

# 01 – CAP teorém a jeho vztah k NoSQL databázím

## Distribuované systémy

Hardware je levný, co se může pokazit, tak se nejspíš pokazí. Případný výpadek je brán jako očekávaný stav.

- Systémy s distribuovaným výpočetním výkonem
  - o paralelizace = úspora času
- Systémy s distribuovaným úložištěm
  - o vyšší redundance, propustnost a dostupnost
  - o vyšší kapacita



*Author of CAP theorem is bald so he needs a CAP*

## NoSQL (Not only SQL) databáze

- nepoužívají tradiční relační DBMS
- nemá předem definované schéma tabulky
- Tradiční RDBMS vyžadují vertikální škálování
  - o Nákup lepšího HW stojí exponenciálně více
- moderní aplikace vyžadují škálovat horizontálně
  - o Nákup dalšího serveru roste lineárně spočtem uživatelů
- Vyvinuty pro běždný HW, který je levný

## CAP teorém

CAP teorém je věta popisující 3 základní vlastnosti distribuovaných systémů (např. NoSQL databází). Je ovšem dokázáno, že v jednu chvíli může systém splňovat pouze 2 ze 3 daných vlastností.

### Consistency

- každé čtení vrátí aktuální data nebo chybu

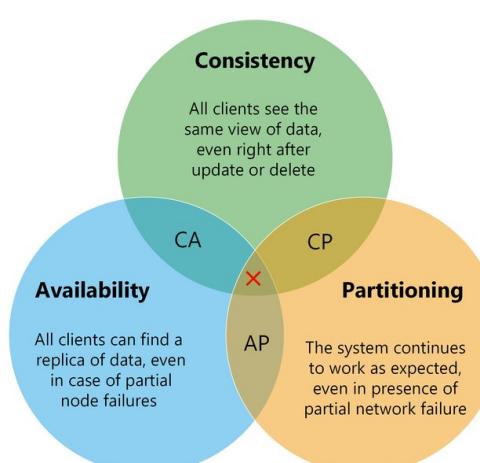
### Availability

- na každý požadavek přijde správná odpověď

### Partition tolerance

- systém funguje i po výpadku (partition je ve smyslu rozdelení sítě)

Reálně by každý systém měl splňovat Partition tolerance, tudíž jde v podstatě o tradeoff mezi prioritou Availability (PA) a Consistency (PC).



### P + C = not available (MongoDB)

- Pokud systém nedokáže vrátit aktuální data tak vrátí chybu.

### P + A = not consistent (Cassandra) = největší možná dostupnost

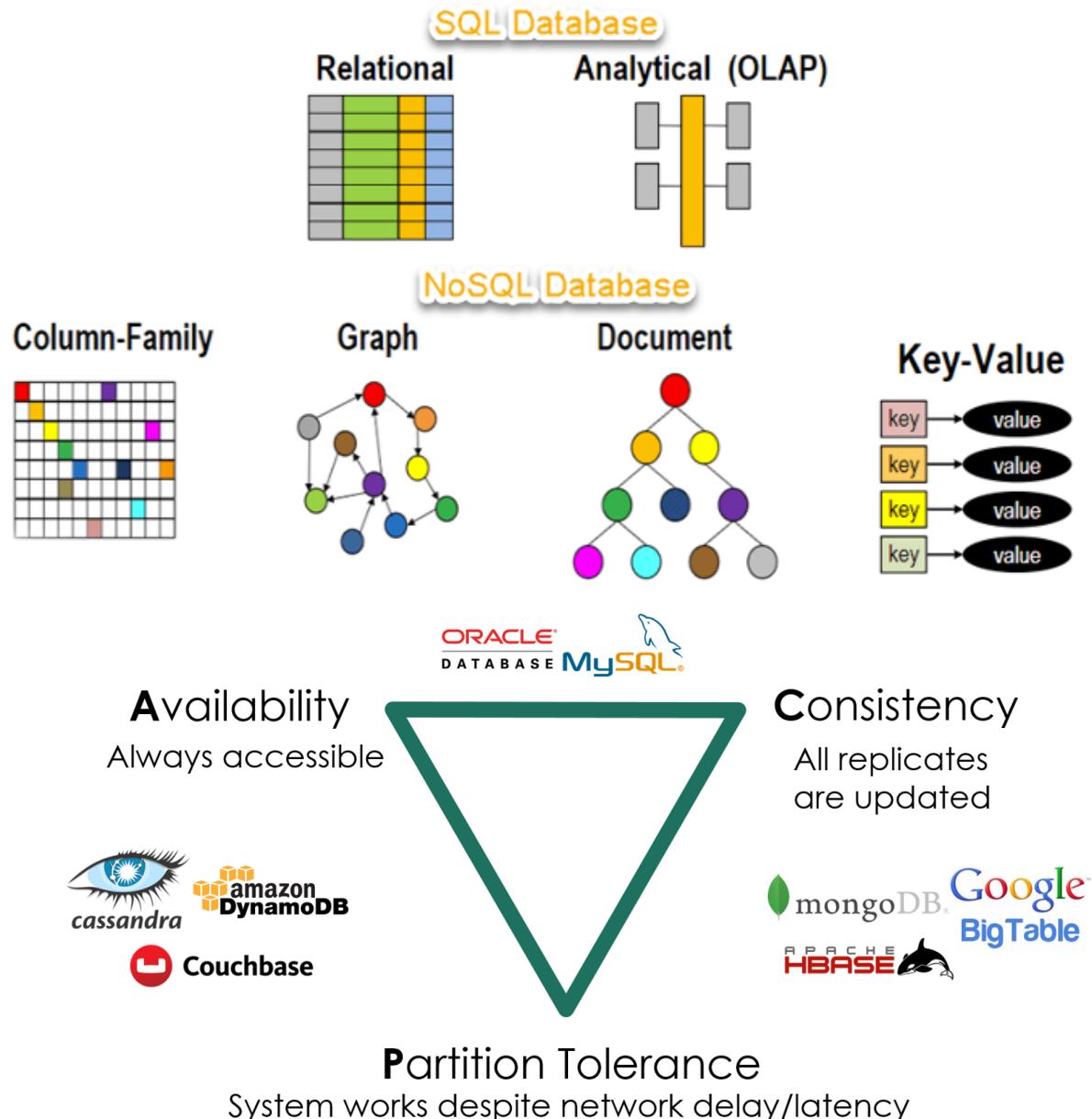
- Neexistuje záruka, že čtená data jsou aktuální.
- Tolerantní k výpadku dílčí služby.

### C + A = not partition tolerant (RDBMs) = maximální možná dostupnost

- V případě výpadku části (nebo sítě) je to problém.
- Než aby systém vracel nekonzistentní výsledky, radši neodpoví vůbec.

## Segmentovaná konzistence a dostupnost

- Každá komponenta systému splňuje část CAP teorému.
- podle dat (filesystems)
  - o jiná data mohou vyžadovat jinou úroveň dostupnosti a segmentace
- podle operací (databases)
  - o zápis může splňovat něco jiného, než čtení



# 07 – Aplikační server a JNDI služba

popis aplikačního serveru, webový kontejner, EJB kontejner, využití JNDI ke konfiguraci datových zdrojů

