**Umělá inteligence**
**Literatura**

[1] Vladimír Mařík, Olga Štěpánková, Jiří Lažanský a kolektiv: Umělá inteligence, Akademie věd České republiky, Academia Praha
(1) 1993, ISBN 80-200-0496-3
(2) 1997, ISBN 80-200-0504-8
(3) 2001, ISBN 80-200-0472-6
(4) 2003, ISBN 80-200-1044-0
(5) 2007, ISBN 80-200-1470-2
(6) 2013, ISBN 80-200-2267-9
(soubor) ISBN 80-200-0502-1
[2] S. J. Russell, P. Norvig: Artificial Intelligence: A Modern Approach. Global Edition. 4th ed. Pearson, 2020. ISBN-13: 978-0-13-461099-3.
[3] P. H. Winston: Artificial Intelligence, Third Edition, Addison Wesley, Reading, MA, 1992, ISBN 0-201-53377-4.
[4] J. Glenn Brookshear: Computer Science: An Overview (8th Edition), Addison Wesley, 2004, ISBN: 0-321-26971-3.
[5] Wolfgang Ertel: Introduction to Artificial Intelligence. 2nd ed. Springer, 2017. ISBN-13: 978-3319584867.
[6] Charu C. Aggarwal: Neural Networks and Deep Learning. Springer, 2018. ISBN-13: 978-3319944623.
[7] Sandro Skansi: Introduction to Deep Learning. Springer, 2018. https://doi.org/10.1007/978-3-319-73004-2.

**Definice a aplikace** [1 (1)]

3 definice umělé inteligence
Marvina Minského z roku 1967: UI je věda o vytváření strojů nebo systémů, které budou při řešení určitého úkolu užívat takového postupu, který – kdyby ho dělal člověk – bychom považovali za projev jeho inteligence.
Elaine Richové z roku 1991: UI se zabývá tím, jak počítačově řešit úlohy, které dnes zatím zvládají lidé lépe.
Zdeňka Kotka z roku 1983: UI je vlastnost člověkem uměle vytvořených systémů vyznačujících se schopností rozpoznávat předměty, jevy a situace, analyzovat vztahy mezi nimi a tak přijímat účelná rozhodnutí, za pomoci schopností předvídat důsledky těchto rozhodnutí a objevovat nové zákonitosti mezi různými modely nebo jejich skupinami.

Umělá inteligence jako vědní disciplína se postupně formuje v posledních 30 letech jako interdisciplinární průsečík několika zdánlivě různorodých, avšak při bližším zkoumání relevantních vědních disciplín, jakými jsou např. psychologie, neurologie, kybernetika, matematická logika, teorie rozhodování, informatika, teorie her, lingvistika atd.
Mezi vědními disciplínami má poněkud specifické postavení, a to ze dvou důvodů. Za prvé, dosud neexistuje všeobecně přijímaná definice umělé inteligence; za druhé, umělá inteligence neposkytuje jednotící teorie – spíše volně sdružuje různorodé teorie, metody a techniky, které lze úspěšně používat k počítačovému řešení některých složitých úloh rozhodování, plánování, diagnostiky apod.
Umělá inteligence je jednou z nejrychleji se vyvíjejících vědeckých a technických disciplín v historii. Začátek její historie se obvykle datuje rokem 1956. Nicméně již rok 1950 byl pro umělou inteligenci velmi významný – vynikající britský matematik Alan Turing jednak formuloval proslavený test, jednak shromáždil řadu argumentů proti inteligentním strojům a postupně je vyvrátil. V témže roce další velikán vědy John von Neumann vyjádřil své přesvědčení, že v krátké době počítače dosáhnou či dokonce překonají intelektuální schopnosti člověka. Obě osobnosti tak významně přispěly k tomu, že na – tehdy ještě velmi objemné a nepříliš výkonné – číslicové stroje začala odborná veřejnost nahlížet nikoliv jenom jako na mechanicky a slepě fungující kalkulačky, nýbrž v nich začala spatřovat nástroje pro simulaci některých hledisek inteligentního chování.
V létě roku 1956 zorganizoval John McCarthy z MIT poměrně malou konferenci na Dartmouth College, New Hampshire, na kterou byli pozváni přední odborníci zajímající se o mentální schopnosti lidí i strojů. Cílem dartmouthské konference bylo prodiskutovat domněnku, že „každé hledisko učení nebo jakýkoli jiný příznak inteligence může být v principu tak přesně popsán, že může být vyvinut stroj, který ho simuluje“. Dalším výsledkem dartmouthské konference bylo přijetí předpovědí rozvoje umělé inteligence. Bylo předpovězeno, že v roce 1970 počítač:
- bude velmistrem šachu,
- odhalí nové významné matematické teorémy,
- porozumí přirozenému jazyku a bude sloužit jako překladatel,
- bude schopen komponovat hudbu na úrovni klasiků.

Velmi ambiciózně formulovaná předpověď přinesla později díky svému nesplnění jisté zklamání a krizi umělé inteligence, avšak sehrála nesmírně důležitou stimulující roli v prvních měsících a letech „života“ umělé inteligence. I jednoduché výsledky jednoduchých experimentů byly nekriticky zobecňovány a vytvářely se nové přehnané až fantastické hypotézy o možnostech napodobovat myšlení. S přibývajícími léty bylo čím dál tím zřejmější, že cílů předpovídaných pro rok 1970 nebude dosaženo ani s mnohaletým zpožděním. V druhé polovině 60. let začíná převládat skepse, rozčarování z poměrně chudých výsledků vědní disciplíny. Říká se, že umělá inteligence vstoupila do „doby ledové“, která trvala až do druhé poloviny 70. let.
Přestože systémy jako STRIPS a PLANNER nenalezly širšího praktického uplatnění, jde o systémy velice významné z hlediska vývojového. Zejména se prokázalo (spolu s pokusy o praktické využití Robinsonova rezolučního principu),

že univerzálnost systémů a schopnost odvozovat bez uvažování specifik problému, po níž komunita v oblasti umělé inteligence tolik prahla, je hlavní slabinou univerzálních systémů. Týmy výzkumníků si postupně začínají uvědomovat, že obecné metody jsou příliš slabé pro řešení vysoce specializovaných úloh, které jsou však na druhé straně efektivně řešeny specialisty-experty.
Ukázalo se, že rozhodující pro vysokou efektivitu systémů umělé inteligence jsou použité znalosti, zatímco obecný formální aparát pro řešení úloh poskytuje pouze nástroj pro využívání znalostí a hraje tedy druhořadou roli. Ira Goldstein a Seymour Papert v roce 1977 uvádějí, že: „Základním problémem umělé inteligence není odhalení několika efektivních obecných technik, ale spíše otázka, jak reprezentovat velké množství znalostí ve tvaru, který by dovolovat jejich efektivní využívání a interakci.“ Tak začaly vznikat expertní systémy.
Expertní systémy jsou počítačové programy simulující rozhodovací činnost specialistů (expertů) při řešení složitých úloh rozhodování a využívání vhodně zakódovaných speciálních znalostí převzatých od expertů s cílem dosahovat ve zvolené problémové oblasti kvality rozhodování na úrovni experta. Znalosti převzaté od experta (včetně znalostí neurčitých) tvoří tzv. bázi znalostí, která je obvykle implementována, spravována a udržována jako samostatný soubor. Podstatnou součástí expertních znalostí jsou znalosti nepřesné, neurčité, vágní. Proto je teorie zpracování neurčitosti ve znalostech i v datech velmi důležitou partií teorie umělé inteligence.
Efektivita expertního systému je rozhodujícím způsobem ovlivňována kvalitou báze znalostí. Tvorba báze znalostí je obvykle dlouhodobým procesem získávání znalostí od experta a jejich kódování do tvaru, akceptovatelného příslušným vyvozovacím neboli inferenčním mechanismem. Tohoto procesu se účastní jak expert ve zvolené problémové oblasti, tak i specialista pro tvorbu báze znalostí, tzv. znalostní inženýr. Rozvoj expertních systémů lze sledovat od přelomu 70. a 80. let, a to v neposlední řadě díky jejich prokazatelnému ekonomickému dopadu. Jde v podstatě o jediný v praxi využitelný a využívaný výsledek výzkumu z oblasti umělé inteligence.

Aplikační oblasti umělé inteligence
V 60. a 70. letech byla velmi důležitou součástí UI robotika, viz systémy SHRDLU, PLANNER, STRIPS a SHAKEY. Dnes jsou hlavním obsahem robotiky úlohy plánování v reálném čase. Robotika je chápána jako inteligentní vazba od vnímání k akci. Úloha automatického vnímání (percepce) pro potřeby robotiky se poměrně brzy rozpadla na samostatné oblasti zkoumání: na počítačové vidění, automatické rozpoznávání a porozumění přirozenému jazyku a na zpracování taktilní (dotykové) informace.
Počítačové vidění zahrnuje komplexní soubor metod a technik pro zpracování a porozumění dvojrozměrným vizuálním obrazům (snímkům), které reprezentují dvojrozměrnou či dokonce trojrozměrnou scénu. Jde tedy o úlohy snímání a digitalizace snímku. V případě, že dvojrozměrný snímek reprezentuje trojrozměrnou scénu, hovoří se často o „porozumění scéně“.
Dlouhodobým cílem automatického porozumění přirozenému jazyku je komunikace s počítačem v přirozeném jazyce, a to jak v jeho psané, tištěné i mluvené formě. Systém s uvedenými vlastnostmi by měl být schopen přijmout a rozpoznat člověkem vyslovenou (napsanou) zprávu, porozumět jí, připravit a realizovat „rozumnou“ odpověď, tj. vykonat požadovanou akci nebo napsat či syntetizovat „umělým“ hlasem odpověď.
Jako jistou extrapolaci robotiky chápeme dnes stále intenzivnější snahy o rozvoj tzv. subsystémů „virtuální reality“. Jde o komplikované, avšak dosti přesné počítačové modely prostředí, které například umožňují simulovat či emulovat vjemy, pohyby či akce člověka či stroje v tomto prostředí. Takovéto modelování je využíváno například v trenažérech, při simulaci práce člověka v kosmickém prostoru, při navrhování fyzické kooperace robotů atd., tedy tam, kde kvalitní, obvykle trojrozměrná, počítačová simulace může nahradit velice drahé (a často nerealizovatelné) fyzikální modely či experimenty v reálném prostředí. Metody umělé inteligence, a to jak již „klasické“ metody reprezentace znalostí, prohledávání stavového prostoru, plánování apod., či nejnovější speciální techniky (jako techniky neuronových sítí, genetických algoritmů atd.) bývají zcela přirozenou součástí systémů virtuální reality.
Historicky velmi důležitou aplikační oblastí UI je lékařská diagnostika i terapeutika. Zde bývají znalosti vyjádřeny často explicitně, bývají neurčité, nejisté a mívají typicky expertní charakter. Proto zejména expertní systémy zde nalézaly při svém vývoji četnou inspiraci. Navíc, lékařské diagnostice bývají k dispozici rozsáhlé soubory dat (kartotéky), tedy důležitý „výchozí materiál“ pro rozvoj induktivních metod strojového učení. V širším měřítku se začínají uplatňovat metody kvalitativního modelování.
V posledních letech se do popředí dostává další významná aplikační oblast UI – počítačem podporovaná výroba (CIM – Computer Integrated Manufacturing). Cílem systému CIM je integrovat všechny procesy i agendy spojené s průmyslovou výrobou pomocí sítě (typicky různorodých) počítačů. Jde o automatizaci a integraci všech technických, manažerských a obchodních aktivit podniku. Bez znalostí, a tedy bez znalostně orientovaných přístupů nelze poměrně náročného globálního cíle CIMu dosáhnout.
Zcela zvláštní postavení mezi aplikačními oblastmi umělé inteligence má softwarové inženýrství. Tato disciplína na straně jedné bezprostředně využívá metod umělé inteligence k automatizované tvorbě, integraci a údržbě složitých softwarových systémů. Přispívá především k inteligentnímu navrhování architektury rozsáhlých programů a k integraci mnohdy různorodých programových celků. Samostatnou perspektivní partii, která je v úplném počátku své existence, představuje automatizované programování, především automatizovaná syntéza programů. Na straně druhé, přesně v průsečíku UI a softwarového inženýrství se nalézá znalostní inženýrství, které lze považovat za organickou součást obou disciplín. Znalostní inženýrství vneslo zcela novou metodologii do tvorby rozsáhlých softwarových produktů. Tato metodologie se úspěšně uplatňuje všude tam, kde specifikace úloh, které mají být naprogramovány, není a nemůže být zcela přesná, tedy jde o specifikace vágní, vyjadřované přibližnou formulací požadovaného výkonu programu

(performance-based specification). V takových případech exaktní ověřování programů není možné a nahrazuje ho testování postupně zdokonalovaných prototypů založených na reálných datech.

