Masarykova univerzita

Fakulta informatiky

PB016 Umělá inteligence

Sbírka příkladů

Semestr Podzim 2022

Chyby hlašte na: 469088@mail.muni.cz

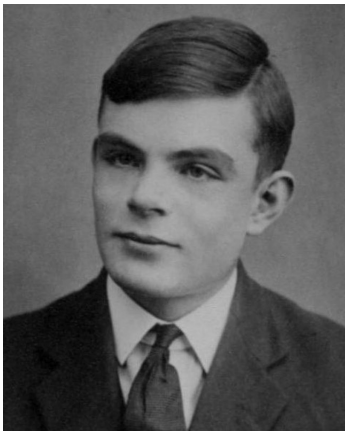
Obsah

1 Úvod do umělé inteligence

Úvodní kapitola nabízí čtenáři sbírky první setkání s umělou inteligencí. Představuje nejprve Turingův test k určení schopnosti stroje chovat se inteligentně a příklady k zamyšlení nad hranicemi a možnostmi oboru umělé inteligence jako takového (v této jediné části sbírky jsou příklady ponechány bez řešení). Další část pak prezentuje pojem *problému* a na řadě příkladů ilustruje, jak může volba vhodné datové struktury či strategie k řešení daného úkolu ovlivnit samotnou možnost úkol vyřešit.

1.1 Umělá inteligence a Turingův test

V roce 1950 navrhnul Alan Turing tzv. *Turingův test* (který původně sám nazval „imitační hrou“, viz původní Turingův článek), jež si klade za cíl ověřit schopnost stroje vykazovat inteligentní chování.



Alan Turing ve věku 16 let

Turing v článku nejprve uvádí, že navrhuje uvážit otázku, zda stroje umí myslet. Jelikož však není snadné definovat, co znamená „myslet“, navrhuje nahradit tuto poněkud vágní otázku jinou, a sice zda „Lze sestrojit počítač, který by byl schopen složit Turingův test?“

Definice 1: *Turingův test* je navržen jako hra tří hráčů, z nichž jeden je stroj podrobený zkoušce a ostatní dva jsou lidé. V základní variantě Turingova testu jsou do různých místností umístěni A) testovaný stroj, B) člověk, C) rozhodčí. Rozhodčí může komunikovat se zbylými dvěma formou textových zpráv a má za úkol zjistit, který z nich je stroj a který je člověk. Stroj uspěl v Turingově testu, nedokáže-li ho rozhodčí spolehlivě rozeznat od člověka.

Složení Turingova testu počítačem vyžaduje zvládnutí několika oblastí oboru. Pro porozumění zpráv a jejich generování je potřeba *zpracování přirozeného jazyka (NLP)*. Pro uchování údajů a vyvozování závěrů je nutné ovládnout *reprezentaci a vyvozování znalostí*. V neposlední řadě je potřeba metod *strojového učení*, které umožňují učit se a adaptovat se na změny vnějšího prostředí.

Na druhou stranu není pro složení testu nutné zvládnout obory jako *robotická manipulace* či *počítačové vidění*, jelikož veškerá komunikace probíhá výhradně textovými zprávami.

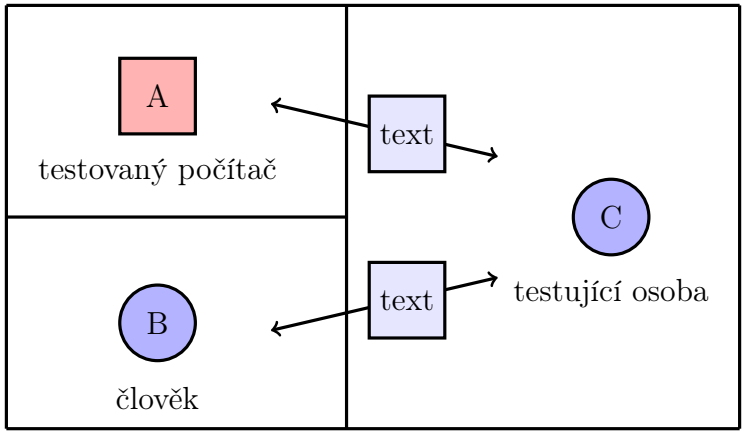


Schéma uspořádání Turingova testu

Příklad 1.1.1. Definujte vlastními slovy následující pojmy:

a) inteligence,

b) umělá inteligence,

c) agent,

d) racionálnost,

e) logické usuzování.

Nalezněte možné námitky proti uvedeným definicím a zpřesněte se je.

Příklad 1.1.2. Jsou reflexy racionální? Jsou inteligentní? Zdůvodněte.

Příklad 1.1.3. Do jaké míry jsou následující počítačové systémy příklady umělé inteligence?

a) Čtečka čárových kódů v supermarketu.

b) Internetové vyhledávače.

c) Hlasové zadávání příkazů telefonu.

d) Síťové směrovací algoritmy, které dynamicky reagují na stav sítě.

Příklad 1.1.4. Přečtěte si původní článek Alana Turinga Computing Machinery and Intelligence. Turing v sekci 6 ve článku rozebírá různé námitky vznesené vůči Turingovu testu. Které z námitek jsou stále platné? Napadnou vás nové námitky, které vzešly z pozdějšího vývoje v oboru?

Příklad 1.1.5. Upravíme-li Evansův program pro řešení problémů s geometrickou analogií, aby získal v standardním IQ testu hodnotu 200, jednalo by se o program inteligentnější než člověk? Vysvětlete.

Příklad 1.1.6. Alan Turing ve své práci představil seznam věcí, které by stroje nemusely nikdy umět: být milé, být nápadité, být krásné, být přátelské, být iniciativní, mít smysl

pro humor, rozeznat správné od špatného, dělat chyby, zamilovat se, vychutnat si jahody se šlehačkou, okouzlit někoho, poučit se ze zkušenosti, používat správná slova, přemýšlet samy o sobě, mít chování rozmanité jako člověk, provést něco skutečně nového. Kterých se již podařilo dosáhnout? Které jsou v principu dosažitelné počítačem? Které z nich jsou stále problematické, protože vyžadují vědomé mentální stavy?

Příklad 1.1.7. Argument čínského pokoje je myšlenkový experiment, který se snaží prokázat, že počítače nemohou mít „mysl“, „chápání“ či „vědomí“, a to nezávisle na tom, jak inteligentní chování vykazují. Nastudujte si o argumentu čínského pokoje více. Znamená vyvrácení argumentu čínského pokoje, že vhodně naprogramované počítače mohou mít mentální stavy? Plyne z přijetí argumentu, že počítače mentální stavy mít nemohou?

1.2 Řešení problémů

V této sekci se poprvé setkáme s některými problémy a úvodem do strategií, které lze uplatnit při jejich řešení. K některým z nich se budeme v dalších kapitolách vracet při probírání konkrétních metod řešení problémů (jako například heuristické prohledávání).

Problémem rozumíme zadání množiny konfigurací (např. přiřazení čísel volným políčkům sudoku), mezi nimiž hledáme konfigurace, které splňují konkrétní vlastnosti (např. vlastnost, že je sudoku vyplněno správně). Alternativou k hledání konfigurací splňujících nějakou vlastnost je hledání konfigurací, které jsou v jistém smyslu „nejlepší“. V takovém případě hovoříme o *optimalizačním problému*. Co přesně rozumíme problémem, může být navíc v konkrétním kontextu blíže upřesněno.

Definice 2: Při řešení *optimalizačního problému* je cílem nalézt mezi jeho konfiguracemi takovou, která je mezi všemi ostatními nejlepší podle předem daného kritéria.

Plánujeme-li si semestr tak, abychom získali co nejvíce kreditů a zároveň nám zůstalo co nejvíce volného času na své koníčky a kamarády, řešíme optimalizační problém. Naopak snažíme-li se naskládat nákup do ledničky tak, aby se dala zavřít, problém optimalizační neřešíme, protože jsme spokojeni s jakýmkoliv řešením, které nám umožní dovřít dveře ledničky.

Při řešení problémů hraje důležitou roli počet prohledávaných konfigurací. Přirozenou snahou je tento počet co nejvíce zredukovat, čehož lze dosáhnout využitím znalostí o problému a vhodnou volbou datové struktury.

Příklad. Uvažte *problém n dam*. V problému n dam je rozmisťováno n dam na šachovnici o rozměru $n \times n$. Řešením jsou taková rozmístění (všech dam), v nichž se žádná dvojice dam neohrožuje na řádku, ve sloupci ani diagonálně.

a) Spočtěte počet různých konfigurací problému, tj. počet různých rozmístění dam na šachovnici.

b) Navrhněte datovou strukturu reprezentující konfigurace problému.

c) Navrženou datovou strukturu upravte tak, aby svým návrhem omezovala počet různých konfigurací, ale ne počet potenciálních řešení. Vyjděte z povahy problému a promítněte požadavky na řešení problému do návrhu datové struktury. Spočítejte nový počet konfigurací.

a) Každá dáma může být umístěna na jednu z $n \times n = n^2$ pozic. Pro n dam tedy celkově získáme $\underbrace{n^2 \cdot \ldots \cdot n^2}_{n} = n^{2n}$ rozmístění. Pro 8 dam máme $8^{16} \approx 2,8 \cdot 10^{14}$ konfigurací.

b) Na reprezentaci souřadnic jedné dámy lze použít dvojici (x, y). Konfiguraci pak reprezentuje pole n takových dvojic $[(x_1, y_1), \ldots, (x_n, y_n)]$.

c) Jistě nelze umístit více dam na jedno políčko šachovnice. Můžeme tedy předpokládat unikátnost součadnic a místo pole použít množinu souřadnic $\{(x_1, y_1), \ldots, (x_n, y_n)\}$. Počet konfigurací v takovém případě je $n^2 \cdot (n^2 - 1) \cdot \ldots \cdot (n^2 - n + 1) = \frac{(n^2)!}{(n^2-n)!}$. Konkrétně pro 8 dam je to $64 \cdot 63 \cdot \ldots 57 \approx 1,8 \cdot 10^{14}$ konfigurací.

V návrhu lze jít ještě dále. Z povahy problému víme, že v každém sloupci bude právě jedna dáma – jinak by v nějakém sloupci byly alespoň dvě a ty by se vzájemně ohrožovaly. Dámám v seznamu lze tedy postupně přiřadit x-ovou souřadnici 1 až n: $[(1, y_1), \ldots, (n, y_n)]$ a celý seznam zredukovat na seznam pouze y-ových souřadnic: $[y_1, \ldots, y_n]$. Pokud nepřipustíme opakování hodnot v seznamu (čili více dam na řádku), omezíme počet možných konfigurací na $n \cdot (n-1) \cdot \ldots \cdot 1 = n!$. Pro 8 dam je to $8! = 40320$ konfigurací.

Příklad demonstruje zásadní vliv promítnutí znalostí o problému do přístupu k jeho řešení. Zatímco naivní řešení problému n dam vyžaduje pro $n = 8$ prohledávání prostoru přibližně $2,8 \cdot 10^{14}$ konfigurací, vhodnou úpravou strategie lze číslo redukovat na 40320. Už pro relativně malá čísla n tedy volba strategie rozhoduje o tom, zda je problém prakticky řešitelný či ne.

Příklad 1.2.1. Pro uvedné problémy spočítejte počet různých konfigurací, tj. potenciálních řešení. Pro každý problém navrhněte datovou strukturu pro reprezentaci konfigurací problému.

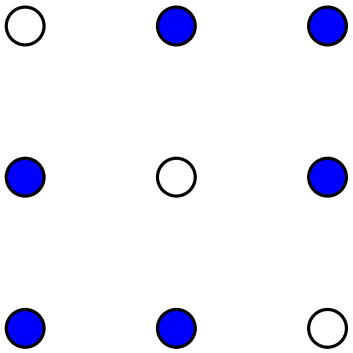
a) *Sudoku.* Uvažujte obecné sudoku $n \times n$ s k hodnotami zadanými.

b) *Problém přiřazení.* V problému přiřazení je dána množina n úkolů T, množina n pracovníků W a cenová funkce $c : T \times W \to \mathbb{R}$, která určuje náklady na vykonání úkolů jednotlivými pracovníky, tj. cena práce pracovníka w na úkolu t je dána jako $c(t, w)$. Cílem je rozdělit úkoly mezi pracovníky (jeden úkol na jednoho pracovníka) tak, aby celková cena práce byla co nejmenší.

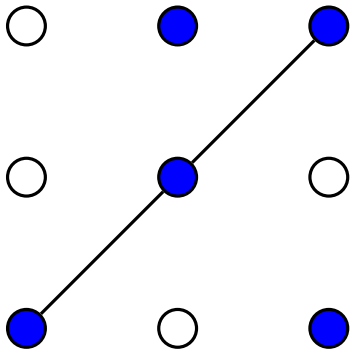
c) *Problém batohu.* Je zadána množina n položek $1, \ldots, n$ o hmotnostech $w_1, \ldots, w_n$ a hodnotách $v_1, \ldots v_n$ a maximální nosnost batohu W. Cílem je z položek vybrat takové,

aby součet jejich hmotností nepřekročil W a zároveň byl co nejvyšší.

Příklad 1.2.2. Uvažte problém *žádné tři v řadě*. V tomto problému je zadána mřížka rozměrů $n \times n$ (kde $n \in \mathbb{N}$) a cílem je zjistit, kolik do ní lze umístit bodů tak, aby žádné tři body neležely v jedné přímce.



Na mřížce 3×3 lze rozmístit maximálně 6 bodů (zvolené body jsou vyplněny modře).



Uvedené rozmístění nesplňuje požadavek, aby tři body neležely v jedné přímce.

a) Spočtěte počet různých konfigurací problému, tj. různých rozmístění bodů (i těch nesplňujících požadavek zadání).

b) Navrhněte datovou strukturu reprezentující konfigurace problému.

c) Navrženou datovou strukturu upravte tak, aby svým návrhem omezovala počet různých konfigurací, ale ne počet potenciálních řešení. Vyjděte z povahy problému a promítněte požadavky na řešení problému do návrhu datové struktury. Spočítejte nový počet konfigurací.

d)* Proveďte asymptotické srovnání počtu konfigurací v naivním a vylepšeném řešení.

e)* Ještě dále vylepšete návrh datové struktury, aby byl počet konfigurací asymptoticky menší než vzorové řešení části c).

Příklad 1.2.3. Které z následujících problémů jsou problémy optimalizační? Zdůvodněte.

a) Problém n dam.

b) Sudoku.

c) Problém přiřazení.

d) Žádné 3 v řadě.

Příklad 1.2.4. Implementujte řešení problému n dam využitím naivního přístupu a přístupu omezujícího počet prohledávaných konfigurací. Experimentálně zjistěte časy nalezení všech řešení pro různá malá n oběma způsoby. Porovnejte s dříve vypočítaným prohledávaným počtem konfigurací. Lze pozorovat úměru mezi časem výpočtu a počtem konfigurací?

Příklad 1.2.5. Implementujte řešení problému žádné 3 v řadě využitím naivního přístupu a přístupu omezujícího počet prohledávaných konfigurací. Experimentálně zjistěte

časy nalezení všech řešení pro různá malá n oběma způsoby. Porovnejte s dříve vypočítaným prohledávaným počtem konfigurací. Lze pozorovat úměru mezi časem výpočtu a počtem konfigurací?

2 Prohledávání stavového prostoru

V předchozí kapitole jsme si vyzkoušeli, jakými způsoby se dá řešit několik klasických problémů, které spadají do oblasti umělé inteligence. Obecně samozřejmě existuje velké množství různých druhů problémů, a tedy i strategií jak je řešit. Často se ale dají využít některé známé univerzální postupy.

V této kapitole si formálně představíme jednu z takových strategií, a to *prohledávání stavového prostoru*. Ukážeme si jak správně problémy formulovat, představíme si několik různých prohledávacích algoritmů, a zaměříme se i na to, v čem se jednotlivé algoritmy liší.

2.1 Formulace problému

Abychom problémy mohli řešit (algoritmicky) pomocí prohledávání, je potřeba je nejdříve formálně zadefinovat. Vhodná formální definice problému se pak obvykle skládá z několika hlavních komponent, kterými jsou

- *iniciální stav*,
- *přechodové akce* (a případně jejich *cena*),
- *cílová podmínka*.

Zmíněné komponenty nám dohromady *implicitně* zadávají takzvaný *stavový prostor* problému, tedy množinu všech stavů dosažitelných z iniciálního stavu. Tyto stavy můžeme interpretovat jako vrcholy grafu, přechodové akce pak určují hrany daného (přechodového) grafu. Díky této interpretaci můžeme řešit různorodé problémy pomocí obecných algoritmů grafového prohledávání. *Řešení problému* je obvykle posloupnost přechodových akcí, která nás dostane z iniciálního stavu do cílového, hledáme tedy cestu v přechodovém grafu. Zajímá-li nás *optimální řešení*, hledáme obvykle takovou cestu, která je nejkratší (nebo nejlevnější) ze všech řešení.

Příklad. Uvažte *problém n dam* z minulé kapitoly. Nejdříve zadefinujte problém formálně. Určete

a) iniciální stav,

b) přechodové akce (a případně jejich cenu),

c) cílovou podmínku.

Až budete mít tuto implicitní definici přechodového grafu, zamyslete jak v takovém případě vypadá graf explicitně (jaká je množina vrcholů a množina hran).

Problém můžeme zadefinovat například následovně.

a) Iniciální stav je prázdná hrací plocha.

b) Přechodová akce je přidání dámy na hrací plochu, pokud ta již neobsahuje n dam. Každá akce má jednotkovou cenu.

c) Cílová podmínka vyžaduje, aby bylo na ploše všech n dam tak, aby se neohrožovaly. Stavový prostor v takovém případě tvoří množina všech rozestavení *0 až n* dam na hrací ploše. Přechodová hrana je mezi stavy s, t právě tehdy, když s obsahuje o jednu dámu méně než t a zároveň t má všechny kromě jedné dámy na stejných pozicích jako s.

Příklad 2.1.1. Zadefinujte formálně problém *8-posunovačky* z přednášky. Zamyslete se i nad tím, jak bude vypadat přechodový graf.

Příklad 2.1.2. Aplikujeme-li na číslo 4 sekvenci operací *faktoriál*, *odmocnina* a *dolní celá část*, můžeme získat libovolné přirozené číslo[1]. Například, číslo 5 lze získat jako

$$\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \rfloor = 5$$

Formulujte tento problém formálně a opět se zamyslete i nad tím, jak bude vypadat přechodový graf. Liší se nějak stavový prostor od předchozích příkladů?

Příklad 2.1.3. Uvažte zjednodušený problém hledání letecké trasy mezi světovými hlavními městy, který musí řešit webové stránky aerolinek. Předpokládejte, že zákazníci hledají takovou (rozumnou) trasu, která je nejlevnější. Zamyslete se, jak byste řádně formulovali tento problém.

Příklad 2.1.4. Uvažujte situaci, kdy se snažíte naplánovat dovolenou po hlavních městech Evropy. Cesta bude začínat i končit v Praze, nejste nijak časově omezeni a máte v úmyslu procestovat právě všechna evropská hlavní města (některá klidně vícekrát). Jak byste řádně formulovali tento problém?

2.2 Algoritmy prohledávání

Existuje mnoho různých strategií jak prohledávat stavový prostor. Výběr vhodného algoritmu závisí na typu problému, našich požadavcích na průběh výpočtu a našich případných dodatečných znalostech o problému. Pokud o problému nemáme žádnou doplňující znalost, která by nám ulehčila práci, obvykle volíme některý z algoritmů *neinformovaného prohledávání*. Naopak, máme-li nějakou dodatečnou informaci, můžeme ji využít například jako základ heuristiky pro některý z algoritmů *informovaného prohledávání*. Obě tyto skupiny si představíme v následujících sekcích.

Jak už jsme naznačili, existuje velké množství strategií, ze kterých můžeme vybírat. Mezi nejdůležitější vlastnosti, podle kterých můžeme nejrůznější algoritmy porovnávat a hodnotit, patří následující čtyři.

[1] Donald Knuth, 1964

Definice 3:

- Algoritmus je *úplný*, jestliže nalezne řešení vždy, když existuje.
- Algoritmus je *optimální*, pokud platí, že nalezne-li nějaké řešení, je toto řešení nejlepší ze všech (například z pohledu ceny nebo délky cesty).
- *Časová složitost* algoritmu udává maximální čas potřebný k vyřešení problému.
- *Prostorová složitost* algoritmu udává maximální množství paměti potřebné k vyřešení problému.

Poznamenejme, že časovou i prostorovou složitost obecně uvažujeme *asymptotickou*, a to v *nejhorším případě*. V některých případech se však může hodit uvažovat i např. složitost očekávanou. Dále, řešíme-li, zda je algoritmus optimální, je třeba myslet na *metriku*, kterou uvažujeme (často je jasná z kontextu). Některé algoritmy mohou být optimální z pohledu hloubky/délky řešení (tedy uvažujeme délku cesty v grafu), jiné zase z pohledu obecné ceny řešení (důležité pro ohodnocené grafy). Pokud uvažujeme verzi prohledávání, kde hledáme řešení více, pak optimalitu můžeme intuitivně chápat tak, že pro každá dvě nalezená řešení algoritmus nalezne dříve to lepší.

Při diskusi o složitosti algoritmů je vždy velmi důležité si uvědomit, vůči čemu ji vyjadřujeme. Jelikož prohledávaný stavový prostor většinou reprezentujeme implicitně (pomocí iniciálního stavu a přechodových akcí), k vyjádření složitosti algoritmů používáme následující metriky.

Definice 4:

- *Faktor větvení* (branching factor) b je maximální počet následníků kteréhokoli uzlu.
- *Hloubka cíle* (goal depth) d je délka nejkratší cesty z iniciálního stavu do některého cílového uzlu.
- *Maximální hloubka* (maximal depth) m je délka nejdelší cesty v grafu.

Příklad. Uvažte přechodový graf pro *problém n dam*, který jsme si definovali v ilustrativním příkladu v Podsekci 2.1. Budeme pracovat s konkrétním grafem pro $n = 8$. Určete pro něj

a) faktor větvení b,

b) hloubku cíle d,

c) maximální hloubku m.

a) Na prázdnou plochu můžeme umístit dámu kdekoli, tedy iniciální stav má 64 následníků. Žádný jiný stav tolik následníků mít nemůže. Proto $b = 64$.

b) K řešení se nemůžeme dostat před přidáním 8 dam. Zároveň existuje rozmístění 8 dam takové, které splňuje cílovou podmínku. Proto $d = 8$.

c) Maximum dam, které můžeme postupně položit je 8, a tedy $m = 8$.

Příklad 2.2.1.
Zamyslete se a pokuste se určit faktor větvení b, hloubku cíle d a maximální hloubku m i pro přechodový graf problému *8-posunovačky* z příkladu v Podsekci 2.1.

2.3 Neinformované prohledávání

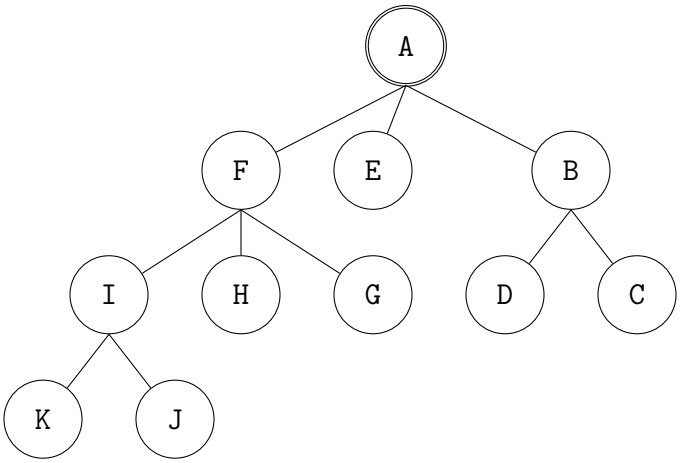
V případě, že o problému nemáme žádnou další informaci (kromě těch z formální definice), jsme obecně nuceni graf prohledávat takzvaně „slepě“. Tyto strategie prohlédavají stavový prostor prostým expandováním uzlů a testováním cílové podmínky. Různé algoritmy se liší například v tom, v jakém pořadí uzly expandují. Nejznámějšími strategiemi jsou *prohledávání do šířky* (BFS) a *prohledávání do hloubky* (DFS). S těmito algoritmy jste se jistě hlouběji setkali již v rámci některého předchozího předmětu.

Prohledávání do šířky je jednoduchá strategie, kde nejprve expandujeme iniciální uzel grafu, poté všechny jeho následníky, poté všechny jejich následníky, a tak dále. Obecně platí, že veškeré uzly v určité vzdálenosti („úrovni“) od iniciálního uzlu jsou prozkoumány předtím, než expandujeme uzly v úrovni o jedna vyšší.

Prohledávání do hloubky naopak nejdříve vždy expanduje jen prvního následníka každého uzlu. Tímto způsobem intuitivně prozkoumá vždy „aktuálně nejhlubší“ neprozkoumaný uzel. Jakmile expanduje listový uzel, vrací se zpět pomocí *backtrackingu* a prozkoumává stejným způsobem další dosud nenavštívené následníky.

Mezi další algoritmy, se kterými budeme pracovat, patří *prohledávání do hloubky s limitem*, *prohledávání podle ceny* (uniform-cost search) a *prohledávání s postupným prohlubováním* (IDS). Jistě je znáte z přednášky.

Příklad. Mějme následující graf, který reprezentuje stavový prostor problému. Uvažujte situaci, kdy postupně spustíme prohledávací algoritmy BFS a DFS ze stavu `A`. V jakém pořadí dané algoritmy navštíví jednotlivé stavy? Máte-li na výběr více následníků, postupujte abecedně.

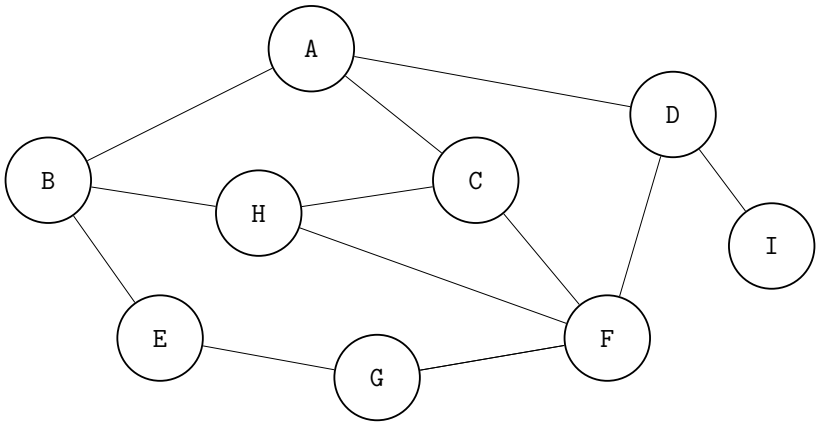


a) BFS: A - B - E - F - C - D - G - H - I - J - K

b) DFS: A - B - C - D - E - F - G - H - I - J - K

Příklad 2.3.1. Uvažujte stejný strom jako v předchozím příkladě. Jak se změní pořadí uzlů, spustíme-li tentokrát oba algoritmy z uzlu `G`?