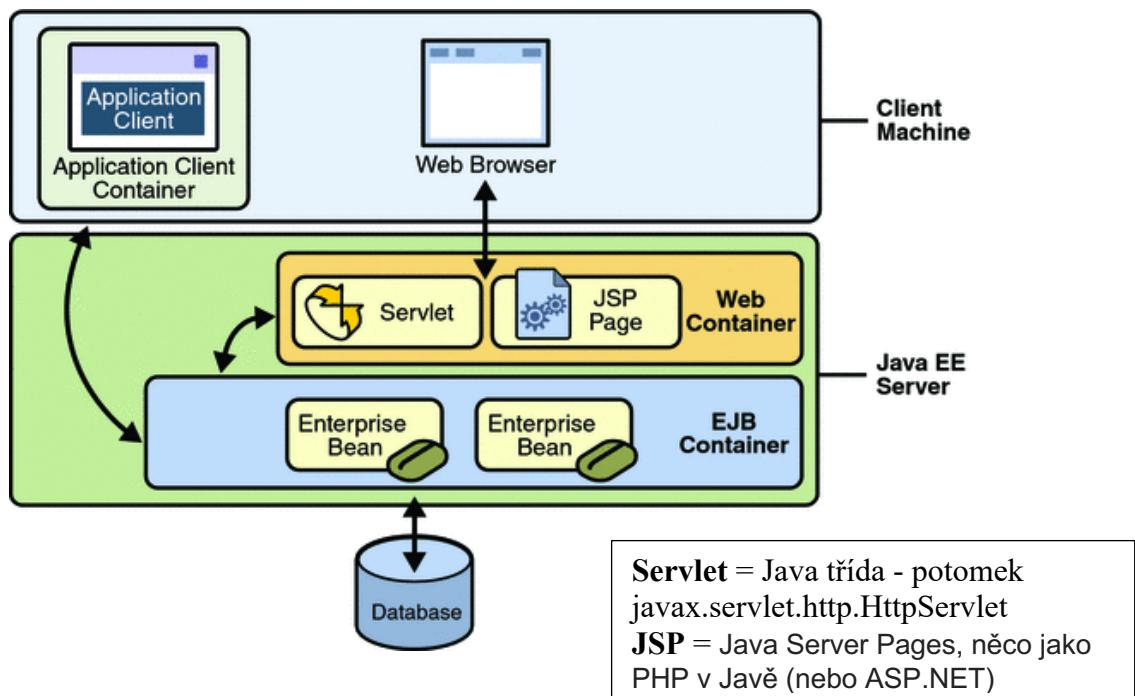


## Aplikační server

Serverová aplikace poskytující **knihovny dle specifikací Java EE platformy**

- Tyto knihovny implementují **veškerá API obsažená v Java EE**.
- Kromě toho poskytuje aplikační server další klasické služby jako např.
  - o administrátorskou konzolu,
  - o logování atp.
- Např: **GlassFish** (Oracle), JBoss (Red Hat), JRun (Adobe systems)
- Aplikační server zajišťuje
  - o **Vývoj webových aplikací** - Java Servlets, Java Server Pages (JSP),...
  - o **Dependency management**
  - o **Přístup k DB** – Java Persistence API (**ORM**)
  - o **Vývoj sdílené business logiky** – Enterprise Java Beans (**EJB**)
  - o **Podpora technologií Webových služeb** – SOA, REST
- Na aplikační server se nasazují **komponenty** – jednotky, ze kterých je složena aplikace
- Komponent existuje několik druhů, nejdůležitější dva:
  - o **Webové komponenty** – servlety, JSP soubory a JSF soubory.
  - o **Enterprise JavaBeans (EJB) komponenty** – javovské třídy, které tvoří logiku aplikace a manipulují s daty
- Komponenty se spouštějí v **kontejneru** – ty komponentům přidávají funkcionality
- Aplikační server obsahuje dva kontejnery
  - o **Webový kontejner** – má na starosti správu webových komponent.
  - o **EJB kontejner** – má na starosti správu EJB komponent.

## GlassFish



## EJB kontejner

Dedikovaný virtuální prostor v aplikačním serveru, kam se nasazují EJB komponenty

- **Komunikace se vzdáleným klientem** – zjednodušuje komunikaci mezi klientem a aplikací
- **Dependency injection** – zajišťuje naplnění deklarovaných proměnných (datových atributů) např. dalšími EJBeany, JMS, datovými zdroji (zejména SQL připojení), atd.
- **Pooling** – vytváření poolu instancí pro bezstavové bean a message-driven bean (MDB)
- **Řízení životního cyklu** – stará se o vytváření, inicializaci a destrukci instancí beanů

## Webový kontejner

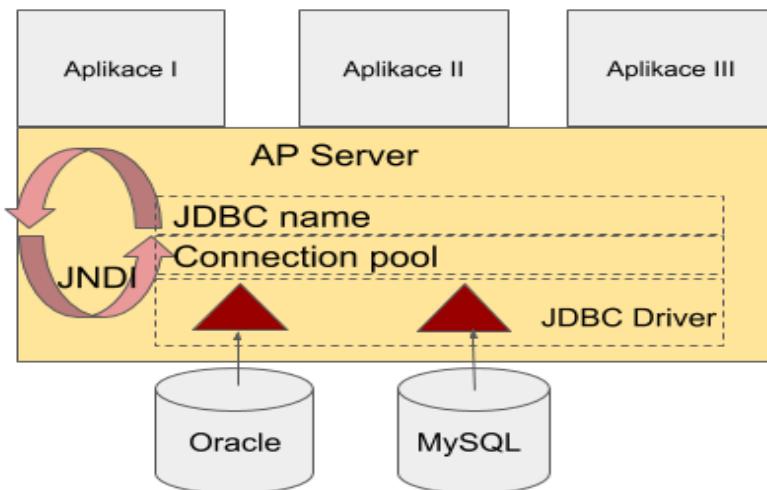
(podobně jako EJB kontejner)

- **Mapuje URL adresy na servlety**
- **Řídí životní cyklus servletů** - vytváření, mazání
- **Pracuje s dotazy a odpověďmi serveru**

Spravuje **pool servletů**

## JNDI (Java Naming and Directory Interface)

- Služba aplikačního serveru která zajišťuje **jednotný přístup k datům** (pro přístup k DB nebo získání instance EJB)
- Administrátor nastaví JDBC (Java DataBase Connectivity) driver, connection pool (username, password, url,...) a nastaví název JDBC zdroje (JNDI jméno)
- Tvůrce aplikace nenastavuje připojení k DB, zadá jen název **datového zdroje** a JNDI ho připojí k DB



# 17 – Architektura ZS, možnosti reprezentace znalostí

# 18 – Inferenční mechanizmus, způsoby realizace IM

## Znalostní systém

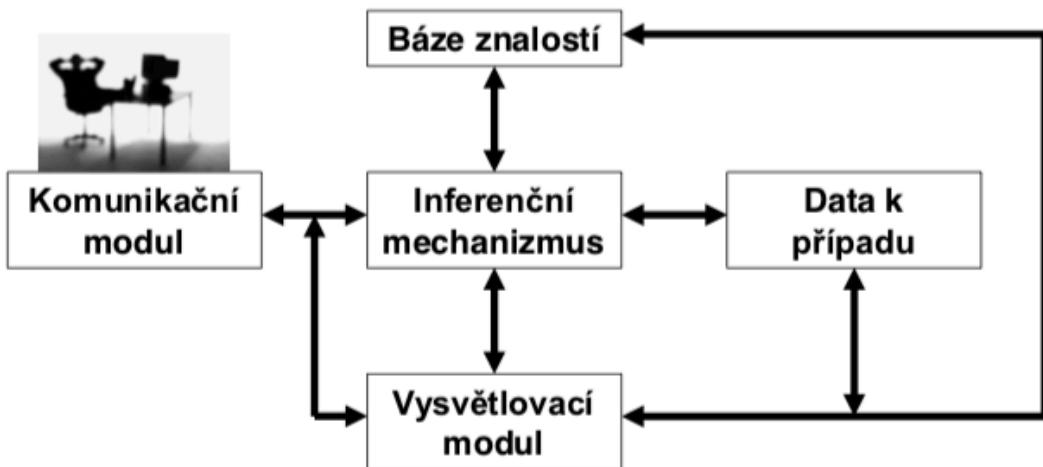
- počítačový program, který využívá znalosti a inferenční procedury k řešení problémů

## Vybrané charakteristické rysy ZS

- Oddělení báze znalostí od ZS (stejný ZS se dá využít pro jinou bázi znalostí)
- Dialogový režim („Dotaz ZD – odpověď uživatele“)
- Vysvětlovací činnost (systém by měl zdůvodnit jak přišel k závěrům)

## Součásti ZS

- **Báze znalostí** - zakódované znalosti experta
- **Inferenční mechanizmus** - odvozovací mechanizmus (vyvozování závěrů)
- **Báze dat** (ke konzultovanému případu)
- **Vysvětlovací modul**
- **GUI** – Rozhraní pro komunikaci s uživatelem