**Řešení problémů**

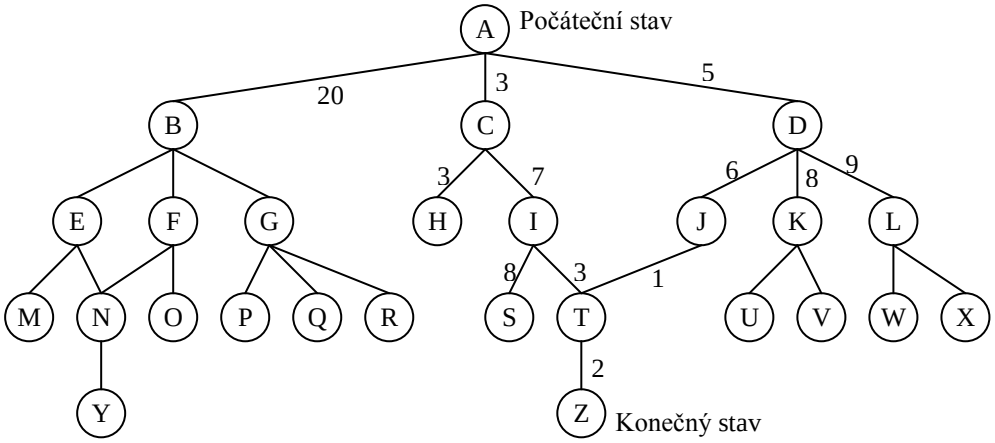
**Definice problému:**
Počáteční stav
Koncový stav – Po každé akci testovat, zda byl již dosažen.
Možné akce – operátor měnící stav na stavy následné
Stavový prostor – množina všech stavů dosažitelných z počátečního stavu libovolnou posloupností akcí
Cesta ve stavovém prostoru je řešením.
Cesta má cenu rovnou sumě cen jednotlivých akcí. Její cena se obvykle značí *g*.

**Příklady problémů:**
nalezení trasy na mapě [2 str. 95-99],
směrování dat v počítačových sítích,
automatické hledání v jízdních a leteckých řádech,
sestavování rozvrhů (kdo-kde-kdy)
8-puzzle, 15-puzzle [1 (1) str. 33, 36, 45, 2 str. 63, 101], (Analýza řešení 8-puzzle: http://www.geocities.com/csmba/)
obarvení mapy 3 barvami [2 str. 105],
problém 8 královen [2 str. 64],
kryptoaritmetika [2 str. 65],
odměřování vody pomocí nádob o různých objemech [1 (1) str. 33, 36, 40, 49],
misionáři a kanibalové [2 str. 67, 3 str. 628],
farmář, liška, husa a zrní [3 str. 16].

**Prohledávání stavového prostoru**
**neinformované (uninformed or blind search)**
– Neznáme počet kroků z počátečního do koncového stavu ani jejich cenu.
– Například máme jít z města A do města B a neznáme, kterým směrem leží město B od města A.
**– do šířky (breadth-first search)**
**– do hloubky (depth-first search)**
**– s omezenou hloubkou (depth-limited search)**
– Například když je na mapě 20 měst, víme že nejkratší cestu z města A do města B není nutné vést přes víc než 18 měst.
**– s iterativním prohlubováním (iterative deepening search)**
– Často nevíme, jaký určit limit pro hloubku prohledávání, a proto můžeme vyzkoušet všechny hloubky počínaje nulovou.
– Je vhodné pro velké stavové prostory s neznámou hloubkou, ve které se nachází řešení.
**– obousměrné (bidirectional search)**
– Hledáme současně z počátečního a cílového stavu. Musíme si pamatovat všechny navštívené uzly z jednoho konce. Hledání skončí, když na druhém konci najdeme uzel, který si pamatujeme z prvního konce.
**informované (informed or heuristic search)**
– Známe všechny prvky definice problému včetně cen akcí, tj. přechodů mezi stavy.
– Například máme jít z města A do města B a známe, kterým směrem leží město B od města A, nebo máme dokonce mapu.
**– gradientní algoritmus, hill-climbing algorithm, greedy search** – Minimalizuje cenu dosažení cíle. Odhad ceny se počítá **heuristickou funkcí** značenou jako *h*. Vždy expanduje ten uzel, který byl dosud vyhodnocen pomocí funkce *h* jako nelepší, a vyhodnocuje jeho následovníky. Nejlepší z následovníků je vybrán k expanzi. „Rodič“ stejně jako „sourozenci“ tohoto uzlu jsou okamžitě zapomenuti – algoritmus uchovává v paměti jen právě rozvíjený uzel. Prohledávání je zastaveno, když je dosaženo stavu, který má „lepší“ ohodnocení pomocí funkce *h* než jeho následovníci. Čistě gradientní strategie má základní nevýhodu všech gradientních metod – může skončit jak v globálním, tak i v pouze lokálním extrému. Navíc, vzhledem k tomu, že se neuchovává historie prohledávání, není vyloučen ani pohyb po nekonečně dlouhé cestě (zacyklení).
**– algoritmus uspořádaného prohledávání, best-first search** – Vznikl rozšířením gradientního algoritmu o paměť. Tato paměť se skládá ze seznamů OPEN a CLOSED, ve kterých jsou trojice ze jména uzlu, hodnoty *h* a jména rodičovského uzlu.
– **A* search** – Minimalizuje součet ceny dosud ušlé cesty *g* a odhadu vzdálenosti cíle *h*.
Při prohledávání stavového prostoru se stává, že se můžeme vrátit do stavu, ve kterém jsme už byli. Proto je vhodné si pamatovat již navštívené stavy.

**Algoritmy prohledávání**

**Hypotetický stavový prostor s cenou cest** [1 (1) str. 44]




**Algoritmus prohledávání do šířky** [1 (1) str. 39]

1. Zapiš počáteční stav do seznamu OPEN, seznam CLOSED je prázdný. Je-li počáteční stav současně stavem cílovým, ukonči prohledávání.
2. Pokud je seznam OPEN prázdný, řešení neexistuje, ukonči prohledávání.
3. Vymaž první stav (označíme jej *i*) v seznamu OPEN a zapiš tento stav do seznamu CLOSED.
4. Expanduj stav *i*. Pokud tento stav nemá následovníky nebo všichni následovníci byli již expandování (tj. jsou v seznamu CLOSED), pokračuj krokem č. 2.
5. Zapiš všechny následovníky stavu *i*, kteří nejsou v seznamu OPEN nebo CLOSED, na konec seznamu OPEN.
6. Pokud některý z následovníků stavu *i* je cílovým stavem, řešení bylo nalezeno, ukonči prohledávání. Jinak pokračuj krokem č. 2.

**Zpracování hypotetického stavového prostoru algoritmem prohledávání do šířky** [1 (1) str. 39]

1. OPEN (A), CLOSED (0)
2. OPEN (B, C, D), CLOSED (A)
3. OPEN (C, D, E, F, G), CLOSED (A, B)
4. OPEN (D, E, F, G, H, I), CLOSED (A, B, C)
5. OPEN (E, F, G, H, I, J, K, L), CLOSED (A, B, C, D)
6. OPEN (F, G, H, I, J, K, L, M, N), CLOSED (A, B, C, D, E)

atd. až do nalezení stavu Z nebo do okamžiku, kdy je seznam OPEN prázdný.

**Algoritmus prohledávání do hloubky (s omezením na hloubku prohledávání)** [1 (1) str. 39]

1. Zapiš počáteční stav do seznamu OPEN, seznam CLOSED je prázdný. Je-li počáteční stav současně stavem cílovým, ukonči prohledávání.
2. Pokud je seznam OPEN prázdný, řešení neexistuje, ukonči prohledávání.
3. Vymaž první stav (označíme jej *i*) v seznamu OPEN a zapiš tento stav do seznamu CLOSED.
4. Pokud se hloubka uzlu *i* rovná maximální přípustné hloubce, pokračuj krokem č. 2.
5. Expanduj stav *i*. Pokud tento stav nemá následovníky nebo všichni následovníci byli již expandování (tj. jsou v seznamu CLOSED), pokračuj krokem č. 2.
6. Zapiš všechny následovníky stavu *i*, kteří nejsou v seznamu CLOSED, na začátek seznamu OPEN.
7. Pokud některý z následovníků stavu *i* je cílovým stavem, řešení bylo nalezeno, ukonči prohledávání. Jinak pokračuj krokem č. 2.

**Zpracování hypotetického stavového prostoru algoritmem prohledávání do hloubky** [1 (1) str. 40]

1. OPEN (A), CLOSED (0)
2. OPEN (B, C, D), CLOSED (A)
3. OPEN (E, F, G, C, D), CLOSED (A, B)
4. OPEN (M, N, F, G, C, D), CLOSED (A, B, E)
5. OPEN (N, F, G, C, D), CLOSED (A, B, E, M)
6. OPEN (Y, F, G, C, D), CLOSED (A, B, E, M, N)
7. OPEN (F, G, C, D), CLOSED (A, B, E, M, N, Y)
8. OPEN (O, G, C, D), CLOSED (A, B, E, M, N, Y, F)
9. OPEN (G, C, D), CLOSED (A, B, E, M, N, Y, F, O)
10. OPEN (P, Q, R, C, D), CLOSED (A, B, E, M, N, Y, F, O, G)

atd. až do nalezení stavu Z nebo do okamžiku, kdy je seznam OPEN prázdný.

**Algoritmus uspořádaného prohledávání** [1 (1) str. 43]

1. Počáteční stav zapiš do seznamu OPEN, seznam CLOSED je prázdný.
2. Pokud je seznam OPEN prázdný, řešení neexistuje, ukonči prohledávání.

3. Ze seznamu OPEN vyber stav *i* s nejmenší hodnotou *f*(*i*). V případě většího počtu stavů se stejnou minimální hodnotou *f*(*i*) prověř, zda některý z těchto stavů není stavem cílovým, v takovém případě jej vyber; jinak vyber mezi stavy se stejnou minimální hodnotou *f*(*i*) libovolně.
4. Vymaž stav *i* ze seznamu OPEN a zařaď jej do seznamu CLOSED.
5. Je-li stav *i* cílovým stavem, řešení je nalezeno, ukonči prohledávání.
6. Expanduj stav *i*; pro každého následovníka *j* stavu *i* vypočítej hodnotu *f*(*j*). Pokud stav *j* není ani v seznamu OPEN ani v seznamu CLOSED, zařaď jej do seznamu OPEN. Pokud je stav *j* již v seznamu OPEN nebo CLOSED, avšak s ohodnocením větším než právě vypočtené *f*(*j*), změň jeho ohodnocení na *f*(*j*), změň jméno rodičovského uzlu v zápisu uzlu a zařaď ho do seznamu OPEN.
7. Pokračuj krokem č. 2.