Příklad 2.3.2. V předchozích příkladech jste si vyzkoušeli simulaci algoritmů na stromu. Uvažte nyní následující graf. Určete v jakém pořadí algoritmy BFS a DFS navštíví jednotlivé stavy, spustíme-li je nejdříve ze stavu `A`, a poté také z `B`.



Opět, máte-li na výběr více následníků, postupujte abecedně.

Příklad 2.3.3. Uvažte opět graf z předchozícho příkladu. Nechť spustíme prohledávání do hloubky z uzlu `I` a nechť cílový uzel je `F`. Uvažujte, že výpočet skončí nalezením prvního řešení.

a) Je alogritmem nalezené řešení optimální?

b) Co kdybychom namísto DFS použili DFS s limitem $l = 4$?

c) A jak by tomu bylo při použití prohledávání s postupným prohlubováním (IDS)?

Příklad 2.3.4. Mějme opět situaci, kdy prohledáváme výše uvedený graf. Tentokrát nechť je iniciální stav `A` a cílový stav `F`. Jako algoritmus zvolíme DFS s limitem. Není-li řečeno jinak, uvažujte, že výpočet skončí nalezením prvního řešení.

a) Jaký je nejvyšší limit, pro který algoritmus nalezne optimální řešení?

b) Od jaké hodnoty limitu algoritmus nalezne vždy stejné řešení?

c) Uvažujte scénář, kdy prohledáváme celý graf (výpočet neskončí prozkoumáním cílového stavu). Od jaké hodnoty limitu bude výpočet pro DFS s limitem probíhat zcela stejně jako pro klasické DFS?

Příklad 2.3.5. Vymyslete příklady přechodových grafů (případně přímo problémů) takových, že:

a) Prohledávání do hloubky nikdy nenalezne řešení.

b) Prohledávání do hloubky nalezne řešení dříve než prohledávání do šířky.

c) Prohledávání do hloubky bude procházet uzly ve stejném pořadí jako prohledávání do šířky.

Každou odrážku řešte zvlášť.

Příklad 2.3.6. Rozhodněte o pravdivosti následujících tvrzení, své odpovědi odůvodněte. Můžete předpokládat že prohledáváme strom s konečným faktorem větvení, kladnou cenou přechodů a alespoň jedním dosažitelným cílovým uzlem.

a) Prohledávání do hloubky s limitem je úplné.

b) Prohledávání s postupným prohlubováním má stejnou prostorovou složitost jako prohledávání do hloubky.

c) Prohledávání s postupným prohlubováním má horší časovou složitost než prohledávání do šířky.

d) Prohledávání do šířky je optimální.

Příklad 2.3.7. Popište stavový prostor takový, že prohledávání s postupným prohlubováním má daleko větší složitost než prohledávání do hloubky (jako například $\mathcal{O}(n^2)$ vs. $\mathcal{O}(n)$).

Příklad 2.3.8. Uvažte stavový prostor, kde iniciální stav je číslo 1 a každý stav k má dva následníky – čísla $2k$ a $2k + 1$.

a) Načrtněte část stavového prostoru pro stavy 1 až 15.

b) Předpokládejte, že cílový stav je 11. Určete v jakém pořadí algoritmy BFS, DFS s limitem $l = 3$ a prohledávání s postupným prohlubováním navštíví jednotlivé stavy.

c) Pokuste se nalézt algoritmus, který vypíše řešení bez jakéhokoli prohledávání. Využijte znalostí o doméně a formulaci problému.

2.4 Heuristické prohledávání

Máme-li o problému nějakou dodatečnou informaci, často ji můžeme využít k efektivnějšímu prohledávání stavového prostoru. Konkrétně se budeme zabývat strategiemi, které při výběru

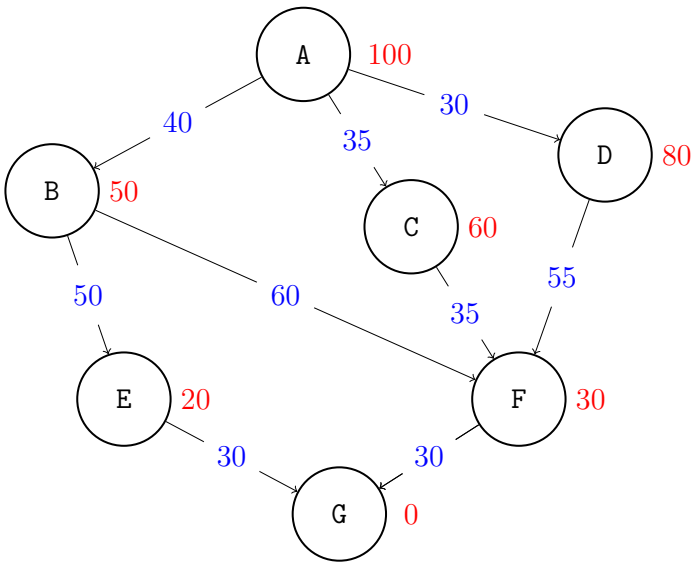
uzlů k expandování využívají informaci o (odhadu) blízkosti stavů k cíli. Tyto algoritmy obvykle expandují nejdříve ty uzly, které se jeví (v určitém smyslu) jako *nejslibnější*. Souhrně se tyto prohledávácí strategie označují jako *best-first search*. Informaci o tom, jak přínosný se jeví daný uzel (jaký je odhad jeho ceny), nám dává takzvaná *ohodnocovací funkce* $f(n)$. V průběhu výpočtu si držíme prioritní frontu uzlů uspořádaných dle této ceny a expandujeme stav s aktuálně nejmenší cenou.

Jako komponentu při odhadu ceny většina algoritmů používá *heuristickou funkci* $h(n)$. Heuristika může například odhadovat cenu cesty z uzlu k cíli. Aby byl tento odhad užitečný, některé algoritmy kladou na použité heuristiky podmínky. Jednou z nich je takzvaná *přípustnost*. Intuitivně, heuristický odhad ceny cesty z uzlu do cíle nesmí být větší než cena opravdová. V jistém ohledu silnější podmínkou je pak *konzistence* heuristiky.

Definice 5:

1. Heuristická funkce h je *přípustná* pokud pro všechny stavy n platí $0 \leq h(n) \leq h'(n)$, kde $h'(n)$ je skutečná cena cesty ze stavu n do cíle.
2. Heuristická funkce h je *konzistentní* (neboli *monotonní*) pokud pro každý stav n a každého jeho následníka m platí, že $h(n) \leq c(n,m)+h(m)$, kde $c(n,m)$ je cena přechodu z n do m.

V několika dalších demonstrativních příkladech budeme pracovat s následujícím ohodnoceným grafem reprezentujícím stavový prostor.



Pro daný graf platí, že ohodnocení přechodů jsou zobrazena modře, například `A-B` má cenu 40. Červenou barvou pak jsou pak zaznačeny hodnoty heuristiky pro každý stav, například heuristická hodnota stavu `A` je 100. Počáteční stav je `A`, cílový stav je `G`.

Stručně si představme dvě konkrétní strategie informovaného prohledávání. První je známa jako *greedy best-first search*, neboli *hladové heuristické hledání*. Druhý algoritmus se pak nazývá *A**. Obě strategie mají společné to, že si v průběhu výpočtu udržují prioritní frontu s uzly k expandování, uspořádanými podle hodnoty $f(n)$. V čem se algoritmy liší, je vzorec, pomocí kterého počítají $f(n)$. Tento rozdíl se zdá být malý, ale celkově jsou jak úvahy za oběma algoritmy, tak i jejich vlastnosti, velmi odlišné.

Greedy best-first search jednoduše ohodnocuje uzly jen podle hodnoty heuristiky, $f(n) = h(n)$. Algoritmu se říká *hladový*, protože se v každém kroku snaží dostat tak „blízko“ k cíli, jak jen to jde. To vede například k tomu, že algoritmus není obecně úplný ani optimální.

A* je sofistikovanější algoritmus. Kombinuje užitečné vlastnosti *Dijkstrova algoritmu* (doporučujeme si zopakovat) a zároveň rychlost heuristického prohledávání. Při ohodnocování uzlu bere kromě heuristiky v potaz i dosavadní cenu cesty, kterou jsme se do uzlu dostali, označme ji $g(n)$. Ohodnocovací funkce má tedy tvar $f(n) = g(n) + h(n)$. Použijeme-li heuristiku s vhodnými vlastnostmi, algoritmus je pak garantovaně optimální.

Příklad. Uvažujte situaci, kdy prohledáváme výše uvedený přechodový graf pomocí hladového heuristického prohledování.

a) V jakém pořadí navštíví algoritmus jednotlivé stavy?

b) Jak bude vypadat výsledná nalezená cesta? Je toto řešení optimální?

Pořadí stavů je `A-B-E-G`, což je v tomto případě zároveň i výsledná cesta. Toto řešení ovšem optimální není. Zkuste se zamyslet nad lepší cestou.

Příklad. Nyní uvažujte prohledávání daného grafu pomocí algoritmu A*.

a) Je heuristika zvolená v daném příkladu přípustná? A je konzistentní?

b) V jakém pořadí navštíví A* jednotlivé stavy?

c) Jak bude vypadat výsledná nalezená cesta? Liší se od cesty nalezené hladovým algoritmem? Je řešení tentokrát optimální?

a) Heuristika je přípustná, jelikož pro každý stav je hodnota heuristiky menší nebo rovna ceně cesty do cílového stavu. Není ale konzistentní, jelikož $h(A) > c(A, C) + h(C)$.

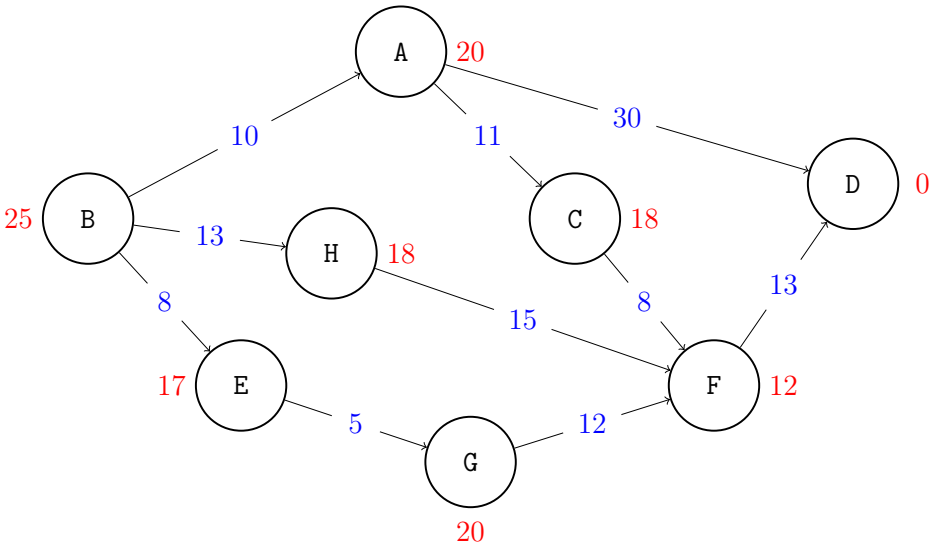
b) Pořadí stavů při výpočtu je `A-B-C-F-G`.

c) Výsledná cesta má tvar `A-C-F-G`, a tedy se liší od cesty nalezené hladovým algoritmem. Můžete si ověřit, že toto řešení je opravdu optimální.

Příklad 2.4.1. Uvažme následující stavový prostor. Iniciální stav je `B`, cílový stav je `D`. Ohodnocení přechodů jsou opět zobrazena modře, hodnoty heuristiky červeně. Simulujte výpočet hladového heuristického algoritmu.

a) V jakém pořadí navštíví algoritmus jednotlivé stavy?

b) Jak bude vypadat výsledná nalezená cesta? Je optimální?



Příklad 2.4.2. Opět simulujte prohledávání grafu z předchozího příkladu, tentokrát ale pomocí algoritmu A*.

a) Je heuristika zvolená v daném příkladu přípustná? A je konzistentní?

b) V jakém pořadí navštíví A* jednotlivé stavy?

c) Jak bude vypadat výsledná nalezená cesta?

Příklad 2.4.3. Uvažujte, že věž se může na šachovnici pohybovat o jakýkoli počet políček po přímce, vertikálně nebo horizontálně, ale nemůže přeskakovat ostatní figurky. Je Manhattanská vzdálenost přípustná heuristika pro problém posunutí věže z políčka A na políčko B v co nejmenším počtu tahů? Svou odpověď odůvodněte.

Příklad 2.4.4. Rozhodněte o pravdivosti následujících tvrzení. Odpovědi zdůvodněte. Pokud není řečeno jinak, uvažujte konečný faktor větvení, cenu přechodů vyšší než nějaké kladné ϵ a alespoň jeden dosažitelný cílový uzel.

a) Hodnota přípustné heuristiky nikdy nepřevyšuje zbylou opravdovou cenu (vzdálenost) do cíle.

b) Algoritmus heuristického prohledávání, jehož fronta je uspořádána dle hodnoty $f(n) = g(n) + h(n)$ je úplný i optimální, pokud používá přípustnou heuristiku a zároveň ohodnocení uzlů $f(n)$ monotónně stoupá po jakékoliv cestě do cíle.

c) Prohledávání podle ceny je jak úplné, tak i optimální, pokud cena cesty nikdy neklesá.

d) Hladové heuristické prohledávání je jak úplné, tak i optimální, pokud je použitá heuristika přípustná a cena cesty nikdy neklesá.

Příklad 2.4.5. Která z následujících tvrzení jsou pravdivá? Odpovědi zdůvodněte.

a) Prohledávání do hloubky vždy expanduje alespoň tolik uzlů jako A* s přípustnou heuristikou.

b) $h(n) = 0$ je přípustná heuristika pro 8-posunovačku.

c) Prohledávání do šířky je úplné i pokud jsou povoleny přechody s nulovou cenou. Uvažujeme konečný faktor větvení.

d) Součet několika přípustných heuristik je opět přípustná heuristika.

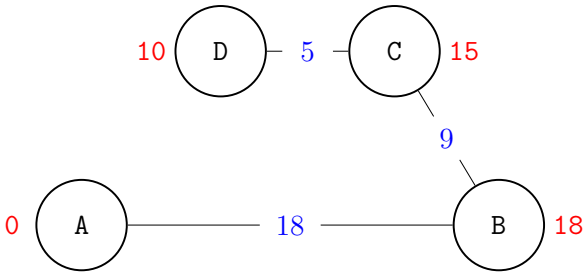
Příklad 2.4.6. Uvažujte prohledávání (konečného) stavového prostoru algoritmem A* s konzistentní heuristikou. Nechť existuje jeden dosažitelný cílový uzel n_g a nechť $C*$ je cena optimálního řešení.

a) Může se stát, že algoritmus neexpanduje některý uzel n_1, pro který platí $f(n_1) < C*$?

b) Může algoritmus prohledat jen a pouze uzly n takové, že $f(n) < C*$?

c) Můžeme před spuštěním výpočtu určit, kolik uzlů m takových, že $f(m) > C*$, bude expadováno?

Příklad 2.4.7. Mějme následující stavový prostor. Ohodnocení přechodů jsou jako obvykle zobrazena modře, hodnoty heuristiky červeně, cílový stav je `A`. Simulujte prohledávání pomocí greedy best-first tree-search algoritmu ze stavu `C`. Jaké bude pořadí prohledávaných uzlů?



Poznámka – tree-search verze prohledávacích algoritmů neřeší cykly.

Příklad 2.4.8. Jsou následující tvrzení ohledně heuristik pravdivá? Své odpovědi dokažte.

a) Každá přípustná heuristika je i konzistentní.

b) Každá konzistentní heuristika je i přípustná.

Příklad 2.4.9. Rozhodněte a dokažte, zda jsou následující tvrzení o prohledávacích algoritmech pravdivá.

a) BFS je speciální případ prohledávání podle ceny (uniform-cost search).

b) Algoritmy DFS, BFS i uniform-cost search jsou speciálními případy best-first tree search.

c) Prohledávání podle ceny je speciální případ A* prohledávání.

3 Dekompozice problému, Problémy s omezujícími podmínkami

V této kapitole prozkoumáme aplikace prohledávání grafů, kterému jsme se věnovali v kapitole předchozí. Konkrétně si ukážeme, co je AND/OR graf a na jaké široké spektrum problémů ho lze aplikovat – od problémové dekompozice, přes hry až po modelování logických výrazů.
Ve druhé části se budeme zabývat deklarativním přístupem k programování, který umí být překvapivě elegantní a stručný. Zjistíme spolu, že na jeho pozadí se však opět vykonává prohledávání stavového prostoru, které pro mnoho příkladů funguje překvapivě dobře, jindy však narazíme na hranice efektivity.

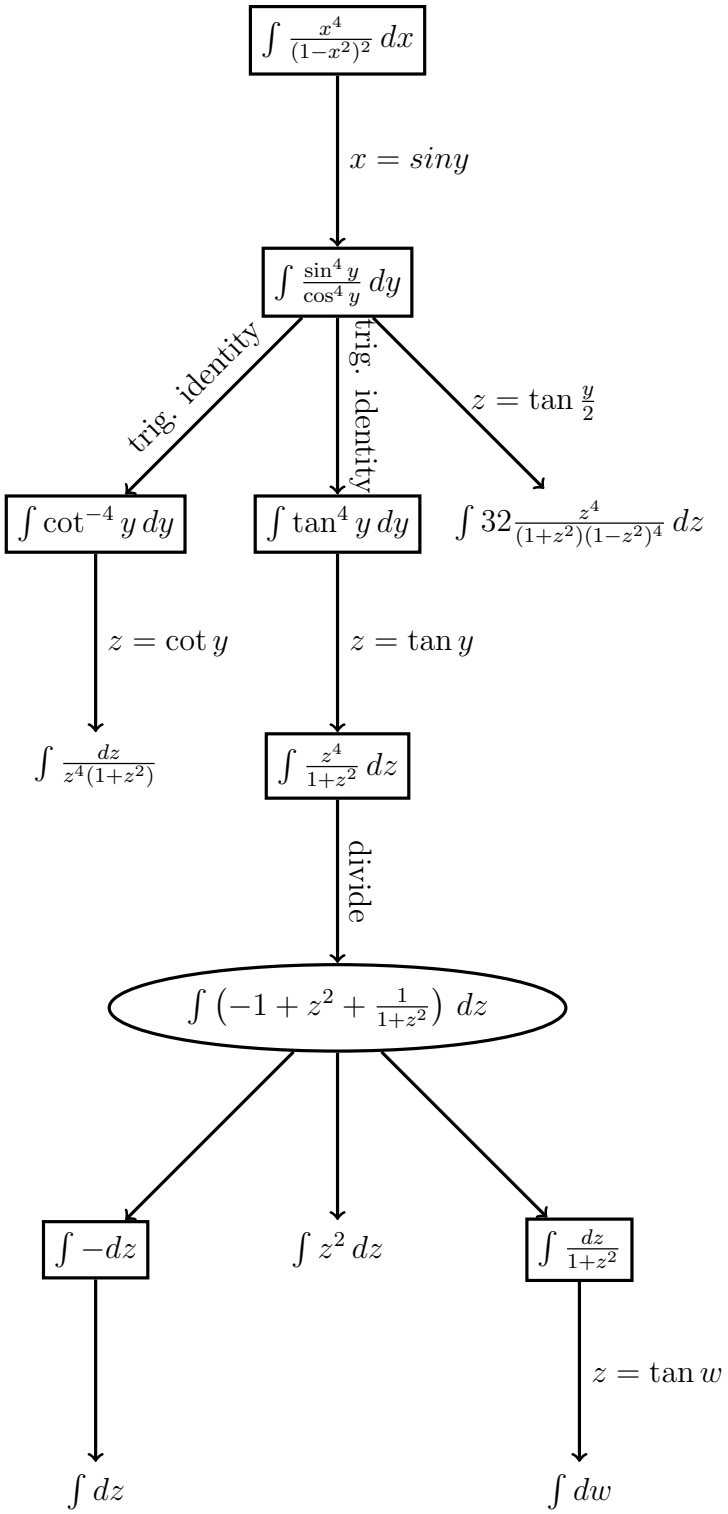
3.1 Dekompozice problému a AND/OR grafy

Definice 6: *AND/OR graf* je orientovaný graf s vrcholy typu AND nebo OR (souhrnně zvané vnitřní) a sérií koncových vrcholů $t_1, t_2, \ldots, t_n$.
Typ vnitřního vrcholu nejčastěji interpretujeme tak, že k vyřešení OR uzlu je potřeba vyřešit kterýkoliv z následníků (potomků). V případě AND uzlu musíme vyřešit všechny následníky (potomky). Koncové vrcholy pak reprezentují dále nedělitelné podproblémy, ať už neřešitelné či se známým řešením. V jiném kontextu lze koncové vrcholy chápat jako splněné či nesplněné.

Příklad. Rozdělení na vnitřní a koncové uzly v definici není potřeba. Zamyslete se, jak ji upravit tak, abychom s pomocí vnitřních vrcholů mohli modelovat i uzly koncové.

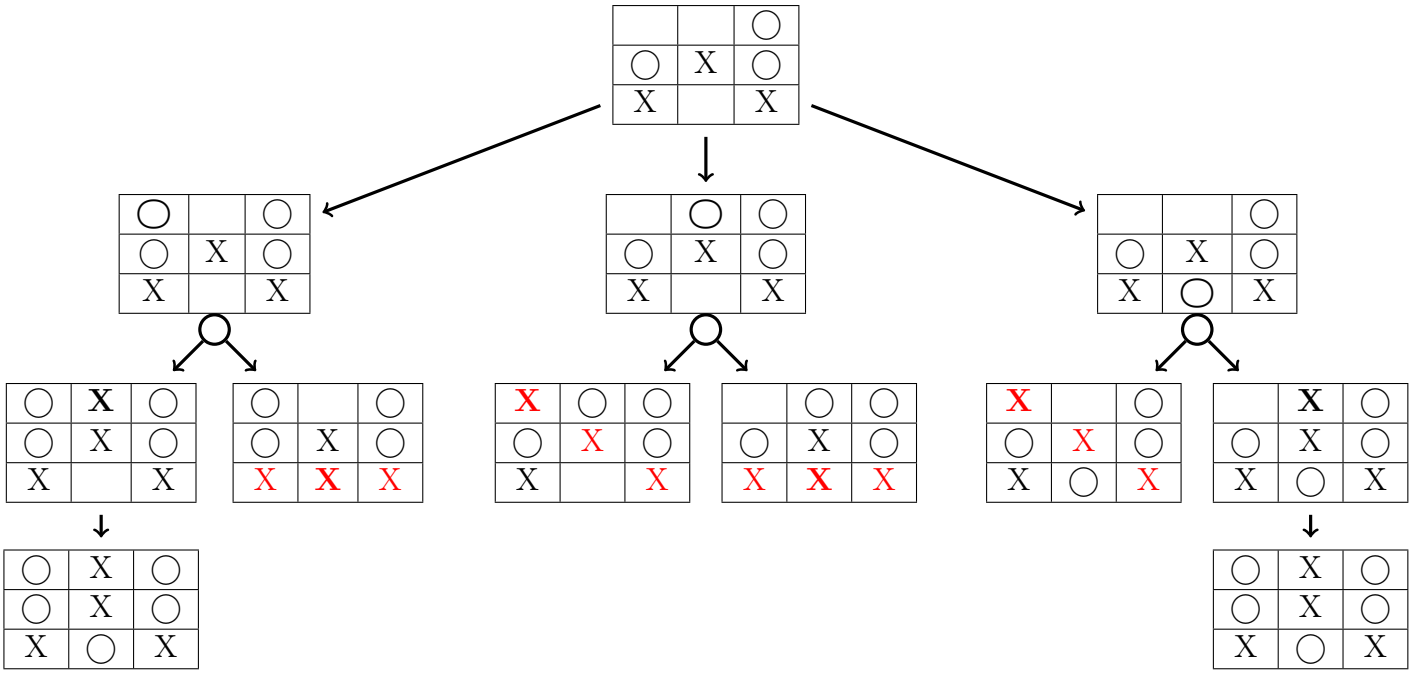
Stačí chápat AND uzel bez následníků jako splněný (řešitelný), zatímco vrchol typu OR bez následníků jako triviálně nesplněný. Listům bychom tedy dali typ AND nebo OR podle toho, zda je chceme mít splněné či nikoliv.

Jako příklad aplikace takového přístupu na problémovou dekompozici (a tedy i příklad AND/OR grafu) si uvedeme strategii řešení neurčitého integrálu. Postup, který si tu budeme spolu ilustrovat, se v nějaké míře nachází u většiny matematických symbolických řešičů – jako je například WolframAlpha.
Na následujícím obrázku značíme *AND vrcholy elipsou*, zatímco *OR vrcholy obdélníkem*. Koncové uzly necháváme bez označení. Vrcholy s jediným následníkem mohou být AND i OR bez vlivu na sémantiku (význam) grafu – my v takovém případě volíme konzistentně obdélníkové ohraničení. OR vrcholy zde tedy reprezentují možné přístupy k řešení, kdežto AND uzel rozděluje problém na sadu podproblémů, které je třeba vyřešit všechny. V některých případech, kde je orientace hran grafu zřejmá (například z rozmístění vrcholů), se šipky v grafu vynechávají.



Častěji budeme vídat AND/OR grafy v kontextu tahových her dvou hráčů – například piškvorek. Omezíme se na *hry s perfektní informací*, což stručně znamená, že vždy známe úplný aktuální stav hry. Mimo piškvorek toto splňují například šachy či go. Uvědomme si, že tímto způsobem bychom nemohli modelovat třeba poker či kostky.
Na následujícím obrázku používáme pro přehlednost malé kolečko pod vrcholy, které jsou

typu AND. Uzly OR a koncové vrcholy jsou neoznačené, neboť jsou rozlišitelné z kontextu. Takové značení můžeme potkat i později tam, kde by prvně dohodnuté znepřehledňovalo nákresy.