## Typy úloh pro ZS

- Diagnóza (proč systém nefunguje)
- Interpretace (určení významu dat)
- Monitorování (sleduje data a určí když se má zasáhnout – např. pacient začne umírat)
- Plánování (nalezení akcí k dosažení cíle)
- Design (např. návrh počítačové sestavy)
- Predikce (předpověď budoucích událostí)

### Reprezentace znalostí v ZS

1. Predikátová logika
2. Sémantické sítě
3. Bayesovské sítě
4. Rámce
5. Pravidla

### Inference v ZS

1. Logické metody
2. Přímé a zpětné zřetězení
3. Generování a testování
4. Analogie

## Reprezentace znalostí v ZS

### Predikátová logika

- Jednoduchá reprezentace znalostí

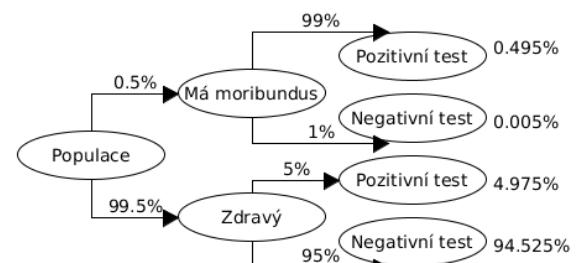
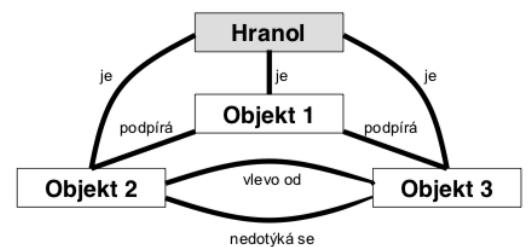
### Sémantické sítě

- Reprezentace znalostí za pomocí grafu – objekty jsou uzly, relace (vyjadřují znalosti) jsou hrany

### Bayesovské sítě

- Pro popis pravděpodobnosti sítě vzájemných vazeb
- Hledáme PST hypotézy ( $H$ ) a předpokládáme, že jsou evidence ( $E$ ) nezávislé (Bayesova v.)  

$$P(H|(E_1 \cap E_2 \cap \dots \cap E_n))$$
- Jak vyřešit vstup, který nejsou jen konjunkce?
  - o Nagace:  $\neg E_1 \wedge E_2 \rightarrow H_1$ 
    - $P(\neg E_1 | H_1) = 1 - P(E_1 | H_1)$
  - o Disjunkce:  $E_1 \vee E_2 \rightarrow H_1$ 
    - $\max(P(H_1 | E_1), P(H_1 | E_2))$



	$P(A = a)$	$P(A = n)$
	0,80	0,20
	0,20	0,80

	$P(B = a)$	$P(B = n)$
	0,02	0,98
	0,98	0,02

A	$P(D = a A)$	$P(D = n A)$
a	0,90	0,10
n	0,01	0,99

C	$P(E = a C)$	$P(E = n C)$
a	0,70	0,30
n	0,10	0,90

Křeslo (jev A)      Cvičení (jev B)

Spolupracovník (jev D)      Zranění zad (jev C)

Bolesti zad (jev E)

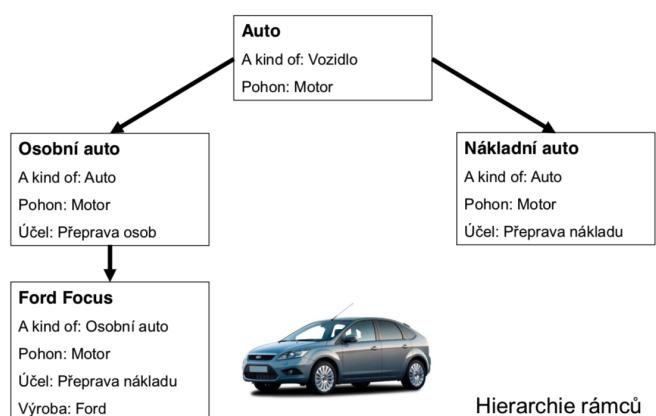
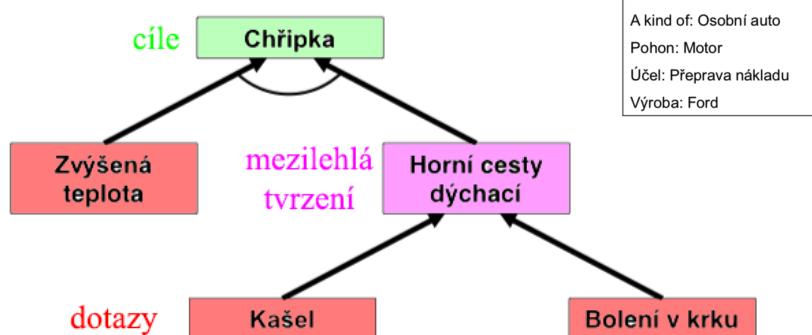
A	B	$P(C = a A, B)$	$P(C = n A, B)$
a	a	0,90	0,10
a	n	0,20	0,80
n	a	0,90	0,10
n	n	0,01	0,99

### Rámce

- V podstatě objekty z OOP
- Vhodné pro reprezentaci statických znalostí

### Pravidla

- Nejběžnější způsob reprezentace znalostí:
  - o IF předpoklad THEN závěr
  - o IF situace THEN akce
- znázornění pomocí AND/OR grafu



Hierarchie rámčů

## Inference ve znalostním systému

- Inferenční mechanismus (IM) je součást ZS
- Běh znalostního systému:
  - o Porovnání (vytvoříme množinu pravidel, která splňují předpoklady)
  - o Rozhodnutí sporu (výběr jednoho pravidla z rozhodovací množiny)
  - o Úkon (Důsledkem může být např. přidání předpokladu)

### Logické metody

- Rozhodování pomocí **rezoluční metody**
- Konflikty (stav kdy se dá splnit více pravidel) se dají řešit různými způsoby (preferujeme složitější pravidla, jednodušší, novější data,...)

Table 2. Three Kinds of Inference

	Abduction	Deduction	Induction
Premises	Fact	Rule	Case
Premises	Rule	Case	Fact
Outcome	Case	Fact	Rule

### Dedukce

- $A, A \Rightarrow B | B$  (platí pravidlo a předpoklad, odvozujeme platnost závěru)
- $\neg B, A \Rightarrow B | \neg A$  (platí pravidlo a nezávěr, odvozujeme neplatnost předp.)  
o *Jestliže prší, je mokro. Není mokro, tedy neprší.*
- $\neg(A \wedge B) \wedge A \Rightarrow \neg B$  (nemůže platit současně A i B, A platí  $\Rightarrow B$  nemůže platit)  
o *Není možné, aby vyhráli červení i modří. Vyhráli červení. Z toho vyplývá že modří nevyhráli.*

### Abdukce

- $B, A \Rightarrow B | A$  (platí pravidlo a závěr, domníváme se že tudíž platí předpoklad)

### Indukce

- Opakovaně pozorujeme A a B současně
- Odvozujeme že mezi A a B je vztah implikace ( $A \Rightarrow B$  nebo  $B \Rightarrow A$ )

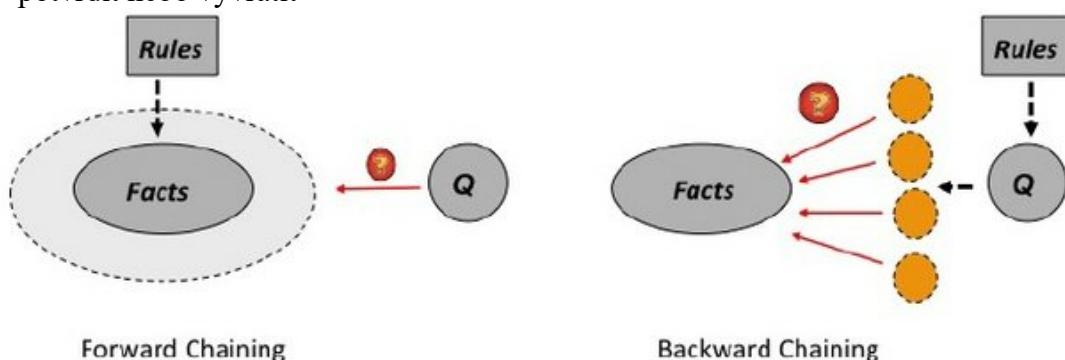
### Zpětné a přímé zřetězení

#### Přímé zřetězení (předpoklady $\rightarrow$ cíle)

- Vycházíme z předpokladů a odvozujeme závěry které slouží jako předpoklady pro další pravidla