**Zpracování hypotetického stavového prostoru algoritmem uspořádaného prohledávání** [1 (1) str. 43]
1. OPEN (A-0-0), CLOSED (0)
2. OPEN (C-3-A, D-5-A, B-20-A), CLOSED (A-0-0)
3. OPEN (D-5-A, H-6-C, I-10-C, B-20-A), CLOSED (A-0-0, C-3-A)
4. OPEN (H-6-C, I-10-C, J-11-D, K-13-D, L-14-D, B-20-A), CLOSED (A-0-0, C-3-A, D-5-A)
5. OPEN (I-10-C, J-11-D, K-13-D, L-14-D, B-20-A), CLOSED (A-0-0, C-3-A, D-5-A, H-6-C)
6. OPEN (J-11-D, K-13-D, T-13-I, L-14-D, S-18-I, B-20-A), CLOSED (A-0-0, C-3-A, D-5-A, H-6-C, I-10-C)
7. OPEN (T-12-J, K-13-D, L-14-D, S-18-I, B-20-A), CLOSED (A-0-0, C-3-A, D-5-A, H-6-C, I-10-C, J-11-D)
8. OPEN (Z-13-T, K-13-D, L-14-D, S-18-I, B-20-A), CLOSED (A-0-0, C-3-A, D-5-A, H-6-C, I-10-C, J-11-D, T-12-J)

Stop – cílový stav byl nalezen.
Díky záznamu jmen rodičovských uzlů lze zrekonstruovat nejlevnější cestu A-D-J-T-Z

**Hry** [1 (1) str. 58-64, 2 str. 129-148, 3 str. 101-118]
*α-β* Procedure [2 str. 132]

**inputs**: *state* – current state in game,
*game* – game description,
*α* – the best score for Maximizer along the path to state,
*β* – the best score for Minimizer along the path to state,

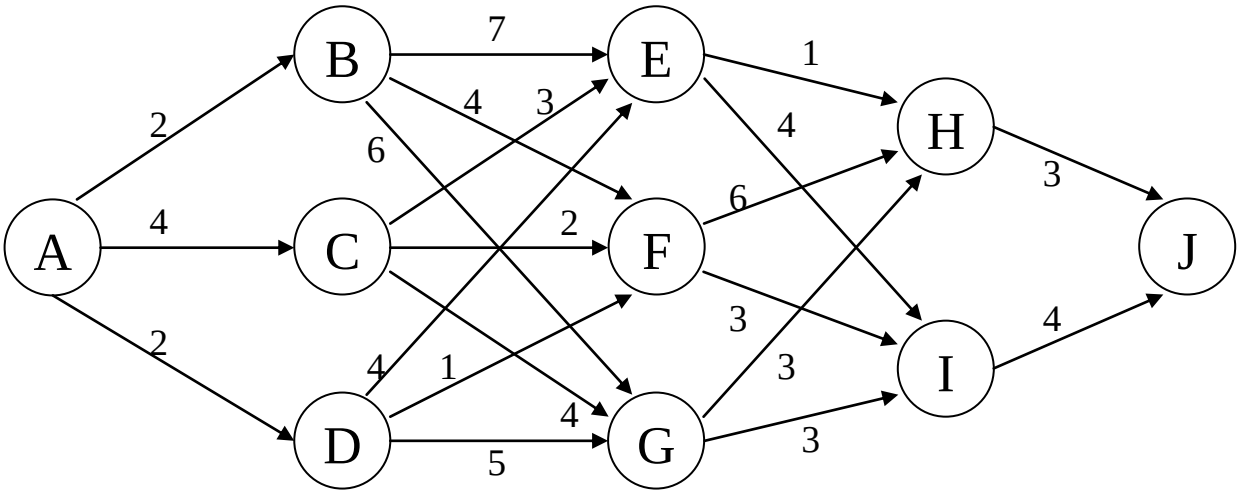
**function** MaxValue (*state*, *game*, *α*, *β*) **returns** the minimax value of *state*
**if** CutOffTest (*state*) **then return** Eval (*state*)
**for each** *s* **in** Successors (*state*) **do**
*α* = Max (*α*, MinValue (*s*, *game*, *α*, *β*))
if *α* >= *β* **then return** *β*
**end**
**return** *α*

**function** MinValue (*state*, *game*, *α*, *β*) **returns** the minimax value of *state*
**if** CutOffTest (*state*) **then return** Eval (*state*)
**for each** *s* **in** Successors (*state*) **do**
*β* = Min (*β*, MaxValue (*s*, *game*, *α*, *β*))
if *β* <= *α* **then return** *α*
**end**
**return** *β*

ALPHA-BETA procedure is started on the root node with an *α* value of -∞ and *β* value of +∞. ALPHA-BETA then calls itself recursively with a narrowing range between the *α* and *β* values. [3 str. 110]

**Složitost úloh (Teorie složitosti)**
Úplné prohledávání stavového prostoru je často nemožné provést z důvodu příliš velkého počtu uzlů, které v prostoru jsou. Teorie informatiky proto rozlišuje úlohy polynomiální a nepolynomiální, které se dále dělí na NP úlohy, NP úplné úlohy a NP těžké úlohy. V testu bude požadována schopnost tyto 4 typy úloh rozlišit. Více se dočtete v [1 (3) str. 262, 2 str. 11 – 12, 851 – 853, 4 str. 482 – 484] a v přednáškách z předmětu Počítače I (kapitola 2.2.5.2) na http://multiedu.tul.cz/~dana.nejedlova/multiedu/Prednask.pdf.

**Dynamické programování**
Jednou z možností, jak zredukovat prohledávaný prostor, je dynamické programování. [2 str. 502]. Následující úloha je převzata z http://mat.gsia.cmu.edu/classes/dynamic/node3.html#SECTION00030000000000000000.




Máme najít nejkratší cestu z uzlu A do uzlu J. Délka cest je vyjádřena čísly na spojích mezi uzly.
Řešení pomocí dynamického programování:
Uzel A je výchozí.
V uzlech B, C a D nic neřešíme, protože do nich vede jen jediná cesta.
V ostatních uzlech vypočteme, ze kterého předchozího uzlu jsme se do nich dostali nejkratší cestou a tento uzel a dosavadní nejkratší délku cesty si zapamatujeme. Po dosažení cíle využijeme zapamatované uzly pro rekonstrukci nejkratší cesty.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| E | 2 + 7 = 9 | F | 2 + 4 = 6 | G | 2 + 6 = 8 | H | 6 + 1 = 7 (E) | I | 6 + 4 = 10 | J | 7 + 3 = 10 (H) |
| | 4 + 3 = 7 | | 4 + 2 = 6 | | 4 + 4 = 8 | | 3 + 6 = 9 | | 3 + 3 = 6 (F) | | 6 + 4 = 10 (I) |
| | 2 + 4 = 6 (D) | | 2 + 1 = 3 (D) | | 2 + 5 = 7 (D) | | 7 + 3 = 10 | | 7 + 3 = 10 | | |

Nejkratší cesta: J-H-E-D-A nebo J-I-F-D-A.
Stejným typem příkladu je řešení úlohy č. 10 a 11 v souboru http://multiedu.tul.cz/~dana.nejedlova/multiedu/AI-Tutor.pdf uvedené v materiálu http://www.cambridge.org/resources/0521882672/7934_kaeslin_dynpro_new.pdf.
Jednou z aplikací dynamického programování je algoritmus porovnání dvou vektorů metodou Dynamic Time Warping neboli algoritmem Minimal Edit Distance někdy zvaným Levenshtein distance.

**Metodologie trénovacích a testovacích množin**
V umělé inteligenci jde často o nalezení pravidelností v pozorovaných datech. Výsledkem je model, pomocí kterého lze například vypočítat výstupní hodnoty ze vstupních pozorovaných hodnot. Model je nějaká funkce obsahující zpravidla velké množství parametrů, které byly natrénovány podle pozorovaných dat. Příkladem modelu je rozhodovací strom, bayesovská síť nebo neuronová síť, vše z toho bude v tomto předmětu vyučováno.
Účelem modelu je zpravidla předvídat hodnoty podle aktuálně pozorovaných dat. Před tím potřebujeme zhodnotit přesnost modelu. To se dělá tak, že se nasbírají data vstupní i výstupní tak, jak byla pozorována. Potom se data rozdělí na dvě části zvané **trénovací** a **testovací data.** Podle **trénovacích dat** se natrénuje model a na **testovacích datech** se vyhodnotí jeho přesnost.
Při některých formách trénování modelu (například u neuronové sítě) se přesnost modelu zvyšuje postupně. Potom je dobré průběžně sledovat jak přesnost modelu na **trénovacích** tak na **testovacích datech**. Může se totiž stát, že přesnost modelu na **testovacích datech** nejdříve stoupá a potom klesá. To je známka toho, že model začíná být příliš věrný **trénovacím datům** tak, že zachycuje jejich specifika, která nejsou přítomná v **testovacích datech.** V okamžiku, kdy se začne přesnost modelu na **testovacích datech** zhoršovat, by se mělo ukončit trénování. Další funkcí **testovacích dat** je tedy zabránění takzvanému **přetrénování** (**overfitting**) modelu. Jinou doporučovanou metodou jak zabránit přetrénování je snížit počet trénovaných parametrů modelu, když je to možné.
U některých úloh, jako je například rozpoznávání řeči, se trénuje několik modelů podle **trénovacích dat** a potom se vybere ten z nich, který dosáhl nejlepšího výsledku na **testovacích datech**. Tím se ale **testovací data** tak trochu zatáhnou do procesu modelování, takže poctivá prezentace výsledku výzkumu je možná pouze s použitím ještě jiných dat – skutečných **testovacích dat**. Ta původní **testovací data** použitá pro výběr nejlepšího modelu jsou potom nazývána **vývojová** nebo **validační data** (**development test set**, **devtest set, validation set**).
https://www.quora.com/Why-do-we-have-separation-of-data-into-training-held-out-and-test-data

**Logika**

Úlohou logiky v umělé inteligenci je převést fakta na formalizované výroky, se kterými se dá automatizovaně operovat tak, že se z počáteční množiny výroků postupně vyvozují nová fakta. Prvním známým formálním systémem takové činnosti byly Aristotelovy sylogismy pro dedukci. (Například: Kapr je ryba. Všechny ryby žijí ve vodě. A proto kapr žije ve vodě.)

**Výroková logika (Propositional Calculus)**
http://courses.cs.vt.edu/~cs1104/BuildingBlocks/Chapter4.128.htm
Řešení úloh pomocí tabulek s pravdivostními hodnotami
Král má dceru, kterou chce vdát, ale musí si vybrat ze tří nápadníků. Král proto připraví tři hrnky a do jednoho z nich dá lístek se jménem své dcery. Nápadník, který vybere hrnek se jménem jeho dcery, ji získá za manželku. Zlatý hrnek nese nápis "V tomto hrnku je jméno", stříbrný hrnek nese nápis "V tomto hrnku není jméno" a na olověném hrnku je napsáno "Stříbrný hrnek neobsahuje jméno". Král řekl nápadníkům, že právě jeden z nápisů je nepravdivý. Který hrnek byste zvolili?
Pravdivostní tabulka:

| ZLATÝ hrnek | | STŘÍBRNÝ hrnek | | OLOVĚNÝ hrnek | |
|---|---|---|---|---|---|
| Umístění jména | V tomto hrnku je jméno | Umístění jména | V tomto hrnku není jméno | Umístění jména | Stříbrný hrnek neobsahuje jméno |
| X | PRAVDA | | PRAVDA | | PRAVDA |
| | NEPRAVDA | X | NEPRAVDA | | NEPRAVDA |
| | NEPRAVDA | | PRAVDA | X | PRAVDA |

Který řádek obsahuje právě jedno nepravdivé tvrzení?