V našem AND/OR grafu jsme vhozeni do již rozehrané piškvorkové partie na omezeném hracím poli 3×3. Na tahu je hráč kreslící kolečka a zvažuje všechny své možné tahy. Stačí mu, aby pouze jeden z jeho tahů byl výherní (či alespoň končící remízou) – a tedy vrchol grafu reprezentující aktuální stav hracího pole je typu OR. O úroveň níže zvažujeme soupeřovu odpověď na náš tah. Nás soupeř je velmi dobrý hráč a nedělá chyby, proto potřebujeme, aby *žádná* z jeho reakcí nevedla k naší prohře. Všechny uzly ve druhé úrovni (reprezentující odehrání jednoho našeho tahu) jsou proto typu AND. Při delší hře dvou hráčů se i nadále typy uzlů střídají podle vzdálenosti od počátku – jsme-li na tahu my, jsou typu OR, a naopak typu AND, je-li na tahu protihráč.

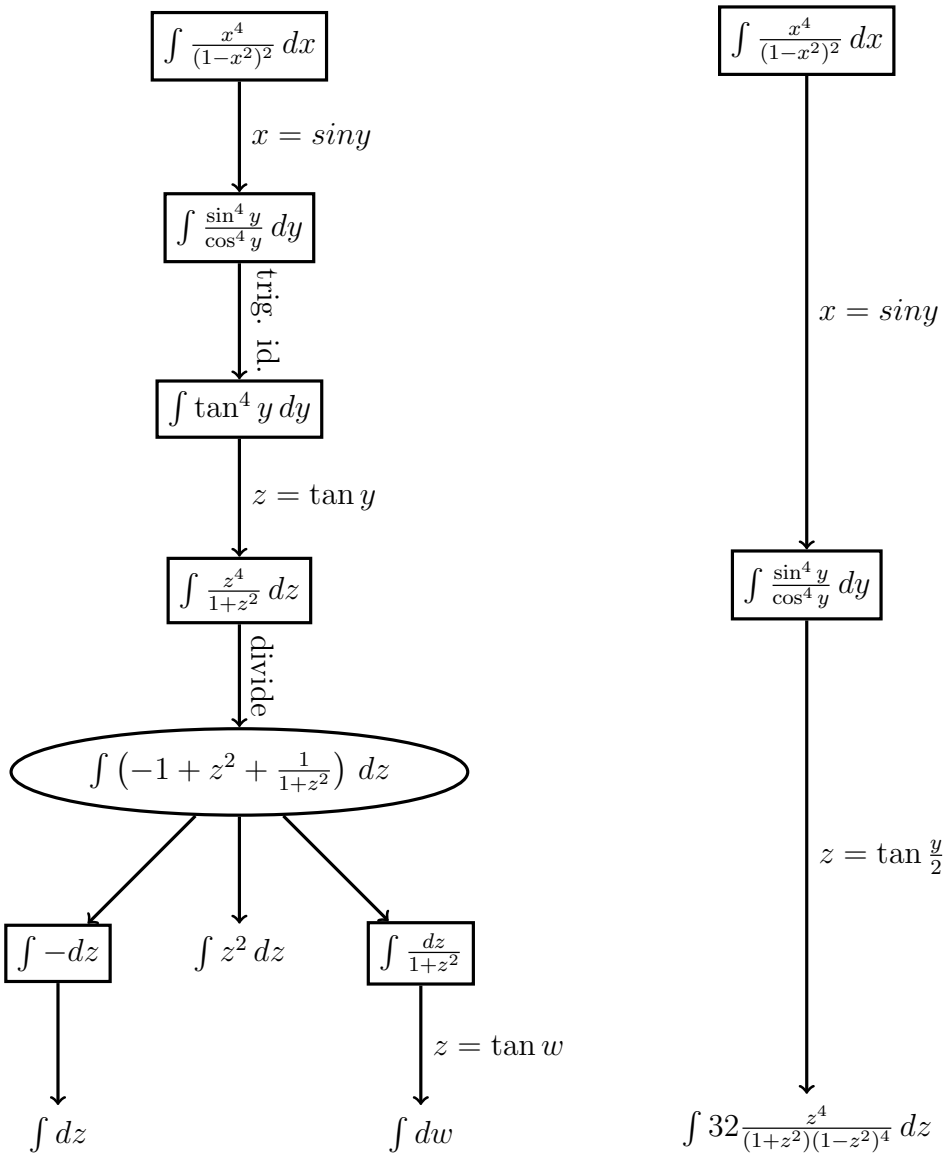
Po pečlivé analýze námi vykresleného AND/OR grafu zjišťujeme, že za předpokladu, že náš soupeř neudělá chybu, prohráli jsme. Každý z AND uzlů na druhé úrovni je nesplněný, neboť alespoň jeden jeho následníků značí naši prohru. Startovní OR vrchol reprezentující aktuální stav hracího pole je tedy také nesplněný, ačkoliv by bylo možné (s nedokonalým soupeřem) dostat se do stavu remízy.

Uvedený příklad je skutečně hodně jednoduchý. V první řadě nevede žádná možnost k naší výhře, v nejlepším případě můžeme doufat jen v remízu. Stojí za zmínku, že typicky se piškvorky nehrají na omezeném hracím poli a i pokud ano, naší naivní metodou vytvořený AND/OR graf by pro běžný čtverečkovaný papír velikosti A4 obsahoval více vrcholů, než je atomů v celém pozorovatelném vesmíru.

Definice 7: *Stromem řešení T problému P s AND/OR grafem G* je podgraf grafu G, který je stromem a

- jeho kořen je vrchol reprezentující problém P,
- je-li N vnitřní uzel T typu AND, pak každý jeho následník v G je i v T,
- je-li N vnitřní uzel T typu OR, pak právě jeden z jeho následníků v G je i v T.

Pro náš příklad s výpočtem integrálu může strom řešení vypadat například ilustrovanými dvěma způsoby. Praktický význam má však spíše prvně vykreslený strom, kde koncové vrcholy umíme snadno vyřešit.



Příklad. Seznam se s dekompozicí problému Hanojských věží z přednášky, kde problém

redukujeme na jednoduchý AND/OR graf, kde jsou všechny vnitřní uzly typu AND. Ke kolika fyzickým přesunům disku dojde při počtu kotoučů n rovno

a) 1,

b) 3,

c) 4?

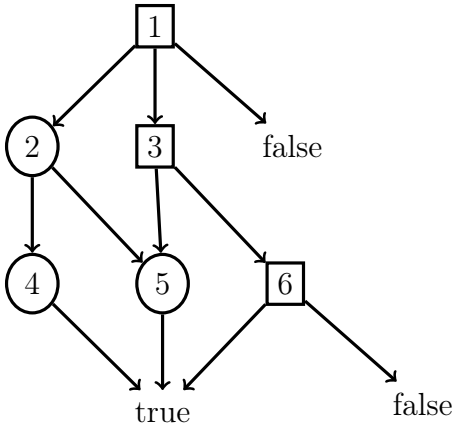
a) 1,

b) 7,

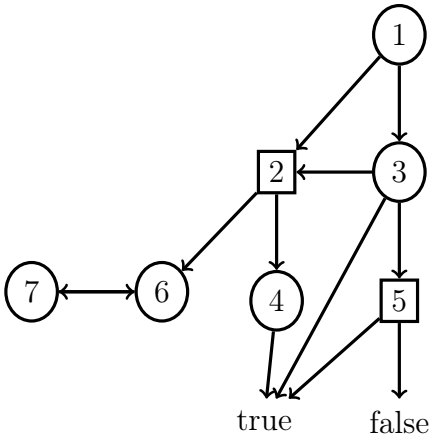
c) 15

Příklad. Rozhodněte, zda je počáteční uzel, značený 1, splněný v následujících AND/OR grafech. Vrchol chápeme jako splněný tehdy, když pro něj existuje v daném grafu strom řešení se všemi konečnými vrcholy nastavenými na *true*. Spočtěte také, kolik takových stromů řešení existuje.

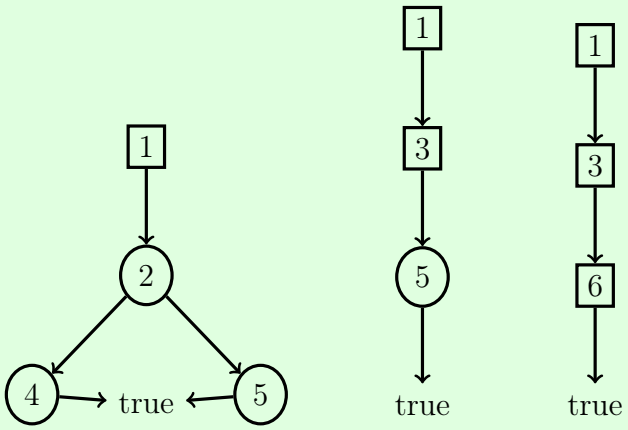
a)



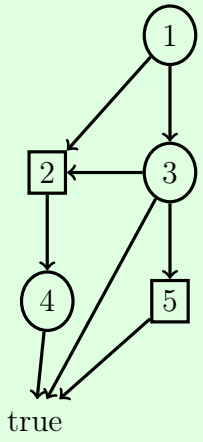
b)



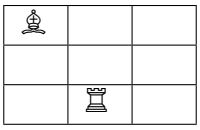
a) Ano, svoji odpověď můžeme potvrdit ukázkou příslušného stromu řešení. My ukazujeme všechny 3 možné stromy řešení.



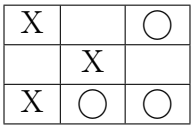
b) Opět ano. Tentokrát je však vyhovující strom řešení pouze jeden. Všimněte si smyčky mezi AND vrcholy 6 a 7, kterou v tomto textu ukazujeme především proto, aby čtenář nezapomněl, že jsou povolené a možné. Učit se s nimi nakládat však nebudeme a konkrétně zde využíváme toho, že ji lze zcela ignorovat díky předchozímu OR uzlu 2.



Příklad 3.1.1. Uvažme podivnou hru odehrávající se na hracím poli velikosti 3×3. Hráči se střídají a každý posouvá svou figuru podle pravidel šachu, nesmí však vstoupit na pole, které již bylo v minulosti obsazeno. Podaří-li se hráči sebrat soupeři figuru, vyhrává, naopak je-li hráč na tahu a nemůže se nikam pohnout (neboť dosažitelná pole již byla v minulosti obsazena), prohrává. Úkolem je zanalyzovat situaci, kde proti sobě hraje věž a střelec, první zmíněný je na tahu a počáteční rozmístění je dáno obrázkem níže. Zkonstruuj příslušný AND/OR graf a urči, kdo v takovém případě vyhraje.



Příklad 3.1.2. Uvažte rozehranou partii piškvorek, kde je právě na tahu hráč kreslící kolečka. Sestrojte AND/OR graf a okomentujte stromy řešení tohoto grafu.



3.2 Problémy s omezujícími podmínkami

Zlatým grálem programování měl být řešič programů s omezujícími podmínkami, častěji zkracováno jako CSP (constraint satisfaction problem). Je typickým příkladem deklarativního programování, které si můžeme představit tak, že deklarujeme, co si přejeme, aby bylo splněno, a umělá inteligence to má za úkol zařídit.
Na přednášce jste již jako příklad takového problému viděli problém obarvení grafu, algebrogram a problém N dam. Jako jiný příklad je vhodné uvést typovou inferenci – tedy automatické otypování funkcí a výrazů, které znáte třeba z Haskellu či Pythonu. Podívejme se na tento jednoduchý program.

```
x = f(g(0))
x = f(x)
x = g(x)
```

Jeho analýzou získáme sérii omezení (též podmínek, angl. constraints), jejich konkrétní podoba závisí na uvažovaném typovém systému, zde se pokusíme alespoň o intuitivní náčrt.

- `f` bere jako argument stejný typ, který vrací `g`
- `g` bere argument typu `int`
- `f` vrací stejný typ, který bere
- `g` bere jako argument stejný typ, který vrací `f`

Inteligentní řešič by nám po krátkém zpracování těchto omezení řekl, že obě uvedené funkce jsou typu `int` $\to$ `int`.
Jako názornější příklad uvedeme sudoku.

	2						9	
3		1	9		6	5		2
			8		4			
	9						5	
5			2		3			6
	7						2	
			4		7			
8		2	5		1	7		3
	5						8	

Chceme vytvořit sadu omezení pro naši umělou inteligenci, aby nám poskytla řešení tohoto hlavolamu. Jenže co do našich omezení napsat? První krok je vždy rozmyslet proměnné našeho problému, tedy hodnoty, které bych rád od řešiče získal. Zde se přímo nabízí zavést si sérii proměnných $x_1, \ldots, x_n$, kde každá reprezentuje hodnotu prázdného políčka a n je počet prázdných políček. Typicky při deklaraci proměnné zároveň říkáme, jakých můžeme nabývat hodnot – tedy specifikujeme její doménu. Z pravidel sudoku je jasné, že každá proměnná x_i může nabývat celočíselných hodnot mezi 1 a 9.

Dále víme, že budeme muset vytvořit omezení pro každý sloupec, řádek i čtverec. Pokud si pojmenujeme prázdná políčka popořadě po řádcích, omezení pro první řádek by vypadala jako: $x_1 \neq 2, x_1 \neq x_2, \ldots, x_2 \neq 2, x_2 \neq x_3, \ldots$. Mnoho programovacích jazyků nabízí místo této série 35 podmínek psát pouze něco ve smyslu `ALL_DISTINCT`$(x_1, x_2, x_3, \ldots, x_7)$, při omezení domén těchto 7 proměnných na $\{1, 3, 4, 5, 6, 7, 8\}$. To je nejen výrazně úspornější na zápis, ale především na pozadí schovaný řešič dostane možnost optimalizovat pro toto často se vyskytující omezení. Tento proces bychom opakovali pro každý řádek, sloupec i čtverec a spustili hledání řešení.

Na začátku této sekce jsme zmínili, že takovýto styl programování měl být zlatým grálem informatiky. Není těžké si představit proč – místo psaní dlouhého kódu aplikace bych jen zavedl sérii podmínek jako „když kliknu sem, stane se...“. Takové deklarativní programování by samozřejmě bylo poněkud složitější, než tu ukazujeme, ale to je nakonec imperativní kód ještě mnohem víc. Kde je tedy problém? Pozorný čtenář již jistě uhodl, že v samotném řešiči. Skutečně efektivní umíme sestrojit pro určité typy omezení, všude jinde spoléháme na chytré heuristiky při prohledávání všech možných ohodnocení proměnných.

Pro zvídavého čtenáře může být poněkud překvapivé i to, že takové řešení je problematické i tehdy, omezíme-li domény všech proměnných na binární $\{0, 1\}$. *Pokud totiž naše omezení píšeme jazykem výrokové logiky, chceme po řešiči, aby řešil NP-úplný problém známý jako SAT.*

Praktické využití má však dnes CSP při rozvrhování (používá ho i naše fakulta) či konstrukci mikročipů. S jeho pomocí lze modelovat kromě zmíněné typové inference a logických hádanek také maximální tok grafem či lexikální analýzu.

Po nepříliš stručném úvodu jsme připraveni si tyto nové pojmy definovat formálněji.

Definice 8: *Problém s omezujícími podmínkami (CSP)* je

- soubor proměnných $X_1, \ldots, X_n$, každá s neprázdnou doménou $D_1, \ldots, D_n$;
- soubor omezení $C_1, \ldots, C_m$; každé omezení je podmnožinou $D_1 \times \ldots \times D_n$;
- (někdy) účelová funkce $f : D_1 \times \ldots \times D_n \to \mathbb{R}$.

Řešením nazveme takovou n-tici $(x_1, \ldots, x_n) \in D_1 \times \ldots \times D_n$, která splňuje všechna omezení C_i, tj. $\forall i.(x_1, \ldots, x_n) \in C_i$.
Má-li CSP více než jedno řešení, může nás zajímat některé konkrétní, potom využíváme účelové funkce f a hledáme takové řešení, které funkci maximalizuje (či minimalizuje). V některých zdrojích pak takový problém značíme COP (*constraint optimization problem*).

Příklad. Uvažme následující program. (Syntax neodpovídá přesně syntaxi z přednášky, ale je dostatečně intuitivní na to, aby nepotřebovala vysvětlení.)

```
X in 1..7
Y in 1..7
X + Y = 7
```

a) Kolik má takový problém řešení?

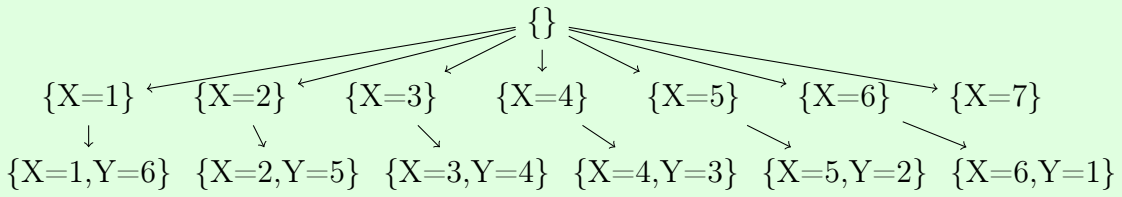
b) Kolik má řešení, přidáme-li omezení `X * Y is even`?

c) Jak vypadá příslušný graf stavů, který prohledáváme?

a) 6 – konkrétně `{(X=1,Y=6),(X=2,Y=5),(X=3,Y=4),(X=4,Y=3),(X=5,Y=2),(X=6,Y=1)}`.

b) 6, jelikož vždy právě jedna proměnná `X`, `Y` musí být sudá, aby v součtu daly 7. Přidané omezení tedy nemění zadání.

c) Začínáme v prázdném stavu, kde neznáme hodnotu ani jedné proměnné. Z něj se můžeme dostat do stavu, kde hodnota proměnné `X` je celé číslo mezi 1 a 7, jelikož jedině tyto stavy jsou v souladu se zadáním. Z každého z těchto stavů, kde známe pouze hodnotu proměnné `X` se však již dostaneme v souladu s omezeními v zadání do nanejvýš jediného stavu, kde je dána i hodnota proměnné `Y`. Všimněte si, že v jednom uzlu (`{X=7}`) naše cesta končí, z něj se nedokážeme dostat při splnění podmínek programu. Přísně vzato v našem grafu existuje mnoho jiných vrcholů – například `{X=3,Y=3}` – ty nekreslíme, neboť do nich nevede žádná cesta z počátečního stavu, a tak jsou pro naše prohledávání irelevantní.



Příklad 3.2.1. Sestavte graf stavů pro následující CSP. Proměnné přiřazujte v sekvenčním pořadí podle jejich deklarace. Popište řešení.

```
A in 2..4
B in 2..3
C in 0..6

A - B >= C
A * (B-1) != B + C
A != B
```

Příklad 3.2.2. Převeď následující algebrogram na CSP. Každé písmenko P v rozepsaném součinu zastupuje prvočíselnou číslici – ne však nutně *stejnou*. Řešení hledat nemusíš (ačkoliv je to s trochou přemýšlení a zkoušení možné). Nezapomeň specifikovat zavedené proměnné a jejich domény.

```
   PPP
    PP
 -------
  PPPP
 PPPP
 -------
 PPPPP
```

Příklad 3.2.3. Jako kouzelný čtverec označíme čtvercová matice, kde součet čísel na každém řádku a v každém sloupci je stejný. Jednotlivé buňky mohou nabývat celočíselných hodnot mezi 1 a n^2, kde n je dimenze uvažované matice.

Jedním z netriviálních řešení pro $n = 3$ je tento kouzelný čtverec.

2	9	4
7	5	3
6	1	8

Formulujte tento problém jako CSP pro uvedené $n = 3$. Kromě omezení nezapomeňte uvést význam zavedených proměnných a jejich domény.

Příklad 3.2.4. Latinský čtverec o velikosti n je čtvercová matice $n \times n$, obsahující v každém řádku (a stejně i sloupci) neopakující se čísla $1, \ldots, n$. Každý řádek i sloupec je tedy permutace na množině $\{1, \ldots, n\}$.

Uvádíme konkrétní příklad pro $n = 3$.

1	2	3
2	3	1
3	1	2

Formulujte tento problém jako CSP pro uvedené $n = 3$. Kromě omezení nezapomeňte uvést význam zavedených proměnných a jejich domény.

Příklad 3.2.5. Zajímavou aplikací CSP je známá Einsteinova hádanka, někdy nazývána zebra. Jejího autora neznáme, často se za něj uvádí Albert Einstein či Lewis Carroll. Týká se pěti sousedů, kteří žijí na ulici v domech v řadě vedle sebe. Každý z nich má jiné povolání, jiný dům, chová jiné zvíře, je jiné národností a preferuje jiné pití. Máme k dispozici několik výroků.

- Angličan žije v červeném domě.
- Španěl chová psa.

- Japonec je malíř.
- Ital pije čaj.
- Nor žije v prvním domě nalevo.
- Majitel zeleného domu pije kávu.
- Zelený dům je bezprostředně napravo od bílého.
- Sochař chová šneky.
- Diplomat žije ve žlutém domě.
- V prostředním domě pijí mléko.
- Nor žije vedle modrého domu.
- Houslista pije ovocný džus.
- Liška žije vedle doktora.
- Kůň je ustájen v domě vedle diplomatova.

K úloze patří i otázka. Kdo má zebru a kdo pije vodu?
Vašim úkolem není tuto úlohu vyřešit (ačkoliv to není těžké), ale převést ji na CSP, z jehož řešení bychom uměli odpovědět na zadanou otázku.

Příklad 3.2.6. Zamyslete se, jak formulovat jako CSP problém rozmístění jedné sady šachových figurek po klasické šachovnici velikosti 8×8 tak, aby se žádné dvě figurky neohrožovaly. Pěšce nemusíte uvažovat, stačí tedy umístit krále, dámu, 2 střelce, 2 koně a 2 věže.

4 Hry a herní strategie

V této kapitole naučíme umělou inteligenci řešit deterministické hry dvou hráčů s úplnou informací. Ukážeme si, že jednoduchý algoritmus MINIMAX je pro složitější hry jako jsou piškvorky na dostatečně velké hrací ploše či šachy zcela nevhodný, neboť prohledávaný stavový graf je enormní. Následně spolu MINIMAX zefektivníme při zachování korektnosti, a objevíme tak ALFA-BETA prořezávání.

Nakonec se ještě krátce zamyslíme nad tím, jak bychom námi nalezené postupy mohli aplikovat i na nedeterministické hry dvou hráčů.

4.1 MINIMAX

Zabývejme se nyní spolu *tahovou hrou pro dva hráče s perfektní informací*. Dobrým příkladem takové hry, která nás již nějakou dobu touto sbírkou provází, jsou piškvorky. Nejprve se spolu podíváme na hru, kde nám přestává stačit AND/OR graf, a poté přejdeme k MINIMAX grafu.

Pokud chceme modelovat hru s komplikovanějším vyhodnocením než jen výhra či prohra, tedy takovou, kde končíme s nějakým skóre, AND/OR stromy nám to neumožní. Koncové vrcholy totiž mohou být jen splněny/nesplněny, tj. výhra či prohra, nejsme v nich schopni ukládat číselnou hodnotu skóre a ani s ní následně vyhodnocovat splněnost vnitřních uzlů. Budeme tedy generalizovat myšlenku AND/OR grafů, ovšem omezíme se teď již pouze na stromy.

Definice 9: *MINIMAX strom* je strom, kde každý vrchol je buď typu MIN, nebo MAX. Každý list je ohodnocen rozšířeným reálným číslem – tj. reálným číslem či kladným/záporným nekonečnem.
Obecně lze místo reálných čísel použít jakoukoliv uspořádanou množinu.

MINIMAX strom se vyhodnocuje směrem od listů, dokud nespočteme hodnotu kořene. Hodnota vrcholu typu MAX je rovna maximální hodnotě jeho následníků, hodnota uzlu typu MIN pak odpovídá hodnotě nejmenší napříč všemi následníky.

Definice 10: *Algoritmus MINIMAX* je rekurzívní procedura, která ohodnotí každý vrchol MINIMAX stromu.

- Hodnota listů je dána jako součást definice MINIMAX stromu.
- Hodnota vrcholu typu MIN je nejmenší hodnota mezi jeho následníky.
- Hodnota vrcholu typu MAX je největší hodnota mezi jeho následníky.

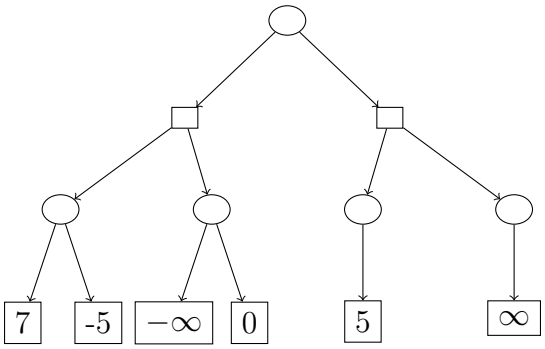
Příklad. Dokažte odvážné tvrzení, že MINIMAX stromy rozšiřují AND/OR stromy. Jinými slovy ukažte, že každý AND/OR strom lze chápat jako MINIMAX strom.

Každý koncový vrchol ohodnotíme 1, je-li splněný, v opačném případě 0. Jelikož i listy MINIMAX stromu musí mít nějaký typ, označíme je všechny například za MIN. Vnitřní uzly typu AND pak odpovídají vrcholu typu MIN, jelikož to, že všichni následníci musí být splněni, je v tomto případě ekvivalentní požadavku, aby minimum hodnot následníků bylo 1. Obdobně OR uzly pak odpovídají typu MAX – má-li být splněný alespoň jeden následník, musí být maximum z jejich hodnot rovno 1.