#### Zpětné zřetězení (cíle $\rightarrow$ předpoklady)

- Vycházíme z cílů, které chceme odvodit a hledáme pravidla, umožňující tyto cíle potvrdit nebo vyvrátit



### Generování a testování

- Opakovaně generujeme možná řešení a testujeme, zda vyhovují všem požadavkům (IF nastane situace THEN proved akci)

### Analogie

- Hledáme podobné příklady, které byly už vyřešené (např. podobný soudní výrok)

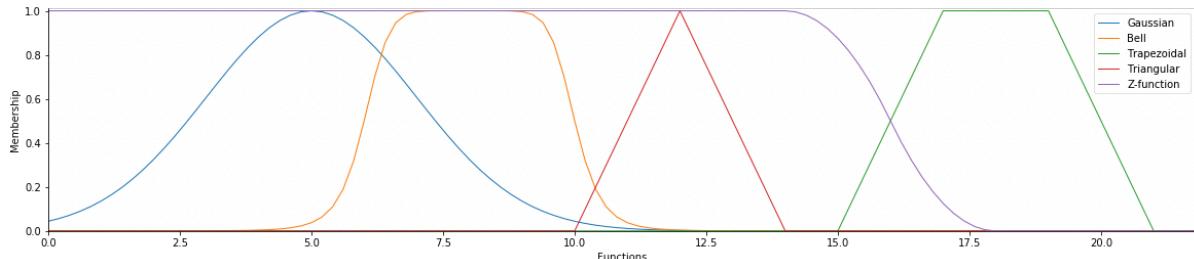


# 21 – Reprezentace znalostí a inference pomocí fuzzy systémů

- Použití: Termostaty, pračky, automatické ostření, pohyb robota,..

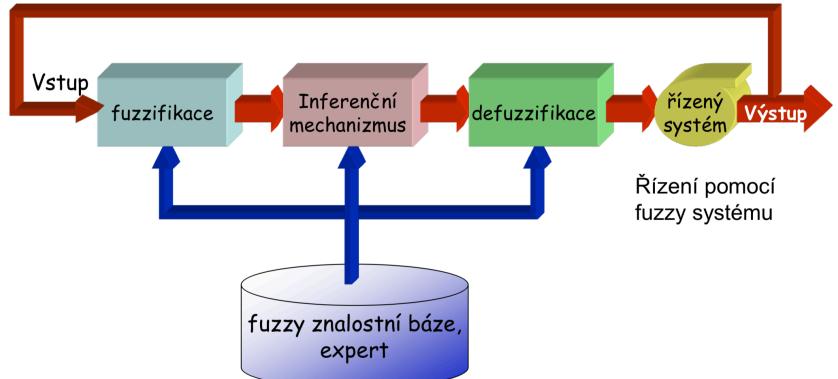
## Funkce příslušnosti

- určuje „jak moc“  $x$  patří do fuzzy množiny  $A$
- různé tvary (trapezoidal, triangle, gaussian,..)



## Operace ve fuzzy logice

- **Negace** – NOT A
  - o  $1 - A$
- **Součin** – A and B
  - o  $\min(A, B)$
  - o  $A \cdot B$
  - o  $\max(0, A+B-1)$
- **Součet** – A or B
  - o  $\max(A, B)$
  - o  $A+B-A \cdot B$
  - o  $\min(A+B, 1)$

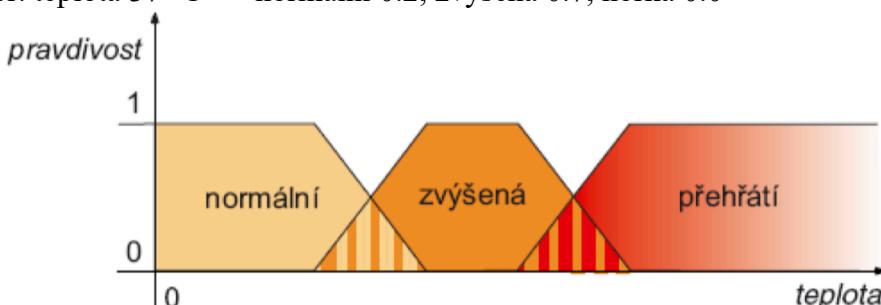


## Fuzzy systém

- hodí se pro modelování a řízení specifických a komplexních procesů

### Fuzzifikace

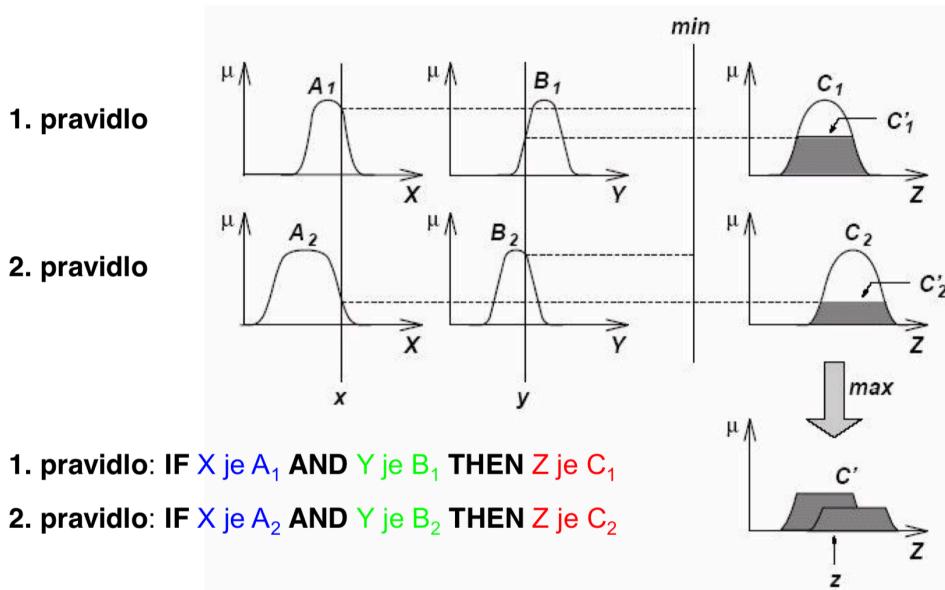
- převod číselné vstupní hodnoty na pravdivostní hodnotu souboru vstupních termů
- např. teplota  $37^{\circ}\text{C} \Rightarrow$  normální 0.2, zvýšená 0.7, horká 0.0



- příklad s autem
  - o auto stáří 15 let  $\Rightarrow$  staré 0.9, nové 0.2
  - o auto najeto 250000  $\Rightarrow$  málo 0.0, středně 0.3, hodně 0.9

### Inferenční mechanizmus

- hodnoty termů  $\Rightarrow$  pravdivostní hodnoty výstupních proměnných
- pravidla ve tvaru:
  - o IF **stáří** je **staré** AND **najeto** je **hodně** THEN **porucha** je **pravděpodobná**

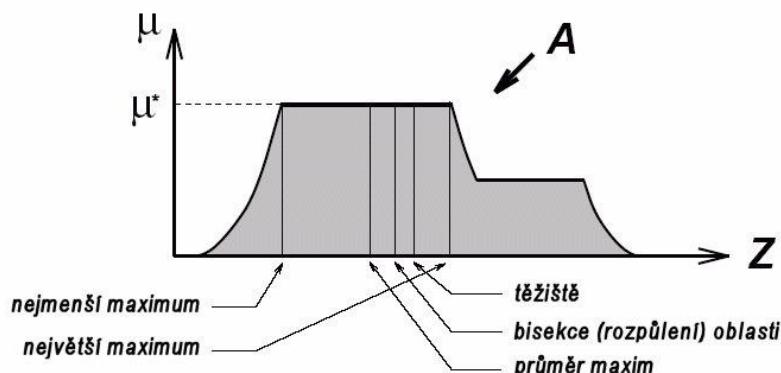


### Typy inferenčních mechanismů

- Mamdani (tady popsaný, výše vyobrazený)
  - o IF  $x$  je malé AND  $y$  je střední THEN  $z$  je velké
  - o Na pravé straně je podmínka vyjádřena jako stupeň příslušnosti proměnné k fuzzy množině
- Sugeno
  - o IF  $x$  je malé AND  $y$  je střední THEN  $z = f(x,y)$
  - o Na pravé straně je podmínka vyjádřena jako funkce proměnných
  - o Nejčastěji se volí funkce  $f$  jako polynom (lineární funkce)  $z = f(x,y) = a.x+b.y$

### Defuzzifikace

- převod výsledku inferenčního mechanismu na výslednou hodnotu
- hledání **těžiště**, největší maximum, nejmenší maximum, průměr maxim,..