Výroková logika jako obecný soubor pravidel správného usuzování vznikla z potřeb matematiky, přírodních věd i rétoriky již ve 4. století před n. l. ve starověkém Řecku.

**Základní pojmy**

**Výrok** je každá oznamovací věta, o jejíž pravdivosti lze jednoznačně rozhodnout. Výrok má pravdivostní hodnotu pravda nebo nepravda.

**Hypotéza** (domněnka) je výrok, o jehož pravdivosti momentálně neumíme rozhodnout.

**Výroková forma** je věta obsahující proměnné. Stane se výrokem teprve při dosazení konkrétních hodnot za proměnné.

**Kvantifikovaný výrok** vymezuje počet objektů, o nichž vypovídá.

**Obecný kvantifikovaný výrok** přisuzuje určitou vlastnost všem uvažovaným objektům bez výjimky. Obsahuje slova: *všichni*, *každý*, *všechno*, *žádný*, *nikdo*, *nic*, apod. Symbolicky se zapisuje jako $\forall x[T(x)]$: Tvrzení *T* platí pro každé *x*.

**Existenční kvantifikovaný výrok** vypovídá o tom, že existuje alespoň jeden objekt dané vlastnosti. Obsahuje slova: *existuje*, *najde se*, *některý*, *někdo*, *alespoň jeden*, *lze nalézt*, apod. Symbolicky se zapisuje jako $\exists x[T(x)]$: Existuje *x*, pro něž platí tvrzení *T*.

**Tautologie** (valid or analytic sentence) je výrok, který je pravdivý, ať jsou dílčí výroky jakékoliv. *Pachatelem je Petr nebo Pavel nebo někdo jiný.*

**Operace s výroky**

**Negace** výroku V má označení $\neg V$.

**Konjunkce** výroků A a B má označení $A \wedge B$. Jazykově se vyjadřuje obvykle spojkami *a*, *i*, *ale*, *ani*.

**Disjunkce** neboli **alternativa** výroků A a B má označení $A \vee B$. Jazykově se vyjadřuje obvykle spojkou *nebo*.

**Implikace** výroků A a B má označení $A \Rightarrow B$. Má význam: Jestliže platí výrok A, pak platí i výrok B (když A tak B, z platnosti A vyplývá platnost B). Výrok A se nazývá antecedens, antecedent nebo premisa. Výrok B se nazývá konsekvens, konsekvent, konkluze nebo závěr.

**Ekvivalence** výroků A a B má označení $A \Leftrightarrow B$. Má význam: A je ekvivalentní s B (Výrok A platí tehdy a jen tehdy, když platí výrok B).

**Exkluzivní disjunkce** výroků A a B má označení $A \oplus B$ nebo A XOR B. Má význam: A není ekvivalentní s B (Když platí výrok A, neplatí výrok B, a když platí výrok B, neplatí výrok A.).

Priorita operátorů je od nejvyšší do nejnižší $\neg$, $\wedge$, $\vee$, $\Rightarrow$ a $\Leftrightarrow$.

Proto například věta $\neg P \vee Q \wedge R \Rightarrow S$ je ekvivalentní větě $((\neg P) \vee (Q \wedge R)) \Rightarrow S$.

Pravdivostní tabulka operátorů

| **A** | **B** | $\neg$ **A** | **A** $\wedge$ **B** | **A** $\vee$ **B** | **A** $\Rightarrow$ **B** | **A** $\Leftrightarrow$ **B** | **A** $\oplus$ **B** |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |

**Úsudek** neboli **inference** či **dedukce** je akt myšlení, který má tuto strukturu: známe pravdivostní hodnoty jednoho či více výroků (předpokladů) a přiřadíme pravdivostní hodnotu dalšímu výroku, neboli provedeme závěr. Například předpoklady jsou: *Když klesá tlak, zkazí se počasí. Dnes klesá tlak.* Závěr je: *Dnes se zkazí počasí.* Logicky správný úsudek (sound inference) je takový, kdy ze splnění všech předpokladů plyne splnění závěru, což se pozná tak, že na každém řádku tabulky pravdivostních hodnot, na němž mají všechny předpoklady hodnotu 1, má i závěr hodnotu 1.
A = *Klesá tlak*. – 2. předpoklad
B = *Zkazí se počasí.* – závěr
$A \Rightarrow B$ = *Když klesá tlak, zkazí se počasí.* – 1. předpoklad

| **A** | **B** | **$A \Rightarrow B$** |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Na tomto řádku mají oba dva předpoklady hodnotu 1 a i závěr má hodnotu 1, tudíž úsudek je logicky správný.

A = *Pavel chodí na angličtinu.* – 2. předpoklad
B = *Pavel chodí na němčinu.*
$(A \wedge \neg B) \vee (\neg A \wedge B)$ = *Každý z chlapců chodí jen na jeden ze dvou jazyků – angličtinu nebo němčinu.* – 1. předpoklad
$\neg B$ = *Pavel nechodí na němčinu.* – závěr

| **A** | **B** | **$(A \wedge \neg B) \vee (\neg A \wedge B)$** | **$\neg B$** |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |

Na tomto řádku mají oba dva předpoklady hodnotu 1 a i závěr má hodnotu 1, tudíž úsudek je logicky správný.

A = *Je jasné počasí.*
B = *Je chladná noc.*
$A \Rightarrow B$ = *Když se vyjasní, bude chladná noc.* – 1. předpoklad
$\neg A$ = *Dnes se nevyjasnilo.* – 2. předpoklad
$\neg B$ = *Dnes nebude chladná noc.* – závěr

| **A** | **B** | **$A \Rightarrow B$** | **$\neg A$** | **$\neg B$** |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 |

Na těchto řádcích mají oba dva předpoklady hodnotu 1, ale závěr nemá vždy hodnotu 1, tudíž úsudek není logicky správný.

Je možné sestavit věty obsahující implikaci a konjunkci, které mají pro všechny možné kombinace hodnot proměnných hodnotu 1. Takové věty patří mezi tautologie a mohou být využity jako inferenční pravidla, jejichž využití je ukázáno níže. Příklady takto dokázaných inferenčních pravidel jsou v následujících tabulkách.

| **A** | **B** | **$A \vee B$** | **$(A \vee B) \wedge \neg B$** | **Unit resolution $(A \vee B) \wedge \neg B \Rightarrow A$** |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |

| **A** | **B** | **C** | **$A \vee B$** | **$\neg B \vee C$** | **$(A \vee B) \wedge (\neg B \vee C)$** | **$A \vee C$** | **Resolution $(A \vee B) \wedge (\neg B \vee C) \Rightarrow A \vee C$** |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| A | B | C | $\neg A \Rightarrow B$ | $B \Rightarrow C$ | $(\neg A \Rightarrow B) \wedge (B \Rightarrow C)$ | $\neg A \Rightarrow C$ | Another view of resolution – implication is transitive $(\neg A \Rightarrow B) \wedge (B \Rightarrow C) \Rightarrow (\neg A \Rightarrow C)$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Predikátová logika prvního řádu (First-Order Predicate Calculus)**

Logika prvního řádu se tak jmenuje proto, že kvantifikuje objekty (entity prvního řádu pojmenovatelné podstatnými jmény) ale ne vztahy mezi těmito objekty nebo funkce těchto objektů. Logika vyššího řádu (higher-order logic) dovoluje kvantifikovat také vztahy a funkce. Pomocí logiky vyššího řádu můžeme například říct, že dva objekty jsou stejné, když a právě když všechny jejich vlastnosti jsou stejné:

$$\forall x, y \left[(x = y) \Leftrightarrow (\forall p \; p(x) \Leftrightarrow p(y))\right]$$

Nebo bychom mohli říct, že dvě funkce jsou stejné, když a právě když vrací ty samé hodnoty pro všechny argumenty:

$$\forall f, g \left[(f = g) \Leftrightarrow (\forall x \; f(x) \Leftrightarrow g(x))\right]$$

Logika vyššího řádu má větší vyjadřovací schopnost, ale dosud nejsou vyvinuty efektivní metody pro její používání. Proto se běžně využívá logika prvního řádu, která je teoreticky dobře zpracována a má dostatečné vyjadřovací schopnosti. [2 str. 195]

**Základní pojmy**

**Termy** označují popisované objekty, proměnné mající hodnotu objektů, funkce a jejich argumenty a návratové hodnoty. Termy jsou jedinými možnými argumenty predikátů.

**Predikát** je funkce, která má pravdivostní hodnotu. Může obsahovat proměnné, konstanty a funkce.

*Vše, co má peří, je pták.*

$$\forall x[\text{MaPeri}(x) \Rightarrow \text{Ptak}(x)]$$

**Konstanta** označuje konkrétní objekt. V predikátu „Ptak(Albatros)“ je „Albatros“ konstantou.

**Funkce** označuje objekt, který je v určitém vztahu k nějakému jinému objektu. Například, návratovou hodnotou funkce Podstavec($x$) je objekt, na kterém leží objekt $x$.

**Atomické formule** jsou jednotlivé predikáty spolu s argumenty.

**Literály** jsou atomické formule a negované atomické formule.

**Dobře utvořené formule** (**Well-Formed Formulas** – **WFFs**) jsou definovány rekurzivně: Literály jsou WFFs, WFFs spojené pomocí $\neg$, $\wedge$, $\vee$ a $\Rightarrow$ jsou WFFs, a WFFs obklopené kvantifikátory jsou také WFFs.

**Věty** jsou WFFs, ve kterých všechny proměnné, pokud tam nějaké jsou, jsou v působnosti kvantifikátorů. Takže větou je $\forall x[\text{MaPeri}(x) \Rightarrow \text{Ptak}(x)]$ i $\text{MaPeri}(\text{Albatros}) \Rightarrow \text{Ptak}(\text{Albatros})$.

**Vázané proměnné** (**bound variables**) jsou proměnné v působnosti kvantifikátorů.

**Volné proměnné** (**free variables**) nejsou v působnosti kvantifikátorů.

WFF, která neobsahuje žádnou vázanou proměnnou, je otevřená, v opačném případě je uzavřená.

Následující výraz není věta, protože obsahuje volnou proměnnou $y$.

$$\forall x[\text{MaPeri}(x) \vee \neg\text{MaPeri}(y)]$$

**Proměnné** mohou reprezentovat pouze objekty nikoliv predikáty. Z tohoto důvodu se tento druh logiky nazývá predikátová logika prvního řádu. Pokročilejší predikátová logika druhého řádu může operovat s proměnnými reprezentujícími predikáty. Méně pokročilá výroková logika nemůže operovat s žádnými proměnnými.

**Klauzule** (**clause**) je WFF z disjunkce literálů.

Objekty v logice korespondují s objekty v reálném světě. Vztahy mezi objekty v reálném světě korespondují s predikáty v logice.

**Důkaz (Proof)**

Platí například následující výrazy:

$$\text{MaPeri}(\text{Albatros})$$

$$\forall x[\text{MaPeri}(x) \Rightarrow \text{Ptak}(x)]$$

Tyto výrazy jsou **axiomy**.

Máte dokázat, že když platí tyto axiomy, tak také platí výraz $\text{Ptak}(\text{Albatros})$.

Když se vám to povede, dokázali jste, že $\text{Ptak}(\text{Albatros})$ je **teorém** vzhledem k dvěma předešlým axiomům.

