FIT VUT MGM MSZ vypracované okruhy 2016/17

Miloslav Číž

Připomínky a děkovné dopisy můžete zasílat na tastyfish@seznam.cz.

Přeju hodně štěstí u státnic.

CREDITS: Adam Jež poskytl hodně bugfixů.

Obsah

1. Metodika návrhu HW/SW codesign, platformy, programovatelné obvody. HSC

Hardware/software codesign znamená souběžný návrh HW a SW. Motivace tohoto postupu jsou následující:

- Oproti běžnému postupu (nejdříve navrhneme HW, ten syntetizujeme a potom až SW), ušetříme čas, dostaneme se dříve na trh atd.
- Souběžný návrh vede na současný návrh zdola nahoru (používáme už existující komponenty) a shora dolů (HW se dělá "na míru" pro SW), což může pomoci optimalizaci.
- Nezbytnost - složitost návrhu obvodů roste exponenciálně, od jisté úrovně se to už nedá dělat manuálně.
- Automatizace umožňující rychle“experimentovat” např. s různým rozdělením úlohy mezi HW/SW či různými platformami a najít tak lepší řešení.
- Flexibilita – můžeme zkušebně měnit parametry (např platformu, rozdělení HW/SW apod.), rychle syntetizovat systémy a porovnat je. Toto by při ručním návrhu bylo nemyslitelné.

Metodika znamená určitý postup práce, v našem případě postup návrhu systémů v HW/SW codesignu. Návrh zohledňuje určitá kritéria, která jdou často proti sobě a musí se dělat kompromisy. Mezi kritéria patří např.:

- cena výrobku
- cena a doba návrhu
- příkon (v dnešní době čím dál důležitější)
- plocha na čipu
- výnosy – souvisí s dobou uvedení na trh apod.
- atd.

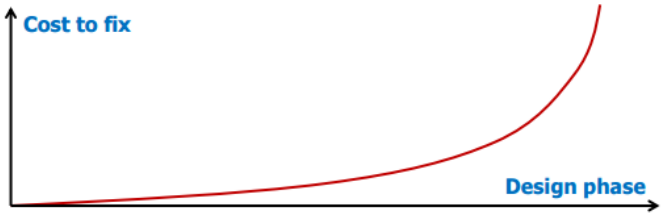


Illustration 1: cena oprav chyb v různých fázích návrhu

Metodiky návrhu zahrnují:

- **Shora-dolů** – Systém rozkládáme, až dojdeme k základním komponentám. V praxi se sám o sobě příliš nepoužívá, těžko se automatizuje, může být vhodný, pokud žádná část navrhovaného systému ještě neexistuje – dá se směřovat k optimální syntéze.
- **Zdola-nahoru –** Systém skládáme z již existujících komponent, což se v praxi typicky děje – komponenty jsou ověřené (ale můžou také být ne zcela optimální pro náš účel).
- **Kombinace shora-dolů a zdola-nahoru** – Cílem je použití existujících komponent a zároveň vysoké úrovně abstrakce.
- **Adaptivní metody návrhu** (také agilní metodiky) – Kladou důraz spíš na proces návrhu než na vytvoření detailního plánu. Byly vytvořeny jako reakce na waterfall či spiral metody. Typicky se návrh skládá z krátkých iterací.

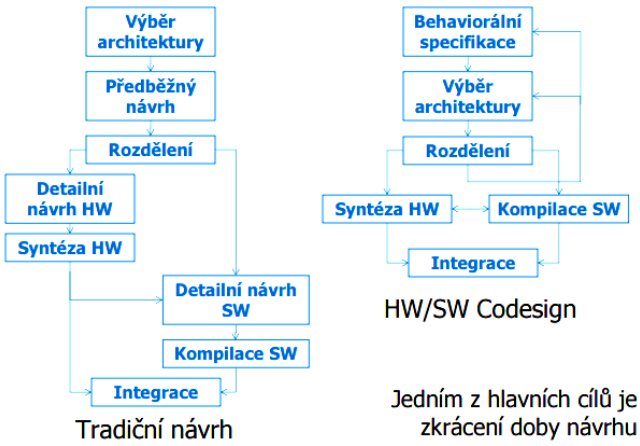


Illustration 2: tradiční vs HW/SW codesign návrh systému

Tradičně je větší důraz kladen na efektivitu procesu návrhu než na efektivitu výsledného systému. Běžně se specifikace systému mění během jeho návrhu – toto stojí další peníze.

Platformou je v HW/SW codesignu hardware, pro nějž návrh provádíme. Platforma může být zcela obecná (obecný obvod, tedy popis **RTL** – register transfer level), velmi specifická (**ASIC** – application-specific integrated circuit, např. čip na rozpoznávání řeči, který je možné programovat jen omezeně), nebo “něco mezi” (např. **FPGA** – tzv. hradlové pole, kompromis mezi rychlostí a flexibilitou, **DSP** – digital signal processor, **ASIP** – application-specific instruction set processor). Čím je platforma aplikačně specifičtější, tím je

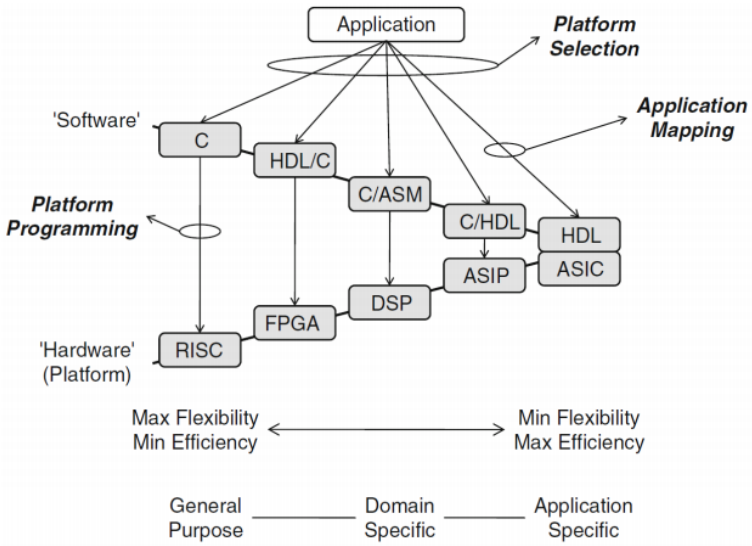


Illustration 3: vztah programování SW a HW (platformy)

efektivnější z hlediska příkonu (je pro konkrétní účel optimalizována), ale méně flexibilní, tzn. hůře programovatelná.

Programovatelný obvod je obvod, jehož funkce není v době výroby ještě definovaná a programuje jej až zákazník. Patří sem

- obecné procesory
  - **CISC** – Mnoho instrukcí.
  - **RISC** – Málo instrukcí, dosahují CPI (cycles per instruction) = 1.
  - **VLIW** – Dlouhé instrukce, paralelismus na úrovni instrukcí, velká registrová pole.
  - **Superskalární** – Paralelismus ve formě provádění několika instrukcí současně na různých jednotkách na čipu.
- **DSP** (**digital signal processor**) – Procesory pro úlohy orientované na vysoký datový tok, tzn. propustnost, s minimem podmíněných skoků, např. obrazové či zvukové filtry. Bývají vybaveny specializovanými instrukcemi, např. jednocyklovou násobičkou, multiply-accumulate (častá operace, vynásobení dvou vektorů a jeho přičtení do bufferu) apod.
- **ASIP** (**application-specific instruction set processor**) – Procesor navržený speciálně pro konkrétní účel, jeho instrukční sadu, jednotky, architekturu paměti apod. si určí zákazník. Pro takovýto procesor se potom automaticky generuje překladač, obvod atd.
- **Nevolatilní** – konfigurační informaci si obvod pamatuje i po odpojení od zdroje napájení (buď jednou pro vždy, nebo např. V EEPROM, FLASH apod.). Základní typy nevolatilních programovatelných obvodů jsou:
  - **PAL** (**programmable array logic**) – Implementují logickou funkci v normální disjunktní formě (suma součinů) pomocí programovatelného AND pole (součin, programuje se pomocí destrukce propojek v poli) a fixního OR pole (suma) – tyto jsou často ještě doplněny o klopné obvody.
  - **GAL** (**generic array logic**) – Obdobná architektura jako PAL, avšak propojky jsou programovatelné elektronicky.
  - **CPLD** (**complex programmable logic devices**) – Skládá se z AND-OR polí s elektricky programovatelnými křížovými přepínači.
- **FPGA** (**field programmable gate array**) – volatilní, konfigurace uložena v SRAM buňkách (tato konfigurace zabírá značnou plochu na čipu). Skládá se z pole programovatelných logických bloků, tzv. **CLB** (každá se skládá z logických hradel, násobiček, pamětí, DSP, I/O apod.) a konfigurovatelné propojovací sítě.

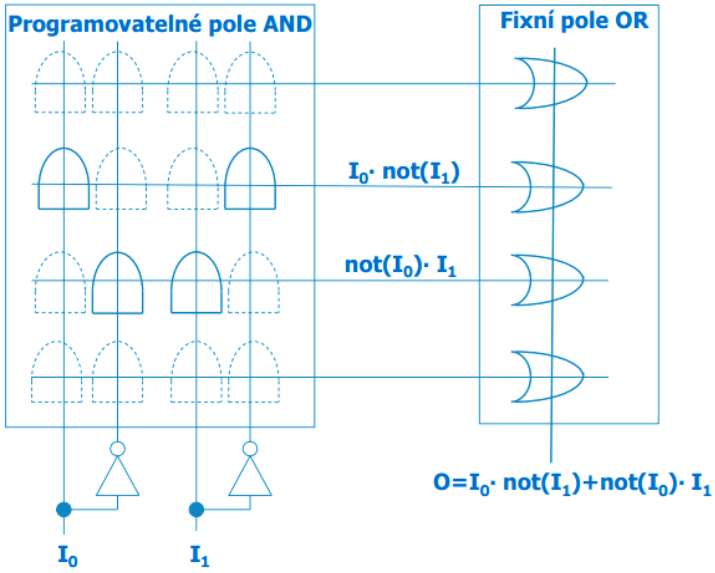


Illustration 4: PAL obvod

2. Výpočetní modely (StateCharts, Kahnova síť procesů, synchronní dataflow). HSC

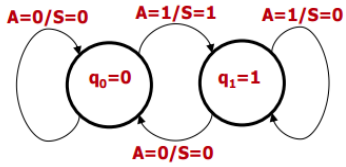
Výpočetní modely představují vysokoúrovňový nástroj pro analýzu, simulace, odhady a návrh při hardware-software codesignu, každý klade důraz na určité aspekty, abstrahuje je a zanedbává některé nepodstatné aspekty. Tyto modely dělíme na:

- **controlflow** – Zaměřují se na řízení.
- **dataflow** – Zaměřují se na data.

Modely můžeme dělit i jinak, např. podle času (diskrétní, spojitý).

Mezi důležité výpočetní modely patří:

- **konečné automaty** (**FSM**, controlflow) – Dále se dělí na **Mealy** (výstup na hranách) a **Moore** (výstup ve stavech).
- **rozšířený konečný automat** (**EFSM**, controlflow) – Redukuje počet stavů FSM zaváděním proměnných a vstupních podmínek na hranách. Pokud používáme neomezenou doménu celých čísel, je výpočetně ekvivalentní zásobníkovému automatu.
- **StateCharts** (controlflow) – Rozšiřuje EFSM zaváděním dvou typů superstavů:
  - **and** – Modeluje souběžnost, 2 nebo více podstavů jsou aktivní současně (kartézský součin potomků).
  - **or** – Jen jeden z podstavů je v daném čase aktivní.



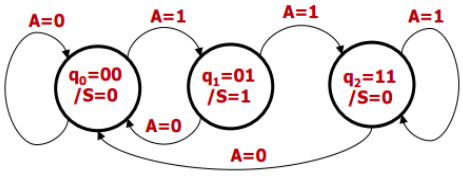


Illustration 5: konečný automat: detektor hrany (Mealy vs Moore)

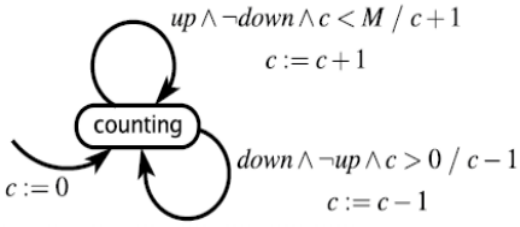


Illustration 6: EFSM: čítač

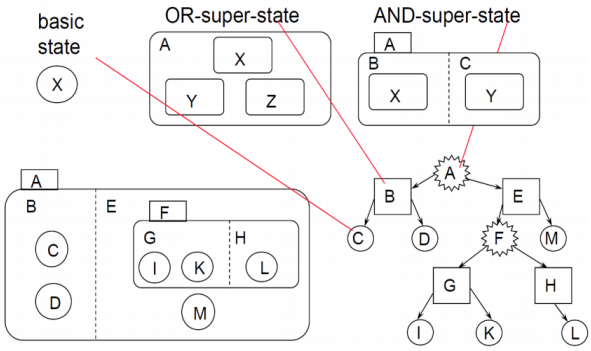


Illustration 7: StateCharts

Opět umožňuje mít podmínky u přechodů, ve formátu: *event*[*condition*]/*action*. Může obsahovat nedeterminizmus.

- **Codesign konečný automat** (**CFSM**, controlflow) – Jedná se o více FSM, tzv. **procesů**, které jsou propojené komunikačními **kanály** (vyžaduje registry). Po kanálech může přijít asynchronně **signál**, tzn. událost. spolu s daty, které se zapíšou do registru konzumenta (přičemž přepíší předchozí data, proto musí být implementace obsluhy signálu dostatečně rychlá). Signál způsobí přerušení a jeho obsluhu konzumentem (tj. samotným FSM).
- **Kahnova síť procesů** (**KPN**, dataflow) – Modeluje sekvenční procesy (algoritmy), které spolu komunikují zasíláním zpráv. Každý proces má vstupní FIFO kanál, do kterého může jiný proces neblokujícím (nikdy nečeká) způsobem zapisovat, a ze kterého může sám proces blokujícím (když nejsou data, čeká) a destruktivním způsobem číst. KPN je deterministická

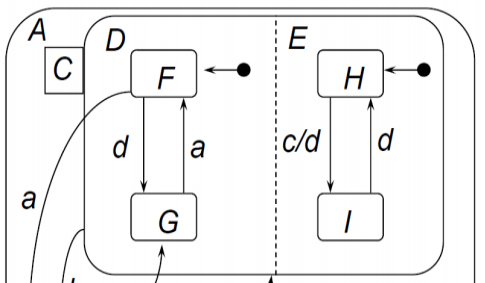


Illustration 8: StateCharts: příklad

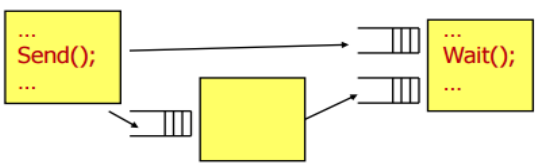


Illustration 10: KPN

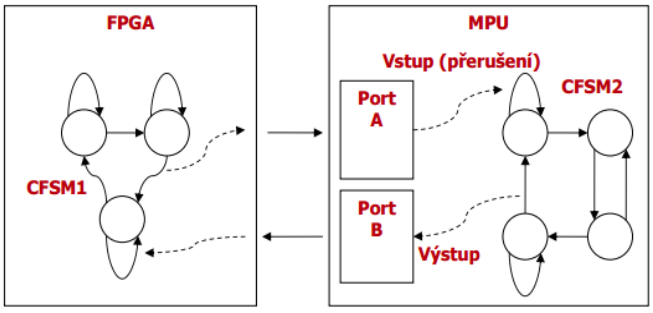


Illustration 9: CFSM: komunikace mezi FPGA a MPU, zápisy do registrů spouští přechody.

(nedeterminismus lze zavést např. neblokujícím čtením).

- **Synchronní dataflow** (**SDF**, dataflow) – Máme procesy (**aktory**), které jsou propojeny FIFO kanály. Každý proces při spuštění generuje daný počet tokenů a zapisuje je na výstup. Proces se spustí, když má na vstupu daný počet tokenů. Kanály mohou mít inicializační počet tokenů (kosočtverce). Stav SDF je vektor udávající počet tokenů v každém kanálu (počáteční vektor je dán inicializačními tokeny). SDF neumí modelovat konkrétní datové závislosti (není zde hodnota dat, pouze počty tokenů). SDF je deterministický model.
- **Statický dataflow graf** – Jednoduchý model, hodnoty jsou reprezentovány **tokeny**, výpočet je uzel grafu, který se spustí, má-li na svtupu všechny potřebné tokeny. Není zde řadič.

Mezi další výpočetní modely patří např. **Algorithmic state machine** (ASM, controlflow), FSM s datovou cestou (FSMD, controlflow), nedeterministický EFSM nebo **SpecCharts** (controlflow, rozšíření FSM o možnost psát procesy v programovacím jazyce), **Petriho sítě** apod.

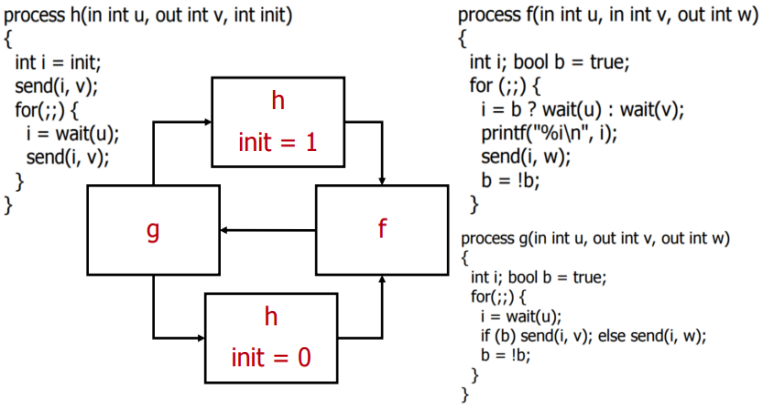


Illustration 11: KPN: tiskne alternující posloupnost 0 a 1

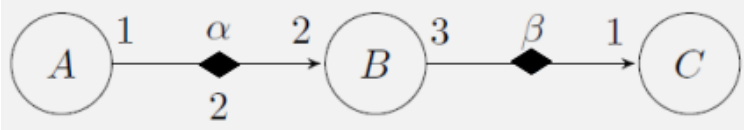


Illustration 12: SDF: Počáteční vektor je [2,1]. A se spustí a generuje 1 token, B se spustí se 2 tokeny na vstupu a generuje 3 tokeny atd.

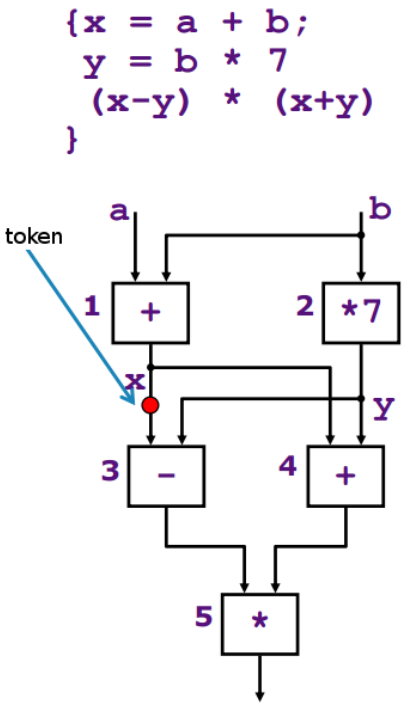


Illustration 13: statický dataflow graf

3. Specifikace (chování, struktura), syntéza (alokace, přidělení, plánování) a integrace systémů (rozhraní, synchronizace, komunikace). HSC

HW/SW codesign se skládá z následujících fází:

- **specifikace** – Výstupem je popis toho, co má systém dělat. Specifikace může být:
  1. **chování** (**behavior**) – Říká, jak se má systém chovat (dle úrovně abstrakce např. algoritmem, dif. rovnicí, logickou rovnicí apod.).
  2. **struktury** – Určuje strukturu systému, z níž chování vyplývá, tzn. popisuje komponenty systému a jejich propojení.
- **návrh** – Výstupem je popis toho, jak systém bude dělat to, co má dělat.
- **syntéza** – Transformuje algoritmus (návrh) na výpočetní strukturu a postup výpočtu, přičemž bere v potaz zadaná omezení a žádané optimalizace. Skládá se z těchto kroků:
  3. **alokace** – Výběr výpočetních prostředků z množiny všech dostupných.
  4. **přidělení** – Přidělení alokovaných prostředků daným úlohám.
  5. **plánování** – Vytváří plán, který říká, jak budou výpočetní prostředky sdíleny mezi úlohami (např. dvě úlohy potřebují násobičku, ale v HW je jenom jedna, tak se na ní musí střídat). Pokud má každá úloha dedikovaný výpočetní prostředek, není plánování potřeba.
- **integrace** – Spojuje syntetizované subsystémy do celku – zahrnuje:
  - volbu **rozhraní** - sběrnice, paměť, …, Izoluje od sebe různé technologie použité v systému a umožňuje jeho evoluci.
  - volbu **komunikace** - zasílání zpráv (jednosměrně/obousměrně, point-to-point/vícecestně, ...), sdílená paměť, …, blokující/neblokující, …
  - volbu **synchronizace** – semafory (sdílený prostředek), handshake (zaslání zprávy), mutexy, ..., časová jednotka: hodinový cyklus, transakce, …
- **validace** – Ověřuje, že systém dělá to, co má (verifikace, testivání, simulace, ...).

Typicky platí:

- Požadavky na flexibilitu, složitost a krátkou dobu do uvedení na trh favorizují SW implementaci. U SW se v podstatě model rovná přímo implementaci. SW je rovněž jednoduše znovupoužitelný.
- Požadavky na výkonnost a příkon favorizují HW implementaci. U HW máme "paralelizmus zadarmo".

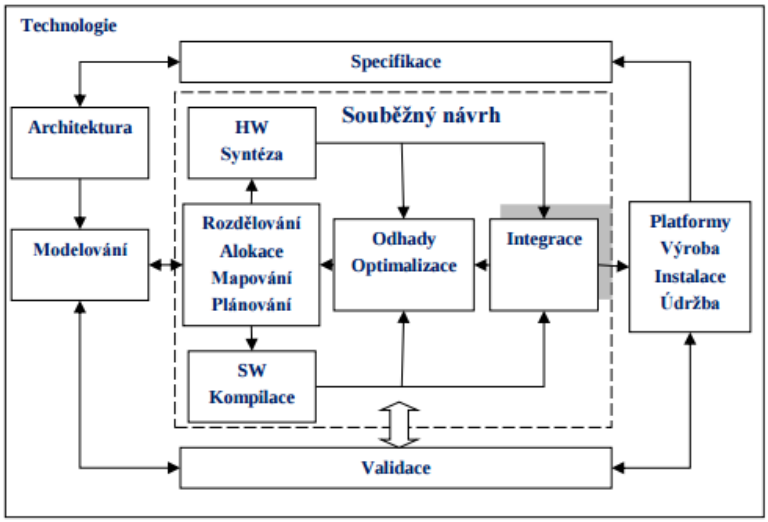


Illustration 14: HW/SW codesign

Syntéza je transformace algoritmu (popisu chování) na výpočetní struktury a postup výpočtu, přičemž zároveň probíhá optimalizace dle daných parametrů (cena, latence, příkon, ...). Skládá se z následujících částí:

- **alokace** (**allocation**) – Výběr výpočetních prostředků z množiny možných. Alokace určuje cenu návrhu.
- **přidělení** (**binding**) – Přiřazení vybraných výpočetních prostředků jednotlivým úlohám.
- **plánování** (**scheduling**) – Plán vícenásobného využití dostupných výpočetních prostředků (např. Jedna úloha využije násobičku, pokud ji zrovna nevyužívá jiná). Plánování není potřeba, pokud má každá úloha vlastní výpočetní prostředek. Plánování určuje latenci návrhu.

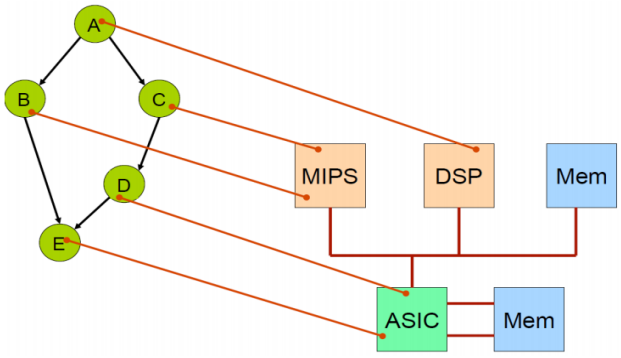


Illustration 15: mapování

Někdy se také setkáme s dělením na

- **rozdělení** (alokace + přidělení) – Přiděluje *n* výpočetních úloh *m* alokovaným výpočetním prostředkům. Při *m* = 2 mluvíme o dvoucestném rozdělení (bipartioning) – např. Rozdělení na HW a SW část, rozdělení do dvou čipů apod.

- **mapování** (přidělení + plánování)

Úlohu většinou specifikujeme **grafem úlohy** (**task graph**). Mezi tyto grafy patří:

- **data flow graph** (**DFG**) – Uzly jsou operace, hrany data.
- **control flow graph** (**CFG**) – Uzly jsou operace, hray jsou podmínky (podobně jako ve vývojovém diagramu).
- **sequence graph** (**CDFG**) – Kombinace CFG a DFG, uzly jsou operace, podmínky nebo speciální operace (NOP – začátek/konec, BR – větvení, LOOP – iterace apod.).

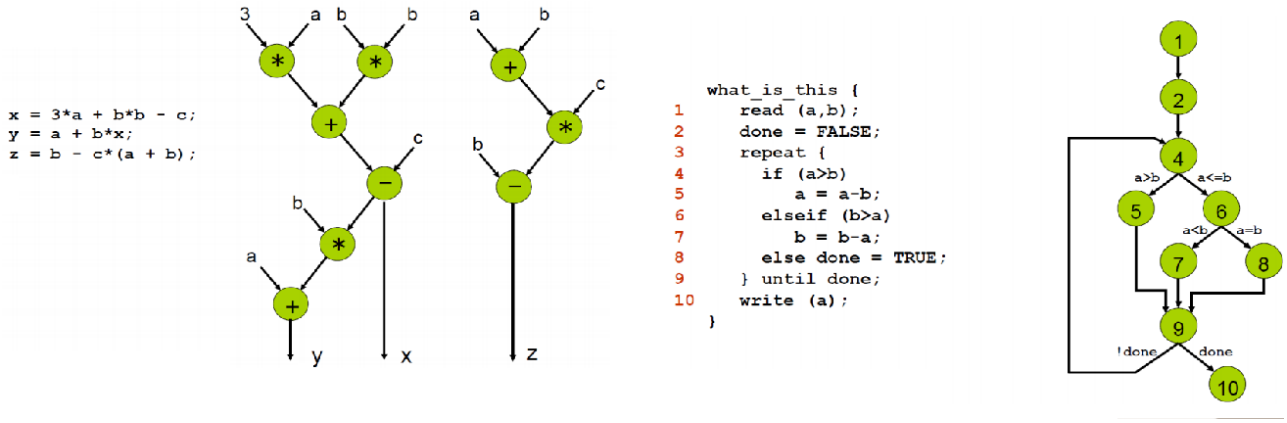


Illustration 16: DFG a CFG

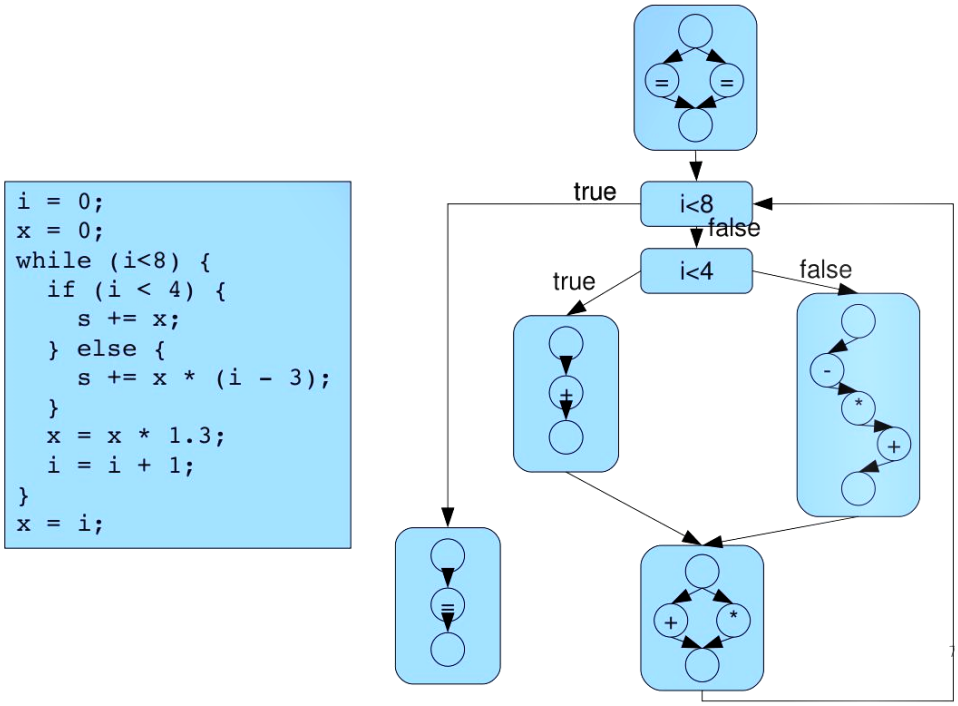


Illustration 18: CDFG příklad

4. Syntéza HW z vyšších programovacích jazyků (reprezentace, alokace, plánování, přiřazení) a jazyk Catapult C. HSC

Catapult C je prostředí umožňující syntetizovat HW reprezentaci kódu napsaného v jazyce ANSI C/C++ na **RTL** (register transfer level) popis obvodu, tzn. např. VHDL. Toto se nazývá **HLS** – high-level synthesis. Dále jej umí simulovat, ladit, verifikovat apod. Lze využívat výhody jazyka C++, jako např. OOP, šablony apod., avšak některé konstrukce, jako např. dynamická alokace paměti, float/double (nahrazeno typem fixed) nebo rekurze s neznámou hloubkou zanoření nejsou podporovány. Jako cílovou platformu lze zvolit FPGA (maticový obvod) nebo ASIC (Application-specific integrated circuit, obecný obvod). Lze vybrat cíle optimalizace (latence, plocha, spotřeba, ...). Lze taky optimalizovat smyčky pomocí:

- **rozbalení** (unrolling) – Všechny iterace smyčky se provádějí paralelně (musí být známý jejich počet). Toho nelze vždy dosáhnout kvůli datovým závislostem (iterace využívá výsledek předchozí iterace). **Částečné rozbalení** znamená, že se iterace překrývají po blocích a bloky se provádějí sekvenčně.
- **zřetězení** (pipelining) – Při zřetězení se iterace začne provádět ještě před koncem předchozí iterace. Takto lze optimalizovat i smyčky s neznámým počtem iterací (např. main).
- **spojování** (merging) – Provádění dvou nezávislých smyček současně.

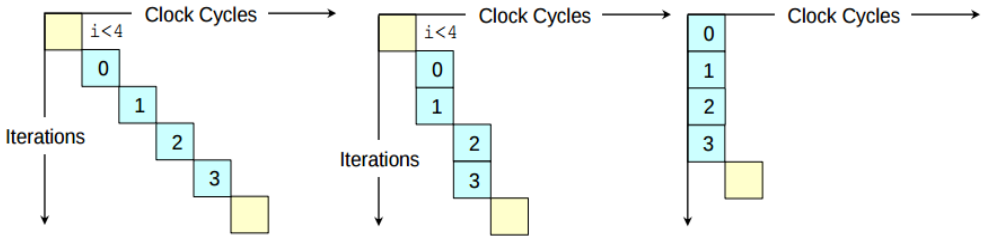


Illustration 19: rozbalení smyčky: žádné, částečné a úplné

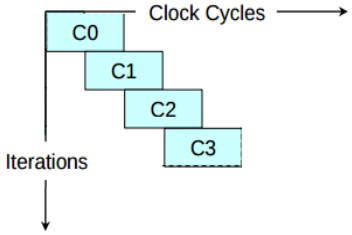


Illustration 20: zřetězení smyčky

Podmíněné výrazy se syntetizují pomocí tzv. **flatteningu** – obě větve se provádějí paralelně a podle podmínky se multiplexorem vybere jeden z výsledků.

Syntéza v Catapul C probíhá rozdělením výpočtu do kroků, tzv. **c-steps**, které představují v podstatě stavy konečného automatu. Tato část je plánováním výpočtu a jejím výstupem je **Ganttův diagram** (časový plán výpočtu). Poté se generuje **RTL** (**register transfer level**) schéma v některém z možných jazyků, jako např. VHDL nebo Verilog. Obvod se dá následně verifikovat pomocí tzv. **testbench**, tj. programu napsaného v C/C++ - testbench se spustí jednak na klasickém CPU (program lze přeložit pro CPU díky tomu, že používáme jazyk C/C++) a jednak odsimulován na vygenerovaném HW schématu, a výsledky se porovnají.

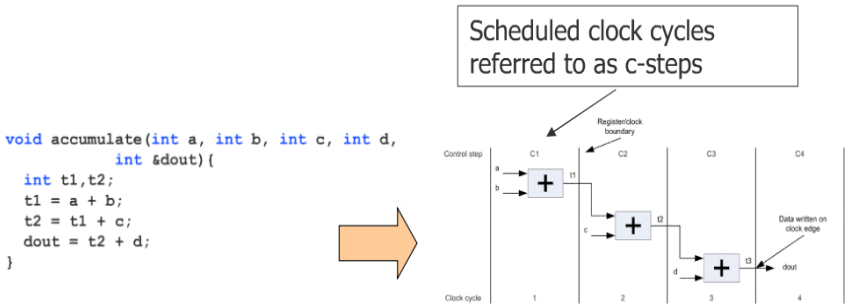


Illustration 21: plánování v Catapult C, Ganttův diagram

Catapult C využívá vlastní datové typy s přesným počtem bitů. Mezi některé patří:

- celočíselný typ (šablonový, délka v bitech a znaménkovost): `ac_int<width, signed>`
- číslo s pevnou čárkou: `ac_fixed<width, whole_part_length, signed>`

Dále existují speciální konstrukce: [] pro výběr bitu, << a >> pro bitové posuny, metody pro čtení více bitů, metody pro konverze typů apod. Funkce main se chová jako nekonečná smyčka.

Reprezentací rozumíme model pro reprezentaci syntetizovaného hardwaru, která je nezávislá na konkrétním hardwaru a na vyšším jazyce, z něhož je syntetizována. Reprezentace obvodu je podobná intermediálnímu kódu v překladačích, lze nad ní provádět optimalizace (redukci výšky grafu apod.) apod. Reprezentací může být např. CFG, DFG (lze snadno vidět paralelizaci, hrany reprezentují datové závislosti, ...), CDFG (kombinace CFG a DFG, zohledňuje jak data, tak řízení programu, existuje více forem) apod. (viz otázka č. 3).

Alokace, přiřazení a plánování jsou fáze syntézy obvodu (viz otázka č. 3). V kontextu reprezentace obvodu představují tyto fáze následující:

- **plánování** – Rozděluje vstupní CDFG do podgrafů, z nichž každý je vykonán v jednom hodinovém taktu. Mezi základní algoritmy plánování patří:
  - **ASAP** (**as soon as possible**) – Každý operátor plánuje na nejbližší možný takt.
  - **ALAP** (**as late as possible**) – Každý operátor plánuje na nejvzdálenější možný takt.
  - **FDS** (**Force-Directed Heuristic**) – Používá se pro plánování při omezeném počtu taktů výsledného řešení, což je NP-těžký problém. Využívá tzv. **mobility** operátoru, tj. rozdílu naplánovaného taktu v ALAP a v ASAP (mobilita tedy udává, v jakém rozmezí taktů může být operátor naplánován). Algoritmus se na jejím základě snaží vyrovnat očekávanou cenu daného operátoru v daném taktu.
  - **LBS** (**List-Based Scheduling**) – Používá se pro plánování při omezeném počtu zdrojů (funkčních jednotek) výsledného obvodu, což je NP-těžký problém. Jedná se o zobecněnou verzi algoritmu ASAP. Algoritmus si udržuje seznam vrcholů, které se dají v daném kroku naplánovat, a podle priority z nich vybere ty, které skutečně naplánuje.

```
if (X = 0) then
   A := B + C;
   D := B - C;
else
   D := D - 1;
end if;
```

Plánování podmíněných sekcí se provádí tzv. **flatteningem** (vit obr.), smyčky se plánují rozbalováním, zřetězováním (viz výše) apod. Volání funkcí se chová jako inline funkce.

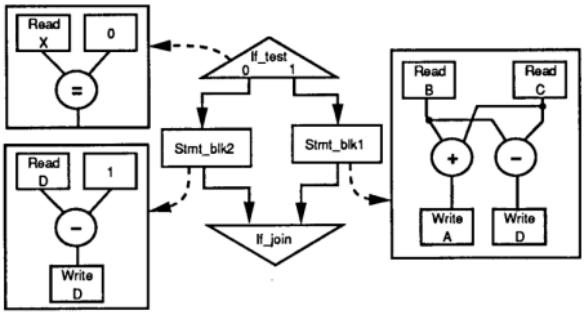


Illustration 22: možná reprezentace algoritmu v CDFG – zde se obě větve konstrukce if provádějí paralelně a výsledek se vybere multiplexorem na základě výsledku testu X, toto se nazývá flattening.

- **přiřazení** – Mapuje proměnné a operace CDFG grafu na
  1. funkční jednotky – pro operace
  2. registry – pro proměnné, konstanty apod.
  3. propojovací sítě – propojení funkčních jednotek a registrů

Mezi algoritmy přiřazení patří např. **left-edge** – Pracuje s dobou života proměnných, které se mapují do registrů. Je rychlý ale nebere v úvahu ostatní fáze přiřazení.

- **alokace** – Jde o vybírání konkrétních jednotek pro výsledný obvod, tzn. např. registry vs ROM vs FIFO pro paměťový element, sdílená sběrnice vs multiplexovaná sběrnice pro datové přenosy atd. Dostupné jednotky závisí na zvolené technologii.

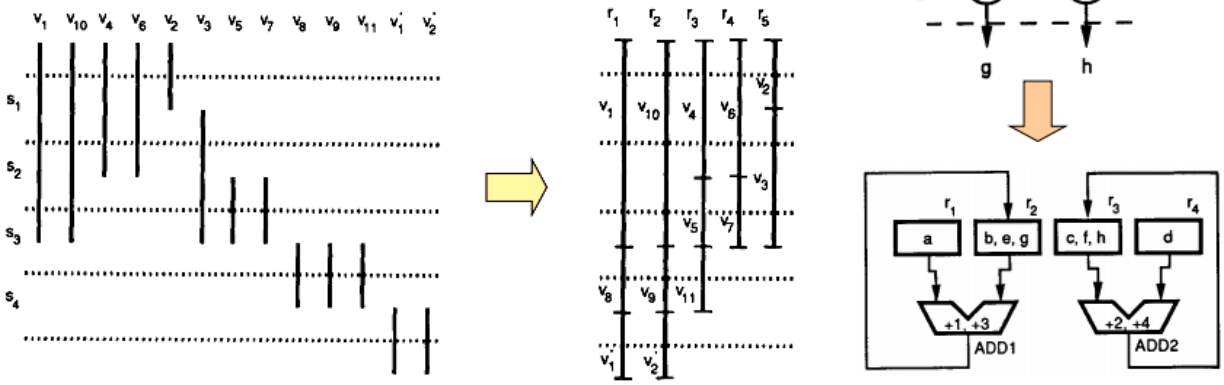
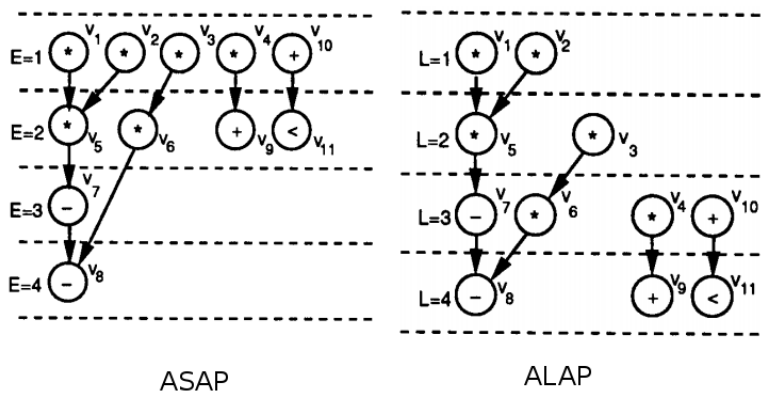


Illustration 23: příklad přiřazení zdrojů, tzn. registrů a sčítaček proměnným a operacím

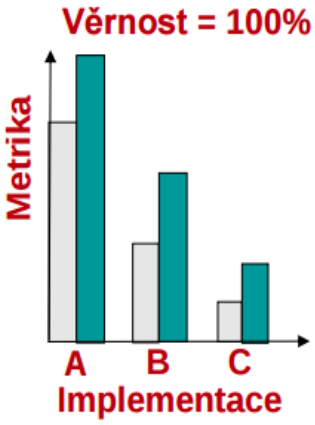
Illustration 24: left-edge algoritmus



5. Odhady (přesnost, věrnost, metriky, metody) a optimalizace vlastností systému (příkon, energie). HSC

Odhady (estimations) se snaží zjistit vlastnosti systému ještě před jeho implementací, což je nezbytné při syntéze, která probíhá s ohledem na optimalizaci některých parametrů, neboli **metrik**, mezi něž patří:

- cena (HW)
- výkonnost – změřená doba běhu (SW), benchmark (aplikační, algoritmu), parametry architektury (CPU, ...)
- počet tranzistorů/hradel (HW)
- WCET (worst-case execution time), odhad by měl být vždy horší než skutečnost
- příkon (HW)
- latence
- propustnost
- testovatelnost
- spolehlivost
- doba návrhu



Tyto parametry mají obecně protichůdnou povahu – např. vyšší výkonnost většinou vede na vyšší příkon. Provádění odhadů je komplexní problém a většinou hledáme kompromis mezi přesností a složitostí odhadu – rozlišujeme potom **high-level** (rychlé, méně přesné) a **low-level** (přesnější, pomalejší) odhady.

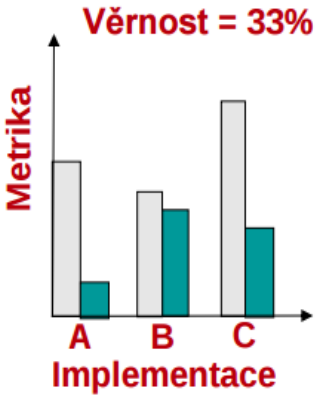


Illustration 25: věrnost

Parametry systému, z nichž odvozujeme odhady, se dají získat pomocí následujících **metod**:

- měření (rychlé prototypování)
- simulace
- statistika – statistická abstrakce systému
- formální analýza

Přesnost (**accuracy**) odhadu je dána vztahem:

$$A = 1 - \frac{|E(D) - M(D)|}{M(D)}$$

kde $E(D)$ je odhad určitého parametru implementace systému D a $M(D)$ je skutečná hodnota parametru implementace systému D.

Věrnost (**fidelity**) odhadu je procento správných predikcí pro dvojice implementací. Je dána vztahem:

$$F = 100 \times \frac{2}{n(n-1)} \times \sum_{i=1}^{n} \sum_{j=i+1}^{n} u_{i,j}$$

kde n je počet implementací a

$$u_{i,j} = \begin{matrix} 1 \\ 0 \end{matrix} \quad \begin{matrix} (E(D_i) > E(D_j) \wedge M(D_i) > M(D_j)) \vee (E(D_i) < E(D_j) \wedge M(D_i) < M(D_j)) \vee (E(D_i) = E(D_j) \wedge M(D_i) = M(D_j)) \\ jinak \end{matrix}$$

V podstatě vezmeme každou možnou dvojici implementací a díváme se, jak často se jejich porovnání shoduje s porovnáním jejich odhadů.

Nevyužitá doba periody hodin se nazývá **slack** (máme-li např. tři výpočetní jednotky a v některé periodě hodin pracuje jen jedna, je slack 66,6...% T). Slack můžeme změřit pro konkrétní periodu, nebo zpočítat průměrný slack. Slack se snažíme minimalizovat, tzn. Snažíme se maximalizovat využití (**utilization** = 1 – avgslack(T) / T) periody hodin.

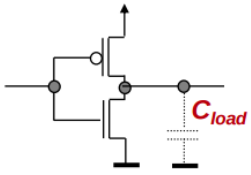


Illustration 26: CMOS invertor

Dynamický **příkon** tranzistoru v technologii CMOS se dá odhadnout jako:

$P = C_{load} \cdot \alpha \cdot f \cdot V^2$

kde C_{load} je kapacita na výstupu, *f* frekvence, α normalizovaný (0 - 1) **faktor aktivity** (je dán pravděpodobnostmi přechodů mezi log. úrovněmi) a V napájecí napětí (cca 1 volt).

Tento příkon je součtem:

- **statického příkonu** – Příkon potřebný pro udržení obvodu v chodu, když se nemění logické úrovně. Je dán svodovými proudy ve struktuře hradel.
- **dynamického** (přechodového) **příkonu** – Příkon vznikající přechody mezi logickými úrovněmi, u CMOS je dominantní. Tento příkon je dále součtem:
  - **přechodového příkonu** – skládá se z:
    - **přepínacího příkonu** – Způsobován parazitními kapacitami tranzistorů.
    - **zkratového příkonu** – Způsobem krátkým otevřením dvou tranzistorů (tedy na chvíli se přímo/zkratově spojí napájecí vstup se zemí), závisí na délkách hran signálů.
  - zátěže – parazitní kapacity na výstupu hradla

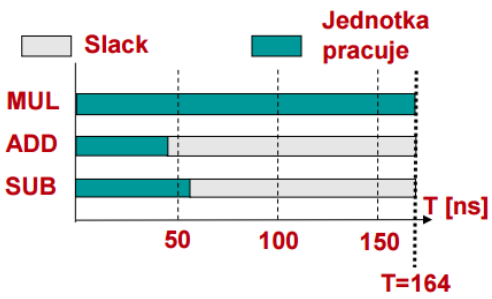


Illustration 27: využití a nevyužití (slack) periody hodin

Energie je příkon krát čas. Zpoždění CMOS obvodu je přibližně $\tau = 1 / V$.

Př.: Snížíme napájecí napětí na polovinu, jak moc klesne příkon? $P = C_{load} \cdot \alpha \cdot f \cdot (V / 2)^2 = ¼ \cdot C_{load} \cdot \alpha \cdot f \cdot (V / 2)^2$, tedy klesne 4 krát. Zpoždění ale vzroste 2 krát ($\tau = 1/(V/2) = 2 / V$).

Př.: Kolik je α pro NAND pro přechod z 0 do 1 (s rovnoměrným rozložením pravd.)? Celkem existuje 16 možných přechodů (2 vstupy NAND, to je 8 bin. možností, a 2 stavy, před přechodem a po, tj. 8 * 2 = 16), z toho 3 (11 → 00, 11 → 01 a 11 → 10) způsobí přechod výstupu z 0 do 1. α je tedy 3 / 16.

Problém s příkonem je, že roste úměrně složitosti obvodu a užitečnost (efektivní výkon) obvodu roste cca s druhou odmocninou složitosti (tzv. **Pollackovo pravidlo**), tzn. zvyšujeme-li výkon obvodu lineárně, příkon roste zhruba kvadraticky. Problém u velkého příkonu je odvod tepla.

Příkon snižujeme pomocí snižováním faktoru aktivity, parazitních kapacit (malé tranzistory, krátké spoje), snižováním napětí (což ale snižuje možnou maximální frekvenci), snižováním frekvence apod. Obvod také může dynamicky měnit napětí a frekvenci dle potřeby (dynamic voltage and frequency scaling).

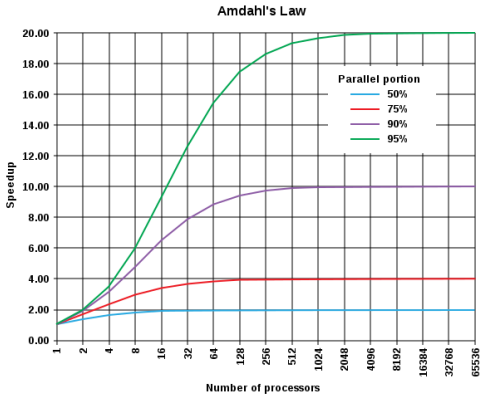


Illustration 28: Amdhalův zákon

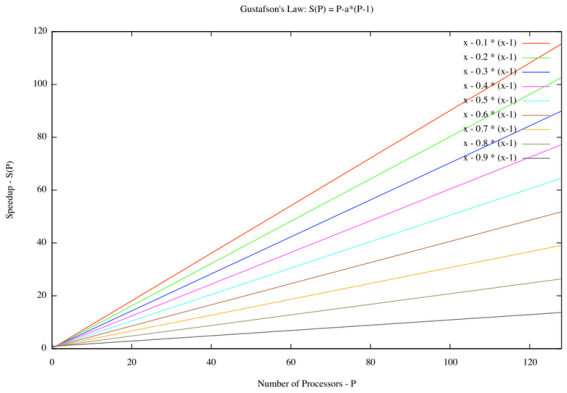


Illustration 29: Gustafsonův zákon

Pro odhad urychlení výpočtu paralelizací se využívají dva zákony:

- **Amdhalův**: $S = \frac{1}{(1-p)+\frac{p}{s}}$
- **Gustafsonův**: $S = 1 - p + sp$

kde

- *S* je celkové zrychlení.
- s je urychlení paralelizovatelné části.
- *p* je zlomek výpočetního času (pro Amdhalův) nebo práce (pro Gustafsonův), který paralelizovatelná část původně trvala/zabrala.

Rozdíl mezi těmito zákony je, že Amdhalův předpokládá, že máme fixní objem výpočetní práce a pracujeme s časem (který se snažíme zkrátit), kdežto Gustafsonův předpokládá, že máme daný čas, v němž chceme udělat co nejvíce práce. Amdhalův zákon říká, že urychlit úlohu lze dobře pouze tehdy, je-li z velké většiny paralelizovatelná (např. pouze 50% nestačí), Gustafsonův zákon dává odhad urychlení vyšší. V praxi se často tyto zákony používají jako dolní a horní mez.

6. Jazyk a sémantika predikátové logiky (termy, formule, realizace jazyka, pravdivost formulí). MAT

Logika je formální odvozovací systém, který se používá k popisu matematických teorií, vět a důkazů. Základní pojmy logiky jsou např.:

- **axiom** – Výchozí tvrzení, které nedokazujeme, jeho platnost předpokládáme. Z axiomů odvozujeme další tvrzení. Pro axiomy požadujeme **bezespornost** (důsledkem axiomů nesmí být nějaké tvrzení a současně jeho negace) a většinou také **nezávislost** (jeden axiom nesmí být důsledkem jiného).
- **důsledek** – Odvozené tvrzení.
- **důkaz** – Posloupnost odvozující jisté tvzení z jiných.

Predikátová logika je rozšířením **výrokové logiky** – zatímco výroková logika zkoumá stavbu složitějších tvrzení ze základních výroků, predikátová logika rozlišuje jednotlivá individua. Prvky jazyka predikátové logiky 1. řádu (proměnnými jsou individua, ale ne např. množiny) jsou:

- **logické symboly**:
  - **proměnné**: x, y, ..., x_1, x_2, ...
  - **logické spojky**: ¬, &, ∨, →, ≡
  - **kvantifikátory**: ∀, ∃
  - **závorky**: (,)
  - **predikátový symbol rovnosti**: =
- **speciální symboly**:
  - **funkční symboly**: f, g, ... f_1, f_2, ..., každý má určitou četnost (nezáporné celé číslo) (v sémantice se funkční symboly používají pro obecné funkce, např. *věk*(*člověk*))
  - **predikátové symboly**: p, q, ..., p_1, p_2, ..., každý má určitou četnost (kladné celé číslo) (v sémantice se predikátové symboly používají zjednodušeně řečeno jako funkce vracející true nebo false, predikátem může být např. *je_člověk(kdo)*)

Obsahuje-li jazyk symbol = pro rovnost, mluvíme o **jazyku s rovností**. Příkladem jazyků můžou být jazyk teorie uspořádání (jazyk s rovností, který má jeden predikátový symbol: <) nebo jazyk teorie grup (jazyk s rovností, nulární funkční symbol 1 pro neutrální prvek a binární funkční symbol . pro grupovou operaci). V jazyce se dále rozlišují následující pojmy:

- **term** (proměnná nebo funkce nebo konstanta), formálně:
  1. Každá proměnná je term.
  2. Je-li *f* funkční symbol četnosti *n* a jsou-li t_1, ... t_n termy, pak také $f(t_1, ..., t_n)$ je term. Z tohoto plyne, že i každá konstanta je term.
  3. Každý term vznikne konečným počtem použití předchozích pravidel.
- **atomická formule** (predikát aplikovaný na term, tzn. nějaké tvrzení), formálně:
  Je-li *p* predikátový symbol četnosti *n* a jsou-li t_1, ..., t_n termy, pak $p(t_1, ..., t_n)$ je atomická formule. Máme-li jazyk s rovností a t_1 a t_2 jsou termy, pak $t_1 = t_2$ je term (psáno místo $=(t_1,t_2)$).
- **formule** (atomické formule kombinované pomocí spojek a kvantifikátorů), formálně:
  1. Každá atomická formule je formule.
  2. Jsou-li φ a ψ formule, pak také (¬φ), (φ & ψ), (φ ∨ ψ), (φ → ψ), (φ ≡ ψ) jsou formule.
  3. Je-li *x* proměnná a φ formule, pak také (∀*x*φ) a (∃*x*φ) jsou formule.
  4. Každá formule vznikne konečným použitím výše uvedených pravidel.

Místo ¬(*x* = *y*) píšeme (*x* ≠ *y*) a pokud to nenarušuje jednoznačnost, vynecháváme závorky. Výskyt proměnné *x* ve formuli může být buď vázany (nachází se v nějaké podformuli tvaru ∀*x*φ nebo ∃*x*φ) nebo volný (pokud není vázaný). Formule neobsahující žádnou volnou proměnnou se nazývá **uzavřená formule** nebo také **výrok**.

Dále následuje vysvětlení sémantiky predikátové logiky. Nechť *L* je dále jazyk predikátové logiky 1. řádu.

Realizace jazyka *L* je algebraická struktura, která se skládá z následujících částí:

- **univerzum**: neprázdná množina *M*
- Pro každý funkční symbol *f* četnosti *n* je dáno zobrazení f_M: $M^n \to M$. Pro $n = 0$, tedy pro konstantu, se jedná v podstatě o vyznačení určitého symbolu z *M*.
- Pro každý predikátový symbol *p* četnosti *n*, kromě rovnosti, je dána relace $p_M \subseteq M^n$.

Ohodnocení proměnných je libovolné zobrazení *e* všech proměnných jazyka do množiny *M*. Ohodnocení, které proměnné *x* přiřazuje ohodnocení m a na všech ostatních prvcích splývá s ohodnocením *e*, značíme e(*x*/*m*).

Pravdivost formule φ v realizaci *M* jazyka *L* při ohodnocení *e* (zapisujeme $M \vDash \varphi[e]$) definujeme indukcí podle složitosti:

- φ je atomická formule tvaru $p(t_1, ..., t_n)$, kde p je predikátový symbol četnosti *n* a $t_1, ..., t_n$ jsou termy ⇒ $M \vDash \varphi[e] \Leftrightarrow (t_1[e], ..., t_n[e]) \in p_M$.

- φ je atomická formule tvaru $t_1 = t_2$, kde t_1 a t_2 jsou termy ⇒ $M \models \varphi[e] \Leftrightarrow$ $t_1[e]$ a $t_2[e]$ jsou tentýž prvek v M.
- φ je tvaru ¬ψ, kde ψ je formule jazyka L⇒ $M \models \varphi[e] \Leftrightarrow M \not\models \psi[e]$.
- φ je ve tvaru (η & ψ), (η ∨ ψ), (η → ψ) nebo (η ≡ ψ), kde η a ψ jsou formule, klademe
  - $M \models (\eta \,\&\, \psi)\ [e] \Leftrightarrow M \models \eta\ [e]$ a současně $M \models \psi\ [e]$
  - $M \models (\eta \vee \psi)\ [e] \Leftrightarrow M \models \eta\ [e]$ nebo $M \models \psi\ [e]$
  - atd.
- φ je ve tvaru (∀xψ), kde ψ je formule jazyka L ⇒ $M \models \varphi\ [e] \Leftrightarrow$ pro každý prvek $m \in M$ je $M \models \varphi\ [e(x/m)]$.
- φ je ve tvaru (∃xψ), kde ψ je formule jazyka L ⇒ $M \models \varphi\ [e] \Leftrightarrow$ existuje prvek $m \in M$ je $M \models \varphi\ [e(x/m)]$.