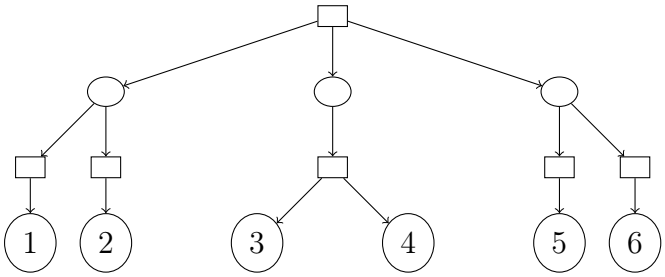
Vypůjčíme si tedy notaci AND/OR grafů a uzly typu MAX budeme značit obdélníkem ▭ zatímco MIN vrcholy elipsou ⬭. Jiný častý způsob značení používá pro vrcholy MAX symbol $\bigwedge$, pro MIN uzly $\bigvee$. Jelikož MINIMAX stromy použijeme pouze pro modelování tahové hry dvou hráčů, budeme se zabývat především těmi stromy, kde se uzly MIN a MAX střídají podle své vzdálenosti od kořene. Dále budeme uvažovat pouze hry s nulovým součtem, což znamená, že obdržím-li v nějaké hře skóre s, můj soupeř musí získat $-s$ bodů – každý list nám tak bude stačit ohodnotit jen podle našeho skóre a soupeřovo si snadno domyslíme. Jak se později ukáže, tato myšlenka je velmi důležitá: MINIMAX stromy budou fungovat jen pro hry s nulovým součtem či obecně hry, kde platí, že když maximalizuji svoje skóre, zároveň minimalizuji to soupeřovo – a naopak.

Příklad. Určete hodnotu kořene zadaných MINIMAX stromů.

a)

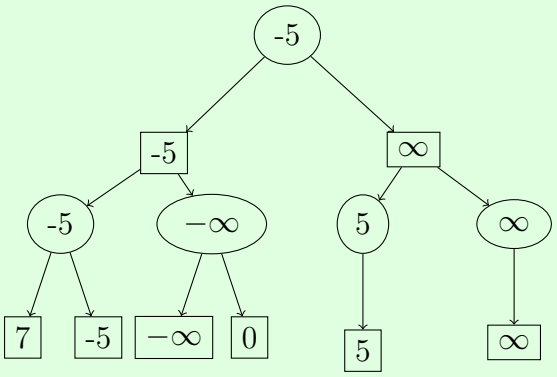


b)

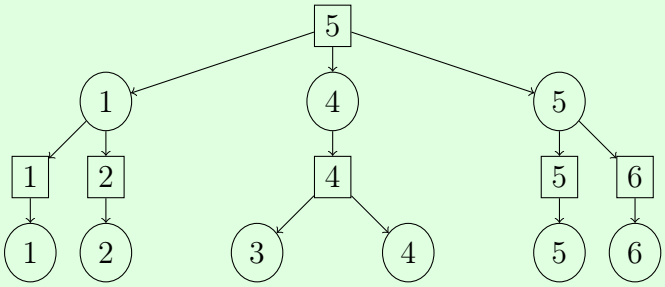


Vyhodnocujeme směrem od listů, uzel po uzlu.

a)

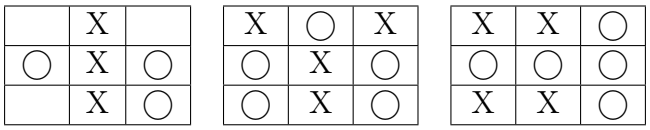


b)



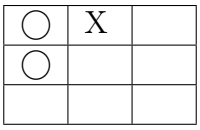
Nyní nám zbývá se spolu podívat, jakým způsobem můžeme využít tohoto modelu k popisu her. A jako obvykle si při ilustraci pomůžeme piškvorkami na hracím poli 3×3.

Každý list v našem stromu stavů (tedy herních konfigurací) ohodnotíme 0, 1 či -1 podle toho, zda se jedná o remízu, naši výhru či prohru. V této hře budeme první na tahu a kreslíme kolečka. Prohlédněme si tři vybrané listy a jejich přiřazenou hodnotu.

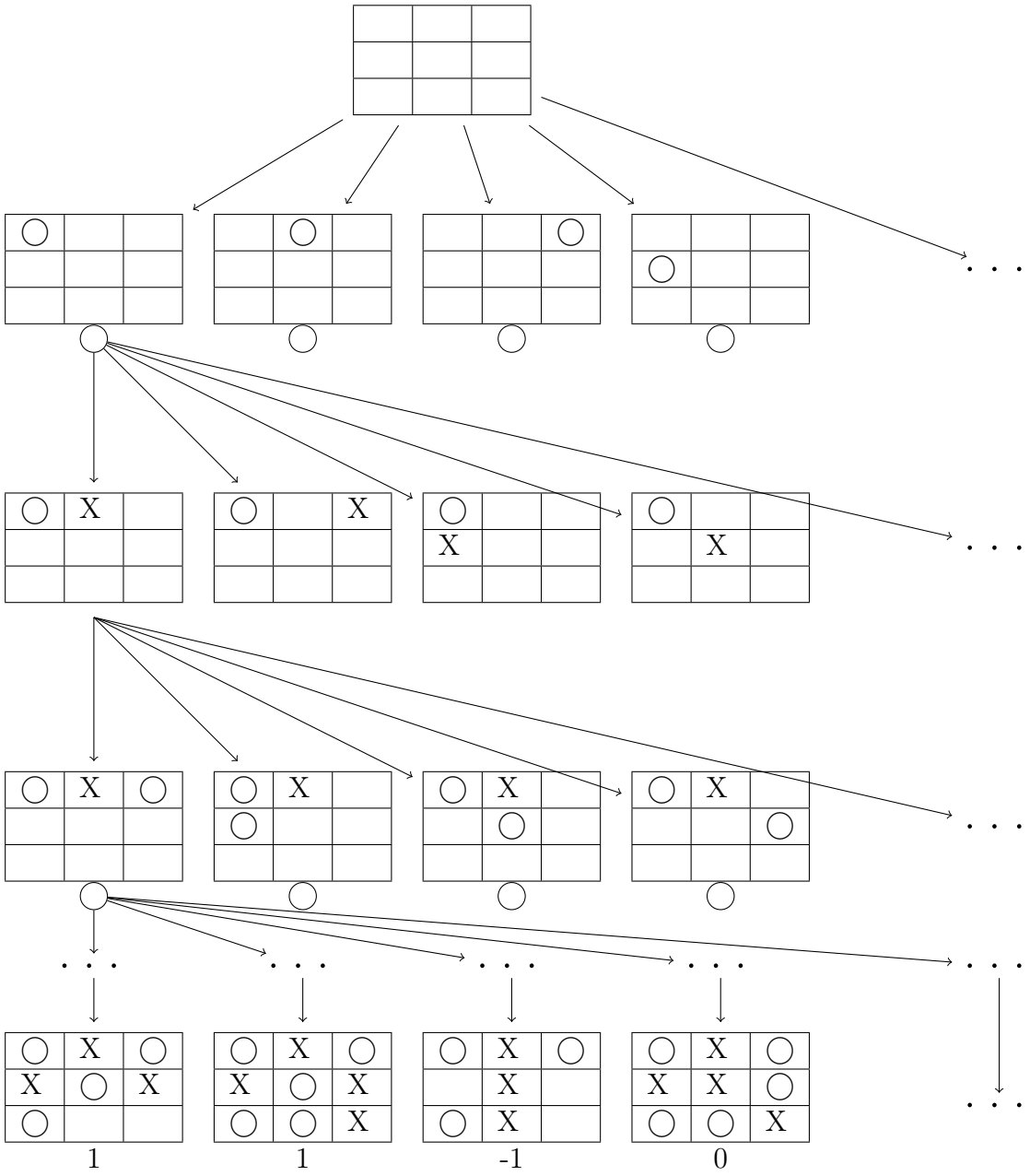


Podle našeho ohodnocení je jasné, že kdykoliv budu na tahu já, budu se snažit svoje skóre *maximalizovat*, kdežto protihráč vždy *minimalizovat*. Uzly grafu, kde jsem na řadě já tedy budou typu MAX a ostatní typu MIN.

Uvědomme si, že ne všechny listy jsou ve stejné hloubce (hra může skončit po různém počtu kol). Některé vrcholy se mohou vyskytovat duplicitně, neboť do nich lze „dojít“ různými cestami. Například následující konfigurace lze dosáhnout tak, že levý horní roh zaplníme kolečkem v prvním či až druhém kole.



Připomínáme, že vrcholy typu MIN budeme značit malým kolečkem dole, ostatní vrcholy jsou pak typu MAX. Následuje ukázka malé části celého grafu.



Příklad 4.1.1. Rozhodněte, zda je následující hry možné modelovat MINIMAX stromy. Pokud ano, zamyslete se nad tím, jak by takové stromy vypadaly.

a) šachy

b) sudoku

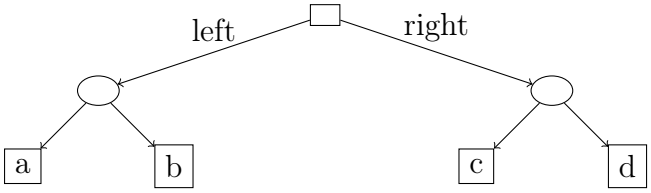
c) piškvorky na neomezené hrací ploše

d) kámen, nůžky, papír

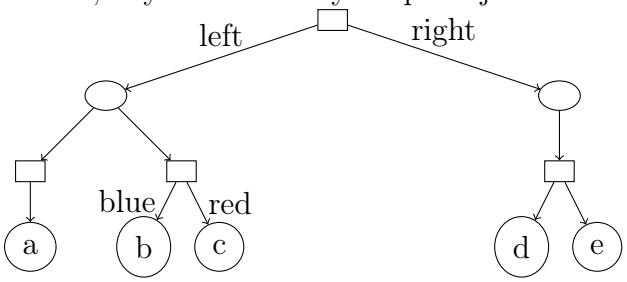
e) tenis

Příklad 4.1.2. Doplňte nějaké konkrétní ohodnocení konečných stavů her s naznačeným MINIMAX stromem tak, aby začínající hráč dosáhl nejlepšího možného výsledku hraním zadaných strategií, předpokládáme-li, že jeho soupeř nedělá chyby? Jaké vztahy obecně musí v takovém případě platit pro hodnoty listů?

a) Výherní strategie nechť je *left*.

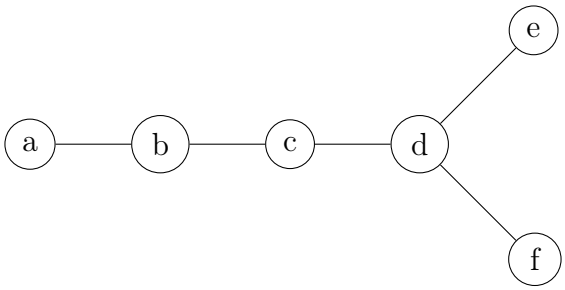


b) Výherní strategie nechť je volit *left* v prvním kole a *blue* ve druhém. Nezapomeňte zařídit, aby nás dokonalý soupeř s jistotou dovedl k volbě potřebné strategie.



Příklad 4.1.3. Ukažte, že obecný MINIMAX strom lze efektivně transformovat na ekvivalentní MINIMAX strom, kde se střídají typy vrcholů – tzn. následník má vždy jiný typ než jeho rodič. Efektivní transformací rozumíme takový algoritmus, která je polynomiální vzhledem k počtu vrcholů vstupního stromu. Ekvivalentní MINIMAX strom musí mít stejné listy se stejnými hodnotami jako strom původní a algoritmus MINIMAX musí napočítat stejnou hodnotu v kořeni.

Příklad 4.1.4. Uvažme hru dvou hráčů – pronásledovatele P a zloděje Z – na hracím poli zadaném náledujícím grafem.



Pronásledovatel je první na tahu a začíná na poli b, zloděj začíná na poli d. Každý se během svého tahu posouvá na vedlejší pole, přičemž se po jednom kroku střídají. Hra končí, jestliže se zároveň ocitnou na stejném poli. Zloděj se snaží přežít co nejdéle a hodnota koncové konfigurace je pro něj úměrná tomu, kolik kol odehrál.

- Lze tuto hru modelovat MINIMAX stromem? Lze ji vyřešit?
- Vykreslete a vyřešte strom pro hru omezenou 3 koly – tedy hráč P i Z se každý posunou nejvýše třikrát.

4.2 Alfa-beta prořezávání

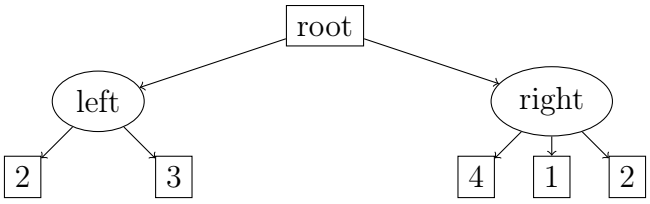
Řešení deterministické hry s perfektní znalostí můžeme dostat i efektivnější procedurou, než je MINIMAX algoritmus. Zkusme ji spolu objevit.
Až do této chvíle jsme pro zjednodušení tvrdili, že algoritmus MINIMAX vyhodnocuje vrcholy směrem od listů bez toho, abychom podrobněji vysvětlili, proč tomu tak je a jak to detailně funguje. Podívejme se detailně na skoro-pythonovský (pseudo)kód této procedury.

```
def minimax(node):
    if node.neighbors is empty:  # nachazime se v listu
        return node.value
    best_so_far = minimax(node.neighbors[0])
    # vynech nulteho naslednika (souseda), protoze jsme ho jiz zpracovali
    for neighbor in node.neighbors[1:]:
        value = minimax(neighbor)
        if node.type == "MIN" and value < best_so_far:
            best_so_far = value
        if node.type == "MAX" and value > best_so_far:
            best_so_far = value
    return best_so_far
```

Příklad. Jaká je asymptotická časová složitost algoritmu MINIMAX?

Algoritmus probíhá zcela stejně jako prohledávání do hloubky (DFS), v každém uzlu se provedou navíc jen nějaká porovnání s konstantní časovou složitostí. Výsledná asymptotická časová složitost tedy je $\mathcal{O}(V+E)$, kde V je počet vrcholů prohledávaného grafu a E je počet hran. Toto lze vidět i z toho, že každý uzel a každá hrana se v průběhu výpočtu navštíví právě jedenkrát.
V našem případě uvažujeme pouze souvislé grafy, můžeme tedy výsledek napsat stručněji jako $\mathcal{O}(E)$.

Spustíme-li tedy výpočet nad kořenem našeho MINIMAX stromu (tj. zavoláme `minimax(root)`), dostaneme se do for cyklu, kde se rekurzivně zavoláme nad prvním následníkem kořene atp. až se dostaneme k listu. Jedná se tedy o prohledávání do hloubky s několika přidanými výpočty.
Podívejme se teď spolu na jednoduchý příklad.



Budeme předpokládat, že uspořádání seznamu následníků odpovídá tomu na nákresu při čtení zleva doprava. Při vykonávání našeho (pseudo)kódu při spuštění `minimax(root)` se tedy nejprve dostaneme do uzlu *left* a skrze něj okamžitě do listu s hodnotou 2. Dosavadní nejlepší (a tedy v tomto případě nejmenší) hodnota pro vrchol *left* je tedy 2, ale může se pochopitelně změnit při procházení dalšího následníku, ke kterému dochází hned vzápětí. V tomto případě se však nic nezmění a konečně zjišťujeme, že hodnota uzlu *left* je 2.

Kořen tedy skončil zpracování prvního následníka a nejlepší hodnota, kterou zatím viděl, je 2. Protože je typu MAX, víme, že nižší hodnotu už mít nemůže, ale jeho další následník (*right*) by ji mohl ještě zvýšit.

Při vyhodnocování uzlu *right* nejprve vidíme hodnotu 4, poté 1. Zde se na chvíli zastavíme (těsně předtím než bychom zpracovali poslední list). Víme, že konečná hodnota vrcholu *right* nebude vyšší než 1, protože minimalizuje. Na druhou stranu jsme si již před začátkem zpracování tohoto uzlu všimli, že maximalizující kořen *root* nebude mít nižší hodnota než 2, kterou mu nabízí jeho levý následník. Prohledávání tedy můžeme okamžitě ukončit, jakýkoliv další následník uzlu *right* je nezajímavý a nemůže změnit hodnotu kořene, ačkoliv by pochopitelně ještě mohl snížit hodnotu svého předchůdce.

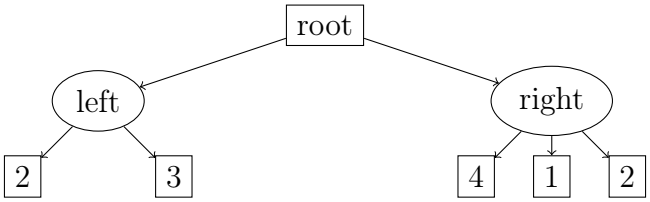
Ačkoliv jsme si v tuto chvíli ušetřili průchod jediným uzlem, představíme-li si místo listů našeho vzorového příkladu dostatečně velké podstromy, můžeme si hodně pomoci. Tato úspora se neprojeví na nejhorší časové složitosti algoritmu ALFA-BETA, zlepší se však v průměrném případě nad omezenou vstupní množinou MINIMAX stromů. Hodnotu kořenového uzlu vsak napočítáme stejnou (což obecně neplatí o ostatních vnitřních uzlech), tedy ALFA-BETA je korektní.

Nyní se již spolu podíváme na implementaci algoritmu ALFA-BETA. Pro přehlednost zcela oddělujeme rutinu pro uzly typu MIN a MAX, z důvodu stručnosti používáme v našem pseudokódu nekonečno `INFINITY` v inicializaci. Iniciální volání spustíme nad kořenem stromu obdobně jako předtím, tedy příkazem `alphabeta(root, -INFINITY, INFINITY)`.

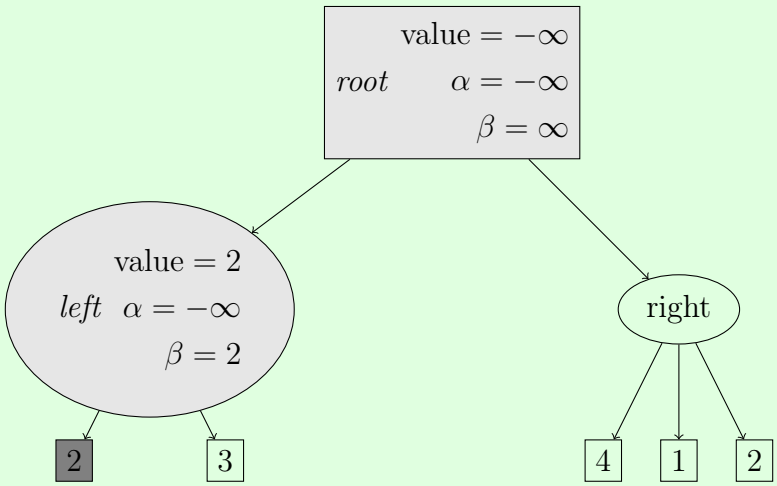
```
def alphabeta(node, alpha, beta):
    if node.neighbors is empty:  # nachazime se v listu
        return node.value
    if node.type == "MAX":
        value = -INFINITY  # aby se prepsala pri objevu prvni hodnoty
        for neighbor in node.neighbors:
            value = max(value, alphabeta(neighbor, alpha, beta))
            if value >= beta:
```

```
                break
            alpha = max(alpha, value)
        return value
    else:
        value = INFINITY
        for neighbor in node.neighbors:
            value = min(value, alphabeta(neighbor, alpha, beta))
            if value <= alpha:
                break
            beta = min(beta, value)
        return value
```

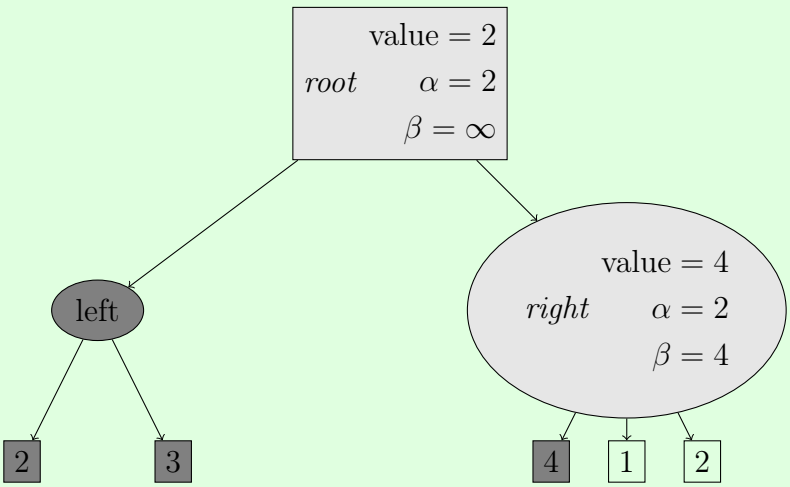
Příklad. Odkrokujte si algoritmus nad motivačním příkladem z úvodu této podsekce a přesvědčte se, že skutečně dojde k ořezání výpočtu tak, jak jsme předpověděli.



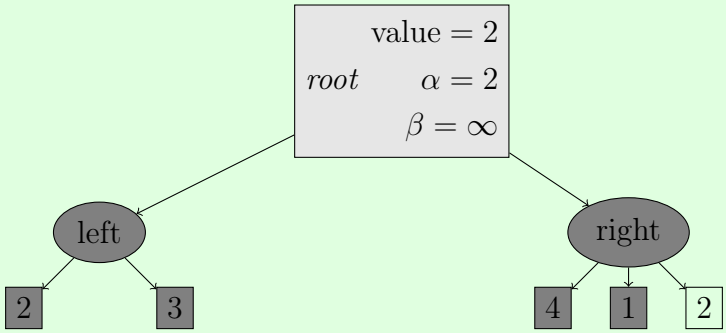
Výpočet začíná voláním `alphabeta(root, -∞, ∞)`, počáteční hodnoty jsou tedy $\alpha = -\infty, \beta = \infty$ a začínáme zpracovat kořenový vrchol. Při jeho výpočtu je potřeba najít hodnotu uzlu *left*, který se tedy bude muset vyčíslit (se stejnými hodnotami alfy a bety). Nejprve se objeví list s hodnotou 2, po jeho zpracování ukazuje hodnoty proměnných následující ilustrace. Světle šedý uzel je rozpracovaný, tmavě šedý je již úplně zpracovaný.



Následně se obdobně zpracuje list s hodnotou 3, čímž dokončíme výpočet hodnoty vrcholu *left* a vrátíme se s výpočtem zpět do kořene *root*. Ten aktualizuje svoje hodnoty a spustí výpočet nad svým druhým následníkem *right*, přičemž se dostaneme až k uzlu s hodnotou 3.



Jako další se zpracovává list s hodnotou 1, kdy se konečně projeví smysl zavedených proměnných α a β. Jelikož se nám hodnota *value* nastaví na 1, což je méně než aktuální hodnota proměnné α ve vrcholu *right*, výpočet v tomto uzlu se ukončí (a vrátí hodnotu 1 do kořene *root*).



Výpočet v tuto chvíli končí s výslednou hodnotou 2, přičemž poslední list jsme během výpočtu vůbec neviděli.

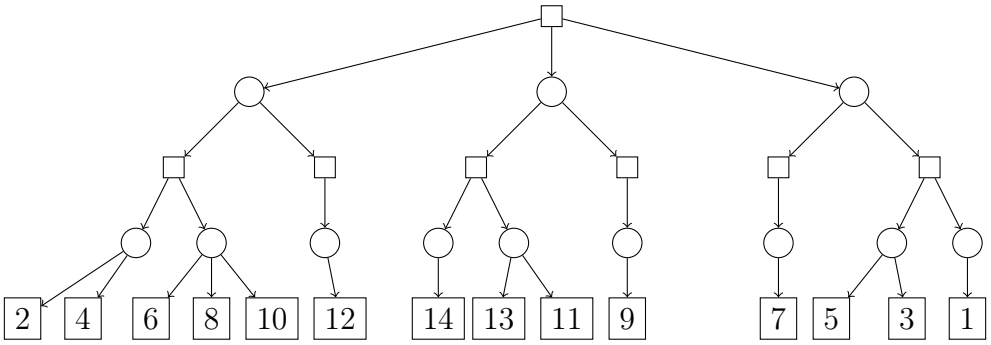
Příklad 4.2.1. Ukažte, že asymptotická časová složitost v nejhorším případě algoritmu ALFA-BETA nad MINIMAX stromy je stejná jako pro algoritmus MINIMAX.

Příklad 4.2.2.
Zadaný strom MINIMAX střídá po úrovních typy svých uzlů a každý jeho list je ve stejné hloubce. Takové v literatuře (a hlavně v praxi) budete vídat nejčastěji, neboť přirozeně vznikají při modelování mnoha zajímavých her. Jako v celém zbytku sbírky předpokládejte i zde, že algoritmus prochází vrcholy v pořadí zleva doprava.

a) Vyřešte strom s pomocí algoritmu ALFA-BETA. Především tedy zjistěte výslednou hodnotu kořene a které podstromy budou uřezány, tj. nebudou vůbec navštíveny. Při výpočtu věnujte zvláštní pozornost tomu, abyste porozuměli, jakým způsobem α odpovídá dolní hranici a β naopak horní.

b) Při zachování struktury grafu navrhněte vhodné hodnoty listů tak, aby nedošlo k žádnému prořezávání (a algoritmus tedy navštívil všechny listy).

c) Při zachování struktury grafu navrhněte vhodné hodnoty listů tak, aby došlo k největšímu možnému prořezávání (a algoritmus tedy navštívil co nejméně uzlů).



Příklad 4.2.3. Při kterých z následujících transformací MINIMAX stromu může dojít ke změně nalezené optimální strategie?

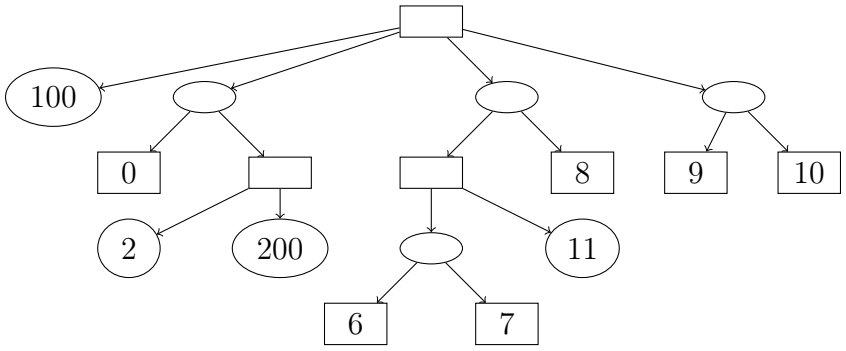
a) Ke všem hodnotám listů přičteme stejnou reálnou konstantu c.

b) Všechny hodnoty v listech vynásobíme stejnou konstantou c.

c) Hodnoty ve všech listech se libovolně změní tak, aby mezi nimi zůstalo zachováno jejich původní uspořádání.

d) Všechny uzly typu MIN změníme na MAX a naopak. Hodnoty v listech pronásobíme -1.