Teorém logicky vyplývá z axiomů. Teorém se většinou dokazuje tak, že z axiomů se odvozují další výrazy, dokud se neobjeví požadovaný teorém. Odvozování se provádí podle pravidel **inference**. Nejjednodušší pravidlo inference se nazývá **modus ponens**, které říká, že když existuje axiom ve formě $E_1 \Rightarrow E_2$ a platí axiom $E_1$, potom z toho logicky

vyplývá $E_2$. Kdyby $E_2$ byl dokazovaný teorém, byl by po prvním použití modus ponens dokázán. V opačném případě se $E_2$ přidá k axiomům a modus ponens se dál uplatňuje na rostoucí seznam axiomů.

V našem příkladě s ptáky musíme nejdříve specializovat druhý výraz: Když výraz $\text{MaPeri}(x) \Rightarrow \text{Ptak}(x)$ platí pro všechna $x$, tak také musí platit pro speciální případ, kdy $x$ = Albatros. Následně musí platit $\text{MaPeri}(\text{Albatros}) \Rightarrow \text{Ptak}(\text{Albatros})$. Uplatnění pravidla modus ponens na výrazy $\text{MaPeri}(\text{Albatros}) \Rightarrow \text{Ptak}(\text{Albatros})$ a $\text{MaPeri}(\text{Albatros})$ má za výsledek dokazovaný teorém $\text{Ptak}(\text{Albatros})$, čímž je teorém dokázán.

Jedno z nejdůležitějších pravidel inference je pravidlo **rezoluce**, které říká, že když existuje axiom ve formě $E_1 \vee E_2$ a platí také axiom $\neg E_2 \vee E_3$, potom z nich logicky vyplývá $E_1 \vee E_3$. $E_1 \vee E_3$ se nazývá **rezolventa** axiomů $E_1 \vee E_2$ a $\neg E_2 \vee E_3$. Pravidlo rezoluce se dá zobecnit tak, že oba axiomy mohou být klauzulemi o libovolném množství literálů včetně jednoho. Jedinou podmínkou je, že jeden axiom musí obsahovat literál, který je negací nějakého literálu ve druhém axiomu.

Pravdivostní tabulka dokazující správnost pravidla rezoluce

| **A** | **B** | **C** | **A ∨ B** | **¬ B ∨ C** | **A ∨ C** |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | **1** | **1** | **1** |
| 1 | 0 | 0 | **1** | **1** | **1** |
| 1 | 0 | 1 | **1** | **1** | **1** |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | **1** | **1** | **1** |

S pravidlem rezoluce lze dosáhnout stejného závěru o Albatrosovi jako pomocí pravidla modus ponens. Nejdříve specializujeme kvantifikovaný výraz na Albatrosa. Potom nahradíme výraz $\text{MaPeri}(\text{Albatros}) \Rightarrow \text{Ptak}(\text{Albatros})$ za jeho ekvivalentní formu $\neg\text{MaPeri}(\text{Albatros}) \vee \text{Ptak}(\text{Albatros})$ a druhým axiomem bude $\text{MaPeri}(\text{Albatros})$. Rezoluce z těchto dvou axiomů přímo vyvodí $\text{Ptak}(\text{Albatros})$.

Modus ponens je speciálním případem rezoluce, protože vše odvoditelné pomocí modus ponens se dá odvodit i pomocí pravidla rezoluce. Z rezoluce je odvoditelné i další pravidlo inference zvané **modus tollens**, které říká, že když existuje axiom ve formě $E_1 \Rightarrow E_2$ a platí axiom $\neg E_2$, potom z nich logicky vyplývá $\neg E_1$.

Nejsamozřejmější strategií dokazování teorému je prohledávání stavového prostoru axiomů s nadějí, že náhodou se objeví i teorém. Častěji se však používá **metoda rezolučního zamítnutí** (**proof by refutation**). Při ní se dokazovaný teorém neguje a potom se přidá k množině axiomů. Teorém je dokázán, když při prohledávání stavového prostoru axiomů nalezneme rozpor.

Metodu rezolučního zamítnutí ukážeme opět na příkladu Albatrosa. K počátečním axiomům $\neg\text{MaPeri}(\text{Albatros}) \vee \text{Ptak}(\text{Albatros})$ a $\text{MaPeri}(\text{Albatros})$ přidáme negaci výroku, který má být dokázán, $\neg\text{Ptak}(\text{Albatros})$. Stejně jako v předchozím příkladu nejdříve rezolvujeme první a druhý axiom a výsledek rezoluce $\text{Ptak}(\text{Albatros})$ rezolvujeme s třetím axiomem $\neg\text{Ptak}(\text{Albatros})$. Výsledkem této druhé rezoluce je prázdná klauzule, označovaná jako **nil.** Vyjde-li během prohledávání prázdná klauzule, znamená to, že v množině axiomů existuje rozpor, který tam byl vnesen přidáním negovaného teorému. Je-li negovaný teorém v rozporu s axiomy, znamená to, že je dokázán.

Prohledávání stavového prostoru axiomů spočívá v určování, které dva axiomy se v dané fázi výpočtu vyberou k rezolvování. Existují pro něj různé strategie, jejichž nejdůležitějším atributem je, zda jsou **úplné** (**complete**) (vždy najdou řešení) nebo **neúplné** (**incomplete**) (ty jsou rychlejší). Mezi nejznámější patří následující:

**Strategie prohledávání do šířky** (**breadth-first strategy**) nejdříve rezolvuje všechny možné páry počátečních klauzulí, potom všechny možné páry výsledné množiny spolu s původní množinou a tak dál úroveň po úrovni.

**Strategie podpůrné množiny** (**set-of support strategy**) povoluje rezoluce jen z párů, kde jeden z rodičů je odvozen od negovaného teorému, protože mezi vstupními axiomy spor naleznout nelze.

**Strategie jednotkové preference** (**unit-preference strategy**) ve výběru klauzulí pro rezoluci preferuje ty s nejmenším počtem literálů.

**Vstupní strategie** generuje jen rezolventy z párů, ve kterých alespoň jeden z prvků je klauzule přímo z výchozí množiny klauzulí, jejíž spornost dokazujeme. Tato strategie není úplná.

**Filtrační strategie** je takovým rozšířením vstupní strategie, které nabízí další možnosti pro odvození nové klauzule. Je zde povoleno rezolvovat také klauzule, které jsou v příbuzenském poměru.

**Lineární strategie** využívají vždy poslední generovanou klauzuli jako jednu ze složek páru, ze kterého bude generována rezolventa. Využívá ji jazyk Prolog. Někdy může vést k zacyklení. Není to úplná strategie.

Každá strategie prohledávání generuje i rezolventy, které z různých důvodů nemohou přispět k důkazu sporu. Takové rezolventy je vhodné z dalšího zpracování vynechat. Vynecháváme klauzule, které jsou za všech okolností pravdivé.

příkladem takových klauzulí je tautologie, například $P(a) \vee Q(Y) \vee \neg P(a)$, nebo predikát Vetsi(*a*, *b*), říkající že *a* je větší než *b*, když *a* = 6 a *b* = 4. Z klauzulí je možné vynechat literály, které jsou určitě nepravdivé, například Vetsi(4, 6).

Dokazování teorému pomocí rezolučního pravidla vyžaduje, aby axiomy byly ve formě klauzulí. Převod na tuto formu se děje podle následujícího postupu.

1. Eliminace implikací podle identity $E_1 \Rightarrow E_2 = \neg E_1 \vee E_2$
2. Přesun negací k atomickým formulím

   $\neg(E_1 \wedge E_2) = \neg E_1 \vee \neg E_2$ Když neplatí oba současně, znamená to, že neplatí buďto jeden nebo druhý.

   $\neg(E_1 \vee E_2) = \neg E_1 \wedge \neg E_2$ Neplatí ani jeden výrok.

   $\neg(\neg E_1) = E_1$ Negace se vzájemně ruší.

   $\neg\forall x[E_1(x)] = \exists x[\neg E_1(x)]$ Když pro všechna *x* výrok neplatí, tak existuje *x*, pro které výrok neplatí.

   $\neg\exists x[E_1(x)] = \forall x[\neg E_1(x)]$ Když neexistuje *x*, pro které výrok platí, tak výrok neplatí pro všechna *x*.
3. Eliminace existenčních kvantifikátorů neboli skolemnizace

   $\exists x[E_1(x)] = E_1(x)$ Zde můžeme existenční kvantifikátor beze ztráty informace vynechat.

   $\exists y[\text{JeNa}(x, y)] = \text{JeNa}(x, \text{Podstavec}(x))$ Predikát JeNa(*x*, *y*) vyjadřuje, že objekt *x* leží na objektu *y*. Pomocí existenčního kvantifikátoru chceme říci, že existuje něco, na čem *x* leží. Pro *x* tedy existuje *y*, na kterém *x* leží. K získání objektu *y* nám tedy plně postačí *x* a proto *y* může být funkcí *x*. Funkce Podstavec(*x*) je příkladem Skolemovy funkce, která v tomto příkladě vrací objekt, na kterém leží *x*.
4. Přejmenování proměnných tak, aby v nich neexistovaly duplicity

   $\forall x[E_1(x) \wedge E_2(x) \wedge \forall y[E_3(y)] \wedge \forall y[E_4(y)]] = \forall x[E_1(x) \wedge E_2(x) \wedge \forall y[E_3(y)] \wedge \forall z[E_4(z)]]$
5. Přesun univerzálních kvantifikátorů doleva

   $\forall x[E_1(x) \wedge E_2(x) \wedge \forall y[E_3(y)] \wedge \forall z[E_4(z)]] = \forall x \forall y \forall z[E_1(x) \wedge E_2(x) \wedge E_3(y) \wedge E_4(z)]$
6. Přesun disjunkcí k literálům podle distributivního zákona $E_1 \vee (E_2 \wedge E_3) = (E_1 \vee E_2) \wedge (E_1 \vee E_3)$
7. Eliminace konjunkcí neboli roztrhání konjunkcí na jednotlivé axiomy.

   $\forall x \forall y \forall z[E_1(x) \wedge E_2(x) \wedge E_3(y) \wedge E_4(z)] =$

   $\forall x[E_1(x)]$

   $\forall x[E_2(x)]$

   $\forall y[E_3(y)]$

   $\forall z[E_4(z)]$

   Každá část konjunkce musí být pravdivá, aby celá konjunkce platila. Ve výsledných axiomech již mohou být jen disjunkce.
8. Přejmenování proměnných tak, aby dva různé axiomy neměly stejný název proměnné

   $\forall x \forall y \forall z[E_1(x) \wedge E_2(x) \wedge E_3(y) \wedge E_4(z)] =$

   $\forall v[E_1(v)]$

   $\forall x[E_2(x)]$

   $\forall y[E_3(y)]$

   $\forall z[E_4(z)]$
9. Eliminace univerzálních kvantifikátorů neboli přijetí konvence, že všechny proměnné platí obecně.

Při procesu inference se za proměnné dosazují jiné proměnné, konstanty a funkce platné pro danou úlohu tak, aby se vybrané axiomy mohly rezolvovat. Substituce, která umožní rezolvovatelnost dvou klauzulí, se nazývá **unifikace.**

Při řešení složitějších problémů rezoluční metodou se může objevit **problém exponenciální exploze** (**exponential-explosion problem**) spočívající ve vygenerování příliš mnoha nových axiomů. V případě, že dokazovaný vztah není pravdivý, se může objevit **problém zastavení algoritmu** (**halting problem**) spočívající v tom, že algoritmus se zastaví, až když už nejdou generovat nové axiomy, a to může trvat příliš dlouho. Rezoluční metoda není jediným v praxi používaným algoritmem. Kromě ní se využívá **Hornova logika**, ve které musí mít zpracovávané znalosti formu Hornových klauzulí. Ne každá znalostní báze je převoditelná do Hornových klauzulí, ale ty, které jsou, lze potom zpracovat v polynomiálním čase.