### Reprezentace znalostí ve fuzzy systémech

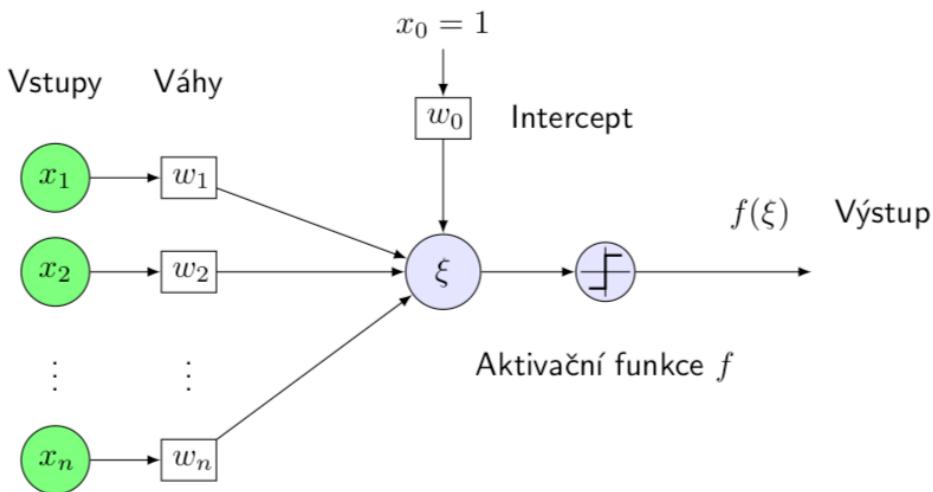
- **funkce příslušnosti** – velikost univerza, tvar,..
- **pravidla** – IF  $a$  is  $x$  AND  $b$  is  $y$  THEN  $c$  is  $z$ 
  - o díky převodu číselných vstupů na slovní ohodnocení se dobře píšou
- Pro nastavení parametrů modelu se často používá neuronová síť

# 22 – Reprezentace znalostí a inference pomocí neuronových sítí

- Biologická inspirace
- klasifikace, regrese, predikce časových řad

## Perceptron

- Nejjednodušší model, který se skládá z 1 umělého neuronu
- Funguje jen pro lineárně separovatelná data (najde pouze lineární fci)
  - o Selže například pro XOR
- Výstup neuronu se získá aplikací nelineární **aktivační funkce**  $f$  na hodnotu **vnitřního potenciálu**  $\xi$  daného součtem vstupů  $x_1, \dots, x_n$  pronásobených příslušnými vahami  $w_1, \dots, w_n$  a interceptu  $w_0$  (**bias**).



## Vnitřní potenciál

- Vstupy  $x = (x_1, \dots, x_n)^T$ , váhy  $w = (w_1, \dots, w_n)^T$ ,  $w_0$  je bias

$$\xi = w_0 + \sum_{i=1}^n w_i x_i = \mathbf{w}^T \mathbf{x} + w_0,$$

## Výstup perceptronu

- Tento výpočet se nazývá **forward pass**

$$\hat{Y} = f(\xi) = f(\mathbf{w}^T \mathbf{x} + w_0),$$

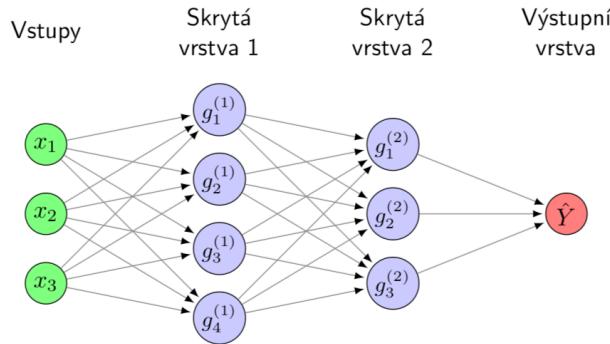
## Inkrementální update vah

- Označuje se jako **backward pass**
- $\hat{Y}$  je predikce,  $Y$  je ground truth,  $\mu$  je learning rate

$$\begin{aligned}\delta w_i &= \eta(Y - \hat{Y}) x_i, \\ w_i &\leftarrow w_i + \delta w_i,\end{aligned}$$

## Multi Layer Perceptron

- Výstupy neuronů z jedné vrstvy tvoří vstupy neuronů do další vrstvy
- V problému XOR dokáže skrytá vrstva zajistit transformaci do souřadného systému, kde jsou již body separabilní.



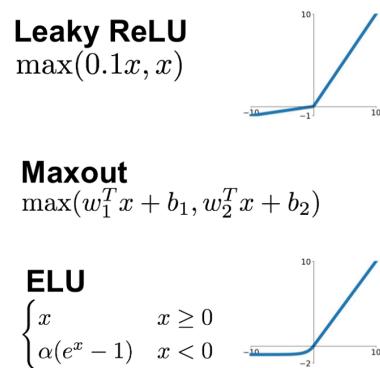
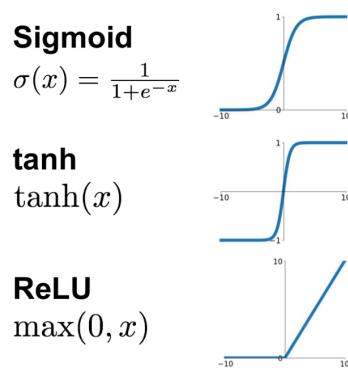
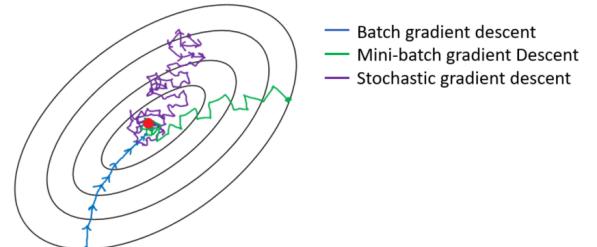
- Třívrstvá NN se dá rozložit jako  $g(x) = g^{(3)} \left( g^{(2)} \left( g^{(1)}(x) \right) \right)$

## Trénování MLP

- K učení lze využít i black-box metody (genetické algoritmy)
- Revoluce v učení NN: **back-propagation** (zpětné šíření chyby)
- Vyžaduje, aby byla celá NN diferencovatelná podle parametrů sítě (vah w) podle kterých minimalizujeme loss funkci:

$$J(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N (Y_i - \hat{Y}_i)^2 = \frac{1}{N} \sum_{i=1}^N (Y_i - g(\mathbf{x}_i))^2$$

- Gradient descend – iterativě upravujeme parametry sítě (váhy) a minimalizujeme tak chybu sítě. Než upravíme váhy, můžeme využít různá množství dat (**batch size**)
  - o Batch training (chyba se napočítá z celé trénovací množiny)
  - o Minibatches (chyba se napočítá z části trénovací množiny)
  - o Online training (chyba se napočítá z 1 samplu)
- **Aktivační funkce**
  - o cílem je dodat nelinearitu do sítě (bez ní je NN lineární regrese bez ohledu na počet vrstev)
  - o pro multiclass klasifikaci SoftMax (predikce pravděpodobnosti, že vstup náleží do třídy)