Formule A, která je pravdivá v libovolném ohodnocení se nazývá **tautologie** a značí se $\models A$ (např. $X \vee \neg X$). Věta o korektnosti říká, že libovolná dokazatelná formule výrokové logiky je tautologie.

7. Formální systém predikátové logiky (axiomy a odvozovací pravidla, dokazatelnost, model a důsledek teorie, věty o úplnosti a kompaktnosti, prenexní tvar formulí). MAT

Axiomy predikátové logiky (je jich nekonečně mnoho) jsou dány schematy:

- **výrokové axiomy**: jsou-li φ, ψ, η formule, pak následující jsou axiomy:
  - $\varphi \to (\psi \to \varphi)$
  - $(\varphi \to (\psi \to \eta)) \to ((\varphi \to \psi) \to (\varphi \to \eta))$
  - $((\neg\psi) \to (\neg\varphi)) \to (\varphi \to \psi)$
- **axiom kvantifikátoru**: jsou-li φ, ψ formule a x proměnná, která nemá volný výskyt ve φ, pak následující je axiom:
  - $(\forall x(\varphi \to \psi)) \to (\varphi \to (\forall x\psi))$
- **axiom substituce**: je-li φ formule, x proměnná a t term substituovatelný za x do φ, pak následující je axiom:
  - $(\forall x\varphi) \to \varphi_x[t]$
- **axiomy rovnosti**: Je-li x proměnná, pak $x = x$ je axiom. Jsou-li $x_1, ..., x_n, y_1, ..., y_n$ proměnné a f funčkční symbol četnosti n a p predikátový symbol četnosti n, pak následující jsou axiomy:
  - $(x_1 = y_1 \to (x_2 = y_2 \to (... (x_n = y_n \to f(x_1, ..., x_n) = f(y_1, ..., y_n))) ...))$
  - $(x_1 = y_1 \to (x_2 = y_2 \to (... (x_n = y_n \to p(x_1, ..., x_n) \to p(y_1, ..., y_n))) ...))$

Odvozovací pravidla jsou:

- **modus ponens** (**odloučení**) – Z formulí φ a $\varphi \to \psi$ se odvodí ψ.
- **generalizace** (**zobecnění**) – Pro libovolnou proměnnou x z formule φ se odvodí formule $\forall x\varphi$.

Důkaz v predikátové logice prvního řádu je posloupnost formulí $\varphi_1, ..., \varphi_n$,v níž pro každé i je φ_i buď axiomem nebo ji lze odvodit z předchozích formulí posloupnosti pomocí odvozovacích pravidel (odloučení nebo zobecnění).

Formule φ je **dokazatelná** právě tehdy, když existuje důkaz, jehož poslední formulí je φ. Píšeme $\vdash\varphi$.

Důkaz může mít množinu **předpokladů** T. V takovém případě se v posloupnosti důkazu může navíc (k axiomům a odvoditelným formulím) vyskytovat i formule z T. Je-li formule φ dokazatelná z předpokladů T, píšeme $T \vdash \varphi$.

Věta o dedukci říká, že $T \vdash A \to B$ právě když $T \cup \{A\} \vdash B$.

Teorie je množina T formulí jazyka L predikátové logiky 1. řádu. Tyto formule spolu s axiomy predikátové logiky tvoří soustavu všech axiomů dané teorie. Jestliže pro každou formuli φ jazyka L platí $T \vdash \varphi$, pak je teorie T **sporná**, jinak je **bezesporná**.

Nechť T je teorie s jazykem L a M realizace jazyka L (realizace definuje univerzum a zobrazení funkčních a predikátových symbolů). M je **modelem** teorie T, jesliže $M \vDash \sigma$ (σ je pravdivá v M) pro každou formuli σ z T. Pak píšeme $M \vDash T$.

Formule σ je **důsledkem** teorie T, jestliže pro každý její model M platí $M \vDash \sigma$. Pak píšeme $T \vDash \sigma$.

Gödelovy věty o úplnosti tvrdí:

- Teorie T je bezesporná právě tehdy, když má nějaký model.
- Je-li T teorie s jazykem L, pak pro libovolnou fomuli σ z T platí: $T \vdash \sigma \Leftrightarrow T \vDash \sigma$.

Z těchto vět vyplývá **věta o kompaktnosti**: Teorie T má nějaký model právě tehdy, když každá její konečná podmnožina má model.

Prenexní tvar formule je určitý "standardizovaný", jednotný tvar používaný pro formule. Formule v prenexní formě má tvar:

$Q_1x_1...Q_nx_nB$, kde

- Pro každé $i \geq 0$ je Q_i buď $\forall$ nebo $\exists$.

- $x_1 ... x_n$ jsou navzájem různé proměnné.
- B je otevřená formule (neobsahuje kvantifikátory).

Ke každé formuli lze sestrojit ekvivalentní formuli v prenexním tvaru. Převod lze provést následujícími transformacemi:

- Vyloučíme zbytečné kvantifikátory – Vynecháme všechny kvantifikátory $\forall x$ resp. $\exists x$ v podformulích tvaru $\forall xB$ resp. $\exists xB$, pokud se proměnná x nevyskytuje volně v B.
- Přejmenujeme proměnné – Vyhledáme podformuli QxA, kde x se vyskytuje volně v A. Pokud je ve výchozí formuli ještě jiný výskyt proměnné x, nahradíme nalezenou formuli QxA její variantou $Qx'A'$, kde x' je proměnná různá od všech ostatních ve výchozí formuli. Toto opakujeme, dokud nemají všechny kvantifikátory různé proměnné a žádná proměnná není současně volná i vázaná.
- Eliminujeme spojky $\Leftrightarrow$ – Použijeme schéma $A \Leftrightarrow B = (A \Rightarrow B) \wedge (B \Rightarrow A)$.
- Přesuneme negace dovnitř – Toto opakujeme tak dlouho, dokud se negace vyskytují maximálně přímo před atomickými formulemi. Použijeme schémata:

  $\neg(\forall xA) = \exists x \neg A$

  $\neg(\exists xA) = \forall x \neg A$

  $\neg(A \Rightarrow B) = A \wedge \neg B$

  ...
- Přesuneme kvantifikátory doleva – Je-li B formule, v níž se nevyskytuje x, používáme následující schémata:

  $(QxA) \vee B = Qx(A \vee B)$

  $(QxA) \wedge B = Qx(A \wedge B)$

  $(QxA) \Rightarrow B = \overline{Q}x(A \Rightarrow B)$

  $B \Rightarrow (QxA) = Qx(B \Rightarrow A)$

Př.: Převeďte do prenexního tvaru formuli $\forall y(\exists x\, P(x,y) \rightarrow \exists u\, R(y,u)) \rightarrow \forall x S(x,y)$.

1. přejmenujeme prom.: $\forall y_1(\exists x_1\, P(x_1,y_1) \rightarrow \exists u\, R(y_1,u)) \rightarrow \forall x_2\, S(x_2,y_2)$
2. přesuneme kvan. ven: $\forall x_2\, (\forall y_1(\exists x_1\, P(x_1,y_1) \rightarrow \exists u\, R(y_1,u)) \rightarrow S(x_2,y_2))$
3. přesuneme kvant. ven: $\forall x_2\, \exists y_1\, ((\exists x_1\, P(x_1,y_1) \rightarrow \exists u\, R(y_1,u)) \rightarrow S(x_2,y_2))$
4. přesuneme kvant. 2x ven: $\forall x_2\, \exists y_1\, \exists x_1\, ((P(x_1,y_1) \rightarrow \exists u\, R(y_1,u)) \rightarrow S(x_2,y_2))$
5. přesuneme kvant. 2x ven: $\forall x_2\, \exists y_1\, \exists x_1\, \forall u\, ((P(x_1,y_1) \rightarrow R(y_1,u)) \rightarrow S(x_2,y_2))$

Př. 2 (jednoduchý): Převeďte $(\forall x: (x \wedge y)) \rightarrow x$.

1. $(\forall x_1: (x_1 \wedge y)) \rightarrow x_2$
2. $\exists x_1\, ((x_1 \wedge y) \rightarrow x_2)$
3. $\forall x_2 \forall y \exists x_1\, ((x_1 \wedge y) \rightarrow x_2)$

8. Algebraické struktury (grupy, okruhy, obory integrity a tělesa, svazy a Boolovy algebry, univerzální algebry). MAT

Univerzální algebra je struktura $(A, \omega_1, \omega_2, ..., \omega_l)$, kde ω_i je n_i-nární operace a A je nosná množina. Algebry se označují tzv. **typem**, např. algebra typu (1, 2) znamená, že má daná algebra jednu unární a jednu binární operaci.

Pologrupa je algebraická struktura $(H, \cdot)$, pokud $\cdot$ je asociativní binární operace.

Grupa je algebraická struktura $(G, \cdot)$, kde G je nosná množina a $\cdot$ je binární operace na G. Pro grupu musí platit:

- **uzavřenost**: $\forall a, b \in G: a \cdot b \in G$
- **asociativita**: $a \cdot (b \cdot c) = (a \cdot b) \cdot c, a\ b\ c \in G$
- existence **neutrálního prvku**: $\exists e \in G\ \forall a \in G: a \cdot e = e \cdot a = a$
- existence **inverzního prvku** ke každému prvku: $\forall a \in G\ \exists b \in G: a \cdot b = b \cdot a = e$

Abelovská grupa je grupa, pro kterou platí komutativita: $\forall x\ y \in G: x \cdot y = y \cdot x$. Grupy typicky popisují symetrické struktury, např. všechny povolené transformace Rubikovy kostky tvoří grupu. Dalšími příklady jsou celá čísla se sčítáním, nenulová racionální čísla s násobením apod.

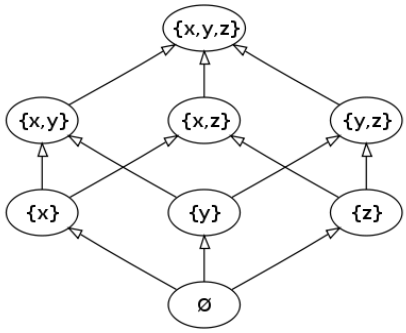


Illustration 30: příklad svazu

Okruh je algebra $(R, +, \cdot)$ typu (2,2) právě tehdy, když:

- $(R, +)$ je abelovská grupy.
- $(R, \cdot)$ je pologrupa.
- $\cdot$ je distributivní nad $+$.

Příkladem okruhu je např. množina celých čísel se sčítáním a násobením.

Obor integrity je komutativní (pro obě operace) okruh $(R, +, 0, –, \cdot, 1)$ s jednotkovým prvkem, pro který navíc platí:

1. $\forall a \in R, a \neq 0, \forall b \in R, b \neq 0: a \cdot b \neq 0$ (Neexistují netriviální dělitelé nuly.)
2. $0 \neq 1$

Těleso je okruh s jednotkovým prvkem (R, +, 0, -, ·, 1) právě tehdy, když existuje inverzní prvek pro násobení:

$$\forall\ x \in R / \{0\}\ \exists\ y \in R: x \cdot y = y \cdot x = 1, \text{ y značíme } x^{-1}$$

Příkladem tělesa jsou např. racionální čísla. Komutativní těleso se nazývá **pole**. Každé pole je obor integrity.

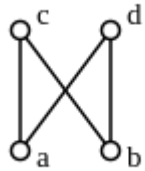


Illustration 31: toto <u>není</u> svaz (c a d nemají společné supremum)

Svaz je množina S s uspořádáním $\leq$, kde platí, že každá dvouprvková podmnožina S obsahuje své supremum i infimum. **Supremum**, resp. **infimum** množiny $T \subseteq S$ jsou zobecněním pojmu největší, resp. nejmenší prvek a definují se jako x: x je nejmenší, resp. největší prvek z množiny všech prvků, které jsou větší nebo rovny, resp. menší nebo rovny než každý prvek množiny T. Tzn. např. otevřený interval (0,1) v reálných číslech nemá největší ani nejmenší prvek, ale má supremum (1) a infimum (0).

Algebra $(V, \cap, \cup)$ je svazem právě tehdy, když pro každé $a, b, c \in V$ splňuje axiomy:

- $a \cap b = b \cap a, a \cup b = b \cup a$ (komutativita)
- $a \cap (b \cap c) = (a \cap b) \cap c, a \cup (b \cup c) = (a \cup b) \cup c$ (asociativita)
- $a \cap (a \cup b) = a, a \cup (a \cap b) = a$

Operace $\cap$ a $\cup$ představují výběr suprema, resp. infima.

Booleova algebra je algebra $(B, \cap, \cup, 0, 1, ')$ typu (2, 2, 0, 0, 1) právě tehdy, když platí:

- $x \cup y = y \cup x, x \cap y = y \cap x$ (komutativita)
- $x \cup (y \cap z) = (x \cup y) \cap (x \cup z), x \cap (y \cup z) = (x \cap y) \cup (x \cap z)$ (distributivita)

- $x \cup 0 = x$, $x \cap 1 = x$ (neutralita 0 a 1)
- $x \cup x' = 1$, $x \cap x' = 0$ (komplementarita)

Počet prvků Booleovy algebry je 2^n.

9. Základní algebraické metody (podalgebry, homomorfismy, přímé součiny, kongruence a faktorové algebry, normální podgrupy a ideály okruhů). MAT

Nechť $(A, \omega_1, \omega_2, ..., \omega_I)$ je algebra, pak $(B, \omega_1, \omega_2, ..., \omega_I)$ je její **podalgebra** právě tehdy, když $B \subseteq A$ a pro každé $i \in \{1, ..., I\}$ platí $x_1 x_2 ... x_{ni} \in B \Rightarrow \omega_i x_1 x_2 ... x_{ni} \in B$ (tj. každá operace je uzavřená na B).

Nechť $A = (M, \omega_1, \omega_2, ..., \omega_I)$ a $A^* = (M^*, \omega_1^*, \omega_2^*, ..., \omega_I^*)$ jsou algebry téhož **typu** (mají stejný počet operací stejných četností). Zobrazení $f: A \to A^*$ se nazývá **homomorfismem** A do A^* právě tehdy, když:

- Pro každou operaci ω_i z A, jejíž arita je větší než 0 platí: $\forall x_1 x_2 ... x_{ni} \in A$: $f(\omega_i x_1 x_2 ... x_{ni}) = \omega_i^* f(x_1) f(x_2), ... f(x_{ni})$.
- Pro každou operaci ω_i z A, jejíž arita je 0 platí: $f(\omega_i) = \omega_i^*$.

Izomorfismus je bijektivní homomorfismus.

Nechť $\alpha_k = (A_k, w_1^k, w_2^k, ..., w_i^k)$ jsou algebry téhož typu $(n_1, n_2, ..., n_i)$ a A kartézský součin všech množin A_k (tzn. množina $k$tic představující kombinace každého prvku s každým ze všech množin). Pro všechna $i \in I$ buď operace w_i definována jako:

$$w_i(a_1, a_2, ..., a_{n_i}) = \begin{cases} (w_i^1(a_1, a_2, ..., a_{n_i}), ..., w_i^k(a_1, a_2, ..., a_{n_i})) & n_i > 0 \\ (w_i^1, w_i^2, ..., w_i^k) & n_i = 0 \end{cases}$$

Algebra $(A, w_1, w_2, ..., w_i)$ se nazývá **přímý součin** algeber α_k.

Buď $\alpha = (A, w_1, w_2, ..., w_k)$ algebra typu $(n_1, n_2, ..., n_i)$ a π relace ekvivalence (reflexivní, symetrická a tranzitivní, dělí množinu na třídy ekvivalence) na A. π se nazývá **kongruence** na α, právě tehdy, když

$\forall i \in I, a_1, ... a_{ni}, b_1, ... b_{ni} \in A$: $a_1 \pi b_1 \wedge ... a_{ni} \pi b_{ni} \wedge \Rightarrow w_i(a_1, ... a_{ni}) \pi w_i(b_1, ... b_{ni})$

Příklad kongruence: mějme binární relaci ekvivalence r = "žijí ve stejné zemi" na množině lidí. Pokud máme operaci w, která vezme dva lidi, podívá se na země, ve kterých žijou, a vybere tu větší z nich, pak r je kongruencí na algebře (lidé, w).

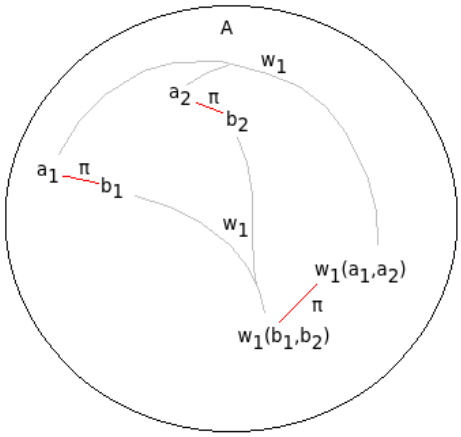


Illustration 32: kongruence

Je-li A množina a Π relace ekvivalence na množině A, pak množina A / Π je množina tříd ekvivalence relace Π, tzn.:

$A / \Pi = \{[a]_\Pi \mid a \in A\}$,

kde $[a]_\Pi$ značí třídu prvku a.

Máme-li algebru $\alpha = (A, w_1, w_2, ..., w_k)$ a kongruenci Π na α, pak **faktorovou algebrou** α / Π je algebra:

$\alpha / \Pi = (A / \Pi, w_1^*, w_2^*, ..., w_k^*)$,

kde operace jsou definovány jako:

$w_i^*([a_1], ..., [a_{ni}]) = [w_i(a_1, ..., a_{ni})]$

Příkladem faktorové algebry je např. algebra, která vznikne pomocí relace modulo n (zbytku po dělení n, tzn. dva prvky jsou v relaci právě tehdy, mají-li stejný zbytek po dělení n) na algebře nezáporných čísel se sčítáním a odčítáním.

Podgrupa N grupy $(G, \cdot, e, ^{-1})$ se nazývá **normální podgrupou** (zapisujeme $N \lhd G$) právě tehdy, když platí:

$\forall x \in G: xN = Nx, xN \in N$

kde:

$xN = \{x \cdot n \mid n \in N\}$ $\quad$ $Nx = \{n \cdot x \mid n \in N\}$.

V abelovské grupě je každá podgrupa normální. Máme-li okruh $(R, +, \cdot)$, pak jeho podokruh G se nazývá **ideálem** okruhu R, právě tehdy, když:

$\forall a \in R, b \in G: a \cdot b \in G \wedge b \cdot a \in G$.

Množiny R a $\{0\}$ jsou vždy ideálem a nazývají se **triviálním ideálem**, ostatní se nazývají **vlastním ideálem**.

10. Obory integrity a dělitelnost (okruhy polynomů, pravidla dělitelnosti, Gaussovy a Eukleidovy okruhy). MAT

Okruh je algebra $(R, +, \cdot)$ typu (2,2) právě tehdy, když:

- $(R, +)$ je abelovská grupa.
- $(R, \cdot)$ je pologrupa.
- $\cdot$ je distributivní nad $+$.

Příkladem okruhu je množina {0,1,2,3} s operacemi + a · modulo 4. Pro $R = \{0\}$ jde o tzv. triviální okruh.

V okruhu se prvek a nazývá levý (pravý) **dělitel nuly** právě tehdy, existuje-li x takové že $ax = 0$ ($xa = 0$). Nenulový dělitel nuly se nazývá **netriviální dělitel nuly**.

Okruh polynomů $F[x]$ je okruh, jehož prvky tvoří polynomy nad proměnnou (tzv. neurčitou) x, jejichž koeficienty tvoří další okruh (často je to navíc **pole** – množina s operacemi +, -, · a /, s vlastnostmi jaké mají na reálných číslech). R[x] jsou polynomy s reálnými koeficienty, Z[x] s celočíselnými koeficienty, dále můžou být binární koeficienty atd. Dva polynomy jsou si rovny, právě když jsou si rovny všechny jejich korespondující koeficienty. Polynomy sčítáme sečtením korespondujících koeficientů (např. $(x + 1) + (3x^2 + x) = 3x^2 + 2x + 1$) a násobíme klasicky roznásobením závorek. Pro dělení polynomů existuje algoritmus (dělíme vždy nejvyšší mocninu nejvyšší mocninou, pak zpětně vynásobíme, odečteme a opakujeme), např.:

```
x² − 9x − 10 : x + 1 = x − 10
x² + x
  − 10x − 10
  − 10x − 10
          0          // zbytek, může být nenulový
```

Základní pojmy týkající se polynomů jsou:

- **stupeň** – nejvyšší mocnina polynomu
- **kořen** – hodnota proměnné, pro kterou je polynom roven 0 (kořen nemusí existovat a může jich být víc)

Euklidovský okruh je komutativní okruh, v němž může být definována alespoň jedna **euklidovská funkce**. Euklidovská funkce je funkce f: $R \setminus \{0\} \rightarrow \mathbb{Z}^+_0$, pro kterou platí:

> Pro a a b, $b \neq 0$, existují prvky q a r takové, že $a = bq + r$ (dělení se zbytkem) a buď $r = 0$ nebo $f(r) < f(b)$.

Příkladem euklidovské funkce je např. absolutní hodnota na množině celých čísel, nebo stupeň polynomu na množině polynomů. Tato funkce zavádí možnost porovnávání a tedy zavádění výsledků dělení se zbytkem (bez ní je možné zavádet jenom samotnou dělitelnost, ale ne výsledek dělelní).

Gaussův okruh je okruh, v němž lze každý prvek rozložit na jednoznačný součin prvočinitelů (základní věta aritmetiky). Gaussův okruh je vždy oborem integrity. Pro každé dva prvky Gaussova okruhu existuje **největší společný dělitel** (**NSD**) a **nejmenší společný násobek** (**NSN**). Každý Euklidovský okruh je Gaussův okruh. Euklidovský I Gaussův okruh je vždy oborem integrity.

Obor integrity je komutativní (pro obě operace) okruh $(R, +, 0, –, \cdot, 1)$ s jednotkovým prvkem (pro násobení, u sčítání mluvíme o nulovém prvku), pro který navíc platí:

1. $\forall a \in R, a \neq 0, \forall b \in R, b \neq 0: a \cdot b \neq 0$ (Neexistují netriviální dělitelé nuly.)
2. $0 \neq 1$

Příkladem oboru integrity je množina celých čísel (s operacemi + a ·).

Buď $(I, +, 0, –, \cdot, 1)$ obor integrity a $a, b \in I$. Prvek a je **dělitelný** prvkem b (zvaným **dělitel**), formálně $b \mid a$, právě tehdy, když $\exists c \in I: a = bc$. **Jednotka** oboru integrity je každý prvek, který dělí jednotkový prvek okruhu, tzn. např. pro celá čísla jsou jednotky 1 a -1 (protože jednotkový prvek je 1 a $1 \cdot 1 = 1$ a $-1 \cdot -1 = 1$). **Asociované prvky** jsou prvky, které se liší jen vynásobením jednotkou, např. pro celá čísla jsou asociovaná čísla 5 a -5. **Triviální dělitelé** prvku x jsou všechny jednotky a všechny prvky asociované s x. **Vlastní dělitelé** jsou netriviální dělitelé. **Ireducibilní prvek** je prvek, který má jenom triviální dělitele, tzn. pro celá čísla jsou např. prvočísla a 1 ireducibilní. **Prvočinitel** p je speciální případ ireducibilního prvku, který není ani jednotkovým, ani nulovým prvkem a navíc pro každý součin ab dělitelný p je a nebo b dělitelné p.

Elementární **pravidla dělitelnosti** jsou (pro $a, b, c \in I$):

1. $a \mid 0$

2. $1 \mid a$
3. $a \mid a$
4. $a \mid b \land b \mid c \Rightarrow a \mid c$
5. $a \mid b \Rightarrow a \mid b\,c$
6. $a \mid b \land a \mid c \Rightarrow a \mid b + c$
7. $c \neq 0 \Rightarrow a \mid b \Leftrightarrow a\,c \mid b\,c$
8. $a \mid b \land c \mid d \Rightarrow a\,c \mid b\,d$
9. $a \mid b \Rightarrow a^n \mid b^n$

11. Teorie polí (minimální pole, rozšíření pole, konečná pole a jejich konstrukce). MAT

Pologrupa je algebra $(H, \cdot)$ typu (2) (tzv. **grupoid**), právě když $\cdot$ je asociativní.

Grupa je algebra (H, +, 0, –) typu (2, 0, 1) právě tehdy, když

1. + je uzavřená na H.
2. Platí asociativita, tzn. $a + (b + c) = (a + b) + c$.
3. 0 je neutrální prvek vzhledem k +, tzn. $a + 0 = 0 + a = a$.
4. Pro každý prvek a existuje inverzní prvek $–a$ takový, že $a + –a = –a + a = 0$.

Příkladem (abelovské) grupy je $(Z,+)$, neboli celá čísla se sčítáním.

Abelovská grupa je grupa, kde + je komutativní, tzn. $a + b = b + a$.

Okruh je algebra $(R, +, 0, –, \cdot)$ typu (2, 0, 1, 2), právě když

1. $(R, +, 0, –)$ je abelovská grupa.
2. $(R, \cdot)$ je pologrupa.
3. $\cdot$ je distributivní nad + (tzn. $A \cdot (B + C) = A \cdot B + A \cdot C$).

Příkladem okruhu je množina {0,1,2,3} s operacemi + a $\cdot$ modulo 4 (tzv. Z_4).

Okruh s jednotkovým prvkem je okruh $(R, +, 0, –, \cdot, 1)$, kde 1 je neutrální (jednotkový) prvek vzhledem k $\cdot$, tzn. $1 \cdot x = x \cdot 1 = x$.

Těleso je okruh, pro který platí:

$$\forall x \in R \setminus \{0\}\ \exists\, y \in R\text{: } x \cdot y = y \cdot x = 1, \qquad \text{neboli } y = x^{-1}$$

Příkladem tělesa je množina racionálních čísel.

Pole je komutativní těleso (tzn. $(R \setminus \{0\}, \cdot)$ je abelovská grupa). Pole lze definovat jako $(R, +, 0, –, \cdot, 1, ^{-1})$, kde:

- + a $\cdot$ jsou asociativní ($(a + b) + c = a + (b + c)$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$).
- Existuje aditivní a multiplikativní jednotkový prvek (0 a 1).
- Existuje aditivní a multiplikativní inverze (– a $^{-1}$).
- $\cdot$ je distributivní nad + ($a \cdot (b + c) = (a \cdot b) + (b \cdot c)$).

Příkladem pole je množina komplexních nebo reálných čísel.

Podpole A' pole A je podalgebra A, která je sama polem. Podalgebra X' algebry X má stejné operace (které musí dávat stejné výsledky a musí být uzavřené na nosné množině) jako X, jenom její nosná množina je podmnožinou nosné množiny X'.

Minimální pole je pole, které nemá jiné podpole než sebe sama.

Pole P_1 je **rozšířením** pole P_2, pokud P_1 má více prvků a P_2 je podpole pole P_1.

Konečné (**Galoisovo**) **pole** je pole s konečně mnoha prvky q, což nazýváme jeho **řádem**. q musí být celočíselná mocnina prvočísla.

Pro každé prvočíslo p můžeme jednoduše sestrojit konečné pole řádu GP(p) jako celá čísla modulo p. Např. GP(3) = ({0,1,2}, + mod 3, 0, –, $\cdot$ mod 3, 1, $^{-1}$). Inverze – prvku $a \neq 0$ je $p – a$. Inverze $^{-1}$ lze spočítač buď hrubou silou nebo Euklidovým algoritmem.

Pro GP(a), kde a je jiná než první mocnina prvočísla $a = p^n$, $n \neq 1$, je GP(p) **charakteristickým polem** tohoto pole. Za pomoci tohoto pole můžeme najít rozšíření pole GP(p) nad polynomy s koeficienty 0 nebo 1, které vytvoří pole GP(n).

Toto pole je *n*-rozměrným vektorovým prostorem nad GP(*p*). Např. GP(4): $4 = 2^2$, charakter. pole je GP(2) a tvoří tedy 2D (podle exponentu 2) vektorový prostor nad GP(2): (0,0), (0,1), (1,0), (1,1). Skaláry tohoto pole jsou prvky charakteristického pole (tzn. 0,1).

Nadpole L pole K nad ním tvoří vektorový prostor. Dimenze tohoto prostoru se značí [L : K] a nazývá se **stupeň rozšíření**. **Konečné rozšíření** je rozšíření s konečným stupněm.

12. Metrické prostory (příklady, konvergence posloupností, spojitá a izometrická zobrazení, úplnost, Banachova věta o pevném bodu). MAT

Metrický prostor je množina bodů X, na níž je dána tzv. **metrika** (vzdálenost), což je jednoznačná nezáporná reálná funkce funkce $\delta(x,y)$, $x, y \in X$ taková, že:

- $\delta(x,y) = 0 \Leftrightarrow x = y$
- $\delta(x,y) = \delta(y,x)$ (symetrie)
- $\delta(x,y) + \delta(y,z) \geq \delta(x,z)$ (trojúhelníková nerovnost)

Diskrétní metrický prostor je metrický prostor s metrikou: $\delta(x,y) = 0 \Leftrightarrow x = y$, jinak $\delta(x,y) = 1$.

Příklady metrických prostorů jsou:

- n-rozměrné **Euklidovské prostory** R^n (n-tice reálných čísel) s **Euklidovskou vzdáleností**
  $$\delta(x,y) = \sqrt{\sum_{k=1}^{n} (y_k - x_k)^2}$$
- n-rozměrné prostory s jinými metrikami, jako např.:
  $$\delta(x,y) = \sum_{k=1}^{n} |y_k - x_k| \quad \text{(značeno } R^n{}_1\text{)}$$
  nebo
  $$\delta(x,y) = max_{1 \leq k \leq n} |y_k - x_k| \quad \text{(značeno } R^n{}_0\text{)}$$
- množina reálných čísel se vzdáleností $\delta(x,y) = |x - y|$
- množina všech spojitých reálných funkcí na intervalu $\langle a,b \rangle$ se vzdáleností $\delta(f,g) = \max_{a \leq t \leq b} |g(t) - f(t)|$

Otevřená koule $S(x_0,r)$ je množina bodů x takových, že

$$\delta(x, x_0) < r$$

Jedná li se o neostrou nerovnost (≤), pak je koule **uzavřená**. Otevřená koule s poloměrem ε se také nazývá **ε-okolí**.

Je-li $x_1, x_2, \ldots$ posloupnost bodů metrického prostoru, říkáme, že **konverguje** k bodu x, jestliže každé ε-okolí bodu x obsahuje všechny body x_n počínaje nějakým indexem n. Tzn. pokud ke každému okolí s $\varepsilon > 0$ existuje index n, za nímž již všechny body leží v ε-okolí.

Zobrazení $f : X \rightarrow Y$ z metrického prostoru (X,δ_1) do dalšího prostoru (Y,δ_2), se nazývá **spojité** v bode x_0, pokud pro libovolné $\varepsilon_1 > 0$ lze najít $\varepsilon_2 > 0$ takové, že pro všechny body x s vlastností $\delta_1(x,x_0) < \varepsilon_1$ platí $\delta_2(f(x),f(x_0)) < \varepsilon_2$. Zobrazení se nazývá spojité, pokud je spojité v každém bodě.

Zobrazení se nazývá **homeomorfismem**, pokud je vzájemně jednoznačné a jak toto zobrazení, tak jeho inverzní zobrazení jsou spojitá. Dva metrické prostory jsou homeomorfní, pokud mezi nimi existuje homeomorfismus.

Izometrismus je speciální případ homeomorfismu. Pro izometrické zobrazení musí pro každé x_1 a x_2 platit

$$\delta_1(x_1, x_2) = \delta_2(f(x_1),f(x_2))$$

Opět platí, že dva prostory jsou izometrické, existuje-li mezi nimi izometrismus. Izometrismus znamená, že metrické vztahy mezi prostory jsou stejné a z hlediska metrických prostorů je můžeme považovat za totožné.

Cauchyovská posloupnost je posloupnost bodů $x_1, x_2, \ldots$, jestliže pro každé $\varepsilon > 0$ existuje n takové, že

$$\forall\, a\ b \geq n : \delta(x_a,x_b) < \varepsilon$$

Tzn. od určitého indexu je vzdálenost libovolných dvou bodů menší než ε. Každá konvergentní posloupnost je Cauchyovská. Pokud zároveň platí, že každá Cauchyovská posloupnost je konvergentní, nazýváme metrický systém **úplným**.

Pevný bod zobrazení f je bod x, pro který platí $f(x) = x$. Jinak řečeno x je řešení rovnice $f(x) = x$.

Zobrazení f: $X \rightarrow X$ se nazývá **kontraktivní** (**kontrakce**), jestliže existuje číslo $a < 1$ takové, že

$$\forall\, x\, y : \delta(f(x),f(y)) \leq a\ \delta(x,y)$$

Banachův princip pevného bodu (BPPB) říká, že každé kontraktivní zobrazení v neprázdném úplném metrickém prostoru X má právě jeden pevný bod.

BPPB je důležitý pro důkazy o řešeních různých rovnic. Také dává praktickou metodu přibližného výpočtu těchto řešení.

Uzávěrový bod množiny je bod, jehož libovolné okolí leží v množině M. **Vnitřní bod** množiny M je bod, pokud existuje jeho okolí, které celé leží v M. Množina se nazývá **uzavřená**, pokud M = uzávěr M. Množina se nazývá **otevřená**, pokud obsahuje jen vntřní body. Množina je uzavřená (otevřená) právě tehdy, je-li její doplněk otevřený (uzavřený) – viz obrázek.

13. Normované a unitární prostory (základní vlastnosti a příklady, normované prostory konečné dimenze, uzavřené ortonormální systémy a Fourierovy řady). MAT

Lineární (také **vektorový**) prostor nad polem (T,+, 0 , – , · , 1) (T je množina skalárů, např. R) je algebra (V, +, 0, –, w_1, w_2, …, $w_{|T|}$) (V je množina vektorů, např. R × R, operace w_i je unární operace násobení vektoru skalárem $i \in$ T), pro kterou platí:

1. (V, +, 0, –) je abelovská grupa
2. ∀ x, y ∈ V, a, b ∈ T:
   1. $w_a(x + y) = w_a(x) + w_a(y)$ (distributivita násobení skalárem)
   2. $w_{a+b}(x) = w_a(x) + w_b(x)$ (distributivita násobení vektorem)
   3. $w_{ab}(x) = w_a(w_b(x))$ (asociativita násobení skalárem)
   4. $w_1(x) = x$ (násobení jednotkovým skalárem je ten samý vektor)

Nejčastěji bereme jako vektrový prostor množinu *n*tic reálných čísel nad polem R. Pak operace násobení skalárem značíme klasicky tečkou.

Normovaný prostor je lineární prostor, u nějž je pro každý prvek (vektor) x definována tzv. norma $||x||$ taková, že:

1. $||x|| = 0 \Leftrightarrow x = 0$ (první nula je skalár, druhá vektor)
2. $||a \odot x|| = |a| \cdot ||x||$
3. $||x + y|| \leq ||x|| + ||y||$ trojúhelníková nerovnost

Příklady normovaných prostorů jsou

- množina R se sčítáním a násobením a absolutní hodnotou jakožto normou
- ***n*-rozměrný prostor**, značeno R^n (popř. C^n pro komplexní čísla): množina uspořádaných n-tic (x_1, …, x_n) s operacemi sčítání n-tic (po složkách) a násobení n-tice číslem (rovněž po složkách) s normou:
  $$\|x\| = \sqrt{\sum_{k=1}^{n} x^2}$$
- množina všech spojitých reálných funkcí na intervalu $\langle a,b \rangle$ s operacemi sčítání funkcí a násobení funkce číslem a normou $||f|| = \max_{a \leq t \leq b} |f(t)|$.

Každý normovaný prostor je zároveň **metrickým prostorem** (metrický prostor je obecnější struktura, skládá se jenom z množiny a metriky).

Prvky $x, y, ..., w$ lineárního prostoru se nazývají **lineárně závislé**, pokud existují skaláry $a, b, ..., l$, z nichž alespoň jeden je různý od nuly, takové, že

$$a \odot x + b \odot y + ... + c \odot l = 0$$

Jestliže v lineárním prostoru lze najít D lineárně nezávislých prvků (vektorů), ale libovolné D + 1 prvky jsou už vždy lineárně závislé, říkáme, že **dimenze** (**rozměr**) prostoru je D. Dimenze může být nekonečná. Systém n lineárně nezávislých vektorů v n-rozměrném prostoru nazýváme jeho **bází**.

Platí, že každý lineární prostor konečné dimenze D je izomorfní s množinou vektorů Euklidovského prostoru R^D, proto můžeme prvky tohoto prostoru považovat za Dtice čísel.

Každý lineární prostor konečné dimenze n je izomorfní s euklidovským prostorem R^n. Můžeme proto uvažovat poouze vektory, jejichž prvky jsou reálná čísla.

Skalárním součinem v lineárním prostoru P nazáváme zobrazení $(\_ , \_) : P \times P \rightarrow R$, pro kterou platí (λ je reálné číslo):

1. $(x,y) = (y,x)$
2. $(x_1 + x_2, y) = (x_1,y) + (x_2,y)$
3. $(\lambda x, y) = \lambda(x,y)$
4. (x,x) ≥ 0, přičemž $(x,x) = 0 \Leftrightarrow x = 0$

Prostor, v němž je definován skalární součin, se nazývá **unitární**. V unitárním prostoru se norma zavádí jako $\|x\| = \sqrt{(x, x)}$ (což je to samé jako výše uvedený vztah pro normu).

Množina nenulových vektorů se nazývá **ortogonální**, jestliže skalární součin každých dvou různých z nich je roven 0. Dále se nazývá **ortonormální**, jestliže skalární součin každého vektoru se sebou samým je roven 1.

Máme-li ortogonální bázi $x_1, \ldots x_n$ n-rozměrného prostoru, můžeme každý prvek p vektor prostoru zapsat pomocí této báze jako

$$p = \sum_{k=1}^{n} c_k x_k$$

kde každý koeficienty c_k získáme pomocí skalárního součinu jako $c_k = (p, x_k)$. Koeficienty ck nazýváme **souřadnicemi** nebo **Fourierovými koeficienty** prvku p vzhledem k této bázi. Výše zmíněná suma je potom tzv. **Fourierovou řadou**.

Ortonormální systém je **uzavřený**, pokud pro každý jeho prvek p a jeho Fourierovy koeficienty c_k platí tzv. Parsevalova rovnost

$$p = \sum_{k=1}^{\infty} c_k^2 = \|f\|^2$$

Uzavřenost znamená, že pro každý prvek prostoru k němu posloupnost částečných součtů Fourierovy řady konverguje.

14. Obyčejné grafy (stupně uzlů, sledy, souvislost, izomorfismy, stromy, kostry, Kruskalův a Primův algoritmus pro hledání minimální kostry ohodnoceného grafu, eulerovské a hamiltonovské grafy, planarita a obarvitelnost). MAT

Obyčejný graf je graf, který nemá orientované hrany. Je to dvojice G = (U, H), kde U je konečná množina **vrcholů** (**uzlů**) a H ⊆ {{u,v} | u, v ∈ U ∧ u ≠ v} je konečná množina **hran**. **Stupeň uzlu** je počet hran, které do daného uzlu zasahují. Dále definujeme následující pojmy:

- **sled** mezi uzly *u* a *v*: posloupnost střídajících se uzlů a hran grafu: $u = w_0, h_0, w_1, h_1, ..., w_{n-1}, h_{n-1}, w_n = v$, taková, že $h_i = \{w_i, w_{i+1}\}$.
- **tah** mezi uzly *u* a *v*: sled mezi *u* a *v* takový, že se hrany neopakují ($i \neq j \Rightarrow h_i \neq h_j$), pokud je první a poslední uzel stejný, nazývá se tah **uzavřeným**.
- **cesta** mezi uzly *u* a *v*: tah mezi *u* a *v* takový, že se uzly neopakují ($i \neq j \Rightarrow w_i \neq w_j$).

Souvislý graf je takový graf, pro nějž platí, že pro libovolné dva uzly mezi nimi existuje sled (tzn. graf není rozdělený na více nepropojených).

Kružnice je uzavřený tah, v němž se uzly neopakují, s výjimkou prvního a posledního, které jsou stejné.

Podgraf G' grafu G je graf, jehož množina uzlů a hran je podmnožinou grafu G.

Faktor grafu je podgraf, jehož množina uzlů je stejná jako množina uzlů G (liší se případně jen hranami).

Strom je graf, jehož žádný podgraf není kružnicí.

Ohodnocení je přiřazení hodnoty (nejčastěji číselné) hranám (popř. uzlům či obojím) grafu pomocí tzv. ohodnocovací funkce. Ohodnocení hran může představovat např. vzdálenost, nebo cenu cesty, mezi městy představovanými uzly. Ohodnocovací funkce je funkce:

$$v: H(G) \rightarrow (0,\infty)$$

Planární graf (také rovinný graf) je graf, který lze v rovině nakreslit tak, že se žádné jeho hrany nekříží. Pro rovinné grafy platí:

- **Eulerův vzorec**:

$$|U| - |H| + s = 2,$$

  kde $|U|$ je počet uzlů, $|H|$ počet hran a s počet stěn (oblastí vymezených rovinně nakresleným grafem, včetně celé vnější oblasti). Pokud však Eulerův vztah pro nějaký graf platí, nemusí být nutně planární (jde pouze o implikaci).
- Kuratowského věta: Graf je rovinný právě tehdy, není-li žádný jeho podgraf izomorfní dělení grafu K_5 ani $K_{3,3}$.

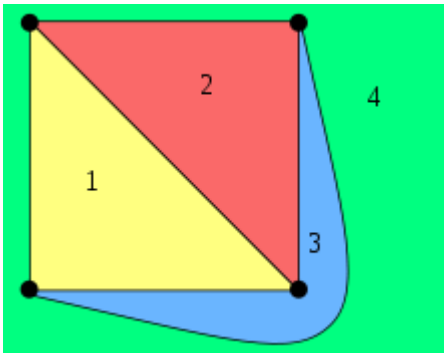


Illustration 33: planární graf, spolu s vymezenými oblastmi (stěnami)

Kostra grafu je jeho faktor, který je stromem.

Pro hledání minimální kostry ohodnoceného grafu používáme následující algoritmy:

- **Kruskalův**: Hrany seřadíme od nejnižší ceny a postupně je přidáváme. Pokud by přidáním hrany vznikl graf, který není stromem (a tedy ani kostrou), hranu přeskočíme.
- **Primův**: Začne v libovolném uzlu a vybere hranu s nejnižším ohodnocením takovou, že jejím přidáním zůstane graf souvislý a přitom nevznikne kružnice. Oproti Kruskalovu algoritmu je výhodou to, že se nemusí řadit všechny hrany.

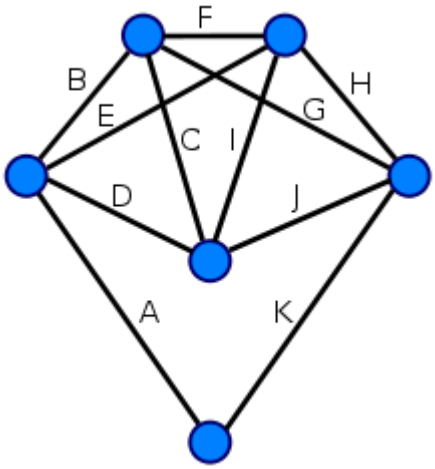


Illustration 34: Eulerovský graf

Izomorfismus mezi dvěma grafy je bijektivní (vzájemně jednoznačné) zobrazení uzlů jednoho grafu na uzly grafu druhého takové, že zobrazení zároveň zachovává přesně stejné propojení uzlů. Neformálně jsou dva grafy izomorfní, když jsou (co se týká struktury) stejné. Formálně jsou grafy *G* a *G'* izomorfní, pokud:

$\exists f: U(G) \rightarrow U(G'): \{x,y\} \in H(G) \Leftrightarrow \{f(x),f(y)\} \in H(G')$.

Eulerovský graf je souvislý neorientovaný graf, který má všechny uzly sudeého stupně, což zároveň znamená, že existuje uzavřený tah obsahující všechny jeho hrany (dá se nakreslit "jedním tahem").

Hamiltonovský graf je graf, který se dá projít cestou obsahující každý jeho uzel právě jednou, s výjimkou počátečního uzlu, který je zároveň uzlem cílovým – tato cesta je kružnicí a nazývá se **Hamiltonovská kružnice** (tzn. graf je Hamiltonovský, právě když obsahuje Hamiltonovskou kružnici).

Barvení grafu se zabývá přiřazováním barev (celých čísel) jeho vrcholům (barvení ploch nebo stěn se dá na tento případ převést). **Obarvením grafu** nazveme takové přiřazení barev, že žádné dva sousední vrcholy nemají stejnou barvu. **Chromatické číslo** grafu je číslo udávající minimální počet barev potřebný k obarvení grafu. Pro libovolný rovinný graf je chromatické číslo maximálně 4 (tento problém se nazývá problém čtyř barev).

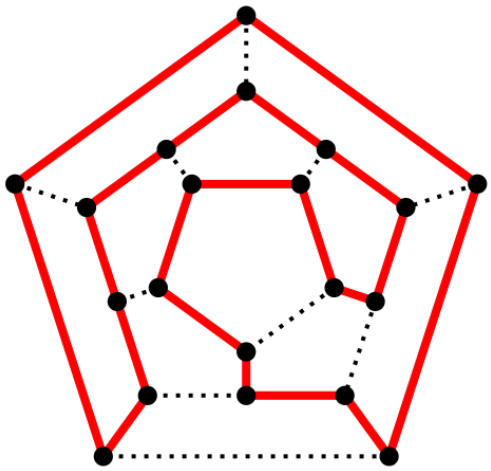


Illustration 35: Hamiltonovský graf s vyznačenou Hamiltonovskou kružnicí

15. Orientované grafy (orientované sledy, souvislost a silná souvislost, turnaje, eulerovské a hamiltonovské grafy, Dijkstrův a Floyd-Warshallův algoritmus pro hledání cesty minimální délky). MAT

Orientovaný graf je dvojice (U, H), kde U je konečná množina vrcholů a $H = \{(u, v): u, v \in U\}$ je konečná množina uspořádaných hran.

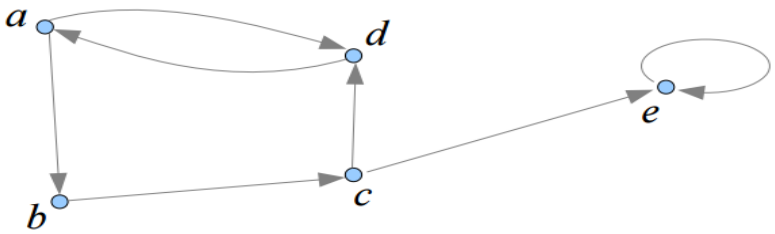


Illustration 36: orientovaný graf

Symetrizace grafu je operace, která udělá z orientovaného grafu graf obyčejný. Provede se zanedbáním směrů a smyček v původním grafu. Formálně:

$H' = \{\{u,v\}: u, v \in U, u \neq v, \exists\, h \in H: h = (u, v) \vee h = (v, u)\}$

Řekneme, že orientovaný graf je **souvislý**, jestliže jeho symetrizace je souvislý graf.

Řekneme, že orientovaný graf je **silně souvislý**, jestliže pro jeho libovolné dva uzly existuje orientovaná cesta (střídající se posloupnost uzlů a hran, přičemž jde jenom po směru hran a neopakují se uzly ani hrany).

Turnaj je orientovaný graf bez smyček, kde je mezi každými dvěma uzly právě jedna hrana. Formálně pro graf $T = (U, H)$: $\forall \{u, v\}, u\ v \in U, u \neq v\ \exists$ právě jedna hrana $h \in H$: $h = (u, v) \vee h = (u, v)$.

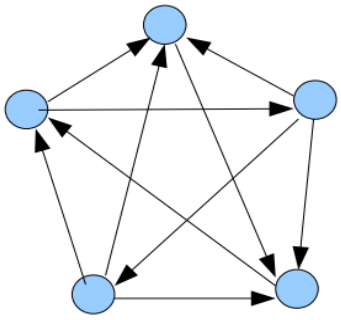


Illustration 37: turnaj

Orientovaný graf se nazývá **eulerovský**, jestliže v něm existuje uzavřený orientovaný tah (posloupnost vrcholů a hran, hrany se neopakují), který obsahuje všechny hrany. Souvislý orientovaný graf je eulerovský právě tehdy, když pro každý jeho uzel u platí

$$deg_+(u) = deg_-(u),$$

kde deg_+ resp. deg_- značí vstupní resp. výstupní stupeň uzlu (kolik hran jde do resp. z uzlu).


Illustration 38: souvislý, ale ne silně souvislý graf

V grafu chceme často hledat cestu minimální délky mezi dvěma uzly u a v. Označme délku této cesty jako vzdálenost $d(u, v)$ a definujme, že pokud taková cesta neexistuje, pak $d(u, v) = \infty$. V grafu se nesmí vyskytovat kružnice záporné délky, protože potom by bylo možné opakovaným průchodem takové kružnice minimalizovat délku cesty pod libovolnou hodnotu. Pro nalezení cesty minimální délky existují dva hlavní algoritmy:

- **Dijkstrův**: Nelze jej použít, pokud se v grafu vyskytují záporně ohodnocené hrany. Má kvadratickou časovou složitost.
- **Floyd-Warshallův**: Lze použít i pro záporně ohodnocený graf. Pokud v daném grafu existuje kružnice záporné délky, dokáže ji odhalit. Má kubickou složitost.

Dijkstrův algoritmus je následující (ohodnotí všechny uzly nejkratší vzdáleností k počátečnímu uzlu, z čehož lze potom snadno zpětně zrekonstruovat nejkratší cestu z počátečního uzlu k jakémukoliv jinému uzlu tak, že jdeme zpětně přes minimálně ohodnocené sousedy):

1. Počáteční uzel ohodnoť 0, ostatní ∞. Označ všechny uzly jako *unvisited*.
2. Počáteční uzel označ *current*.
3. Označ *current* jako *visited* a pro *current* vypočti vzdálenost všech sousedních uzlů *u* jako *ohodnocení*(*current*) + *ohodnocení_hrany_k*(*u*).
4. Pokud nezbývá žádný *unvisited* uzel, skonči, jinak vyber ze všech *unvisited* uzlů ten s nejmenším ohodnocením, označ jej *current* a jdi na krok 3.

Floyd-Warshallův algoritmus má pro graf s N uzly vždy N iterací je následující (předpokládá pouze nezáporné hrany):

1. inicializuj N × N matici (min. vzdáleností každého uzlu ke každému) A_0: $A_0[i][j]$ = vzdálenost po hraně nebo ∞
2. inicializuje pomocnou N × N matici P_0: $P_0[i][j] = j$
3. pro k = 0 až N - 1:
    1. pro každé *i j*:
        1. $d := A_{ij}^{j-1} + A_{jk}^{j-1}$
        2. pokud $A_{ik}^{j-1} > d$
            1. $A_{ik}^{j} := d$, $\quad P_{ik}^{j} := P_{ij}^{j-1}$
            2. jinak $A_{ik}^{j} := A_{ik}^{j-1}$, $\quad P_{ik}^{j} := P_{ik}^{j-1}$

Případné záporné hrany se projeví zápornou hodnotou na diagonále matice.

Úplný graf je graf, který má každé dva uzly propojené. Označuje se K*n*, kde *n* je počet vrcholů.

16. Postrelační SŘBD (definice, vymezení problematiky a specifik pro O-R, prostorové, temporální, XML a deduktivní DB).

Relační databázový systém je systém založený na pojmu matematické relace. Těmito relacemi se myslí tabulky samotné (záznam v tabulce je jedna *n*-ární relace mezi doménami sloupců, kterých je *n*), nikoli relace mezi tabulkami! **SŘBD** (**systém řízení báze dat**) je systém spravující databázi, komunikující s uživatelem, programy apod. **DML** (**data manipulation language**) je jazyk pro výběr, vyhledávání a vkládání dat, např SQL. Formálně je relační tabulka na **doménách** (množinách skalárních hodnot téhož typu, složená doména = doména složená z více jednoduchých domén) D_1, … D_n dvojice:

$$R = (\rho, D),$$

kde ρ je schéma relace (názvy sloupců, ve tvaru D_1: A_1, … D_n: A_n) a $D \subseteq D_1 \times D_2 \ldots \times D_n$ je n-ární (*n* je tzv. **stupeň** relace) relace mezi doménami, tzn. samotné řádky tabulky (neexistují duplicitní záznamy ani nezáleží na jejich pořadí). Počet řádků, tzn. |R|, se nazývá **kardinalita** relace. Tabulka je způsob znázornění relace. **Přimární klíč** je atribut (I složený), který jednoznačně určuje řádek tabulky. **Kandidátní klíč** je atribut, který může být primárním klíčem – musí být vždy neprázdný a musí být unikátní pro každý řádek.

DB lze tzv. **normalizovat**, což je proces převodu tabulek do formátu, který zabraňuje redundanci, předchází chybám apod. Existujě několik tzv. **normální forem** relací -

- nultá (0NF) – Tabulka má alespoň jeden sloupec, který může obsahovat více druhů hodnot.
- první (1NF) – Neexistují sloupce se složenými doménami (např. místo adresa, máme: město, číslo, PSČ).
- druhá (2NF) – 1NF + všechny atributy jsou závislé na celém klíči (tedy řeší se pouze u složených klíčů).
- třetí (3NF) – 2NF + neexistují závisloti mezi neklíčovými atributy.
- čtvrtá (4NF)
- pátá (5NF)

Postrelační DB systém je systém, který už nevystačí pouze s relačním schématem a bez rozšíření na implementační úrovni by byl neefektivní. Postřelační DB není DB, která má přidanou funkcionalitu pouze na aplikační úrovni. Mezi Postrelační DB patří objektové DB, deduktivní DB, prostorové DB, časové DB, multimediální DB a další.

Objektové DB jsou DB umožňující modelovat perzistentní objekty ve smyslu OOP. Neexistuje zde jasný standard jako je SQL u relačních DB, často se používají klasické OO jazyky. Vyskytují se zde třídy, atributy, zapouzdření, polymorfismus apod., každý objekt má jednoznačné OID (object ID). Snadno se zde vytváří vztahy typu M:M (narozdíl od relačních DB).

Objektově-relační DB jsou DB propojující relační a objektové DB. DB se skládá z tabulek, které se ale chovají více jako objekty, mohou mít operace, mají obdobu OID. O-R jazykem je např. SQL 1999.

Prostorové DB spravují data vztahující se ke geometrickému prostoru (2D, 3D). Používají se v geografii, astronomii, medicíně, molekulární biologii apod. Ukládají se v nich tzv. **entity** (bod, úsečka, oblast, graf, ...), ale i informace o prostoru samotném, obsahují prostorové datové typy a operace s nimi (průniky, překrytí, nejbližší soused, ...). Problémem je diskrétní povaha prostoru reprezentovaného v počítači pomocí floating point čísel – např. máme li dvě analyticky zadané úsečky, jejich průsečík pravděpodobně nebude ležet přesně na mřížce prostoru. Toto se řeší použitím:

- ***d*-simplexů –** *d*-simplex je nejmenší objekt rozměru *d* (0 – bod, 1 – úsečka, 2 – trojúhelník, ...). Složitější objekty se skládají ze simplexů.
- **deskriptorů** (**realms**, tzv. úplný popis) **–** deskriptor popisuje danou entitu tak, že:
  1. Každý bod nebo koncový bod útvaru (např. úsečky) je bodem sítě prostoru.
  2. Žádný vnitřní bod útvaru (např. úsečky) není bodem sítě. (Zadáme-li úsečku, která prochází bodem sítě, pak se rozdělí na dvě.)
  3. Žádné dva útvary (např. úsečky) nemají průsečík ani se nepřekrývají. (Zadáme-li dvě úsečky s průsečíkem, opět se rozdělí, avšak průsečík nemusí být bodem sítě - vybere se pro něj nejbližší bod. Tímto vznikají nepřesnosti, které

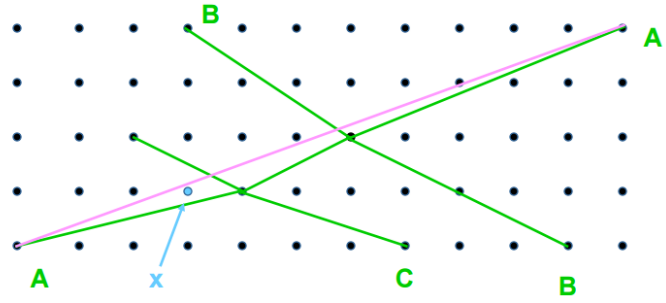


Illustration 39: problém s deskriptory - zelené úsečky představují reprezentaci pomocí deskriptorů, růžová úsečka je původní úsečka, modrý bod leží v jednom případě nalevo a v druhém napravo od úsečky, toto se dá řešit pomocí obálek (ty zajistí, že úsečka nikdy nepřeskočí žádný bod mřížky).

se řeší např. pomocí obálek.)