Hornovy klauzule (věty) mají formu: $P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow Q$, kde $P_i$ a $Q$ jsou atomické formule. Pokud jsou převedeny do disjunkce literálů $\neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_n \vee Q$, tak v nich musí být **maximálně jedna kladná atomická formule.** Nazývají se také Hornova normální forma. Při práci s touto formou používáme pravidlo modus ponens.

Předpokládejme, že máme následující bázi znalostí: [2 str. 276]

1. $P \Rightarrow Q$, 2. $\neg P \Rightarrow R$, 3. $Q \Rightarrow S$, 4. $R \Rightarrow S$

Z této báze bychom chtěli vyvodit tvrzení *S*. Pokud bychom to chtěli dokázat pomocí modus ponens, zjistíme, že to nejde. Důvodem je to, že fakt $\neg P \Rightarrow R$ nemůže být převeden do Hornovy formy, protože převeden do formy disjunkce

literálů $P \vee R$ obsahuje více než jednu kladnou atomickou formuli a tudíž na něj nemůže být uplatněn modus ponens. To znamená, že dokazování pomocí modus ponens je neúplné. Pokud bychom stejnou bázi převedli do konjunktivní normální formy, což je konjunkce disjunkcí vhodná pro uplatnění pravidla rezoluce, tak se nám tvrzení *S* dokázat povede. [2 str. 279]

Následující úlohu lze vyřešit jak pomocí modus ponens tak pomocí rezolučního principu. Zadání je uvedeno v angličtině, protože potom je možné si pro něj vyhledat na Internetu řešení. [2 str. 267]
*The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.*
*What we wish to prove is that West is a criminal.*

*Jack owns a dog.*
*Every dog owner is an animal lover.*
*No animal lover kills an animal.*
*Either Jack or Curiosity killed the cat, who is named Tuna.*
*Did Curiosity kill the cat?*

*1. Lucy is a professor*
*2. All professors are people.*
*3. Fuchs is the dean.*
*4. Deans are professors.*
*5. All professors consider the dean a friend or don't know him.*
*6. Everyone is a friend of someone.*
*7. People only criticize people that are not their friends.*
*8. Lucy criticized Fuchs.*
*Question: Is Fuchs no friend of Lucy?*

*A knowledge base has the following statements:*
*If there is gas in the tank and the fuel line is okay, then there is gas in the engine;*
*If there is gas in the engine and a good spark, the engine runs;*
*If there is power to the plugs and the plugs are clean, a good spark is produced;*
*If the battery is charged and the cables are okay, then there is power to the plugs.*
*a. Convert the rules above to CNF using proposition symbols such as GasInTank, FuelLineOK, GasInEngine, etc.*
*b. Suppose that you are given the facts that there is gas in the tank, the battery is charged, the fuel line and cables are both okay, and the plugs are clean. Using resolution, prove that the engine runs.*

*I am my own grandfather*
*The following story is quoted from N. Wirth's "Algorithms + data structures = programs" (Wirth 1976).*
*I married a widow (let's call her W) who has a grown-up daughter (call her D). My father (F), who visited us quite often, fell in love with my step-daughter and married her. Hence my father became my son-in-law and my step-daughter became my mother. Some months later, my wife gave birth to a son (S1), who became the brother-in-law of my father, as well as my uncle. The wife of my father, that is, my step-daughter, also had a son (S2).*
*Using predicate calculus, create a set of expressions that represent the situation in the above story. Add expressions defining basic family relationships such as the definition of father-in-law and use modus ponens on this system to prove the conclusion that "I am my own grandfather".*

Výchozím předpokladem úspěšnosti uplatňování výše popsaných metod logického usuzování je **monotónnost** logiky. Monotónnost znamená, že přidáním nového axiomu mezi ty stávající se nezmenší množství nových tvrzení, které lze dokázat. Jinak řečeno, mezi axiomy, ze kterých se má dokázat teorém, nesmí být spor. Příkladem nemonotónního uvažování je budování znalostní báze, při kterém se později narazí na výjimky. Například dítě se nejdříve dozví, že všichni ptáci létají a teprve potom se dozví o tučňácích.

**Prolog** patří mezi takzvané **deklarativní programovací jazyky**. Deklarativnost spočívá v tom, že je nejdříve vytvořen univerzální program pro řešení problému (inferenční mechanismus). V případě Prologu je to program pro rezolvování. Programátor už potom pouze popíše problém způsobem kompatibilním s oním univerzálním programem, ale nestanovuje algoritmus jeho řešení. V případě Prologu deklaruje fakta a implikace. Inferenční mechanismus Prologu z nich vyvodí řešení.
**Lisp** je příkladem **funkcionálních jazyků**. Na rozdíl od Prologu má procedurální charakter. Řešení problému pomocí programovacího jazyka Lisp spočívá ve spojování elementárních funkcí do složitějších funkcí.
**Imperativní** neboli **procedurální programovací jazyky** představují tradiční přístup k programování. Mezi jejich představitele patří BASIC, C a Pascal. Programátor pomocí nich stanoví algoritmus řešení.
[1 (2) str. 257, 4 str. 231]
Překladač jazyka Prolog vhodný pro instalaci do lokálního počítače:

SWI Prolog (open source) http://www.swi-prolog.org/, LPA Win Prolog (trial) http://www.lpa.co.uk/dow_tri.htm.

**Prolog a Lisp online**

http://courses.cs.vt.edu/~cs1104/TowerOfBabel/LISP/Lisp.outline.html
http://www.csse.monash.edu.au/~lloyd/tildeLogic/Prolog.toy/
http://ioctl.org/logic/prolog-latest
http://swish.swi-prolog.org/
https://www.tutorialspoint.com/execute_prolog_online.php

**Úloha nereprezentovatelná pomocí Prologu**

1. *Zkoušky složí jen ten, kdo studuje nebo má štěstí.*(*Když někdo složí zkoušky, znamená to, že studuje nebo má štěstí.*)

$\forall x[sloziZkousky(x) \Rightarrow [studuje(x) \vee maStesti(x)]]$

Stejná věta ve formě disjunkce literálů

$\forall x[\neg sloziZkousky(x) \vee studuje(x) \vee maStesti(x)]$

má více než jednu kladnou atomickou formuli, proto nemůže být reprezentována ve formě Hornových klauzulí. Prolog dovoluje negaci jen na straně antecedentu, čímž jsme nuceni zavést nové predikáty (atomické formule) s opačným významem, který je nutno počítači nadefinovat pomocí pravidel dávajících je do vztahu s původními predikáty. To však nelze přesně reprezentovat, protože přesný výsledek je funkce XOR (například buďto *studuje* nebo *nestuduje* ale ne obojí) a ta není reprezentovatelná pomocí Hornových klauzulí.

$\forall x[[\neg studuje(x) \Rightarrow nestuduje(x)] \wedge [nestuduje(x) \Rightarrow \neg studuje(x)]]$.

Druhá implikace není v Prologu reprezentovatelná. První implikace je reprezentovatelná, ale není reprezentovatelný fakt, který by s ní bylo možné rezolvovat. Například *not studuje(jan)* nelze v Prologu zapsat, protože to není antecedent implikace ale samostatný fakt.

2. *Když někdo studuje nebo má štěstí, tak složí zkoušky.*

$\forall x[[studuje(x) \vee maStesti(x)] \Rightarrow sloziZkousky(x)]$ je ekvivalentní dvojici implikací

$\forall x[studuje(x) \Rightarrow sloziZkousky(x)]$ a $\forall x[maStesti(x) \Rightarrow sloziZkousky(x)]$.

Obě implikace jsou reprezentovatelné v Prologu, ale nelze odvodit například, jestli měl někdo štěstí, když nestudoval, ale přesto složil zkoušky. Jednak proto, že nemůžeme v Prologu reprezentovat, že někdo nestudoval, jinak, než že ve faktech nebude *studuje(jan)*, jednak proto že z výchozí věty to nevyplývá, protože dle definice implikace může být antecedent nepravdivý a konsekvent pravdivý, tedy zkoušku je možné složit i z jiných důvodů, než že někdo studuje nebo má štěstí.

**Rozhodovací stromy (Decision trees)**

Literatura: [2], Chapter 18, str. 531 – 544. Soubor AI.xls, list Decision Trees.

Rozhodovací stromy se užívají pro reprezentaci faktů, které spolu souvisí jako Hornovy klauzule. Atomické formule premis jsou uzly a závěry implikací jsou listy rozhodovacích stromů. Algoritmus minimalizující informační entropii, jehož základy jsou ukázány v souboru AI.xls, se nazývá ID3 a má pokročilejší varianty C4.5 a C5.0.

**Bayesovská síť (Bayes net, bayesian network, belief network)**

Literatura: [2], Chapter 15, str. 436 – 470. Soubor AI.xls, list Bayes.

Bayesovsé sítě se užívají pro reprezentaci faktů, mezi kterými existují kauzální souvislosti. Bayesovská síť není neuronová síť ale reprezentace podmíněnosti pravděpodobností. Stejně jako při práci s neuronovými sítěmi nejdříve vypočteme parametry (pravděpodobnosti) sítě ze vstupních dat a potom je používáme pro předpovídání neúplných údajů.

**Příklad**: Analýza souvislostí v informačním zajištění organizací pomocí bayesovské sítě

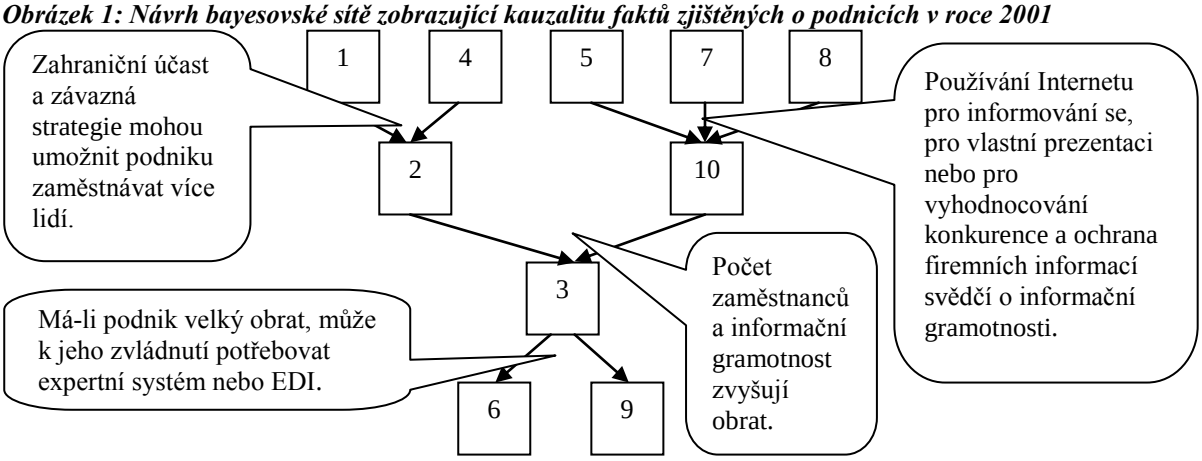
Jen stručně naznačíme, jak by se daly získávat informace o zákonitostech informačního zajištění v českých podnicích pomocí bayesovské sítě. Za předmět zkoumání jsme si zvolili následující vlastnosti podniků:

1 Zahraniční účast,
2 Počet zaměstnanců,
3 Obrat,
4 Závazná strategie,
5 Využívání WWW pro informování se nebo pro vlastní prezentaci,
6 Využívání EDI,
7 Využívání Internetu pro monitorování a vyhodnocování konkurence,
8 Vlastní způsob ochrany firemních informací,
9 Využívání expertních systémů,
10 Informační gramotnost zaměstnanců.

