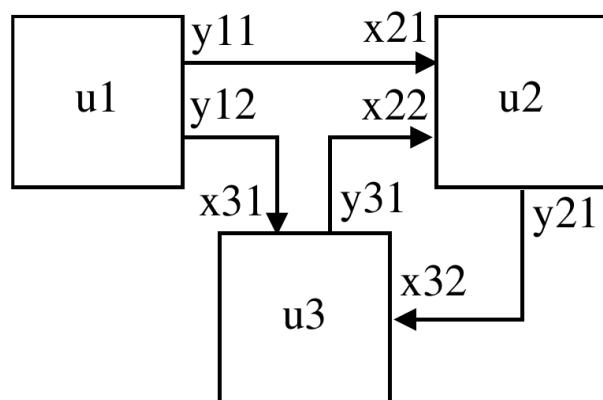
$$R = \bigcup_{i,j=1}^N R_{ij}$$

Propojení prvku  $u_i$  s  $u_j$ :

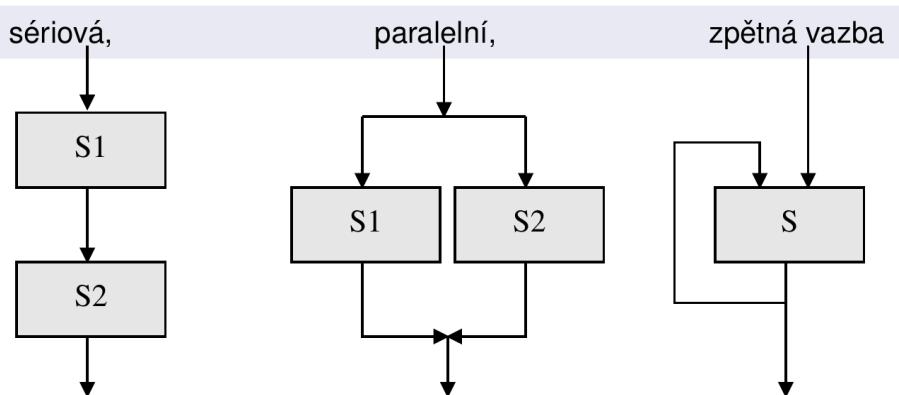
$$R_{ij} \subseteq Y_i \times X_j$$

Příklad formálně definovaného systému může vypadat následovně:

$$\begin{aligned} U &= \{u_1, u_2, u_3\} \\ R &= \{(y_{11}, x_{21}), (y_{12}, x_{31}), (y_{31}, x_{22}), (y_{21}, x_{32})\} \end{aligned}$$

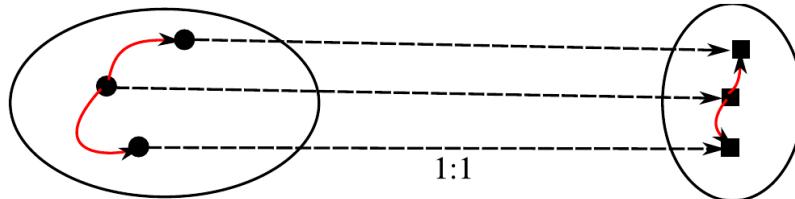


Vazba mezi prvky může být **sériová**, **paralelní** a nebo **zpětná**.

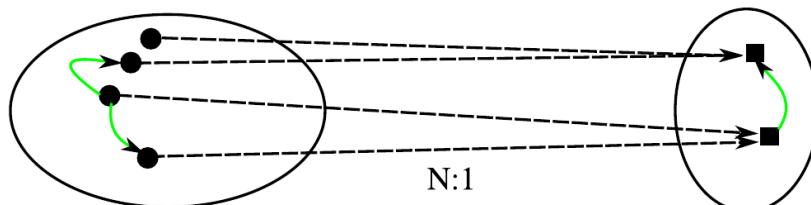


### Vztahy mezi systémy

- **Izomorfní systémy** - Systémy  $\mathbf{S}_1 = (\mathbf{U}_1, \mathbf{R}_1)$  a  $\mathbf{S}_2 = (\mathbf{U}_2, \mathbf{R}_2)$  jsou izomorfní pokud mezi nimi **existuje bijekce**:
  1. Prvky  $\mathbf{U}_1$  lze vzájemně jednoznačně přiřadit  $\mathbf{U}_2$  (**bijektivní zobrazení - 1:1**).
  2. Prvky  $\mathbf{R}_1$  lze **bijektivně** zobrazit na  $\mathbf{R}_2$  se stejně **orientovanými vztahy** na prvky univerz.