V prostorových DB existují formální algebry pro manipulaci oddílů (regionů). Tato algebra zahrnuje operace jako např. projekce (obdoba SELECT v SQL) nebo selekce (obdoba WHERE v SQL) apod. Příkladem takové algebry je např. algebra **ROSE** (robust spatial extension). Ta umožňuje např. zjišťovat míru vnoření oddílů, míru disjunkce apod. K reprezentaci regionů používá R-plochy (matematická definice bodů a hran takových, že tvoří region). Datové typy v prostorové DB můžou být bod, úsečka, polygon (nevyplněný), oblast (vyplněná) apod.. Operacemi mohou být např. překrytí (průnik), Voronoi (každému bodu prostoru přiřadí nejbližší bod z množiny) apod. Př prostorového dotazu:

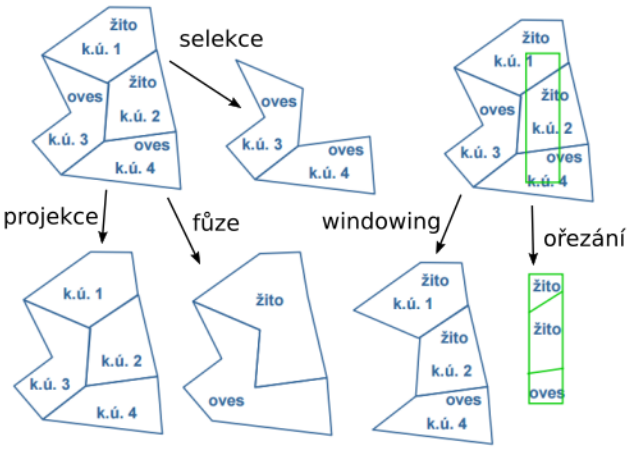


Illustration 40: operace algebry pro manipulaci oddílů

SELECT * FROM rivers WHERE route intersects window

Temporální DB jsou DB dávají důraz na práci s časem. Používají se např. v bankovnictví, katastrech, medicíně apod. Typicky se pracuje s časem:

- **platnosti** – časový interval, během něhož je fakt pravdivý v reálném světě
- **transakce** – čas, v němž byl fakt zaznamenán v DB

Tyto DB podporují časové dat. typy, čas. dotazy, omezení apod.

Deduktivní DB jsou DB, které umí vyvozovat nová fakta z fakt uložených v DB. Typicky se zde jako dotazovací jazyk používá **datalog**, což je deklarativní jazyk založený na prologu (využívá predikátovou logiku, tzn. vyžívají se inferenční pravidla - modus ponens a zobecnění). DB obsahuje jak samotná fakta, tak i odvozovací pravidla. Způsob dedukce faktů může být buď **sémantický** nebo **syntaktický** (aplikuje inferenční pravidla všemi možnými způsoby – zespoda-nahoru, shora-dolů). Deduktivní DB se používají např. v burzovních aplikacích, molekulární biologii apod. Některé rozdíly oproti klasickému logickému programování v prologu jsou:

- Deduktivní DB jsou méně expresivní (avšak pořád více expresivní než relační model).
- V deduktivních DB nezáleží na pořadí pravidel.

Příkladem dotazu v deduktivní DB může být (J = jméno, P = plat):

```
zamestnanci_s_vysokym_platem(J) :- zamestnanci(J,P), P > 30K
```

XML DB je DB s podporou XML – tzn. dovoluje data specifikovat, a někdy i ukládat, v XML formátu. XML DB patří mezi DB zaměřené na dokumenty, které patří mezi NoSQL DB. Motivací pro XML DB je velké rozšíření tohoto formátu a má-li pro něj DB přímou podporu, je snadnější s ním pracovat.

Multimediální DB je DB s podporou pro ukládání různých médií – text, grafika (vektory), obrázky, zvuk, video, … Musí se být podpora pro práci s médii, tzn. např. vyhledávání obrázků podle podobnosti, nestačí jen možnost ukládat multimediální soubory.

Zkratka **ACID** (atomicity – transakce jsou atomické, consistency – DB přechází z konzistentního stavu do jiného konzistentního stavu, isolation – současně prováděné transakce se chovají, jakoby byly vykonávány jedna po druhé, durability – provedená transakce je trvalá, i po vypnutí systému) označuje vlastnosti DB transakce.

17. Metody indexování bodových a plošných útvarů - typicky obalujících hyperobdélníků - v prostorových DB (principy, metody, postupy, ke každé třídě typický algoritmus, minimálně (adaptivní) kD strom, Grid File, R strom, R+ strom).

Index je datová struktura sloužící k akceleraci vykonávání dotazů. U prostorových DB je indexace důležitá pro spoustu dotazů – bez indexu by se např. dotaz na nejbližšího souseda daného bodu musel řešit hrubou silou, což je velmi neefektivní. Index jsou většinou založeny na nějaké aproximaci (diskrétní – mřížka, spojitá – obalovací tvary). Mezi typické akcelerovatelné dotazy patří: nejbližší soused (nearest neighbour) bodu (k-D tree, quad tree, ...), průnik/kolize (obalující tvary, quad tree, …), obsažení, určení vzdáleností apod. Následuje výčet a vysvětlení některých důležitých indexovacíxh metod:

- 1D
  - **lineární hashování** – Rozšíření klasické hashovací tabulky – tato tabulka je dynamická a přidává nové sloty, když je v některém hodně záznamů. Díky tomuto lze rozdělit prostorový interval <a,b) na buňky, které jsou mapovány do hashovací tabulky.
  - **adaptivní hashování** – Podobné jako lineární hashování, ale pro přetokové oblasti se používá adresář (slotted page, záznamy plus metadata).
  - **B-stromy** – Samovyvažující se strom, který udržuje data seřazená v listech a umožnuje přístup, mazání a vkládání v log. čase.
- *n*D – Dá se použít mapování do 1D, avšak to nezachovává sousednost, proto se používají specializované indexy.
  - **k-D tree** – Speciální případ BSP stromu, staví strom nad *k*-dimenzionálním prostorem, kde každý uzel je bod v tomto prostoru. Každý vnitřní uzel je bod definující hyperplochu (2D – přímka, 3D – rovina, …) v jednom rozměru prostoru (ty se s hloubkou v stromu pravidelně střídají, např. ve 2D – x, y, x, y, …). Uzly v levém podstromu pak představují body ležící nalevo od této hyperroviny, ostatní jsou vpravo od ní. Listové uzly jsou samotné body prostoru uložené v tomto indexu. k-D strom umožňuje rychlé hledání nejbližších sousedů a rychlé vyhledání bodů v určité oblasti. Nevýhodou je problém s mazáním a ukládání pouze bodů.

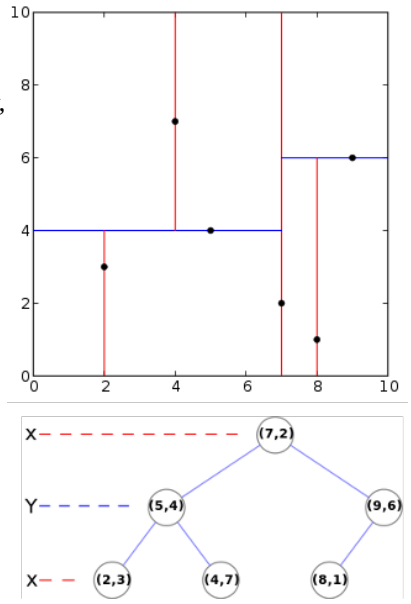


Illustration 41: příklad k-D stromu

  - **adaptivní k-D strom** – Odstraňuje nevýhodu závislosti na pořadí vkládání, dělící hyperlochy nemusí procházet body (=> data jsou pouze v listech), snaží se sestavovat vyvážený strom (vkládá dělící plochy tak, aby na každé straně bylo přibližně stejné množství bodů), vhodné pro statická data) nebo bin-tree.
  - **Bin tree** – Rekurzivně dělí podprostor na hyperkrychle stejné velikosti, až každá obsahuje max. jeden bod.
  - **BSP tree** (binary space partitioning) – Rovněž dělí prostor hyperplochami na poloviny, avšak hyperplochy nemusí procházet body (jako u adaptivního k-D stromu) a mohou být navíc libovolně natočené. Prostor dělí, dokud počet bodů v podprostoru neklesne pod určitou hodnotu.
  - **Quad-Tree** – Jako k-D strom, ale v každé úrovni dělí na 2^n podstromů. Např. 3D prostor pokaždé rozdělí na 8 krychlí (ne nutně stejně velkých), každou z nich na dalších 8 podkrychlí atd. Dělí do určitého počtu elementů na podstrom. Existují dvě varianty: point (dělí pomocí bodů) a region (dělí rovnoměrně).
  - **Grid File** – Adaptivní hashovací metoda pro ukládání bodů, dělí prostor n-rozměrnou mřížkou (ne nutně pravidelnou). Je zde **adresář** (je velký, uložený na disku), který mapuje každou buňku mřížky (nebo i více buněk) do datové jednotky (tzv. **bucket**), kde jsou body uloženy (už sekvenčně). Vkládání bodů může způsobit přetečení datové jednotky – pak se mřížka dělí další hyperplochou, což může být **nelokální** změna (může ovlivnit i vzdálené body). Nevýhodami jsou vyšší paměťové nároky (adresář) a overhead při vkládání a mazání bodů (úprava dat. jednotek). Výhodami jsou jednoduchost a pouze dva přístupy na disk (adresář plus dat. jednotka).

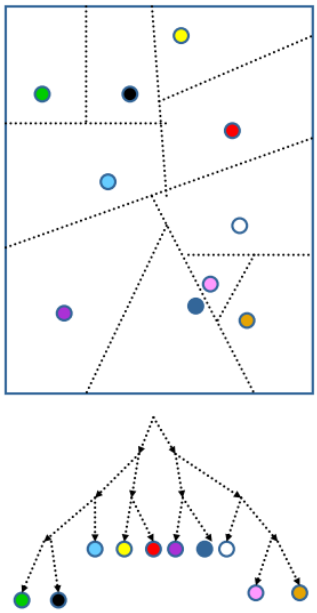


Illustration 42: BSP tree příklad

- **EXCELL** – Jako Grid File, ale dat. jednotky mají stejnou velikost, plus jsou zde přetokové stránky.
- **two-level Grid File** – Má druhou úroveň dělení mřížky, změny jsou častěji lokální, ale ne vždy.
- **Twin Grid File** – Dva nezávislé Grid File, když data v jednom (primární) přetečou, ukládají se do druhého, tzn. předchází se dělení mřížky.
- BANG File – Jako Grid File, strom je vyvážený, dat. Jednotky se překrývají, ...
- **R-tree** (rectangle-tree) – Dokáže indexovat i celé tvary, obaluje je minimálními ohraničujícími (hyper)obdelníky. Podobně jako B-strom jde o vyvážený strom organizující data do stránek (listy). Stránka má maximální možný počet dat, který se do ní vejde.
- **R+ tree** – Kompromis mezi R-tree a kD tree. Obdelníky na stejné úrovni stromu se nikdy nepřekrývají a objekt je vložen do více uzlů, je-li to potřeba. Daný bod prostoru je tedy pokryt maximálně jedník obdelníkem a bodové dotazy jsou tak rychlejší, avšak strom může být větší a je náročnější ho udržovat.

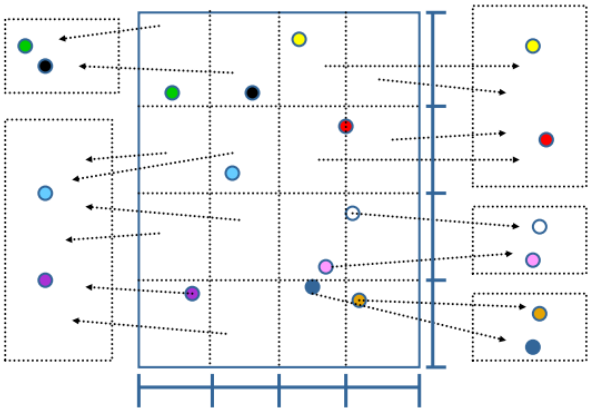


Illustration 43: Grid File - příklad

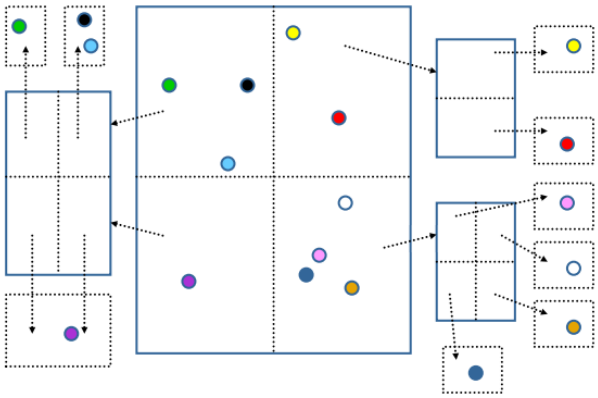


Illustration 44: two-level Grid File - příklad

Pro přístup k bodům existují další algoritmy jako Buddy Tree, K-D-B Tree (kD tree + B-Tree), hB Tree, LSD Tree (local split decision, vyvážený, externí dat. stránky, ...) apod.

Alg. pro nalezení objektů pokrývajících bod v R-tree je násl.:

```
GetObjectsOnPoint(point: P): set ob Objs:
  result = emptySet()
  rects = RTreeTraversal(root, P)
  for r in rects:
    for obj in r:
      if contains(obj,P):
        result += obj
  return result

RtreeTraversal(node: N, point: P):
  result = emptySet()
  if N is leaf:
    return N
  for obj in N:
    if contains(obj,P):
      result += RtreeTraversal(N,P)

  return result
```

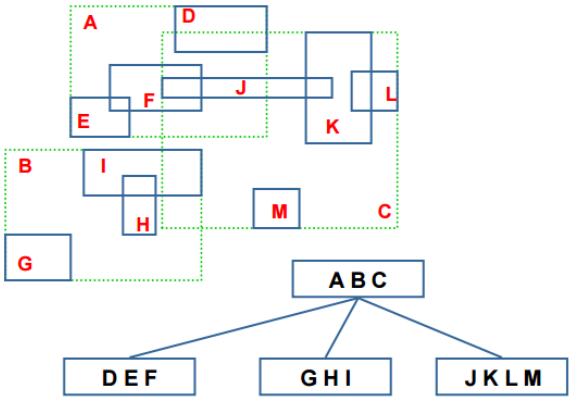


Illustration 45: R-tree příklad

18. Temporální DB (modely času, generičnost dotazu a shlukování, [integritní] omezení v historii).

Temporální DB jsou DB dávají důraz na práci s časem. Používají se např. v bankovnictví, právu, katastrech, medicíně apod. Typicky se pracuje s časem:

- **platnosti** – časový interval, během něhož je fakt pravdivý v reálném světě
- **transakce** – čas, v němž je fakt zaznamenán v DB a může být získán

Pracuje-li DB s časem platnosti i transackce, nazývá se **bitemporální**. Tyto DB podporují časové dat. typy, čas. dotazy, omezení apod. Čas měří hodiny v jednotkách zvaných **chronony**. Časová doména může potom být:

- diskrétní – přirozená/celá čísla, okamžiky mají předchůdce/následníky
- racionální čísla
- spojitá – reálná čísla
- nelineární – Vyskytuje se např. v systémech pro správu verzí (git, …).

Typicky používáme lineárně uspořádanou neomezenou časovou doménu (T, <). Dále máme relační DB nad (D,ρ) (kde D je množina domén a ρ schéma DB), nad níž chceme implementovat temporální podporu. Temporální DB je tedy funkce: $T \rightarrow (D,\rho)$.

Některé DB modely, podle nichž můžeme temporální DB implementovat, jsou (matematicky jsou ekvivalentní):

- **snapshot model** - Zavádíme **snímky** (**snapshots**) DB z (D,ρ) v různých okamžicích (ne nutně všech) z časové domény. Máme potom různé DB, každou s vlastním **časovým razítkem** platnosti. Zavádíme potom snímkovou tabulku, která uchovává snímky DB. Tneto snapshot model není vhodný pro některé dotazy - např. nalezení všech okamžiků, kdy byla splněna určitá podmínka (musel by se kontrolovat každý snímek).
- **model s časem platnosti** – Jednotlivé záznamy v DB mají razítka s časem platnosti. Toto je vhodné pro dotazy na **historii** (posloupnost stavů DB, např. titul: bc., ing., PhD). Čas je jenom další typ v DB.

Mezi časové datové typy v temporálních DB patří:

- **okamžik** – bod na časové ose
- **množina okamžiků**
- **časový úsek** (**interval**) – doba mezi dvěma okamžiky, někdy může být nekonečný
- **trvání** – časový úsek se známou délkou, ale s neznámým začátkem a koncem

Shlukování (coalescing) je způsob spojování intervalů. Pokud např. DB obsahuje dva záznamy – (hodnota1, [1,6]), (hodnota2, [5,7]) – můžeme je shlukovat do jednoho – (hodnota1, [1,7]). Toto se provádí přímo v DB nebo při dotazech. Říkáme, že jednorozměrná temp. DB obsahuje shluky (je shlukovaná), právě pokud každý fakt je spojen s maximálně konečným počtem nepřekrývajících se intervalů. Shlukování je třeba zaručit, pokud chceme vykonávat ne-logické operace. Pro více rozměrů je shlukování nejednoznačné.

Dotaz je **generický**, pokud je jeho výsledek nezávislý na způsobu uložení dat v DB. Mějme např. DB_1 = {(a, [0,2]), (a, [1,3])} a DB_2 = {(a,[0,3])}. DB_2 je tedy jenom shlukovaná DB_1. Dotaz

$$\exists i,j: \exists x: (R(i,x) \wedge R(j,x) \wedge i \neq j)$$ (Existují dva různé intervaly, v nichž platí *x*?)

není generický, protože v DB_1 platí, ale v DB_2 ne. Typicky chceme, aby byly dotazy vždy generické. Toto řeší právě shlukování.).

U temp. DB je potřeba mít indexování v čas. oblasti. U indexů lze využít znalosti o temp. DB – např. temp. DB jsou většinou “append only”, intervaly jsou neprázdné, nejčastěji se dotazujeme na přítomnost apod. Využívá se např. R-tree.

Integritní omezení v temp. DB jsou zadávána pomocí temporální logiky na datech s časovým razítkem. Využívají se zde operátory jako (výroková logika):

- $\Diamond x$ – *x* platí někdy v budoucnosti
- $\blacklozenge x$ – *x* platí někdy v minulosti
- $\boxdot x$ – *x* platí vždy v budoucnosti
- $\blacksquare x$ – *x* platí vždy v minulosti
- $\circ x$ – *x* platí v příštím kroku (pro diskrétní čas)
- $\bullet x$ – *x* platí v předchozím kroku (pro diskrétní čas)

emp. omezení jsou formule prvního řádu temp. dotazovacího jazyka. (logika prvního řádu je predikátová logika, která se může dotazovat na individuální objekty, např. $\forall x$, logika druhého řádu se může dotazovat i na vztahy mezi objekty).

Temporálním jazykem je např. TQUEL.

19. Objektově-relační databáze (charakteristika, porovnání s relačními, podpora v SQL:1999 a SQL:2003).

Objektově-relační DB (OR DB) jsou DB propojující relační a objektové DB. DB se skládá z tabulek, které se ale chovají více jako objekty, mohou mít operace, mají obdobu OID. O-R jazykem je např. SQL:1999. Následující tabulka nabízí srovnání různých typů DB:

	relační DB	**objektová DB**	**OR DB**
model dat	relační, DB = tabulky, málo dat. typů	objektový (třídy, atributy, ...), není standard, ADT, objekty mají jednoznačné OID, DB = perzistentní objekty	tabulky obohacené o OO, obecnější relace, ADT, obdoba OID
dotaz. jazyk	SQL, deklarativní	OOP jaz. (Java, ...), není standard.	SQL-1999
výpočetní model	navigace v tab. pomocí kurzoru	navigace pomocí OID referencí	navigace v tab. pomocí kurzoru i referencí
výhody	formalizovány, jednoduché	M:M vztahy – snadné, DB a aplikace mají stejný jazyk	M:M vztahy – snadné (není potřeba join)
nevýhody	M:M vztahy vyžadují zvláštní tabulky a join operaci při výběru	složitější, není standard	složitější

ADT (**abstraktní datový typ**) je datový typ zapouzdřující možné hodnoty a operace nad nimi.

Jazyk **SQL:1999** je OR jazyk. Nabízí např.:

- **strukturované uživatelské typy** (**UDT**) – obdoba třídy, má atributy, metody, podporuje polymorfismus, jednoduchou dědičnost apod. Např.:
  ```
  CREATE TYPE zamestnanec_t UNDER osoba_t
    AS (os_cislo INTEGER, plat REAL) INSTANTIABLE NOT FINAL
    REF (os_cislo)
    INSTANCE METHOD Zvys_plat(abs_proc BOOLEAN, castka REAL)
      RETURN REAL
  ```
  Typ sloupce tabulky může být UDT. Na atributy se lze odkazovat tečkovou notací (např. `zam.plat`).
- **typované tabulky** – Tabulka, jejíž řádky jsou určitého UDT typu. Např.:
  ```
  CREATE TABLE Zamestnanci OF zamestnanec_t
  ```
- **hodnoty typu REF** – Hodnota identifikující řádek (konkrétní “objekt”) v typované tabulce.

SQL:2003 je novější revize jazyka SQL:1999. Nově nabízí např. podporu pro XML, generátory sekvencí, nové příkazy apod.

Objektově-relační mapování je technika, kdy se objekty OOP jazyka mapují na čistě relační DB.

20. XML databáze (typy XML dokumentů, klasifikace úrovně podpory, XML typ v SQL a jeho použití).

XML DB je DB s podporou XML – tzn. dovoluje data specifikovat, a někdy i ukládat v XML formátu. Většina DB podporuje XML alespoň tak, že jej umožňuje do DB ukládat (tzv. **XML enabled** DB). Je-li podpora vnitřní (DB "rozumí" struktuře XML, pracuje s DOM apod.), mluvíme o **nativní podpoře XML**. Motivací pro XML DB je velké rozšíření tohoto formátu a má-li pro něj DB přímou podporu, je snadnější s ním pracovat. Mohou existovat wrappery umožňující provádět SQL dotazy nad XML DB. Mezi nativní XML DB patří např. open source DB eXist. XML Databáze je většinou namísto relačního modelu vystavěna nad modelem XML, tzn. DOM.

SQL zahrnuje podporu pro XML v SQL:2003 (tzv. SQL/XML). Tato podpora zahrnuje:

- Zavádí datový typ XML (XMLType) umožňující ukládat "XML hodnoty" (pouze Oracle DB).
- Konstruktory XML hodnot, např.:
  - XMLElement – Ze jména a atributů a obsahu vytvoří XMLType hodnotu.
  - XMLForest – Vytvoří skupinu XML elementů z názvů sloupců relační tabulky.
  - XMLConcat – Sloučí dvě a více XMLType hodnot do jedné.
- Mapování SQL na XML a naopak.

Oracle server od roku 2002 podporuje tyto věci plus některé další (XPath, XML Schémata, ...).

Middleware je program mezi DB a aplikací, který převádí XML do/z formátu DB – nejedná se o nativní podporu.

Problematika XML DB zahrnuje:

- ukládání XML do DB, řešení:
  - ukládání přímo jako textu do relační tabulky, používané dříve, ztrácí výhody XML
  - ukládání jako **CLOB** (character large object – velké množství textu uložené jinde než přímo v tabulce, v tabulce je jen odkaz), podobné jako předchozí řešení, špatná práce s částmi dokumentu
  - ukládání pomocí XMLType v O-R DB – efektivní, protože XML dokument se rozparsuje na objekty a tabulky, dá se efektivně pracovat s částmi dokumentu
- dotazování:
  - XPath (popř. i XSLT pro transformaci výsledku) – Např. ZAKAZNICI/* vybere všechny potomky uzlu ZAKAZNICI.
  - XQuery – Funkcionální dotazovací jazyk.
  - XSQL – Jazyk kombinující XML a SQL.

21. Grafická knihovna OpenGL: vykreslovací řetězec (funkční bloky, možnosti nastavení), frame buffer, stencil buffer. PGR

OpenGL (Open Graphics Library) je multiplatformní API pro vykreslování 2D a 3D vektorové grafiky pomocí GPU. Specializuje se pouze na grafiku, nikoliv na zvuky, vstupy z klávesnice myši apod., na ty existují knihovny:

- **GLM** – Knihovna pro matematiku (matice, pomocné funkce), napodobuje jazyk GLSL (existují zde stejné funkce apod.).
- **GLUT** (gl utility toolkit) – Knihovna pro platformě nezávislou práci s okny.
- **GLEW** (gl extension wrangler) – Knihovna pro správu rozšíření OpenGL.
- SDL – Řeší okna, klávesnici, zvuky atd.
- OpenAL – Jako OpenGL, ale na zvuky.
- ...

Konkurentem je Direct3D od Microsoftu, popř. nové API **Vulkan**, které je však mnohem složitější, aby dosáhlo lepšího výkonu.

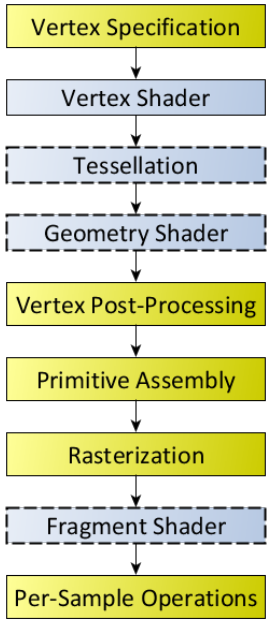


Illustration 46: OpenGL vykreslovací řetězec

Jádrem OpenGL je tzv. **vykreslovací řetězec** (**pipeline**), což je cesta, kterou putují data během vykreslování. Některé části řetězce jsou programovatelné – jejich programy se potom nazývají **shadery**. Programovatelnými bloky jsou především:

- **vertex shader** – Provádí se pro každý vrchol, může upravovat jejich vlastnosti, ale nemůže vrcholy zahazovat nebo vytvářet nové.
- **tesselation shader** – Provádí teselaci, tzn. rozdělování povrchu na trojúhelníky. Používá se při pokročilých metodách, lze jej snadno vynechat. Skládá se ze tří částí, z nichž dvě jsou programovatelné (tessellation control a tesselation evaluation).
- **geometry shader** – Pracuje s primitivy (trojúhelníky, triangle fany, body apod.), může modifikovat geometrii, tzn. upravovat, vytvářet nebo zahazovat vrcholy, trojúhelníky apod. Rovněž se používá spíše při pokročilých metodách.
- **fragment shader** – Provádí se pro každý **fragment** (bod, který se může ve výsledku stát pixelem, ale taky nemusí, pokud je např. překryt jiným pixelem). Jeho výstupem je barva fragmentu. Fragment může být zahozen, ale nemůže být měněna jeho pozice nebo vytvářen nějaký jiný fragment.

Compute shader je od OpenGL 4.5 nový shader, který slouží k obecným výpočtů na GPU, podobně jako např. OpenCL nebo CUDA. Není součástí vykreslovacího řetězce a musí se volat samostatně. Dá se využít pro výpočet fyzikálních simulací, procedurální generování, výpočet pomocných struktur a mnoho dalšího.

OpenGL hodně pracuje s **buffery**, tzn. s pamětmi na GPU, které je možné alokovat a potom je používat např. pro předávání hodnot z CPU do GPU a naopak.

OpenGL pracuje jako stavový stroj, tzn. funkcemi se nastaví stav, který ovlivní následné vykreslování.

OpenGL je spravováno neziskovým konsorciem **Khronos Group**, jehož součástí jsou firmy, jako např. Nvidia, AMD, Apple, Pixar, Google apod. Konsorcium vydává specifikace nových verzí. Vývoj OpenGL probíhá do značné míry díky **rozšířením** (**extensions**). Rozšíření je funkcionalita, kterou může některý výrobce do OpenGL přidat a pokud se ujme, bude konsorciem zahrnuto v dalším oficiálním standardu. Rozšíření je identifikováno řetězcem, který má předponu udávající typ či výrobce (NV – Nvidia, INTEL – Intel, WIN – Microsoft, ARB – oficiální, ...). Pomocí knihoven, jako je GLEW, lze zjistit, která rozšíření jsou na dané platformě k dispozici, a případně je využít. Příklady rozšíření jsou GL_ARB_texture_cube_map, GL_NV_fog_distance, GLX_NV_copy_imag apod.

Transform feedback je jednotka umožňující nahrát data zpět do bufferu po tom, co byla zpracována vertex shaderem, popř. geometry shaderem.

Frame buffer je datová struktura OpenGL sloužící k tomu, aby se do ní vykreslovalo. Přistupuje se k ní pomocí frame buffer objektu (FBO). Frame buffer má různé attachmenty (barva, hloubka, stencil, ...), k nimž se mohou připojit textury, do nichž se potom renderuje, pokud je frame buffer aktivní. Vykreslováním do textur lze dosáhnout různých efektů, jako např. deferred shadingu apod.

Stencil buffer je dvourozměrné pole celočíselných hodnot sloužící k realizaci "šablony" při vykreslování. Příkladem využití je např. maskování při vykreslování stínů pomocí shadow volumes nebo při vykreslování zrcadlových odrazů na

povrchu nějakého tělesa. Při práci se stencil bufferem využíváme především tyto funkce:

- glStencilFunc – Nastaví funkci a referenční hodnotu, které určí, zda stencil test uspěl nebo selhal.
- glStencilOp – Nastaví akce, které se provedou při třech různých situacích (stencil fail, stencil pass depth fail, stencil fail depth fail). Akce se týká stencil bufferu, může se jednat např. o nastavení určité hodnoty, inkrementace apod.

Práce se stencil bufferem probíhá typicky takto: nastavi se glStencilFunc na GL_NEVER a glStencilOp nastaví zápis určité hodnoty na test fail, pak se vykreslí scéna, čímž se do stencil bufferu zapíše daná hodnota tam, kde se renderují objekty – vytvoří se šablona. Potom se nastaví glStencilFunc na porovnávací funkci a proběhne další vykreslení – vykreslení proběhne jenom tam, kde stencil test projde.

22. Afinní 3D transformace, kamera, projekce, skládání transformací. PGR

Lineární transformace je transformace, která zachovává vektorové sčítání a násobení skalárem. lineární transformaci z *n*rozměrného prostoru do *m*rozměrného prostoru lze reprezentovat maticí $n \times m$ a tuto transformaci potom provádět maticovým násobením. Mezi lineární transformace patří např. změna měřítka, rotace, překlopení, zkosení, ale ne např. posun (ten vyžaduje přičítání konstantní hodnoty, lineární transformace umožní jen přičítat násobek druhého vektoru). Každá lineární transformace je afinní. Lineární transformace mapuje úsečky na úsečky (nebo na nulu). Taková transformace může a nemusí být revertibilní. Formálně: transformace $f(x)$, kde x je vektor, je lineární, právě když platí:

- $f(x + y) = f(x) + f(y)$
- $f(ax) = af(x)$

Afinní transformace je transformace, která zachovává kolinearitu – tzn. body, které leží na přímce, leží na přímce i po transformaci – a poměry vzdáleností – tzn. střední bod úsečky zůstane středním bodem úsečky i po transformaci. Rovněž rovnoběžné úsečky zůstanou po transformaci rovnoběžné, ale úhly obecně zachovány nejsou. Tyto transformace se dají v *n*rozměrném prostoru relizovat pomocí matic $(n + 1) \times (n + 1)$, které mají spodní řádek roven vektoru (0, 0, ..., 1), pravý sloupec roven vektoru posunu, a pokud použijeme vektory, které mají $n + 1$ komponent, přičemž poslední je rovna 1 (toto ještě nejsou homogenní souřadnice, u těch může být poslední komponenta libovolná a dělí se jí ostatní komponenty). Mezi afinní transformace patří např. posun nebo zrcadlení. Ne každá afinní transformace je lineární. Na afinní transformaci lze nahlížet jako na lineární transformaci plus posun.

Říkáme, že vektor x je **lineární kombinací** vektorů $x_0 \dots x_k$ právě tehdy, když existuje $k$tice taková, že

$x = \alpha_0 x_0 + \dots \alpha_k x_k$

Afinní kombinace je podtřída lineárních kombinací, která má součet všech koeficientů (tedy $\alpha_0 + \dots + \alpha_k$) roven 1.

Homogenní souřadnice v *n*rozměrném prostoru mají $n + 1$ komponent (např. 4 ve 3D). Pro převod z homogenních souřadnic do kartézských souřadnic se prvních n komponent vydělí poslední komponentou. Každý bod prostoru má tedy v homogenních souřadnicích nekonečně mnoho reprezentací (např. [10,8,6,2] = [5,4,3,1]). Ve 3D grafice nastavujeme v homogenních souřadnicích poslední komponentu vektorů na 0 (tzn. při maticových operacích se na nich neprojeví posun, to je výhodné např. při transformaci normál, což jsou vektory), u bodů (vrcholů modelů) na 1.

Typicky rozlišujeme dva druhy projekcí:

- **paralelní** (**ortographic**) – Ignoruje změnšování objektů s jejich vzdáleností, hodí se tedy např. pro strojírenské nebo architektonické aplikace, kdy je výhodné zachovávat při zobrazování měřítka. Název je odvozen od faktu, že se k promítání používají rovnoběžné paprsky. V homogenních souřadnicích vypadá matice pro tuto transformaci takto (jednoduchá verze, v OpenGL by se např. musely vzít v potaz roviny near a far):

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

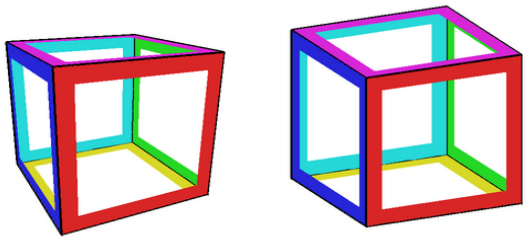


Illustration 47: perspektivní (vlevo) vs paralelní projekce

- **perspektivní** – Bere v potaz perspektivu, tzn. vzdálené objekty se změnšují – tato projekce je věrnější realitě a používá se proto např. ve hrách, je však o něco komplikovanější. V podstatě se však jedná pouze o dělení souřadnicí z. Jednoduchá verze projekce v homogenních souřadnicích vypadá takto:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{-1}{near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformace reprezentované maticemi se dají skládat pomocí maticového násobení, využívá se asociativity, tzn. $T_1T_2...T_nP = (T_1T_2...T_n)\,P$. Získáme tak jedinou matici reprezentující několik transformací. Při násobení záleží na pořadí – máme-li postupnou řadu transformací T_1, ..., T_n, pak složení můžeme provést jejich vynásobením (záleží zde také na tom, jestli používáme násobení vektor-matice (řádkové vektory) nebo matice-vektor (sloupcové vektory), následující se týká typu matice-vektor):

- **zprava doleva** – Transformujeme body vzhledem k stacionárnímu systému (např. Kameře).
- **zleva doprava** – Transformujeme systém.

Celkově probíhá vykreslení scény takto:

1. Máme souřadnice vrcholů zobrazovaného objektu O (tzv. **object space** souřadnice).
2. Objekt O je umístěn do scény pomocí transformační matice, kterou se násobí jeho vrcholy – souřadnice takto získané jsou v tzv. **world space**.
3. Celá scéna se nyní transformuje do prostoru kamery, tzn. násobí se další maticí – tím získáme tzv. **view space** souřadnice (v OpenGL x doprava, y nahoru, -z dopředu). Většinou máme k dispozici jednu matici, která provede transformaci do world space a view space najednou, tzv. model-view matici.
4. Provede se projekce, tzn. násobíme většinou maticí perspektivní projekce – tímto získáme výsledné tzv. **scren space** souřadnice vrcholů.

23. Osvětlení: způsob výpočtu, Phongův model, stínování, materiály. PGR

Phongův osvětlovací model (neplést s Phongovým stínováním) je empirický (na zkušenostech založený) model lokálního osvětlení (tzn. nepočítá globální osvětlení, tzn. stíny, propagaci světla apod). Pro daný bod povrchu udává, jak moc světla je odraženo. K tomu využívá 3 složky světla:

- **ambientní** – rozptýlené světlo, jedná se o konstantní složku
- **difúzní** – světlo rovnoměrně odražené do všech směrů, jeho intenzita závisí na úhlu dopadu světla
- **spekulární** – světlo odražené zrcadlově, tzn. do jednoho směru, jeho intenzita závisí na úhlu dopadu světla a úhlu vektoru k pozorovateli

Existují další osvětlovací modely, jako např. **Lambertův**, který počítá pouze ambientní a difúzní složku.

Výsledná intenzita Phongova osvětlení je dána rovnicí:

$$I = I_A + (L \cdot N) \cdot I_D + (R \cdot V)^e \cdot I_S$$

Kde I_A je ambientní intenzita, I_D je difúzní intenzita, I_S je spekulární intenzita, L je vektor směřující ke světlu, N je normála povrchu, R je vektor odrazu (odvozený z normály a vektoru L), V je vektor směřující k pozorovateli a e je spekulární exponent (ovlivňuje velikost spekulárních odrazů).

Stínování (**shading**) je způsob, jakým púoužíváme osvětlovací model k výpočtu barvy pixelů (nesouvisí s vykreslováním stínů!). Existují tři hlavní způsoby stínování:

- **ploché** (**flat**) – Barvu počítá pro každou stěnu modelu, osvětlovací model spočítá pro každou stěnu jen jednou, na základě normály stěny – rychlé, ale ne příliš kvalitní (hranaté stěny).
- **Goraudovo** – Osvětlovací model počítá pro každý vrchol modelu (podle normály v tomto vrcholu), a spočítané barvy interpoluje – pomalejší a kvalitnější než flat shading, ale pořád nepřesné, obzvláště na modelech s malým počtem vrcholů, velmi špatně se projevují ostré spekulární odrazy.
- **Phongovo** (neplést s Phongovým osvětlovacím modelem) – Osvětlovací model počítá pro každý pixel tak, že interpoluje normálu mezi vrcholy modelu – velmi kvalitní, pomalejší na výpočet.
- **Blinn-Phongovo** – Obětuje přesnost Phongova modelu pro rychlejší výpočet, je lepší než Goraduovo stínování.

Illustration 48: ploché, Goraudovo a Phongovo stínování

Materiál je kombinace parametrů, které vytvářejí vzhled povrchu, a někdy také jeho další, např. fyzikální, vlastnosti. Parametry mohou být zadány pro celý materiál (pak bude jeho vzhled uniformní), nebo mohou být kontrolovány texturami (abychom dosáhli neuniformního vzhledu, např. škrábanců na povrchu, různé odrazivosti na různých místech apod). Složité materiály se mohou skládat z mnoha textur, které ovlivňují parametry, jako jsou:

- **difúzní barva** – Základní barva povrchu.
- **normála** – Normálové mapy umožňují ovlivňovat výpočet osvětlovacího modelu na malých měřítcích, čímž můžeme dosáhnout efektů poškrábání povrchu, vrypů a jiných malých nerovností (avšak ty se projevují jen barvou pixelů, nikoli skutečným ovlivněním geometrie).
- **spekulární barva** – barva spekulárních odrazů (ta je pro většinu materiálů reálného světa vždy bílá, avšak existují výjimky).
- **spekularita** – Určuje intenzitu spekulárních odrazů.
- **shininess** – Určuje velikost spekulárních odrazů.
- **reflektivita** – Jak moc světla materiál odráží.
- **ambientní barva** – Barva ambientní složky osvětlení.
- **průhlednost** – Jak moc je materiál průhledný.
- **refrakční index** – Index lomu.
- **subsurface scattering** – Šíření světla pod povrchem materiálu, umožňuje simulaci materiálů jako je vosk, lidská kůže apod.
- **displacement** – Modifikuje přímo geometrii posunem ve směru normály.
- prolínání více materiálů – Umožnuje kombinovat materiály s plynulými přechody.
- obecné parametry pro shadery – Pro vlastní shadery můžeme dodávat obecné parametry.

24. Realistické zobrazování: metoda sledování paprsku, radiozita, distribuované sledování paprsku. PGR

Zobrazovací metody dělíme na:

- **přímé** (**object order**) – Jednotlivé objekty se vykreslují nezávisle, jeden po druhém. Patří sem např. Klasická rasterizace v OpenGL.
- **po pixelech** (**image order)** – Scéna se vykresluje po jednotlivých pixelech průmětny. Patří sem např. Sledování paprsku.
- **globální** – Scéna se zpracovává celá najednou, tzn. že objekty se mohou navzájem ovlivňovat. Globální metody bývají výpočetně náročné. Patří sem např. radiozita.

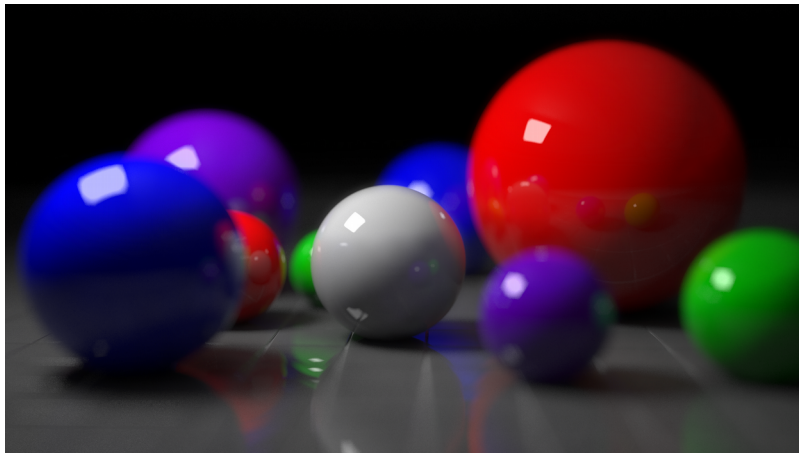


Illustration 49: distribuovaný raytracing (hloubka ostrosti)

(Zpětné rekurzívní) **sledování paprsku** (**ray tracing**) je image order metoda pro realistické zobrazování. Algoritmus funguje následujícím způsobem: pro každý pixel průmětny se vrhne do scény paprsek a barva, kterou paprsek vrátí, se zapíše na daný pixel. Rozlišujeme následujícíc druhy paprsků:

- **primární** – Prvotní vržený paprsek přes pixel průmětny do scény.
- **sekundární** – Rekurzivně vržené paprsky z povrchu objektu, může jít např. o odrazové paprsky, lomové paprsky apod.
- **stínové** – Paprsky, které se při dopadu na povrch objektu vrhají ke všem světelným zdrojům, aby se zjistilo, zda je daný bod zastíněný. Tento paprsek nevrací na rozdíl od ostatních barvu, ale jenom hodnotu ano/ne.

Illustration 50: kaustiky pomocí photon mappingu

Distribuovaný ray tracing je vylepšení metody ray tracing, které dokáže modelovat "neostré" jevy, jako např. měkké stíny, hloubku ostrosti, motion blur, "rozmazaný" odraz a lom apod. Na rozdíl od klasického ray tracingu všude tam, kde se vrhá jeden paprsek, vrhá paprsků více v mírně odkloněných směrech, tzn. V určitém intervalu, a výsledek všech potom určitým způsobem průměruje.

Raytracing bez rekurze se nazývá **ray casting**.

Je dobré si uvědomit, že paprsky mohou být vrhány paralelně, nicméně metoda je dnes stále ještě příliš náročná pro zobrazování v reálném čase.

I když dává (distribuovaný) raytracing velmi dobré výsledky, pořád nedokáže modelovat všechny jevy reálného světa, jako jsou např. kaustiky (soustředění světla čočkou do určitého místa) nebo vzájemné difúzní osvětlování objektů ve scéně.

Existují metody, které dávají realističtější výsledky, např. photon mapping (sleduje jak paprsky z kamery, tak ze světelných zdrojů) nebo path tracing (iterativní metoda, která je v určitých případech fotorealistická). Tyto metody dosahují realismu tím, že implementují **zobrazovací rovnici** (**rendering equation**), což je rovnice, která pro daný bod prostoru a směr dává množství světla, které je tímto směrem vysíláno. Rovnice má tvar:

$L(x, w_0) = L_E(x, w_0) + \int_\Omega L_I(x, w_i) \cdot f_r(x, w_i, w_0) \cdot \cos(\theta_i)\, dw_i$

kde

- $L(x, w_0)$ je intenzita světla vyslaného z bodu x ve směru w_0.
- $L_E(x, w_0)$ je intenzita světla emitovaného zdrojem z bodu x ve směru w_0.
- $L_I(x, w_i)$ je intenzita světla jdoucího do bodu x ze směru w_i.
- $f_r(x, w_i, w_0)$ je BRDF v bodě x ze směru w_i do w_0.

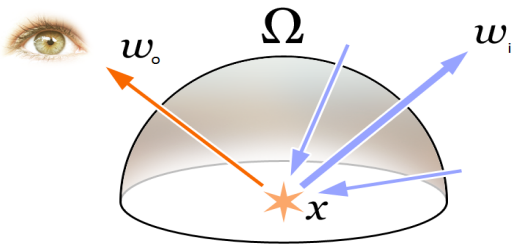


Illustration 51: zobrazovací rovnice

Radiozita je metoda přibližného řešení zobrazovací rovnice pro scény s objekty, které odráží světlo difůzně. Je to metoda **globálního osvětlení**, tzn. jejím výsledkem není snímek scény z pohledu pozorovatele, jako u raytracingu nebo rasterizace, ale množství světla v různých částech scény. Proto nelze pro zobrazování použít pouze radiozitu. Výhodou je naopak fakt, že u statické scény stačí radiozitu spočítat jednou a pak můžeme její výsledek využít pro každý snímek při zobrazování v reálném čase. Principem radiozity je rozdělení všech povrchů ve scéně na malé plošky a potom počítat, jak každá ploška ovlivňuje každou jinou plošku ve scéně.

Pro realkistické výsledky se sledování paprsku a radiozita kombinují.

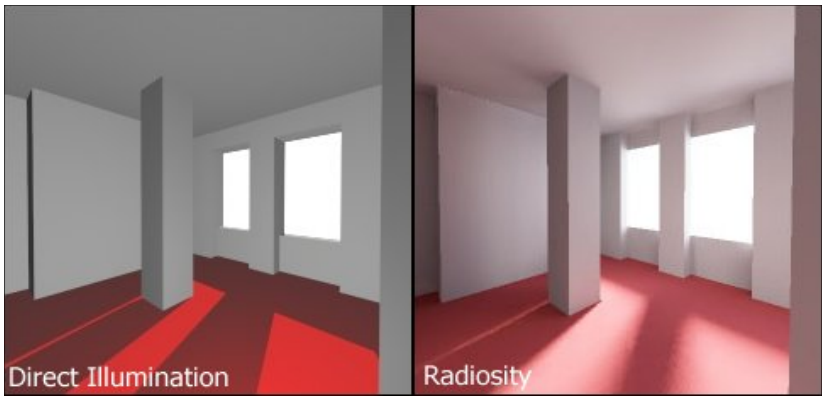


Illustration 52: přímé osvětlení vs radiozita

25. Textury a texturování: texturování, MIP mapping, procedurální textury, mřížkové šumy. PGR

Jako **textura** se označují data nesoucí určitou informaci o povrchu objektu (např. barva, normála, odrazivost, průhlednost, vyvýšení, ...). Textury dělíme podle dimenze nejčastěji na:

- **2D** – Jedná se o bitmapu nebo rastrový obrázek. Na povrch se nanáší pomocí texturovacích souřadnic – každý vrchol 3D modelu má 2D **texturovací souřadnice** (tzv. *uv* souřadnice), které se při vykreslování trojúhelníků interpolují a používají jako adresa do textury. Mapování 2D textury na 3D model je obecně obtížný proces a většinou se musí pro dobrý výsledek provádět ručně. Při použití lineární interpolace texturovacích souřadnic (tzv. afinní texturování) dochází ke zkreslení textury, protože se nebere v potaz perspektiva. Moderní HW však zvládá **perspektivně korektní** interpolaci, která bere v potaz souřadnici *z*.
- **3D** – Jedná se o funkci definovanou v prostoru. Textura vytvořená relativně jednoduchou funkcí se nazývá **procedurální**, proto jde 3D texturování a procedurální texturování ruku v ruce. 3D textura má tu výhodu, že se nemusí mapovat, každý bod povrchu modelu může přímo zaadresovat texturu svými 3D souřadnicemi.

Existují i 1D textury. např. v OpenGL se používají, např. pro barevné přechody apod.

Procedurální texturování využívá primitivní funkce (mix, step, pulse, sin, cos, clamp, ...) a různé druhy šumových funkcí, např.:

- **Lattice noise** (**mřížkový šum**) – šum založený na mřížce určitých hodnot, které se náhodně generují. Mezi mřížkovými body se hodnoty interpolují z uzlů mřížky. Dělí se na:
  - **Value noise –** V mřížce jsou přímo náhodně generované hodnoty.
  - **Gradient noise** – V uzlech jsou generovány gradienty, hodnota šumu se vypočítá jako skalární součin gradientu a vektoru udávajícího směr z uzlu k danému vzorku. Pro daný vzorek se takto vypočítá hodnota vzhledem k uzlům, mezi nimiž se nachází, a tyto hodnoty se interpolují podle pozice vzorku. To má za následek fakt, že přímo v uzlech jsou vypočtené hodnoty rovny 0 (vektor vzdálenosti je 0). Lattice noise zahrnuje Perlin noise a Simplex noise.
- **Perlin noise** – Lattice noise, který používá pravidelnou čtvercovou mřížku a který se nejčastěji dále implementuje jako tzv. **fraktální šum** - sčítají se šumy o různých frekvencích přičemž platí, že čím vyšší frekvence, tím nižší amplituda, frekvence často tvoří tzv. oktávy, tzn jsou celočíselnými násobky nějaké základní frekvence.
- **Simplex noise** – Lattice noise, který využívá **simplexy** (*n*dimenzionální trojúhelník, např. 1d – úsečka, 2d – trojúhelník, 3d – čtyřstěn atd.), na něž dělí prostor (nevyužívá tedy čtvercovou mřížku, ale “trojúhelníkovou”). Má méně směrových artefaktů a ve vyšších dimenzích je výpočetně výhodnější (je škálovatelnější).

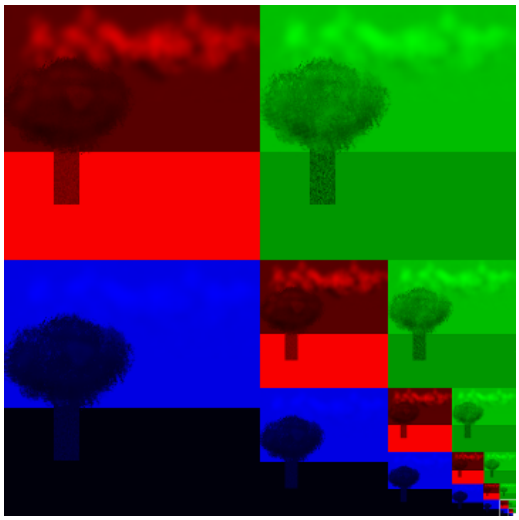


Illustration 53: MIP mapa

MIP (multum in parvo, hodně v málu) **mapping** je technika zabraňování aliasingu při texturování, dá se však použít i k jiným účelům, např. level of detail. Spočívá v předpočítání sekvence zmenšených verzí textury, které posléze vytvoří tzv. MIPmap pyramidu. Když vykreslujeme objekt, který je kvůli perspektivě hodně malý a má při tom velkou texturu, může docházet k aliasingu, protože se vzorkovací frekvence textury je velmi malá. Navíc při každém překreslení se může textura navzorkovat jinak a pixely mohou "blikat". Grafická karta v takovém případě vybere z MIP mapy patřičně zmenšenou texturu a použije ji. Problém může nastat, pokud je textura zmenšená jenom v jednom směru, typicky např. Při zobrazování země ve hrách. To řeší vylepšením zvané anizotropní filtrování, které zavádí speciální MIP mapy obsahující verze textury zmenšené v jednom směru a zachovávající detaily v druhém. Vylepšení MIP map pro akomodaci zmenšení v různých směrech se nazývá **anizotropní filtrování** – obdoba MIP mapy potom shromažďuje obrázek ve velikostech různých podle různých os, což umožňuje snížit rozlišení bez aliasingu v jednom rozměru a zachovat vysoké rozlišení v jiném rozměru.

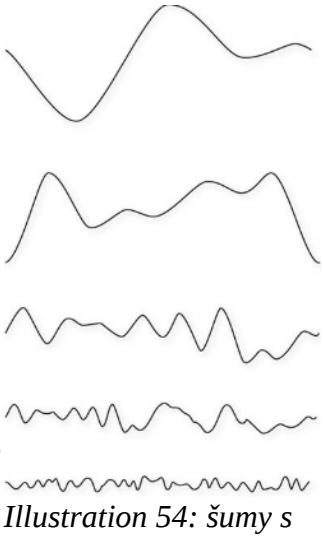


Illustration 54: šumy s klesající amplitudou a vzrůstající frekvencí, jejich součet dá Perlinův šum

Při adresování textury většinou řešíme:

- přístup na souřadnice mimo texuru (zalamování) – Můžeme zakázat, vracet nulu, použít modulo souřadnic (textura se bude opakovat), zrcadlení textury, apod.
- přístup na souřadnice mzi texely (**filtrace**, **interpolace**) – metody interpolace:
  - **nejbližší soused** (**nearest neighbour**) – Vrací se barva nejbližšího texelu, rychlé, jednoduché, ne moc pěkné (“hranaté” pixely).
  - **lineární** – Lineární interpolace mezi 4 texely.

- **kubická** – Je náročnější než lineární, ale vypadá lépe. Prokládá vzorky polynomem stupně 3. Využívá 16 vzorků (4 krát 4).

Před název interpolace se někdy přidává předpona podle rozměru, např. bilinéární = lineární v 2D, trilineární = lineární v 3D apod.

Rovněž musíme brát v potaz **perspektivní korekci** při interpolaci texturovacích souřadnic. Bez korekce se souřadnice interpolují bez ohledu na perspektivu a výsledek vypadá špatně, perspektivní korekce bere v potaz z souřadnici. V OpenGL je korekce automatická.

Textury slouží také k tvorbě materiálů.