## Matematický model vícevrstvé neuronové sítě

- Uvažujme  $l$  vrstvou neuronovou síť a označme  $n_1, \dots, n_l$  počty neuronů v jednotlivých vrstvách. Dále označně počet vstupních proměnných jako  $n_0$ .
- Uvažujme  $i$ -ou vrstvu této sítě. Výstup  $j$ -tého neuronu může reprezentovat funkci  $g_j^{(i)} : \mathbb{R}^{n_{i-1}} \rightarrow \mathbb{R}$ , která má na vstupu výstupy  $n_{i-1}$  neuronů z předchozí vrstvy.
- Interně se  $g_j^{(i)}(\mathbf{x})$  opět počítá jako  $f(\mathbf{w}^T \mathbf{x} + w_0)$ , kde  $f$  je aktivační funkce daného neuronu a  $w_0, w_1, \dots, w_n$  jsou jeho váhy.
- $i$ -ou vrstvu této neuronové sítě jako celek pak můžeme chápat jako vícehodnotovou funkci  $\mathbf{g}^{(i)} : \mathbb{R}^{n_{i-1}} \rightarrow \mathbb{R}^{n_i}$ , kde  $\mathbf{g}^{(i)} = (g_1^{(i)}, \dots, g_{n_i}^{(i)})^T$ .
- Celá neuronová síť při dopředném chodu je tedy reprezentována funkcií  $\mathbf{g} : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_l}$ , která vznikne složením jednotlivých vrstev

$$\mathbf{g} = \mathbf{g}^{(1)} \circ \mathbf{g}^{(2)} \circ \dots \circ \mathbf{g}^{(l-1)} \circ \mathbf{g}^{(l)}.$$

- Např. pro  $l = 3$  tak máme

$$\mathbf{g}(\mathbf{x}) = \mathbf{g}^{(3)}(\mathbf{g}^{(2)}(\mathbf{g}^{(1)}(\mathbf{x}))).$$

## Dávkové učení neuronové sítě

- Máme neuronovou síť s parametry  $\mathbf{w} = (w_1, \dots, w_m)^T$  a trénovací data  $(Y_1, \mathbf{x}_1), \dots, (Y_N, \mathbf{x}_N)$ .
- Inicializujeme všechny váhy  $\mathbf{w}$  jako malá náhodná čísla.
- Opakujeme dokud nejsou splněna kritéria zastavení:
  - ▶ Položme  $J(\mathbf{w}) = 0$ .
  - ▶ Pro každou trénovací dvojici  $(Y_i, \mathbf{x}_i)$ :
    - Spočteme  $\hat{Y}_i$  v bodě  $\mathbf{x}_i$ .
    - provedeme přepočet celkové chyby,

$$J(\mathbf{w}) \leftarrow J(\mathbf{w}) + \frac{1}{N} (Y_i - \hat{Y}_i)^2.$$

- ▶ Spočteme gradient

$$\nabla_{\mathbf{w}} J = \left( \frac{\partial J}{\partial w_1}, \dots, \frac{\partial J}{\partial w_m} \right)^T.$$

- ▶ provedeme přepočet vah

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} J.$$



# 28 – Evoluční výpočetní techniky

genetický algoritmus, genetické programování, evoluční programování, evoluční strategie. Genetické operátory (selekce, křížení, mutace)

- metoda pro iterativní optimalizaci
- inspirace evoluční biologií, zejm. pak:
  - genetická dědičnost (Mendel)
  - natural selection, survival of the fittest (Darwin)

## Genetický algoritmus

- navržen jako black-box solver optimalizující binární řetězce (**chromozomy**) délky  $n$
- např. 101001011

### Initialization

- Řetězce mohou být inicializovány buď úplně náhodně, nebo pomocí nějaké heuristiky ke „slibným“ oblastem stavového prostoru

### Evaluation

- Jednotlivé jedince ohodnotíme za pomocí **fitness funkce** a získáme tak číselné ohodnocení jejich performance

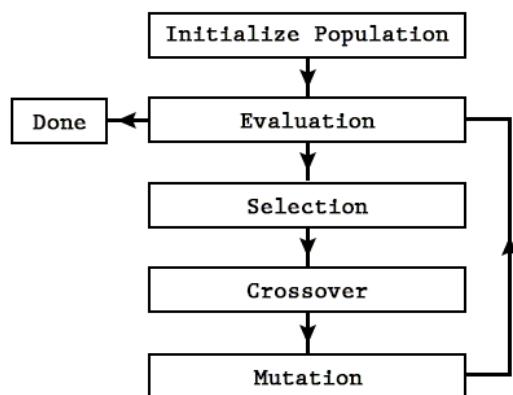
### Selection

- Podobně jako v přírodě, nejlepší jedinci mají největší předat svoje geny dále.
- Za pomocí operátorů selekce vybereme vždy 2 jedince k páření (crossoveru).

### Operátory selekce

- **Roulette wheel selection** (ruletová selekce)
  - PST výběru jedince  $i$  je  $P_i = \frac{\text{fitness jedince } i}{\text{fitness všech jedinců v populaci}}$
- **Tournament selection** (turnajová selekce)
  - Náhodně vybereme k jedinců a vybereme nejlepšího z nich

GENETIC ALGORITHM FLOW CHART



### Crossover (křížení)

- Ke křížení se dostanou 2 jedinci vybraní v předchozím kroku a „smíchají“ svoje geny.
- Výsledkem jsou 2 potomci, každý nese jinou část genomu rodičů.
- Operátory křížení pouze řeší, která část genomu ze kterého rodiče připadne potomkové

### Operátory křížení

Parent #1 1 0 0 1 | 1 0 1 1 1 0 0 1

Parent #2 1 0 1 1 | 1 0 0 1 1 1 0 0

Offspring #1 1 0 0 1 | 1 0 0 1 1 1 1 0 0

Offspring #2 1 0 1 1 | 1 0 1 1 1 1 0 0 1

**Jednobodové křížení**  
(One-Point Crossover)

Parent #1 1 0 0 1 | 1 0 1 1 1 0 0 1

Parent #2 1 0 1 1 | 1 0 0 1 1 1 0 0

Offspring #1 1 0 0 1 | 1 0 0 1 1 1 1 0 1

Offspring #2 1 0 1 1 | 1 0 1 1 1 1 0 0 0

**Dvoubodové křížení**  
(Two-Point Crossover)

Parent #1 1 0 0 1 | 1 0 1 1 | 1 1 0 0 1

Parent #2 1 0 1 1 | 1 0 0 1 | 1 1 1 0 0

Offspring #1 1 0 1 1 | 1 0 1 1 1 1 1 0 1

Offspring #2 1 0 0 1 1 | 1 0 0 1 1 1 1 0 0

**n-bodové křížení**  
(n-Point Crossover)

Parent #1 1 0 0 1 | 1 0 1 1 | 1 1 1 0 0 1

Parent #2 1 0 1 1 | 1 0 0 1 | 1 1 1 0 0

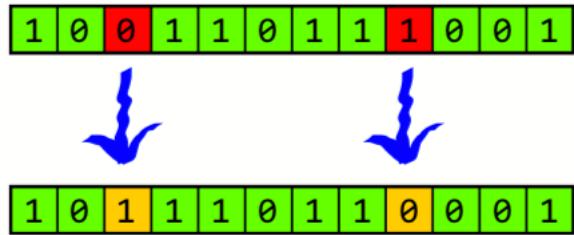
Offspring #1 1 0 1 1 | 1 0 1 1 1 1 1 0 0 1

Offspring #2 1 0 0 1 1 | 1 0 0 1 1 1 1 0 0

**Uniformní křížení**  
(Uniform Crossover)

## Mutation

- Model, který by se skládal jen z genomu rodičů by byl náchylný k uvíznutí v lokálním minimu. Nedokázal by navíc získat jiné konfigurace chromozomů, nežli náhodně vygenerované v inicializaci.
- Mutace je nástroj, který s malou PSTí (0.01% pro každý bit) **přehodí některé bity** v potomkově.