Příklad 4.2.4. Bez výpočtu určete a zdůvodněte, které podstromy by algoritmus ALFA-BETA v zadaném stromu ořezal. Následníci se prochází v pořadí zadaném obrázkem.

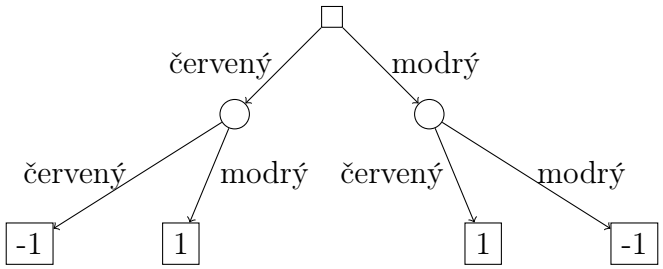


Příklad 4.2.5. Představme si MINIMAX strom, který se skládá pouze z uzlů typu MAX. Může při jeho vyhodnocování ALFA-BETA algoritmem dojít k nějakému prořezání?

4.3 Nedeterministické hry

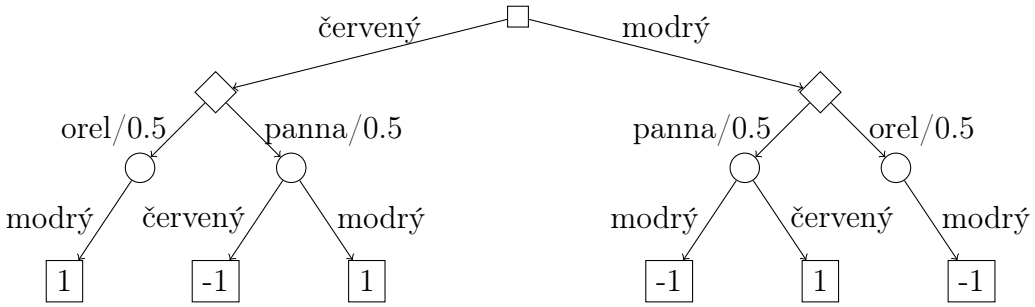
Až do této chvíle jsme tvrdě trvali na tom, aby uvažované hry neobsahovaly náhodu, neboť bychom ji neuměli modelovat. Nyní si ukážeme, že lze vhodně rozšířit MINIMAX stromy tak, aby náhodu v jisté míře postihly a my nad nimi pak mohli provádět smysluplné výpočty a předpovědi. Zjistíme však také, že některé vlastnosti a zákony, které dosud platily, se s tímto rozšířením zcela rozbíjí a mění.

Na začátek uvažme velmi triviální ilustrativní hru. V prvním kole si vezmu modrý, či červený žeton. V dalším kole provede to samé provede soupeř (může si zvolit i žeton stejné barvy). První hráč vítězí, drží-li jiný žeton než jeho soupeř, v opačném případě prohrává. Pro přehlednost přikládáme ještě příslušný MINIMAX strom této jednoduché hry.



Nyní naši hru obohatíme o náhodu. Před tím, než si druhý hráč zvolí svůj žeton, hodíme mincí. Padne-li orel, soupeř přijde o možnost volby – musí si vzít modrý žeton. V opačném případě se nic nemění a hra pokračuje jako předtím.

Pro modelování nedeterministické hry použijeme speciální vrchol (značený kosočtvercem $\diamond$) tam, kde dochází k rozhodnutí na základě náhody. Hrany, které z něj vychází, budou vždy označeny pravděpodobností, se kterou k výběru té konkrétní cesty dojde.



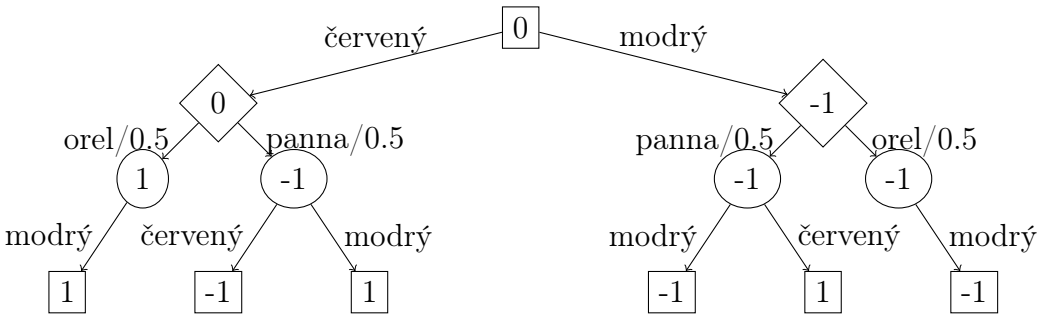
Už tedy chápeme, jak nedeterministické hry modelovat, zajímá nás však, jak se dopočítáme ke zjevnému závěru, že má začínající hráč volit červený žeton. K tomu využijeme upravený algoritmus MINIMAX, který rozšíříme o schopnost počítat hodnotu nového typu uzlu reprezentujícího náhodnou volbu $\diamond$.

Pro většinu aplikací je přirozené volit jako hodnotu vrcholu náhody jeho očekávanou hodnotu – tj. hodnotu každého potomka vynásobíme pravděpodobností, že bude vybrán, a výsledky pak sečteme napříč všemi potomky. Poněkud přehlednější zápis říká

$$\text{minimax(X)} = \sum_{\text{n...child of X}} \text{minimax(n)} \cdot \text{P(n)},$$

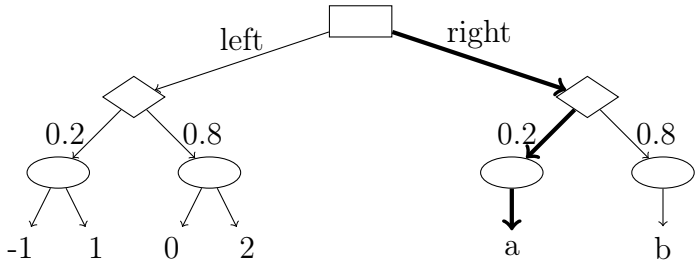
kde X je uzel reprezentující náhodu, $minimax(n)$ je spočtená hodnota potomka n a $P(n)$ je pravděpodobnost, že náhodně zvolíme právě potomka n – tj. číslo na odpovídající hraně grafu.

S touto znalostí již celý graf snadno doplníme. Je vidět, že očekávaný výnos začínajícího hráče je 0 a získá ho za předpokladu, že volí strategii *červený*.



Příklad 4.3.1. Sestavte vhodný strom a následně jej vyřešte pro následovně zadanou nedeterministickou hru dvou hráčů.
Začínající hráč si zvolí číslo 1, 2 nebo 3. Následně se hodí 3 mincemi. Padne-li na všech z nich stejná strana, soupeř si smí zvolit číslo -1 nebo 0. V opačném případě vybere číslo 1 či 2. Začínající hráč obdrží od soupeře částku rovnou součinu jejich čísel – v případě záporného součinu naopak musí příslušnou sumu protihráči zaplatit.

Příklad 4.3.2. Uvažme obrázkem zadaný MINIMAX strom rozšířený o vrcholy reprezentující náhodu. Hodnoty dvou listů jsou neznámé, označeny jako a a b.

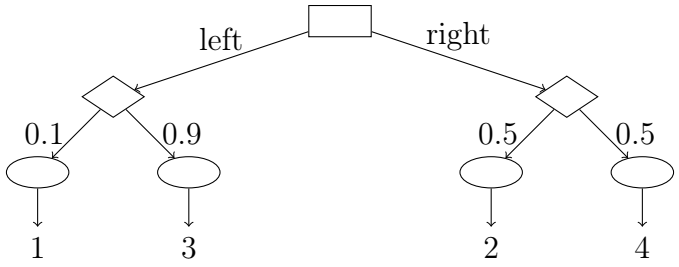


a) Co musí platit pro hodnoty a, b, aby optimální strategie odpovídala tučně vyznačené?

b) Nechť $b = 1$. Co musí platit pro hodnotu a, aby maximalizující hráč volil v prvním kole strategii *left*?

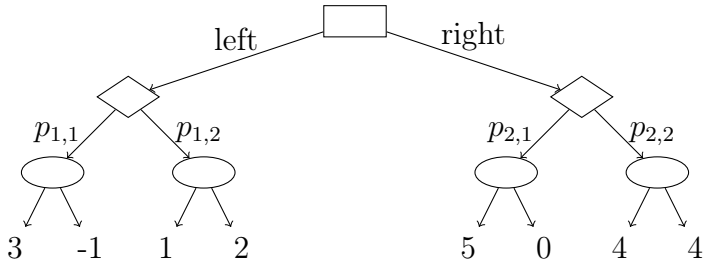
c) Nechť $b = -1$. Co musí platit pro hodnotu a, aby maximalizující hráč volil v prvním kole strategii *right*?

Příklad 4.3.3.



Nejprve pro obrázkem zadaný rozšířený MINIMAX strom ověřte, že optimální strategie začínajícího hráče je *right*.
Poté ukažte, že lze vhodně transformovat hodnoty listů tak, aby se optimální strategie změnila na *left*. Transformace musí zachovat původní uspořádání mezi listy; tedy jestliže měl list A před transformací menší hodnota než list B, bude tomu tak i po ní.

Příklad 4.3.4.
Uvažujeme rozšířený MINIMAX strom s neznámými pravděpodobnostmi větvení.



a) Nechť $p_{1,1} = \frac{2}{7}$. Co platí pro $p_{2,1}, p_{2,2}$, aby optimální strategie začínajícího hráče byla *right*?

b) Co pro neznámé pravděpodobnosti platí obecně, má-li být optimální strategie *right*?

c) Jak by musely vypadat hodnoty v listech, aby bez ohledu na hodnoty neznámých pravděpodobností byla optimální strategií *right*?

5 Výroková logika

Výroková logika je základní formalismus, kterým můžeme modelovat jednoduchá tvrzení, jejich pravdivost, vyplývání a jiné vztahy mezi nimi. Staví na pojmu *výrok*, což je elementární tvrzení, o jehož pravdivosti má smysl uvažovat. Výroky reprezentujeme výrokovými proměnnými, které s využitím logických spojek (a, nebo, pokud-pak atp.) můžeme skládat do složitějších formulí a tím modelovat složitější tvrzení (jako např. souvětí).

5.1 Syntax

Syntax vyjadřuje, jakým způbem ve výrokové logice zapisujeme platné formule – zatím však bez toho, abychom jim přiřazovali nějaký význam. První definice popisuje symboly, které používáme...

Definice 11: *Abeceda výrokové logiky* zahrnuje

- spočetně mnoho symbolů výrokových proměnných $p, q, r, \ldots,$
- symboly pro logické spojky $\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \ldots,$
- pomocné symboly závorek (a).

Ozn. $\mathcal{P} = \{p, q, r, \ldots\}$ množinu symbolů výrokových proměnných.

... další už nám přesně říká, jak vypadá správně utvořená formule ve výrokové logice.

Definice 12: *Formule výrokové logiky.*

- Každý symbol $p \in \mathcal{P}$ je formule.
- Jsou-li φ, ψ fomule, pak rovněž $\neg(\varphi)$, $(\varphi) \vee (\psi)$, $(\varphi) \wedge (\psi)$, $(\varphi) \Rightarrow (\psi)$, $(\varphi) \Leftrightarrow (\psi)$, ... jsou formule.
- Nic jiného formulí není.

Závorková konvence umožňuje vynechávat přebytečné závorky, nedojde-li k porušení sémantické jednoznačnosti formule.
Ozn. $\mathcal{F}$ množinu formulí výrokové logiky.

Poznáte nyní správně utvořenou formuli výrokové logiky?

Příklad 5.1.1. Uvažujte formule dle Definice 12. Rozhodněte, která z následujících slov jsou formule (neuvažujte závorkovou konvenci, tj. závorky musí přesně odpovídat definici):

a) $w_1 = \neg(\neg p)$
b) $w_2 = (p) \wedge (q) \wedge (r)$
c) $w_3 = (p) \Rightarrow ((\neg(q)) \vee (\neg(r)))$
d) $w_4 = ((p)\neg(q)) \vee (r)$
e) $w_5 = (p) = (r)$

5.2 Sémantika

Sémantika popisuje, jak máme logické formule chápat, přiřazuje celému formalismu konkrétní význam. První důležitý pojem, interpretace, vyjadřuje, zda konkrétní elementární výroky považujeme za pravdivé či nepravdivé.

Definice 13: *Interpretace* I je zobrazení $I : \mathcal{P} \to \{0, 1\}$ přiřazující pravdivostní hodnoty 0 (nepravda), 1 (pravda) jednotlivým výrokovým proměnným množiny $\mathcal{P}$.

V konkrétní interpretaci elementárních výroků pak můžeme vyhodnotit pravdivostní hodnotu celé formule.

Definice 14: *Valuace* (též *vyhodnocení*) příslušící interpretaci I je zobrazení $I : \mathcal{F} \to \{0, 1\}$ přiřazující pravdivostní hodnoty 0 (nepravda), 1 (pravda) jednotlivým formulím z $\mathcal{F}$.
(Jedná se o rozšíření interpretace z atomických formulí na všechny formule podle sémantiky logických spojek. Přesná definice je induktivní a zde ji neuvádíme.)

To s sebou přináší celou řadu pojmů, které blíže popisují vlastnosti dané formule, ať už obecně, či ve vztahu k dalším formulím.

Definice 15:

- Formule φ je *pravdivá v interpretaci* I, jestliže $I(\varphi) = 1$ (tj. vyhodnotí se v příslušné valuaci na hodnotu 1). V opačném případě je formule *nepravdivá v interpretaci* I.
- Formule φ je *logicky pravdivá* či *tautologie*, jestliže je pravdivá v libovolné interpretaci.
- Formule φ je *kontradikce*, jestliže je nepravdivá v libovolné interpretaci.
- Formule φ je *splnitelná*, jestliže je pravdivá v nějaké interpretaci. Tato splňující interpretace se nazývá *modelem* formule φ.
- Formule φ, ψ jsou *sémanticky ekvivalentní*, značeno $\varphi \approx \psi$, jestliže $I(\varphi) = I(\psi)$ pro libovolnou interpretaci I.

Podívejme se na vše zblízka na vzorovém příkladě.

Příklad. Mějme formuli výrokové logiky $\varphi \equiv p \wedge (q \Rightarrow \neg p)$. Sestavte pravdivostní tabulku formule φ a určete

a) zda je φ pravdivá v interpretaci $I(p) = 0$, $I(q) = 1$,

b) zda je φ kontradikcí či tautologií,

c) zda je φ splnitelná; případně nalezněte nějaký její model.

Pravdivostní tabulka formule vypadá následujícím způsobem.

ř.	p	q	p	$\wedge$	$(q$	$\Rightarrow$	$\neg p)$
1	0	0	0	**0**	0	1	1
2	0	1	0	**0**	1	1	1
3	1	0	1	**1**	0	1	0
4	1	1	1	**0**	1	0	0

Jednotlivé řádky odpovídají různým interpretacím (sloupečky p, q) a jim příslušícím valuacím jednotlivých částí (podformulí) formule φ (následující sloupečky). Sloupeček s tučně vyznačenými hodnotami odpovídá valuacím formule φ.

a) Formule φ je nepravdivá v interpretaci I, viz ř. 2.

b) Formule φ není ani kontradikcí, ani tautologií. (Všechny zvýrazněné hodnoty by musely být 0, resp. 1.)

c) Formule φ je splnitelná, neboť existuje interpretace, v níž je formule pravdivá. Viz ř. 3 tabulky. Modelem formule φ (jediným) je tedy interpretace $I(p) = 1$, $I(q) = 0$.

Příklad 5.2.1. Mějme formuli výrokové logiky $\varphi \equiv p \Rightarrow \neg(\neg q \vee r)$. Sestavte pravdivostní tabulku formule φ a určete

a) zda je φ pravdivá v interpretaci $I(p) = 0$, $I(q) = I(r) = 1$,

b) zda je φ kontradikcí či tautologií,

c) zda je φ splnitelná; případně nalezněte nějaký její model.

Příklad 5.2.2. Mějme formuli výrokové logiky $\varphi \equiv (p \wedge q) \Leftrightarrow (\neg q \wedge r)$. *Bez použití pravdivostní tabulky* určete

a) zda je φ pravdivá v interpretaci $I(p) = 0$, $I(q) = I(r) = 1$,

b) zda je φ kontradikcí,

c) zda je φ tautologií,

d) zda je φ splnitelná; případně nalezněte nějaký její model.

Příklad 5.2.3. Udejte příklad formule φ takové, že:

a) φ obsahuje právě 3 různé výrokové proměnné a je pravdivá právě ve 3 interpretacích,

b) φ obsahuje právě 3 různé výrokové proměnné, je pravdivá právě ve 3 interpretacích a obsahuje pouze logickou spojku $\Rightarrow$.

c) φ je kontradikce, obsahuje právě 2 různé výrokové proměnné, každou dvakrát.

(Uvažujte interpretaci jako zobrazení přiřazující hodnoty právě výrokovým proměnným vyskytujícím se v φ.)

Příklad 5.2.4. Zjistěte, kolik existuje vzájmeně neekvivalentních formulí výrokové logiky

a) obsahujících pouze 3 výrokové proměnné p_1, p_2, p_3 (i vícekrát),

b) obsahujících pouze n výrokových proměnných $p_1, \ldots, p_n$ (i vícekrát),

c) obsahujících pouze n výrokových proměnných $p_1, \ldots, p_n$ (i vícekrát) pravdivých právě v polovině interpretací.

5.3 Množiny formulí, splnitelnost, vyplývání

Občas je potřeba pracovat s několika tvrzeními zároveň a zkoumat vztahy mezi nimi. Proto zavádíme následující pojmy.

Definice 16: Množina formulí $\mathcal{T}$ je *splnitelná*, jestliže existuje interpretace I, v níž jsou pravdivé všechny formule $\varphi \in \mathcal{T}$. Potom říkáme, že množina $\mathcal{T}$ je *pravdivá v interpretaci* I a tato interpretace se nazývá *modelem* množiny $\mathcal{T}$. Není-li množina splnitelná, mluvíme o množině *nesplnitelné*.

Všimněte si, že prázdná množina $\varnothing$ je splnitelná a je pravdivá v libovolné interpretaci. Dále zavedeme pojem logického vyplývání.

Definice 17: Formule φ *logicky vyplývá* z množiny formulí $\mathcal{T}$, zapisováno $\mathcal{T} \vDash \varphi$, právě když je formule φ pravdivá v každém modelu množiny $\mathcal{T}$.

Jinak řečeno, formule (závěr) logicky vyplývá z množiny předpokladů, právě když je pravdivá v každé interpretaci, v níž je pravdivá i množina předpokladů.

Vyplývá-li tedy formule φ z prázdné množiny, $\varnothing \vDash \varphi$, jedná se o tautologii (rozmyslete si). Píšeme jenom $\vDash \varphi$. V podobném duchu vynecháváme množinové závorky, je-li množina předpokladů jednoprvková, tedy $\psi \vDash \varphi$ namísto $\{\psi\} \vDash \varphi$.

Následující věta dále propojuje logickou spojku implikace s logickým vyplýváním.

Věta 18 (o dedukci): Logické vyplývání $\{\psi_1, \ldots, \psi_n\} \vDash \varphi$ platí, právě když platí vyplývání $\{\psi_i, \ldots, \psi_{n-1}\} \vDash \psi_n \Rightarrow \varphi$.

Následující vzorový příklad ilustruje právě zavedené koncepty.

Příklad. Uvažujte množinu formulí $\mathcal{T} = \{\neg p \Rightarrow r, \neg(q \wedge r), p \vee \neg q\}$. Sestavte pravdivostní tabulku množiny $\mathcal{T}$ a určete

a) zda je $\mathcal{T}$ pravdivá v interpretaci $I(p) = I(q) = I(r) = 0$,

b) zda je $\mathcal{T}$ splnitelná; případně nalezněte nějaký její model,

c) zda platí logické vyplývání $\mathcal{T} \vDash p \wedge \neg r$,

d) zda platí logické vyplývání $\mathcal{T} \vDash r \Rightarrow \neg q$.

Pravdivostní tabulka množiny $\mathcal{T}$ vypadá následujícím způsobem.

ř.	p	q	r	$\neg p \Rightarrow r$	$\neg(q \wedge r)$	$p \vee \neg q$	$\mathcal{T}$	$p \wedge \neg r$	$r \Rightarrow \neg q$
1	0	0	0	0	1	1	0	0	1
2	0	0	1	1	1	1	1	0	1
3	0	1	0	0	1	0	0	0	1
4	0	1	1	1	0	0	0	0	0
5	1	0	0	1	1	1	1	1	1
6	1	0	1	1	1	1	1	0	1
7	1	1	0	1	1	1	1	1	1
8	1	1	1	1	0	1	0	0	0

a) Množina $\mathcal{T}$ není pravdivá v interpretaci I, viz ř. 1.

b) Množina $\mathcal{T}$ je splnitelná, neboť existuje interpretace, v níž je množina pravdivá. Viz např. ř. 2 tabulky. Modelem množiny $\mathcal{T}$ je tedy např. interpretace $I(p) = I(q) = 0$, $I(r) = 1$.

c) Vyplývání neplatí. Existuje interpretace, v níž je pravdivá množina $\mathcal{T}$ (předpoklad), ale neplatí závěr. Viz např. ř. 2.

d) Vyplývání platí. Ve všech interpretacích, v nichž je pravdivá množina $\mathcal{T}$ (předpoklad), je platný i závěr. Viz ř. 2, 5, 6, 7.

Příklad 5.3.1. Uvažujte množinu formulí $\mathcal{T} = \{r \wedge \neg q, \neg(\neg r \wedge p), p \Rightarrow r\}$. Sestavte pravdivostní tabulku množiny $\mathcal{T}$ a určete

a) zda je $\mathcal{T}$ pravdivá v interpretaci $I(p) = I(q) = I(r) = 1$,

b) zda je $\mathcal{T}$ splnitelná; případně nalezněte nějaký její model,

c) zda platí logické vyplývání $\mathcal{T} \models r$,

d) zda platí logické vyplývání $\mathcal{T} \models r \wedge p$.

Příklad 5.3.2. Ukažte, že pro každou množinu formulí $\mathcal{T}$ existuje formule φ, která je pravdivá v interpretaci I, právě když je I modelem $\mathcal{T}$.

Příklad 5.3.3. Uvažujte množinu formulí $\mathcal{T} = \{(p \Rightarrow q) \vee \neg r,\ \neg p \Rightarrow s,\ \neg t \vee p \vee \neg q\}$. Zadejte formuli φ tak, aby množina $\mathcal{T} \cup \{\varphi\}$ byla nesplnitelná.

Příklad 5.3.4. Rozhodněte, zda pro libovolnou neprázdnou množinu formulí $\mathcal{T}$ existuje formule φ taková, že φ není kontradikce a $\mathcal{T} \cup \{\varphi\}$ je nesplnitelná. Své tvrzení dokažte. (Argumentujte formálně s využitím pojmů interpretace, valuace atp.)

Příklad 5.3.5. Uvažujte množinu formulí výrokové logiky $\mathcal{T}$ a formule φ, ψ. Rozhodněte o platnosti následujících tvzení:

a) Pokud $\mathcal{T} \models \varphi$ a $\mathcal{T} \models \psi$, pak $\mathcal{T} \models \varphi \vee \psi$.

b) Pokud $\mathcal{T} \models \varphi$ a $\mathcal{T} \models \psi$, pak $\mathcal{T} \models \varphi \wedge \psi$.

c) Pokud $\mathcal{T} \vDash \varphi \vee \psi$, pak $\mathcal{T} \vDash \varphi$ nebo $\mathcal{T} \vDash \psi$.

d) Pokud $\mathcal{T} \vDash \varphi \wedge \psi$, pak $\mathcal{T} \vDash \varphi$ a $\mathcal{T} \vDash \psi$.

e) Pokud $\mathcal{T} \vDash \varphi$, pak $\mathcal{T} \vDash \varphi \vee \psi$.

f) Pokud $\mathcal{T} \vDash \varphi$, pak $\mathcal{T} \vDash \varphi \wedge \psi$.

g) Pokud $\mathcal{T} \vDash \varphi$, pak $\mathcal{T} \cup \{\psi\} \vDash \varphi$.

h) Pokud $\mathcal{T} \cup \{\psi\} \vDash \varphi$, pak $\mathcal{T} \vDash \varphi$.

Příklad 5.3.6. Ukažte, že logické vyplývání $\mathcal{T} \vDash \varphi$ platí, právě když množina $\mathcal{T} \cup \{\neg\varphi\}$ je nesplnitelná.
Poznámka: Tato vlastnost odpovídá principu důkazu sporem. Znova ji uvidíme v části zabývající se rezolucí, kde ukazujeme vyplývání pomocí nesplnitelnosti množiny formulí.

Příklad 5.3.7. S využitím věty o dedukci (a bez použití pravdivostních tabulek) rozhodněte, zda je formule $\xi \equiv p \Rightarrow (q \Rightarrow (\neg p \Rightarrow r))$ tautologie, kontratikce, splnitelná formule.

5.4 Logické spojky

Se základními logickými spojkami ($\vee$, $\wedge$, $\Rightarrow$, $\neg$, ...) jsme se již na intuitivní úrovni setkali. Podívejme se na ně ještě jednou zblízka.

Definice 19: Sémantika *n-ární logické spojky* $\square$ je dána funkcí $f_\square : \{0,1\}^n \to \{0,1\}$ následujícím způsobem: Valuace formule $\psi \equiv \square(\varphi_1, \ldots, \varphi_n)$ v interpretaci I je dána jako $I(\psi) = f_\square(I(\varphi_1), \ldots, I(\varphi_n))$, kde $I(\varphi_1), \ldots, I(\varphi_n)$ jsou valuace podformulí $\varphi_1, \ldots, \varphi_n$ příslušné interpretaci I.

Intuitivně, pro novou spojku můžeme zadat její sémantiku tabulkou (předepisující funkci $f_\square$) a formule s touto spojkou pak odpovídající způsobem vyhodnocovat. Všimněte si rovněž, že logické spojky nemusí být nutně unární (jako např. $\neg$) nebo binární (jako např. $\vee$), ale mohou mít libovolnou aritu.