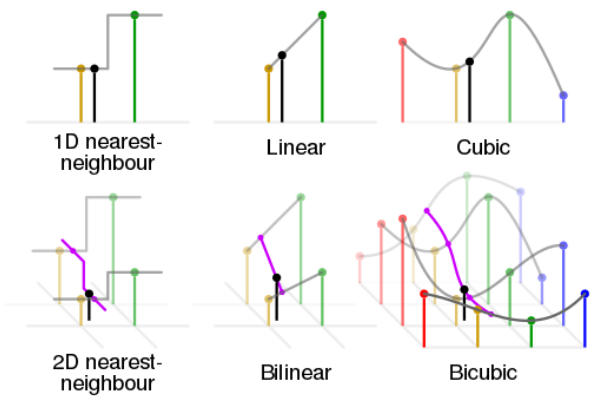


Illustration 55: interpolace

26. Klasifikace gramatik, formálních jazyků a automatů přijímajících jazyky. TIN

Gramatiky klasifikujeme nejčastěji dle **Chomského hierarchie** jazyků na:

- **typ 0** – **obecné neomezené gramatiky**:
  $\alpha \rightarrow \beta$ $\quad\quad$ $\alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^*,\ \beta \in (N \cup \Sigma)^*$

  Popř. také **Turingovy stroje** nebo lambda kalkul.
- **typ 1** – **kontextové gramatiky**:
  $\alpha A \beta \rightarrow \alpha B \beta$ $\quad\quad$ $A \in N, \alpha,\beta \in (N \cup \Sigma)^*, B \in (N \cup \Sigma)^+$
  nebo
  $S \rightarrow \varepsilon$ $\quad\quad$ pokud jazyk obsahuje ε

  Popř. také **lineárně omezené automaty** (nedeterministický TS používající jen tu část pásky, na níž má vstup).
- **typ 2** – **bezkontextové gramatiky**:
  $A \rightarrow \alpha$ $\quad\quad$ $A \in N, \alpha \in (N \cup \Sigma)^*$

  Popř. také **zásobníkové automaty**.
- (**deterministické bezkontextové**:)
  Jsou přijímány deterministickými zásobníkovými automaty.
- **typ 3** – **pravé lineární gramatiky**:
  $A \rightarrow xB$ nebo $A \rightarrow x$ $\quad\quad$ $A, B \in N, x \in \Sigma^*$

  popř. také (všechny mohou mezi sebou být převáděny):
  - **pravé lineární**: $A \rightarrow xB$ nebo $A \rightarrow x$ $\quad\quad$ $A, B \in N, x \in \Sigma^*$
  - **levé lineární**: $A \rightarrow Bx$ nebo $A \rightarrow x$ $\quad\quad$ $A, B \in N, x \in \Sigma^*$
  - **pravé regulární**: $A \rightarrow aB$ nebo $A \rightarrow a$ nebo $S \rightarrow \varepsilon$ $\quad\quad$ $A, B \in N, a \in \Sigma$
  - **levé regulární**: $A \rightarrow Ba$ nebo $A \rightarrow a$ nebo $S \rightarrow \varepsilon$ $\quad\quad$ $A, B \in N, a \in \Sigma$

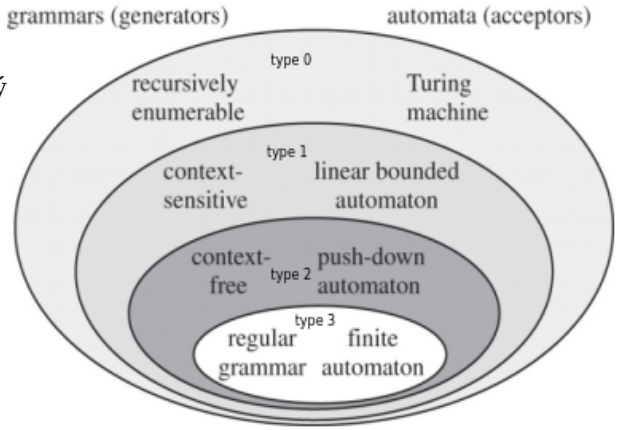


Illustration 56: Chomského hierarchie jazyků

Regulární jazyky mohou rovněž být popsány následujícími prostředky:

- **regulární výrazy**: konečné výrazy popisující potenciálně nekonečné množiny řetězců
  Regulární množiny jsou právě:
  - $\emptyset$ (prázdná množina)
  - $\{\varepsilon\}$ (množina obsahující pouze prázdný řetězec)
  - $\{a\}$ (množina obsahující pouze symbol a)
  - Jsou-li P a Q regulární množiny, pak také $P \cup Q$, $P \cdot Q$, P^* jsou regulární množiny.

  Regulární výrazy (RV) jsou právě:
  - $\emptyset$ označuje množinu $\emptyset$
  - ε označuje množinu $\{\varepsilon\}$
  - a označuje množinu $\{a\}$
  - jsou-li p a q RV označující množiny P a Q, pak také:
    - $(p + q)$ je RV označující množinu $P \cup Q$
    - (pq) je RV označující množinu $P \cdot Q$
    - (p^*) je RV označující množinu P^*
- **konečné automaty** (KA):
  KA je pětice $(Q, \Sigma, \delta, q_0, F)$, kde
  - Q je konečná množina stavů.
  - Σ je vstupní abeceda.
  - δ je zobrazení $Q \times \Sigma \rightarrow 2^Q$, které nazýváme přechodová funkce.
  - $q_0 \in Q$ je počáteční stav.
  - $F \subseteq Q$ je množina koncových stavů.

27. Vlastnosti formálních jazyků (typické vlastnosti a jejich rozhodnutelnost). TIN

U formálních jazyků nás typicky zajímají vlastnosti:

- **strukturální** – Platnost různých vět.
- **uzávěrové** – Vzhledem k jakým operacím jsou jazyky uzavřené.
- **rozhodnutelnost problémů**

Regulární jazyky (*L*3):

- strukturální:
  - Každý konečný jazyk je regulární.
  - **pumping lemma** – Dá se použít k důkazu, že daný jazyk není regulární (avšak nedá se jím dokázat, že regulární je). Pumping lemma zní:

    Nechť L je nekonečný regulární jazyk. Pak existuje celé číslo $p > 0$ takové, že:
    $\omega \in L \wedge |\omega| \geq p \Rightarrow \quad \omega = x\,y\,z \quad \wedge \quad 0 < |y| \leq p \quad \wedge \quad x\,y^i\,z \in L$ pro $i \geq 0$.

    Např.: Dokažte, že jazyk $L = \{0^n1^n \mid n \geq 1\}$ není regulární.
    Předpokládejme, že L je regulární. Pro dostatečně velké k můžeme mít řetězec $\omega = x\,y\,z = 0^k1^k$, $0 < |y| \leq k$, $x\,y^i\,z \in L$. Při hledání podřetězce y mohou nastat 3 možnosti:
    - $y \in \{0\}^+$ - v $x\,y^i\,z$ nesouhlasí počet 0 a 1 v ω, tzn. $\omega \notin L$.
    - $y \in \{1\}^+$ - v $x\,y^i\,z$ nesouhlasí počet 0 a 1 v ω, tzn. $\omega \notin L$.
    - $y \in \{0\}^+ \cdot \{1\}^+$ - v $x\,y^i\,z$ neplatí, že všechny 0 předchází všechny 1, tzn. $\omega \notin L$.

    ω tedy nemůže patřit do L, což je v rozporu s předpokladem. L tedy nemůže být regulární.

  - **Myhill-Nerodova věta** – Dá se použít k důkazu, že daný jazyk není regulární. Věta zní:

    Nechť L je jazyk nad Σ. Pak následující tvrzení jsou ekvivalentní:
    - L je přijímaný deterministickým konečným automatem.
    - L je sjednocením některých tříd rozkladu určeného pravou kongruencí na Σ^* s konečným indexem.
    - Relace $\sim_L$ má konečný index.

    Relace ekvivalence ~ na Σ^* je **pravou kongruencí**, pokud pro každé $u, v, w \in \Sigma^*$ platí $u \sim v \Rightarrow uw \sim vw$.
    Relace $\sim_L$ je tzv. **prefixová ekvivalence** definovaná jako: $u \sim_L v \Leftrightarrow \forall w \in \Sigma^*: uw \in L \Leftrightarrow vw \in L$. **Index ekvivalence** je počet tříd rozkladu podle ní.

- uzávěrové:
  - *L*3je uzavřena vůči sjednocení, konkatenaci a iteraci.
  - *L*3 tvoří množinovou Booleovu algebru.
- rozhodnutelnost problémů:
  - V *L*3 jsou rozhodnutelnými problémy:
    - neprázdnost jazyka (lze sestrojit KA a ověřit, zda existuje koncový stav dostupný z počátečního)
    - náležitost řetězce (lze ověřit pomocí KA)
    - ekvivalence jazyků (lze sestrojit relaci nerozlišitelnosti stavů KA)

Bezkontextové jazyky (*L2*):

- strukturální:
  - **pumping lemma** – Podobné jako pumping lemma pro regulární jazyky, dá se použít k důkazu, že jazyk není bezkontextový (ale ne k důkazu, že bezkontextový je). Pumping lemma zní:

    Nechť L je bezkontextový jazyk. Pak existuje celé číslo $p > 0$ takové, že je-li $z \in L$ a $|z| \geq p$, pak lze z zapsat jako: $z = u\,v\,w\,x\,y$, $v\,x \neq \varepsilon$, $|v\,w\,x| \leq p$
    a pro všechna $i \geq 0$ platí: $u\,v^i\,w\,x^i\,y \in L$.

    Např. důkaz, že $L = \{a^nb^nc^n \mid n \geq 1\}$ není bezkontextový - nelze zvolit řetězce v a x tak, aby zůstal stejný počet a, b a c a zachovalo se jejich pořadí.

- uzávěrové:

- *L*2 je uzavřená vůči sjednocení, konkatenaci, iteraci a pozitivní iteraci.
- *L*2 je uzavřena vůči průniku s *L*3.
- *L*2 je uzavřená vůči substituci. **Substituce** jazyka *L* z nad abecedou $\Sigma = \{a_1, \ldots, a_n\}$ je jazyk $\sigma_{La1}, \ldots, {}_{Lan}(L)$, kde každý symbol z Σ je nahrazen nějakým slovem z jazyka L_{ai}, který patří do stejné třídy jako *L*. Např.: $\sigma_{\{0,1\}\{a,b,c\}}\{X,XY\} = \{0,1,0a,0b,0c,1a,1b,1c\}$
- *L*2 je uzavřená vůči morfismu. **Morfismus** je zobrazení mapující symboly jedné abecedy na druhou (morfismus jazyka je potom jazyk, který dostaneme aplikací morfismu na každé slovo jazyka). Morfismus je speciální případ substituce, kdy se každé slovo jedné abecedy mapuje právě na jedno slovo výstupní abecedy.
- *L*2 je uzavřena vůči inverznímu morfismu. Inverzní morfismus je inverzní zobrazení morfismu.
- *L*2 není uzavřena vůči průniku (lze najít protipříklad) a doplňku (z De Morganových zákonů).
- Deterministické bezkontextové jazyky jsou uzavřeny vůči průniku s regulárními jazyky.
- Deterministické bezkontextové jazyky nejsou uzavřeny vůči průniku, sjednocení, konkatenaci a iteraci.

- rozhodnutelnost problémů:
  - V *L*2 jsou rozhodnutelnými problémy:
    - neprázdnost jazyka
    - náležitost řetězce
    - konečnost (lze odvodit z pumping lemmatu)
  - V *L2* jsou nerozhodnutelnými problémy:
    - ekvivalence jazyků
    - inkluze jazyků

Kontextové jazyky (*L1*):

- uzávěrové:
  - Třída *L1* je uzavřena vůči sjednocení, průniku, iteraci, konkatenaci a doplňku.
- rozhodnutelnost problémů:
  - V *L1* není rozhodnutelný problém prázdnosti jazyka.

Rekurzívně vyčíslitelné (*L0*):

- uzávěrové:
  - Třída *L0* je uzavřena vůči průniku, sjednocení, iteraci a konkatenaci.
  - Jazyky pouze rekurzívní jsou navíc uzavřeny i vůči doplňku.

28. Konečné automaty (jazyky přijímané jazyky KA, varianty KA, minimalizace KA, Mihill-Nerodova věta). TIN

Konečný automat (**KA**) je pětice $M = (Q, \Sigma, \delta, q_0, F)$, kde

- Q je konečná množina stavů.
- Σ je konečná vstupní abeceda.
- δ je zobrazení $Q \times \Sigma \to 2^Q$, které nazýváme **přechodovou funkcí**, (2^Q je množina podmnožin množiny Q).
- $q_0 \in Q$ je počáteční stav.
- $F \subseteq Q$ je množina koncových stavů.

Je-li $\forall q \in Q\ \forall a \in \Sigma$: $|\delta(q,a)| \leq 1$, nazýváme M **deterministickýcm** KA (DKA), jinak **nedeterministickým** KA (NKA). Je-li δ totální funkce, pak nazýváme M **úplně definovaným** KA.

Ke každému DKA existuje ekvivalentní úplně definovaný KA. Dále ke každému DKA existuje ekvivalentní NKA a naopak a existují algoritmy převodu mezi nimi ($Q' = 2^Q$, $q_0' = \{q_0\}$, ...).

Konfigurace automatu M je dvojice $C = (q, w) \in Q \times \Sigma^*$. Konfigurace tvaru (q_0, w) je **počáteční**, konfigurace tvaru (q, ε), $q \in F$ je **koncová** (těch může být více).

Přechod automatu M je reprezentován binární relací $\vdash_M$ na množině konfigurací C. Pro všechna $q, q' \in Q$ a $w, w' \in \Sigma^*$ definujeme, že platí $(q, w) \vdash_M (q', w')$ právě tehdy, když $w = aw'$ pro nějaké $a \in \Sigma$ a $q' \in \delta(q, a)$.

$\vdash_M^k$ označuje k-tou mocninu relace (právě k přechodů automatu). $\vdash_M^+$ označuje tranzitivní uzávěr, $\vdash_M^*$ označuje tranzitivní a reflexivní uzávěr.

Jazyk přijímaný KA M je množina:

$$L(M) = \{w \mid (q_0, w) \vdash_M^* (q, \varepsilon), q \in F\}.$$

Jazyky přijímané KA jsou **regulární jazyky**.

Minimalizace je technika umožňující najít ke každému KA ekvivalentní KA s minimálním (nejmenším možným) počtem stavů. Algoritmus minimalizace je následující:

1. Eliminujeme **nedosažitelné stavy** (**dosažitelný stav** je každý stav q: $\exists\, w \in \Sigma^*$: $(q_0,w) \vdash (q,\varepsilon)$). To uděláme tak, že inicializujeme množinu dosažitelných stavů $S_0 = \{q_0\}$ a v iteracích přidáváme stavy, do nichž se lze dostat ze stavů v S_i. Jakmile $S_{i+1} = S_i$, skončíme a máme množinu dosažitelných stavů.
2. Eliminujeme **nerozlišitelné stavy**. Stav je nerozlišitelný, pokud neexistuje řetězec, který je rozlišuje. Řetězec w **rozlišuje** stavy q_1 a q_2, pokud $(q_1,w) \vdash^* (q_3,\varepsilon) \wedge (q_2,w) \vdash^* (q_4,\varepsilon) \wedge$ zároveň právě jeden ze stavů q_3 a q_4 patří do F. (Stavy jsou k-nerozlišitelné, pokud neexistuje w: $|w| \leq k$, který je rozlišuje, píšeme $p \equiv^k q$. Relace $\equiv$ je ekvivalence na Q.) Algoritmus je následující:
   1. $i = 0$; $\equiv^0 := \{(p, q) \mid p, q \in F\}$
   2. repeat until $\equiv^i = \equiv^{i-1}$:
      a) $\equiv^{i+1} := \{(p, q) \mid p \equiv^i q \wedge \forall a \in \Sigma: \delta(p, a) \equiv^i \delta(q, a)\}$; $i = i + 1$
   3. $Q' := Q / \equiv^i$ // nové Q budou třídy rozkladu ekvivalencí $\equiv^i$
   4. $q'_0 := [q_0]$; $F' = \{[q] \mid q \in F\}$

Jestliže žádný stav úplného KA není nedostupný a žádné dva stavy nejsou nerozlišitelné, automat je **redukovaný** (minimalizovaný).

Myhill-Nerodova věta specifikuje určité vztahy mezi KA nad abecedou Σ a relacemi nad Σ a specifikuje nutné a postačující podmínky pro regularitu jazyka – používá se pro důkazy, že daný jazyk není regulární. Myhill-Nerodovu větu si specifikujeme následujícím způsobem:

- Nechť relace $\sim$ je relace ekvivalence (tzn. je reflexivní, symetrická a tranzitivní) nad Σ^* (množina řetězců nad Σ). **Index** ekvivalence $\sim$ je počet tříd, na něž množinu Σ^* dělí (může být i nekonečný).
- Relace $\sim$ je tzv. **pravá kongruence** právě tehdy, když: $\forall u, v \in \Sigma^*, a \in \Sigma$: $u \sim v \Rightarrow ua \sim va$.
- Relaci prefixové ekvivalence $\sim_L$ na jazykem L (ne nutně regulárním) definujeme jako:

$u \sim_L v \quad \Leftrightarrow \quad \forall w \in \Sigma^*: uw \in L \Leftrightarrow vw \in L$

- Myhill-Nerodova věta zní: Nechť L je jazyk nad Σ. Pak následující tvrzení jsou ekvivalentní:
  - L je přijímaný nějakým DKA.
  - L je sjednocením některých tříd rozkladu určeného pravou kongruencí na Σ* s konečným indexem.
  - Relace $\sim_L$ má konečný index.

př.: Dokažte, že jazyk $\{a^i b^i\}$ není regulární.

1. Relace $\sim_L$ je relace nad všemi řetězci nad Σ, tzn. můžeme se ptát, do jakých tříd patří např. řetězce:

   ε, a, a^2, a^3, ...

2. Zjistíme, že každý z nich patří do jiné třídy (nejsou ekvivalentní), tzn. je nekonečně mnoho tříd, tzn. jazyk není regulární.

3. Jaktože patří každý do jiné třídy? Vezměme např. a a a^2. Kdyby byly ve stejné třídě, pak by muselo platit:

   $a\ w \in \mathrm{L} \Leftrightarrow a^2\ w \in \mathrm{L}$

   Nicméně pro $w = bb$: aabb patří do L, ale $aaabb$ nepatří do L. a a a^2 jsou tedy v jiných třídách.

29. Regulární množiny, regulární výrazy a rovnice nad regulárními výrazy. TIN

Regulární množinu nad konečnou abecedou Σ definujeme takto:

- Prázdní množina Ø je regulární množina nad Σ.
- {ε} je regulární množina nad Σ.
- {a} pro všechna $a \in \Sigma$ je regulární množina nad Σ.
- Jsou-li P a Q regulární množiny nad Σ, pak také $P \cup Q$, $P \cdot Q$ a P^* jsou regulární množiny nad Σ.
- Žádné jiné množiny než ty, které lze získat předchozími pravidly, neexistují.

Regulární výrazy jsou notací pro zápis regulárních množin. Definujeme je takto:

- Ø je regulární výraz značící množinu Ø.
- ε je regulární výraz značící množinu {ε}.
- a je regulární výraz značící množinu {a} pro každé $a \in \Sigma$.
- Jsou-li p a q regulární výrazy označující množiny P a Q, pak
  - ($p + q$) je regulární výraz označující regulární množinu $P \cup Q$.
  - (pq) je regulární výraz označující množinu $P \cdot Q$.
  - (p^*) je regulární výraz značící množinu P^*.
- Jiné než výše uvedené regulární výrazy neexistují.

Rovnice nad regulárními výrazy jsou rovnice, jejímiž neznámými a koeficienty jsou regulární výrazy.

Platí, že "nejmenším" řešením (pevným bodem) rovnice $X = pX + q$ je $X = p^*q$ (důkaz se provede dosazením a zjednodušením obou stran na stejný tvar).

Tyto rovnice mohou tvořit **soustavy rovnic nad regulárními výrazy**. Říkáme, že soustava je ve standardním tvaru vzhledem k neznámým $\Delta = \{X_1, X_2, ..., X_n\}$, má-li tvar

$$\bigwedge_{i \in \{1, ..., n\}} X_i = \alpha_{i0} + \alpha_{i1} X_1 + \alpha_{i2} X_2 + ... + \alpha_{in} X_n$$ kde α_{ij} jsou regulární výrazy nad Σ.

Je-li soustava ve standardním tvaru, pak existuje její minimální pevný bod a algoritmus jeho nalezení: vyjadřujeme hodnotu jednotlivých proměnných jako řešení rovnice $X = pX + q$ (viz výše). Např.:

(1) $X_1 = (01^* + 1) X_1 + X_2$
(2) $X_2 = 11 + 1 X_1 + 00 X_3$
(3) $X_3 = \varepsilon + X_1 + X_2$

1. dosadíme (3) do (2):
   (4) $X_1 = (01^* + 1) X_1 + X_2$
   (5) $X_2 = 11 + 1 X_1 + 00 (\varepsilon + X_1 + X_2) = 00 + 11 + (1 + 00) X_1 + 00 X_2$
2. Ze (4) vyjádříme X_1 (pomocí $X = pX + q$):
   (6) $X_1 = (01^* + 1)^* X_2 = (0 + 1)^* X_2$
3. Dosadíme do (5):
   (7) $X_2 = 00 + 11 + (1 + 00) (0 + 1)^* X_2 + 00 X_2 = 00 + 11 + (1 + 00) (0 + 1)^* X_2$
4. Vyjádříme řešení X_2 (pomocí $X = pX + q$):
   (8) $X_2 = ((1 + 00) (0 + 1)^*)^* (00 + 11)$
5. Dosadíme do (6):
   (9) $X_1 = (0 + 1)^* ((1 + 00)(0 + 1)^*)^* (00 + 11) = (0 + 1)^* (00 + 11)$
6. Dosadíme do (3):
   (10) $X_3 = ... = \varepsilon + (0 + 1)^* (00 + 11)$

Regulární výrazy tvoří tzv. **Kleeneho algebru** (algebra (M, +, ·, *, 0, 1) splňující určité axiomy).

30. Transformace a normální formy bezkontextových gramatik. TIN

Nechť $G = (N, \Sigma, P, S)$ je gramatika.

Gramatika G je v **Chomského normální formě** (CNF)právě tehdy, když každé pravidlo z P má jeden z těchto tvarů:

- $A \rightarrow BC$ $\quad A, B, C \in N$
- $A \rightarrow a$ $\quad A \in N, a \in \Sigma$
- Pokud $\varepsilon \in L(G)$, pak $S \rightarrow \varepsilon$ je pravidlo z P, S je výchozí symbol, který se neobjevuje na pravé straně žádného pravidla.

Gramatika G je v **Greibachové normální formě** (GNF), je-li G gramatikou bez ε-pravidel (pravidla tvaru $A \rightarrow \varepsilon$) a jestliže každé pravidlo (s výjimkou případného $S \rightarrow \varepsilon$) má tvar $A \rightarrow a\alpha$, kde $a \in \Sigma$ a $\alpha \in N^*$.

Transformace gramatiky je operace, které gramatiku modifikuje, ale zachovává jazyk, který generuje (tzn. gramatika je ekvivalentní gramatice původní).

ε-pravidlo je pravidlo tvaru $A \rightarrow \varepsilon$. Gramatika bez ε-pravidel je gramatika, která neobsahuje žádná ε-pravidel pravidla s výjimkou případného pravidla $S \rightarrow \varepsilon$. Existuje algoritmus na převod gramatiky na gramatiku bez ε-pravidel (konstrukcí množiny N_t – viz dále – a poté úpravou všech pravidel zohledňujících tuto množinu).

Symbol X je v gramatice G **nedostupný**, jestliže se G nemůže vyskytnout v žádné **větné formě** (řetězec generovaný gramatikou, složený z terminálů a případně neterminálů). Algoritmus na odstranění nedostupných symbolů je následující:

1. vytovř množinu $V_0 = \{S\}$, a nechť $i = 1$
2. opakuj, dokud platí $V_i \neq V_{i-1}$: $\quad V_i = \{X \mid A \rightarrow \alpha X \beta \in P \wedge A \in V_{i-1}\} \cup V_{i-1}$
3. $N' = V_i \cap N$, $\Sigma' = V_i \cap \Sigma$, P' obsahuje právě ta pravidla z P, která jsou tvořena symboly z V_i.

Symbol X je v gramatice G **zbytečný**, jestliže neexistuje derivace tvaru $S \Rightarrow^* wXy \Rightarrow^* wxy$, řetězce $w, x, y \in \Sigma^*$. Algoritmus pro odstranění zbytečných symbolů je následující:

1. $N_0 = \{\}$, $i = 1$
2. $N_i = \{A \mid A \rightarrow \alpha$ je v $P \wedge a \in (N_{i-1} \cup \Sigma)^*\}$
3. Pokud $N_i \neq N_{i-1}$, polož $i = i + 1$ a jdi na 2., jinak polož $N_t = N_i$
4. $G' = (N_t \cup \{S\}, \Sigma, P_1, S)$, kde P_1 obsahuje pouze pravidla tvořená symboly z $N_t \cup \Sigma$.
5. Na gramatiku G' aplikuj algoritmus na odstranění nedostupných symbolů.

Gramatika G je **bez cyklu**, jestliže v ní neexistuje derivace tvaru $A \Rightarrow^+ A$, pro žádné A z N.

Je-li gramatika G bez zbytečných symbolů, cyklů a ε-pravidel, říkáme, že G je **vlastní gramatika**. Pro každou bezkontextovou gramatiku existuje ekvivalentní vlastní gramatika.

Gramatika je **rekurzivní**, jestliže v ní existuje derivace $A \Rightarrow^+ \alpha A \beta$. Je-li a, resp. b rovno ε, gramatika se nazývá **levě**, resp. **pravě** rekurzivní. Le-li jazyk nekonečný, gramatika musí být rekurzivní. Existuje algoritmus na ostranění levé rekurze (což je důležité pro analýzu shora-dolů).

Jednoduchým pravidlem v gramatice nazýváme pravidlo $A \rightarrow B$, kde A a B jsou neterminály. Existuje algoritmus na ostranění jednoduchých pravidel (např. $A \rightarrow B$, $B \rightarrow c$ upravíme na $A \rightarrow c$).

Algoritmus pro převod gramatiky do Chomského normální formy je následující:

1. Transformuj vstupní gramatiku G na vlastní gramatiku bez jednoduchých pravidel. Množinu Σ ponech stejnou.
2. Všechna pravidla tvaru $A \rightarrow a$ a $A \rightarrow BC$ a $S \rightarrow \varepsilon$ (pokud jej G obsahuje) vložíme do nové množiny pravidel.
3. Pro každé pravidlo tvaru $A \rightarrow X_1 ... X_k$, kde $k > 2$ a X_i je nenterminál nebo terminál, přidej do nové množiny pravidel pravidla (X_i' je buď neterminál X_i, pokud $X_i \in$ N, nebo nový neterminál, pokud $X_i \in \Sigma$):
   $A \rightarrow X_1' \langle X_2 ... X_k \rangle$
   $\langle X_2 ... X_k \rangle \rightarrow X_2' \langle X_3 ... X_k \rangle$
   …
   $\langle X_{k-1} ... X_k \rangle \rightarrow X_{k-1}' ... X_k'$
4. Pro každé pravidlo $A \rightarrow X_1 X_2$ (X_1 nebo $X_2 \in \Sigma$) přidej nové pravidlo $A \rightarrow X_1' X_2'$ (obdobně jako výše).
5. Pro každý neterminál tvaru a' přidej pravidlo $a' \rightarrow a$.

Existuje algoritmus pro převod gramatiky do Grebachové normální formy (vytvoří se lineární uspořídání nad neterminály, pomocí něj se převedou všechna pravidla na tvar, kdy začínají terminálem, a ostatní terminály se nahradí neterminály a novými pravidly).

Dalšími důležitými pojmy v oblasti gramatik jsou:

- **derivace** – Zachycuje proces přepisování a generování řetězce pomocí aplikace pravidel. Derivaci definujeme takto:
  - **přímá derivace** (vyjadřuje použití jednoho pravidla) je binární relace $\Rightarrow$ mezi větnými formami. λ a μ jsou v této relaci (tedy $\lambda \Rightarrow \mu$) právě tehdy, pokud (α, β, γ, δ jsou větné formy):
    $\lambda = \gamma\,\alpha\,\delta \quad \wedge \quad \mu = \gamma\,\beta\,\delta \quad \wedge \quad \alpha \rightarrow \beta$ je pravidlo gramatiky
  - **derivace** (délky n) je binární relace $\Rightarrow^+$ mezi větnými formami. λ a μ jsou v této relaci (tedy $\lambda \Rightarrow^+ \mu$) právě tehdy, pokud existuje posloupnost přímých derivací délky $n \geq 1$: $\lambda \Rightarrow v_1 \Rightarrow v_2 \Rightarrow \ldots \Rightarrow \mu$.
- **věta** (také **slovo**) – řetězec daného jazyka, u gramatiky se skládá pouze z terminálů
- **větná forma** – řetězec vznikající kdykoliv v rámci derivace, skládá se z terminálů a/nebo neterminálů
- **fráze větné formy** $\lambda = \alpha\beta\gamma$ vzhledem k neterminálu A – β se nazývá frází větné formy λ, pokud platí:
  $S \Rightarrow^* \alpha\beta\gamma \wedge A \Rightarrow^+ \beta$ (tedy pokud se daný podřetězec vetné formy dá vygenerovat z jediného neterminálu).
- **derivační strom** – Strom znázorňující derivaci v gramatice. Listové uzly jsou terminály, vnitřní uzly neterminály, hrany značí přepisovací pravidla. Každá derivace má právě jeden derivační strom, avšak derivační strom může korespondovat s více derivacemi (ze stromu není jasné pořadí aplikace pravidel na stejné úrovni stromu).
- **syntaktická analýza** – Proces konstrukce derivačního stromu dané věty jazyka. Dá se provádět:
  - **shora dolů** – Začínáme startovním symbolem (S) a snažíme se vygenerovat analyzovanou větu.
  - **zdola nahoru** – Snažíme se na analyzovanou větu zpětně aplikovat pravidla, abychom došli k počátečnímu symbolu (S).
- **levá**/**pravá derivace** – Přepisuje se vždy nejlevější/nejpravější neterminál větné formy.
- **víceznačná věta** – Věta je víceznačná, existuje-li pro ni více než jeden derivační strom (např. pro pravidla $S \rightarrow AA$, $A \rightarrow a$, $A \rightarrow \varepsilon$ je věta "a" víceznačná, protže nevíme, ze kterého A vzniklo a).

Algoritmus pro převod do GNF využívá lineární uspořádání na množině neterminálů. Je následující:

1. Odstraň levou rekuri. Nyní existuje uspořádání na množ. Neterminálů N takové, že je-li pravidlo $A \rightarrow Bw$ v P, pak $A < B$. Zvolme takové uspořádání (pokud není jednoznačně určeno, nějaké zvolíme podle sebe).
2. Polož $i = |N| - 1$.
3. Je-li $i = 0$, přejdi na krok 5. Jinak nahraď každé pravidlo $A_i \rightarrow A_j w$, kde $j > i$, pravidly $A_i \rightarrow x_1 w \mid x_2 w \mid \ldots$, podle pravidel $A_j \rightarrow x_1 \mid x_2 \mid \ldots$ (x_1, x_2, … začínají neterminálem).
4. Polož $i = i - 1$ a jdi na kork 3.
5. Všechna pravidla (kromě $S \rightarrow \varepsilon$) teď začínají neterminálem. Terminály a, které nejsou začáteční, nahraď neterminály A_a a pravidly $A_a \rightarrow a$.

Algoritmus na odstranění ε-pravidel:

1. Sestroj množinu $N_\varepsilon = \{A \mid A \in N \text{ a } A \Rightarrow^* \varepsilon\}$. (Konstrukci množiny pro jednoduchost nezmiňujeme).
2. Nahraď každé pravidlo novými pravidly, kde na pravé straně se vyskytnou všechny možné kombinace nahrazení symbolů z N_ε za ε.
3. Přidej případné $S \rightarrow \varepsilon$.

31. Zásobníkové automaty (jazyky přijímané ZA, varianty ZA). TIN

Zásobníkový automat (ZA) je výpočetní model, který by se dal jednoduše charakterizovat jako konečný automat s přidaným zásobníkem, tzn. nekonečnou pamětí. Formálně je to sedmice $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde

- Q je konečná množina stavů.
- Σ je konečná vstupní abeceda.
- Γ je konečná zásobníková abeceda.
- δ je **přechodová funkce**, je to zobrazení $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$ (ZA je tedy implicitně nedeterministický, vždy musí číst symbol ze zásobníku)
- $q_0 \in Q$ je počáteční stav.
- $Z_0 \in \Gamma$ je počáteční symbol na zásobníku.
- $F \subseteq Q$ je množina koncových stavů.

Konfigurace ZA P je trojice $(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$, kde

- q je momentální stav.
- w je doposud nepřečtená část vstupního řetězce, jeho první symbol je nyní pod čtecí hlavou.
- α je obsah zásobníku, je to řetězec, jehož nejlevější symbol je na vrcholu zásobníku.

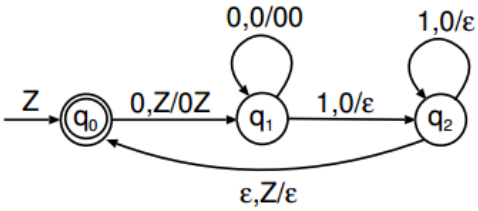


Illustration 57: zásobníkový automat

Přechodová relace je binární relace $\vdash_P$ definovaná na množině konfigurací ZA P. Definujeme, že platí

$(q, w, \beta) \vdash_P (q', w', \beta')$ $\Leftrightarrow$ $w = aw'$ pro nějaké $a \in (\Sigma \cup \{\varepsilon\})$, $\beta = Z\alpha$ a $\beta' = \gamma\alpha$ pro nějaké $Z \in \Gamma$, $\alpha, \gamma \in \Gamma^*$ a $\delta(q, a, Z)$ obsahuje prvek (q', γ).

Je-li a = ε, pak jde o tzv. **ε-přechod**, tzn přechod, při němž se nečte vstupní symbol. Ten může nastat i tehdy, když už byly přečteny všechny symboly ze vstupu. Počáteční konfigurace má tvar (q_0, w, Z_0), koncová má tvar (q, ε, α), $q \in F$.

Jazyk přijímaný ZA P je množina $L(P) = \{w \in \Sigma^* \mid c_0 \vdash_P^* c_F\}$, kde c_0 je počáteční konfigurace a c_F koncová.

ZA přijímají právě bezkontextové jazyky.

Rozšířený ZA (RZA) je varianta ZA, která může z vrcholu zásobníku číst řetězec (i prázdný), ne jenom jeden symbol. Jeho přechodová funkce má tedy tvar $\delta: Q \times (\Sigma \times \{\varepsilon\}) \times \Gamma^* \rightarrow 2^{Q \times \Gamma^*}$. Automat může tedy provádět přechody i tehdy, je-li zásobník prázdný. ZA a RZA jsou výpočetně ekvivalentní.

Deterministický ZA (DZA) je ZA, pro který platí: $\forall q \in Q, Z \in \Gamma$ platí jedno z následujících:

- $\forall a \in \Sigma$ obsahuje $\delta(q, a, Z)$ nanejvýš jeden prvek a $\delta(q, \varepsilon, Z) = \{\}$.
- $\forall a \in \Sigma$: $\delta(q, a, Z) = \{\}$ a $\delta(q, \varepsilon, Z)$ obsahuje nejvýše jeden prvek.

Neplatí, že každý ZA lze převést na DZA. Jazyky přijímané DZA nazýváme **deterministické bezkontextové jazyky**.

Někdy dále definujeme tzv. přijímání **vyprázdněním zásobníku** (u ZA i RZA). Tehdy považujeme za koncovou konfiguraci navíc každou konfiguraci $(q, \varepsilon, \varepsilon)$, kde q nemusí patřit do F. Automaty přijímající vyprázdněním zásobníku a obyčejně přijímající automaty lze mezi sebou převádět.

32. Turingovy stroje (jazyky přijímané TS, varianty TS, lineárně omezené automaty, univerzální TS). TIN

Turingův stroj (**TS**) je model výpočetního systému, který se skládá z

- řídící jednotky – konečný automat
- jednosměrně neohraničené pásky rozdělené na "buňky"
- čtecí/zapisovací hlavy

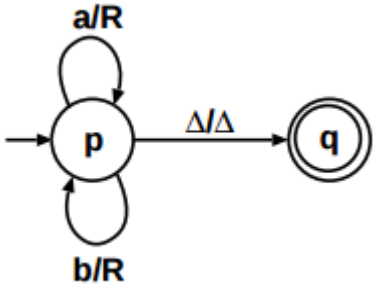


Illustration 58: TS posunující se na první blank

Formálně je TS šestice $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, kde

- Q je konečná množina vnitřních řídících stavů.
- Σ je konečná vstupní abeceda, $\Delta \notin \Sigma$.
- Γ je konečná pásková abeceda, $\Sigma \subset \Gamma, \Delta \in \Gamma$.
- $\delta: (Q \setminus \{q_F\}) \times \Gamma \rightarrow Q \times (\Gamma \cup \{L, R\})$, kde $L, R \notin \Gamma$, je parciální přechodová funkce. Pro **nedeterministický TS** (NTS) má funkce tvar $\delta: (Q \setminus \{q_F\}) \times \Gamma \rightarrow 2^{Q \times (\Gamma \cup \{L, R\})}$.
- $q_0 \in Q$ je počáteční stav
- $q_F \in Q$ je koncový stav

Symbol Δ se nazývá **blank** a nachází se na místech pásky, na která nebylo ještě nic zapsáno, může ale být zapsán i později.

Konfigurace pásky je dvojice (w, n), kde w je nekonečný řetězec reprezentující obsah pásky a n je pozice na pásce. Zapisuje se např. jako Δxyz<u>z</u>ΔxΔ Δ... (podtržení značí pozici).

Konfigurace stroje je konfigurace pásky společně s aktuálním stavem, je to tedy trojice (q, w, n).

Přechodová relace TS je relace, která udává, jaké přechody může TS provádět. Nechť γ_n je n-tý symbol řetězce $\gamma \in \Gamma^w$ (množina nekonečných řetězců nad Γ) a $s_b^n(\gamma)$ označuje řetězec, který vznikne z γ záměnou γ_n za b. Krok výpočtu TS M definujeme jako nejmenší binární relaci $\vdash_M$ takovou, že $\forall q_1, q_2 \in Q\ \forall \gamma \in \Gamma^w\ \forall n \in N\ \forall b \in \Gamma$:

- $(q_1, \gamma, n) \vdash_M (q_2, \gamma, n + 1)$ pro $\delta(q_1, \gamma_n) = (q_2, R)$ – posun doprava při γ_n pod hlavou.
- $(q_1, \gamma, n) \vdash_M (q_2, \gamma, n - 1)$ pro $\delta(q_1, \gamma_n) = (q_2, L)$ a $n > 0$ – posun doleva při γ_n pod hlavou.
- $(q_1, \gamma, n) \vdash_M (q_2, s_b^n(\gamma), n)$ pro $\delta(q_1, \gamma_n) = (q_2, b)$ – operace zápisu při γ_n pod hlavou.

Výpočet TS M je posloupnost konfigurací $K_0, K_1, K_2, ...$, ve které $K_i \vdash_M K_{i+1}$ pro všechna $i \geq 0$, že K_{i+1} je v dané posloupnosti, a která je:

- nekonečná (TS se nikdy nezastaví)
- konečná s koncovou konfigurací (q, γ, n), přičemž rozlišujeme následující typy zastavení:
  - **normální** – přechodem do koncového stavu, tj. $q = q_F$
  - **abnormální** – jedna z následujících situací:
    - δ není definována pro (q, γ_n)
    - hlava je na nejlevější pozici pásky a dojde k posunu doleva

Řetězec w je přijat TS M, jestliže M při aktivaci z počáteční konfigurace (<u>Δ</u>wΔ ...) a počátečního stavu (q_0) zastaví přechodem do koncového stavu q_F, tj. $(q0, \Delta w \Delta, 0) \vdash_M^* (q_F, \gamma, n)$ pro nějaké $\gamma \in \Gamma^*$ a $n \in N$.

Jazyk přijímaný TS M je množina $L(M) = \{w \mid M \text{ přijme } w\}$.

TS může mít alternativní definice, které jsou však výpočetně ekvivalentní a lze mezi nimi provádět převody. Patří mezi ně:

- množina koncových stavů namísto jen jednoho
- koncové stavy mohou být rozlišeny na přijímací a odmítací

- na prvním políčku pásky je napevno zapsán symbol, z něhož nelze provést posun doleva
- δ může být zavedena jako totální funkce
- přepis a posun jsou spojeny do jedné operace
- zavedení více pásek, definujeme *k* jednostranně nekonečných pásek, z nichž každá má čtecí/zapisovací hlavu a je připojena ke společné řídící jednotce, přechodová funkce má potom tvar:

  $\delta: (Q \setminus \{q_F\}) \times \Gamma_1 \times \Gamma_2 \times ... \times \Gamma_k \rightarrow Q \times \cup_{i \in \{1, ..., k\}} \{i\} \times (\Gamma_i \cup \{L, R\})$, kde *i* značí číslo pásky
- zavedení nedeterminismu (NTS, mění vlastnosti z hlediska výpočetní složitosti) - $\delta: (Q \setminus \{q_F\}) \times \Gamma \rightarrow 2^{Q \times (\Gamma \cup \{L, R\})}$, přijímaný jazyk je potom množina řetězců, pro které NTS může zastavit.

Church-Turingova teze: TS definují svou výpočetní silou to, co intuitivně považujeme za efektivně vyčíslitelné.

Lineárně omezené automaty (**LOA**) jsou omezenou variantou TS, které přijímají právě kontextové jazyky. Jedná se o nedeterministický TS, který nikdy neopustí část pásky, na které má zapsaný vstup (formálně lze zavést symbol na konci vstupu, který nelze přepsat a nelze se z něj posunout doprava). Není známo, zda deterministický LOA je striktně slabší než nedeterministický LOA.

Univerzální TS je TS, který dokáže simulovat chod jakéhokoliv jiného TS, který spolu s jeho vstupem obdrží zakódovaný na svůj vstup.

Zakódování TS *M* se vstupem *D* se provede následujícím způsobem:

- kód(*M*)#*D*
- *M* kódujeme takto:
  - Očíslujeme stavy, každému přiřadíme jeho kód jako počet nul jeho sekvenčního čísla (např. stav $q_3 \approx 3 \approx 000$).
  - Stejným způsobem kódujeme symboly z Γ.
  - Přechod $\delta(p, x) = (q, y)$ kódujeme jako čtveřici (p, x, q, y), složky oddělujeme znakem 1.
  - Celý TS kódujeme jako posloupnost přechodů oddělených a ohraničených symbolem 1.

Univerzální TS potom většinou definujeme jako trojpáskový TS, který má na 1. pásce zadání, 2. používá k simulaci a na 3. má zaznamenaný aktuální stav simulovaného TS.

NTS a **DTS** jsou ekvivalentní, co do výpočetní síly. Pro jakýkoliv NTS můžeme sestrojit 3-páskový DTS takto:

- 1. páska obsahuje přijímaný vstup.
- 2. páska je pracovní, obsahuje kopii první pásky, při neúspěchu se znovu obnoví z 1. pásky.
- 3. páska obsahuje záznam provedených přechodů, díky čemuž lze provádět backtracking a ověřit všechny možnosti.

33. Nerozhodnutelnost (problém zastavení TS, princip diagonalizace a redukce, Postův korespondenční problém). TIN

Turingův stroj (**TS**) se nazývá **úplný**, pokud pro každý vstup zastaví. Jazyk L se nazývá:

- **rekurzívně vyčíslitelný**, pokud existuje TS M, který jej přijímá – tzn. pro každé slovo patřící do L se M zastaví a přijme jej, pro slovo mimo L se M buď zastaví a odmítne, nebo cyklí donekonečna.
- **rekurzívní**, pokud existuje úplný TS M, který jej přijímá – tzn. pro slovo z L se M zastaví a přijme, pro slovo mimo L se M zastaví a odmítne. Říkáme, že M **rozhoduje** L.

Rozhodovací problém (**decision problem**) je funkce f s oborem hodnot {true, false}, ta je určena definičním oborem a jeho podmnožinou, pro kterou je výstup true. V teorii formálních jazyků reprezentujeme takové problémy řetězci, tzn. problém p je jazyk $L_p = \{\omega \in \Sigma^* \mid \omega = \text{code}(p) \wedge f(\text{p}) = \text{true}\}$. Problém p se nazývá:

- **rozhodnutelný**, pokud L_p je rekurzívní jazyk.
- **nerozhodnutelný**, když není rozhodnutelný.
- **částečně rozhodnutelný**, L_p je rekurzívně vyčíslitelný jazyk.

Jazyky typu 0 jsou přijímány Turingovými stroji, existují však i jazyky mimo tuto třídu – ty reprezentují nerozhodnutelné problémy. K důkazu nerozhodnutelnosti se nejčastěji používají techniky **diagonalizace** a **redukce**.

Problém zastavení (**halting problem**) je problém rozhodování, zda daný TS při daném vstupu zastaví v konečném čase či nikoliv. Tento problém je nerozhodnutelný.

Důkaz **diagonalizací**:

1. Zavedeme tzv. **univerzální TS**, což je TS schopný simulovat jakýkoliv jiný TS, který je jeho vstupem jakožto zakódovaný řetězec. Kódování TS lze provést zakódováním stavů, symbolů a přechodových pravidel do binární abecedy.
2. Problém zastavení odpovídá rozhodování jazyka $HP = \{M\#w \mid M \text{ zastaví při vstupu } w\}$, kde M je zakódovaný TS a w je vstup.
3. Předpokládejme nyní, že existuje TS M realizující zobrazení f: $X \times Y \rightarrow \{c, z\}$, které kódu jakéhokoliv TS X a vstupu Y přiřadí c, pokud se X zacyklí na Y, jinak z. Toto zobrazení můžeme znázornit tabulkou:

↓ TS, → vstup	ε	0	1	10	...
x_1	a_{11}	a_{21}	a_{31}	a_{41}	
x_2	a_{12}	a_{22}	a_{32}	a_{42}	
x_3	a_{13}	a_{23}	a_{33}	a_{43}	
x_4	a_{14}	a_{24}	a_{34}	a_{44}	
...					...

4. Nyní zkonstruujme TS N, který jako svůj vstup vezme kódovaný TS X a vstup Y, odsimuluje X se vstupem Y a pokud X přijme, N odmítne, jinak N přijme (negace diagonály v tabulce).
5. TS N se evidentně nenachází v tabulce, jelikož se liší od každého TS x_i minimálně ve výstupu pro vstup i (např. N nemůže být x_3, protože výstup N je pro vstup 1 z definice opačný než výstup x_3, atd.). To je ovšem spor s předpokladem, že zobrazení f a tedy i tabulka jsou definována pro libovolný TS. TS M rozhodující problém zastavení proto nemůže existovat a problém je nerozhodnutelný. ∎

Podobně lze např. dokázat, že množina 2^{Σ^*} (množina všech jazyků nad Σ) je nespočetná (sloupce: řetězce nad Σ, řádky: jazyky, buňky: patří/nepatří) nebo že množina reálných čísel je nespočetná (uvažujeme pouze čísla mezi 0 a 1 v binární reprezentaci, slupce: bity, řádky: čísla).

Redukce je technika algoritmického převodu problému na jiný problém. Je to tedy vyčíslitelná (redukční) funkce f, která každé instanci I problému P přiřadí instanci $f(I)$ problému Q tak, že $f(I)$ je právě řešením I. Princip je následující:

- Víme, že jazyk A není rekurzívní (rekurzívně vyčíslitelný).
- Zkoumáme jazyk B.
- Ukážeme, že A lze pomocí úplného TS převést (redukovat) na B.
- To ale znamená, že B rovněž není rekurzívní (rekurzívně vyčíslitelný), jinak by šlo použít úplný (neúplný) TS přijímající B a příslušnou redukci k sestrojení úplného (neúplného) TS přijímajícího A, což by byl spor.
- Redukci lze tedy použít i k dokazování, že daný jazyk je rekurzívní.

Formálně je redukce jazyka $A \subset \Sigma^*$ na jazyk $B \subset \psi^*$ rekurzívně vyčíslitelná funkce
σ: $\Sigma^* \rightarrow \psi^*$ taková, že $w \in A \Leftrightarrow \sigma(w) \in B$.
Existuje-li redukce jazyka A na B, říkáme, že A je **redukovatelný** na B, což zapisujeme $A \leq B$. Platí-li $A \leq B$, pak:

- *A* není rekurzívně vyčíslitelný ⇒ *B* není rekurzívně vyčíslitelný.
- *A* není rekurzívní ⇒ *B* není rekurzívní.
- *B* je rekurzívně vyčíslitelný ⇒ *A* je rekurzívně vyčíslitelný.
- *B* je rekurzívní ⇒ *A* je rekurzívní.

Postův korespondenční problém (**PCP**) je podobně jako problém zastavení důležitým nerozhodnutelným problémem. Jeho definice je následující:

- **Postův systém** je neprázdný seznam dvojic neprázdných řetězců: $S = <(\alpha_1, \beta_1), ..., (\alpha_k, \beta_k)>$.
- Řešení postova systému je neprázdná posloupnost přirozených čísel $I = <i_1, ... i_m>$ taková, že: $\alpha_{11}\,\alpha_{12}...\alpha_{im} = \beta_{i1}\,\beta_{i2}...\,\beta_{im}$.
- PCP zní: existuje pro daný Postův systém řešení?

Riceova věta je dost složitá věta, která říká, že každá netriviální vlastnost (neformálně nějaké tvrzení, které není buď vždy pravdivé nebo vždy nepravdivé) rekurzívně vyčíslitelných jazyků je nerozhodnutelná,

34. Parciální rekurzivní funkce. TIN

Při zkomání vyčíslitelnosti funkcí se bez újmy na obecnosti omezujeme na funkce tvaru:
$f: N^m \rightarrow N^n$, kde $N = \{0, 1, 2, ...\}$.

n-tici značíme jako $\bar{x}$.
Funkce dělíme na:

- **totální**: Funkce je definována na celé množině (např plus).
- **parciální**: Může a nemusí být definována na celé množině.
- **striktně parciální**: Existuje alespoň jedna hodnota, pro kterou funkce není definována (např. div).

Nejdříve definujeme **počáteční funkce**:

- **nulová funkce** (zero): $\xi() = 0$
- **funkce následníka** (successor): $\sigma: N \rightarrow N,\ \sigma(x) = x + 1$
- **projekce** (projection): $\pi^n_k: N^n \rightarrow N$, vybírá $k$tou složku z $n$tice, např.: $\pi^3_2(7,6,4) = 6$
  speciální případ: $\pi^n_0: N^n \rightarrow N^0$ (tzn. prázdná $n$tice)

Dále definujeme třídu **primitivně rekurzivních** funkcí, které vzniknou z počátečních následujícími způsoby:

- **kombinace**: Výsledek dvou funkcí daný do dvojice, např.: $\pi^3_1 \times \pi^3_3\ (4,12,8) = (4, 8)$.
  Formálně pro dvě funkce $f: N^k \rightarrow N^m$ a $g: N^k \rightarrow N^n$ je výsledkem nová funkce:
  $f \times g: N^k \rightarrow N^{m+n}$, $f \times g(\bar{x}) = (f(\bar{x}), g(\bar{x}))$, $\bar{x} \in N^k$.
- **kompozice**: Dává výsledek jedné funkce jako argument druhé funkce, např.: $\sigma \circ \xi() = 1$.
  Formálně je výsledkem kompozice dvou funkcí $f: N^k \rightarrow N^m$ a $g: N^m \rightarrow N^n$ nová funkce:
  $g \circ f: N^k \rightarrow N^n$, $g \circ f(\bar{x}) = g(f(\bar{x}))$, $\bar{x} \in N^k$.
- **primitivní rekurze**: Umožňuje vytvořit rekurzivní vyhodnocování na základě dvou funkcí.
  Formálně ze dvou funkcí $g: N^k \rightarrow N^m$ (funkce pro koncový případ) a $h: N^{k+m+1} \rightarrow N^m$ (funkce pro nekoncový případ) dostáváme novou funkci:
  $f: N^{k+1} \rightarrow N^m$:
  $f(\bar{x}, 0) = g(\bar{x})$
  $f(\bar{x}, y + 1) = h(\bar{x}, y, f(\bar{x},y))$, $\bar{x} \in N^k$. ($\bar{x}$ je původní vstup, y zanoření a $f(\bar{x},y)$ nižší úroveň)

Všechny primitivně rekurzivní funkce jsou totální (důkaz: počáteční funkce jsou totální, kombinací, kompozicí a primitivní rekurzí dostaneme totální funkce).

Existují funkce, které jsou vyčíslitelné, ale nejsou primitivně rekurzivní. Jsou to všechny striktně parciální funkce (např. div), ale i některé totální funkce (např. Ackermanova funkce).

Třída totálních vyčíslitelných funkcí se nazývá **μ-rekurzivní** funkce.

Třída **parciálně rekurzivních** funkcí je třída primitivně rekurzivních funkcí plus funkce, které lze získat tzv. **minimalizací**.

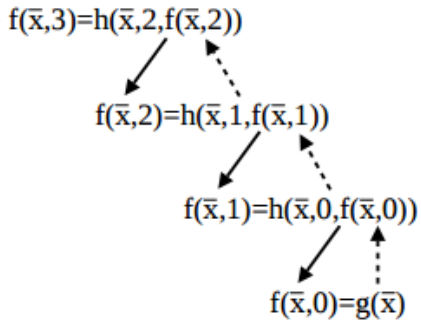


Illustration 59: ilustrace primitivní rekurze

Minimalizace vytvoří funkci $f: N^n \rightarrow N$ z jiné funkce $g: N^{n+1} \rightarrow N$ tak, že $f(\bar{x})$ je nejmenší y takové, že:

1. $g(\bar{x}, y) = 0$
2. $g(\bar{x}, z)$ je definována pro všechna $z < y$, $z \in N$.

Píšeme: $f(\bar{x}) = \mu y[g(\bar{x},y) = 0]$.

Třída **Turingovsky vyčíslitelných funkcí** obsahuje funkce, které jdou vyčíslit nějakým TS (TS může funkce vyčíslovat tak, že v počátečním stavu přečte parametry na pásce, provede výpočet a zapíše výsledek na pásku) a je ekvivalentní třídě parciálně rekurzivních funkcí. Důkaz se provede dvěma směry:

- Dokážeme, že jakoukoliv parciálně rekurzivní funkci může vyčíslit nějaký TS. Nalezneme TS pro počáteční funkce a potom TS pro realizaci kombinace, kompozice, primitivní rekurze a minimalizace.
- Dokážeme, že jakýkoliv TS můžeme reprezentovat parciálně rekurzivní funkcí. Tzn. nalezneme funkce simulující chod TS.

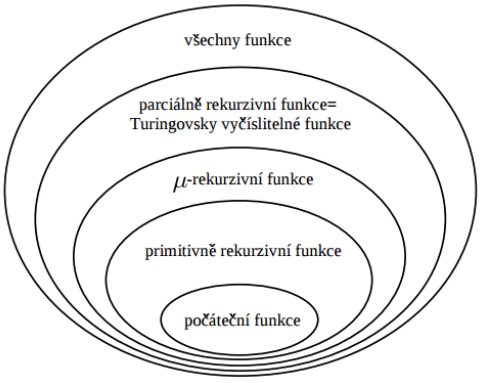
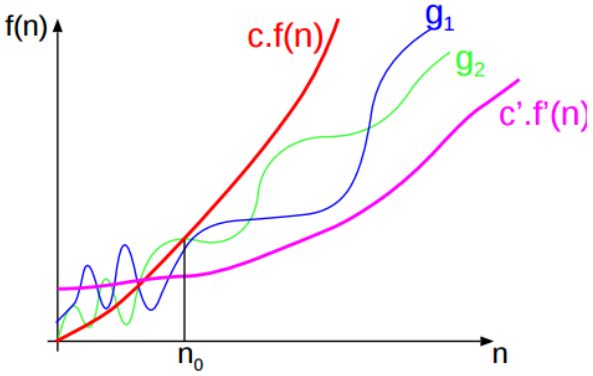


Illustration 60: třídy funkcí

35. Časová a paměťová složitost (třídy složitosti, úplnost, SAT problém). TIN

Určování složitosti využívá Turingovy stroje a jiné formální prostředky k určování tříd složitosti, které říkají, jak rychle rostou čas nebo paměť (souhrně zdroje) potřebné k výpočtu v závisloti na vstupních datech. Mějme TS M. Složitost je funkce $Compl_M$: $N \to N$, jejímž vstupem je délka vstupu a výstupem cena výpočtu. Tato funkce může být definována různě podle toho, kterou z následujících možností uvažujeme:

- nejhorší případ
- nejlepší případ
- průměrný případ (průměr všech možných výsledků výpočtu vážený jejich pravděpodobnostmi)



Podle zdrojů rozlišujeme složitost na:

- **časovou**: počet kroků (přechodů) provedených TS při výpočtu
- **prostorovou**: počet buněk pásky TS potřebný pro výpočet

Platí: je-li časová složitost výpočtu TS rovna n, pak prostorová složitost tohoto výpočtu není větší než $n + 1$.

Většinou nás nezajímá přesná hodnota složitost, ale spíše její třída. Proto přistupujeme k tzv. **asymptotické složitosti**. Nechť F je množina funkcí f: $N \to N$. Pro danou funkci $f \in F$ definujeme následující třídy funkcí:

- **asymptotické horní omezení** funkce $f(n)$ je množina
  $O(f(n)) = \{g(n) \in F \mid \exists c \in R^+, \exists n_0 \in N\ \forall n \in N: n \geq n_0 \Rightarrow 0 \leq g(n) \leq c \cdot f(n)\}$.
- **asymptotické dolní omezení** funkce $f(n)$ je množina
  $\Omega(f(n)) = \{g(n) \in F \mid \exists c \in R^+, \exists n_0 \in N\ \forall n \in N: n \geq n_0 \Rightarrow 0 \leq c \cdot f(n) \leq g(n)\}$.
- **asymptotické oboustranné omezení** funkce $f(n)$ je množina
  $\Theta(f(n)) = \{g(n) \in F \mid \exists c_1, c_2 \in R^+, \exists n_0 \in N\ \forall n \in N: n \geq n_0 \Rightarrow 0 \leq c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)\}$.