- **Homomorfní systémy** - Je základním **principem** modelování, systémy jsou homomorfní pokud mezi nimi **existuje surjekce**:
  1. Prvkům  $\mathbf{U}_1$  je možné přiřadit **jednoznačně** prvky  $\mathbf{U}_2$  (**surjektivní zobrazení - N:1**).
  2. Prvkům  $\mathbf{R}_1$  je možné jednoznačně přiřadit prvky  $\mathbf{R}_2$  se **stejně orientovanými vztahy** s univerzity.



### Chování systémů

Každému časovému průběhu **vstupních proměnných** přiřazuje časový průběh **výstupních proměnných**. Je dán vzájemnými interakcemi mezi prvky. Jinak

řečeno jedná se **způsob, jakým systém převádí vstupy na výstupy**. Jedná se o zobrazení:

$$\chi : [\sigma_i(S)]^T \rightarrow [\sigma_o(S)]^T$$

kde:

- $[A]^T$  je množina všech zobrazení  $T$  do množiny  $A$ ,
- $\sigma_i(S)$  je vstupním prostorem systému  $S$ ,
- $\sigma_o(S)$  je výstupním prostorem systému  $S$ .

**Ekvivalence chování systémů** - Pokud **stejné podněty** u obou vyvolají **stejné reakce**, tak mají ekvivalentní chování.

## Modely

Model je **napodobenina** systému jiným systémem. Reprezentuje znalostí, které máme o systému. Klasifikace:

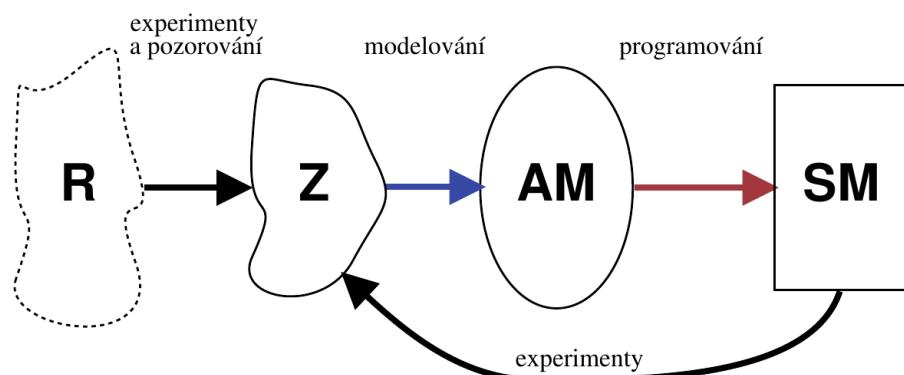
- fyzikální modely, matematické modely - přírodní zákony jsou matematické modely (Ohmův zákon:  $U = R*I$ ).

## Modelování

Vytváření modelů systému na základě znalostí, které o nich máme. Proces modelování je následující:

1. Experimenty a pozorování reality (někdy není možné, modelujeme a simulujeme neexistující věc),
2. Zisk znalostí o modelovaném systému,
3. Tvorba **abstraktního modelu** - formování zjednodušeného popisu systému,
4. Tvorba **simulačního modelu** - zápis abstraktního modelu programem,
5. **Verifikace a validace** - Ověřování správnosti modelu.
6. **Simulace** - experimentování se simulačním modelem,
7. Analýza a interpretace získaných výsledků, což vede zpět na bod 2 a celý proces lze opakovat např. dokud nejsme spokojení s výsledkem.