Význam nové spojky ale můžeme definovat i ekvivalencí s jinou formulí. Lze tedy definovat např. nulární falsum (čili konstantní nepravdu) jako $\bot :\approx p \wedge \neg p$, či implikaci jako $\varphi \Rightarrow \psi :\approx \neg\varphi \vee \psi$.

Příklad. Definujte nulární verum $\top$ (čili konstantní pravdu) ekvivalencí s jinou formulí. Definujte disjunkci tabulkou.

Nulární verum: $\top :\approx p \vee \neg p$.
Disjunkce:

ř.	φ	ψ	$\varphi \vee \psi$
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	1

Ještě zavedeme pojem úplného systému logických spojek.

Definice 20: Množina logických spojek tvoří *úplný systém logických spojek*, pokud lze formulemi obsahujícími pouze spojky z dané množiny vyjádřit libovolnou logickou funkci (vnímáme-li formule jako funkce přiřazující zadané interpretaci svou valuaci).

Množina $\{\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow\}$ tvoří úplný systém logických spojek (důkaz vynecháváme). Ve skutečnosti, jak dále uvidíme, nám stačí pouze spojky $\{\neg, \Rightarrow\}$. Zavést ostatní spojky pomocí těchto dvou může v mnoha případech usnadnit argumentaci o formulích (stačí uvažovat o vlastnostech dvou spojek). V praxi (např. při tvorbě elektronických obvodů) se ještě používají úplné systémy logických spojek {NAND} a {NOR}[2], které si vystačí s jedinou spojkou.

Příklad 5.4.1. Kolik existuje různých vzájemně neekvivalentních n-árních spojek?

Příklad 5.4.2. Uvažujte formule výrokové logiky nad abecedou obsahující pouze logické spojky $\Rightarrow, \neg$. Zaveďte spojky $\vee, \wedge, \Leftrightarrow$ jako syntaktické zkratky v tomto systému tak, aby měly obvyklý význam. (Tj. pro slova $\varphi \vee \psi, \varphi \wedge \psi, \varphi \Leftrightarrow \psi$, kde φ, ψ jsou správně utvořené formule, uveďte formule v uvažovaném systému, které jsou jimi reprezentovány.)

Příklad 5.4.3. Mějme formuli výrokové logiky $\varphi \equiv \neg(p \wedge (q \square r)) \vee (q \Rightarrow r)$. Určete (např. tabulkou) sémantiku binární logické spojky $\square$ tak, aby formule φ byla:

a) tautologie,

b) kontradikce,

c) ekvivalentní formuli $p \Rightarrow q$,

nebo konstatujte, že formule nemůže danou podmínku splnit. Odpovědi zdůvodněte.

Příklad 5.4.4. Kolika způsoby lze definovat sémantiku spojky $\square$ z bodu a) předchozího příkladu?

Příklad 5.4.5. Ukažte, že následující množiny tvoří úplné systémy logických spojek. Vyjděte z předpokladu, že $\{\Rightarrow, \neg\}$ je úplný systém logických spojek. (Tj. vyjádřete formule $\varphi \Rightarrow \psi$ a $\neg\varphi$ pomocí formulí obsahující pouze spojky z příslušných množin.)

- {NOR}
- {NAND}

Příklad 5.4.6. Uvažujte ternární logickou spojku Δ definovanou takto: $\Delta(\varphi, \psi, \theta) :\approx (\varphi \Rightarrow \psi) \wedge \neg\theta$. Rozhodněte, zda množina $\{\Delta\}$ tvoří úplný systém logických spojek, a svou odpověď dokažte. Můžete vyjít z předpokladu, že $\{\neg, \Rightarrow\}$ je úplný systém logických spojek.

Příklad 5.4.7. Uvažujte spojku $\not\Rightarrow$ definovanou takto: $\varphi \not\Rightarrow \psi :\approx \neg(\varphi \Rightarrow \psi)$. Rozhodněte, zda množina $\{\not\Rightarrow\}$ tvoří úplný systém logických spojek, a svou odpověď dokažte. Můžete vyjít z předpokladu, že $\{\neg, \Rightarrow\}$ je úplný systém logických spojek.

[2] $\varphi \text{NAND} \psi :\approx \neg(\varphi \wedge \psi)$, $\varphi \text{NOR} \psi :\approx \neg(\varphi \vee \psi)$

5.5 Normální formy

Normální formy představují speciální způsob zápisu, z kterého lze přímo vyčíst některé sémantické vlastnosti formule. Základní stavební kameny formulí v normální formě jsou literály, znichž skládáme klauzule, resp. duální klauzule.

Definice 21:

- *Literál* je výroková proměnná nebo její negace.
- *Klauzule* je disjunkce literálů.
- *Duální klauzule* je konjunkce literálů.

Spojením klauzulí konjunkcí nebo duálních klauzulí disjunkcí získáme formuli v konjukntivní, resp. disjunktivní normální formě. Když navíc každá klauzule obsahuje všechny uvažované výrokové proměnné, bavíme se o úplné normální formě.

Definice 22: Uvažujme výrokové proměnné $p_1, \ldots, p_n$.

- Formule je v *konjunktivní normální formě (KNF)*, jedná-li se o konjunkci klauzulí (s navzájem různými množinami literálů).
- Formule je v *disjunktivní normální formě (DNF)*, jedná-li se o disjunkci duálních klauzulí (s navzájem různými množinami literálů).
- Obsahuje-li navíc každá klauzule (resp. duální klauzule) formule v KNF (resp. DNF) každou z výrokových proměnných $p_1, \ldots, p_n$ právě jednou, hovoříme o *úplné* konjunktivní (resp. disjunktivní) normální formě (ÚKNF, resp. ÚDNF)

Formule $(p \vee \neg q) \wedge (\neg p \vee \neg q \vee r)$ je v KNF, ale není v ÚKNF, protože první klauzule neobsahuje všechny implicitně uvažované výrokové proměnné p, q, r. Ekvivalentní formulí v ÚKNF je formule $(p \vee \neg q \vee r) \wedge (p \vee \neg q \vee \neg r) \wedge (\neg p \vee \neg q \vee r)$.
Pro převod do úplných normálních forem existuje poměrně přímočarý algoritmus, který je ilustrován na následujícím příkladu.

Příklad. Pomocí pravdivostní tabulky převeďte formuli $p \Leftrightarrow q$ do ÚDNF a ÚKNF.

Vytvoříme pravdivostní tabulku zadané formule a pro interpretace, v nichž je formule pravdivá, přidáme duální klauzuli do ÚDNF; pro interpretace, v nichž je formule nepravdivá, přidáme klauzuli do ÚKNF.
Klauzuli pro danou interpretaci I tvoříme tak, že pro každou výrokovou proměnnou p:

- je-li $I(p) = 0$, přidáme do klauzule literál p,
- je-li $I(p) = 1$, přidáme do klauzule literál $\neg p$.

V případě duální klauzule pro každou výrokovou proměnnou p, je-li $I(p) = 0$, přidáme literál $\neg p$, je-li $I(p) = 1$, přidáme literál p.

p	q	$p \Leftrightarrow q$	duální klauzule	klauzule
0	0	1	$(\neg p \wedge \neg q)$	
0	1	0		$(p \vee \neg q)$
1	0	0		$(\neg p \vee q)$
1	1	1	$(p \wedge q)$	

ÚDNF: $(\neg p \wedge \neg q) \vee (p \wedge q)$
ÚKNF: $(p \vee \neg q) \wedge (\neg p \vee q)$

Příklad 5.5.1. Pomocí pravdivostní tabulky převeďte formuli $r \Rightarrow \neg s$ do ÚDNF.

Příklad 5.5.2. Pomocí pravdivostní tabulky převeďte formuli $\neg r \Rightarrow (r \wedge \neg s)$ do ÚKNF.

Příklad 5.5.3. Ekvivalentními úpravami převeďte formuli $\neg(p \Leftrightarrow \neg q) \vee \neg r$ do KNF.

Příklad 5.5.4. Ekvivalentními úpravami převeďte formuli $(p \vee \neg(\neg r \Rightarrow q)) \wedge (r \Rightarrow p)$ do DNF.

Příklad 5.5.5. Ekvivalentními úpravami převeďte formuli $(\neg q \wedge r) \vee (r \wedge \neg p)$ do ÚDNF.

6 Predikátová logika

Predikátová logika přirozeně rozšiřuje logiku výrokovou. Namísto výroků zde hovoříme o predikátech, na které můžeme pohlížet jako na parametrizované výroky, jejichž pravdivost závisí i na hodnotě parametru: např. pravdivost tvrzení „x je sudé (párne) číslo“ závisí na tom, zda se za hodnotou x schovává číslo 1 nebo 2.

6.1 Syntax

Než začneme zkoumat sémantiku predikátové logiky, podívejme se opět nejprve na její syntax.

Definice 23: *Abeceda predikátové logiky* zahrnuje

- symboly pro proměnné $x, y, z, \ldots$,
- predikátové symboly $P, Q, R, \ldots$,
- funkční symboly $f, g, h, \ldots$ (příp. $a, b, c, d, \ldots$ pro konstanty),
- symboly pro logické spojky $\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \ldots$,
- pomocné symboly závorek (a) a kvantifikátorů $\forall$ a $\exists$.

Ozn. $\mathcal{V} = \{x, y, z, \ldots\}$ množinu všech proměnných.

Pozor, proměnná je v kontextu predikátové logiky něco jiného než výroková proměnná. Výrokovým proměnným ve skutečnosti odpovídají predikáty, které jsou jejich zobecněním. (Podrobněji dále.)

Novým a zcela klíčovým je zde pojem *termu*, který reprezentuje objekt a slouží jako argument pro predikáty.

Definice 24: *Term.*

- Každý symbol pro proměnnou je term.
- Je-li f n-ární funkční symbol a $t_1, \ldots, t_n$ termy, pak také $f(t_1, \ldots, t_n)$ je term.
- Nic dalšího není term.

Nulární funkce, čili konstanty, zapisujeme bez prázdných závorek, tedy c namísto $c()$.

U běžných binárních funkčních symbolů lze používat i infixový zápis bez závorek. U běžných unárních symbolů lze používat i prefixový zápis bez závorek.

Máme-li k dispozici nulární funkční symbol c, binární funkční symboly f, $+$ a proměnné $x, y, z, \ldots$, pak např. $f(x, c)$; $f(f(x, y), z)$; c; $+(x, z)$ jsou termy. Poslední uvedený lze navíc

v souladu s výše uvedenou konvencí zapisovat infixově jako $x + z$.

V matematickém kontextu tak například matematické výrazy (nikoliv rovnice!) představují termy: $x + 2$, $7 + 15$, $x^2/0$. Stále je potřeba mít na mysli, že termy reprezentují **hodnoty** či **objekty**, **nikoliv tvrzení**. Jak tedy reprezentovat tvrzení v predikátové logice?

Definice 25: *Formule predikátové logiky.*

- Je-li P n-ární predikátový symbol a $t_1, \ldots, t_n$ termy, pak $P(t_1, \ldots, t_n)$ je formule.
- Jsou-li t_1, t_2 termy, je $t_1 = t_2$ formule.
- Jsou-li φ, ψ fomule, pak rovněž $\neg(\varphi)$, $(\varphi) \vee (\psi)$, $(\varphi) \wedge (\psi)$, $(\varphi) \Rightarrow (\psi)$, $(\varphi) \Leftrightarrow (\psi)$, ... jsou formule.
- Je-li φ fomule a x proměnná, pak rovněž $\exists x(\varphi)$ a $\forall x(\varphi)$ jsou formule.
- Nic jiného formulí není.

U běžných binárních predikátových symbolů lze používat i infixový zápis bez závorek.

Formule vzniklá aplikací predikátového symbolu na termy se nazývá *atomická formule*.

V matematickém kontextu jsou matematické rovnice či nerovnice jako $x+5 = y-1$, $x^2+1 \leq x$ jednoduchými příklady formulí. Formulí však lze běžně vyjadřovat i složitější tvrzení:

$$\forall x \exists y(x + y = 0).$$

Použité funkční a predikátové symboly pak společně tvoří *jazyk* predikátové logiky[3] $\mathcal{L}$, který doplňuje definice termu a formule konkrétním kontextem.

Příklad. Ve formuli

$$\varphi \equiv \forall x(x \cdot x \geq 0 \Rightarrow \exists y(f(x, y, z) < \sin(0 + x))),$$

kde všechny symboly mají obvyklý (matematický) význam, identifikujte všechny

a) proměnné,

b) termy,

c) funkční symboly,

d) predikátové symboly.

U funkčních a predikátových symbolů určete i jejich aritu.

a) x, y, z

b) x, $x \cdot x$, 0, y, z, $f(x, y, z)$, $0 + x$, $\sin(0 + x)$

[3] Jazyk je v tomto kontextu pouze označení pro množinu funkčních a predikátových symbolů, čili jedná se o jiný pojem než formální jazyk.

c)+d) Symboly klasifikujeme v tabulce.

symbol	typ	arita	zápis
$\cdot$	funkční	2	infix
$\geq$	predikátový	2	infix
0	funkční (konstanta)	0	n/a
f	funkční	3	prefix
sin	funkční	1	prefix
$<$	predikátový	2	infix
$+$	funkční	2	infix

Jazyk formule φ je $\mathcal{L} = \{\cdot, \geq, 0, f, \sin, <, +\}$.

Proměnné navíc mohou být ve formuli vázány kvantifikátorem (tedy patřit do „scope“ kvantifikátoru), nebo být volné.

Definice 26: Výskyt proměnné x ve formuli φ (výskytem rozumíme její použití v rámci termu, nikoli kvantifikátoru) je *vázaný*, existuje-li podformule φ, ozn. ψ, která obsahuje tento výskyt a začíná $\exists x$ nebo $\forall x$. V opačném případě je výskyt proměnné *volný*.

Formule, která nemá žádný volný výskyt proměnné se nazývá *uzavřená* nebo též *sentence*.

Běžně používané značení $\varphi(x_1, \ldots, x_n)$ znamená, že formule φ má volný výskyt proměnných $x_1, \ldots, x_n$, ostatní proměnné jsou ve formuli vázané.

Příklad. Klasifikujte výskyty proměnných ve formuli $\forall z(x = y \Rightarrow \exists x P(x, y)) \vee Q(x)$.

Proměnná z nemá ve formuli žádný výskyt (za výskyt považujeme, když je proměnná součásti termu, ne kvantifikátoru). Proměnná x má ve formuli tři výskyty: nejprve volný, potom vázaný a nakonec opět volný. Proměnná y má ve formuli dva výskyty, oba volné.

Příklad 6.1.1. Uvažujte jazyk $\mathcal{L}$ predikátové logiky obsahující jeden unární funkční symbol f a jeden nulární funkční symbol c. Popište množinu všech termů jazyka $\mathcal{L}$.

Příklad 6.1.2. Uvažujte jazyk $\mathcal{L}$ predikátové logiky obsahující funkční a predikátové symboly zadané následující tabulkou:

symbol	typ	arita
f	funkční	2
c	funkční	0
P	predikátový	1
Q	predikátový	2

Rozhodněte, která z následujících slov jsou **termy** jazyka $\mathcal{L}$:

a) $y = x$,

b) $Q(c, c)$,

c) $f(x, c)$,

d) $\forall y(f(f(c, c), y))$,

e) z,

f) $f(f(c))$,

g) $f(x, y) \vee f(c, x)$,

h) $f(f(c, x), f(y, z))$.

Příklad 6.1.3. Uvažujte jazyk $\mathcal{L}$ predikátové logiky z předchozího příkladu. Rozhodněte, která z následujících slov jsou **formulemi** jazyka $\mathcal{L}$:

a) $y = x$,

b) $\forall x(Q(c, c) = x)$,

c) $f(x, c)$,

d) $\forall y(f(f(c, c), y))$,

e) $Q(f(x, c))$,

f) $\forall x(Q(f(f(c, c), y), f(z, c)) \vee P(f(c, z)))$,

g) $\forall x \exists c(f(x, c) = c)$.

Příklad 6.1.4. Uvažujte jazyk $\mathcal{L}$ predikátové logiky obsahující jediný funkční symbol f arity 2 a žádný predikátový symbol. Uveďte příklad formule jazyka $\mathcal{L}$

a) s jedinou proměnnou x s právě dvěma vázanými výskyty a žádným volným výskytem,

b) s jedinou proměnnou x s právě jedním vázaným a jedním volným výskytem,

c) pouze s proměnnými x, y se dvěma vázanými a žádným volným výskytem, resp. s jedním volným a žádným vázaným výskytem,

d) pouze s proměnnými x, y, každou s jedním volným a jedním vázaným výskytem.

Příklad 6.1.5. Uvažujte jazyk $\mathcal{L}_1$ predikátové logiky, který obsahuje právě dva funkční symboly:

- nulární symbol (konstantu) k,
- unární symbol f

a žádný predikátový symbol. Dále uvažujte jazyk $\mathcal{L}_2$, který navíc k těmto dvěma funkčním symbolům obsahuje unární predikátový symbol P. Rozhodněte o pravdivosti následujících tvrzení a svou odpověď zdůvodněte:

1. Množina $\mathcal{T} = \{f^i(k) \mid i \in \mathbb{N}_0\} = \{k, f(k), f(f(k)), f(f(f(k))), \dots\}$ jsou právě všechny termy jazyka $\mathcal{L}_1$.
2. Neexistuje formule predikátové logiky jazyka $\mathcal{L}_1$.
3. Uvažujte formuli $\varphi \equiv \forall x P(x)$ jazyka $\mathcal{L}_2$. Existují podformule ψ_1, ψ_2 formule φ takové, že všechny proměnné formule ψ_1 mají v ψ_1 volný výskyt a všechny proměnné formule ψ_2 mají v ψ_2 vázaný výskyt?

6.2 Sémantika

Podobně jako u výrokové logiky je nyní namístě dodat formulím predikátové logiky konkrétní význam. Poslouží k tomu opět *interpretace*, zde ovšem mající podobu složitější struktury.

Definice 27: *Interpretace* (též *realizace*) jazyka $\mathcal{L}$ je struktura I zahrnující
- neprázdné *univerzum* (či *doménu*) $\mathcal{D}_I$,
- n-ární relaci $I(P) \subseteq \mathcal{D}_I^n$ pro každý n-ární predikátový symbol P jazyka $\mathcal{L}$,
- zobrazení $I(f) : \mathcal{D}_I^n \to \mathcal{D}_I$ pro každý n-ární funkční symbol f jazyka $\mathcal{L}$.

Univerzum specifikuje množinu objektů, nad kterými pracujeme. Termy pak reprezentují hodnoty z tohoto univerza. Místo jednoduchého přiřazení jedniček a nul výrokovým proměnným jsou predikáty chápány jako relace a jejich pravdivost závisí na hodnotách jejich argumentů.

Abychom mohli vyhodnocovat termy, je potřeba ještě definovat, jak interpretovat proměnné. K tomu slouží valuace.

Definice 28: *Valuace* je zobrazení $V : \mathcal{V} \to D_I$ přiřazující proměnným prvky univerza.

Hodnotu termu v interpretaci a valuaci pak upřesňuje následující induktivní definice.

Definice 29: *Hodnota termu t v interpretaci I a valuaci V* je prvek univerza $|t|_{I,V} \in \mathcal{D}_I$, kde
- $|t|_{I,V} = V(x)$ pro $t = x$, kde x je proměnná,
- $|t|_{I,V} = I(f)(|t_1|_{I,V}, \dots, |t_n|_{I,V})$ pro $t = f(t_1, \dots, t_n)$, kde f je n-ární funkce a $t_1, \dots, t_n$ jsou termy.

Příklad. Uvažujte jazyk $\mathcal{L}$ s jedním nulárním funkčním symbolem c a jedním binárním funkčním symbolem f a jeho interpretaci I, kde
- Univerzum tvoří celá čísla 0, 1 a 2, tj. $\mathcal{D}_I = \{0, 1, 2\}$.

- Symbol f se realizuje jako sčítání modulo 3, tj. $I(f)(x,y) = (x+y) \mod 3$.
- Symbol c se realizuje jako číslo 1, tj. $I(c) = 1$.

Dále uvažujte valuaci V, která všem proměnným přiřazuje hodnotu 2, tj. $V(x) = 2$ pro všechna $x \in \mathcal{V}$. Určete hodnoty následujících termů v interpretaci I a valuaci V:

a) c

b) $f(c,x)$

c) $f(x,f(x,y))$

a) Podle druhého bodu definice hodnoty termu je $|c|_{I,V} = I(c) = 1$.

b) Podle druhého bodu definice hodnoty termu je $|f(c,x)|_{I,V} = I(f)(|c|_{I,V}, |x|_{I,V})$. Podle druhého, resp. prvního bodu definice máme $|c|_{I,V} = I(c) = 1$ a $|x|_{I,V} = V(x) = 2$, a tedy celkem $|f(c,x)|_{I,V} = I(f)(1,2) = (1+2) \mod 3 = 0$.

c) Máme postupně $|f(x,f(x,y))|_{I,V} = I(f)(|x|_{I,V}, |f(x,y)|_{I,V}) = (2+1) \mod 3 = 0$, protože $|x|_{I,V} = V(x) = 2$ a $|f(x,y)|_{I,V} = I(f)(|x|_{I,V}, |y|_{I,V}) = (2+2) \mod 3 = 1$, neboť $|x|_{I,V} = |y|_{I,V} = V(x) = 2$.

Nyní můžeme přistoupit k definování pravdivosti formule v dané interpretaci a valuaci.

Definice 30: Zda formule φ je *pravdivá v interpretaci I a valuaci V*, značeno $\vDash_I^V \varphi$, definujeme induktivně takto:

- $\vDash_I^V P(t_1,\ldots,t_n)$, právě když $(|t_1|_{I,V},\ldots,|t_n|_{I,V}) \in I(P)$,
- $\vDash_I^V t_1 = t_2$, právě když $|t_1|_{I,V} = |t_2|_{I,V}$,
- $\vDash_I^V \neg\psi$, právě když není $\vDash_I^V \psi$,
- $\vDash_I^V \psi \wedge \xi$, právě když $\vDash_I^V \psi$ a $\vDash_I^V \xi$ (a podobně pro zbylé logické spojky; vynecháno),
- $\vDash_I^V \forall x\psi$, právě když pro všechny prvky univerza $d \in \mathcal{D}$ platí $\vDash_I^{V'} \psi$, kde valuace V' vznikla z V nahrazením obrazu proměnné x prvkem d,
- $\vDash_I^V \exists x\psi$, právě když pro nějaký prvek univerza $d \in \mathcal{D}$ platí $\vDash_I^{V'} \psi$, kde valuace V' vznikla z V nahrazením obrazu proměnné x prvkem d.

Příklad. Uvažujte jazyk $\mathcal{L}$ predikátové logiky obsahující jeden unární funkční symbol f a jeden binární predikátový symbol P. Dále uvažujte interpretaci I jazyka $\mathcal{L}$:

- Univerzum tvoří všechna reálná čísla, tj. $\mathcal{D}_I = \mathbb{R}$.
- Symbol f se realizuje jako unární minus, tj. $I(f)(x) = -x$.
- Symbol P se realizuje jako ostré menší než, tj. $I(P) = \{(x,y) \in \mathbb{R}^2; x < y\}$.

Nechť V je valuace přiřazující všem proměnným hodnotu 1, tj. $V(x) = 1$ pro všechna $x \in \mathcal{V}$. Rozhodněte, zda následující formule jsou pravdivé v interpretaci I a valuaci V:

a) $P(x,f(x))$

b) $\exists x P(x, f(x))$

c) $\exists x \forall y P(x, y)$

d) $\exists x P(x, y)$

a) Máme $|x|_{I,V} = 1$, $|f(x)|_{I,V} = -1$. Dle prvního řádku definice je formule $P(x, f(x))$ pravdivá, právě když $(|x|_{I,V}, |f(x)|_{I,V}) \in I(P)$, tj. $(1, -1) \in \{(x, y) \in \mathbb{R}^2; x < y\}$, neboli $1 < -1$, což zjevně neplatí.

Více neformálně: Termy x a $f(x)$ se vyhodnotí na 1, resp. -1. Poněvadž se symbol P realizuje jako ostrá nerovnost a neplatí $1 < -1$, je formule nepravdivá.

b) Ve formuli se vyskytuje pouze proměnná x, zúžíme tedy argumentaci pouze na ni. Formule je pravdivá, jestliže existuje prvek univerza, jehož dosazením za x docílíme toho, že je formule $P(x, f(x))$ pravdivá; takovým prvkem je ovšem např. číslo -1.

c) Formule je pravdivá, jestliže existuje prvek, jehož dosazením za x docílíme toho, že je formule $\forall y P(x, y)$ pravdivá. Takový prvek nicméně neexistuje, neboť při dosazení libovolného reálného čísla $d \in \mathcal{D}_I$ za x budou existovat hodnoty y, pro které není $P(x, y)$ pravdivá – např. číslo $d - 1 \in \mathcal{D}_I$ (a tudíž není $P(x, y)$ pravdivá pro libovolnou hodnotu y).

Pozor na pořadí kvantifikátorů! Aby byla uvažovaná formule pravdivá, musí existovat hodnota x taková, že již formule $P(x, y)$ platí pro libovolné y.

d) Proměnná y má v uvažované valuaci hodnotu 1, zjevně existuje hodnota x, aby formule $P(x, y)$ byla pravdivá (kupř. $V(x) = 0$), a tedy uvažovaná formule je pravdivá.

Pravdivost formule (pouze) v interpretaci je definována tak, aby byla ekvivalentní doplnění univerzálních kvantifikátorů pro volné proměnné, tedy

$$\vDash_I P(x, y), \quad \text{právě když} \quad \vDash_I \forall x \forall y P(x, y) :$$

Definice 31:

- Formule predikátové logiky je *pravdivá v interpretaci* I, ozn. $\vDash_I \varphi$, jestliže je pravdivá v interpretaci I a valuaci V pro libovolnou valuaci V.
- Formule predikátové logiky je *logicky pravdivá* či *tautologie*, ozn. $\vDash \varphi$, jestliže je pravdivá v každé interpretaci.

Příklad 6.2.1. Ukažte, že existenční kvantifikátor $\exists$ lze v predikátové logice zavést jako syntaktickou zkratku, tj. ekvivalentními úpravami vyjádřete $\exists x \varphi$ bez použití symbolu $\exists$.