Můžeme např. tvrdit, že časová složitost insert-sortu patří do $O(n^2)$.

Mějme funkce t, s: $N \to N$ a nechť S_M, resp. T_M značí časovou, resp. prostorovou složitost TS M (jsou to tedy také funkce). Definujeme následující třídy složitosti (všimněme si, že jsou to třídy jazyků, stejně jako např v Chomského hierarchii):

- ***DTime***$[t(n)] = \{L \mid \exists$ k-páskový DTS M: $L = L(M)$ a $T_M \in O(t(n))\}$.
- ***NTime***$[t(n)] = \{L \mid \exists$ k-páskový NTS M: $L = L(M)$ a $T_M \in O(t(n))\}$.
- ***DSpace***$[s(n)] = \{L \mid \exists$ k-páskový DTS M: $L = L(M)$ a $S_M \in O(s(n))\}$.
- ***NSpace***$[s(n)] = \{L \mid \exists$ k-páskový NTS M: $L = L(M)$ a $S_M \in O(s(n))\}$.

Běžně užívané třídy jsou potom:

- deterministický/nedeterministický polynomiální čas:
  P = $U_{k=0}^{\infty} DTime(n^k)$ NP = $U_{k=0}^{\infty} NTime(n^k)$
- deterministický/nedeterministický polynomiální prostor:
  PSPACE = $U_{k=0}^{\infty} DSpace(n^k)$ NSPACE = $U_{k=0}^{\infty} NSpace(n^k)$
- deterministický/nedeterministický logaritmický prostor (podtřída PSPACE, resp. NSPACE):
  LOGSPACE = $U_{k=0}^{\infty} DSpace(k \log(n))$ NLOGSPACE = $U_{k=0}^{\infty} NSpace(k \log(n))$
- deterministický/nedeterministický exponenciální čas (nadtřída P, resp. NP):
  EXP = $U_{k=0}^{\infty} DTime(2^{n^k})$ NEXP = $U_{k=0}^{\infty} NTime(2^{n^k})$
- deterministický/nedeterministický k-exponenciální čas (nadtřída EXP, resp. NEXP):
  k-EXP = $U_{k=0}^{\infty} DTime(2^{\cdot^{\cdot^{\cdot^{n^k}}}})$ k-NEXP = $U_{k=0}^{\infty} NTime(2^{\cdot^{\cdot^{\cdot^{n^k}}}})$
- ...

Jedním z největších otevřených problémů informatiky je, zda se třída P rovná třídě NP (předpokládá se že ne).

Problémy ze třídy PSPACE se často řeší tak, že se zhorší jejich prostorové nároky výměnou za alespoň částečné zlepšení časové náročnosti (např. z $O(2^{n^2})$ na $O(2^n)$).

Úplnost a **těžkost** nám dovolují definovat spodní hranici složitosti jazyků pomocí výše uvedených tříd.

Mějme třídu funkcí *R*. Jazyk L_1 je *R* **redukovatelný** na jazyk L_2, což zapisujeme $L_1 \leq^m_R L_2$, jestliže existuje funkce *f* z *R* taková, že $w \in L_1 \Leftrightarrow f(w) \in L_2$. Neformálně řečeno, problém reprezentovaný jazykem L_1 můžeme převést (v čase daném třídou *R*, např. polynomiálním) na problém reprezentovaný jazykem L_2.

Nechť *R* je třída funkcí a *C* třída jazyků. Jazyk L_0 je ***C*-těžký** (*C*-hard) vzhledem k *R* redukovatelnosti, jestliže $\forall L \in C$: $L \leq^m_R L_0$. *C*-těžký jazyk nemusí patřit do *C* a může být ve složitější třídě.

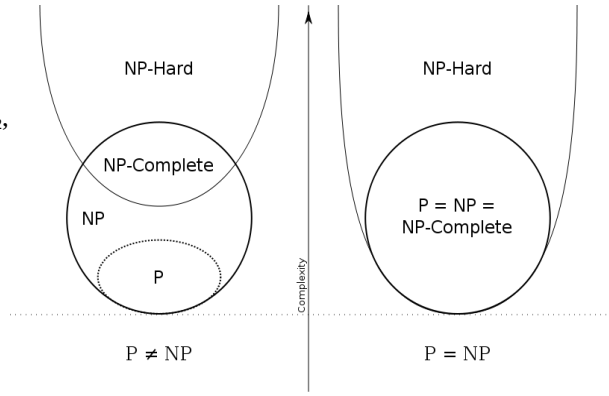


Illustration 61: NP-těžké a NP-úplné jazyky

Nechť *R* je třída funkcí a *C* třída jazyků. Jazyk L_0 je ***C*-úplný** (*C*-complete) vzhledem k *R* redukovatelnosti, jestliže L_0 je *C*-těžký a $L_0 \in C$.

Neformálně řečeno např. NP-těžký problém je problém, který je alespoň tak složitý, jako nejsložitější problém z NP (např. halting problem je NP-těžký, ale ne NP-úplný). NP-úplný problém je potom takový NP (a NP těžký) problém, na který je polynomiálně redukovatelný každý jiný problém z NP (např. problém obchodního cestujícího).

SAT problém (satisfiability) je NP-úplný problém, který lze použít k důkazu NP-těžkosti jiných problémů pomocí redukce. Formulace SAT problému je následující:

Máme množinu booleovských proměnných $V = \{v_1, v_2, ..., v_n\}$ a formuli *X* složenou z těchto proměnných a logických spojek ¬ (negace), ∧ a ∨ (normální disjunktní forma). Je tato formule splnitelná? Tzn. existuje přiřazení hodnot TRUE a FALSE proměnným množiny *V* takové, že se *X* vyhodnotí jako TRUE?

SAT problém můžeme samozřejmě reprezentovat jazykem, řekněme L_{SAT}. Ten sestrojíme tak, že vymyslíme kódování formule a prvky jazyka potom budou všechny řetězce reprezentující splnitelné formule.

Vícepáskový TS sníží oproti jednopáskovému složitost maximálně polynomiálně.

36. Interference světla (skládání dvou a více koherentních vln, intenzita složené vlny, interferenční člen, konstruktivní a destruktivní interference, princip interferometru). FYO

Interference je vzájemné ovlivňování vln, které se střetávají. Interference je možná díky **principu superpozice** – setkají-li se dvě vlny v místě prostoru, jejich výchylky se algebraicky sčítají, dále potom vlny postupují stejně, jako kdyby se nikdy nesetkaly. Interferenci dělíme na

- **konstruktivní** – Výsledná vlna je zesílená. Nastává pro fázové rozdíly vln = 0, ± 2π, ± 4π, ...
- **kestruktivní** – Výsledná vlna je zeslabená. Nastává pro fázový rozdíl rovný lichému násobku ± (2*m* + 1) π, kde *m* je celé číslo.

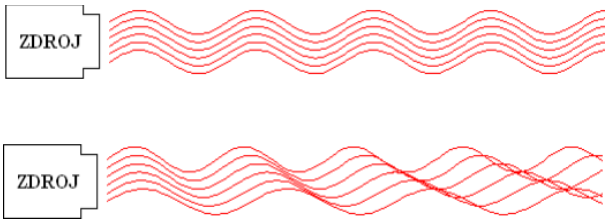


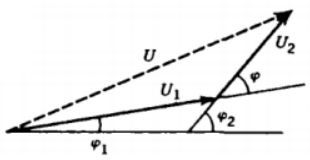
Illustration 62: koherentní (nahoře) vs nekoherentní vlna

Intenzita vlny udává přenesený výkon na jednotku plochy kolmé ke směru jejího šíření, tedy W/m^2. Intenzita je skalár a spočítá se jako je abs. hodnota komplexní amplitudy (což je vektor) na druhou:

$$I = |U(r)|^2$$

Výsledek interference, tzn. intenzita výsledné vlny, závisí na **intenzitě** a **fázi** (argument vlnové funkce, v podstatě posun). **Fázor** (vektor o velikosti amplitudy a s výchylkou fáze), neboli také **komplexní amplituda**, výsledné vlny je součtem fázorů vstupních vln:

$U(r) = U_1(r) + U_2(r)$



Pokud uvažujeme dvě vlny o stejné frekvenci a amplitudě, pak je intenzita výsledné vlny dána tzv. **interferenční rovnicí**:

$$I = I_1 + I_2 + 2\sqrt{(I_1 I_2)}\cos(\phi_2 - \phi_1)$$

kde $\phi_2 - \phi_1$ je fázový rozdíl a $2\sqrt{(I_1 I_2)}\cos(\phi_2 - \phi_1)$ je tzv. **interferenční člen**.

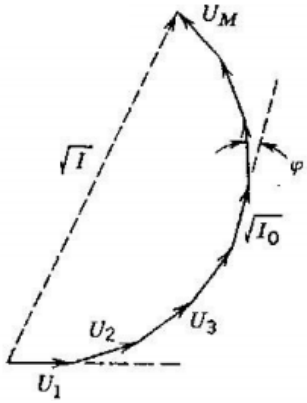


Illustration 63: komplexní amplituda (fázor) interference M vln

Abychom interferenci mohli pozorovat, musí být vlny **koherentní**. Dvě vlny jsou koherentní, pokud mají stejnou frekvenci a jejich fázový rozdíl v libovolném bodě se s časem nemění. Koherentní vlnu můžeme z normálního světla získat např. tak, že jej necháme projít úzkou štěrbinou (to z dále nespecifikovaného důvodu sjednotí fáze vlny) a potom filtrem, který propustí jenom jednu vlnovou délku.

Interferují-li dvě rovinné vlny odchýlené o úhel θ, můžeme na stínítku zachytit interferenční obrazec. Ten je periodický – perioda je λ /sinθ.

Interferenční obrazec rovinné a kulové vlny jsou soustředné kruhy.

Interferenční obrazec dvou sférických vln tvoří proužky s konstantní periodou, jak dokazuje **Youngův pokus**.

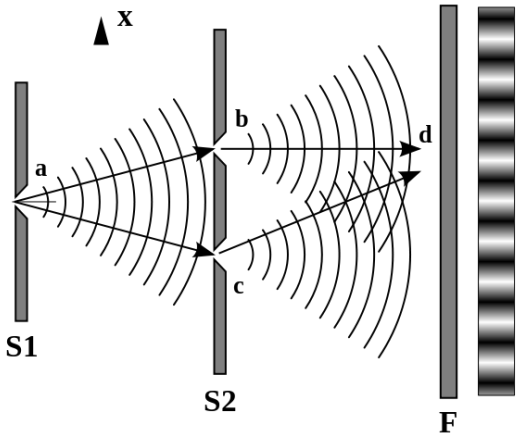


Illustration 64: Youngův pokus

Interferometr je přístroj využívající interferenci k provádění velmi přesných měření (např. délek, indexů lomu, struktury materiálu apod.). Dělíme je na:

- **lineární** – Štěpí vlnu na dva paprsky, referenční a měřený. Ty potom procházejí různými drahami, na konci jsou znovu spojeny, interferují spolu, čímž vytvářejí studovaný pobrazec.
- **nelineární** – Při fázovém zpožďování využívají nelineární jevy, jako např. závislost indexu lomu na intenzitě.

Mezi používané interferometry patří:

- **Michelsonův** – Světlo dopadá na dělit paprsků uprostřed, kde

se dělí na dva svazky, každý jde jiným směrem a odráží se od zrcadel zpět. Do cesty odraženému paprsku se vkládá kompenzační destička, aby zaručila stejnou dráhu paprsku, jako má paprsek, který děličem prošel. Měření můžeme provádět tak, že jednomu paprsku vložíme do cesty měřený předmět a pozorujeme změnu interferenčního obrazce. Tento interferometr byl použit pro detekci gravitačních vln a vyvrátil existenci éteru.

- **Machův-Zenderův** – Měřený paprsek neprochází měřeným prostorem dvakrát, proto není tak přesný jako Michelsonův interferometr. Nevýhodou je také nutnost použít velmi kvalitní optické prvky. Měřené objekty se vkládají do jedné větve interferometru. Používá se např. pro měření změny tlaku vzduchu aerodynamických tunelech.
- **Fabryův-Perotův** – Je tvořen dvěma rovnoběžnými polopropustnými zrcadly oddělenými vzduchovou mezerou, jejíž velikost můžeme regulovat. Světlo jdoucí z jednoho bodu a v jednom směru se mezi zrcadly mnohanásobně odráží a na konci je soustředěno do jediného bodu, kde se zobrazuje výsledná interferenční hodnota tohoto paprsku se sebou samotným. Tato hodnota závisí na vzdálenosti mezi zrcadly. Využívají se tam, kde potřeba vysoké rozlišení, nebo v úzkopásmových interferenčních filtrech.

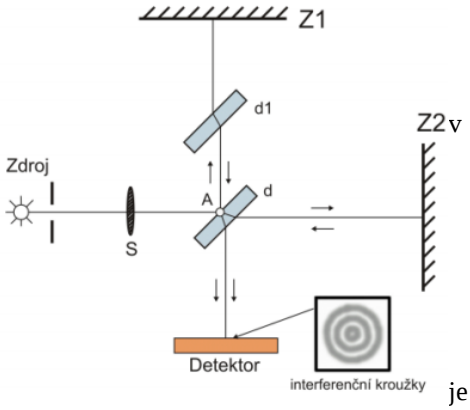


Illustration 65: Michelsonův interferometr

je Interferenci můžeme v reálném světě pozorovat např. na vrstvě oleje, což je tzv. **interference na tenké vrstvě**. Dochází zde k rozkladu dopadajícího bílého světla (obsahujícího všechny vlnové délky) – část dopadajícího světla se odrazí od vrchní části vrstvy, část projde a odrazí se od spodní části vrstvy. Tloušťka olejové vrstvy udává konstantní dráhový rozdíl, který musí paprsek, který se odrazí od spodní části vrstvy, projít. Tyto dva paprsky spolu poté interferují. V závislosti na úhlu, pod kterým vrstvu pozorujeme se mění dráhový rozdíl. Každý dráhový rozdíl představuje pro určité vlnové délky konstruktivní a destruktivní interferenci, proto z bílého světla nakonec zůstanou jen některé barvy.

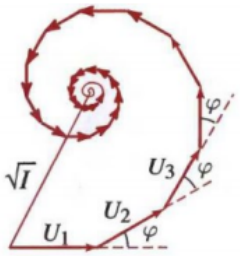


Illustration 66: výsledná interference ve Fabryově-Perotově interferometru je interference nekonečně mnoha vln s klesající amplitudou a konstantním fázovým rozdílem

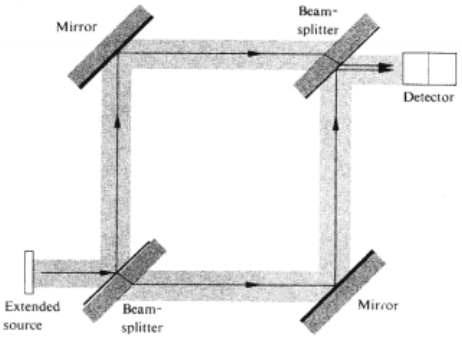


Illustration 67: Machův-Zenderův interferometr

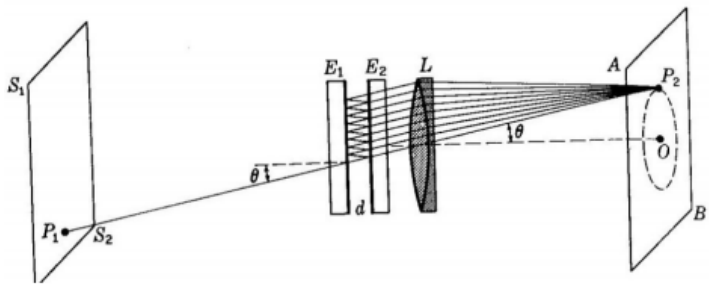


Illustration 68: Fabryův-Perotův interferometr

37. Difrakce světla (rozložení intenzity světla za obdélníkovou a kruhovou štěrbinou, Airyho obrazec, rozlišovací schopnost optických přístrojů, oka). FYO

Difrakce (**ohyb**) je v optice jev, kdy se světlo dostává díky své vlnové povaze do geometrického stínu, tzn tam, kam by se podle čistě geometrické optiky dostat nemělo. Difrakce je důsledkem Huygensova principu. Ohyb světla je patrný tam, kde světlo naráží na překážku podobných rozměrů, jako je vlnová délka světla (štěrbina, hrana, ...).

Illustration 69: difrakce

Huygensův [**hajchensův**] **princip** říká, že každý bod vlnoplochy je sám bodovým zdrojem vlnění.

Fraunhoferova difrakce je model difrakce, kdy výsledný obrazec zachycujeme ve velké vzdálenosti od vstupní štěrbiny a vlnu aproximujeme rovinnou vlnou (pokud je vzdálenost malá, je vlna sférická a používá se tzv. Fresnelova difrakce).

Fraunhoferova difrakce na 1D štěrbině nám na stínítku zobrazí střídavě maxima a minima. Podmínka pro minima je:

$$a \sin \beta = m \lambda$$

kde a je šířka štěrbiny, β je úhel od štěrbiny k promítnutému bodu, λ je vlnová délka světla a m celé číslo.

Difrakční obrazec na 2D apertuře je jeho Fourierovou transofrmací.

Na kruhové apertuře vzniká obrazec, který nazýváme **Airyho obrazec**. Střední kruh obrazce se nazývá **Airyho disk** a dopadá na něj 84% světla.

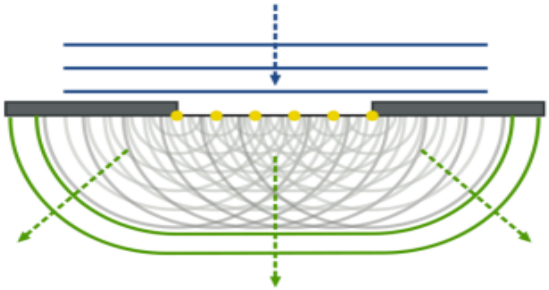


Illustration 70: Huygensův princip

Pokud zobrazujeme obraz reálným zobrazovacím systémem, pak obrazem každého bodového zdroje je vždy Airyho obrazec. Na základě tohoto faktu definujeme **rozlišitelnost** dvou bodů:

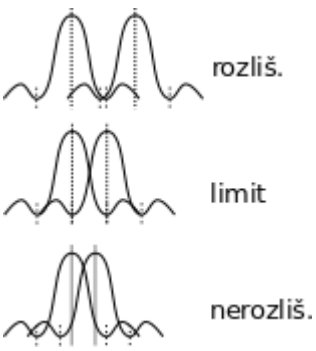
Rayleighovo kritérium říká, že dva body jsou rozlišeny, jestliže maximum Airyho disku prvního bodu spadá alespoň do minima difrakčního obrazce druhého bodu. Minimální rozlišovací úhel je dán vztahem:

$$\theta = 1{,}22 \lambda / D,$$

kde λ je vlnová délka světla a D průměr otvoru.

Rozlišovací úhel oka je 1' (1 minuta, šedesátina stupně).

Př.: Která barva přestane při oddalování být nejdřív rozlišitelná? červená (největší λ, tzn. největší potřebný rozlišovací úhel)



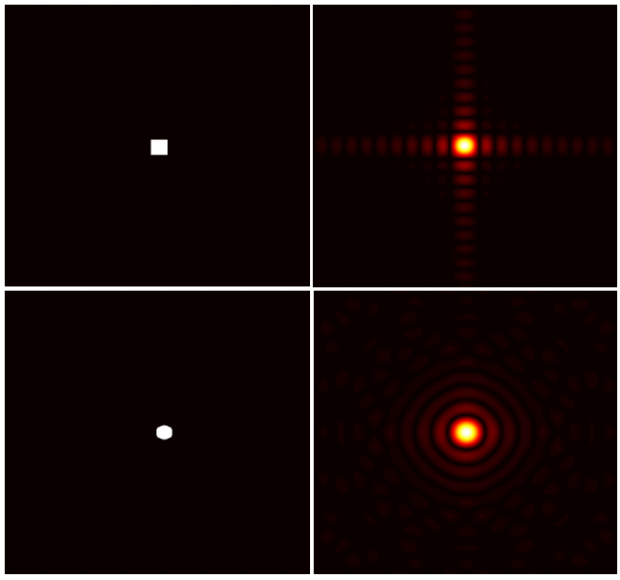


Illustration 71: apertury a jejich obrazce (Fourierovy transformace)

38. Polarizace světla (přirozené a lineárně polarizované světlo, polarizační rovina, způsoby polarizace světla, elipticky polarizované světlo, polarizační filtry). FYO

Polarizace je vlastnost vlny udávající, v jakých směrech vlna kmitá (proto o ní mluvíme jenom u vln, které mohou kmitat ve více různých směrech, jako např. světlo). Elektromagnetické vlny mají dvě složky, elektrickou a magnetickou, které jsou vždy ve fázi a navzájem kolmé, proto když zde mluvíme o polarizaci, mluvíme z konvence o polarizaci elektrické složky. I když lidské oko není na polarizaci citlivé, má polarizace vliv na různé jevy, jako např. množství světla odraženého na rozhraní, množství světla absorbovaného na rozhraní, rozptyl světla v látce apod. Hmyz dokáže polarizaci vnímat přímo. Polarizace se využívá v LCD displejích, polarizačních filtrech (např. 3D brýle, brýle proti odleskům, ...), měření napětí materiálů apod.

Běžné světlo (denní světlo, žárovka, ...) je nepolarizované, tzn. orientace jeho kmitů je nahodilá. Abychom polarizaci dále specifikovaly, zavedeme tzv. **polarizační rovinu**, což je dvourozměrná rovina, v níž znázorňujeme pohyb kmitajícího koncového bodu vlnění elektrické složky. Kmitání tak můžeme rozdělit do dvou složek – *x* a *y*. Podle výsledného obrazce potom rozlišujeme následující typické případy:

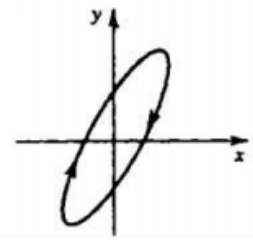


Illustration 72: polarizační rovina – eliptická polarizace

- **lineární polarizace** – Fázový rozdíl *x* a *y* složky je 0 nebo π, obrazcem je úsečka.
- **eliptická polarizace** – Obrazcem je elipsa, můžeme dále rozlišovat pravotočivou a levotočivou eliptickou polarizaci. Takováto vlna se dá vyrobit použitím lineárního polarizátoru a speciální destičky, která mění polarizaci světla.
- **kruhová polarizace –** Speciální případ eliptické polarizace, kdy je fázový rozdíl *x* a *y* složky $\pm \pi / 2$ a amplituda složek si je rovna, obrazcem je kružnice.

Polarizátor je zařízení, které propouští složku elektrického pole ve směru určité dané orientace a blokuje složku kolmou k této orientaci. Dosahuje toho např. selektivní absorbcí, odrazem od izotropního prostřední (tj. prostředí, kde rychlost šíření světla nezávisí na směru) nebo selektivním odrazem či lomem na povrchu anizotropního prostředí. Polarizátorem tedy prochází průmět vektoru intenzity do osy polarizace.

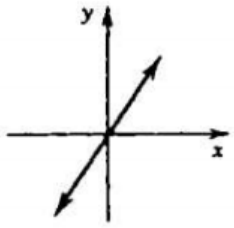


Illustration 73: lineární polarizace

Světlo se může rovněž polarizovat odrazem na rozhraní dvou prostředí – složka E odraženého paprsku kmitá převážně v rovině kolmé na rovinu dopadu (složka E paprsku lomeného do prostředí naopak kmitá převážně v rovině dopadu). Odraz a lom je dán **Fresnelovými rovnicemi**. Polarizace závisí na úhlu dopadu jen částečně. **Brewsterův úhel** je úhel, při němž lomený a odražený paprsek svírají 90° (závisí na indexu lomu obou prostředí) – při tomto úhlu je odražený paprsek polarizovaný úplně (lomený jen částečně) a to ve směru, jaký je znázorněn na obr. Tento úhel se dá využít např. ke konstrukci skel (Brewsterova okna) u nichž dochází k velmi malým ztrátám kvůli odrazu.

Polarizační filtr je filtr propouštějící pouze světlo o určité polarizaci, používá se např. při fotografování, ve slunečních brýlích apod. Filtry se označují jako HN-*X*, kde *X* je počet procent propuštěného světla.

BONUS: důležité pojmy FYO

vlny dělíme na:

- podélné/příčné (např. zvuk vs světlo)
- mechanické/elektromagnetické/vlny hmoty (např. lano vs světlo vs zvuk)
- kulové/parabolické/rovinné

Vlnová funkce je libovolná funkce argumentu φ = t -z/c (φ: fáze, t: čas, z: místo, c: rychlost postupu fáze). **Harmonická vlna** má tvar:

$$u = U \cos(w(t - z/c) + \varphi_0)$$

Pulz je neperiodická vlna.

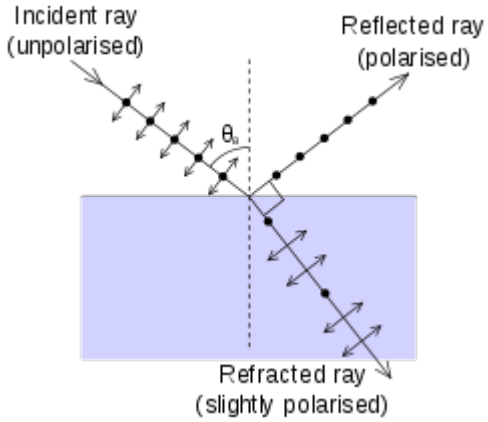


Illustration 74: Brewsterův úhel

39. Holografie a laser (holografický kód, jeho dekódování, mimoosový hologram, objemový hologram, vztah holografie a laseru). FYO

Holografie (holos – úplný, grafie – záznam) je metoda záznamu a rekonstrukce trojrozměrného obrazu založená na interferenci světla, objevená roku 1947. V praxi je holografický záznam podobný fotografii – jedná se o záznam na speciálním médiu plochém médiu, které když nasvětlíme laserem, uvidíme skrz něj původní 3D obraz, který se mění podle úhlu, v němž jej pozorujeme (a je tedy i stereoskopický).

Illustration 75: holografický záznam pozorovaný ze dvou různých úhlů

Záznam na médiu se nazývá **hologram** a nese informaci jak o intenzitě (= $|U(x,y)|^2$, což je **komplexní amplituda**, tedy velikost a fáze, v bodě x, y. Intenzitu zaznamenává i klasická fotografie), tak o fázi světla odraženého od zaznamenaného obrazu (tu fotografie postrádá). Fáze se zaznamenává tak, že se převede na intenzitu pomocí interference vlny odraženého světla (tzv. **předmětová vlna**) s tzv. **referenční vlnou**. Záznam je tedy interferenčním obrazcem, ze kterého lze zpětně rekonstruovat předmětovou vlnu tak, že jej osvětlíme laserem o vlnové délce, která byla použita při záznamu, čímž pomocí difrakce (ohybu) vzniká původní obraz. Laser se musí používat kvůli nutnosti mít koherentní světlo, aby docházelo k interferenci. Pod mikroskopem vypadá hologram jako velmi jemná mřížka, která je lokálně různě deformovaná (jedná se v podstatě o difrakční mřížku) – holografická emulze musí mít vysokou rozlišovací schopnost.

Holografický kód je způsob, jakým hologram zaznamenáváme. Definujeme **komplexní amplitudovou propustnost** vlny procházející médiem jako poměr komplexních amplitud na začátku ($z = d_1$) a na konci ($z = d_2$) média:

$\tau(x,y) = U(x,y,d_1) / U(x,y,d_2)$.

Hologram je transparentní médium, jehož amplitudová propustnost v každém bodě $[x,y]$ je úměrná intenzitě zaznamenaného interferenčního obrazce. Platí:

$\tau = I_0 + I_r + U_0 U_r^* + U_0^* U_r$,

kde I_0 a U_0, resp. I_r a U_r jsou intenzita a kompl. ampl. předmětové, resp. referenční vlny (* je komplexní sdružení).

Rekonstruovaná vlna je potom:

$U = \tau U_r = I_0 U_r + I_r U_r + U_0 I_r + U_0^* U_r^2$,

což je vlna obsahující informaci o původní vlně (U_0).

Mimoosový (**off-axis**) **hologram** je vylepšení holografie, kdy při záznamu a rekonstrukci volíme specifický úhel předmětové a referenční vlny tak, abychom dosáhli určitých výhod, které zahrnují dobrou oddělitelnost jednotlivých složek obrazu (bude při pozorování "ostřejší"), kontrastnější záznam interferenčního obrazce, lepší expoziční vlastnosti a další.

Objemový (volume) **hologram** je hologram, kdy k záznamu používáme medium, jehož tloušťka je mnohem větší než vlnová délka zaznamenávaného světla. Záznam v médiu je nyní prostorový interferenční obrazec. Výhodou je např. to, že můžeme pro rekonstrukci použít bílé světlo, protože k rekonstrukci dojde jedině při osvětlení vlnou, která splňuje tzv. **Braggovu podmínku**, ostatní vlnové délky se neprojeví. Pokud při rekonstrukci změníme úhel referenšní vlny,

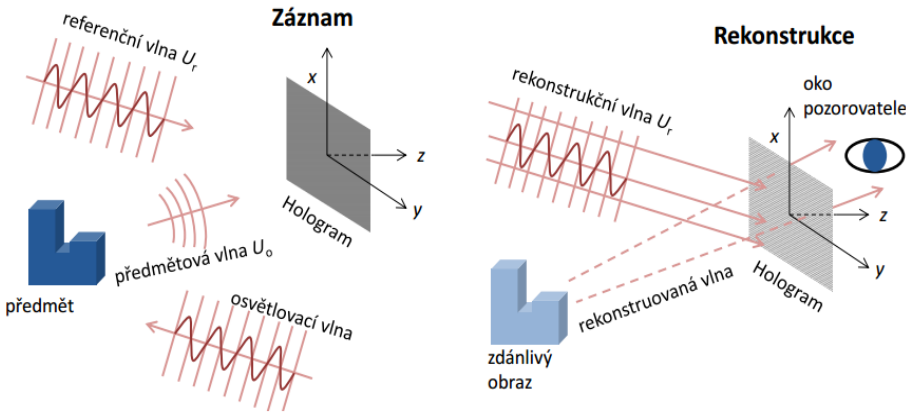


Illustration 76: záznam a rekonstrukce hologramu

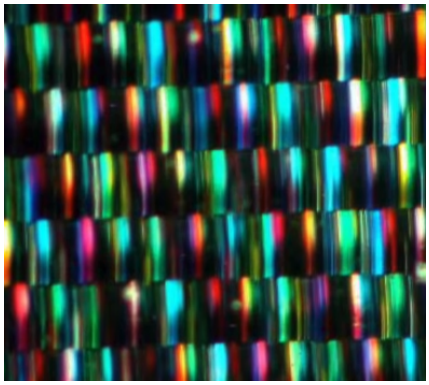


Illustration 77: zvětšený holografický záznam

dojde k rekonstrukci na jiné vlnové délce (výsledný obraz bude mít tedy jinou barvu).

Stimulovaná emise je způsob emise fotonů (např. světla), která na rozdíl od **spontánní emise** vytváří **koherentní vlny** (stejná frekvence a stejný fázový rozdíl v jakémkoli čase), používá se proto např. u **laserů**. Při spontánní emisi (žárovka, Slunce, ...) přechází atom (neboli jeho elektron) z vyššího (excitovaného) energetického stavu E_2 do nižšího E_1 a vyzařuje foton o frekvenci $f = E_2 - E_1$. Při stimulované emisi dopadá foton s frekvencí f na atom ve vyšším (excitovaném) energetickém stavu E_2, atom přejde do stavu E_1 a vyzáří další foton, přičemž původní foton není pohlcen. Nyní máme tedy dva fotony o stejné frekvenci a fázi, kterému můžeme dále stejným způsobem použít k "výrobě" dalších takových fotonů.

Lasery využívají stimulované emist tak, že do trubičky, která obsahuje speciální médium a která staví proti sobě dvě zrcadla, vypustí pár fotonů, které následně naráží do fotonů média a vytváří tak nové koherentní fotony. Fotony se odrážejí na zrcadlech sem a tam a opakují proces stimulované emise. V jednom zrcadle je malý otvor, kterým koherentní světlo vylétá ven.

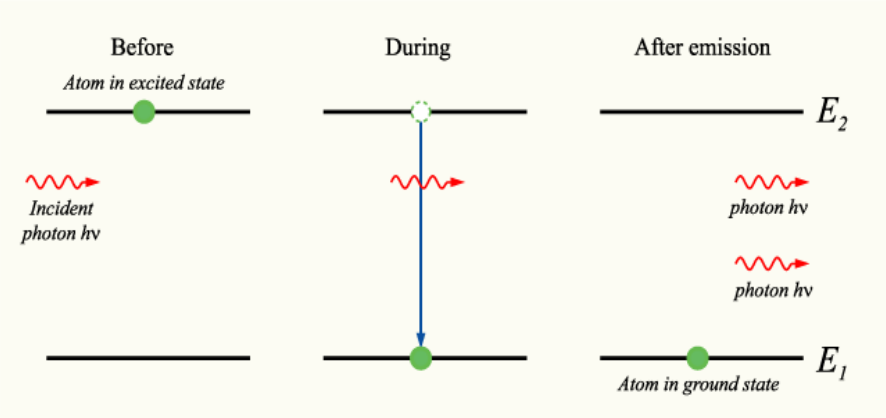


Illustration 78: stimulovaná emise

Metastabilní hladina je energetická hladina, na níž elektron setrvává relativně dlouhou dobu (až sekundy), než přejde na nižší hladinu. Prostředí trubičky v laseru musí obsahovat takové médium, jehož atomy setrvávají na metastabilní hladině, aby bylo zaručeno, že atom počká na této excitované hladině na další foton, který do něj narazí. Kdyby atomy nesetrvávaly na metastabilní hladině, přešli by hned na nižší energetickou hladinu a nárazy fotonů by už nezpůsobovaly stimulovanou emisi.

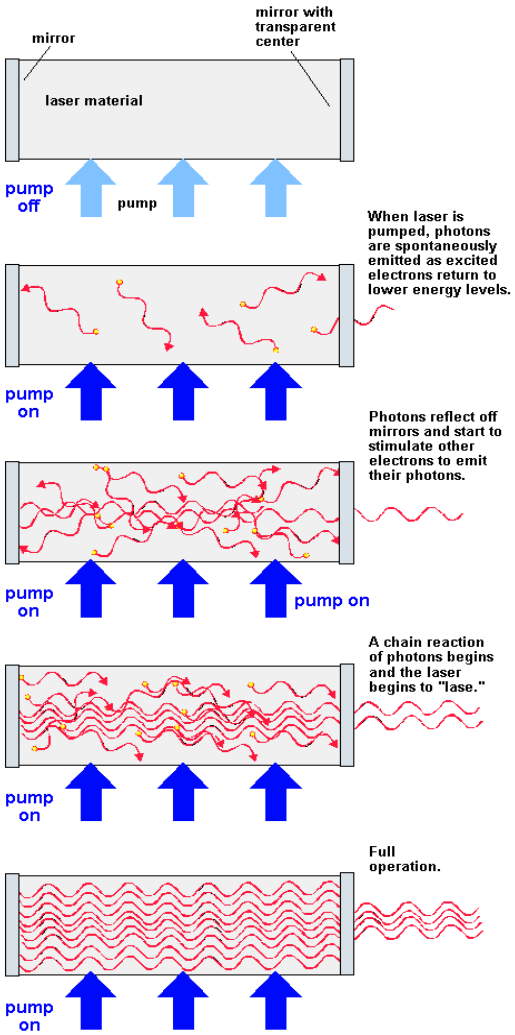


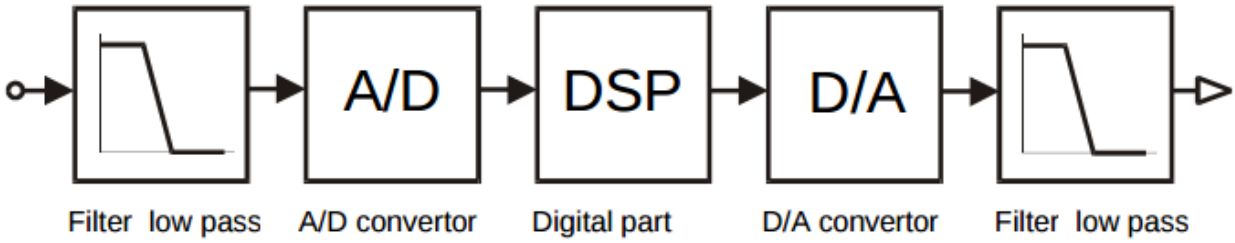
Illustration 79: princip laseru

40. Vztah zpracování signálu a multimédií (proč je zpracování zvukového a obrazového signálu pro multimédia důležité, typické operace při zpracování zvukového a obrazového signálu). MUL

Multimédia kombinují více kanálů, jako např. obraz a zvuk, přičemž tyto kanály lze vnímat jako signály, které chceme většinou syntetizovat a analyzovat, proto je pro multimédia důležitá oblast zpracování signálů. Typicky jde např. o následují operace:

- syntéza grafiky a zvuku
- reprodukce obrazu, zvuku a videa
- syntéza řeči, hudby
- apod.

Typický systém pro zpracování signálu v multimédiích vypadá takto:



První low pass filter se snaží zabránit aliasingu, tzn. snaží se odstranit vysoké frekvence, aby byl splněn **vzorkovací teorém**. Pro správnou funkci musí mít velmi ostrou charakteristiku, čehož je ale těžké dosáhnout. Účelem výstupního filtru je opět odstranit vysoké frekvence vzniklé převedením diskrétního signálu na spojitý (tzn. "zuby"). Na rozdíl od vstupního low pass filtru může být výstupní filtr poměrně jednoduchý díky charakteristikám frekvencí vznikajících v D/A převodníku, které jsou mnohem vyšší než *f*/2 a filtr proto nemusí být tak strmý.

Typicky se v multimédiích setkáváme s **LTI** (lineárními, časově invariantními) systémy. Tyto filtry dělíme na:

- **FIR** (finite impulse response, s konečnou impulzní odezvou) – Jsou vždy stabilní a obecně jednodušší na návrh než IIR filtry.
- **IIR** (infinite impulse response, s nekonečnou impulzní odezvou) – Obtížnější na návrh, problémy se stabilitou.

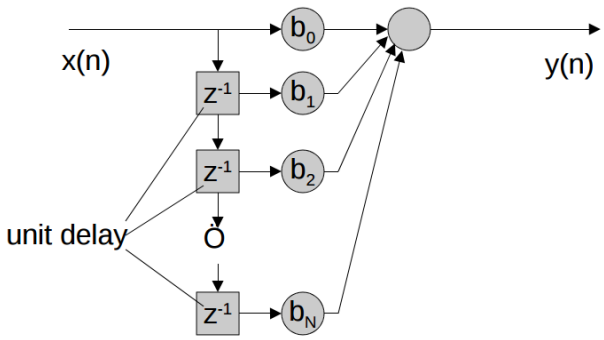
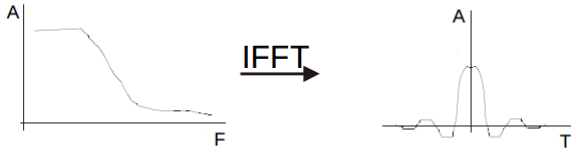


Illustration 80: schéma FIR filtru

Většinou máme danou impulzní charakteristiku a chceme z ní dostat koeficienty FIR filtru. Toho dosáhneme inverzní Fourierovou transformací:



Výsledek nám vždy vyjde symetrický, proto jej musíme posunout doprava. Výsledek je dále nekonečný, proto jej musíme rovněž ořezat, což ale způsobí změny ve frekvenční charakteristice. K ořezávání se používají následující okna:

- **rectangular** – Prostý obdelník, hodně kazí

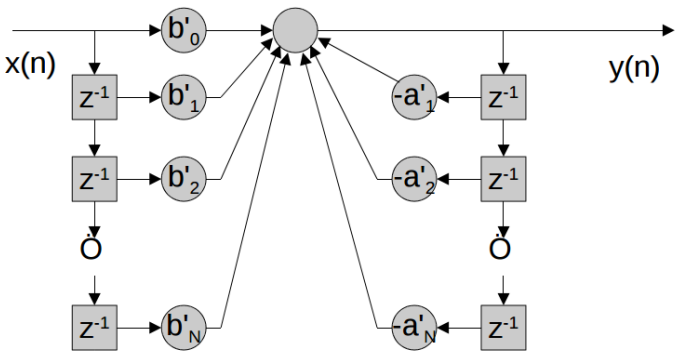


Illustration 81: schéma IIR filtru

charakteristiku (ostré ořezové okraje přidají vysoké frekvence).

- **Hammingovo** – Funkce podobá gaussově křivce, charakteristiku ovlivňuje mnohem míň.
- **Kaiserovo** – Podobné Hammingově oknu.

Mezi typické operace používané při zpracování signálů v multimédiích patří:

- **DFT** (časová složitost n^2), **FFT** (rychlá FFT, čas. složitost je $n \log n$), **DCT** (diskrétní kosinová transformace), **DWT** (diskrétní vlnková transformace) a podobné transformace – Používají se k analýze, kompresi a dekompresi (např. kvantování nebo zanedbávání určitých frekvencí).
- **konvoluce** – Slouží k aplikaci filtrů, vylepšování obrazu apod.
- **interpolace** – Např. při podvzorkovávání.

41. Komprese zvuku (základní postupy při kompresi zvuku, jak se liší od obecné komprese dat, vztah k lidskému sluchu, kompresní poměr). MUL

Při kompresi zvuku se snažíme minimalizovat datový tok při zachování kvality, kompresní algoritmy se snaží být rychlé, jelikož se (de)komprese provádí velmi často v reálném čase. Od obecné komprese se liší tím, že mohou využívat:

- vlastnosti zvuku – Může se využít komprese v **časové/frekvenční oblasti**, převod na MIDI události, převod na parametry hlasu ve vokodérech apod. Dále lze využít redundance ve stereo zvuku a nasadit tzv. **intensity stereo coding** – kóduje se součet obou kanálů spolu s poměrem pro levý a pravý kanál.
- vnímání zvuku lidmi - tzv. **psychoakustický model**. Bylo např. vypozorováno, že lidský sluch vnímá frekvence logaritmicky (týká se frekvence i hlasitosti, člověk vnímá logaritmicky se zvyšující frekvenci/hlasitost jako zvyšující se lineárně), nikoliv lineárně, vnímá hlasitost rozdílně v závislosti na intenzitě a slyší (v závislosti na věku) v rozmezí 20 Hz – 20 KHz. Je také možné provádět **maskování**:
  - **frekvenční** – Určité frekvence o definované intenzitě maskují okolní frekvence o nižší intenzitě => lze je vypustit.
  - **temporální** – Signál o dané frekvenci a intenzitě maskuje i po doznění okolní frekvence o nižší intenzitě => lte je vypustit. Temporální maskování využívá např. MP3.

Častým standardem pro ztrátovou kompresi zvuku založenou na psychoakustickém modelu (a videa) je **MPEG-1**. Využívá dříve vynalezených psychoakustických algoritmů **ASPEC** a **MUSICAM**. Dělí se na tři vrstvy:

- **Layer I** (mp1, m1a) – Nejjednodušší verze, používala se dříve kvůli malým HW nárokům. Bitrate je 384 kbit/s. Využívá 32 frekvenčních pásem, každé široké 625 Hz (lineární => neodpovídá lidskému sluchu), pro každé pásmo se podle intenzity určí kvantizační koeficient, aby byl vzniklý šum přijatelný. Výsledný signál se kóduje Huffmanovým kódováním.
- **Layer II** (mp2) – Vylepšení Layer I, avšak dekódování je stále nenáročné v porovnání s mp3. Bitrate je 192 kbit/s.
- **Layer III** (**mp3**) – Poměrně komplexní kodér/dekodér, nabízí audio ve velmi dobré kvalitě na bitratech 64 kbit/s (mono) nebo 128 kbit/s (stereo). Kromě rozdělení zvuku do pásem se ještě každé dělí do podpásem, filtrují se frekvence mimo slyšitelný rozsah, využívá maskování, neuniformní kvantování (adaptivní alokace bitů podle tzv. odstupu signálu od masky), Huffmanovo kódování atd. Kompresní poměr při 128 kbit/s je cca **11:1**.

42. Komprese obrazu (základní postupy při kompresi obrazu, jak se liší od obecné komprese dat, vztah k lidskému zraku a jeho vlastnostem, dosahovaný kompresní poměr). MUL

Kompresi obrazu můžeme provádět v **prostorové** nebo **frekvenční** oblasi a to buď **ztrátově** (většinou fotografie) nebo **bezztrátově** (např. pro screenshoty). Oproti kompresi obecných dat můžeme využít poznatků ohledně lidského vnímání (např. vyšší citlivost na jas než na barvu, na nižší než vyšší frekvence apod.). Mezi některé metody patří:

- **RLE** (run-length encoding, BMP) – Sekvenci znaků kóduje jako dvojici znak:počet. Tento způsob je bezztrátový a je vhodný především pro prostorovou oblast u syntetizované grafiky, v níž se vyskytují delší jednolité úseky (např. screenshoty plochy apod.).
- **Huffmanovo kódování** – Statisticky nejčastějším znakům se přiřazuje nejkratší kód, přičemž musí platit, že žádné kódové slovo není prefixem jiného. Využití je podobné jako u RLE.
- **Deflate** – kombinace Huffmanova kódování a LZ77 (slovníková metoda bezztrátové komprese, využívá posuvné okno). Využívá se v PNG.
- **LZW84** – Bezztrátový kompresní algoritmus, vylepšení LZ77. Je jednodušší než Deflate, nedosahuje tak dobrých výsledků.
- **prediktor** (PNG, JPEG-LS) – Prediktor předpovídá následující hodnoty z předchozích hodnot a kóduje se jenom chyba predikce. Prediktory dělíme na lineární a nelineární.
- **frekvenční transformace** (JPEG) – Obraz se převede do frekvenční oblasti a vyšší frekvence se kvantizují na méně bitů – využívá se toho, že je lidské oko méně citlivé na vyšší frekvence. Této metody využívá formát JPEG, přičemž existují různé, odlišné verze. Jedna z nich funguje takto:
  1. Obraz se převede z RGB do YCbCr (jas a dvě chromatické složky určující barvu).
  2. Složky Cb a Cr jsou podvzorkovány (lidský zrak je citlivější na jas, proto se tato změna viditelně téměř neprojeví).
  3. Obraz se rozdělí na bloky 8 x 8 pixelů (toto teoreticky není nutné dělat a některé metody skutečně pracují nad celým obrazem, avšak rozdělení na bloky může být výhodné z implementačních důvodů, dochází však k tvorbě "blokových artefaktů"). Nad každým blokem a barevnou složkou je spočítána DCT (diskrétní kosinová transformace).
  4. Jednotlivé hodnoty DCT se nakvantují, přičemž vyšší frekvence jsou kvantovány na méně úrovní, lidský zrak si toho všimne jen málo nebo vůbec. Kvantizace se děje na základě matice, která udává počet úrovní jednotlivých složek, avšak tato matice není součástí standardu a její tvar je ponechán na implementaci.
  5. Koeficienty DCT se serializují zig-zag průchodem bloku tak, že stejné hodnoty budou vedle sebe, a použijeme RLE a Huffmanovo kódování.

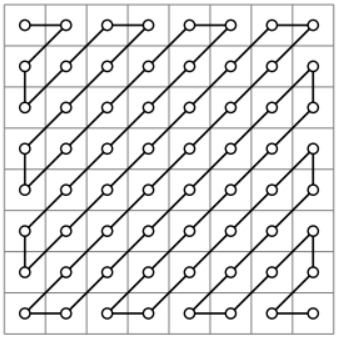


Illustration 82: zig-zag průchod DCT koeficientů

Podobné metody můžou namísto DCT používat jiné transformace, jako např. DFT, DWT (diskrétní vlnková transformace) apod. JPEG 2000 např. Nedělí obraz na bloky a používá DWT.

Kompresní poměry se u ztrátové komprese pohybují v rozmezí 1:50 – 1:100, u bezztrátové cca 1:15.

Illustration 83: DWT

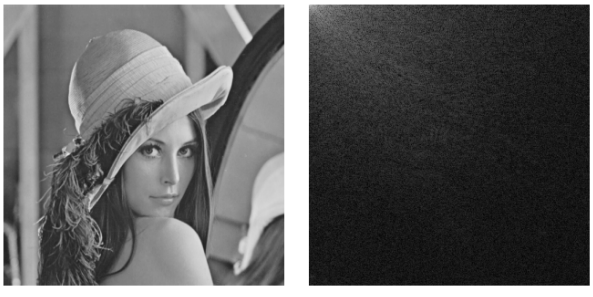


Illustration 84: DCT

43. Komprese videosekvencí (základní postupy při kompresi videa, jak se liší od komprese obrazu, a od obecné komprese dat, vlastnosti a dosahovaný, kompresní poměr). MUL

U komprese videa můžeme využít jak kompresi statických snímků, ze kterých se skládá, tak navíc i kompresi v časové oblasti, jelikož videa typicky obsahují **časovou redundanci** (snímky jdoucí po sobě jsou si podobné), snímky většinou kódujeme rozdílově. Typicky při kompresi videa rozlišujeme různé typy snímků, v MPEG-2 standardu jsou to např. (v jiných algoritmech jsou obdobné):

- **I-frame** (intraframe) – Je zakódován celý, nejčastěji pomocí nějaké komprese pro statické snímky, např. JPEG.
- **P-frame** (predicted frame) – Obraz se dá zrekonstruovat na základě předchozího I nebo P snímku, je tedy kódován rozdílově.
- **B-frame** (bidirectional frame) – Obraz se dá zrekonstruovat z předcházejícího (I nebo P) a následujícího (I nebo P) snímku.
- **D-frame** – Obsahuje informaci o DC složce, používal se pro rychlé náhledy, v moderních algoritmech se už nepoužívá.

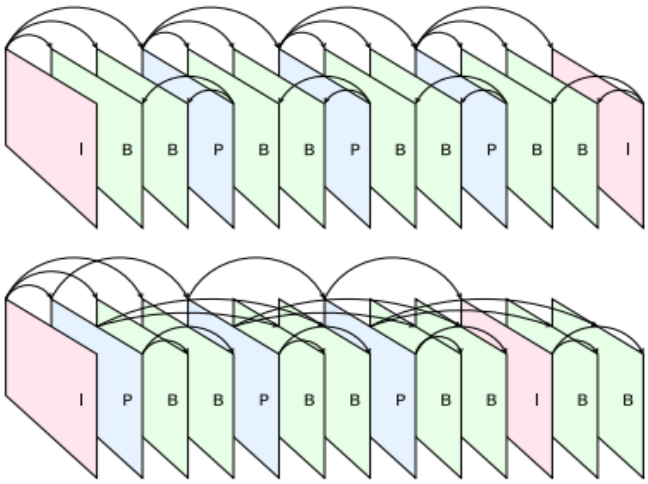


Illustration 85: typy snímků

Sekvence snímků začínající I snímkem až po následující I snímek se nazývá **GOF** (**group of frames**, většinou 12 – 18 snímků). Větší velikost GOF znamená větší kompresní poměr, ale menší kvalitu, vyšší nároky na paměť při přehrávání a náročnější seekování (přeskakování ve videu při přehrávání).

Při kompresi do MPEG formátu se snímky převádí do YC_BC_R a podvzorkovávají. Dále rozeznáváme pojmy **blok** (8 x 8 pixelů, pro DCT) a **makroblok** (2 x 2 bloky, při použitém podvzorkování dostáváme 4 Y bloky, 1 C_B a 1 C_R blok).

Rozdílové kódování snímků spočívá v **odhadu pohybu** (**motion estimation**) makrobloků kódují se pak jenom tzv. **pohybové vektory** (**motion vectors**), které udávají, kam se daný makroblok (nebo i jeho část, tzn. blok) pohne. Toto se nazývá **kompenzace pohybu** (**motion compensation**). Přesnost pohybového vektoru je půl pixelu. Při odhadu pohybu bloků se většinou zanedbávají věci jako rotace, osvětlení apod., jinak by algoritmy pro kódování byly příliš časově náročné. Dále se používá globální kompentace pohybu, která umožňuje zaznamenat afinní transformaci celého snímku.

Pro snížení datového toku videa se také někdy používá tzv. prokládání (**interlacing**, opakem je **progressive**), tzn. každý snímek videa je ve skutečnosti jenom půlsnímek skládající se z lichých, resp. sudých řádků, tzn. přenáší se pouze polovina obrazové informace.

V jednodušších animacích, jaké nabízí např. formát GIF, lze použít čistě pixelově rozdílovou temporální kompresi, tzn. pro další snímek zaznamenáme jenom pixely, které skutečně změnily barvu – těchto pixelů bude relativně málo (to je ale dáno především omezením počtu barev palety ve formátu GIF a pro kvalitní video by tudíž tato technika dobře nefungovala).

Kodér dále kóduje výstupní data běžnými algoritmy, např. RLE.

Kompresní poměr MPEG formátu dosahuje 1:20 až 1:200.


Illustration 86: pohybové vektory při odhadu pohybu ve videu

44. Informace a entropie, Shannova věta o kódování. PDS, ZRE

Entropie je veličina měřící míru nepředvídatelnosti, její jednotkou je **Shannon** (Sh). Více nepředvídatelná zpráva nese více informace. Máme-li událost *i*, jejíž pravděpodobnost je *p*, pak množství informace *I* získané pozorováním události *i* je

$$I(p) = \log_2(1/p) = -\log_2(p)$$

Tento vztah zachovává vlastnosti, které o informaci předpokládáme (pravděpodobnost 1 dává $I = 0$, $I(p_1\, p_2) = I(p_1) + I(p_2)$ apod., pro $p = 0$ se bere limita). Entropie náhodné veličiny X nabývající hodnot $x_1, \dots x_n$ s pravděpodobnostmi $p_1, \dots p_n$ je definována jako:

$$H(X) = \sum_{i=0}^{n} p_i I(p_i) = \sum_{i=0}^{n} p_i \log_2\left(\frac{1}{p_i}\right)$$ (nabývá hodnot 0 – log *n*)

U logaritmu může být použit i jiný základ, avšak často používáme 2 – potom nazýváme jednotku entropie také **bit**. Např. pro hod mincí je entropi rovna 1 (= $\log_2 2$), pro dva hody mincí 2 atd. Entropie je maximální, pokud jsou všechny hodnoty náhodné veličiny stejně pravděpodobné. Entropii používáme např. při komprimaci a jiném kódování – běžný text má nízkou entropii, jejím zvýšením snížíme počet bitů, nebo naopak můžeme entropii snižovat, zvyšovat redundanci a bezpečnost.

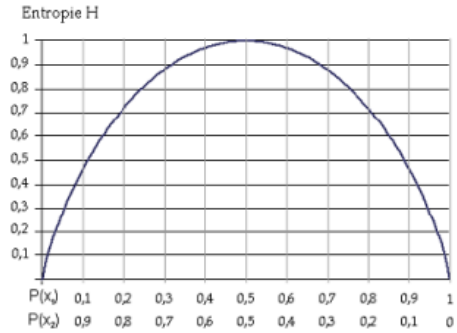


Illustration 87: entropie náhodné veličiny se dvěma jevy v závisloti na jejich pravděpodobnostech

Podmíněná entropie říká, jaká je entropir jedné náhodné veličiny za předpokladu, že známe hodnotu jiné náhodné veličiny.

Kódování je přiřazování zpráv jedné abecedy zprávám jiné abecedy, tedy matematické zobrazení. Mezi kódování zvyšující entropii patří např. **Huffmanův kód**. Jde o prefixový (žádné kódové slovo není prefixem jiného) kód umožňující bezztrátovou kompresi. Znakům přiřazuje kódy proměnné délky na základě jejich pravděpodobnosti. Algoritmus pro vytvoření Huffmanova stromu je:

1. Dvěma nejméně pravděpodobným znakům přiřadíme hodnotu 0 a 1.
2. Dané dva prvky sloučíme a sečteme jejich pravděpodobnost.
3. Opakujeme, dokud je co slučovat.