Tyto vlastnosti byly zjišťovány v dotazníkové akci za rok 2001 a byly vyplněny ve 28 dotaznících.

Bayesovské sítě přehledně reprezentují kauzální závislosti mezi jednotlivými skutečnostmi. Existují metody, které je z dat umí sestavit. Jiným často využívaným postupem je jejich sestavení podle vědomostí lidského experta a následné ověřování správnosti domněnek experta z dat. Obrázek 1 ukazuje ručně sestavenou bayesovskou síť a zdůvodnění jejích jednotlivých spojů. Tato zdůvodnění si nečiní nárok na to být považována za bezchybná. Je to jen příklad, jak by mohlo být postupováno. Ze stejných vlastností by mohly být sestaveny sítě s různými architekturami podle různých názorů na kauzalitu do nich zahrnutých vlastností podniků. Vlastnosti jsou označeny jejich čísly z výše uvedeného seznamu.

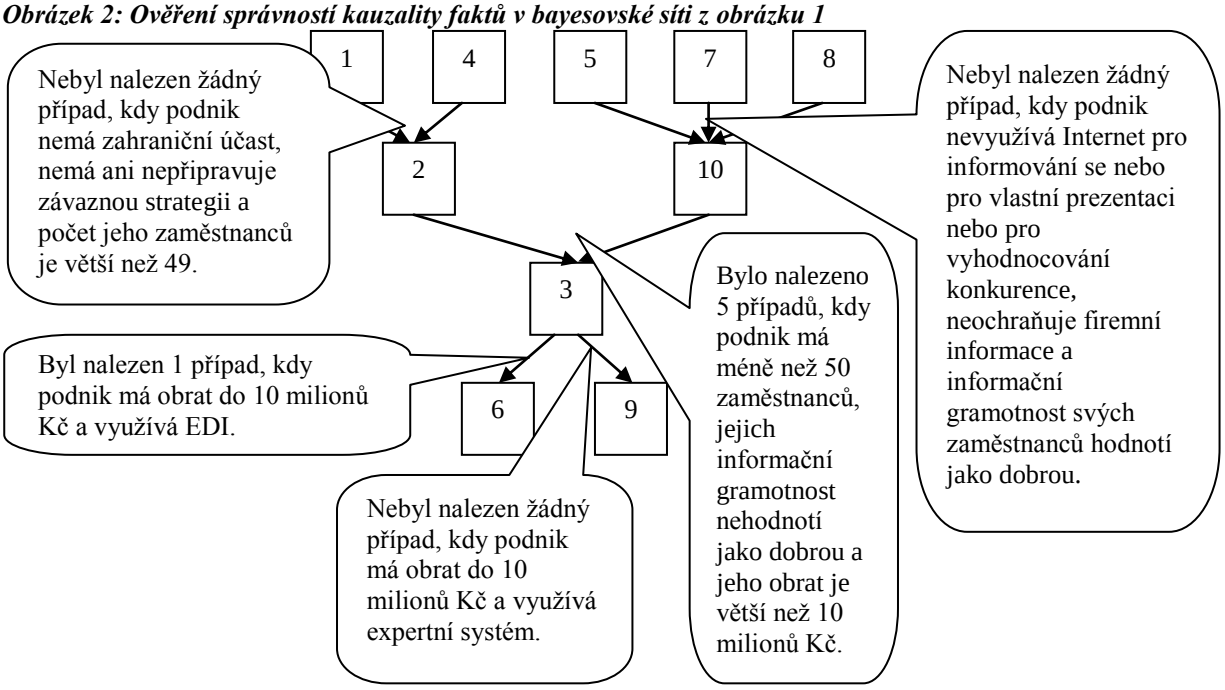
***Obrázek 1: Návrh bayesovské sítě zobrazující kauzalitu faktů zjištěných o podnicích v roce 2001***




*Pramen: Vlastní*

Po sestavení sítě ověříme z dat oprávněnost našich úvah o kauzalitě do ní zahrnutých faktů. Je-li v síti vyjádřeno, že pouze jev A způsobuje jev B a v datech je významný počet případů, kdy platí jev B a neplatí jev A, potom je tvrzení sítě o příčině jevu B zpochybněno. Výsledky jsou v obrázku 2.

***Obrázek 2: Ověření správností kauzality faktů v bayesovské síti z obrázku 1***




*Pramen: Vlastní*

Obrázek 2 ukazuje, že data nejvíce zpochybnila názor, že množství zaměstnanců a jejich informační gramotnost podmiňují schopnost podniku mít vysoký obrat. Zbytek grafu není s daty příliš ve sporu. Při řešení větší úlohy by se graf podle dat musel přestavovat tak dlouho, dokud by mezi ním a daty přetrvávaly zásadní rozpory.

V další fázi řešení se ke každému uzlu sítě sestaví z dat tabulky s podmíněnými pravděpodobnostmi jevu v uzlu za předpokladu, že platí uzel předchozí.

Síť se potom používá na doplnění informací o nových podnicích, o kterých víme jen některé z údajů reprezentovaných v síti, pomocí zákonů pravděpodobnosti.

**Sestavování bayesovské sítě na základě zkušeností s kauzalitou**

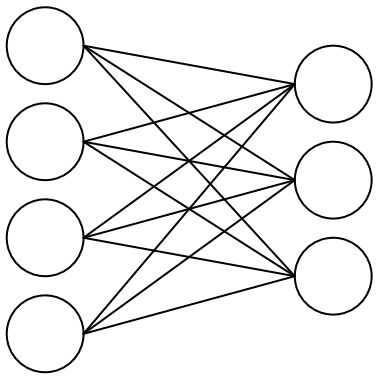
Sestavte bayesovskou síť z následující množiny faktů:

1. Student chodí na přednášky.
2. Student se učí na zkoušky.
3. Student složí zkoušky.
4. Student si najde životního partnera.
5. Studentovi je dobře.

Při sestavování sítě se řiďte vlastním názorem (neexistuje jediné správné řešení). Do hotové sítě doplňte váš vlastní odhad pravděpodobností a vypočtěte pravděpodobnost, že platí fakt 5 za předpokladu, že neplatí fakt 3 a 4.

**Pravděpodobností počet**

Pravděpodobnost současného výskytu jevu A a B = P(A,B).
Podmíněná pravděpodobnost jevu A za předpokladu, že platí jev B = P(A|B) = P(A,B) / P(B).
Bayesovo pravidlo:
P(A|B) = P(A,B) / P(B)
P(B|A) = P(A,B) / P(A)
P(A|B) · P(B) = P(B|A) · P(A)
P(A|B) = P(B|A) · P(A) / P(B)
Jde-li nám pouze o vybrání takového jevu A, pro které je maximální P(A|B), a známe P(B|A), stačí najít A maximalizující vzorec P(A|B) = P(B|A) · P(A), protože P(B) je pro všechna A konstantní.
Když jsou jevy A a B nezávislé, tak P(A|B) = P(A).
Řetězení podmíněných pravděpodobností:
P(A,B,C,D,E,F) = P(F|A,B,C,D,E) · P(E|A,B,C,D) · P(D|A,B,C) · P(C|A,B) · P(B|A) · P(A)
protože
P(F|A,B,C,D,E) = P(A,B,C,D,E,F) / P(A,B,C,D,E)
P(A,B,C,D,E,F) = P(F|A,B,C,D,E) · P(A,B,C,D,E)
P(E|A,B,C,D) = P(A,B,C,D,E) / P(A,B,C,D)
P(A,B,C,D,E) = P(E|A,B,C,D) · P(A,B,C,D)
atd.



***Obrázek 3: Dvouvrstvá neuronová síť.***

Do řady jejích neuronů na levé straně může vstoupit jeden vzor a současně do řady jejích neuronů na pravé straně vstoupí prvky vektoru vzoru, který je s tím, co vstoupil do levé strany, ve dvojici. Váhy na spojích mezi neurony se aktualizují podle hodnot na neuronech, které spojují, a podle pravidla učení, které je síti vlastní.

**Neuronové sítě**

Hlavním účelem neuronových sítí je klasifikace. Hlavní výhodou neuronových sítí oproti jiným způsobům klasifikace je jejich odolnost proti šumu v informacích.

Biologický mozek

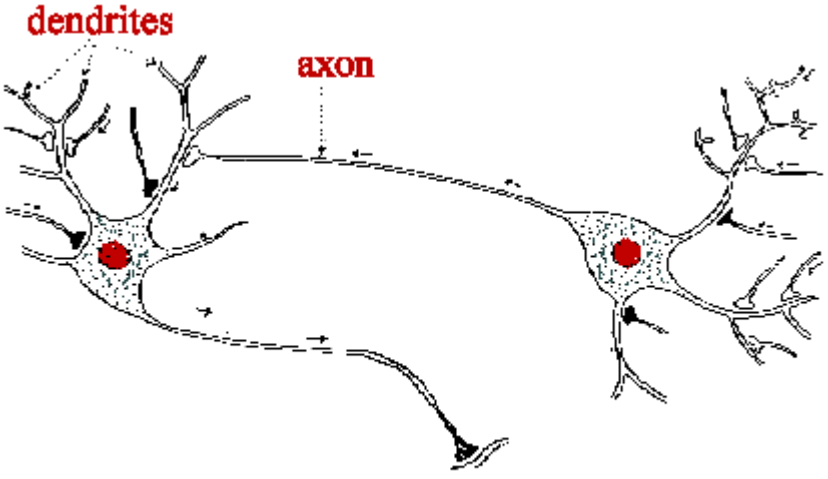



fig 1. Diagram of two neurons

Umělá neuronová síť

Odhaduje se, že v lidském mozku je 25 miliard neuronů. Počet synaptických štěrbin, kterými se neurony stýkají, spočítat nelze.

**Historie**

1943 Warren McCulloch and Walter Pitts: Neuronové sítě mohou reprezentovat logické funkce. Vznikl směr kognitivních věd zvaný konekcionismus jako protiklad tzv. symbolické umělé inteligence.
1949 Donald Olding Hebb vymyslel první pravidlo učení pro neuronové sítě.
1956 John McCarthy zorganizoval konferenci, na které vznikl pojem "umělá inteligence".
1957 Frank Rosenblatt: Neuronová síť zvaná Perceptron, učící se podle lepšího pravidla, než je Hebbovo.
1960 Bernard Widrow a jeho student Marcian Ted Hoff vymysleli ještě lepší pravidlo učení zvané delta rule.
1969 Marvin Minsky a Seymour Papert vydali knihu "Perceptrons" zdůrazňující neschopnost dvouvrstvých sítí řešit lineárně neseparabilní problémy.
1974 Paul Werbos vymyslel pravidlo učení vícevrstvých neuronových sítí "backpropagation" schopné řešit lineárně neseparabilní problémy. V polovině 80. let bylo toto pravidlo nezávisle objeveno znova více lidmi a konečně si získalo širší publicitu. Tím se oživil zájem o neuronové sítě, který byl dříve utlumen knihou "Perceptrons".
1982 John Hopfield a David Tank: Hopfieldovy sítě.
1982 Teuvo Kohonen: samoorganizující se sítě řešící rozpoznání řeči nebo problém obchodního cestujícího.
1986 Terrence Sejnowski a Charles Rosenberg: třívrstvá síť NETtalk pro fonetickou transkripci.

**Reprezentace informace**

- lokální: Pro každou kategorii je vyhrazen zvláštní neuron.
- distribuovaná: Vektory pro různé kategorie nejsou ortogonální. Reprezentace má větší kapacitu ale menší přesnost.