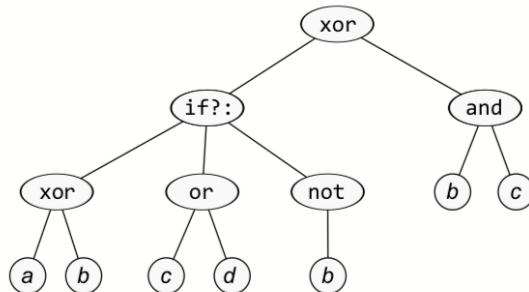


## Genetické programování

genotypem jsou orientované kořenové **stromy**



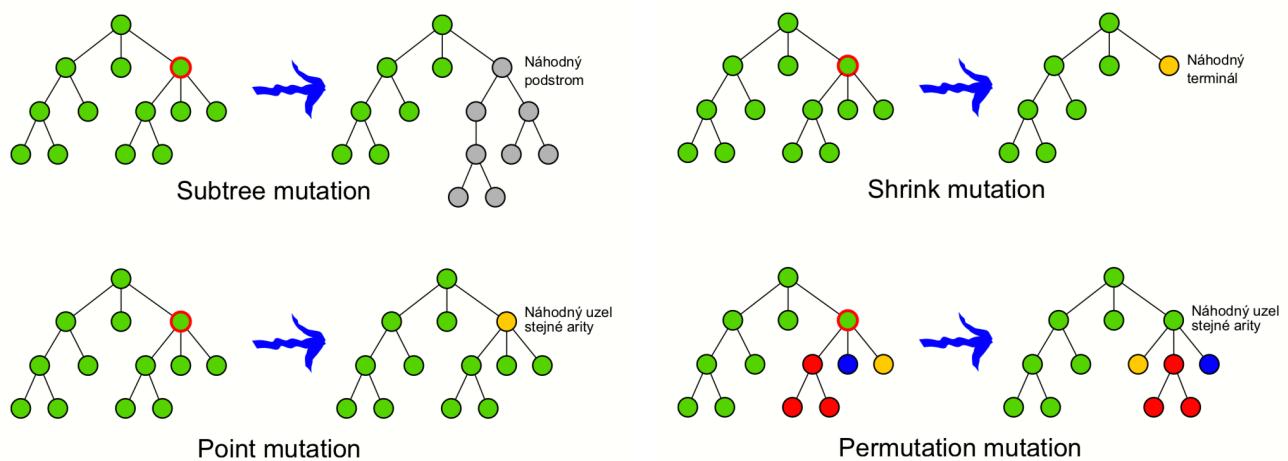
- stromy se v GP skládají z:
  - o **terminálů** (listů) T – vstupy programů (proměnné)
  - o **funkcí** (vnitřních uzlů) F – funkce, aritmetické operace,..



## Inicializace

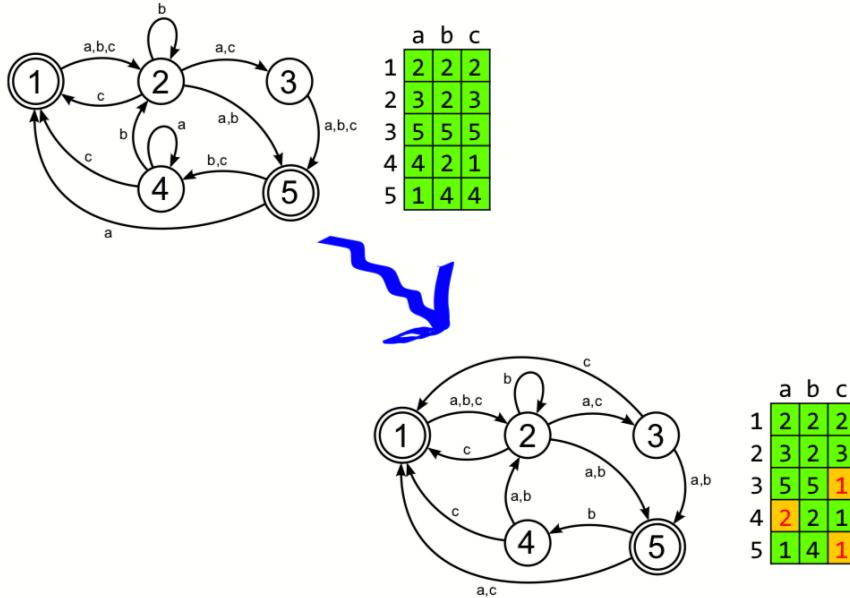
- Vygenerování náhodného stromu o maximální hloubce D
- Více způsobů (stejně hluboké stromy o hloubce D, náhodný výběr pro node hloubce < D – pokud terminál, tak v té hloubce strom zkončí

## Mutace



## Evoluční programování

- Používá **stavové automaty**
- Jenom **mutace**
  - o může jí podléhat počáteční a cílový stav, množina stavů i tabulka přechodů



## Evoluční strategie

- genotypem jsou **vektory reálných čísel** (například float váhy v neuronce se blbě encodují jako binární řetězce kvůli struktuře plavoucí čárky – blbě by se křízily)

**R**

### Selekce

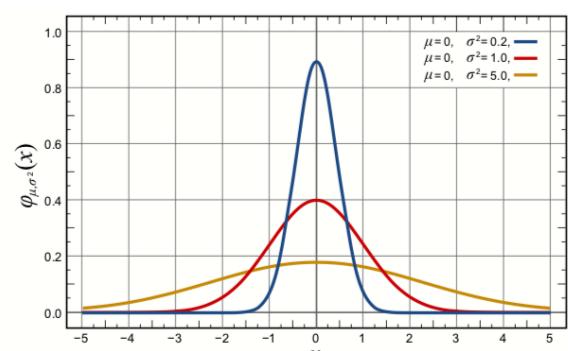
- $(1 + \lambda)$ -ES
  - o Jeden rodič vyprodukuje  $\lambda$  potomků, nejlepší potomek rodičem další generace
- $(\mu + \lambda)$ -ES a  $(\mu, \lambda)$ -ES
  - o  $\mu$  rodičů vyprodukuje  $\lambda$  potomků
  - o  $(\mu + \lambda)$ -ES – vybere  $\mu$  nejlepších z rodičů i potomků
  - o  $(\mu, \lambda)$ -ES – vybere  $\mu$  nejlepších pouze z potomků

### Mutace

- **Gaussovská mutace** – k číslu přičteme náhodnou hodnotu z normálního rozdělení
  - o Změna většinou malá, výjimečně velká
- Pokročilá verze ES mutuje i parametry normálního rozdělení

### Křížení

- Původní verze křížení napoužívala
- Po složkách
  - o Diskrétní – hodnota jednoho z rodičů
  - o Aritmetické – průměr hodnot rodičů
- Přibývá další parametr  $p$  který určuje počet rodičů kteří se podílí na tvorbě potomka





# 30 – Prohledávání herního stromu

algoritmus Minimax, alfa-beta prořezávání

- Hry v **extenzivní formě** – hry vyjádřené stromem (piškvorky, šachy, dáma,..)
- Dvouhrákové **zero-sum hry** (tzn. hráč získá všechno, co protivník ztratí)
- **Utilitní funkce** 2 hráčů nahradíme jednou funkcí  $u: T \rightarrow R$ 
  - Nějaká heuristika (v případě šachů naivní příklad  $u = \#bílých - \#\text{černých}$  figurek – pokud je  $u > 0$ , bílý vyhrává, pokud je  $u < 0$ , černý vyhrává)
  - rychlá, spolehlivá (opravdu koreluje s kvalitou stavu)
- Hráči:
  - **MAX** ( $\Delta$ ) – snaží se maximalizovat utilitní fci
  - **MIN** ( $\nabla$ ) – snaží se minimalizovat utilitní fci
- **Perfektní hra** – hráč volí v každém tahu ideální (bezchybnou) akci
  - Většina her nemožná hrát perfektně (kombinatorická exploze)
    - Využíváme proto heuristiky
  - Prakticky za použití heuristiky prohledáváme herní strom do malé hloubky