Příklad 6.2.2. Nalezněte negace následujících formulí tak, aby se symboly $\neg$ nacházely výhradně bezprostředně před predikátovými symboly.

a) $\forall x(P(x) \vee \exists y Q(y))$

b) $\forall x(P(x) \Rightarrow Q(x)) \wedge \exists x(R(x) \wedge S(x))$

c) $(\exists x P(x) \vee \forall y Q(y)) \Leftrightarrow \forall x(\exists y R(x,y) \Rightarrow Q(x))$

Příklad 6.2.3. Uvažte formuli $\varphi \equiv \exists x(\forall y R(x,y) \Rightarrow Q(x))$. Pro každou z následujících interpretací rozhodněte, zda je modelem formule φ.

a) $\mathcal{D}_I = \{0\}$, $I(R) = \varnothing$, $I(Q) = \varnothing$

b) $\mathcal{D}_I = \{0\}$, $I(R) = \{(0,0)\}$, $I(Q) = \varnothing$

c) $\mathcal{D}_I = \mathbb{Z}$, $I(R) = \mathbb{Z} \times \mathbb{Z}$, $I(Q) = \mathbb{N}_0$

Příklad 6.2.4. Uvažte formuli $\varphi \equiv \forall y(\exists x R(x,y) \Rightarrow Q(y))$. Pro každou z následujících odrážek popište právě ta $I(Q)$, že interpretace I je modelem formule φ.

a) $\mathcal{D}_I = \{0\}$, $I(R) = \{(0,0)\}$

b) $\mathcal{D}_I = \{0\}$, $I(R) = \varnothing$

c) $\mathcal{D}_I = \mathbb{Z}$, $I(R) = <$

d) $\mathcal{D}_I = \mathbb{Z}$, $I(R) = \{(x,y) \in \mathbb{Z} \times \mathbb{Z} \mid x \text{ je sudé číslo}\}$

e) $\mathcal{D}_I = \mathbb{Z}$, $I(R) = \{(x,y) \in \mathbb{Z} \times \mathbb{Z} \mid y \text{ je sudé číslo}\}$

Příklad 6.2.5. Rozhodněte, v jakých interpretacích jsou pravdivé následující formule:

a) $\forall x \exists y \exists z(((x = y) \vee (x = z)) \wedge (y \neq z))$

b) $\forall x \forall y \forall z((Q(x,y) \wedge Q(y,z)) \Rightarrow Q(x,z))$

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

d) $\forall x(x \neq f(x) \wedge x = f(f(x)))$

Příklad 6.2.6. Uvažujte jazyk $\mathcal{L}$ predikátové logiky obsahující právě dva binární funkční symboly $*$ a $+$. Uvažujte interpretaci (realizaci) I jazyka $\mathcal{L}$, kde univerzum tvoří všechna reálná čísla $\mathbb{R}$ a funkční symboly se realizují běžným (matematickým) způsobem, tj. $+$ jako sčítání a $*$ jako násobení. Nalezněte

a) fomuli $\alpha(x,y,z)$, kt. je pravdivá v I právě v těch valuacích V, že $V(x) - V(y) = V(z)$,

b) fomuli $\beta(x)$, která je pravdivá v I právě v těch valuacích V, že $V(x)$ je 0,

c) fomuli $\gamma(x)$, která je pravdivá v I právě v těch valuacích V, že $V(x)$ je 1,

d) fomuli $\delta(x)$, která je pravdivá v I právě v těch valuacích V, že $V(x)$ je kladné číslo nebo 0,

e) fomuli $\varepsilon(x,y)$, která je pravdivá v I právě v těch valuacích V, že $V(x) \leq V(y)$.

Při definování formulí se můžete odvolat na formule, které jste již zavedli v předchozích bodech.

6.3 Normální formy

Jelikož rezoluce v predikátové logice vyžaduje práci s formulemi určitých tvarů, je nutné se s těmito tvary seznámit, abychom mohli rezoluci provádět. Podobně jako ve výrokové logice jim říkáme normální formy. První normální forma, prenexová, má všechny kvantifikátory umístěny před samotným jádrem formule, které je v KNF. Literálem v kontextu predikátové logiky rozumíme libovolnou atomickou formuli nebo její negaci.

Definice 32: Uzavřená formule φ se nachází v *prenexové normální formě (PNF)*, je-li tvaru

$$Q_1 x_1 \dots Q_n x_n \psi,$$

kde $Q_i \in \{\forall, \exists\}$, $x_1, \dots, x_n$ jsou proměnné a formule ψ je v konjunktivní normální formě (zejména tedy neobsahuje žádný kvantifikátor).

Například formule $\forall x \exists y((P(f(y)) \vee \neg Q(x)) \wedge (\neg P(y) \vee P(x)))$ se nachází v PNF.

Skolemova normální forma oproti PNF ještě navíc vyžaduje, aby veškeré proměnné ve formuli byly všeobecně kvantifikované.

Definice 33: Uzavřená formule φ se nachází ve *Skolemově normální formě*, nachází-li se v prenexové normální formě a obsahuje pouze obecné kvantifikátory $\forall$.

Například formule $\forall x((P(f(g(x))) \vee \neg Q(x)) \wedge (\neg P(g(x)) \vee P(x)))$ se nachází ve Skolemově NF, jelikož vznikla skolemizací předchozí formule v PNF. Procesy převodu do PNF a do Skolemovy NF si ukážeme na příkladech.

Příklad. Převeďte formuli $\exists x \forall y(P(x, y) \Rightarrow \neg \forall x Q(y, x))$ do PNF.

Algoritmus převodu sestává z několika kroků.

I. Eliminujeme zbytečné kvantifikátory. (Takové v zadané formuli nejsou.)

II. Přejmenujeme proměnné tak, aby u každého kvantifikátoru byla jiná proměnná:
$\exists x_1 \forall y(P(x_1, y) \Rightarrow \neg \forall x_2 Q(y, x_2))$

III. Eliminujeme jiné spojky než $\vee$, $\wedge$, $\neg$:
$\exists x_1 \forall y(\neg P(x_1, y) \vee \neg \forall x_2 Q(y, x_2))$

IV. Přesuneme negaci až před samotné predikáty (s využitím de Morganových zákonů a pravidel pro negaci kvantifikátoru):
$\exists x_1 \forall y(\neg P(x_1, y) \vee \exists x_2 \neg Q(y, x_2))$

V. Kvantifikátory přesuneme ven z jádra formule: $\exists x_1 \forall y \exists x_2(\neg P(x_1, y) \vee \neg Q(y, x_2))$

VI. Jádro formule upravíme do KNF pomocí distributivních zákonů. (Již je v KNF.)

Nyní si demonstrujeme převod výsledné formule do Skolemovy NF.

Příklad. Převeďte formuli $\exists x_1 \forall y \exists x_2 (\neg P(x_1, y) \vee \neg Q(y, x_2))$ do Skolemovy NF.

Převod z PNF do Skolemovy NF spočívá v citlivém odstranění existenčních kvantifikátorů. Jejich roli v nově vytvořené formuli zastoupí Skolemovy funkce, které se vyhodnotí na hodnotu, kterou by původně „vybral" existenční kvantifikátor.

V zadané formuli je volba hodnoty proměnné x_1 pevná (nezávisí na dalších proměnných, protože existenční kvantifikátor je úplně první), nahradíme tedy proměnnou x_1 konstantou c a kvantifikátor odstraníme $(x_1 \to c)$:
$\forall y \exists x_2 (\neg P(c, y) \vee \neg Q(y, x_2))$

Volba hodnoty proměnné x_2 již pevná není, ale může záviset na konkrétní hodnotě y, protože se obecný kvantifikátor s y nachází před kvantifikátorem s x_2. Nahradíme tedy proměnnou x_2 funkcí $f(y)$ a kvantifikátor odstraníme $(x_2 \to f(y))$:
$\forall y (\neg P(c, y) \vee \neg Q(y, f(y)))$

Nalezená formule je již ve Skolemově NF.

Příklad 6.3.1. Převeďte následující formule do prenexové normální formy a proveďte skolemizaci.

a) $(\forall x \exists y Q(x, y) \vee \exists x \forall y P(x, y)) \wedge \neg \exists x \exists y \forall z R(x, y, z)$

b) $\neg(\forall x \exists y P(x, y) \Rightarrow \exists x \exists y R(x, y)) \wedge \forall x (\neg \exists y Q(x, y))$

Příklad 6.3.2. Uvažujte následující formule:

a) $\alpha \equiv \forall x \forall y \exists z (x + z = y)$,

b) $\beta \equiv \exists x \forall y (x * y = x)$,

c) $\gamma \equiv \forall x \forall y \exists r (x \leq y \Rightarrow x \leq r \leq y)$

d) $\delta \equiv \forall x_1 \forall y_1 \forall x_2 \forall y_2 \exists x_3 \exists y_3 (\sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2} = \sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2})$

e) $\varepsilon \equiv \forall x \exists y (x > 0 \Rightarrow x = 2^y))$

nad doménou reálných čísel, kde matematické symboly $+, *, \cdot^2, \sqrt{\cdot}, \ldots$ se interpretují běžným způsobem (tj. $+$ jako sčítání, $*$ jako násobení, ...). Proveďte skolemizaci těchto formulí a určete nějakou možnou interpretaci pro každou z nově vzniklých Skolemových funkcí (vč. konstant) tak, aby byla zachována pravdivost formulí.

7 Důkazové systémy a rezoluce

Předmětem této kapitoly budou formální systémy, které umožňují vyvozovat závěry na strojové úrovni. Nejprve se seznámíme s axiomatickými důkazovými systémy, posléze s rezolucí ve výrokové i predikátové logice a nakonec s programovacím jazykem Prolog, jehož výpočty jsou na principu rezoluce založeny.

7.1 Axiomatické systémy

Axiomatické systémy umožňují formalizovat pojem matematického důkazu. Axiomatický systém obsahuje axiomy (resp. schémata axiomů) a odvozovací pravidla, která specifikují, jak z platných tvrzení odvodit další platné tvrzení. V rámci výrokové logiky budeme používat Łukasiewiczův axiomatický systém.

Definice 34: *Łukasiewiczův axiomatický systém* zahrnuje schémata axiomů

- A1: $\varphi \Rightarrow (\psi \Rightarrow \varphi)$
- A2: $(\varphi \Rightarrow (\psi \Rightarrow \xi)) \Rightarrow ((\varphi \Rightarrow \psi) \Rightarrow (\varphi \Rightarrow \xi))$
- A3: $(\neg\varphi \Rightarrow \neg\psi) \Rightarrow (\psi \Rightarrow \varphi)$

a odvozovací pravidlo

- MP (modus ponens): Z φ a $\varphi \Rightarrow \psi$ odvoď ψ.

Následující definice formálního důkazu se nevztahuje pouze k Łukasiewiczovému systému, ale obecně k libovolnému zvolenému axiomatickému systému.

Definice 35: *Formální důkaz* formule φ v axiomatickém systému je konečná posloupnost formulí $\psi_1, \ldots, \psi_n$, kde $\psi_n \equiv \varphi$ a pro každé $1 \leq i \leq n$ je ψ_i instancí nějakého schématu axiomu (tj. dosazením konkrétních formulí za symboly φ, ψ, ξ apod.) nebo vzniklo aplikací odvozovacího pravidla na formule, které mu v posloupnosti předcházely.

Formule φ je *dokazatelná*, zapisováno $\vdash \varphi$, jestliže v uvažovaném axiomatickém systému existuje její formální důkaz.

S definicí formálního důkazu již můžeme hovořit o *dokazatelnosti* formule ve zvoleném axiomatickém systému a o *korektnosti* či *úplnosti* axiomatického systému.

Definice 36:

- Je-li každá v uvažovaném systému dokazatelná formule logicky pravdivá (čili pro každou formuli φ platí pokud $\vdash \varphi$, pak $\vDash \varphi$), hovoříme o *korektním* systému.
- Je-li každá logicky pravdivá formule v uvažovaném systému dokazatelná (čili pro každou formuli φ platí pokud $\vDash \varphi$, pak $\vdash \varphi$), hovoříme o *úplném* systému.

Příklad. Uveďte formální důkaz formule $\varphi \equiv p \Rightarrow p$ v Łukasiewiczově ax. systému.

- A1: $p \Rightarrow ((p \Rightarrow p) \Rightarrow p)$
- A2: $\Big(p \Rightarrow \big((p \Rightarrow p) \Rightarrow p\big)\Big) \Rightarrow \Big(\big(p \Rightarrow (p \Rightarrow p)\big) \Rightarrow (p \Rightarrow p)\Big)$
- MP: $(p \Rightarrow (p \Rightarrow p)) \Rightarrow (p \Rightarrow p)$
- A1: $p \Rightarrow (p \Rightarrow p)$
- MP: $p \Rightarrow p$

Příklad 7.1.1. Uveďte formální důkazy následujících formulí v Łukasiewiczově axiomatickém systému:

a) $\varphi \equiv (p \Rightarrow (q \Rightarrow r)) \Rightarrow (p \Rightarrow p)$

b) $\psi \equiv q \Rightarrow (p \Rightarrow p)$

Příklad 7.1.2. Uvažujte axiomatický systém zahrnující schémata axiomů (φ, ψ mohou být libovolné formule)

- A1: $\varphi \Rightarrow \varphi$
- A2: $\varphi \Rightarrow (\psi \Rightarrow (\varphi \Rightarrow \psi))$

a odvozovací pravidlo MP (modus ponens).

a) Dokažte, že je uvažovaný systém korektní, tj. pro každou formuli φ v tomto systému platí, že pokud $\vdash \varphi$, pak $\vDash \varphi$.

b) Dokažte, že uvažovaný systém není úplný, tj. existuje formule φ, pro kterou platí $\vDash \varphi$, ale neplatí $\vdash \varphi$.

Nápověda k b): Hledejte syntaktickou vlastnost, kterou zachovávají formule dokazatelné v tomto systému.

7.2 Obecná rezoluce ve výrokové logice

V rámci pojednávání o rezoluční metodě budeme podobně jako dříve pracovat s *klauzulemi*, tedy disjunkcemi literálů. Budeme ovšem používat množinový zápis, kdy klauzuli $\ell_1 \vee \cdots \vee \ell_n$ zapíšeme jako množinu literálů $\{\ell_1, \ldots, \ell_n\}$. Speciálně zavádíme symbol $\square$ pro prázdnou množinu literálů, tedy kontradikci.

Definice 37: *Rezoluční pravidlo*: Uvažujme dvě klauzule $\{p, \ell_1, \ldots, \ell_n\}$, $\{\neg p, \ell'_1, \ldots, \ell'_m\}$, kde ℓ_i, resp. ℓ'_i jsou literály. Jejich *rezolventou* je klauzule $\{\ell_1, \ldots, \ell_n, \ell'_1, \ldots, \ell'_m\}$.

Rezolventou klauzulí $\{\neg p, r, \neg t, u\}$ a $\{s, \neg q, \neg r\}$ je klauzule $\{\neg p, \neg t, u, s, \neg q\}$. Klauzule $\{\neg p, r, \neg t, u\}$ a $\{\neg r\}$ mají rezolventu $\{\neg p, \neg t, u\}$.

Opakovanou aplikací rezolučního pravidla vzniká rezoluční důkaz (všimněte si podobnosti s definicí formálního důkazu v axiomatickém systému):

Definice 38: *Rezoluční důkaz* klauzule C z množiny klauzulí S je konečná posloupnost klauzulí $C_1, \ldots, C_n$, kde $C_n = C$ a pro každé $1 \leq i \leq n$ je $C_i \in S$ nebo C_i vzniklo aplikací rezolučního pravidla na klauzule, které mu v posloupnosti předcházely.

Klauzule C je *rezolučně dokazatelná* z S, zapisováno $S \vdash_R C$, jestliže existuje její rezoluční důkaz z S.

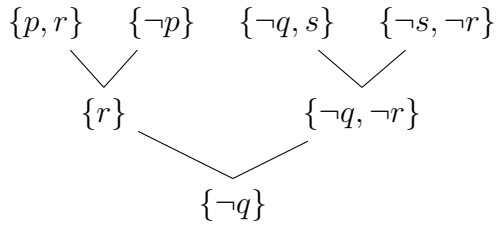
Rezoluční důkaz prázdné klauzule $\square$ z množiny klauzulí S nazýváme *vyvrácením* S.

Důležitá vlastnost rezolučního pravidla je, že zachovává pravdivost v interpretaci (a tudíž i splnitelnost). Dokážeme-li tedy z nějaké množiny klauzulí S klauzuli C, vyplývá klauzule C z množiny S. Dokážeme-li z nějaké množiny klauzulí S prázdnou klauzuli $\square$, je množina S nesplnitelná.

Příklad. Uveďte rezoluční důkaz klauzule $C = \{\neg q\}$ z množiny $S = \{\{p, r\}, \{\neg s, \neg r\}, \{\neg q, s\}, \{\neg p\}\}$.

- $C_1 = \{p, r\} \in S$
- $C_2 = \{\neg p\} \in S$
- $C_3 = \{r\}$ rezolventa C_1 a C_2
- $C_4 = \{\neg s, \neg r\} \in S$
- $C_5 = \{\neg q, s\} \in S$
- $C_6 = \{\neg q, \neg r\}$ rezolventa C_4 a C_5
- $C_7 = \{\neg q\} = C$ rezolventa C_3 a C_6

Pro lepší představu je někdy výhodné zapisovat rezoluční důkazy pomocí stromu místo posloupnosti klauzulí. Kořenem stromu je vždy výsledná klauzule a potomky každého vnitřního uzlu jsou klauzule, jejichž je daný uzel rezolventou. Listy pak tvoří prvky množiny S. Rezoluční strom pro právě uvedený příklad by mohl vypadat následovně:



Mnohdy[4] je výhodnější místo odvození konkrétní klauzule ukázat nesplnitelnost množiny klauzulí. Následující věta shrnuje, jak lze vyplývání dokázat vyvrácením množiny formulí.

[4]Konkrétně například při strojovém dokazování u dalších typů rezoluce.

Věta 39: Pro množinu formulí $\mathcal{T}$ a formuli φ platí $\mathcal{T} \models \varphi$, právě když $\mathcal{T} \cup \{\neg\varphi\}$ je nesplnitelná.

Proto budeme často vyplývání $\mathcal{T} \models \varphi$ ukazovat tak, že namísto rezolučního důkazu $\mathcal{T} \vdash_R \varphi$ nalezneme rezoluční vyvrácení $\mathcal{T} \cup \{\neg\varphi\} \vdash_R \square$.

Důkaz Věty 39 není zvlášť náročný a můžete si ho rozmyslet v rámci Příkladu 5.3.6.

Příklad 7.2.1. Obecnou rezolucí ukažte nesplnitelnost následujících množin klauzulí.

a) $\{\{\neg p, \neg q, \neg s\}, \{\neg p, \neg q, s\}, \{\neg p, q\}, \{p, \neg q\}, \{p, q\}\}$

b) $\{\{p, q, \neg r\}, \{r\}, \{\neg p, q, \neg r\}, \{\neg q, \neg r, \neg s\}, \{s\}\}$

Příklad 7.2.2. Obecnou rezolucí dokažte následující logická vyplývání. Vyplývání dokažte jak přímo, tak pomocí rezolučního vyvrácení.

a) $\{\neg p \vee q, \neg r \Rightarrow \neg q\} \models p \Rightarrow r$

b) $\{(\neg p \wedge r) \Rightarrow q, (\neg p \wedge q) \Rightarrow s, r \Rightarrow \neg p\} \models r \Rightarrow s$

Příklad 7.2.3. Dokažte, že rezoluční pravidlo zachovává pravdivost v interpretaci, tedy že pokud množina klauzulí $\{\{p, \ell_1, \ldots, \ell_n\}, \{\neg p, \ell'_1, \ldots, \ell'_m\}\}$, kde ℓ_i, resp. ℓ'_i jsou literály, je pravdivá v interpretaci I, pak klauzule $\{\ell_1, \ldots, \ell_n, \ell'_1, \ldots, \ell'_m\}$ je rovněž pravdivá v interpretaci I.

Příklad 7.2.4. Uvažujte upravené rezoluční pravidlo, kde rezolventou klauzulí $\{p, q, \ell_1, \ldots, \ell_n\}$, $\{\neg p, \neg q, \ell'_1, \ldots, \ell'_m\}$, kde ℓ_i, resp. ℓ'_i jsou literály, je klauzule $\{\ell_1, \ldots, \ell_n, \ell'_1, \ldots, \ell'_m\}$. (Tedy rezolvovat lze na dvou proměnných naráz.) Zachovává toto rezoluční pravidlo pravdivost v interpretaci? Svou odpověď dokažte.

7.3 SLD rezoluce ve výrokové logice

Kromě obecné rezoluce existují i specifičtější typy rezolucí. Jedním z nich je SLD rezoluce, tedy rezoluce na uspořádaných klauzulích s výběrovým pravidlem, která se používá ke strojovému dokazování.

Výběrové pravidlo specifikuje, na kterém literálu rezolvujeme. Obvykle budeme používat výběrové pravidlo vybírající první literál v klauzuli – dává smysl pouze v uspořádaných klauzulích ($n$ticích), nikoliv však v obecných klauzulích (množinách).

Definice 40: *Rezoluční pravidlo pro SLD rezoluci*: Uvažujme uspořádané klauzule $[\neg p_1, \ldots, \neg p_{i-1}, \neg p_i, \neg p_{i+1} \ldots, \neg p_n]$, $[p_i, \neg q_1, \ldots \neg q_m]$, kde i je určeno výběrovým pravidlem. Jejich *SLD rezolventou* je usp. klauzule $[\neg p_1, \ldots, \neg p_{i-1}, \neg q_1, \ldots, \neg q_m, \neg p_{i+1}, \ldots, \neg p_n]$.

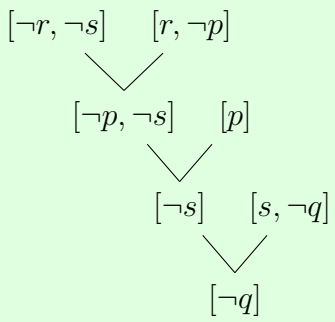
Rezolventou klauzulí $[\neg p, \neg r, \neg s]$ a $[r, \neg t, \neg s]$ je $[\neg p, \neg t, \neg s, \neg s]$. Klauzule $[\neg p, \neg r, \neg s]$ a $[r]$ pak mají rezolventu $[\neg p, \neg s]$.

Uvažujeme-li výběrové pravidlo vybírající první literál v klauzuli, pak je vždy při aplikaci rezolučního pravidla $i = 1$: klauzuli $[\neg p_1, \neg p_2, \ldots, \neg p_n]$ vždy rezolvujeme s klauzulí tvaru $[p_1, \neg q_1, \ldots, \neg q_m]$ s rezolventou $[\neg q_1, \ldots, \neg q_m, \neg p_2, \ldots, \neg p_n]$.

Definice 41: *Důkaz SLD rezolucí* uspořádané klauzule C z množiny uspořádaných klauzulí S je konečná posloupnost klauzulí $C_1, \ldots, C_n$, kde $C_1 \in S$, $C_n = C$ a pro každé $1 \leq i < n$ je C_{i+1} SLD rezolventou C_i s nějakou klauzulí $C' \in S$.

Příklad. Uveďte důkaz SLD rezolucí klauzule $C = [\neg q]$ z množiny $S = \{[r, \neg p], [\neg r, \neg s], [s, \neg q], [p]\}$ a nakreslete příslušný rezoluční strom.

- $C_1 = [\neg r, \neg s]$
- $C_2 = [\neg p, \neg s]$ rezolventa C_1 a $[r, \neg p]$
- $C_3 = [\neg s]$ rezolventa C_2 a $[p]$
- $C_4 = [\neg q] = C$ rezolventa C_3 a $[s, \neg q]$

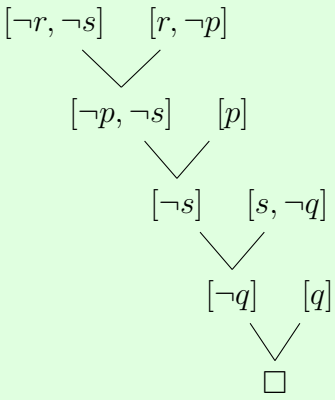


U strojového dokazování pomocí SLD rezoluce se v souladu s Větou 39 používá vyvrácení množiny klauzulí.

Příklad. Pomocí SLD rezolučního vyvrácení dokažte, že klauzule $C = [\neg q]$ vyplývá z množiny $S = \{[r, \neg p], [\neg r, \neg s], [s, \neg q], [p]\}$ a nakreslete příslušný rezoluční strom.

Podle Věty 39 vyplývá klauzule C z množiny S, právě když je množina $S \cup \{\neg C\}$ nesplnitelná. SLD rezolucí tedy vyvrátíme množinu $\{[r, \neg p], [\neg r, \neg s], [s, \neg q], [p]\} \cup \{[q]\}$

- $C_1 = [\neg r, \neg s]$
- $C_2 = [\neg p, \neg s]$ rezolventa C_1 a $[r, \neg p]$
- $C_3 = [\neg s]$ rezolventa C_2 a $[p]$
- $C_4 = [\neg q]$ rezolventa C_3 a $[s, \neg q]$
- $C_5 = \square$ rezolventa C_4 a $[q]$



SLD rezoluce je úplná pro speciální množinu klauzulí, které se nazývají Hornovy klauzule.

Definice 42: *Hornovy klauzule* jsou klauzule, které obsahují nejvýše jeden pozitivní literál (tedy literál bez negace). Ten v případě uspořádaných klauzulí navíc zapisujeme jako první. Dělíme je na

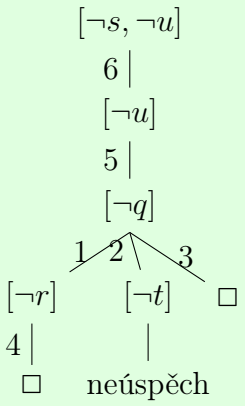
- *fakta* – klauzule s právě jedním, pozitivním literálem (např. $[s]$, $[r]$),
- *pravidla* – klauzule s právě jedním pozitivním literálem a alespoň jedním negativním literálem (např. $[s, \neg r, \neg t]$, $[r, \neg q]$),
- *cíle* – klauzule obsahující výhradně negativní literály (např. $[\neg r, \neg t]$).

V rámci SLD rezoluce může obecně existovat více způsobů, jak se dobrat výsledku. Kompaktní způsob, který zachycuje prohledávání stavového prostoru a způsoby vyvrácení množiny klauzulí, je SLD strom.