Vektor reprezentující instanci (výskyt, konkrétní hodnotu) popisovaného objektu se nazývá **vzor** a značí se většinou jako **x.**

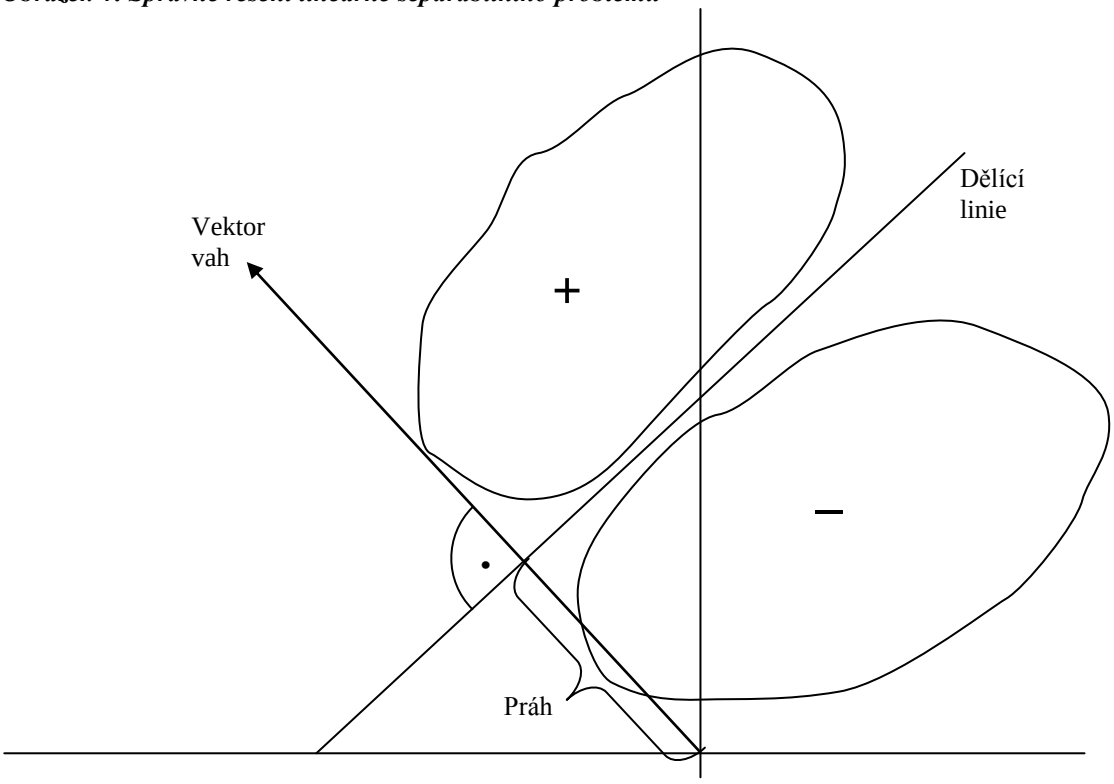
**Model neuronu**

**Způsoby učení neuronových sítí**

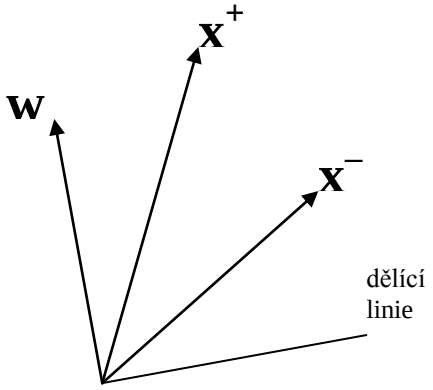
- s dohledem
- bez dohledu (samoorganizace)

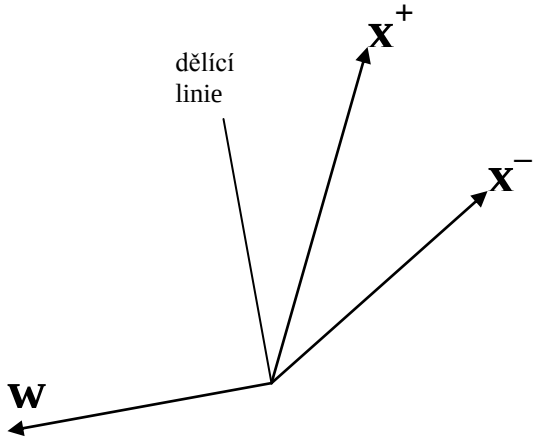
**Perceptron**

***Obrázek 4: Správné řešení lineárně separabilního problému***



***Obrázek 5: Učení v Perceptronu. Na rozdíl od obrázku 4 je v rozměrech vektoru vah w obsažena i prahová hodnota.***




Vektor $\mathbf{x}^+$ je klasifikován správně. Vektor $\mathbf{x}^-$ je nesprávně přiřazen do kategorie +, protože jeho skalární součin s vektorem vah **w** je kladný. Lze to napravit tak, že nový vektor vah **w** bude roven $\mathbf{w} - \mathbf{x}^-$.

Vektor $\mathbf{x}^-$ je klasifikován správně. Vektor $\mathbf{x}^+$ je nesprávně přiřazen do kategorie –, protože jeho skalární součin s vektorem vah **w** je záporný. Lze to napravit tak, že nový vektor vah **w** bude roven $\mathbf{w} + \mathbf{x}^+$.

**Delta rule**

Gradient
$3x + 2y + z = 6$
$x/2 + y/3 + z/6 = 1$
$z = 6 - 3x - 2y$
$\partial z/\partial x = -3$
$\partial z/\partial y = -2$
Gradient, neboli vektor (–3, –2) určuje směr nejvyššího stoupání po ose *z*, jdeme-li pro rovině $3x + 2y + z = 6$.

Příklad:

Byla naprogramována neuronová síť s 5 vstupními neurony pro 5 oblastí, ve kterých v dotaznících sledujeme přínosy po změně IS, a s 1 výstupním neuronem, do kterého vstupovala kvantitativní hodnota vyjadřující sklon podniku vyhodnocovat přínosy IS/IT. Tato neuronová síť se učila odhadovat závislost vstupních a výstupních dat pomocí pravidla zvaného delta rule. Neuronová síť je znázorněna na obrázku č. 4. Vstupy do sítě za určitý podnik tvoří vektor **x**. Na spojích mezi vstupními neurony a výstupním neuronem jsou takzvané váhy tvořící vektor **w**.

Skutečný výstup sítě je skalární součin vektorů $y = \mathbf{x} \cdot \mathbf{w}$.

Chyba výstupu sítě pro jeden podnik je oceněna hodnotou $e = (y - t)^2$.

Parciální derivace chybové funkce *e* podle vah *w* udává směr nejvyššího stoupání funkce *e* neboli gradient. Její opačná hodnota udává, jakým směrem se mají váhy měnit, aby chyba *e* klesala a tudíž síť co nejlépe odhadovala pro každý vstup správný výstup.

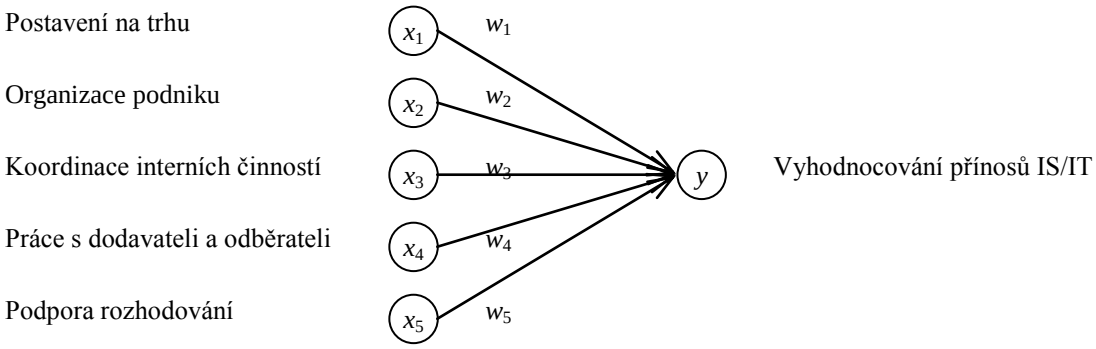
$$-\frac{\partial e}{\partial w} = -\frac{\partial\left[(\mathbf{x}\cdot\mathbf{w} - t)^2\right]}{\partial w} = -2(y - t)\cdot x = r\cdot(t - y)\cdot x\,,$$

kde *r* je parametr rychlosti učení a *t* je správný výstup sítě.

Při zpracování jednoho podniku tedy síť přičte ke každé své váze nějakou malou hodnotu přímo úměrnou součinu rozdílu správného a skutečného výstupu sítě pro tento podnik s hodnotou *x*, která vstupuje do neuronu spojeného s výstupním neuronem synapsí s upravovanou váhou.

Vztah $y = \mathbf{x} \cdot \mathbf{w}$ je dobrý pro teoretické zdůvodnění výše uvedeného vzorce pro úpravu váhy sítě, ale prakticky může způsobit přetečení váhy, když se *t* příliš liší od *y*, takže se *y* musí transformovat na hodnotu blízkou rozsahu *t*. Vhodná funkce pro tuto transformaci je signum, neboli když $\mathbf{x} \cdot \mathbf{w} > 0$, tak $y = 1$, jinak $y = 0$ pro *t* z rozsahu <0; 1>.

***Obrázek 6: Neuronová síť pro analýzu souvislostí přínosů po změně či inovaci podnikového IS s mírou vyhodnocování přínosů IS/IT***




*Pramen: vlastní*

Při učení síť pro každý podnik vypočetla skalární součin vektoru **x** oblastí, ve kterých sledujeme přínosy po změně IS, s vektorem vah **w** na spojích mezi vstupními neurony a jedním výstupním neuronem, a výsledek porovnala s cílovou hodnotou vyjadřující sklon podniku vyhodnocovat přínosy IS/IT. Podle rozdílu síť pozměnila své váhy tak, aby při příštím čtení dat o tom samém podniku, byl její odhad výstupní hodnoty přesnější. Celý soubor podniků byl takto zpracován několikrát, dokud se výkon sítě zlepšoval. Výsledkem tohoto výpočtu je hodnota vah sítě pro oblasti, ve kterých sledujeme přínosy po změně IS. Počáteční váhy měly náhodné hodnoty z intervalu (-0,05; 0,05). Protože počáteční podmínky ovlivňují výsledek, provedl se výpočet vah pro celý soubor podniků 40-krát a potom se váhy za každý pokus zprůměrovaly. Vztah mezi naplněním očekávání zlepšení situace v určité oblasti podniku a horlivostí, s jakou podnik vyhodnocuje přínosy IS/IT, se ve výsledných vahách neuronové sítě projeví tak, že budou-li v datech relativně převládat případy, kdy hodnota splnění očekávání pro určitou oblast byla vysoká a zároveň ocenění míry vyhodnocování přínosů IS/IT bylo vysoké, hodnota váhy pro tuto oblast bude také relativně vysoká.

**Lineárně neseparabilní problém**

Příkladem je funkce XOR:

| **A** | **B** | **A XOR B** |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Lineárně neseparabilní problém nelze řešit sítí, ve které mezi vstupní a výstupní vrstvou není skrytá vrstva. Pro síť, která má počítat jednoduché logické funkce, lze váhy a prahy ve vícevrstvé síti stanovit ručně. Pro reálné problémy je však třeba tuto úlohu algoritmizovat. Řešením je algoritmus **backpropagation**. Tento algoritmus vyžaduje, aby neurony v síti měly sigmoidální přenosovou funkci namísto funkce signum, která stačila pro Perceptron.

***Obrázek 7: Váhy a prahy sítě řešící problém XOR***