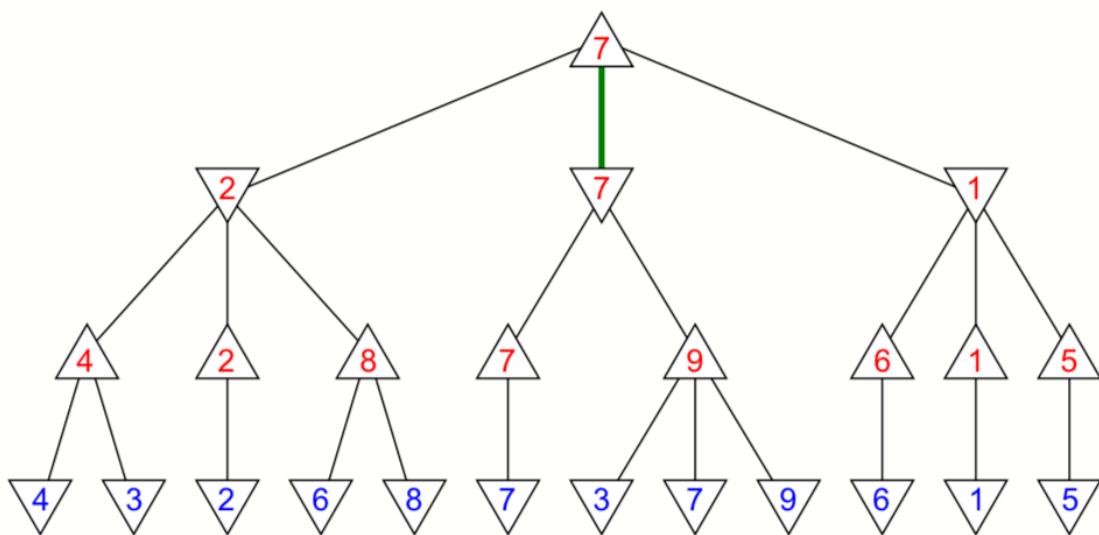
## Minimax algoritmus

- Algoritmus který umožňuje zvolit ideální akci
- Předpokládá, že soupeř hraje ideální hru

### Průběh algoritmu (rekurzivní, do hloubky d)

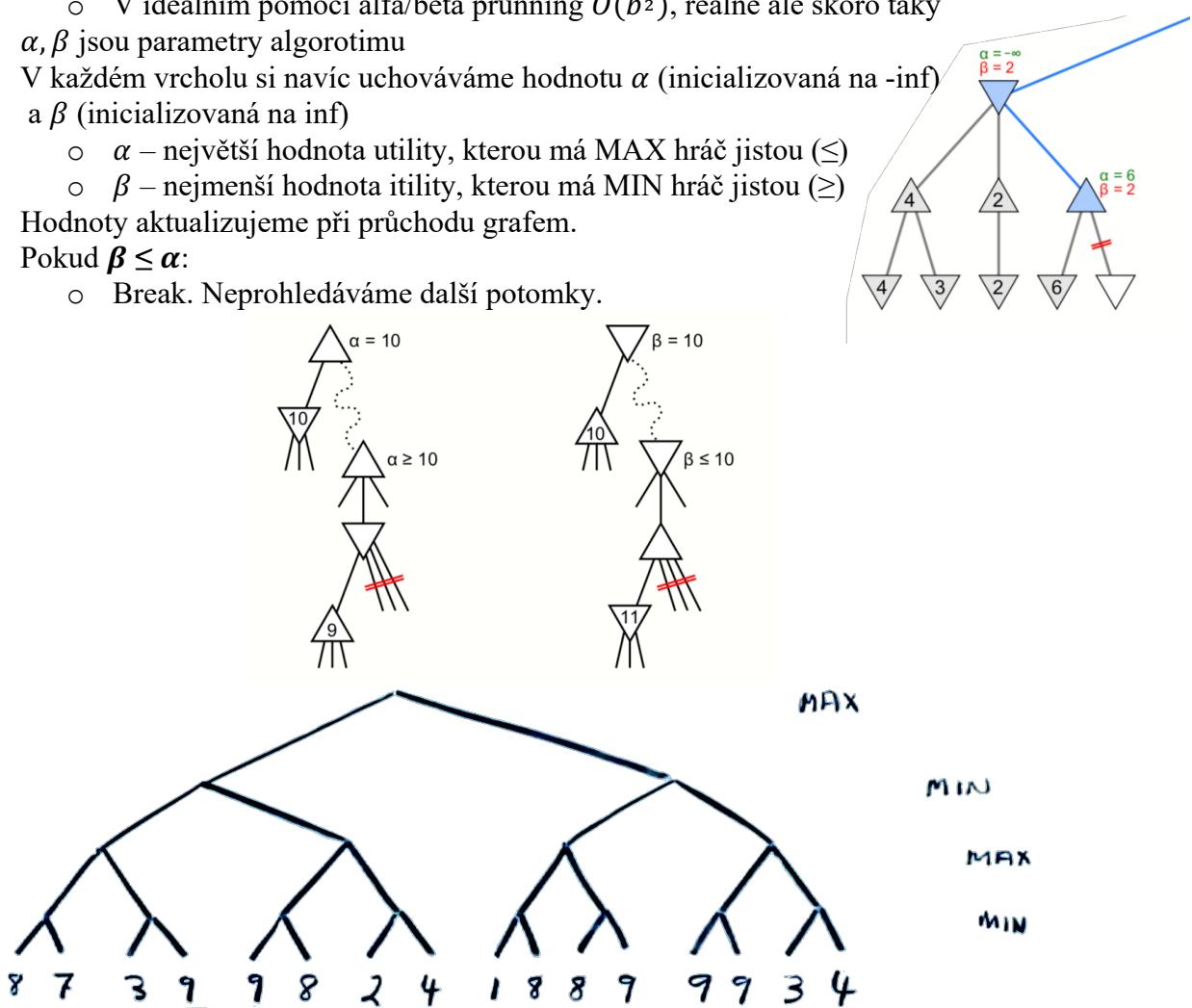
1. Začni v počátečním uzlu  $x_0$  typu MAX
2. Rekurzivně volej potomky vrcholu až do hloubky d
  - Bud' narazíš na terminál, tak se zastav a vrat' terminál. Jinak se zastav v hloubce d kde místo zanorování spočítej utilitní fci.
  - Po návratu hodnot potomků vyber bud' MIN nebo MAX podle typu vrcholu.
  - 3. Po návratu do  $x_0$  vidíš, odkud přišla nejvyšší hodnota a tím směrem se vydej.

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$



## Alfa-beta prořezávání

- Dobrý návod: <https://www.youtube.com/watch?v=l-hh51ncgDI>
- Příklady: <http://people.cs.pitt.edu/~litman/courses/cs2710/lectures/pruningReview.pdf>
- Úspěchy např. Deep Blue porazil Kasparova, AlphaGo (monte carlo tree search)
- MiniMax fuguje dobře jen pro malá d – množství možných stavů roste exponenciálně  
Většina her nemožná hrát perfektně (kombinatorická exploze)
- Alfa-beta prořezávání je jen efektivnější alternativa pro MinMax
  - o Prakticky za použití heuristiky prohledáváme herní strom do malé hloubky
- MinMax má složitost  $O(b^d)$  kde d je hloubka a b branching factor
  - o V ideálním pomocí alfa/beta pruning  $O(b^{\frac{d}{2}})$ , reálně ale skoro taky
- $\alpha, \beta$  jsou parametry algoritmu
- V každém vrcholu si navíc uchováváme hodnotu  $\alpha$  ( inicializovaná na -inf) a  $\beta$  ( inicializovaná na inf)
  - o  $\alpha$  – největší hodnota utility, kterou má MAX hráč jistou ( $\leq$ )
  - o  $\beta$  – nejmenší hodnota utility, kterou má MIN hráč jistou ( $\geq$ )
- Hodnoty aktualizujeme při průchodu grafem.
- Pokud  $\beta \leq \alpha$ :
  - o Break. Neprohledáváme další potomky.



- Řešení: Nebudou se prozkoumávat 4, 7, 8, 10, 15 a 16tý list