Realita → Znalosti → Abstraktní model → Simulační model

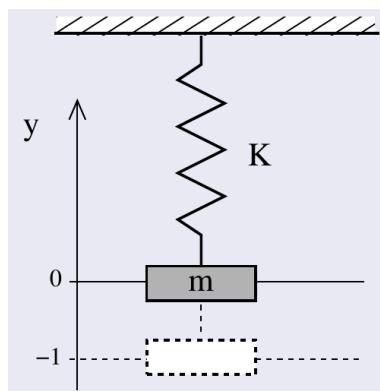


## Verifikace a validace modelů

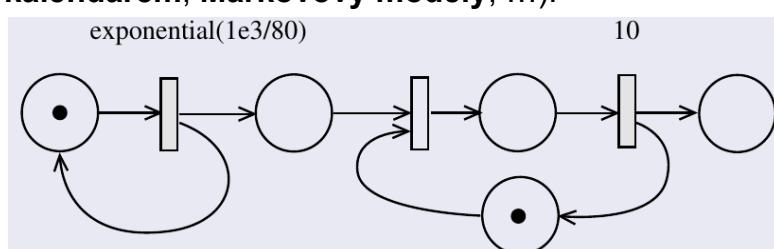
- **Verifikace modelu** - Ověřujeme **korespondenci abstraktního a simulačního modelu**, tj. izomorfní vztah mezi AM a SM. Předchází vlastní etapě simulace.
- **Validace modelu** - Snažíme se dokázat, že **skutečně pracujeme s modelem adekvátním modelovanému systému**. Velmi obtížné a nelze absolutně dokázat. Pokud chování modelu neodpovídá předpokládanému chování, musí model modifikovat, nebo nalézt příčinu odchylky. Je nutné neustále porovnávat informace, které o modelu máme a které získáváme simulací.

## Klasifikace modelů

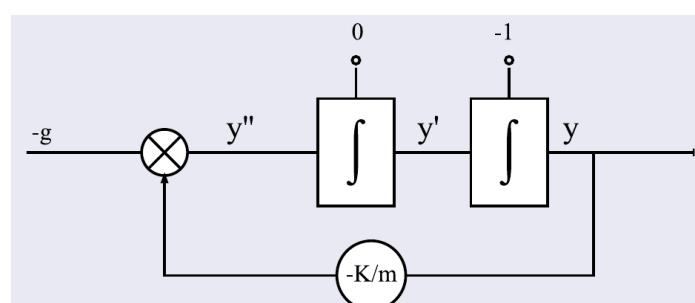
- **Konceptuální** - Jejich komponenty **nebyly** (zatím) **přesně popsány** ve smyslu teorie systémů. Obvykle se používají v **počáteční fázi** modelování pro ujasnění souvislostí a **komunikaci v týmu**. Mají formu **textu** nebo **obrázku**.



- **Deklarativní** - Popis **přechodů** mezi **stavy** systému. Model je definován **stavy a událostmi**, které **způsobí přechod** z jednoho do druhého za jistých podmínek. Vhodné především pro **diskrétní modely**. Obvykle zapouzdřeny do objektů (**konečné automaty**, **petriho sítě**, **událostmi řízené systémy s kalendářem**, **Markovovy modely**, ...).



- **Funkcionální** - Grafy zobrazující **funkce a proměnné**. Buď je uzel grafu proměnná nebo funkce (**systémy hromadné obsluhy**, **bloková schémata** **systémová dynamika**).



- **Popsané rovnicemi (constraint)** - Rovnice (**algebraické, diferenciální, diferenční**), jedná se např. o rovnice kyvadla, RC článku,

Diferenciální rovnice systému dravec-kořist:

$$\begin{aligned}\frac{dx_k}{dt} &= k_1 x_k - k_2 x_k x_d \\ \frac{dx_d}{dt} &= k_2 x_k x_d - k_3 x_d\end{aligned}$$

- **Prostorové (spatial)** - Rozdělují systém na prostorově menší ohrazené podsystémy (**Parciální diferenciální rovnice, celulární automaty, L-systémy, N-body problém**).
- **Multimodely** - Je složen z modelů různého typu, které jsou obvykle heterogenní (spojité + diskrétní, spojité + fuzzy, HLA).

## Simulace

Proces získávání **nových znalostí** o systému pomocí **experimentování s jeho modelem**.

### Výhody

- cena,
- rychlosť,
- bezpečnosť,
- někdy jediný způsob (srážky galaxií)
- atd.

### Problémy

- kontrola validity (nemusí být validní a nepoznáme to),
- náročnosť na vytvárení
- výpočetní náročnosť,
- nepřesnosť numerického měření,
- problémy stability numerických metod
- atd.

### Postup

Opakovane řešení modelu, experimentování s ním. Opakuje se, dokud nezískáme **dostatek informací** o chování nebo dokud **nenajdeme parametry** pro které má systém **žádané chování**. Jeden simulační cyklus vypadá následovně:

- Nastavení hodnot parametrů a počátečního stavu modelu.
- Zadání vstupních podnětů z okolí při simulaci.
- Vyhodnocení výstupních dat (informace o chování systému).