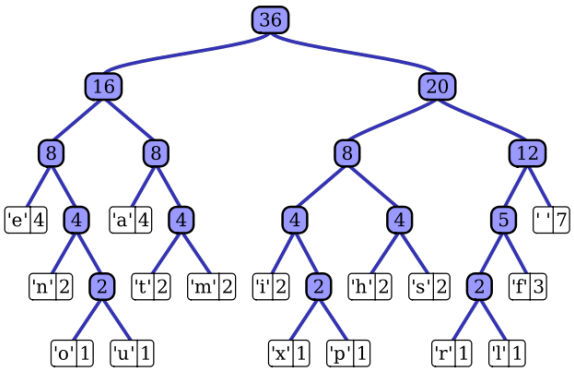


Illustration 88: Huffmanův kód generovaný z věty "this is an example of huffman tree". Cesta ve stromu vlevo znamená zápis bitu 0, vpravo pak 1. Např. znak "a" je kódován jako 010.

U daného kódování můžeme zjišťovat:

- střední délku kódového slova *L*
- účinnost: $u = H(X) / L$
- redundanci: $r = 1 - u$

Nechť dvě strany komunikují pomocí **přenosového kanálu**. Jesliže zdroj zasílá *n* zpráv za sekundu a entropi je *H* bitů na zprávu, pak ryhlost informace je

$R = nH$ (v bitech informace za sekundu)

Dále mějme **kapacitu kanálu** *C* takovou, že pokud je $R \leq C$, pak lze přenos uskutečnit bez chyb, i za přítomnosti šumu, použitím správných kódovacích technik. Je-li $R > C$, nelze se vyhnout chybám. *C* je největší množství informace, které kanál dokáže přenést za sekundu.

Shannonova věta o kódování říká, že má-li kanál kapacitu $C > 0$, pak můžeme ryhlost produkce informace libovolně přiblížit *C* a přitom učinit pravděpodobnost chyby libovolně malou. Snižování chyby lze dosáhnout používáním specifického kódování, které je ale pro menší chybu čím dál složitější (musí pracovat s delšími bloky dat, což implikuje vyšší delay a výpočetní náročnost). Shannonova věta také říká, jak vypočítat kapacitu kanálu.

45. Bezpečnostní kódy: lineární, Hammingovy, cyklické, konvoluční. Detekce a oprava chyb. PDS

Bezpečnostní kódy umožňují kódovat data tak, že je možné do určité míry detekovat, popř. i opravovat chyby vzniklé např. při přenosu dat po síti. Bezpečnostní kódy dělíme na:

- **blokové** – Zabezpečovaná data jsou rozdělena na bloky fixní velikosti, přičemž bloky jsou zabezpečovány nezávisle na sobě. Blokové kódy se označují dvojicí (k, n), kde n je počet kódovaných bitů a k délka kódovaného slova (redundance je tedy $k - n$). Mezi blokové kódy patří např.:
  - **parita** – Kód ($k + 1$, k), doplňuje kódové slovo jedním bitem tak, aby celkový počet jedniček v něm byl sudý (popř. lichý) – např. 1011 zakóduje jako 10111. Parita dokáže detekovat lichý počet chyb, nedokáže slovo opravit.
  - **složená parita** – Zarovná vstupní data do 2D matice a pro každý řádek a sloupec spočítá paritu. Dokáže opravit jednu chybu.
  - **Hammingův kód** (7,4) – Perfektní, lineární kód (viz níže), kóduje 4 bity informace 7 bity. Detekuje 1 nebo 2 chyby a opravuje 1 chybu. U obecného Hammingova kódu jsou redundantní bity na pozicích mocnin dvou, zde jsou to tedy bity číslo 1, 2 a 4. Např. data 0001 jsou kódována jako 1101001 (podtrženy jsou redundantní bity). Kontrolní bit na pozici 2^l kontroluje všechny bity na pozicích p, kde binárně zapsaná pozice p má 1 na pozici l. Kontrolní bit je sudá parita všech bitů, které kontroluje. Matice G a H (viz níže) pro tento kód jsou:

$$G = \begin{matrix}1&1&0&1\\1&0&1&1\\1&0&0&0\\0&1&1&1\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{matrix} \qquad H = \begin{matrix}1&0&1&0&1&0&1\\0&1&1&0&0&1&1\\0&0&0&1&1&1&1\end{matrix}$$

Pozice	Pozice binárně	Datový bit	Paritní bit	Kontroluje bity
1	001		✓	1,3,5,7
2	010		✓	2,3,6,7
3	011	✓		
4	100		✓	4,5,6,7
5	101	✓		
6	110	✓		
7	111	✓		

Illustration 89: bity Hammingova kódu (7,4)

např. zakódujte 1101: $G\ (1101)^T = (1010101)$
zkontrolujte 1010111: $H\ (1010111)^T = (011)$ = syndrom chyby, 011 odpovídá šestému sloupci $H \Rightarrow$ chyba je na šesté pozici.

- **proudové** – Zabezpečuje se kontinuálně bit po bitu, přičemž kódování daného bitu závisí na hodnotách několika předchozích bitů.

Správa chyb nám umožňuje vypořádat se s chybami, které při přenosu nastávají kvůli šumu, poruchám, ztrátě paketů apod. Chyb dělíme na:

- **bitové** – Chyby ovlivňující bity zprávy (např. inverze bitu), dějí se na L1, řešíme je na L2, obvykle pomocí redundance. Souhrnně tyto chyby nazýváme **korupcí bitů**, avšak nejčastější je **bitová inverze**. Tuto chybovost měříme v tzv. **BER** (**bit error rate**) = počet chybných bitů / celkový počet bitů.
- **paketové** – Chyby týkající se přenosu zprávy, patří sem:
  - **ztráta paketu** – Kvůli zahlcení linky, neopravitelné bitové chybě apod.
  - **ztráta fragmentovaných dat** – Dojde-li ke ztrátě fragmentu (části paketu), ztratí se celý paket.
  - **duplikace paketu** – Pro korektně přijatý paket se ztratí potvrzení.
  - **vložení paketu** – Zpožděný paket z už ukončené komunikace dorazí do nově započaté.
  - **změna pořadí paketů** – Pakety jdou různými cestami.

  Měříme v tzv. **PER** (**packet error rate**) = počet chybných paketů / celkový počet paketů. Řešíme na L4 pomocí:
  - **sekvenčních čísel** – Lze detekovat ztrátu a prohození pořadí.
  - **znovuzasílání** ztracených paketů, je potřeba ztrátu detekovat pomocí sekv. čísel a potvrzování. Existují různé strategie:
    - **stop and wait** – Čeká na potvrzení každého paketu, pak až pošle další – nevýhodné, zdržuje.
    - **go back N** – Příjemce potvrzuje jen poslední korektně přijatý paket z (tzv. klouzavého, **sliding window**) okna dané velikosti.
    - selektivní znovuzasílání – Chytřejší způsob než go back N, složitější.
  - **timeoutu** – Čekání na packet po určitou dobu, problém je, jak ji zvolit (např. z round trip time).

- **negativního potvrzování** – Při detekci mezery v sekv. číslech se zašle žádost o znovuzaslání paketu. Problémy: další zahlcování sítě, může se ztratit i NACK (negativní potvrzení) paket.
- **3way handshaku** – Používá se k ustavení spojení v TCP a inicializaci sekv. Čísel. Sekvenční čísla jsou inicializována náhodně, aby se zabránilo chybě vložení paketu. Proces probíhá takto:
  1. *A* si vygeneruje náhodné sekv. číslo *x* a pošle SYN paket s tímto číslem
  2. *B* obdrží, uloží si sekv. číslo pro *A* = *x*, vygeneruje si vlastní náhodné sekv. číslo *y*, pošle SYN paket s *y* a ACK paket s číslem *x* + 1
  3. *A* obdrží, zapíše si sekv. číslo pro *B* = *y* a pošle ACK s číslem *y* + 1

Hammingova vzdálenost, značená d_{min}, je minimální počet bitů, které musíme změnit v jednom kódovém slově, abychom jej transformovali na jiné. Např. Hammingova vzdálenost slov 101101 a 011101 je 2. Hammingova vzdálenost kódu je minimum ze vzdáleností mezi každými dvěma kódovými slovy. Počet detekovatelných chyb je roven $d_{min} - 1$. Např. kódujeme-li 0 jako 000 a 1 jako 111, máme $d_{min} = 3$ a jsme schopni detekovat 2 chyby, např. 011 je nesprávné slovo. Platí:

- Pro detekci *E* chyb musí platit: $d_{min} = E + 1$.
- Pro opravu *E* chyb musí platit: $d_{min} = 2 * E + 1$.

Perfektní (**optimální**) **kód** je kód, který k opravení 1 chyby slova délky *n* využívá minimální možný počet redundantních bitů *c*. Ten je roven:

$c = \log_2(n + 1)$ (např. pro $n = 7$ potřebujeme 3 redundantní bity)

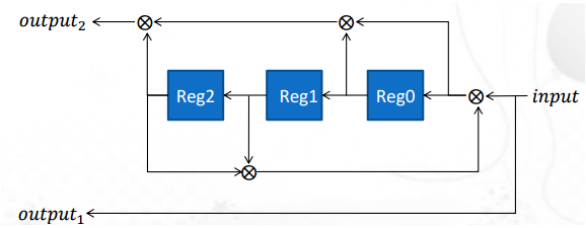


Illustration 90: příklad rekurentního konvol. kodéru

Řetězec k bitů lze chápat jako binární polynom (polynom, kde koeficienty jsou buď 0 nebo 1) stupně *k* – 1, nebo také vektor koeficientů tohoto polynomu.

Lineární kód je korekční kód, jehož slova tvoří lineární (vektorový) prostor – platí tedy, že lineární kombinace libovolných kódových slov je zase kódové slovo. Lineární kód označujeme (*n*, *k*), kde *n* je délka kód. slova a $k < n$ je dimenze vektorového prostoru prostoru (pokud by bylo $n = k$, kód by neměl žádou redundanci a nebyl by to bezpečnostní kód). Počet kódových slov je 2^k. Matici $n \times k$ bázových (lineárně nezávislých) vektorů nazýváme **generující maticí** *G*. Lze také odvodit kontrolní matici *H*, kterou lze zjišťovat a opravovat chyby (vynásobením vektoru s touto maticí dostaneme tzv. syndrom chyby, který udává pozici chyby). Platí $G\,H^T = 0$.

Symetrický kód je kód, který má ve slově *k* informačních a $n - k$ zabezpečujících bitů na pevně daných pozicích.

Cycklický kód je lineární kód, u nějž cyklický posun jakéhokoliv slova dává zase kódové slovo (např. {100,010,001}).

CRC (cyclic redundancy check) je široce používaný cyklický kód. Podle generujícího polynomu existují různé varianty CRC, jako např. CRC-10, CRC-32 apod. CRC dokáže najít 1 bitové chyby, skoro všechny 2 bitové a některé delší shluky chyb. K zabezpečení se používá dělení polynomů.

Př.: Zabezpečte zprávu 1101011011 pomocí CRC polynomem $x^4 + x + 1$:
$x^4 + x + 1$ – binární koeficienty jsou 10011, stupeň je 4, přidáme 4 nuly nakonec: 11010110110000 : 10011 = 1100001, zbytek 001110, zbytek přidáme na konec: 11010110111110, ověření by probíhalo zase vydělením a zjištěním, zda se zbytek rovná nule

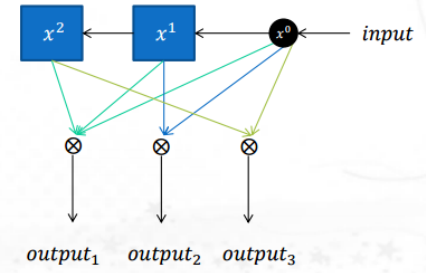


Illustration 91: příklad nerekurentního konv. kodéru s generujícími polynomy: g1 = (1,1,1), g2 = (0,1,1), g3 = (1,0,1)

Shannonova věta o kódování říká, že pro každý přenosový kanál s kapacitou *C* a přenosovou rychlostí $R < C$ platí, že pravděpodobnost chyby můžeme libovolně minimalizovat (použitím vhodného kódování, které je však pro menší chybu čím dál výpočetně komplexnější).

Konvoluční kód je proudový korekční kód využívající binární polynomy. Implementují se pomocí posuvných registrů. Dají se popsat konečnými automaty. Dělíme je na:

- **rekurentní** – Aktuální výstup závisí na hodnotě předchozího výstupu.
- **nerekurentní** - Aktuální výstup nezávisí na hodnotě předchozího výstupu.

46. Základní architektury přepínačů, algoritmy pro plánování, řešení blokování, vícestupňové přepínací sítě. PDS

Přepínač přeposílá data na L2 (MAC adresy), na rozdíl od směrovačů pracujících na L3 (IP adresy). Přepínač na rozdíl od hubu nepřeposílá data na všechny výstupy, ale přeposílá je podle adres (je složitější). Přepínač se skládá z **line cards** (karty starající se o rozhraní) a **switching fabric** – přepínacího obvodu.

Základními architekturami přepínačů jsou:

- **se sdílenou sběrnicí** (**shared backplane**) – Jako přepínací logika je zde klasická sdílená sběrnice, na kterou může v jednu chvíli vysílat jenom jedna jednotka, přijímat ale může neomezený počet jednotek, tzn. výhodou je nativní multicast a broadcast. Tato architektura není vhodná, pokud požadujeme vysokou propustnost.
- **s přepínanou deskou** (**switched backplane**) – Je zde více point-to-point spojení, takže přenosy mohou probíhat paralelně, ale nelze vysílat na jeden výstup z více vstupů, ani na více výstupů u jednoho vstupu. Musí existovat **plánovač** (**scheduler**), který ustanovuje propojení mezi vstupy a výstupy.
- **se sdílenou pamětí** – Speciální případ switched backplane, využívá se sdílené paměti, která je rozdělena na fronty pro každý výstup (fronty můžou být oblasti pevné nebo proměnné velikosti). Je zde problém s přístupovou dobou paměti.
- **křížový přepínač** – Má N^2 spojů, dokáže vysílat z jednoho vstupu na více výstupů (nativní broadcast a multicast), ale ne z více vstupů na jeden výstup. Výhodou jsou paralelní přenosy. Používá se do max. 64 portů.

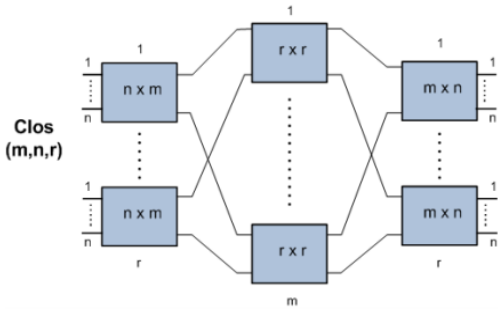


Illustration 92: Clos

- vícestupňové přepínání – Má více úrovní, na kterých se přepíná.
  - **Clos** (m,n,r) – Třístupňová přepínací síť, využívá křížové přepínače. Existuje *m* různých cest mezi daným vstupem a výstupem. Platí Closův teorém: pokud je $m \geq 2n - 1$, pak lze vždy bez přeskládání přidat propojení mezi nevyužitým vstupem a výstupem (síť je tzv. **neblokující**). Má menší počet propojení než křížový přepínač. Používá se do max. 256 portů.
  - **Beneš** – Modifikovaní síť Clos(2,2,1). Konstrukce sítě je definována rekurzovně.

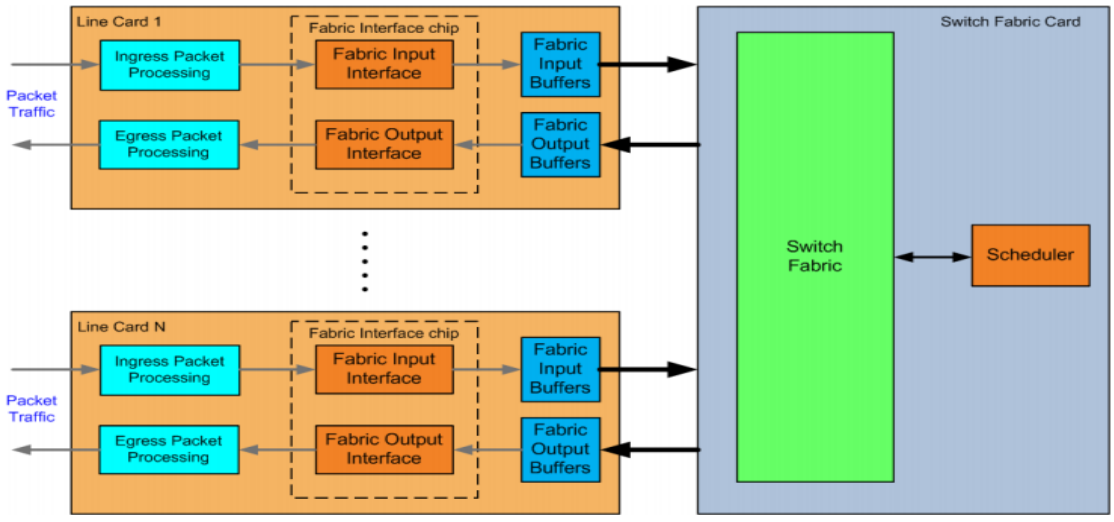


Illustration 93: obecná architektura přepínače

  - **Torus** – Decentralizované přímé přepínání, každý uzel slouží jako vstupní i výstupní port, uzly jsou uspořádány v *n*-dimenzionální síti. Výhodou je existence velkého množství cest mezi uzly díky dimenzionalitě. Hledání cest může probíhat náhodně, nebo deterministickými alg. (např. postupným posouváním se po jednotlivých dimenzích). Používá se do max. 512 portů.

Plánovač u switched backplane provádí **plánování**, což je problém nalezení párování (matching) mezi vstupními a výstupními porty tak, že každý vstup a výstup má maximálně jedno propojení. Párování dělíme na:

- **největší** (**maximum**) – Obsahuje největší možný počet hran. Je optimální a má největší propustnost, ale je obtížné ho nalézt, může způsobovat vyhladovění.
- **maximální** (**maximal**) – Nelze přidat další propojení, aniž bychom zvýšili stupeň některého uzlu na 2 –

lokální maximum. V praxi hledáme tato párování.

Významné algoritmy pro hledání párování jsou:

- **přidělování lístků** – Vstupní port pošle výstupnímu žádost o propojení, ten mu pošle zpět lístek s pořadovým číslem (v rámci všech žádostí, které dostal). Po přidělení lístků se ustanoví propojení a probíhá přenos. Žádosti se přenáší po speciální sběrnici. Je zde problém s **blokováním na začátku fronty** (**HOL**, head of line blocking) – to nastává, když paket na vstupním portu čeká na výstupní port a blokuje pakety ve frontě za ním, které musí taky čekat, I když je momentálně jejich cílový port dostupný. Toto se dá řešit tzv. **virtuálními frontami** – každá fronta na vstupu se rozdělí na více font, pro každý výstup bude jedna.
- **PIM** (parallel iterative matching) – Podobné jako přid. lístků, používá virtuální fronty a pracuje s náhodným výběrem – vstup pošle žádost na všechny požadované výstupy, každý výstup náhodně povolí jednu z žádostí, v případě více propojení se náhodně vybírá jedno. Takto se provede několik iterací, největší párování se může nalézt v nejlepším případě v 1 iteraci, nejhůře v N iteracích, průměrně v log N.
- **iSLIP** – Při soupeření o porty využívá rotující ukazatele. Na začátku si každý vstupní port inicializuje svůj ukazatel I_i na výstupní port, každý výstupní si inicializuje ukazatel na vstupní port O_j. Vstupní porty potom zašlou žádosti výstupním. V případě více žádostí vybere vstupní port tu s číslem portu $\geq O_j$ (pokud je jich víc, vybere se nejmenší hodnota). Pokud vstupní port obdrží více povolení, vybere to s číslem porti $\geq I_i$ (pokud je jich více, vybere se nejmenší hodnota). Potom se inkrementují (s operací modulo) všechny ukazatele, jejichž žádosti byly uspokojeny.

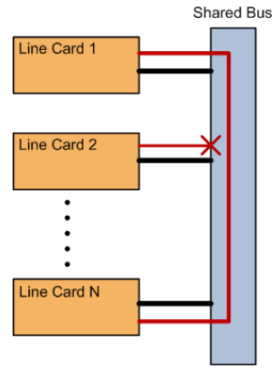


Illustration 94: sdílená sběrnice

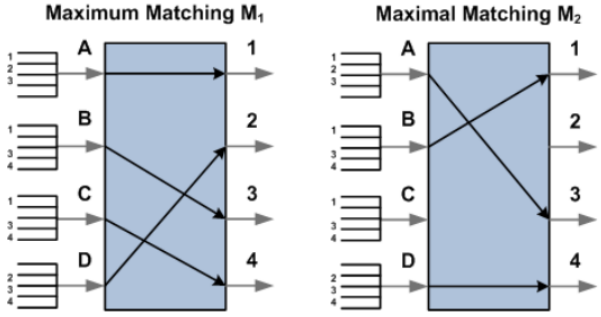


Illustration 95: maximální vs největší plánování při požadavcích na propojení: A1, A2, A3, B1, B3, B4, C1, C3, C4, D2, D3, D4

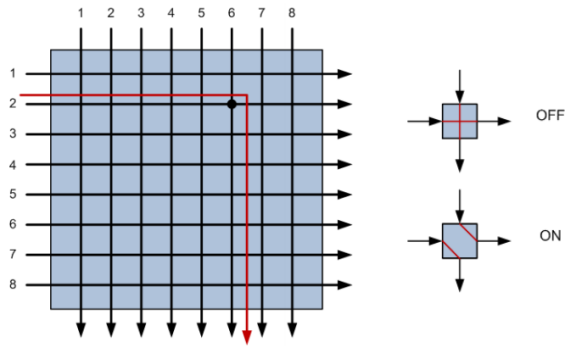


Illustration 96: křížový přepínač

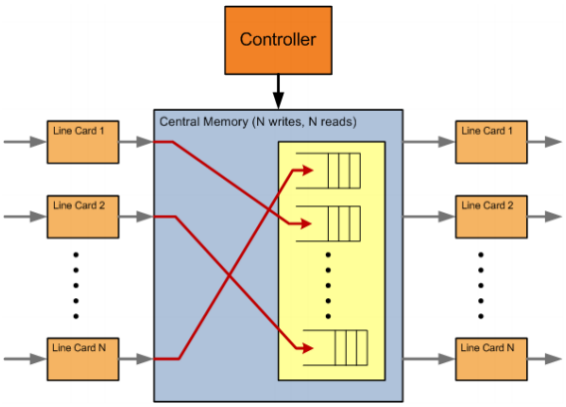


Illustration 97: sdílená paměť

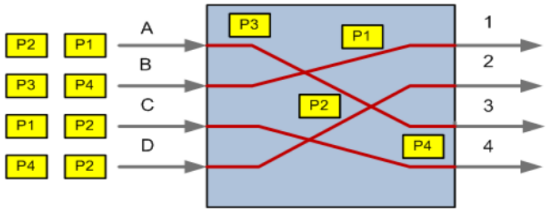


Illustration 98: přepínaná deskou

47. Základní funkce směrovače, zpracování paketů ve směrovači, typy architektur. PDS

Směrovače dělíme na

- **páteřní** (**core**) – Obvykle je používají ISP, propojují tisíce menších sítí, musí být extrémně rychlé a spolehlivé.
- **podnikové** (**enterprise**) – Propojují koncové systémy v podnikových sítích, musí mít hodně portů, efektivní broadcast a multicast a podporovat bezpečnostní nastavení.
- **přístupové** (**access**) – Propojují zákazníka s ISP, musí mít velké přenosové pásmo a podporu přístupových protokolů (např. VPN).

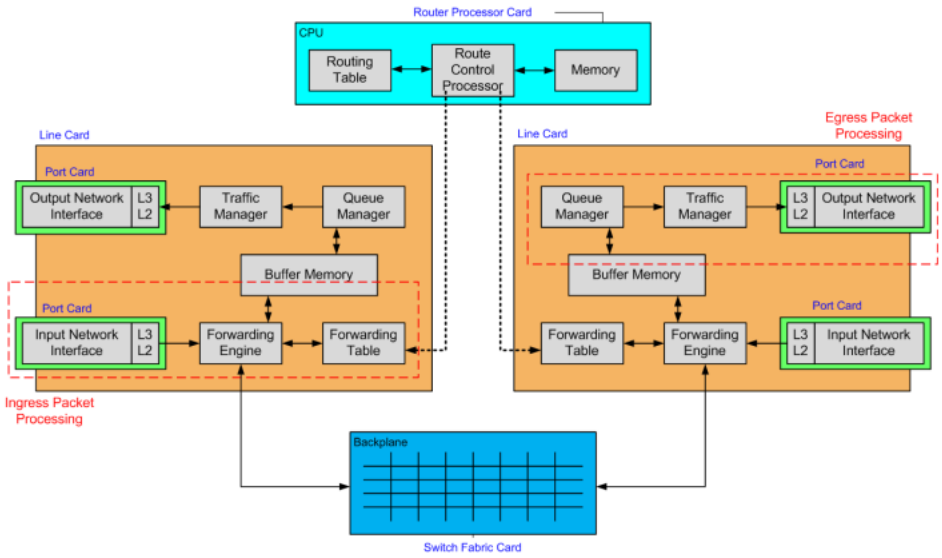


Illustration 99: části směrovače

Funkčními částmi směrovače jsou

- **síťové rozhraní** (**network interface**) – Stará se o funkci rozhraní, převádí mezi L2 a L3.
- **přepínací module** (**FE**, **forwarding engine**) – Rozhoduje, na které rozhraní se paket přepne pomocí přepínací tabulky (**forwarding table**, neplést se směrovací tabulkou). Provádí klasifikaci paketů pro QoS.
- **správce front** (**queue manager**) – Fronta, popř. fronty paketů pro rozhraní.
- **správce provozu** (**traffic manager**) – Stará se o provoz, např. rozhoduje, které pakety se zahodí apod.
- **propojovací deska** (**backplane**) – Sběrnice, přes kterou se posílají pakety mezi rozhraními.
- **CPU** (**route control processor**) – Provádí zpracování paketů, generuje chybová hlášení ICMP, obsluhuje směrovací tabulky apod.

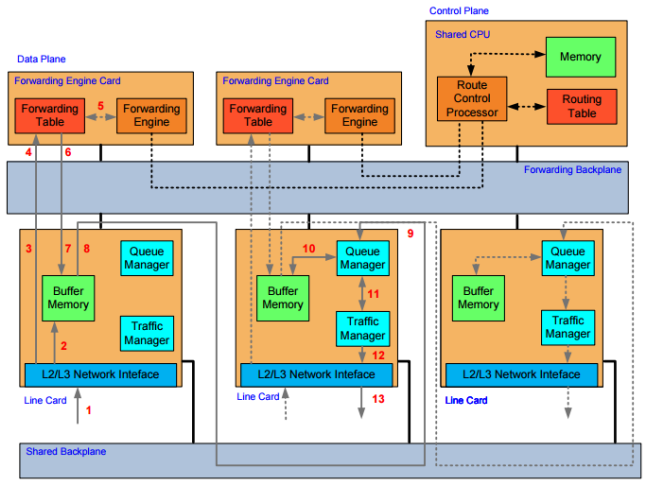


Illustration 100: architektura s nezávislými moduly FE

Zpracování paketů probíhá ve dvou částech, na něž se směrovač dělí:

- **control plane** – CPU, jeho paměť a směrovací tabulka
- **data plane** – ostatní (network interface, FE, buffer memory, forwarding table, ...)

Když přijde paket na vstup (network interface), předá se forwarding enginu, ten jej částečně analyzuje (zjistí jeho adresu), vytvoří tzv. **kontext paketu** a paket samotný uloží do buffer memory, kde bude dále čekat. Kontext paketu je datová struktura zastupující samotný paket během jeho zpracování. Forwarding engine se potom podívá do přepínací tabulky – ta buď obsahuje záznam o tom, kam paket přeposlat nebo ne. Pokud ne, pošle se kontext paketu do CPU, které ze směrovací tabulky zjistí, kam paket přeposlat a tato informace se zapíše do forwarding table. Paket se potom vyjme z buffer memory a pošel se po backplane na patřičné výstupní rozhraní, kde se zařadí do fronty, aby byl odeslán. Paket může jít tedy dvěma cestami:

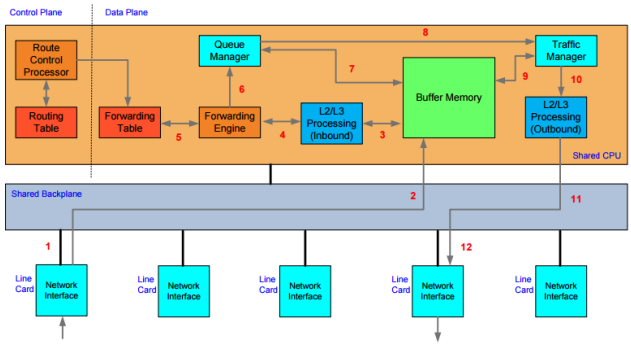


Illustration 101: architektura se sdíleným CPU

- **fast path** – Nejde přes procesor, je určena pro časově kritické operace, tzn. operace, které se provádějí nejčastěji (např. Zpracování hlavičky IP, klasifikace paketu, ...).
- **slow path** – Jde přes procesor, nekritické operace.

Architektura přepínače říká, jak se mapují funkční bloky na fyzické hardwarové části.

- **se sdíleným procesorem** – Jednoduchá, ale ne příliš výkonná architektura. Každý paket je zpracováván pomocí sdíleného CPU, tzn. pakety nejsou zpracovávány paralelně. Paket jde při příchodu na rozhraní (Line Card) přes backplane (sběrnici) do buffer memory, tam forwarding engine zjistí z přepínací tabulky, kam jej poslat dál, a přepošle se zase přes backplane na výstup (Line Card). Každý paket jde tedy po backplane minimálně dvakrát.
- **se sdíleným procesorem a cache** – To samé, jako předchozí architektura, ale na každé Line Card je navíc cache, takže pokud přijde paket, pro který existuje v cahce záznam, nemusí jít na zpracování do procesoru, ale může být rovnou přeposlán.
- **s nezávislými moduly FE** (forwarding engine) – Tato architektura má navíc samostatné karty s forwarding enginy – ty jsou mimo Line Cards a je jich typicky méně než Line Cards. Jsou zde dvě sběrnice – jedna pro přeposílání paketů mezi rozhraními a jedna pro komunikaci o tom, kam co přeposlat. Každý forwarding engine má u sebe přepínací tabulku, která musí být procesorem updatována. Když přijde paket, je velká šance, že forwarding engine podle přepínací tabulky bude shopný paket přeposlat a nebude se tak muset využít CPU.
- **distribuovaná architektura** – Nic kromě procesoru není sdílené, každá Line Card má vlastní forwarding engine, proto můžou být pakety zpracovávány paralelně. Přes procesor jdou jenom pakety, které jsou součástí tzv. **slow path** (málo časté pakety vyžadující speciální zpracování, např. fragmentace, ICMP, ARP apod.).
- **modulární architektura** – Oproti distribuované architektuře zde může být např. Více procesorů. Celá architektura je dělaná tak, aby se dala neustále rozšiřovat přidáváním nových modulů.

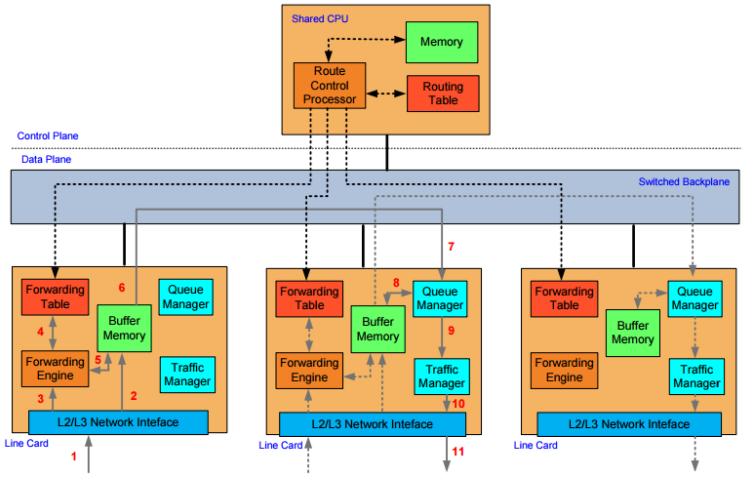


Illustration 102: distribuovaná architektura

48. Formální metody v počítačových sítích. PDS

Formání metody jsou matematické metody specifikace, návrhu, modelování a verifikace systémů. Pomocí formálních metod můžeme v oblasti sítí ověřovat protokoly, bezpečnost, efektivitu směrovacích algoritmů apod. Používá se např. formální logika, formální jazyky, teorie automatů apod.

Protokolové inženýrství je postup návrhu protokolů pro počítačové sítě. Začíná specifikací služeb, kterým má protokol sloužit, pokračuje specifikací samotného protokolu a končí jeho implementací. Ve všech těchto krocích můžeme využít formální metody, např. pro:

- **syntézu** protokolu ze specifikace služeb, dá se provádět dvěma způsoby:
  - **analyticky** – Navrhneme protokol, poté ho validujeme/verifikujeme, podle nedostatků upravíme protokol a opakujeme (tzn. iterativní přístup).
  - **syntetický** – Částečně specifikovaný protokol se dospecifikovává podle požadavků na korektnost a validitu. Výsledek není třeba validovat/verifikovat, měl by být validní a korektní z definice.
- **validaci** (ověření správnosti z hlediska nabíených služeb) a **verifikaci** (ověření správnosti vůči specifikaci) protokolu
- implementaci z formální specifikace
- simulaci (hledáme vhodný model, můžeme použít např. teorii front)
- testování (ověřování korektnosti pomocí testů, netýká se výkonnosti, robustnosti apod.) - Testy dělíme na **statické** (kontroluje rozsahy parametrů, existenci služeb na nižší vrstvě apod.) a **dynamické** (týká se komunikace samotné, včetně časování apod.). Existují standardní testy.
- konverzi protokolů (překlad jednoho protokolu na jiný, pro různé sítě, vrstvy apod.)

Formální prostředky dělíme podle orientace na:

- **orientované na model**
  - **stavové** – konečné automaty, Petriho sítě, ...
  - **přechodové** – gramatiky, stopy, ...
- **orientované na vlastnosti**
  - **algebraické** – algebry procesů, ...
  - **modální** – temporální logika (výroková logika + temp. operátory), ...
  - **axiomatické** – teorie množin, ptedikátová logika, ...

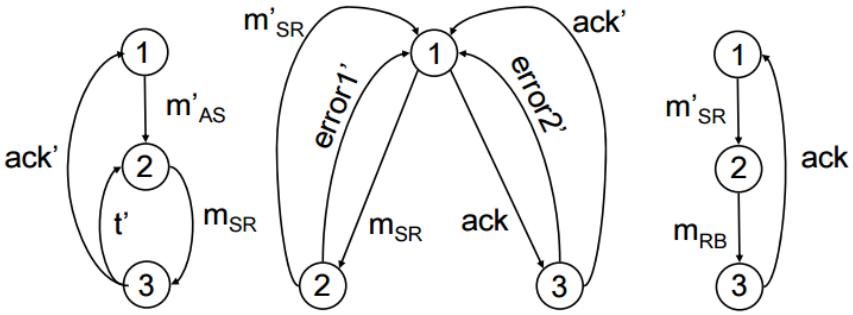


Illustration 103: příklad konečného automatu modelujícího potvrzování zprávy: první automat (vysílač) generuje m'AS (= Sender přijal zprávu od A), pak mSR (= vysílání Sender -> Receiver), druhý automat (přenos. kanál) přejde do stavu 2 a generuje buď m'SR (= příjem Sender -> Receiver) nebo error, třetí automat (přijímač) pokračuje obdobným způsobem, … Tento automat lze samozřejmě také převést na gramatiku.

Formální modely v sítích dělíme na:

- **analytické**
  - **deterministické**
  - **pravděpodobnostní**
- **simulační**

Model sítě obsahuje směrovače, transformace (NAT, tunelování, ...), filtry (propouští pakety podle určitých pravidel, např. firewall), podsítě, rozhraní apod.

K modelování sítí se používá Datalog (**síťový tok** je definován IP adresou, protokolem, portem a QoS parametry, v datalogu je tok predikát s omezením na určité parametry hlavičky paketu).

ACL (**access control list**) je seznam pravidel pro firewall. Tento seznam se prochází shora dolů a použije se první pravidlo, které na daný paket sedí, záleží tedy na pořadí. Např.:

```
permit icmp any any echo-reply
permit icmp any any echo
deny ip any 10.10.10.0 0.0.0.255
deny ip any 10.10.11.0 0.0.0.255
permit ip any any
```

V ACL mohou být chyby, např. pravilo může být výjimkou pravidla, které je někde před ním. Tyto chyby se dají hledat formálními metodami. **Normalizace ACL** znamená převod ACL na formát, kdy obsahuje pouze permit pravidla. Formální validace síťových konfigurací je žádoucí, protože je prováděna často lidmi a může tedy obsahovat chyby, a

navíc se často mění podle stavu sítě. Požadavky na konfiguraci jsou konektivita, bezpečnost, spolehlivost a výkonnost.

49. Obrazová data, jejich pořizování a možná poškození (možné reprezentace obrazu, obrazové snímací čipy a zařízení, jejich vlastnosti, vady pořízeného obrazu, optimální filtrace obrazu). ZPO

Obrazová data pořizujeme různým způsobem – např. fotoaparátem, počítačovou syntézou (rendering), medicínskými zařízeními apod. Fotoaparát pořizuje obraz pomocí optické soustavy (sady čoček, zrcadel apod.) a **snímacího čipu**, což je mřížka buněk citlivých na světlo. Čipy jsou dnes především dvou typů:

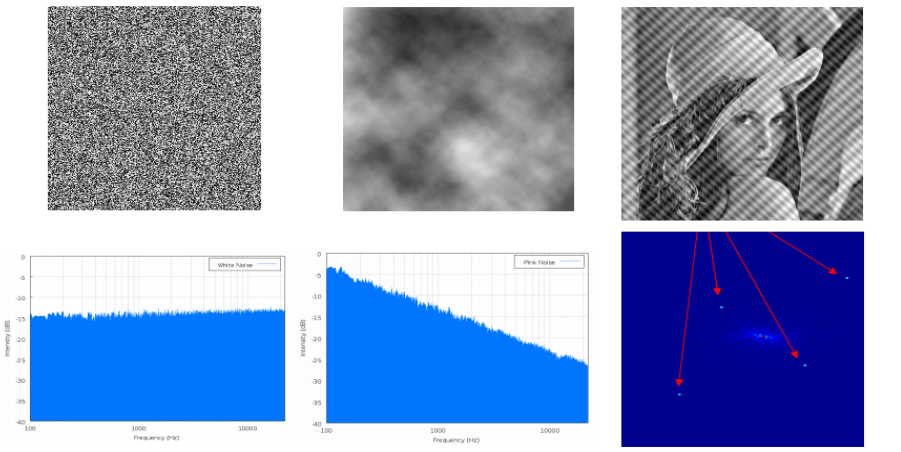


Illustration 104: šumy podle frekvence

- **CCD** (**Charge-coupled device**) – Snímá celý obraz najednou, můžou se vyskytnout bílé pruhy v obraze kvůli struktuře čipu – po sejmutí obrazu se přečte nejvrchnější řádek pixelů, potom se nasune druhý řádek na místo prvního, přečte se atd. Právě kvůli vertikálnímu posouvání obrazu při čtení mohou vznikat pruhy kvůli nedokonalostem přenosu. Každá světlovitlivá buňka je kondenzátor, který se před expozicí nabije a během ní se potom působením světla vybíjí – podle zbývajícího náboje se potom pozná, kolik světla na buňku dopadlo. Používá se spíš u foťáků.
- **CMOS** – Jsou levnější a mají nižší spotřebu než CCD čipy, avšak trpí větším šumem. Obraz při snímání skenují po řádcích, tzn. ne najednou – může se objevit zkreslení při extrémně rychlých pohybech. Používá se spíš u kamer.

Illustration 105: aditivní bílý Gaussův šum

Chyby v obraze můžou vznikat při snímání, přenosu nebo zpracování. Dělíme je na

- **zkreslení** – Např. geometrické, způsobené čočkou, neostrost apod. Lze se snažit o kompenzaci pomocí geometrických transformací apod.
- **šum** – Nežádoucí signál v zaznamenaném signálu. Podle zroje může jít o šum tepelný, kvantizační, zaokrouhlovací, zaostřovací apod. Podle frekvenčních vlastností dělíme šum na:
  - **bílý** – Náhodný (nekorelovaný) šum, má konstantní výkonové spektrum (přibližně stejná energie na všech frekvencích).
  - **růžový šum** – Výkon šumu klesá s rostoucí frekvencí.
  - **úzkopásmový** – Projevuje se pouze na specifických frekvencích, proto jej lze dobře odhalit a filtrovat ve spektrální oblasti.
  - ...

  Dále šum dělíme podle povahy na:

  - **nezávislý** (**aditivní**) – Velikost šumu nezávisí na velikosti obrazového signálu (např. nefunkční pixel na snímacím čipu).
  - **závislý** (**multiplikativní**) – Velikost šumu závisí na velikosti obrazového signálu

Illustration 106: šum typu "pepř a sůl"

Gaussův šum je druh šumu, jenž postihuje všechny pixely obrazu a jeho amplituda má normální (Gaussovo) rozdělení pravděpodobnosti (tzn. většina pixelů má střední hodnotu šumu, jenom pár má extrémní hodnoty šumu). Bílý Gaussův šum většinou dobře aproximuje reálný šum.

Šum typu "**pepř a sůl**" (také **impulzní** nebo **výstřelový**) je šum postihující pouze ojedinělé pixely extrémními hodnotami. Lze odstraňovat mediánovým filtrem.

Šum můžeme odstraňovat např. průměrováním (z více snímků, nebo i v jednom snímku z okolí každého pixelu – rozmazání, např. Gaussovkou), nelineární filtrací (mediánový filtr – vybere hodnotu uprostřed seřazené posloupnosti - nebo rotující maska apod.). **Optimální filtrace** je filtrace snažící se o úplné odstranění vad obrazu díky znalosti

parametrů obrazu a vad. Metody optimální filtrace se liší podle toho, co o systému víme. Patří sem:

- **Wienerův filtr** – Snaží se minimalizovat střední kvadratickou chybu rozdílu původního a vyfiltrovaného obrazu (minimum z $(f(i,j) - f'(i,j))^2$), předpokládá nezávislý šum, jehož spektrum známe. Dá se implementovat jako FIR filtr. Umí odstraňovat např. rozmazání pohybem.
- **vázaná dekonvoluce** – Známe celkovou energii šumu. Většinou se provádí ve spektru.
- **slepá dekonvoluce** – Neznáme nic.

Obraz lze chápat např. jako funkci 2 proměnných nebo 2D signál: $I_{x,y} = f(x,y)$. Také jej lze reprezentovat výčtem bodů s jinou barvou než má pozadí nebo jako množinu bodů splňující určité kritérium (přímka, kružnice, ...). Nejčastěji však reprezentujeme obraz jako **rastr** vzorků – diskrétní reprezentace, musíme pamatovat na vzorkovací teorém ($f_{max} < ½ f_s$, splnění napomáhá nedokonalost optiky fotoaparátů, která trochu "rozmazává" obraz). Hodnoty vzorků jsou obvykle barvy v určité reprezentaci (RGB, HSV, ...). Nejvíce informace je v jasové složce, proto se obraz často reprezentuje v úrovních šedé nebo se převádí do prostoru intenzitní složky plus dvou barevných (např. YCbCr). Hodnoty pixelů se též vzorkují, tzv. kvantují, nejčastěji každá složka na 8 bitů, obšas se však používá i double nebo čistě černobílý obraz. Počet bitů na pixel udává tzv. **barevnou hloubku**.

K filtraci obrazu používáme často kovoluční filtry, které lze rychle aplikovat ve frekvenčním spektru násobením (konvoluční teorém), pomocí FFT. Lze je použít k rozmazávání, doostřování, hledání hran apod. Rozmazáním se lze částečně zbavit šumu. Šum můžeme dále odstraňovat nelineárními filtry, např. mediánem. Pro detaily viz otázku 48.

Lineární filtr *f* je filtr, který je lineárním zobrazením, tzn platí:

$f(a\ x_1 + b\ x_2) = a\ f(x_1) + b\ f(x_2)$

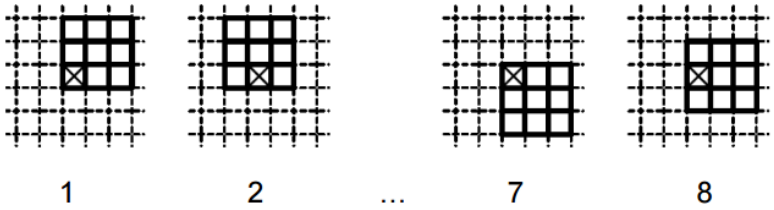


Illustration 107: filtrace rotující maskou - Vybere se pozice filtru, kde má jas nejmenší rozptyl a potom se filtruje mediánem.

50. Transformace obrazu (jaké se používají transformace při zpracování obrazu, typické příklady použití transformací při zpracování obrazu). ZPO

Transformace rozdělujeme podle oblasti, z níž se při zpracování pixelu využívají informace, na:

- **bodové**: Transformace pracují jen s jednotlivými pixely, jedná se tedy o funkci aplikovanou na každý pixel. Příkladem je např. převod na stupně šedi nebo prahování.
- **lokální**: Transformace pracuje s okolím zpracovávaného pixelu. Příkladem je např. konvoluční filtr.
- **globální**: Transformace pracuje s informací z celého obrazu, příkladem je např. ekvalizace histogramu, gamma (mapuje jas na nelineární stupnici a alokuje tak více bitů pro tmavší odstíny, protože člověk vnímá rozdíly v světlejších oblastech méně než v tmavých).

Podle typu můžeme dále dělit transformace na:

- barevné
- geometrické
- frekvenční
- **integrální** – Třída transformací, mezi něž patří např. Fourierova transformace, Laplaceova transform., DCT, vlnková transformace apod. Většinou mají i inverzní transformaci. Integrální transformace T má tvar:

  $$T(u)=\int_a^b f(t)K(u,t)dt$$ kde $K(u,t)$ je tzv. jádro.

  2D (popř. vícedimenzionální) DCT a DFT se dají dělat tak, že se nejdřív udělá transformace pro každý řádek (sloupec) a potom, na vzniklém poli, pro každý sloupec (řádek).

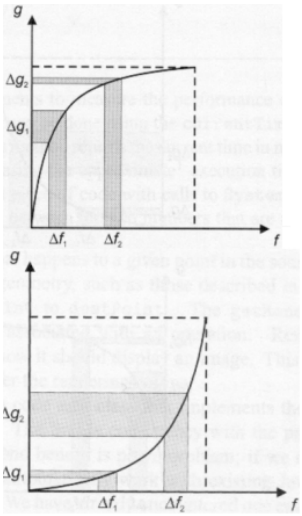


Illustration 108: logaritmická a exponenciální transformace

Lineární transformace je transformace, která zachovává sčítání a násobení akalárem. Tzn. transformace f: $V \rightarrow W$ z vektor. prostoru V do W je lineární, pokud platí obě tyto podmínky:

$f(x + y) = f(x) + f(y)$

$f(ax) = a\,f(x)$ kde a je skalár

Mezi lineární transformace patří identita, změna měřítka, rotace, avšak nikoli např. posun. Lineární transformace v n-rozměrném prostoru se dá vyjádřit $n \times n$ maticí.

Afinní transformace je transformace, která zachovává rovnoběžnost a dělicí poměr (a rovné čáry zůstávají rovné). Je to např. posunutí, rotace, zrcadlení, změna měřítka, zkosení, identita a jejich skládání. Afinní transformace je v podstatě lineární transformace plus možný posun. Každá lineární transform. je tedy afinní, avšak ne naopak. Afinní transformace v n-rozměrném prostoru se dá vyjádřit $(n + 1) \times (n + 1)$ maticí za použití homogenních souřadnic.

Homogenní souřadnice $(x_1, x_2, \ldots, x_n)$ v n-rozměrném prostoru je $(n + 1)$-tice $(x_1 / w, x_2 / w, \ldots x_n / w, w)$. Každý bod má tedy nekonečně mnoho reprezentací v homogenních souřadnicích. Tyto souřadnice se používají díky možnosti reprezentovat afinní transformace pomocí matic a jejich skládání maticovým násobením (matice musíme uspořádat v opačném pořadí, např. pro provedení otočení a poté posunu musíme udělat součin POSUN * ROTACE, ne opačně).

Geometrické transformace mapují pozici v obraze na jinou pozici v obraze, tedy $(x_1, y_1) \rightarrow (x_2, y_2)$. Jedná se např. o posun, rotaci, změnu měřítka, afinní transformace apod. Tyto tranformace se typicky nemůžou provádět průchodem zdrojového obrazu, protože by nebylo zaručeno, že se získají hodnoty všech výstupních pixelů (v obraze by byly "díry"). Proto se prochází obraz cílový a inverzní transformací se hledá, který pixel zdrojového obrazu se na danou pozici namapuje. Využití geometrických transformací je např. pro odstraňování zkreslení (např. čočkou, perspektivou apod.). Geometrické transformace mohou být popsány funkcí:

$$\begin{vmatrix} x_2 \\ y_2 \end{vmatrix} = A \begin{vmatrix} x_1 \\ y_1 \end{vmatrix} + B$$

Tyto transformace lze popisovat maticemi a provádět pomocí homogenních souřadnic.

Při geometrických transformacích se musíme vypořádat s tím, že může být mapovaná pozice mimo obraz nebo mimo pixel. Souřadnice mimo pixel se řeší **interpolací** (nearest neighbour, bilineární, trilineární, DFT, spline, ...).

Matematická morfologie se zabývá studiem tvaru a vnitřní struktury objektů. Mezi morfologické operace patří např. dilatace, eroze, hledání kostry objektu apod.

Příklady transformací jsou:

- **prahování:** Jedná se o kvantizaci hodnoty pixelu, jde tedy o bodovou transformaci. Dá se využít jako jednoduchá segmentace, avšak je zde problém s proměnlivým jasem v obraze, se stíny apod. Z těchto důvodů existují různá vylepšení, jako např. **adaptivní prahování**, které dynamicky upravuje práh podle pixelů v okolí. Výsledkem je většinou binární obraz, ale může mít i více úrovní.
- **logaritmická**, resp. **exponenciální** transformace: Nelineární bodová transformace sloužící pro zvýšení detailů v tmavých, resp. světlých oblastech obrazu.
- **konvoluce**: Klasická 2D konvoluce s daným jádrem, jde o lokální operaci. Pozor: při provádění je potřeba konvoluční masku otočit horizontálně a vertikálně! (jinak jde o korelaci)
- **ekvalizace histogramu**: Globální transformace, která změní rozložení intenzit v obraze tak, aby se v něm vyskytovaly pokud možno intenzity v širokém rozmezí. Tím se využije celá jasová stupnice. Algoritmus nejdřív spočítá kumulovaný histogram: $K(i)=\sum_{j=0}^{iH}(j)$, výsledná intenzita je potom $\frac{K(i)}{M.N}255$.

Homografie hledá transformaci mezi dvěma obrazy (dokáže např. namapovat dva snímky z různých úhlů na sebe), má 9 parametrů (posun a rotace kamery).

Mezi další transformace patří konvoluce (jádro se obrací!), korelace (jádro se neobrací) apod.

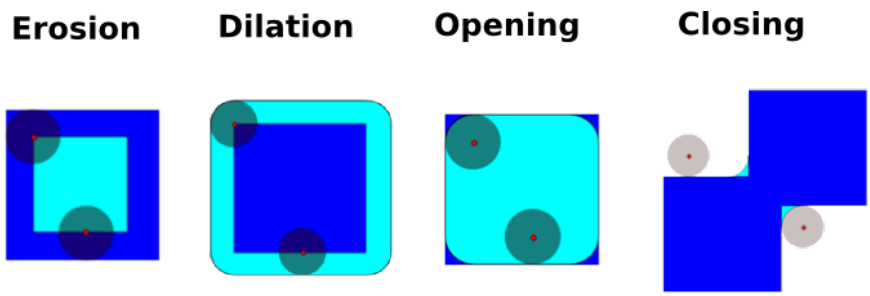


Illustration 109: matematická morfologie

51. Filtrace obrazu (definice lineární filtrace, typické příklady použití filtrů, použití rychlá konvoluce s využitím FFT, návrh lineárních filtrů, nelineární filtrace). ZPO

Lineární filtrace je filtrace *f*, pro kterou platí tzv. **princip superpozice**:

$f(x_1 + x_2) = f(x_1) + f(x_2)$,

tzn. když sečteme dva obrazy a přefiltrujeme, dostaneme stejný výsledek, jako když každý nejdřív přefiltrujeme a potom je sečteme. Mezi lineární filtrace patří např. Fourierova transformace, inverzní Fourierova transformace nebo konvoluční filtry.

Konvoluční filtr, neboli **2D FIR filtr**, je definován svým **jádrem** *k*, tzn maticí lichých rozměrů (aby měla střed). Konvoluce se potom prvede tak, že se jádro otočí horizontálně a vertikálně (pokud se neotočí, jedná se o korelaci), čímž dostaneme *k'*, a jeho střed se postupně posunuje po prvcích vstupní matice, korespondující prvky vstupu a jádra se vynásobí a všechno se nakonec sečte, čímž dostaneme výstupní hodnotu na dané pozici. Matematicky:

$$y(m,n) = \sum_{i \in I} \sum_{j \in J} x_{(m+i,n+j)} k'_{i,j}$$

Rychlá konvoluce je způsob rychlého provedení konvoluce využívající faktu, že konvoluce v prostorové oblasti je ekvivalentní násobení ve spektrální oblasti (tzv. **konvoluční věta**). Tedy:

$y(x) = IFFT(FFT(x) \cdot F)$,

kde *F* je spektrum jádra filtru. Je-li *N* šířka jádra filtru, pak časová složitost běžné konvoluce je $O(N^2)$, zatímco složitost rychlé konvoluce je $O(N \log N)$, tedy linearitmická, což je o řád lepší.

Konvoluční filtry lze použít pro zpracování obrazu, např filtr s jádrem

0	-1	0
-1	5	-1
0	-1	0

realizuje doostřovací filtr (viz obr.). Dále lze takto relizovat např. detekci hran (ovšem pouze v jednom směru, detekce hran ve více směrech není lineární filtrace), rozmazání (dolní propusť, zbavení se vysokofrekvenčního šumu) apod. Filtr obecně zvýrazňuje takové vzory v obraze, které se podobají jeho jádru.

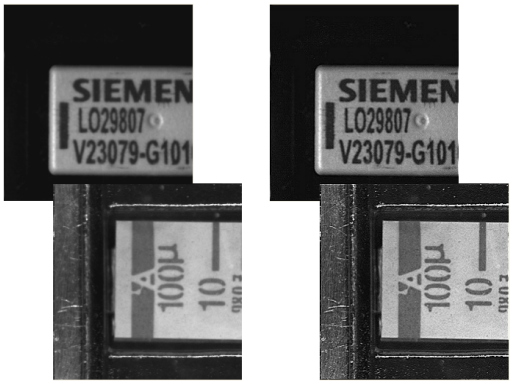
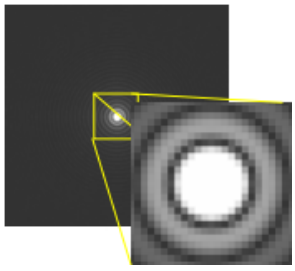


Illustration 110: doostření lineárním filtrem

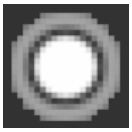
Návrh koeficientů filtru se provádí následujícím způsobem:

1. Máme danou frekvenční charakteristiku filtru, který chceme vytvořit, např.:

2. Z této charakteristiky získáme koeficienty inverzní Fourierovou transformací. Nastává zde však problém: výsledek bude obecně nekonečný, v našem případě:



3. My chceme ovšem jenom malé, konečně velké jádro filtru. Proto výsledek musíme nějakým způsobem omezit, což ovšem způdobí menší či větší artefakty ve výsledném obraze. Výsledné jádro může vypadat např. Takto:



Nelineární filtrace je filtrace, která nesplňuje podmínky lineární filtrace. Používá se většinou pro filtraci výsledku lineární filtrace. Typickým příkladem je např. **Sobelův filtr**, který dokáže detekovat hrany v horizontálním i vertikálním směru zároveň. Je definován jako:

$$S = \sqrt{(f_1^2 + f_2^2)}$$

kde f_1, resp. f_2, jsou lineární filtry pro vertikální, resp. horizontální hrany.

Dalším typickým příkladem jsou tzv. **rank-order** metody. Ty fungují takto:

1. Pro každý pixel vstupu se vezme jeho okolí.
2. Pixely z okolí se seřadí podle hodnoty.
3. Vybere se hodnota, která je daná parametrem rank-order metody, např. 25 % vybere hodnotu na čtvrtinové pozici seřazené posloupnosti.

Mediánový filtr je speciálním případem rank-order filtru, konkrétně 50 % (vybírá prostřední hodnotu, používá se např. pro filtraci šumu typu pepř a sůl).

Dalšími nelineárními filtry jsou tzv. **transformace** – jsou to operace, které mají svou inverzi, tzn. existuje zpětná transformace.

52. Vodoznaky (watermarks) a jejich využití (vymezení pojmu vodoznak, základní principy a vlastnosti vodoznaků, typické příklady využití a vlastností vodoznaků). ZPO

Vodoznak (**watermark**) je určitý druh značky vložený přímo do užitečných dat (nikoliv metadat), jehož účelem je chránit data proti kopírování, neoprávněnému zveřejňování, poskytovat možnost určení původu dat, přidání dodatečných informací (skrytých nebo viditelných) apod. Vodoznaky klasifikujeme následujícím způsobem:

- podle viditelnosti:
  - **viditelné** – např. pro označení autora, obtížně odstranitelné
  - **neviditelné** – při běžném použití se neprojeví, např. pro zjišťování integrity dat
- podle způsobu vložení:
  - **v časové oblasti** (přímá aplikace v obraze)
  - **ve frekvenční oblasti** (modifikace koeficientů DFT, DCT apod.)
  - ...
- podle typu vkládaných dat:
  - binární data
  - obraz
  - ...
- podle robustnosti:
  - **robustní** – Vodoznak je odolný vůči různým operacím, jako např. komprese, změna velikosti obrazu apod., používá se tehdy, když potřebujeme mít data pokud možno neoddělitelně označená.
  - **křehké** – Vodoznak se úpravou dat poškodí, používá se ke zjišťování, zda byl obraz upravován.
- podle požadavků detekce:
  - **privátní** – Pro detekci vodoznaku je potřeba mít originál.
  - **veřejné** – Pro detekci není potřeba originál.
- ...

Mezi základní metody implementace vodoznaků patří např.:

- **LSB modulace** – Data se kódují v nejméně významných bitech obrazu, vodoznak není viditelný, je křehký (např. Komprese jej zničí). U RGB obrazu se používá překódování do YUV a ukládání do Y kanálu, což je odolnější vůči modifikaci barev a kompresi.
- **modifikace DCT koeficientů** – Neviditelný vodoznak, který je zároveň mnohem robustnější než LSB modulace. Modifikace koeficientů nižších frekvencí je robustnější, ale více ovlivňuje kvalitu obrazu. Tato metoda je nativně odolná vůči změně jasu.
- **disjunktní množiny** – Robustní vodoznak, princip je následující:
  1. Obraz se rozdělí na dvě přibližně stejně velké množiny pixelů A a B.
  2. Toto rozdělení je dáno privátním klíčem, což je funkce příslušnosti pixelu do množiny A nebo B.
  3. Intenzita pixelů v A se zvětší o k, intenzita pixelů v B se sníží o k.
  4. Detekce probíhá tak, že obraz znovu rozdělíme na množiny A a B a v každé spočítáme průměrnou intenzitu. Pokud je rozdíl intenzit A a B přibližně 2k, je vodoznak obsažen, pokud je rozdíl intenzit přibližně 0, pak vodoznak není obsažen.
- **aplikace MD5** – Používá se pro detekci změn v obraze na úrovni bloků (dají se zjistit konkrétní změněné bloky), princip je následující:
  1. Obraz se rozdělí na 8 x 8 bloky.
  2. Odstraní se LSB bity.
  3. Zbytek bitů se zřetězí.
  4. Na výsledek zřetězení se aplikuje MD5.
  5. Výsledek se zkombinuje s binárním obrazem vodoznaku.
  6. Výsledek se uloží do LSB bitů původního obrazu.
- **visuální kryptografie** – Vkládá vodoznak do více po sobě jdoucích binárních obrazů. Vodoznak se získá aplikací logické operace na po sobě jdoucí snímky. Při prohlížení jednoho snímku vodoznak není vidět, jsou potřeba oba. Funguje takto:
  1. vezne se zdrojový obraz a zvětší se 2x (každý zdrojový pixel se převede na 2x2 pixel)
  2. každý pixel se umístí náhodně uvnitř nového 2x2 pixelu

3. druhý obraz sestrojíme pomocí tohoto obrazu tak, aby se aplikací logické operace ukázal vodoznak

- **vodoznaky v textu** – Modifikcae typografických pravidel (např. šířka mezery apod.).
- ...

Steganografie je technika, kdy se do určitých dat vkládají jiná skrytá data.

Útoky na vodoznaky, tzn. jejich odstraňování, se dělají metodami závisejícími na typu vodoznaku. Viditelné vodoznaky lze odstraňovat ručně, oříznutím, překrytím apod. Neviditelné vodoznaky se můžeme pokusit zničit vložením šumu, kompresí, různými transformacemi, vložením jiného vodoznaku, průměrováním několika obrazů od různých uživatelů apod.

Vodoznaky lze přidávat i do jiných médií, než je obraz, např. do zvuku (přidání neslyšitelného šumu apod.), videa (kombinace zvuku a obrazu), textu (modifikace parametrů zobrazeného textu, bílé znaky apod.) apod.

53. Detekce hran, segmentace (vymezení pojmů detekce hran a segmentace, možné aplikace algoritmů a jejich důvody, typické případy nasazení algoritmů). ZPO

Hrany jsou prudce kontrastní přechody v obraze, tzn. vysoké frekvence. Hrany jsou pro zpracování obrazu důležité proto, že pomocí nich můžeme vymezit objekty v obraze (oproti pozadí), tzn. provádět **segmentaci** (rozdělení obrazu na množiny pixelů, segmentů) – toto ovšem závisí na interpretaci obrazu, proto neexistuje jedna dokonalá metoda, musí se volit dle situace. Detekce hran je jen jednou metodou, kterou lze segmentaci provádět, dalšími jsou např. statistické (prahování, ...), nebo znalostní (srovnávání se vzory) metody.

Detekce hrany probíhá obecně takto:

1. **filtrace** – Potlačení šumu, např. rozmazáním.
2. **diferenciace** – Zvýraznění kontrastních oblastí (oblastí hran).
3. **detekce** – Samotné nalezení hran, např. pomocí prahování.

Hrana je na úrovni pixelu reprezentována velikostí a směrem, tzn. vektorem.

Kvalita hranového detektoru je dána:

- odolností proti šumu
- dobrou lokalizací – Tzn. vybrané hranové pixely jsou co nejblíže skutečné hraně.
- jedinečnou odezvou – Tzn. ideálně by měla existovat jen jedna odezva na skutečnou hranu (žádné detekce "polohran").

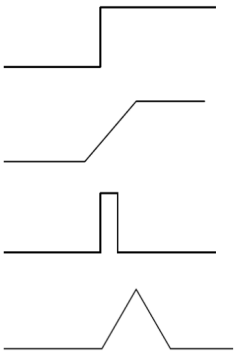


Illustration 111: různé modely hran

Mezi metody detekce hran patří:

- Hledání vysoké první derivace v obraze, tzn. rychlých změn. To lze udělat např. tzv. **Sobelovým operátorem** – to je nelineární operace, která detekuje jednak horizontální (G_x) a jednak vertikální (G_y) hrany v obraze a z těch potom vytvoří výsledný obraz detekce jako $G=\sqrt{\left(G_x^2+G_y^2\right)}$. Obarazy G_x a G_y se získají pomocí konvolučních matic:

$$G_x=\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix};G_y=\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix}$$

  Dalšími podobnými operátory jsou např. Prewittův (používá v maticích pouze 1, 0, -1), Robinsonův apod. Dále existuje **Cannyho detektor**, který dobře splňuje podmínky kvality detekce hran (šumovou odolnost, lokalizaci a odezvu), funguje takto:

  1. Gaussovým filtrem potlačí šum.
  2. Zderivuje obraz (dostane gradient).
  3. Zbaví se hodnot, které nejsou lokálním maximem ve směru gradientu.
  4. Aplikuje prahování s hysterezí (tzn. slabé hrany jsou ponechány pouze tehdy, pokud jsou blízko nějaké silné).

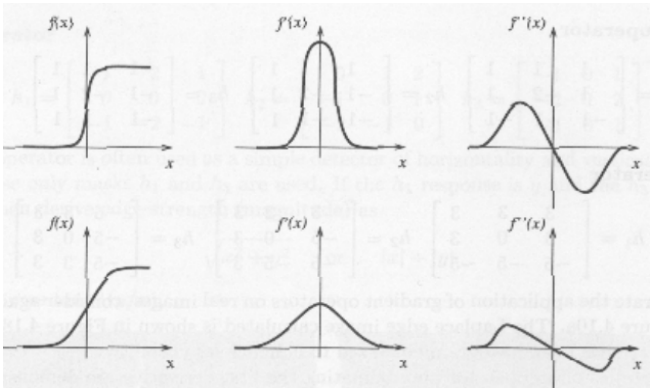


Illustration 112: funkce, její první a druhá derivace můžou sloužit k detekci hran

- Hledání charakteristik v druhé derivaci, např. pomocí hledání průchodu nulou nebo tzv. **Laplaciánu** (součin derivace ve směru *x* a *y*, je invariantní vůči rotaci):

$$H=1/4\begin{bmatrix}0 & -1 & 0\\ -1 & 4 & -1\\ 0 & -1 & 0\end{bmatrix}$$

Illustration 113: detekované hrany

Podobným operátorem je např. **LoG** (**Laplacian of Gaussian**), což je Laplacián obrazu vyhlazeného

Gaussovým rozmazáním (ve výsledku dostaneme opět pouze jedno konvoluční jádro, protože konvoluce je asociativní, tzn. nemusíme nejdřív rozmazat obraz a potom aplikovat Laplacián, stačí konvoluce s předpočítaným jádrem), nevýhodou je nepřesnost lokalizace a detekce falešných hran. Laplacián není separabilní. Dále existuje **DoG** (**difference of Gaussians**), který aproximuje LoG a je separabilní.

- Konvolucí – aproximujeme derivaci pomocí konvolučního jádra, např. (pro vertikální hrany):

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix}$$

- Jiné metody jako např. **Houghova transformace** – hledá v obraze parametrické objekty různých typů, např. úsečky, kružnice apod. Je odolná proti šumu ale výpočetně náročná. Objekty se hledají v parametrickém prostoru, do něhož se body obrazu převádějí. Další metodou jsou **aktivní kontury** (**snakes**) – principem je mít křivku, která se deformuje silami v obraze tak, aby se vyrovnala energie vně a uvnitř křivky, která tak "obepne" objekt v obraze.

Díky detekci hran a segmentaci můžeme provádět rozpoznávání a klasifikaci objektů např. v počítačovém vidění, můžeme provádět analýzu obrazu, jeho processing, stabilizaci videa apod.

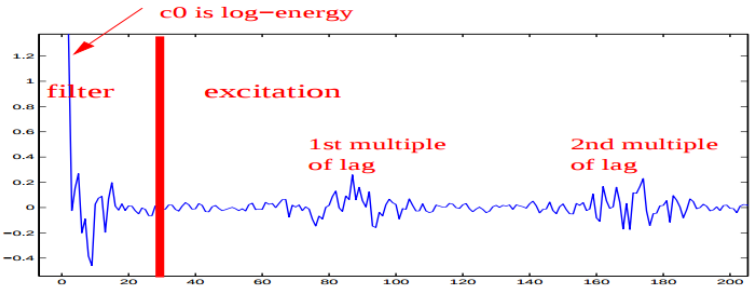
54. Cepstrum (definice, způsoby výpočtu, Mel-frekvenční cepstrální koeficienty). ZRE

Cepstrum je signál, který získáme jako:

$c(n) = \text{IFFT}(\ln|\text{FFT}(s(n))|^2)$.

Tedy uděláme FFT vstupního signálu, tu umocníme na druhou (dostaneme spektrální hustotu výkonu), zlogaritmujeme a uděláme zpětnou FFT. Cepstrum je symetrické, tedy $c(n) = c(-n)$.

Cepstrum je důležité v rozpoznávání řeči, protože dokáže z konvoluce buzení (hlasivek) a filtru (řečového ústrojí) (ve spektrální oblasti je konvoluce součin) udělat součet (díky logaritmu) a obě složky oddělit (tzv. **dekonvoluce**) – odděleny jsou na frekvenční ose (pro vzorkovací frekvenci 8000 Hz je tato hranice na frekvenci 30 Hz), např.:



Typicky nás zajímají jenom parametry řečového ústrojí a buzení zahazujeme. Z cepstra jde zpětně získat původní signál (můžeme např. vynulovat koeficienty filtru a zrekonstruovat signál – pak dostaneme signál samotného buzení), avšak ne zcela – při umocňování na druhou jsme ztratili fázi signálu (ale může se použít např. 0 nebo fáze původního signálu).

Základní tón (značený F_0) je (základní) frekvence, na které kmitají hlasivky – pohybuje se v rozmezí 90 – 400 Hz. Kdybychom se ale snažili zbavit se základního tónu zahozením všech frekvencí pod 400 Hz, nefungovalo by to, protože základní tón ovlivňuje i vyšší frekvence díky konvoluci, a navíc bychom mohli přijít o první formant. Proto musíme použít cepstrum.

Alternativně můžeme použít taky tzv. **Mel-frekvenční cepstrum** (MFCC). Jeho výhodou je, že může mít různé rozlišení na různých frekvencích, stejně jako lidské ucho (to vnímá nižší frekvence s vyšším rozlišením než vyšší). Toto cepstrum sestrojíme následovně:

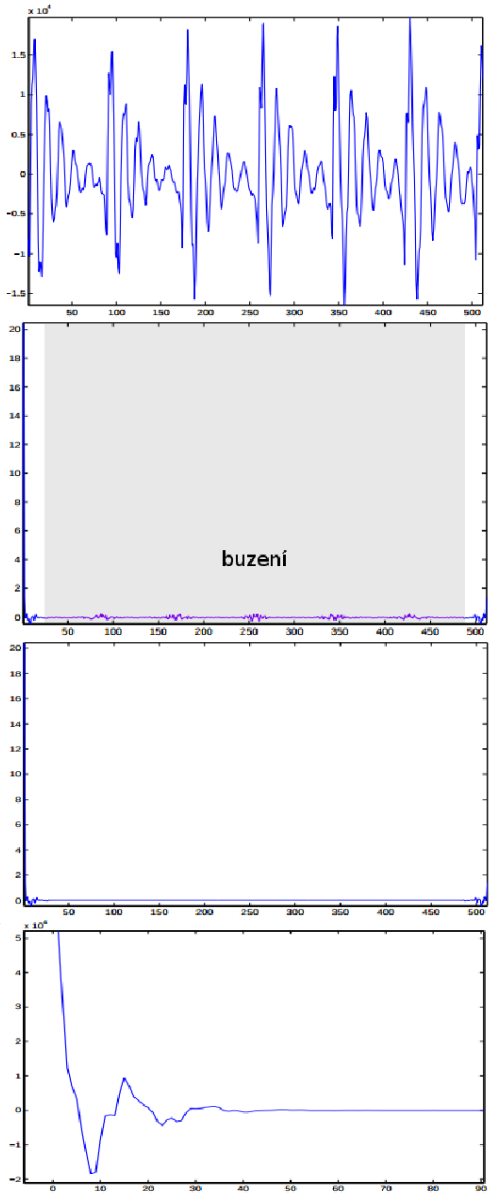


Illustration 114: cepstrum příklad: signál, jeho cepstrum, cepstrum s vynulovaným buzením, rekonstruovaný signál obsahující pouze řečové ústrojí (filtr)

1. Uděláme DFT vstupu.
2. Provedeme nelineární úpravu pomocí logaritmu (abychom se přiblížili lidskému vnímání) – tzn. převedeme na tzv. Melovou osu.
3. Na frekvenční osu rozmístíme filtry pro několik pásem.
4. Pro každý filtr vyfiltrumeje vstupní signál a změříme jeho energii. Z té uděláme logaritmus.
5. Z celého signálu teď uděláme DCT (která nahrazuje IFFT).
6. Nyní máme Mel-frekvenční cepstrální koeficienty pro každý použitý filtr (tedy frekvenční pásmo).

Lokální maxima spektra hlasového ústrojí (cepstra) nazýváme **formanty**. Formanty ukazují, které frekvence jsou ústrojím rezonančně zesilovány. Označujeme je F_1, F_2 atd. Podle jejich pozic můžeme poznat, jaká hláska je vyslovována.

Parametry řeči lze také odhadnout pomocí **LPC** (**linear prediction coding**).

Konvoluce dvou signálů je ekvivalentní sčítání jejich cepster.

Pro příklad formantů, pro hlásku *i* jsou průměrné formanty F1 = 240 Hz, F2 = 2400 Hz.

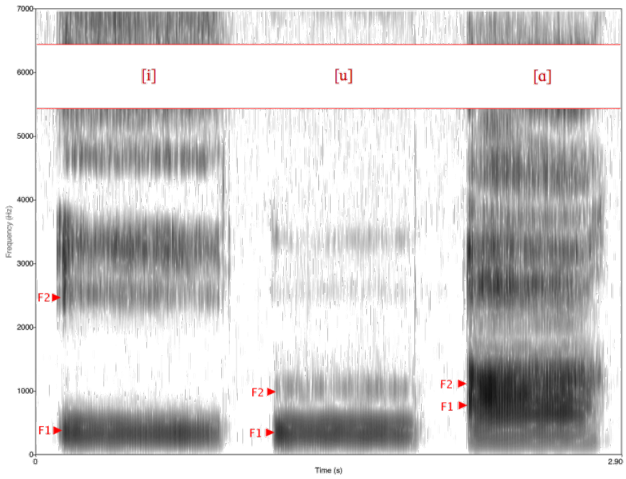


Illustration 115: formanty hlásek "i", "u" a "a"

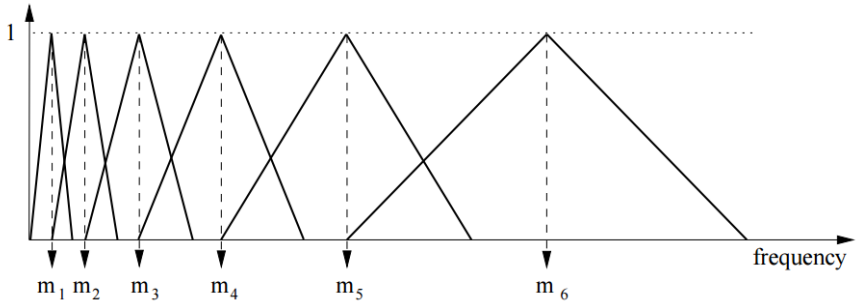


Illustration 116: rozmístění filtrů ve spektru pro Mel-frekvenční spektrum

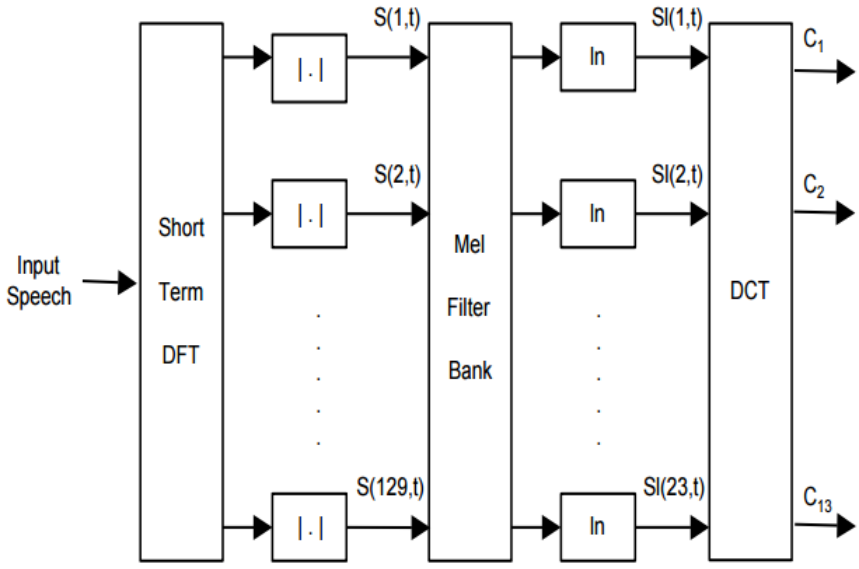


Illustration 117: blokové schéma výpočtu MFCC

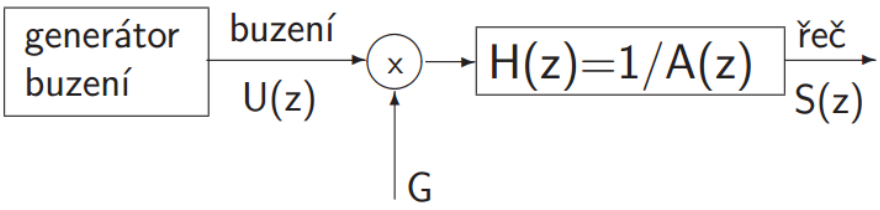
55. Lineární predikce (podstata, výpočet parametrů LP filtru, použití lineární predikce). ZRE

Lineární predikce (**LPC**) je metoda odhadu koeficientů filtru pro tvorbu řeči. **Vokodéry** (hlasové dekodéry) pracují s určitým modelem hlasového ústrojí a ze vstupní řeči odhadují parametry tohoto modelu, tyto parametry potom posílají příjemci a ten z nich poté zpětně řeč syntetizuje.

LPC se snaží odhadnout parametry následujícího modelu hlasového ústrojí:

- **buzení** – šum
- **hlasivky** – dolní propusť, která mění vstupní šum ná základní tón
- **hlasový trakt** (artikulace) – skládá se ze série rezonátorů, které mění vstupní základní tón na řeč přidáním vyšších frekvencí

Dohromady lze tento model realizovat pomocí čistě IIR filtru 1/A(z), kde A(z) je polynom $1 + a_1z^{-1} + ... a_pz^{-P}$ (má řád $P = 2k + 1$, kde k je počet formantů). Za užitečný počet formantů považujeme $k = 4$ nebo 5 (pro vyšší frekvence volíme vyšší k, abychom postihli i vyšší části spektra). Každému rezonátoru hlasového traktu odpovídá jeden tzv. **formant –** koncentrace energie kolem určité hodnoty na frekvenční ose.



Koeficienty filtru 1/A(z) se snažíme odhadnout ze vstupní řeči. LPC postupuje při odhadu takto:

- Vezme vstupní řeč (tedy buzení profiltrované filtrem 1/A(z)) a za něj vloží **inverzní filtr** A*(z).
- Mění parametry filtru A*(z) tak dlouho, dokud není energie výstupního signálu (což je buzení) minimalizovaná.
- V takovém případě A*(z) = A(z) a můžeme tedy zjistit i 1/A(z).

Koeficienty filtru A(z) se dají vypočítat ze soustavy rovnic, která se dá získat z předpokladu, že hledáme minimum funkce (chyby predikce, tzn. rozdílu skutečné a předpovězené hodnoty) klasicky pomocí derivace této funkce.

Konkrétně hledáme minimum funkce $E=\sum_n e^2(n)$. Pro nalezení minima vyjádříme signál e pomocí rozdílu vstupního signálu a výstupního signálu (suma součinů vzorků vstupního signálu a koeficientů) a položíme derivaci E rovnu nule. Tím dostaneme soustavu lineárních rovnic. Signál chyby je:

$$e(n)=s(n)-s'(n)=s(n)-(-\sum_{i=1}^{P} a_i s(n-i))$$ (s je původní a s' předpovězený signál)

Derivaci položíme rovnu 0:

$$\delta / \delta a_j((\sum_n s(n)+\sum_{i=1}^{P} a_i s(n-i))^2)=0$$ (parciální derivace podle a_j)

$$\sum_n 2(s(n)+\sum_{i=1}^{P} a_i s(n-i))s(n-j)=0$$

V dalším odvozování musíme vzít v potaz, jak zacházet se signálem vně rámce. Existují dvě metody:

- **kovarianční** – Signál vně je neznámý, i když je tam signál zpožděný. Tato metoda není moc vhodná, musí se počítat plná soustava rovnic a vede na nestabilní filtr 1/A(z).
- **korelační** – Signál vně je známý, ale nulový. Jedná se o lepší metodu, protože na diagonálách jsou stejné hodnoty a matice je navíc symetrická, tzn. matice je tzv. **Töplitzova**. Díky tomu můžeme použít rychlý

algoritmus pro řešení soustavy – **Levinson-Durbin** algoritmus.

Dalším odvozením za použití korelační metody získáme výslednou P x P matici soustavy rovnic (kde $R(i)$ je i-tý autokorelační koeficient):

$R(0)\ a_1 + R(1)\ a_2 + ... + R(P - 1)\ a_P = -R(1)$ (na diagonále jsou $R(0)$, o jedno vedle $R(1)$ atd.)

$R(1)\ a_1 + R(0)\ a_2 + ... + R(P - 2)\ a_P = -R(2)$

...

$R(P - 1)\ a_1 + R(P - 2)\ a_2 + ... + R(0)\ a_P = -R(P)$

Lineární predikce má svůj název odvozený od faktu, že filtr $A(z)$ je **prediktorem** následujících hodnot signálu z několika hodnot předchozích. Čím lepší je predikce, tím je menší její chyba. Výhodou LPC je, že filtrací vstupního signálu inverzním filtrem dostáváme signál buzení (výstup hlasivek). Lineární predikce je v podstatě odhadování budoucích vzorků z minulých a dá se tak použít i jinde než jen v řeči. LPC je potom kódování založené na lineární predikci, které funguje takto:

- odhadnou se formanty
- odstraní se vliv se jejich vliv, čímž získáme hlasivky
- změříme parametry hlasivek
- pošleme informace o formantech a hlasivkách

LPC můžeme použít pro výpočet cepstra, čímž dostáváme tzv. **LPCC**. LPCC koeficienty se používají např. v rozpoznávání řeči, protže nejsou tak moc korelované.

56. Určení základního tónu (podstata, autokorelace, normalizovaná cross-korelace, metody zlepšení přesnosti). ZRE

Základní tón (**pitch**) je frekvence (tedy tón), na které kmitají hlasivky. Označuje se jako F_0. Perioda základního tónu (tedy $1/F_0$) se nazývá **lag** a vyjadřuje se ve vzorcích . F_0 nabývá hodnot od 50 Hz do 400 Hz (160 až 20 vzorků při F_S = 8000 Hz).

Základní tón je podstatný pro několik oblastí zpracování řeči, např. syntézu (vytváření melodie) nebo kódování řeči (přenáší se parametry řečového ústrojí, příznak znělý/neznělý a frekvence základního tónu, také se používá při tzv. dlouhodobé predikci).

Při určování F_0 se setkáváme s několika problémy: hlásky (ani znělé) nejsou zcela periodické, nízká energie signálu dělá určování obtížnějším apod. Dají se však použít některé metody, mezi něž patří:

- **autokorelace**: Vycházíme z předpokladu, že vstupní signál bude periodický na frekvenci F_0, tzn. bude si velmi podobný po periodě lagu. Podstatou metody je provádění korelace signálu s posunutou kopií sebe sama. Posun postupně zvyšujeme a hledáme, při kterém je korelační hodnota nejvyšší – tento posun potom indikuje lag. Autokorelační funkce má tedy tvar:

  $$R(m)=\Sigma_{n=0}^{N-1-m} s(n)s(n+m)$$

  Hledáme maximum této funkce v rozmezí, v němž se lag vyskytuje (20 až 160 vzorků). Znělost/neznělost se určuje porovnáním hodnoty nalezeného maxima s hodnotou nultého autokorelačního koeficientu, pokud je pod určitým procentem (musí se zvolit) nultého koeficientu, jde o neznělou hlásku, jinak o znělou.
- **normalizovaná cross-korelace**: Autokorelace má nevýhodu v tom, že se s postupným zvyšováním posunu využívá čím dál míň vzorků, protože se jich čím dál míň překrývá. **Cross-korelace** toto řeší tak, že používá celý signál, tzn. hodnoty mimo rámec bere z předcházejícího rámce. Problémem je, že energie předcházejícího rámce může být velmi odlišná a cross-korelační koeficienty budou tímto negativně ovlivněny. Toto řeší tzv. **normalizovaná cross-korelace**. Ta dělí každý vypočítaný koeficient hodnotou $\sqrt{(E_1 E_2)}$, kde E_1 resp. E_2 jsou energie originálního resp. posunutého rámce (tj. zkrácený původní rámec, před nějž se přidá část předchozího rámce).

Obě metody bohužel nedostatečně potlačují vliv formantů, tzn. v koeficientech se objevují další výrazná maxima než jen na F_0. Toto řeší tzv. **centrální klipování**. To předzpracovává signál pro korelaci tak, že ponechá jenom vysoké hodnoty. Existuje více variant centrálního klipování, mezi něž patří (c_L je zvolená ořezávací úroveň):

- $$c_1(x)=\begin{cases} x-c_L & x>c_L \\ 0 & -c_L\le x\le c_L \\ x+c_L & jinak \end{cases}$$
- $$c_2(x)=\begin{cases} 1 & x>c_L \\ 0 & -c_L\le x\le c_L \\ -1 & jinak \end{cases}$$

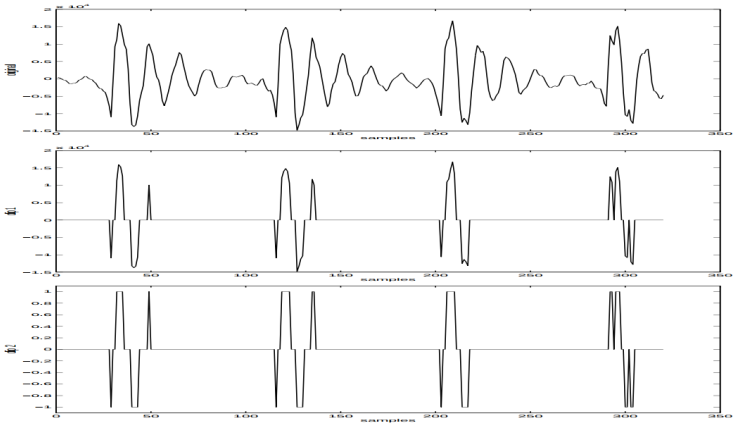


Illustration 118: centrální klipování

Hodnota c_L se musí volit pro každý rámec zvlášť, většinou jako 0,6 až 0,8 krát maximální absolutní hodnota v rámci. Je také vhodné zavést tzv. **úroveň ticha**, tzn. absolutní hodnotu, která když není v rámci překročena, tak se základní tón vůbec neurčuje.

Další metodou potlačení vlivu formantů je použití signálu **chyby lineární predikce** místo přímého vstupního signálu. Tento signál už neobsahuje informaci o formantech, pouze kmitání hlasivek, a je proto mnohem vhodnější pro určování F_0.

Mezi nepřesnosti při určování základního tónu patří např. častá detekce poloviničního nebo několikanásobného lagu, např.: 50, 50, 100, 50, 50 – v prostředním rámci jde evidentně o chybu. Tomuto se můžeme snažit zabránit např. **mediánovým filtrem** (vybere prostřední hodnotu ze seřazené posloupnosti) nebo tzv. **metodou optimálních cest**: Hledáme optimální "cestu" po hodnotách lagu přes několik rámců, přičemž se držíme určitých omezení a kritérií, např. že hodnota lagu se mezi dvěma rámci nemůže změnit příliš moc. Detekci několikanásobného lagu se dá také zabránit tzv. **desetinným vzorkováním**, tzn. nadvzorkováním signálu při výpočtu autokorelačních koeficientů.

(**kros**) **korelace** (**cross correlation**) je operace podobná skalárnímu součinu nebo konvoluci, jenom se signál "nepřeklápí". Formálně je definována jako:

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

Tato operace umožňuje hledat v signálu určitý vzor (nachází se tam, kde je výsledek korelace vysoký).

Autokorelace je korelace signálu se sebou samým. Ta umožňuje např. najít v signálu převažující frekvence (při posunu o periodu odpovídající frekvence je autokorelace vysoká). (pro detaily viz. Otázku 56.).

Následují další důležité věci související se zpracováním signálů:

Diskrétní Fourierova transformace (**DFT**) je operace, která převádí signál z časové domény do frekvenční domény. Je definována jako:

$$X_k = \sum_{n=0}^{N-1} x_n . e^{-2\pi j k n / N} \qquad k = 0 \ldots N-1$$

Výsledné komplexní funkci *X* říkáme **spektrum**. Spektrum zobrazujeme většinou jako **modul** (absolutní hodnota) a **argument** (úhel) komplexního čísla. Zpětná rekonstrukce signálu ze spektra (IDFT) je potom definována jako:

$$n_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k . e^{2\pi j k n / N} \qquad n = 0 \ldots N-1$$

Člen 1 / N v IDFT je tzv. normalizační člen a jedná se o konvenci, která se v různých definicích může lišit.

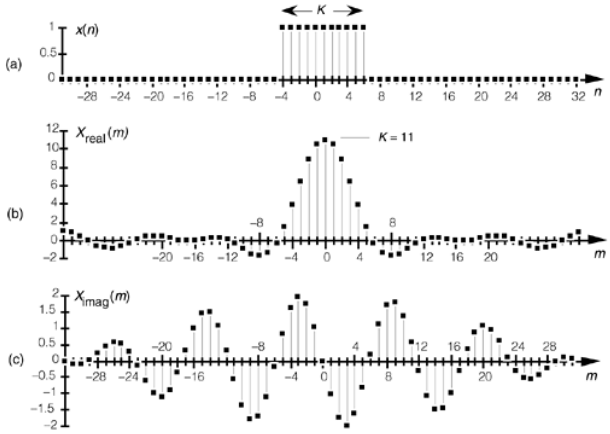


Illustration 119: příklad DFT

DF operací existuje několik:

- FT (Four. transform.) – Pro obecné (I neperiod.) signály, spektrum je spojité, pro reálné signály jsou vzorky komplexně sdružené kolem nuly.
- DTFT (Four. transform. s diskr. časem) – FT pro vzorkované (diskr.) signály, spektrum je periodické.
- DFŘ (diskr. Fourier. řada) – Transformuje diskr. period. signál na nekonečné periodické spektrum.
- DFT (diskr. Fourier. transform.) – Jako DFŘ, ale z výsledku vybere pouze jednu periodu, tzn. výsledek je konečný.
- FFT (fast FT) – algoritmus pro rychlý výpočet DFT ($O(n \log n)$ místo $O(n^2)$).

Každá operace má navíc zpětnou verzi (začíná I, např. IDFT).

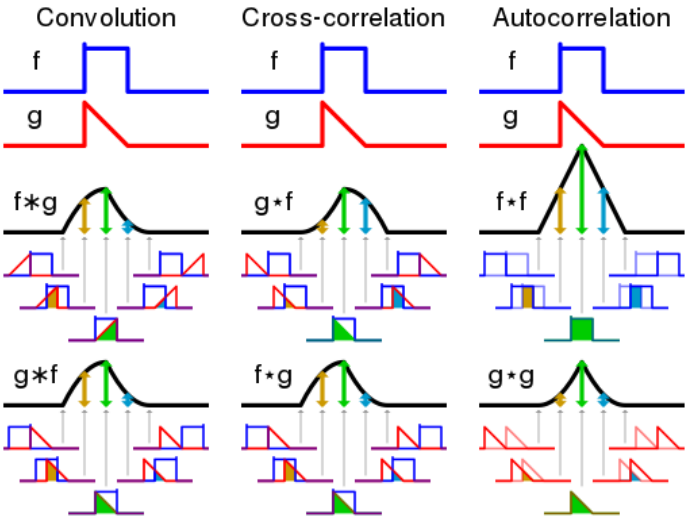


Illustration 120: srovnání konvoluce, korelace a autokorelace

Platí: je-li signál diskrétní, bude spektrum periodické, a naopak.

Rovněž platí **konvoluční teorém**: konvoluce dvou signálů je ekvivalentní násobení spekter těchto signálů. Díky tomu lze provádět tzv. **rychlou konvoluci** (FFT, násobení, IFFT).

Řečové signály typicky zpracováváme po tzv. **rámcích**, neboli částech, na nichž zjednodušeně předpokládáme, že má signál konstantní vlastnosti.

Spektrální hustota výkonu (power spectral density – PSD) říká, kolik výkonu je v různých částech spektra signálu. Běžný se definuje jako absolutní hodnota modulů spektra na druhou

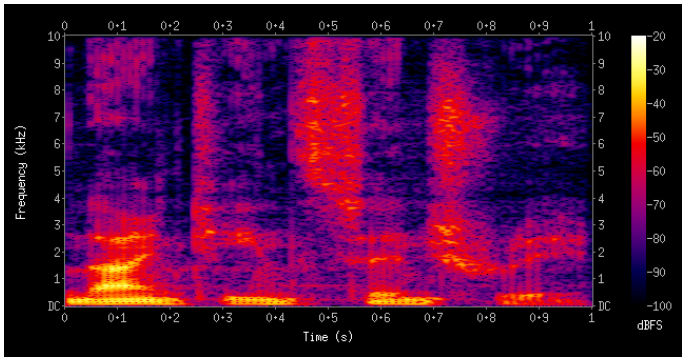


Illustration 121: spektrogram

57. Kódování: waveform (podstata, DPCM), vokodéry, hybridní kodéry (podstata, vektorové kvantování, analýza syntézou, CELP). ZRE

Kódování se zabývá převodem řeči do digitálního formátu tak, aby zabírala co nejméně bitů, byla co nejkvalitnější, odolná proti chybám, aby kódování a dekódování netrvalo příliš dlouho, mělo malé zpoždění apod. Kódování je velmi důležité pro telefonii a je komerčně nejdůležitější součástí zpracování řeči. Kodéry dělíme na:

- **waveform** (**kódování tvaru vlny**) – Obecné kódování zvuku, tzn. ne jenom řeči, např. MP3. Dosahuje vysoké kvality a obecnosti, ale za cenu vysokého datového toku.
- **vokodéry** (**vocoders**) – Jsou založeny na poznatcích o tvorbě řeči lidským hlasovým ústrojím, tzn. kódují pouze parametry řeči (např. hlasitost, nastavení řečového ústrojí apod.), ne samotný zvuk – díky tomu dosahují dobré kvality a malého datového toku, ale nedají se použít pro jiný zvuk.
- **hybridní**, někdy také **CELP** (**code-exited linear prediction**) – Jako vokodéry, ale částečně také kódují waveform (buzení). Momentálně jsou nejpoužívanější.
- **fonetické** – Dělí řeč na delší úseky než rámce, na **fonémy** (vyslovované hlásky). Snaží se natrénovat jejich rozpoznávání a kódování je potom v podstatě fonetickým přepisem řeči. Dekódování zahrnuje syntézu z přepisu. Momentálně existují pouze ve formě experimentů.

U kódování nás zajímá často datový tok, tedy bitrate. Chceme co nejnižší tok při co nejvyšší kvalitě. Za vysoký považujeme > 16 kbit/s, za nízký < 2.4 kbps. Kvalitu kodeku můžeme posuzovat subjektivně (lidským posluchem) nebo objektivně (např. **odstupem signálu od šumu**, tzv. **SNR**, **signal-to-noise-ratio**, vzdálenosti ve vektorovém prostoru apod.).

PCM (**pulse code modulation**) je způsob digitálního kódování vzorkovaného analogového signálu – vzorky jsou od sebe rovnoměrně vzdáleny a naměřená amplituda signálu je kvantizována. LPCM (linear PCM) je PCM, jde je kvantizace lineární (lidské vnímání je ale logaritmické, proto bývá logaritmické měřítko vhodnější). **DPCM** (**differential PCM**) je PCM, při němž kódujeme ne amplitudu signálu, ale rozdíl vzorkované hodnoty a predikované hodnoty – tzn. jedná se o podobný princip jako používá prediktorová komprese (rozdíl je typicky menší než samotná hodnota signálu, tzn. máme kratší kódová slova). Pro DPCM potřebujeme mít tedy nějaký prediktor, který však může být velmi jednoduchý – např. předvídá předchozí vzorek, tzn. kódujeme vždy rozdíl oproti předchozímu vzorku – toto je velmi dobrá metoda kódování obecného audia (waveform), protože to se typicky mění spojitě, ne skokově. Kodér DPCM musí obsahovat celý dekodér, aby odhadoval ze správných vzorků! ADPCM (adaptive DPCM) je DPCM, u něhož se adaptivně mění kvantovací krok DPCM, aby dále snížil SNR.

Vokodéry většinou fungují takto:

1. Pomocí LPC odhadnoue parametry řečového ústrojí, tzn. koeficienty polynomu filtru $A(z)$ – výsledkem je vektor LPC koeficientů. Tyto koeficienty se dále kvantizují, aby se snížil datový tok – používá se tzv. **vektorové kvantování** – vytvoříme tzv. kódovou knihu několika typických vzorků (vektorů), a každý spočtený vektor potom kvantujeme na nejbližší vektor z knihy. Kódovou knihu trénujeme metodami shlukování, např. *K*-means.

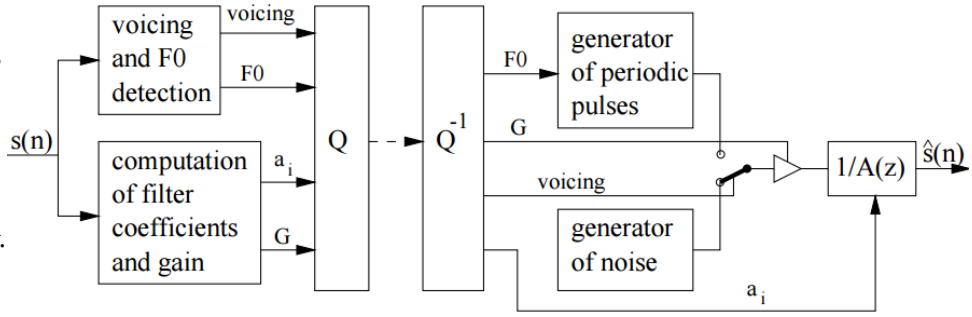


Illustration 122: schéma vokodéru

2. Určí se gain (zesílení, energie) *G* a znělost rámce. Pokud je znělý, určí se dále jeho základní tón (lag).
3. Dále se vypočte chybový signál oproti vstupu, tzv. **residuál**. Tento signál se dále kvantizuje.

Výstupem, který následně jde do dekodéru, je tedy index do kódové knihy LPC vektorů, gain *G*, znělost a residuál. Takovýto vokodér se nazávý RELP (residual-excited linear prediction). Dnes jsou tyto vokodéry nahrazovány CELP vokodéry.

CELP kodeky provádí:

- krátkodobou analýzu signálu pomocí **LP** - linear prediction, hledá predikuje, jak se signál bude vyvíjet, pomocí přímky.
- dlouhodobour analýzu – zjišťování základního tónu, LTP (long term predictor) – ten se dá dělat např. tzv. adaptivní knihovnou pro buzení, která je tvořena úseky minulého excitačního signálu, z nichž se potom pomocí

minimalizace energie chyby vybere ten, který nejlíp pasuje. Kromě adaptivní knihovny je v CELPu ještě i velká fixní knihovna buzení.

Analýza syntézou se snaží odhadnout model prediktoru tak, že mění jeho parametry, použije jej k syntéze řeči, a porovná ho se skutečným vstupem. Tento proces opakuje, dokud neminimalizuje chybu syntetizovaného signálu a skutečného signálu.

ACELP je vylepšení CELPu, které se zbavuje velkých fixních knihoven signálů, které jsou nahrazeny knihovnou pulzů.

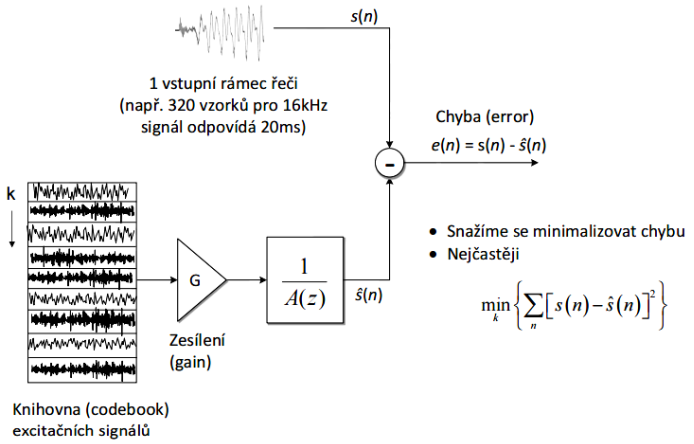


Illustration 123: analýza syntézou

58. Rozpoznávání DTW (variabilita v rozpoznávání řeči, lokální vzdálenost, částečná kumulovaná vzdálenost, DTW cesta). ZRE

DTW (**dynamic time warping**) je metoda porovnání dvou sekvencí, v našem případě slov, které se mohou různě lišit ve své rychlosti – to se nám velmi hodí, protože potřebujeme porovnat vyřčené slovo se všemi slovy ve slovníku, přičemž ale člověk může slovo vyslovit různou rychlostí (tzn. **variabilita řečníka**). Každé slovo je zaznamenáno jako matice parametrů, tj. pro každý rámec slova máme vektor parametrů, např. energii, průchody nulou apod. Vektory parametrů můžeme porovnávat a určovat jejich vzdálenost. Algoritmus DTW je následující:

1. Uděláme mřížku *d* o velikosti $T \times R$. *T* je velikost (počet rámců/vektorů) testovaného slova a *R* je velikost referenčního slova.
2. Do políček mřížky *d* vyplníme vzdálenost každého vektoru s každým.
3. Uděláme mřížku **částečných kumulovaných vzdáleností**, *g*, která má oproti *d* ještě navíc nultý řádek a sloupec.
4. Všechna políčka nultého řádku a sloupce *g* inicializujeme na ∞, políčko [0,0] na 0.
5. Pro všechna ostatní políčka *g* spočítáme částečnou kumulovanou vzdálenost jako: $g(x,y) = \min \forall$ předchůdci $(g(\text{předchůdce}) + \text{váha(cesta k předchůdci)} \cdot d(x,y))$
   Předchůdci a jejich váhy jsou:
   - [x – 1,y] – váha 1
   - [x, y – 1] – váha 1
   - D =[x – 1, y – 1] – váha 2
6. konečná minimální normovaná vzdálenost je potom: $D = g(T,R) / (T + R)$.

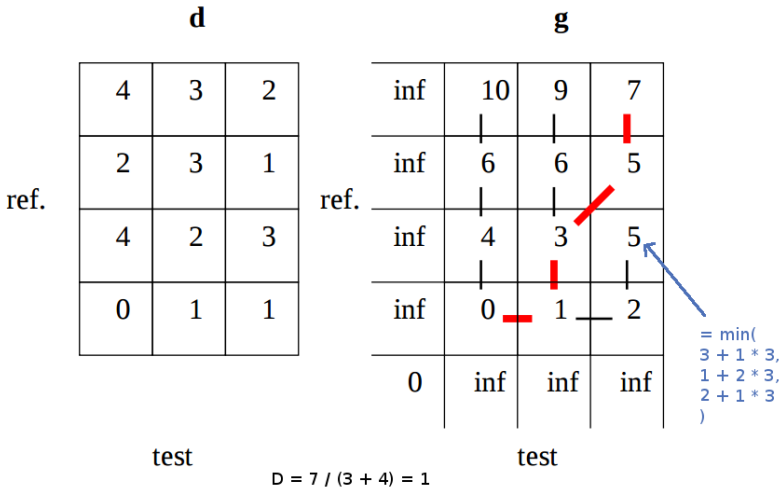


Illustration 124: DTW příklad

Existují i jiné varianty DTW, které se liší tím, jaké předchůdce políčka povolují a jakou váhu přikládají pohybu v mřížce.

59. Rozpoznávání HMM (architektura, přechodová pravděpodobnost, funkce hustoty pravděpodobnosti ve stavech, sekvence stavů, pravděpodobnost promluvy přes sekvenci stavů, Baum-Welch, Viterbi, podstata trénování). ZRE

HMM (**hidden Markov model**) je statistický model slova používaný v rozpoznávání řeči. HMM můžeme použít místo DTW – zatímco u DTW hledáme minimální vzdálenost, u HMM hledáme maximální pravděpodobnost. Pro každé slovo ve slovníku musíme mít jeden HMM model – tomuto modelu lze předložit testované slovo a on nám dá pravděpodobnost, s jakou toto slovo vygeneruje. Díky tomu potom můžeme vybrat model (slovo ve slovníku), který nejpravděpodobněji sedí na testované slovo.

Příklad HMM vidíme na obrázku. Každý stav má tři pravdepodobnosti (kromě posledního, ten má dvě): **setrvání ve stavu** (a_{ii}), **posun do dalšího stavu** (a_{ii+1}) a **přeskočení následujícího stavu** (a_{ii+2}) (podobně jako u DTW byly v tabulce tři různé cesty k předchůdci). Každý stav *i* (kromě prvního a posledního) má dále tzv. **vysílací pravděpodobnost** – pravděpodobnost, že vygeneruje určitý vektor o_t, kterou značíme $b_i(o_t)$ – jelikož je vektor o_t *n*rozměrný, je $b_i(v)$ *n*rozměrná funkce, většinou založená na Gaussovkách. Každému stavu tedy můžeme předložit frame řeči a on nám řekne, s jakou pravděpodobností jej generuje podle funkce pravděpodobnosti, kterou má v sobě uloženou.

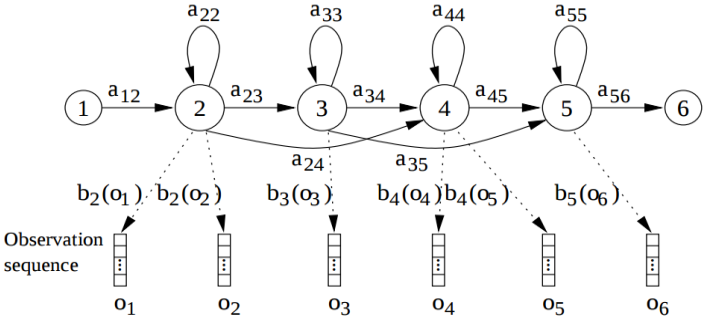


Illustration 125: HMM reprezentace slova

Vysílací pravděpodobnost $b_i(x)$ může být buď opravdová pravděpodobnost (integrál), nebo (v našem případě) hodnota funkce hustoty pravděpodobnosti (což není to samé, může být větší než 1 atd.), čemuž potom říkáme **likelihood**.

Tato funkce (hustota pravděpodobnosti generování vektoru o_t stavem *i*) je realizována pomocí tzv. **směsi Gaussovek** (**GMM** – **Gaussian mixture model**), jelikož je vektor *n*rozměrný (typicky 39 = počet vzorků řečového framu). Jedná se o součet *n* Gaussovek, pro každý vzorek vektoru je jedna. Gaussovka je v 1D definována střední hodnotou a rozptylem, ve vyšších rozměrech pak tzv. **kovarianční maticí**.

Mějme vstupní slovo *O*. Potom můžeme jít různými cestami v HMM modelu a vždycky dostaneme nějakou pravděpodobnost, že jsme vygenerovali *O*. Můžeme mít např. následující cestu *X* = [1, 2, 2, 3, 4, 4, 5, 6] (viz obr.).

Pravděpodobnost generování vektoru *O* po cestě *X* je definována jako:

$$P(O,X)=a_{X(0)X(1)}\prod_{t=1}^{T} b_{X(t)}(O(t))\,a_{X(t)X(t+1)}$$

(První pravděpodobnost, $a_{X(0)X(1)}$, je pravděpodobnost, že se do modelu vůbec vstoupí, tzn. pravděpodobnost celého slova). To je však jenom pravděpodobnost po jedné cestě – abychom získali pravděpodobnost pro dané slovo, musíme projít všechny cesty (moderní algoritmy umí tento počet snížit) a celková pravděpodobnost se potom spočítá jednou ze dvou metod:

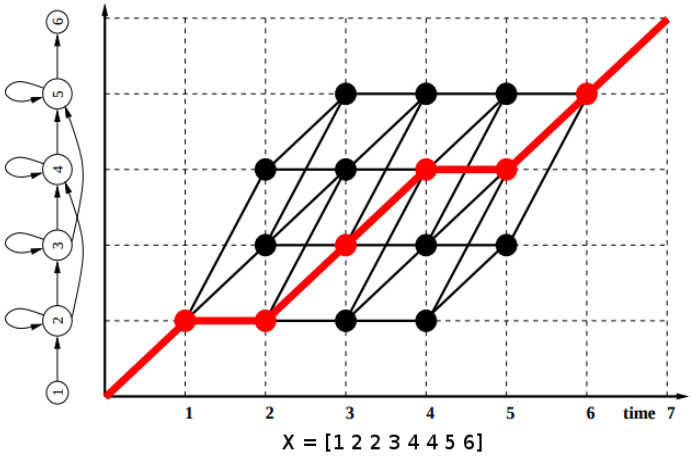


Illustration 126: možná cesta v HMM

- **Baum-Welch** – Součet všech pravděpodobností (používá se např. pro normalizaci), tzn.
  $$P(O)=\sum_X P(O,X)$$
- **Viterbi** – Největší pravděpodobnost, tzn.
  $$P(O)=max_X(P(O,X))$$

Snížit počet testovaných cest lze pomocí tzv. **token passing** (piva). Ten pracuje s log pravděpodobnostmi (abychom místo násobení jenom sčítali). Dále jakoby prochází několik stavů paralelně a někdy dokáže odvodit, že některou cestu dál nemá cenu odhadnovat.

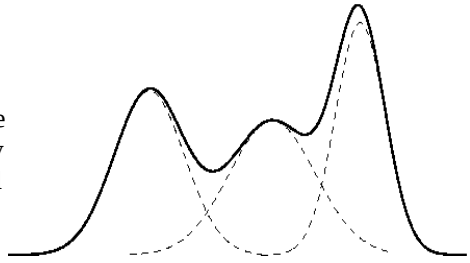


Illustration 127: GMM - 3 Gaussovky

HMM model vytváříme **trénováním** z databáze nahraných slov – pokud chceme např. rozpoznávat 3 slova, nahrajeme každé z nich 20 krát a z každé této sady o 20 nahrávkách dostaneme trénováním model pro jedno slovo, celkem tedy 3 modely. Princip trénování pomocí tzv. **EM** (**expectiation maximization**, také Bauch-Welch, hledání lokálního maxima) je následující:

1. Zhruba odhadneme parametry modelu (vysílací a přechodové pravděpodobnosti).
2. Vekory trénovaného slova přiřadíme stavům. Toto není jednoduché udělat a slouží k tomu tzv. **state occupation function**.
3. Spustíme model pro trénované slovo a podle pravděpodobnosti upravíme parametry modelu, aby se pravděpodobnost jeho generování zvýšila.
4. Pokud se už pravděpodobnost generovaného slova modelem nezvyšuje, skončíme, jinak provádíme znovu bod 3.

Tento algoritmus se snaží maximalizovat celkovou očekávanou likelihood.

60. Standardy pro rychlé vykreslování na GPU (OpenGL, Direct3D, Vulkan) - základní charakteristiky, srovnání, důležité verze. GZN PGR

OpenGL (**Open Graphics Library**) je standardizované, nízkoúrovňové API pro akcelerované vykreslování 2D a 3D grafiky. OpenGL se specializuje výhradně na vykreslování, nenabízí práci s událostmi a okny, proto pro něj existuje spousta pomocných knihoven, mezi které patří **GLM** (GL Math - matematická knihovna s podobným rozhraním jako jazyk GLSL), **GLUT** (GL Utility Toolkit – platformě nezávislá práce s I/O a okny) nebo **GLEW** (GL Extension Wrangler Library – práce s OpenGL rozšířeními).

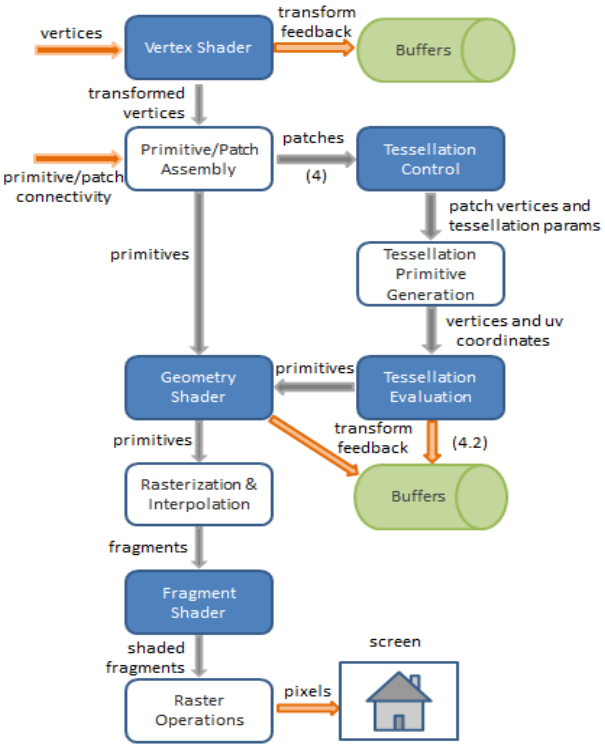


Illustration 128: OpenGL pipeline

Základem OpenGL je **vykreslovací řetězec** (**rendering pipeline**), což je cesta, kterou vykreslovaná data procházejí. Tento řetězec byl v počátcích fixní a v OpenGL se pracovalo se scénou která se sama vykreslovala (tzv. **retained mode**). Od verze 2.0 se některé části (vertex, fragment) pipeline staly programovatelnými pomocí nově vytvořeného jazyka **GLSL** (**GL Shading Language**). Programy napsané pro GPU se nazývají **shadery** (**shaders**). Dodnes je snaha dělat pipeline více a více programovatelnou. Současná pipeline (OpenGL 4.5) se skládá z následujících částí (podtržené jsou programovatelné):

- **vertex shader** – Zpracovává vstupní vrcholy. Shader má informaci pouze o aktuálním vrcholu a výstupem je vždy jeden transformovaný vrchol. Tento shader se používá pro realizaci 3D promítání, deformaci objektů apod.
- **tessellation** – Umožňuje dělit primitiva na podprimitiva, což je užitečné např. pro dynamický LOD, vytváření složitějších modelů z jednodušších (např. na základě displacement masky) apod.
  - **control** – Určuje úroveň teselace, tzn. jak moc se primitivum bude dělit.
  - **primitive generation** – Generuje nová primitiva ze vstupního primitiva.
  - **evaluation** – Upravuje výstupní primitivum, tzn. nově vzniklé vrcholy.
- **geometry shader** – Pracuje s celými primitivy, tzn. body, úsečky, trojúhelníky apod. Vstupem je primitivum a výstupem může být jiné nebo žádné primitivum – tzn. na rozdíl od fragment shaderu lze pracovat s vrcholy v kontextu ostatních vrcholů, vytvářet nové nebo je zahazovat. Toho se dá využít např. pro částicové systémy (generování quadů z bodů), voxelizaci, billboardy apod.
- **rasterization** – Provádí rasterizaci primitiv.
- **fragment shader** – Pracuje s jednotlivými **fragmenty** (pixely, které mohou potenciálně nakonec zobrazeny). Výstupem je barva shaderu. Fragment shader nemá informaci o sousedních či jiných fragmentech.

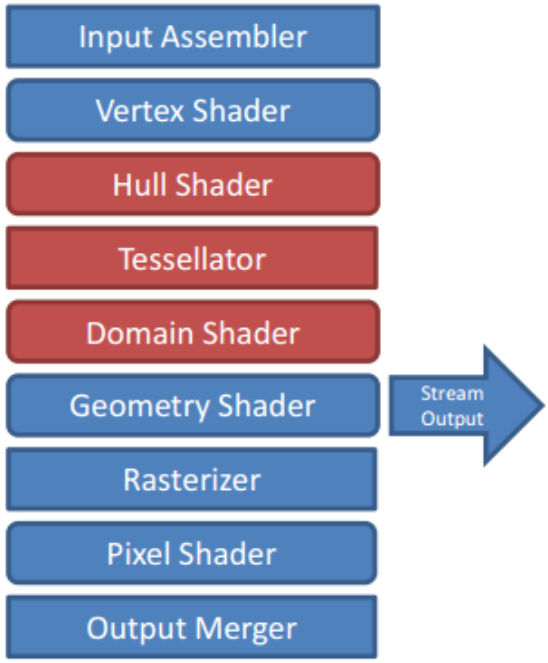


Illustration 129: Direct3D 11/12 pipeline

OpenGL jako standard je udržováno konsorciem **ARB** (Architecture Review Board), což je sdružení významných firem v oblasti počítačové grafiky. Funkcionalita do OpenGL přibývá tak, že různí výrobci přidávají do svých karet specifická **rozšíření** (**extensions**) a pokud se tato rozšíření ujmou, jsou v dalších verzích přidána do standardu OpenGL. Každé rozšíření je identifikováno řetězcem, který začíná prefixem (EXT_ - více výrobců, ARB_ - schváleno konsorciem, NV_ - Nvidia, ...), např. GL_ARB_vertex_array_object.

OpenGL API je koncipováno jako stavový stroj (funkce pro nastavení stavu předchází vykreslovacímu příkazu) z důvodu, aby se minimalizoval provoz na sběrnici mezi CPU a GPU. Od této filozofie se nicméně v poslední době upouští a zavádějí se nové "nestavové" funkce.

Na základě OpenGL vznikly některé další, podobné technologie. Patří sem např. **OpenGL ES** (**OpenGL for Embedded Systems**), což je podmnožina OpenGL pro vestavěné systémy – používá se např. v mobilních telefonech s Androidem. Na OpenGL ES 2.0 je založena technologie **WebGL**, což je API pro vykreslování 2D a 3D grafiky ve webových prohlížečích. Je nutné mít na paměti, že jelikož je WebGL založeno na OpenGL ES 2.0, nabízí jenom minimum toho, co moderní verze OpenGL (nabízí např. jenom vertex a fragment shadery).

Direct3D je API pro akcelerované vykreslování 2D a 3D grafiky, vyvinuté firmou Microsoft pro Windows a podobné platformy. Direct3D je součástí sady knihoven DirectX, konkrétně jeho části DirectXGraphics, kam patří např. ještě Direct2D, DirectDraw (pro zpětnou kompatibilitu), DXGI (API pro zjišťování informací o GPU, monitorech apod.) a DirectWrite (vykreslováné textu). Direct3D se v mnohém podobá OpenGL. Mezi rozdíly oproti OpenGL patří:

- Direct3D používá jazyk **HLSL** (High Level Shading Language), ne GLSL.
- Direct3D narozdíl od OpenGL používá levotočivý souřadnicový systém (x doprava, y nahoru, z dopředu).
- Je to proprietální API od Microsoftu, liší se politika jeho vývoje.
- Liší se některá terminologie, např. místro fragment shader je pixel shader apod.
- Liší se politika vývoje, neudržuje se příliš zpětná kompatibilita.

Důležitými verzemi Direct 3D jsou:

- 2.0 – Umožňovalo **retained** (scene graph) a **immediate** (vykreslovací příkazy) mode.
- 8.0 – Představilo programovatelný vertex a pixel shader.
- 10.0 – Přidalo geometry shader a plnou integraci HLSL.
- 11.0 – Přidalo tessellation shader, compute shader (obecné výpočty na GPU, např. herní fyzika, AI apod.) a **dynamic shader linking** (sestavování shaderu "na míru" aplikaci z menších částí).
- 12.0 – Pdopora telefonů a tabletů.

V roce 2016 bylo Khronos skupinou zveřejněno API nové generace – **Vulkan**. Mělo být nástupcem OpenGL a napravovat jeho nevýhody. Mezi klíčové vlastnosti Vulkanu patří:

- Menší overhead ovladačů nez OpenGL – v ovladačích je nyní jen minimum funkcionality, která se přesouvá do samotné aplikace, dává tak větší možnost kontroly a potenciál k většímu výkonu, avšak důsledkem je vyšší složitost aplikací a tedy jejich náročnější programování.
- Lepší práce s vícejádrovými CPU. OpenGL a Direct3D byly původně navrženy pro jednojádrové procesory a až později rozšířeny pro vícejádrové a i tak jsou na nich špatně škálovatelné.
- OpenGL při každém běhu znovu kompiluje shadery z jazyka GLSL, Vulkan má shadery již předkompilované (v jazyce SPIR-V) a aplikace se tedy rychleji načítá.
- Oproti OpenGL není koncipován jako stavový stroj, ale je objektově orientovaný, bez globálního stavu.

61. Důležité knihovny pro práci s grafem 3D scény. GZN

Graf scény (**scene graph**) je datová struktura pro deklarativní reprezentaci 3D grafiky – tzn., že do grafu vkládáme objekty a ty se potom vykreslují (nevoláme přímo vykreslovací příkazy, což je imperativní přístup). Scene graph bývá **orientovaný acyklický graf** (**DAG**, **directed acyclic graph**). Listy grafu jsou vykreslitelné objekty, ostatní uzly jsou transformace, skupiny apod. Uzly dědí atributy od svých rodičů a jejich transformace jsou relativní vůči nim, což umožňuje jednoduše připojovat objekty k jiným. Mezi důležité knihovny pro práci s grafem scény patří:

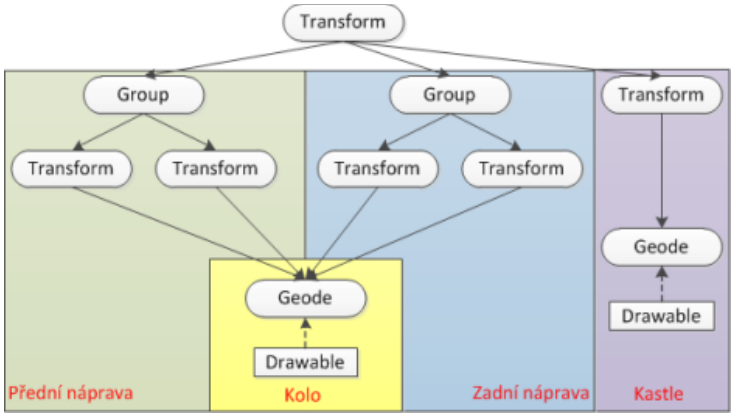


Illustration 130: scene graph auta

- **OpenSceneGraph** (**OSG**) – Open source, multiplatformní C++ zobrazovací toolkit postavený na OpenGL. Řeší jenom vizualizaci, nic jiného. Podporuje QT, spoustu 2D a 3D formátů apod. Vše, co jde udělat v OpenGL, jde teoreticky udělat i v OSG.
  Důležitým pojmem je **OpenGL state** – jedná se o stav pro volání OpenGL funkcí a skládá se z atributů (např. depth test, cull face, shader program, uniform, texture, ...). Atributy jsou sdruženy do **state setů** (tzn. state set pro uzel skupiny může obsahovat např. nastavení depth testu, cull face apod.), které jsou umístěny v uzlech. OpenGL state se v uzlech dědí od rodičů (dědičnost lze řídit ručně pomocí klíčových slov OVERRIDE a PROTECTED).
  Hlavní třídou OSG je **Viewer**, který se stará o vytvoření okna, inicializaci OpenGL a běh vykreslovací smyčky. OSG používá počítání referencí pro správu zdrojů (textur, modelů, ...).
  Vykreslovací smyčka má několik fází:
  - **event traversal** – Zpracovává se I/O a jiné události.
  - **update traversal** – Prochází se graf a provádí se update callbacky (např. Animace, LOD, ...).
  - **cull traversal** – Provádí se culling (ořezávání neviditelných objektů, např. frustum culling) a příprava pro vykreslování (vytváří se vykreslovací graf, který je odlišný od uživatelského grafu). Ve výchozím nastavení se objekty řadí pro vykreslování podle OpenGL statů, aby docházelo k co nejmenšímu počtu změn stavů (např. shaderů), protože ty jsou časově náročné.
  - **draw traversal** – Prochází se vykreslovací graf a vykresluje se.

  Někdy je důležité řídit pořadí vykreslování objektů (např. pro průhlednost, deferred shading, optimalizaci, ...). To lze provést pomocí tzv. **render binů** – ty jsou součástí StateSetu. Biny lze řadit pomocí jejich čísla, objekty uvnitř binu jsou už řazeny podle toho, jak OSG uzná za vhodné. Biny samy o sobě nestačí a pořadí lze dále jemněji řídit pomocí render stage.
  OSG je rozděleno do různých namespaces, mezi něž patří osg (core funkcionality), osgAnimation (obecné a kosterní animace), osgDB (souborový systém, DB), osgFX (speciální efekty), osgShadow (stíny), osgTerrain (terény), osgText(fonty, text), ...
- **Open Inventor** – Open source, multiplatformní C++ zobrazovací toolkit postavený na OpenGL. Od OSG se liší tím, že vznikl dříve a byl průkopníkem tohoto typu softwaru. Liší se v detailech, jako např. v rozdílném souřadnicovém systému nebo API. Dnes je Open Inventor již nahrazován OpenSceneGraphem, který využívá moderní návrhové vzory a staví na základech, které Open Inventor položil. Vývoj Open Inventoru je spojen s vývojem formátu VRML pro ukládání 3D scény.
- **Coin3D** – Open source, multiplatformní C++ zobrazovací toolkit postavený na OpenGL. Jedná se implementaci Open Inevntor API, tzn je s ním plně kompatibilní. Od Open Inventoru se liší tím, že je od začátku open source, kdežto Open Inventor byl ze začátku proprietální.
- Další herní a zobrazovací enginy – např. Irrlicht, Panda3D, OGRE, Unity, …

Na FITu se vyvíjí GPUEngine - open-source grafický engine.

62. Standardy ukládání obrazů, 3D objektů a scén - rozdělení podle účelu, důležití zástupci, moderní trendy. GZN

Mezi důležité 2D rastrové obrazové formáty patří:

- **JPEG** – Formát určený pro fotografie, umožňuje ztrátovou a bezztrátovou kompresi. Ztrátová komprese funguje na principu kvantování koeficientů DCT.
- **JPEG2000** – Vylepšení JPEG fotmáru, využívá DWT místo DCT, nabízí lepší kompresi, nedělí obraz na bloky, čímž redukuje artefakty, lépe se zotavuje z chyb, nabízí ROI (region of interest, v určité oblasti zájmu se obraz komprimuje méně a je kvalitnější).
- **GIF** (**Graphics Interchange Format**) – Starší formát, popisuje tzv. logický prostor, do něhož lze umisťovat více rastrů, přičemž každý rastr může mít paletu maximálně 256 barev. Kombinací více rastrů lze dosáhnout až na true-color kvalitu. Rastrům lze taky přiřazovat spoždění, se kterým se zobrazí, takže lze vytvářet animce. GIF podporuje binární průhlednost. Používá bezztrátovou kompresi LZW.
- **PNG** – Měl nahradit GIF, ale nestalo se tak úplně. Podporuje implicitně true-color (24 bitová RGB barevná hloubka) a spojitou průhlednost pomocí alfa kanálu. Může být ukládán v několika formátech (grayscale, 24bit RGB. 32bit RGBA). Soubor je rozdělen do tzv. chunků. PNG používá bezztrátovou kompresi Deflate.
- **BMP** – Bezztrátový formát používaný především v OS Windows.
- **PPM** – Zřejmě nejjednodušší rastrový formát, je textový a nepoužívá žádnou kompresi, proto je velmi jednoduše parsovatelný.

Mezi důležité 2D vektorové formáty patří:

- **SVG** (**Scalable Vector Graphics**) – Otevřený XML formát standardizovaný W3C konsorciem, jedná se o nejpoužívanější formát pro vektorovou grafiku na webu. Podporuje text, animace, filtry, křivky, skriptování, interaktivitu, metadata atd.
- **PS** (**PostScript**) – Jedná se o Turingovsky-úplný programovací jazyk s postfixovou notací, jehož výstupem je vektorová grafika. PS byl navržen v roce 1982 především pro flexibilní popis stránek pro tisk.
- **EPS** (**Encapsulated PostScript**) – Omezená verze PostScriptu, která splňuje určitá daná kritéria, aby jej bylo možné použít jako obecný vektorový formát při snadnějším parsování. EPS např. musí obsahovat rastrový náhled v malém rozlišení.
- **PDF** (**Portable Document Format**) – Formát pro popis dokumentů, na rozdíl od PS je schopný definovat nejenom vzhled, ale i chování (např. co se stane po kliknutí na určitý text). PDF je textový formát, ale umožňuje v sobě ukládat binární data - obsahuje vše, co je nutné k jeho zobrazení, tzn. např. fonty, obrázky apod. PDF soubor se skládá z objektů. PDF se skládá z hlavičky (obsahuje pouze verzi PDF), těla, tabulky odkazů na objekty a tzv. trailer sekce (obsahuje další metadata).
- **SWF** (Small Web Format) – Formát od Adobe používaný pro multimedia, ActionScript a vektorovou grafiku. Může obsahovat animace i interaktivitu (programovanou pomocí ActionScriptu). Jedná se o binární formát, aplikace mohou přeskakovat data, která nepodporují.

Mezi důležité 3D formáty patří:

- **OBJ** – Od firmy Wavefront Technologies, víceméně uznávaný jako obecný, univerzální formát. Je otevřený, jednoduchý (na parsování, vytváření, i čtení člověkem), textový (ne XML), umožňuje ukládání i netrojúhelníkových stěn, indexované vrcholy, normály, texturovací souřadnice, materiály, sdružování do objektů a další. Mezi nevýhody patří absence podpory animací (ty se občas ukládají jako sekvence více OBJ souborů).
- **VRML** (**Virtual Reality Modeling Language**) – Vzešel z Open Inventoru, v současnosti jej nahrazuje X3D. Je textový, podporuje texturovací souřadnice, barevné vrcholy, animace, skriptování pomocí JavaScriptu atd. VRML soubory se nazývají "worlds" (.wrl) a ukládají scénu jako scene graph. Pro lepší načítání z webu se někdy komprimuje pomocí gzipu (přípona je potom .wrz).
- **X3D** – ISO standard, nahrazuje VRML. Umožňuje kódování pomocí XML (.x3d), binární reprezentace (.x3db) nebo starého VRML textového formátu (.x3dv).
- **STL** (STereoLithography) – Nativní formát CAD programů od 3D Systems. Ukládá pouze povrch objektu bez textur, barev a materiálu. Může být kódován textově nebo binárně (častější, protože je kompaktnější). STL 2.0 se nazývá AMF a využívá XML.
- **PLY** (**Polygon File Format**) – Formát původně určený pro ukládání dat ze 3D scannerů, ukládá data jako posloupnost polygonů, přičemž podporuje i barvu, normály, průhlednost, texturovací souřadnice apod. Může být kódován buď pomocí ASCII nebo binárně.
- **AMF** (Additive Manufacturing File Format) – Otevřený XML ISO standard určený pro 3D tisk. Je to nástupce formátu STL a podporuje navíc barvu, materiály apod. Popisuje jeden nebo více objektů jakožto objem prostoru (ne pouze povrch), popsaný pomocí vrcholů.
- **3DS** – Uzavřený binární formát od Autodesku, nativní pro 3DS Max. Ukládá celé scény jako scene graph, včetně animací a materiálů. Má omezený počet polygonů na objekt na 65536 a nepodporuje křivky a spline plochy.

- **DXF** – Formát od Autodesku pro AutoCAD. Je kódován textově ve formátu podobném XML, nové verze umožňují i binární reprezentaci.
- **COLLADA** (Collaborative Design Activity) – Otevřený ISO standard, XML. Přípona je většinou .dae. Podporuje většinu moderních věcí, které 3D grafik potřebuje.
- **x** – Soubory s příponou .x představil DirectX 2.0, je to jednoduchý formát pro ukládání 3D geometrie, dnes už zastaralý.
- **ICS** – volumetrický formát
- blend – Používáno Blenderem, ukládá nejenom 3D data, ale veškeré nastavení aplikace atd.

Rozdělení dle účelu:
- průmyslové: IGES, DXF, ...
- herní: X, VRML, X3D, 3DS, ...
- volumetrické: ICS

63. Standardy a knihovny ve zpracování videa - standardy kódování, důležité knihovny a nástroje. MUL GZN

Multimediální framework je souhrn API, knihoven, přehrávačů, formátů apod., pro práci s multimédii, tzn. videem, zvukem, obrazem apod.

Mezi standardy kódování, tzn. formáty videa patří:

- **MPEG-2 Part 2** – Je vylepšní dřívějšího MPEG-1 standardu, podporuje prokládané video, není optimalizován pro malé datové toky.
- **MPEG-4 Part 2** – Je optimalizován pro malé datové toky.
- **H.264** – Jeden z nejpoužívanějších standardů, využívá se např. pro Blu-ray nebo streaming videa na internetu (např. YouTube nebo Vimeo).
- **VP8** – Formát vlastněný Googlem, je podporován všemi hlavními internetovými prohlížeči.
- **VP9** – Otevřený formát vyvinutý Googlem, jedná se o nástupce VP8. Oproti VP8 podporuje např. bloky o velikosti 64 x 64 nebo optimalizaci pomocí quadtrees.

Kodek je program, který provádí kódování nebo dekódování videa, tzn implementuje určitý standard kódování. Mezi důležité kodeky patří:

- **DivX** – Proprietární kodek, podporuje kódování videa do různých formátů MPEG-4 standardu.
- **Xvid** – Open-source varianta DivX kodeku, rovněž implemetuje standard MPEG-4.
- **FFmpeg** – Open-source, podporuje spoustu spoustu formátů včetně MPEG-2, MPEG-4, H.264, WMV apod., i když u některých např. jenom dekódování.

Důležité jsou také formáty **multimediálích kontejnerů**. Ty umožňují do jednoho souboru uložit video, zvuk, metadata, titulky apod. Mezi důležité kontejnerové formáty patří:

- **AVI** – Proprietární kontejner od Microsoftu, podporuje téměř všechny formáty videa a audia, nepodporuje nativně titulky, avšak modifikacemi se jich dá dosáhnout. Nepodporuje streaming.
- **MKV** (**Matroska**) – Otevřený standard, jeho implementace jsou většinou open-source. Podporuje streaming, titulky, kapitoly a prakticky všechny formáty videa a audia.
- **MP4** – Patentovaný formát, podporuje MPEG standardy videa a některé další, umožňuje streaming.
- **WebM** – Vychází z MKV, podporuje VP8 a VP9 video formáty.
- 3GP – Určený pro mobilní zařízení.
- OGG – Formát, ke kterému existují open-source implementace a podporuje spoustu video a audio formátů.

Důležitými nástroji pro zpracování videa jsou:

- **FFmpeg** – Open-source sada nástrojů pro práci s multimédii (multimediální framework). Zahrnuje kodeky, které jsou využívány mnoha dalšími projekty, API a nástroje pro příkazovou řádku, např. pro konverzi videí, aplikaci filtrů apod.
- **Libav** – Jedná se o fork FFmpegu, který vznikl hlavně z důvodů neshody mezi vývojáři.
- **MediaCoder** – GUI nástroj pro konverzi videí, je postavená nad open-source kodeky. Dříve byl open-source, ale nyní vyžaduje pro některé funkce platbu.
- **VLC** – Open-source multiplatformní multimediální framework. Zahrnujíce přehrávač, API, streamovací server, je velmi modulární a lze jej použít jako nástroj pro širší práci s videem.
- **GStreamer** – Open-source multimediální framework napsaný v C.

64. MIDI a rozhraní pro profesionální práci se zvukem v reálném čase. GZN

MIDI (**Musical Instrument Digital Interface**) je standard popisující protokol, rozhraní a konektory pro komunikaci hudebních událostí (např. stisknutí/uvolnění klávesy na pianu). MIDI pracuje pouze s událostmi, nikoliv se zvukovým signálem. MIDI se dá použít pro záznam hudby, její přenos, syntézu, generování not apod.

MIDI definuje tři 5pinové konektory: **IN** (vstup), **OUT** (výstup) a **THRU** (kopíruje se na něj beze změny to, co jde do IN, umožňuje sestavovat složitější konfigurace). Kromě tohoto rozhraní se však používají i jiná, běžně např. USB, FireWire, Ethernet apod.

Základním prvkem MIDI protokolu je zpráva. Ta se skládá z:

- Jednoho **stavového bytu** – MSB = 1, dále se dělí na:
  - **kanálové** – 3 bity určují typ zprávy, zbylé 4 kanál. Sem patří zprávy týkající se specifických nástrojů.
  - **systémové** – všech 7 zbývajících bitů je určeno pro typ zprávy, ale využívají se jenom poslední 4. Sem patří např. synchronizační zprávy, reset systému apod.
- Několika **datových bytů** – MSB = 0, zbytek bitů nese data.

Zprávy se posílají rychlostí 31,25 kbit/s. Příkladem MIDI zprávy může být stisknutí klávesy: Pošle se stavový byte (MSB = 1) určující typ zprávy "stisknutí klávesy" a daný kanál a následují dva datové byty **–** kód klávesy (výška tónu) a velocity (síla stisku). Dalšími zprávami jsou např. uvolnění klávesy, změna kontroleru, změna programu, ohýbání tónu apod..
Pro zvýšení propustnosti není nutné opakovaně vysílat stavový byte, pokud se typ zprávy nemění **–** stačí vysílat jenom datové byty.

MIDI kontroler je zařízení, které ovládá MIDI, např. MIDI klávesy nebo táhlo v softwarovém GUI. Taková zařízení lze ovládat pomocí MIDI zpráv tak, že se jim pošle zpráva s číslem parametru (2 stavové byty => 14 bitů => 16384 různých parametrů), který se má změnit, a hodnotou tohoto parametru. Čísla parametrů se dělí na:

- **RPN** (**registered parameter numbers**) – Čísla parametrů daná MIDI standardem.
- **NRPN** (**non-registered parameter numbers**) – Čísla parametrů, které si můžou určovat výrobci.

MIDI zařízení dělíme na:
- **vysílače** – klaviatury, bicí, dechové, ...
- **přijímače** – syntezátory
- **záznamníky** – sequencery, dělíme na:
  - **klasické** – Zaznamenávají MIDI události jako dvojice čas:událost.
  - **krokové** – Zaznamenávají MIDI v poli v paměti, tzn. v buňkách. Typicky sem patří systémy pro programování bicích, kde máme krátký úsek, který se přehrává pořád dokola.
- **Kombinovaná**

Standardní MIDI soubor (SMF) je kontejnerový formát souboru pro záznam MIDI dat. Má hlavičku, ve které se určí metadata, např. základní časová jednotka, a datovou část, ve které jsou zapsány MIDI události (v jedné nebo několika stopách) spolu s jejich časy (ty jsou vždy relativní oproti předchozí události, v daných časových jednotkách). Soubor může obsahovat text skladby jako metadata, např. pro karaoke.

General MIDI (**GM**) je standard, který zpřísňuje klasický MIDI standard – požaduje např. minimálně 24 hlasů, bicí na kanálu 10, 128 zvukových rejstříků, definuje mapování bicích na klaviaturu. Existují další rozšíření, např. od Rolandu.

Nevýhodou MIDI je malý rozsah některých parametrů (7 bitů) a nízká přenosová rychlost, pokud máme více kanálů.

Zpracování zvuku v real-time je důležité např. pro hudebníky, kteří potřebují velmi nízkou latenci.

Na Windows se dá ke zvuku přistupovat pomocí

- WIN API
- DirectX – Specificky DirectSound, je založený na grafu filtrů.
- **ASIO** (Audio Streaming Input/Output) – Rozhraní pro profesionální práci se zvukem, umožňuje přímý přístup ke kartě, nabízí malou latenci.
- **VST** (Virtual Studio Technology) – Simuluje klasický zvukový HW, nabízí syntezátory, editory, efekty apod. Umožňuje i offline zpracování. Nabízí možnost změny parametrů pomocí univerzálního GUI.

Na Linuxu jsou zvuk. zařízení mapována do /dev/dspx, jakožto API zde existují:

- **OSS** (Open Sound System) - Placený, ale standard pro Unix, standardní Unixová volání open, read, write, …
- **ALSA** (Advanced Linux Sound Architecture) - emuluje OSS a přidává něco navíc, standard pro Linux.

Na vyšší úrovni potom eistují multiplatformní rozhraní:

- **OpenAL** - jako OpenGL, ale pro zvuk
- **SDL** (Simple Direct Media Layer) - Jakoby multiplatformní DirectX, tzn. nejen pro zvuk.

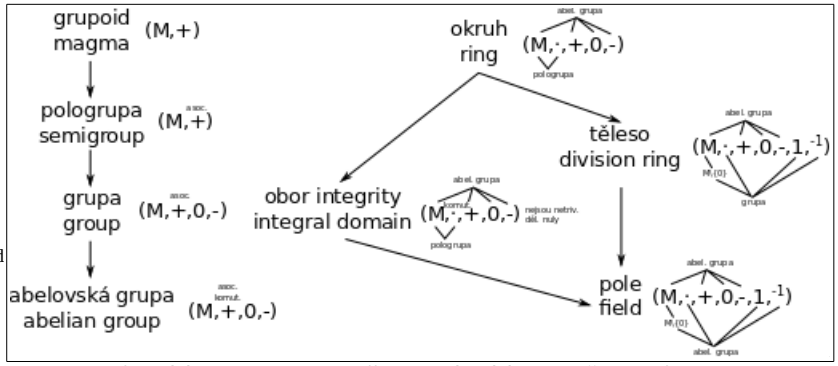
Pro MacOs existuje např. QuickTime, VST, Core Audio apod.

Zvuková karta používá pro přehrávání většinou **kruhový buffer**. CPU do něj musí periodicky nahrávat data pro přehrávání, tzn. čím menší buffer, tím menší latence pro hraní zvuku (důležité např. pro hudebníky), ale zároveň vyšší nároky na CPU. V souvislosti s buffery mohou nastat dvě nežádoucí situace (obecně XRUN):

- **underrun** – Pokud CPU nestihne dodat data, není z čeho hrát.
- **overrun** – Podobná situace, ale u nahrávání – CPU nestihne nahraná data z bufferu stáhnout a ta se přepíšou novými.

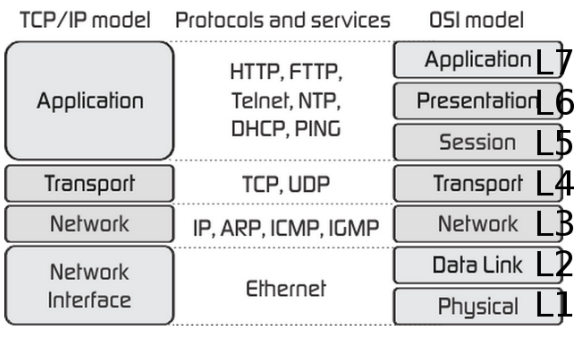
Bonus

pojmy:



Abelovská grupa
access control list
adaptivní hashování
adaptivní k-D tree
adaptivní metody návrhu
adaptivní prahování
adresář (DB)
ACID
ADIP
ADT
afinní kombinace
afinní transformace
Airyho obrazec
Airyho disk
aktivní kontury
ALAP
ALSA
ambientní osvětlení
Amdhalův zákon
analýza syntézou
anizotropní filtrování
alokace (HW/SW cod.)
and stav
argument (imag. číslo)
ASAP
ASIC
ASIO
ASIP
asociativita
asociované prvky
asymptotická složitost
asymptotická omezení
atomická formule
autokorelace
AVI
axiom
backplane
Banachův princ. p. b.
barvení grafu.
Baum-Welch metoda
Báze
BER
bezkontextová gramatika
bezpečnostní kód
Bin tree
bitemporální DB
bitová chyba
bílý šum
blank
Blinn-Phongovo stín.
blokový kód
BMP
bodová transformace obr.
Boolova algebra
Braggova podmínka
Brewsterův úhel
BSP tree
bucket (DB)
buffer (OpenGL)
buzení
B-frame
B-strom
Catapult C
Cauchyovská posl.
CCD
CDFG
CELP
centrální klipování
cepstrum
cesta
charakteristické pole
Chomského hierarchie
Chomského norm. forma
chromatické číslo grafu
chronon
Church-Turingova teze
CISC
CLB
CLOB
CLOS
CMOS
codesign KA
Coin3D
COLLADA
compute shader
controlflow
control plane
CRC
cross-korelace
cyklický kód
c-step
čas platnosti
čas transakce
časová složitost
částečně rozhod. problém
částečné uspořádání
DAG
data plane
dataflow
datalog
DCT
deduktivní DB
Deflate algoritmus
derivace (gramatika)
derivační strom
deskriptorů (prost. DB)
detekce
deterministický KA
determ. bezkont. Jazyk
deterministický ZA
dělitel prvku
DFT
DFX
diagonalizace
diferenciace
difference of Gaussians
difrakce
difůzní osvětlení
Dijkstrův alg. (min. c.)
dimenze
Direct3D
diskrétní metric. Prostor
displacement mapping
distributivita
DivX
DML
dokazatelnost formule
DPCM
DSP
DSpace
Dtime
DTW
duplikace paketu
důkaz
dynamic shader linking
dynamický příkon
D-frame
d-simplex
EFSM
ekvalizace histogramu
ekvivalence
eliptická polarizace
entropie
EPS
Eulerovský graf
Eulerův vzorec
Euklidovská funkce
Euklidovský okruh
Euklidovský prostor
EXCELL
expectiat. maximization
extension (OpenGL)
extrapolace
Fabryův-Perotův interf.
faktor aktivity
faktor grafu
faktorová algebra
fast path (směr.)
fáze vlny
fázor
FDS
FFmpeg
FFT
filtrování textur
FIR filtr
flat shading
flattening
Floyd-Warshallův alg.
Fonetický kodér
foném
formant
formání metoda (sítě)
formule
forwarding engine
Fourierova řada
FPGA
fragment shader
fraktální šum
frame buffer
fráze větné formy
frekvenční maskování
frekvenční transformace
funkce projekce (TIN)
FSM
GAL
Galoisovo pole
Ganntův diagram
Gaussův okruh
Gaussův šum
General MIDI
generalizace
generický dotaz (DB)
generující matice
geometry shader
GIF
GLEW
GLM
GLSL
globální osvětlení
globální transform. obr.
GLUT
GMM
go back N
Goraudovo stínování
Gödelovy věty o úplnosti
gradient noise
graf
graf scény
gramatika bez cyklu
Greibachové norm. forma
Grid File
group of frames
grupa
grupoid
GStreamer
Gustafsonův zákon
Hamiltonovský graf
Hammingova vzdálenost
Hammingovo okno
harmonická funkce
head of line blocking
HLS
HLSL
HMM
holografický kód
holografie
hologram
homeomorfismus
homogenní souřadnice
homografie
homomorfismus
Houghova transformace
hrana
Huffmanovo kódování
Huygensův princip
HW/SW codesign
HW platforma
hybridní kodér
H.264
ideál okruhu
IFFT
IIR filtr
image order
immediate mode (OGL)
implikace
index ekvivalence
infimum
integrace (HW/SW cod.)
integrální transformace
integritní omezení
intensity stereo coding
intenzita vlny
interf. na tenké vrstvě
interference světla
interferenční rovnice
interferometr
interlacing
interpolace
inverzní filtr
inverzní prvek
ireducibilní prvek
iSLIP
izometrizmus
izomorfizmus
I-frame
jazyk s rovností
jádro filtru
jednoduché pravidlo
jednotka
jednotkový prvek
JPEG
JPEG2000
Kahnova síť procesů
kandidátní klíč
kardinalita relace
Kleenoeho algebra
kodek
koherentní vlny
kombinace (TIN)
komplexní amplituda
kompozice (TIN)
kompresní poměr
komutativita
konečný automat
konfigurace automatu
konfigurace pásky
konfigurace stroje
kongruence
kontextová gramatika
konvergence
konvoluce
konvoluční kód
konstruktivní interference
kontext paketu
kontrakce
korelace
korelační metoda (sign.)
kostra grafu
konvoluční teorém
kovarianční matice
kovarianční metoda
kódování zprávy
kruhová polarizace
kružnice (grafy)
Kruskalův algoritmus
křížový přepínač
kubická interpolace
kvantifikátor
k-D tree
lag (zprav. řeči)
LBS
Lambertovo stínování
Lambertův osvětl. Model
Laplacian of Gaussian
Laplacián
laser
latence
left-edge algoritmus
levá lineární gramatika
levá/pravá derivace
levě rekurz. Gramatika
Levinson-Durbin alg.
Libav
likelihood
lineární filtrace
lineární hashování
lineární interferometr
lineární interpolace
lineární kombinace
lineární kód
lineárně omezený aut.
lineární polarizace
lineární prostor
lineární transformace
lineární závislost
lokální transformace obr.
lokální změna (DB)
LPC
LPCC
LSB modulace
LTI filtr
LZW84
Machův-Zenderův inter.
makroblok (video)
mapování (HW/SW cod.)
maskování (zvuk)
materiál
maximální párování
Mealy KA
mediánový filtr
Mel-frekvenční cepstrum
metastabilní hladina
metodika
metrický prostor
metrika
Michelson. interferometr
middleware (DB)
MIDI
mimoosový hologram
minimalizace (TIN)
minimalizace KA
minimální pole
MIP mapping
MKV
mobilita operátoru
model s časem platnosti
model teorie
modul (imag. číslo)
modus ponens
Moore KA
morfizmus jazyka
morfologie
motion compensation

motion estimation
motion vectors
MPEG
MPEG-1
MPEG-2
MPEG-4
mp2
mp3
mp4
mřížkový šum
multimediální DB
multimediální framework
multimediální kontejner
Myhill-Nerodova věta
NP (třída)
NP-těžký jazyk
NP-úplný jazyk
nativní XML DB
návrh (HW/SW cod.)
návrh shora-dolů
návrh zdola-nahoru
nearest neighbour
nedosažitelný stav
negativní potvrzování
nejmen. spol. nás.
největ. spol. děl.
největší párování
nelineární filtrace
nerozlišitelné stavy
netriviální dělitel
network interface (směr.)
neutrální prvek
nezávislý šum
norma
normal mapping
normovaný prostor
normalizace ACL
normalizace DB
normální podgrupa
normální zastavení TS
NSpace
NTime
n-rozměrný prostor
obecná neomez. gram.
OBJ
object order
objektová DB
objektově-rel. mapování
objemový hologram
obor integrity
obyčejný graf
odhady (HW/SW cod.)
ohodnocení grafu
ohodnocení proměnných
okruh
okruh polynomů
Open Inventor
OpenAL
OpenGL
OpenGL ES
OpenSceneGraph
optimální filtrace obr.
or stav
OR DB
orientovaný graf
ortogonální báze
ortonormální báze
OSS
otevřená koule
overrun
P (třída)
P vs NP
paketová chyba
parallel iter. matching
parita
PAL
paralelní projekce
parciální funkce (TIN)
parciální rekurz. Funkce
PDF
pepř a sůl
PER
perfektní kód
Perlin noise
perspektivně korektní int.
perspektivní projekce
pevný bod
Phongovo stínování
Phongův osvětl. model
planární graf
plánování (HW/SW cod.)
PLY
PNG
počáteční funkce (TIN)
podalgebra
podgraf
podmíněná entropie
polarizace světla
polarizační filtr
polarizační rovina
polarizátor
pole
Pollackovo pravidlo
pologrupa
postrelační DB
PostScript
Postův koresp. Problém
Postův systém
prahování
pravá kongruence
pravá lineární gramatika
pravě rekurz. gramatika
pravdivost formule
predikátová logika
predikátový symbol
prediktor (komprese)
prefixová ekvivalence
primární klíč
primární paprsek
primitivně rekurze
primitivně rekurz. funkce
Primův algoritmus
prenexní tvar formule
problém zastavení
procedurální textura
programovatelný obvod
progressive scan
propustnost
prostorová DB
prostorová složitost
prostorový index
protokolové inženýrství
proudový kód
prvočinitel
přechodová funkce
přechodová relace
předmětová vlna
předpoklad
přenosový kanál
přepínací příkon
přesnost (HW/SW cod.)
přidělování lístků
přidělení (HW/SW cod.)
příkon
přímá derivace
přímý součin algeber
PSPACE
psychoakustický model
pumping lemma
pumping lemma pro BKJ
P-frame
quad-tree
queue manager
QuickTime
radiozita
rank-order metody
ray casting
ray tracing
Rayleighovo kritérium
rámec
realizace jazyka
realms
redukce (důkaz)
redukovatelný jazyk
referenční vlnoa
reflektivita (grafika)
registered param. numbers
rekurentní kód
rekurzivní gramatika
rekurzivní jazyk
rekurzívně vyč. jazyk
relace ekvivalence
relační DB
regulární jazyk
regulární množina
regulární výraz
residuál
retained mode
Riceova věta
RISC
RLE
robustní vodoznak
ROSE
route control processor
rozhodnutelnost
rozbalení smyčky
rozlišitelnost dvou bodů
rozdělení (HW/SW cod.)
rozhodovací problém
rozšíření pole
rozšířený ZA
RTL
rychlá konvoluce
R-tree
R+ tree
řád pole
SAT problém
segmentaci (obraz)
sekvenční číslo
substituce jazyka
SDL
sekundární paprsek
sequence graf
Shannonova věta
shared backplane
shared memory archit.
shininess
shlukování (DB)
simplex
simplex noise
síť Beneš
síťový tok
skalární součin
slack
sled
slepá dekonvoluce
sliding window
slow path (směr.)
složená parita
snapshot DB model
SNR
Sobelův filtr
souvislý graf
specifikace (HW/SW cod.)
spektrální hustota výkonu
spektrum
spekularita
spekulární osvětlení
spojité zobrazení
spojování smyček
spontánní emise
sporná teorie
SQL:1999
SŘBD
standard MIDI file
state occupation function
StateCharts
statický dataflow graf
statický příkon
steganografie
stencil buffer
stimulovaná emise
stínový paprsek
STL
stop and wait
striktně parciální funkce
strom
stupeň relace
stupeň uzlu
subsurface scattering
superpozice
supremum
svaz
SVG
switched backplane
symetrický kód
symetrizace grafu
synchronní dataflow graf
syntéza (HW/SW cod.)
šum
tah
tautologie
temporální DB
temporální maskování
teorie
testbench
term
tesselation shader
textura
texturovací souřadnice
těleso
timeout (sítě)
Torus síť
Töplitzova matice
totální funkce
traffic manager
transformace gramatiky
transform feedback
trénování
triviální dělitel
Turing. vyčísl. funkce
Turingův stroj
turnaj
Twin Grid File
two-level Grid File
typ algebry
underrun
unitární prostor
univerzální algebra
univerzální TS
univerzum
utilization
uzavřená koule
uzavřenost
uzavřený tah
uzávěrové vlastnosti
úplný graf
úzkopásmový šum
úplné uspořádání
úplně definovaný KA
úplný graf
úplnost (složitost)
úplný prostor
úplný TS
validace (HW/SW cod.)
value noise
vázaná dekonvoluce
virtuální fronta
visuální kryptografie
vektorové kvantování
verifikaci
vertex shader
věrnost (HW/SW cod.)
věta (gramatika)
věta o dedukci
věta o kompaktnosti
větná forma
Viterbi pravděpodobnost
víceznačná věta
VHDL
vlastní gramatika
VLC
vlnová funkce
vložení paketu
vodoznak
vokodér
volatilní paměť
VP8
VP9
VRML
VST
Vulkan
vykreslovací řetězec
vyprázdnění zásobníku
vysílací pravděpodobnost
výpočet TS
výroková logika
vzorkovací teorém
waveform
WCET
WebGL
WebM
Wienerův filtr
XML DB
XSLT
Xvid
X3D
Youngův pokus
základní tón
zásobníkový automat
zbytečný symbol
zkratový příkon
znovuzasílání
zobrazovací rovnice
zřetězení smyčky
ztráta fragment. dat
ε-okolí
ε-pravidlo
ε-přechod
μ-rekurzivní funkce
2D textura
3D textura
3DS
3way handshake



asymptotické horní omezení
$O(f(n)) = \{g(n) \in F \mid \exists c \in R^+, \exists n_0 \in N\, \forall n \in N: n \geq n_0 \Rightarrow 0 \leq g(n) \leq c \cdot f(n)\}$

axiom kvantifikátoru
$(\forall x(\varphi \rightarrow \psi)) \rightarrow (\varphi \rightarrow (\forall x \psi))$

axiomy metriky
$\delta(x,y) = 0 \Leftrightarrow x = y,\ \delta(x,y) = \delta(y,x),\ \delta(x,y) + \delta(y,z) \geq \delta(x,z)$

axiomy normy
$||x|| = 0 \Leftrightarrow x = 0,\ ||a \odot x|| = |a| \cdot ||x||,\ ||x + y|| \leq ||x|| + ||y||$

axiom substituce
$(\forall x \varphi) \rightarrow \varphi_x[t]$

Baum-Welchova pravděpodobnost
$$P(O) = \sum_X P(O, X)$$

cepstrum
$c(n) = \mathrm{IFFT}(\ln|\mathrm{FFT}(s(n))|^2)$

Chomského normální forma
$A \rightarrow BC,\ A \rightarrow a,\ S \rightarrow \varepsilon$

CMOS příkon
$P = C_{load} \cdot \alpha \cdot f \cdot V^2$

DFT, IDFT
$$F(k) = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} f(n) e^{-j 2\pi k n / N} \qquad f(n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} F(k) e^{j 2\pi k n / N}$$

DTW částečná kumulovaná vzdálenost
$g(x,y) = \min \forall \text{ předchůdci } (g(\text{předchůdce}) + \text{váha(cesta k předchůdci)} \cdot d(x,y))$

DTW konečná normovaná vzdálenost
$D = g(T,R) / (T + R)$

Eulerův vzorec
$|U| - |H| + s = 2$

entropie
$$H(X) = \sum_{i=0}^{n} p_i I(p_i) = \sum_{i=0}^{n} p_i \log_2\left(\frac{1}{p_i}\right)$$

Fourierovy koeficienty
$$p = \sum_{k=1}^{n} c_k x_k$$

harmonická vlna
$u = U \cos(w(t - z/c) + \varphi_0)$

hologram – rekonstruovaná vlna
$U = \tau U_r = I_0 U_\tau + I_\tau U_r + U_0 I_\tau + U_0^* U_r^2$

HMM pravděpodonost cesty
$$P(O, X) = a_{X(0)X(1)} \prod_{t=1}^{T} b_{X(t)}(O(t)) a_{X(t)X(t+1)}$$

integrální transformace
$t(x) = \int K(x,t) f(t)\, dt$

interferenční člen
$$2\sqrt{(I_1 I_2)}\cos(\phi_2 - \phi_1)$$

interferenční rovnice
$$I = I_1 + I_2 + 2\sqrt{(I_1 I_2)}\cos(\phi_2 - \phi_1)$$

izometrizmus
$\delta(x,y) = \delta(f(x), f(y))$

komplexní amplitudová propustnost
$\tau(x,y) = U(x,y,d_1) / U(x,y,d_2)$

lineární závislost vektorů
$\exists$ a, b, …, z nichž alespoň 1 je nenulový, že: a x + b y + … = 0

lineární zobrazení (filtr, …)
$f(a\ x_1 + b\ x_2) = a\ f(x_1) + b\ f(x_2)$

LPC matice koeficientů
$R(0)\ a_1 + R(1)\ a_2 + ... + R(P - 1)\ a_P = -R(1)$

minimalizace
$f(\bar{x})$ je nejmenší y, že: $g(\bar{x}, y) = 0$, $g(\bar{x}, z)$ je definována pro všechna $z < y,\ z \in N$

množství informace
$I(p) = \log_2(1/p)$

modus ponens
$\varphi \wedge \varphi \rightarrow \psi \Rightarrow \psi$

normalizovaná kros korelace
$$R(m) = \Sigma_{n=0}^{N-1-m} \frac{s(n) s(n+m)}{\sqrt{E_1 E_2}}$$

P, NP
$P = U_{k=0}^{\infty} DTime(n^k) \qquad NP = U_{k=0}^{\infty} NTime(n^k)$

perfektní kód – počet red. bitů
$c = \log_2(n + 1)$

pevný bod rovnice nad reg. výrazy
$X = pX + q \Rightarrow X = p^*q$

Phongovo osvětlení
$I = I_A + (L \cdot N) \cdot I_D + (R \cdot V)^e \cdot I_S$

pravá lineární gramatika
$A \rightarrow a^*B,\ A \rightarrow a^*$

pravá kongruence
$\forall$ u, v, w $\in \Sigma^*$: u ~ v $\Rightarrow$ uv ~ vw

pravá regulární gramatika
$A \rightarrow aB,\ A \rightarrow a,\ S \rightarrow \varepsilon$

prefixová ekvivalence
$u \sim_L v \Leftrightarrow \forall uw \in \Sigma^*: uw \in L \Rightarrow vw \in L$

primitivní rekurze
$f: N^{k+1} \rightarrow N^m: f(\bar{x}, 0) = g(\bar{x}),\ f(\bar{x}, y + 1) = h(\bar{x}, y, f(\bar{x}, y))$

přesnost
$$A = 1 - \frac{|E(D) - M(D)|}{M(D)}$$

pumping lemma (bezkontextové)
L je bezkont. Jazyk - existuje $p > 0$:
$z \in L \wedge |z| \geq p \Rightarrow z = u\ v\ w\ x\ y, \qquad v\ x \neq \varepsilon, \qquad |v\ w\ x| \leq p, \qquad u\ v^i\ w\ x^i\ y \in L$

pumping lemma (regulární)
L je nekonečný reg. jazyk - existuje $p > 0$:
$\omega \in L \wedge |\omega| \geq p \Rightarrow \qquad \omega = x\ y\ z \quad \wedge \qquad 0 < |y| \leq p \quad \wedge \qquad x\ y^i\ z \in L$ pro $i \geq 0$

redundance kódu
$r = 1 - u$

signál chyby
$$e(n) = s(n) - s'(n) = s(n) - \left(-\sum_{i=1}^{P} a_i s(n-i)\right)$$

uzavřený orotonormální systém
$$p = \sum_{k=1}^{\infty} c_k^2 = \|f\|^2$$

Viterbiho pravděpodobnost
$$P(O) = max_X(P(O, X))$$

účinnost kódu
$u = H(X) / L$

věrnost
$$F = 100 \times \frac{2}{n(n-1)} \times \sum_{i=1}^{n} \sum_{j=i+1}^{n} u_{i,j}$$

výrokové axiomy
$\varphi\ (\psi \rightarrow \varphi),\ (\varphi \rightarrow (\psi \rightarrow \eta)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \eta)),\ ((\neg\psi) \rightarrow (\neg\varphi)) \rightarrow (\varphi \rightarrow \psi)$

Rayleigho kritérium
$\theta = 1{,}22\ \lambda / D$

zobecnění
pro x z φ se odvodí $\forall x$: φ

zobrazovací rovnice
$L(x, w_0) = L_E(x, w_0) + \int_\Omega L_I(x, w_i) \cdot f_r(x, w_i, w_0) \cdot \cos(\theta_i)\, dw_i$

2D konvoluce
$$y(m,n) = \sum_{i \in I} \sum_{j \in J} x_{(m+i, n+j)} k'_{i,j}$$

intenzita vlny
$I = |U(r)|^2$

alfa	A α
beta	B β
gama	Γ γ
delta	Δ δ
epsilon	E ε
zéta	Z ζ
éta	H η
théta	Θ θ
ióta	I ι
kappa	K κ
lambda	Λ λ
Mí	M μ
Ný	N ν
Ksí	Ξ ξ
omikron	O ο
pí	Π π
ró	P ρ
sigma	Σ σ ς
tau	T τ
ypsilon	Y υ
fí	Φ φ
chí	X χ
psí	Ψ ψ
omega	Ω ω

grafy		
typ	obyčejné	orientované
def.	G = (U,H), H = {{u,v}, u,v ∈ U, u ≠ v}	G = (U,H), H = {(u,v), u,v ∈ U}
pojmy a operace	sled, tah, cesta, kružnice, podgraf, faktor, strom, kostra, ohodnocení, planarita, izomorfismus, Eulerův vzorec, barvení, hamiltonovský graf, eulerovský graf, stupeň uzlu	• **symetrizace**: převod na obyčejný zanedbáním směrů hran turnaj, vstupní a výstupní stupeň uzlu, úplnost
souvislost	**souvislý**: mezi každými 2 uzly existuje sled	• **souvislý**: symetrizace je souvislý graf • **silně souvislý**: pro každé 2 uzly existuje orientovaná cesta
eulerovský	souvislý + každý uzel je sudého stupně	souvislý + pro každý uzel platí: vstupní a výstupní stupeň se rovnají
algoritmy	minimální kostra: • **Kruskalův**: seřadí hrany a postupně přidává, O(|E| log |V|) • **Primův**: postupně přidává hrany (nemusí řadit všechny), O(|E| log |V|)	minimální cesta: • **Dijkstrův**: nelze se zápornými cestami, kvadr. slož, O($\|V\|^2$) • **Floyd-Warshalův**: odhalí kružnice záp. délky, matice O($\|V\|^3$)

Konstanty:

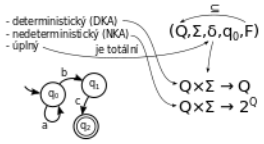
kompresní poměry:
- video: 1:20 – 1:200
- zvuk: 1:11 (mp3)
- obraz: 1-50 – 1:100 (1:15 bezztrát.)

rozlišení oka: 1’ (1/60 stupně)
Vdd: cca 1V
mp3 bitrate: 128 kbps (stereo)
chromatické číslo plan. grafu: 4

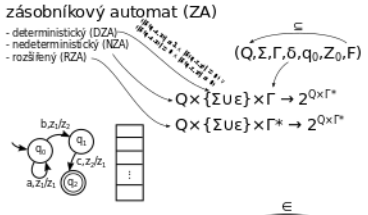
typická délka rámce: 10 ms (100 za s, 400 vzorků)
vln. délka světla: 400 nm (fialová) – 700 nm (červená)

automaty

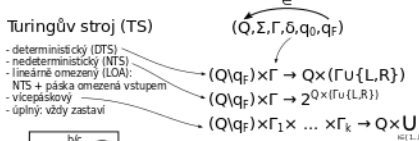
konečný automat (KA)



zásobníkový automat (ZA)



Turingův stroj (TS)



gramatiky (N,Σ,P,S)

typ 3 - regulární gramatika (RG)

- pravá lineární	N → Σ*N, N → Σ*
- levá lineární	N → NΣ*, N → Σ*
- pravá regulární	N → ΣN, N → Σ, S → ϵ
- levá regulární	N → NΣ, N → Σ, S → ϵ

typ 2 - bezkontextová gramatika (BKG)

N → (N∪Σ)*

typ 1 - kontextová gramatika (KG)

α N β → α (N∪Σ)⁺ β α, β ∈ (N∪Σ)*

popř. i S → ϵ

typ 0 - neomezená gramatika

(N∪Σ)* N (N∪Σ)* → (N∪Σ)*

funkce

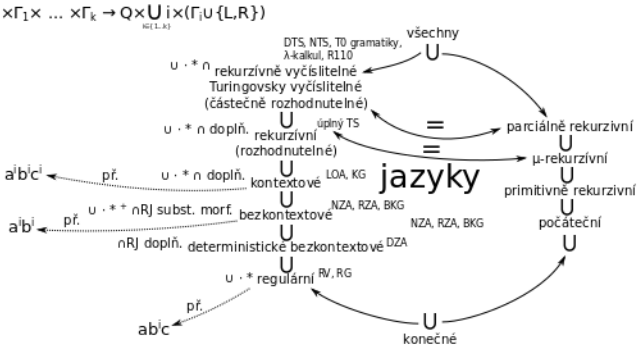
totální - definovány všude
parciální - nemusí být totální
striktně parciální - nejsou totální

počáteční
nulová: ξ() = 0
následník: σ(x) = x + 1
projekce: π^n_k = *k*-tá složka z *n*-tice

primitivně rekurzivní
z počátečních pomocí:
- kombinace: f × g(x) = (f(x),g(x))
- kompozice: g ∘ f(x) = g(f(x))
- primitivní rekurze: z *g* a *h* vytvoří *f*, že
  f(x,0) = g(x)
  f(x,y+1) = h(x,y,f(x,y))

μ-rekurzivní
jsou totální a vyčíslitelné

parciálně rekurzivní
primitivně rekurzivní plus minimalizace:
z g(x⃗,w) vytvoří f(x⃗):
f(x⃗) = nejmenší y takové, že:
g(x⃗,y) = 0 ∧ ∀ z < y: g(x⃗,z) je definována



ostatní

regulární výrazy (RV)

∅	∅
ε	{ε}
a	{a}
p + q	P ∪ Q
pq	P . Q
p*	P*

λ-kalkul
celulární automat Rule 110 (R110)