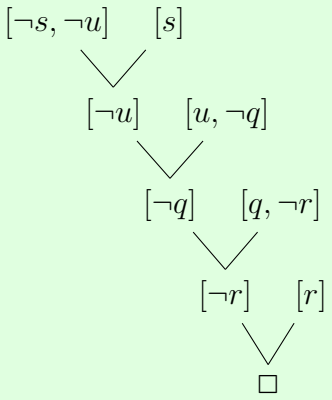
Příklad. Pomocí SLD rezoluce vyvraťte množinu $\{[\neg s, \neg u], [q, \neg r], [q, \neg t], [q], [r], [u, \neg q], [s]\}$. Nakreslete SLD strom systematického hledání řešení. Nakreslete rezoluční strom pro nejlevější úspěšnou větev SLD stromu.

Rezoluci budeme začínat cílem $[\neg s, \neg u]$. Pravidla a fakta seřadíme a očíslujeme:
1: $[q, \neg r]$ 2: $[q, \neg t]$ 3: $[q]$ 4: $[r]$ 5: $[u, \neg q]$ 6: $[s]$

SLD strom zachycuje všechna možná vyhodnocení. Použitá fakta a pravidla jsou naznačena číslem u přísluěné hrany SLD stromu. Strom se větví právě v uzlech, kde lze aplikovat více faktů nebo pravidel.



Každá (úspěšná) větev SLD stromu odpovídá jednomu rezolučnímu důkazu. Rezoluční strom pro nejlevější větev je uveden níže.



Příklad 7.3.1. SLD rezolucí vyvraťte množinu klauzulí $\{[s], [t, \neg s, \neg r], [r, \neg s]\} \cup \{[\neg t, \neg r]\}$.

Příklad 7.3.2. SLD rezolučním vyvrácením ukažte, že $\{q, (r \wedge t) \Rightarrow p, t \Rightarrow r, q \Rightarrow t\} \models p \wedge q$.

Příklad 7.3.3. Vytvořte SLD strom pro vyvrácení následující množiny uspořádaných klauzulí (čísla odpovídají prioritě při výběru daného předpokladu): $\{1 : [s], 2 : [p, \neg t], 3 : [p, \neg s], 4 : [r, \neg p], 5 : [r, \neg t]\} \cup \{[\neg r, \neg s]\}$.

Příklad 7.3.4. Vytvořte SLD strom pro vyvrácení následující množiny uspořádaných klauzulí (čísla odpovídají prioritě při výběru daného předpokladu): $\{1 : [t, \neg s], 2 : [s, \neg t], 3 : [s], 4 : [r]\} \cup \{[\neg t, \neg r]\}$.

7.4 Prolog ve výrokové logice

Programovací jazyk Prolog využívá SLD rezoluci k systematickému zjišťování, zda ze zadaných předpokladů plyne závěr. Nejprve se podíváme, jak vypadá program jazyka Prolog, a následně si ukážeme korespondenci mezi Prologem a SLD rezolucí.

Definice 43: *Program jazyka Prolog* je konečná posloupnost předpokladů $P_1, \ldots, P_n$, kde předpoklady jsou dvojího typu:

- *Fakta*, která jsou tvaru `p.`, kde p je výroková proměnná. Fakt `p.` je ekvivalentní formuli p.
- *Pravidla*, která jsou tvaru `p :- q, ..., r.`, kde $p, q, \ldots, r$ jsou výrokové proměnné. Pravidlo `p :- q, ..., r` je ekvivalentní formuli $p \Leftarrow (q \wedge \cdots \wedge r)$.

Nad zadaným programem jazyka Prolog se vyhodnocují dotazy. Dotaz je tvrzení, jehož vyplývání z předpokladů uvedených v programu chceme ověřit.

Definice 44: *Dotazy* jsou tvaru `?- p, ..., q.`, kde $p, \ldots, q$ jsou výrokové proměnné. Dotaz `?- p, ..., q.` je ekvivalentní formuli $p \wedge \cdots \wedge q$.

Ukažme si nyní korespondenci programu Prologu s Hornovými klauzulemi. Fakt `p.` lze přímočaře vyjádřit uspořádanou klauzulí $[p]$. Pravidlo tvaru `p :- q, r.`, ekvivalentní formuli $p \Leftarrow (q \wedge r)$, lze vyjádřit jako $p \vee \neg(q \wedge r)$, neboli $p \vee \neg q \vee \neg r$. To odpovídá uspořádané klauzuli $[p, \neg q, \neg r]$. Konečně chceme rezolucí dokázat, že z faktů a pravidel vyplývá závěr vyjádřený dotazem `?- s, r.` s významem $s \wedge r$. Pro důkaz rezolučním vyvracením přidáme k pravidlům a faktům negaci formule $s \wedge r$, tedy $\neg s \vee \neg r$, neboli uspořádanou klauzuli $[\neg s, \neg r]$. Uvedené klauzule odpovídají faktům, pravidlům a cíli tak, jak jsou definovány pro Hornovy klauzule (viz Definici 42).

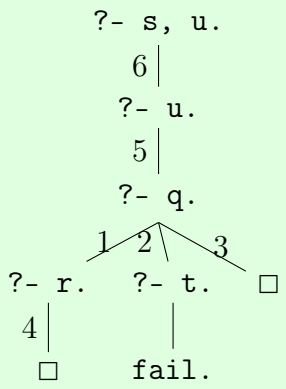
Díky přímé korespondenci s Hornovými klauzulemi lze pro programy Prologu vytvářet SLD rezoluční stromy. Zápis se liší v tom, že místo cíle tvaru $[\neg p, \ldots, \neg q]$ zapisujeme průběžně vyhodnocovaný dotaz ve tvaru `?- p, ..., q.`

Při zadání dotazu nad programem jazyka Prolog probíhá systematické hledání řešení, které odpovídá procházení SLD stromu do hloubky s prioritním výběrem pravidla nebo cíle uvedeného v programu dříve (tj. procházením alternativ zleva doprava). V případě vyvrácení zadaného cíle dojde k vytištění hodnoty `true.`

Příklad. Vytvořte SLD strom pro dotaz `?- s, u.` nad následujícím programem Prologu:

1. `q :- r.`
2. `q :- t.`
3. `q.`
4. `r.`
5. `u :- q.`
6. `s.`

Rezoluci začínáme dotazem `?- s, u.` Z pravidel a faktů programu při vyhodnocování vybíráme postupně shora dolů to, které odpovídá první proměnné dotazu. V případě faktu pak příslušnou proměnnou umažeme, v případě pravidla nahradíme pravou stranou za symbolem `:-`. Použitá fakta a pravidla jsou naznačena číslem u přísluěné hrany SLD stromu. Strom se větví právě v uzlech, kde lze aplikovat více faktů nebo pravidel. Prolog při vyhodnocování vždy vstupuje do nejprve do levých větví (tj. těch s nižším číslem použitého pravidla nebo faktu).



Úspěšné větve jsou zakončeny symbolem □, neúspěšné symbolem `fail.` V běžných implementacích Prologu se programy, které obsahují proměnnou nedefinovanou ve faktu nebo na levé straně nějakého pravidla (jako např. `t` v tomto programu), ani nespustí. V případě výrokové logiky existence takové proměnné přešně odpovídá existenci neúspěšné větve v SLD stromu.

Všimněte si, že ten samý SLD strom jsme již v jiném hávu viděli v ukázkovém příkladě v předchozí sekci o SLD rezoluci.

Vedle úspěšných a neúspěšných větví uvedených v příkladu může navíc SLD strom obsahovat i větve nekonečné, kdy Prolog cyklí.

Příklad 7.4.1. Uvažujte následující program Prologu:

1. `p :- q.`
2. `q :- t.`
3. `q :- r.`
4. `r.`
5. `s :- p, q, r.`

   a) Vytvořte SLD strom pro dotaz `?- s, r.`

   b) Kolik obsahuje SLD strom úspěšných, resp. neúspěšných větví?

   c) Upravte program tak, aby SLD strom neobsahoval neúspěšné větve.

Příklad 7.4.2. Uvažujte následující program Prologu:

1. `p :- q.`
2. `p.`
3. `q :- p.`

   a) Vytvořte SLD strom pro dotaz `?- p.` Jaký výsledek vrátí Prolog po zadání dotazu?

   b) Navrhněte uspořádání pravidel, aby se Prolog nad žádným dotazem nezacyklil.

   c) Kolika způsoby může Prolog v takto upraveném programu dojít k výsledku `true.`?

Příklad 7.4.3. Navrhněte program Prologu a dotaz, který obsahuje právě jednu úspěšnou větev, právě jednu nekonečnou větev a právě jednu neúspěšnou větev (nacházející se v SLD stromu v tomto pořadí zleva). *(Úkol za bonus `*1`: Nalezněte takový program o 3 řádcích.)*

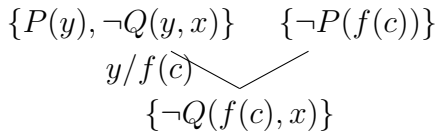
7.5 Obecná rezoluce v predikátové logice

Rezoluce v predikátové logice pracuje s formulemi ve Skolemově normální formě (viz kapitolu 6.3 Normální formy). Klauzule se opět zapisují jako množiny literálů, formule jako množiny klauzulí. Všeobecné kvantifikátory se v zápisu vynechávají. Formule $\forall x \forall y((P(y) \vee \neg Q(y,x)) \wedge \neg P(f(c)))$ se zapíše jako $\{\{P(y), \neg Q(y,x)\}, \{\neg P(f(c))\}\}$.

Co když budeme chtít rezolvovat klauzule $\{P(y), \neg Q(y,x)\}$ a $\{\neg P(f(c))\}$ na predikátu P? Povšimněte si, že jako argument predikátu P je v první klauzuli použit term y, v druhé klauzuli term $f(c)$. Abychom mohli na P rezolvovat, je nutné, aby obě klauzule „hovořily o stejném objektu“, takže je potřeba nejprve sjednotit (neboli *unifikovat*) použité termy.

V uvažovaném příkladě se nabízí nahrazení (neboli *substituce*) proměnné y termem $f(c)$, po kterém bude první klauzule tvaru $\{P(f(c)), \neg Q(f(c),x)\}$ – všimněte si náhrady y za $f(c)$ i v rámci predikátu Q. Jsme ovšem oprávněni takovou náhradu beztrestně provést? Podívejme se, jak vypadá původní klauzule v klasickém zápisu s kvantifikátory: $\forall x \forall y(P(y) \vee \neg Q(y,x))$ a jak vypadá klauzule po substituci: $\forall x(P(f(c)) \vee \neg Q(f(c),x))$. Lze konstatovat, že substituce proměnné za jiný term zachovala pravdivost. Platí-li totiž původní klauzule pro všechna y, pak platí pro libovolný term, kterým y nahradíme.

Jelikož substituce zachovává pravdivost, lze ji provádět v rámci aplikace rezolučního pravidla (a občas je to nevyhnutelné). Odvození rezolventy z výše uvedených klauzulí budeme tedy celkem zapisovat následovně:



Formalizaci obecné rezoluce v predikátové logice zahájíme u pojmu *substituce*.

Definice 45: *Substituce* je množina $\{x_1/t_1, \ldots, x_n/t_n\}$, kde x_i jsou různé proměnné a t_i jsou termy.

Speciálně jsou-li všechna t_i proměnné, hovoříme o *přejmenování proměnných*.

Substituce jsou například množiny $\{x/y, y/f(c)\}$, $\{y/f(x)\}$, $\{x/y, y/z, z/x\}$. Poslední uvedená je navíc přejmenováním proměnných.

Substituci $\Phi = \{x_1/t_1, \ldots, x_n/t_n\}$ lze aplikovat na množinu literálů S tak, že každý výskyt každé proměnné x_i nahradíme termem t_i. Aplikaci substituce zapisujeme jako $S\Phi$.

Příklad. Pro množiny literálů $S_1 = \{P(x, y), \neg Q(y, f(x))\}$, $S_2 = \{P(f(x)), P(y), P(f(f(c)))\}$ a substituce $\Phi_1 = \{x/c, y/b\}$, $\Phi_2 = \{x/f(c), y/f(f(c))\}$ stanovte $S_1\Phi_1$ a $S_2\Phi_2$.

$S_1\Phi_1 = \{P(c, b), \neg Q(b, f(c))\}$
$S_2\Phi_2 = \{P(f(f(c)))\}$.

Substituce Φ_2 ve vztahu k množině S_2 je v jistém smyslu speciální, všechny literály množiny S_2 totiž substitucí přešly na ten samý literál. Takovým substitucím říkáme *unifikátory*.

Definice 46: Substituce Φ je *unifikátorem* množiny S, pokud $|S\Phi| = 1$.

Existuje-li pro množinu unifikátor, hovoříme o *unifikovatelné* množině.

Unifikátorem množiny $\{P(z, f(z)), P(f(x), f(f(c)))\}$ je substituce $\{z/f(c), x/c\}$. Množina $\{P(z, f(z)), P(f(x), f(c))\}$ není unifikovatelná.

Unifikátory budou sloužit k dříve avizovanému sjednocení termů při aplikování rezolučního pravidla. To z technických důvodů aplikujeme na dvojice klauzulí, které nemají společné proměnné. (Pokud mají, nejprve proměnné přejmenujeme.)

Definice 47: *Rezoluční pravidlo pro predikátovou logiku:* Uvažujme dvě klauzule bez společných proměnných $\{P(\vec{t_1}), \ldots, P(\vec{t_r}), L_1, \ldots L_n\}$, $\{\neg P(\vec{t'_1}), \ldots, \neg P(\vec{t'_s}), L'_1, \ldots, L'_m\}$, kde L_i, resp. L'_i jsou literály (neobsahující predikát P), a substituci Φ, která je unifikátorem[5] množiny $\{P(\vec{t_1}), \ldots, P(\vec{t_r}), P(\vec{t'_1}), \ldots P(\vec{t'_s})\}$. Jejich *rezolventou* je klauzule $\{L_1, \ldots, L_n, L'_1, \ldots, L'_m\}\Phi$.

Rezolventou klauzulí $\{P(x), \neg Q(x)\}$ a $\{Q(c)\}$ je $\{P(c)\}$ (příslušný unifikátor $\Phi = \{x/c\}$). Rezolventou klauzulí $\{R(x,y), R(c,y), \neg Q(y)\}$ a $\{\neg R(z,d), P(z)\}$ je klauzule $\{\neg Q(d), P(c)\}$ (příslušný unifikátor $\Phi = \{x/c, y/d, z/c\}$).

Obecnou rezoluci v predikátové logice jinak provádíme stejným způsobem jako v logice výrokové. Pojmy rezolučního důkazu a rezolučního vyvrácení zůstávají nezměněny, pouze využívají rezolučního pravidla pro predikátovou logiku. Navíc na hrany do rezolučního stromu píšeme provedené substituce/unifikace.

Příklad 7.5.1. Nalezněte $S\Phi$ pro zadaná S a Φ.

a) $S = \{P(x), Q(y)\}$, $\Phi = \{x/y, y/x\}$

b) $S = \{P(c,x), \neg P(x, f(z))\}$, $\Phi = \{f(z)/x, x/d\}$

c) $S = \{P(x,y), P(c,z), P(c,c)\}$, $\Phi = \{x/y, y/z, z/c\}$

Příklad 7.5.2. Nalezněte (nejobecnější) unifikátory následujících množin (c, d jsou konstanty).

a) $S = \{P(x), Q(y)\}$

b) $S = \{P(x), P(y)\}$

c) $S = \{Q(x,x), Q(y,c)\}$

d) $S = \{Q(x,x), Q(y,c), Q(d,z)\}$

e) $S = \{P(x, f(x)), P(c,z), P(x, f(y))\}$

f) $S = \{P(x, f(x)), P(x,z), P(x, f(y))\}$

Příklad 7.5.3. Nalezněte rezolventy následujících dvojic klauzulí (c je konstanta).

a) $\{P(x), \neg Q(z)\}$, $\{\neg R(y), Q(y)\}$

b) $\{\neg P(c,x), Q(x)\}$, $\{P(y,y), \neg R(y,z)\}$

c) $\{Q(x), \neg P(c)\}$, $\{P(y), \neg R(y,x)\}$

d) $\{P(x, f(z)), P(y,y)\}$, $\{\neg P(c,z)\}$

e) $\{P(x, f(z)), P(y, y_1)\}$, $\{\neg P(c,z)\}$

f) $\{Q(f(f(f(z))), f(f(y)))\}$, $\{\neg Q(f(f(x)), x)\}$

Příklad 7.5.4. Uvažujte následující tvrzení:

- „Jablka nepadají daleko od stromu.“
- „Jím jenom jablka.“
- „Všechno padá daleko od stromu.“

Dokažte pomocí rezoluce, že z těchto tvrzení plyne „Nejím nic,“ a to dvěma způsoby: (1) přímým odvozením závěru a (2) vyvrácením množiny obsahující předpoklady a negaci závěru. Zejména v řešení

a) převeďte tvrzení do formální reprezentace,

b) určete význam (interpretaci) jednotlivých predikátů,

[5] V rámci definice rezolučního pravidla lze formulovat silnější požadavek na Φ – totiž aby se jednalo o *nejobecnější unifikátor*. Zde se tímto pojmem pro jednoduchost zabývat nebudeme.

c) proveďte skolemizaci jednotlivých formulí a

d) nakonec proveďte samotnou rezoluci.

7.6 Prolog v predikátové logice

Použití Prologu v predikátové logice je o něco zajímavější. Predikáty zapisujeme malými písmeny (podobně jako výrokové symboly), do závorek uvádíme termy. Proměnné se značí velkými písmeny: `X`, `Y`, `Z`. Syntax Prologu i tvorbu SLD stromů si demonstrujeme na příkladě.

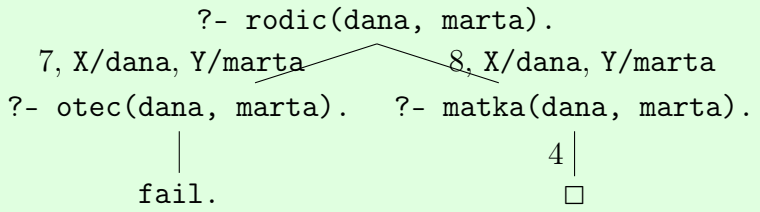
Příklad. Uvažujte následující program jazyka Prolog:

```
1: otec(petr, marta).
2: otec(jan, petr).
3: otec(petr, jana).
4: matka(dana, marta).
5: matka(jitka, petr).
6: matka(dana, kveta).
7: rodic(X, Y) :- otec(X, Y).
8: rodic(X, Y) :- matka(X, Y).
9: babicka(X, Y) :- matka(X, Z), rodic(Z, Y).
```

a) Zjistěte, zda je Dana rodičem Marty. Nakreslete SLD strom pro zadaný dotaz.

b) Zjistěte, koho je Jitka babičkou. Nakreslete SLD strom pro zadaný dotaz.

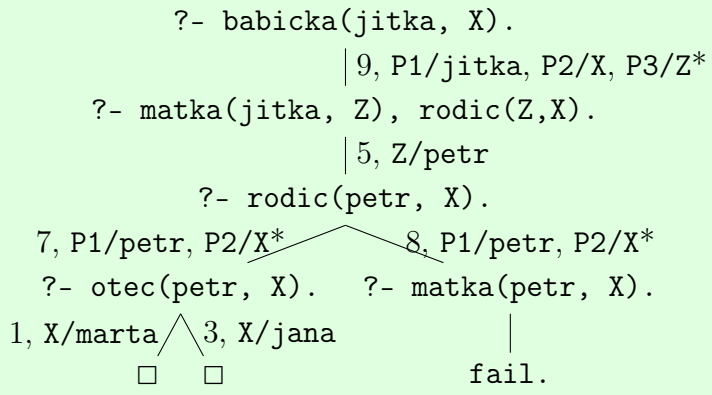
a) Zadaný dotaz je `?- rodic(dana, marta).` SLD strom vypadá takto:



Všimněte si, že při aplikaci pravidel 7 i 8 došlo k unifikaci proměnných `X`, `Y` s konstantami `dana`, `marta`. Úspěšná větev potvrzuje, že Dana je skutečně rodičem Marty.

Při zadání dotazu Prolog vypíše `true`. Protože nalezený výsledek je v nejpravější větvi, další prohledávání nebude umožněno.

b) Zadaný dotaz je `?- babicka(jitka, X).` SLD strom vypadá takto:



Proměnné `P1`, `P2` a `P3` u substitucí s hvězdičkou vznikly přejmenováních proměnných v pravidlech (kvůli konfliktu názvů). Použitá pravidla po přejmenování (v pořadí, v jakém byla použita) byla tvaru

9: `babicka(P1, P2) :- matka(P1, P3), rodic(P3, P2).`

7: `rodic(P1, P2) :- otec(P1, P2).`

8: `rodic(P1, P2) :- matka(P1, P2).`

Z SLD stromu lze vyčíst, že Jitka je babičkou Marty a Jany (dle substituce `X/marta`, resp. `X/jana` ve dvou úspěšných větvích SLD stromu).

Při zadání dotazu Prolog vypíše nejprve `X=marta`, po zadání středníku `X=jana`. Po opětovném zadání středníku vypíše `false` (zbytek stromu neobsahuje úspěšnou větev) a neumožní další prohledávání.

Příklad 7.6.1. Uvažujte následující program jazyka Prolog:
1: `p(f(f(X))) :- p(X).`
2: `p(f(0)).`

a) Vytvořte SLD strom pro dotaz `?- p(f(f(f(0)))).` Co vypíše Prolog po zadání dotazu?

b) Vytvořte SLD strom pro dotaz `?- p(f(f(f(f(0))))).` Co vypíše Prolog po zadání dotazu?

c) Vytvořte SLD strom pro dotaz `?- p(X).` Co vypíše Prolog po zadání dotazu?

d) Jak se bude Prolog chovat nad dotazem `?- p(X).`, prohodíme-li řádky 1 a 2?

Příklad 7.6.2. Uvažujte následující program jazyka Prolog:
1: `even(0).`
2: `even(f(f(X))) :- even(X).`
3: `odd(f(f(X))) :- odd(X).`
4: `odd(f(X)) :- even(X).`

Vytvořte SLD strom pro dotaz `?- odd(f(f(f(f(f(0)))))).`

Příklad 7.6.3. Uvažujte následující program jazyka Prolog:
1: `p(X) :- q(f(X)).`
2: `p(0).`
3: `q(X) :- p(X).`
Vytvořte SLD strom pro dotaz `?- p(0).`

Příklad 7.6.4. Uvažujte následující program jazyka Prolog:
1: `p(0,0).`
2: `p(f(X), Y) :- p(X, f(Y)).`
3: `p(X, f(f(0))) :- p(X, 0).`
Vytvořte SLD strom pro dotaz `?- p(f(f(f(f(0)))), 0).`

a) Bude první větev výpočtu úspěšná? Pokud ne, přeuspořádejte řádky v programu tak, aby byla úspěšná první větev výpočtu. Vytvořte SLD strom tohoto upraveného programu pro stejný dotaz.

b) Jak se přeuspořádání řádků projeví při vyhodnocování dotazu `?- p(X, 0).`?

8 Neklasické logiky

Neklasické logiky jsou logiky, které porušují principy *extenzionality* a *dvouhodnotovosti*, tedy principy, na nichž jsou založeny klasické logiky. Tyto principy klasické logiky různým způsobem omezují, například v oblasti reprezentace výrazů přirozeného jazyka. V této kapitole si představíme několik neklasických logik, které se mimo jiné snaží vypořádat právě s omezeními klasických logik.

8.1 Intenzionální logiky

V přirozeném jazyce má každý výraz svůj význam a svou referenci. Význam výrazu se nazývá *intenze* a naopak reference výrazu, tedy to, na co výraz odkazuje, se nazývá *extenze*. Intenze výrazu je neměnná, extenze se naopak může v čase a prostoru měnit.

Příklad. Pro výraz *pes* určete jeho:

a) intenzi,

b) extenzi.

a) Intenze slova pes je jeho význam, vyjadřuje, co znamená být psem. Může například hovořit o tom, jak musí entita vypadat, aby mohla být označena jako pes, a jaké vlastnosti má mít pes.

b) Extenze slova pes je to, na co výraz odkazuje. Tvoří ji všechny existující entity, které splňují podmínky bytí psem, tedy všichni psi.

Klasické logiky jsou extenzionální, což znamená, že pracují pouze s extenzemi výrazů a tvrzení. Naproti tomu *intenzionální logiky* pracují také s intenzemi výrazů.

Pro lepší pochopení rozdílu mezi intenzí a extenzí si ukážeme ještě jeden příklad.

Příklad. Určete extenzi a intenzi následujících výrazů.

a) Prezident ČR

b) Batman

c) Jednorožec

d) Země je kulatá.

e) Slunce je hvězda.

a) Extenze výrazu je osoba, která je v současné době prezidentem. Intenze výrazu je samotný význam výrazu, to, co znamená být prezidentem – prezidentská role.

b) Extenze výrazu je osoba Bruce Wayne. Intenze výrazu může být superhrdina, který bojuje za spravedlnost.

c) Tento výraz nemá extenzi; žádný konkrétní jednorožec neexistuje. Intenze vyjadřuje, že se jedná o bájného koně, který má uprostřed čela roh.

d) Extenze výroku je pravda. Intenze tvrzení je jeho samotný obsah – informace, kterou nese.

e) Extenze tvrzení je pravda. Intenze je informace o tom, že Slunce je hvězda.

Všimněte si, že extenze tvrzení je vždy jeho pravdivostní hodnota. Dále také platí, že zatímco extenzi můžeme určit poměrně přesně, intenze výrazu je abstraktnější.

Výraz ve větě interpretujeme buď jako jeho intenzi, nebo jako jeho extenzi. V prvním případě říkáme, že je použit v *intenzionálním kontextu*, v druhém případě říkáme, že je použit v *extenzionálním kontextu*.

V extenzionálním kontextu můžeme výraz v tvrzení nahradit jiným výrazem se stejnou extenzí, aniž by se změnila pravdivostní hodnota (extenze) celého tvrzení. Jelikož klasické logiky vždy předpokládají extenzionální kontext, dovolují při práci s výrazy tuto úpravu.

Problém nastává, je-li výraz v tvrzení ve skutečnosti použit v intenzionálním kontextu, ale v rámci klasické logiky s ním zacházíme jako s výrazem v kontextu extenzionálním. Po provedení substituce se tak může změnit pravdivostní hodnota celého výroku, což vede k nekonzistenci, protože klasická logika předpokládá, že hodnota zůstane nezměněna.

Příklad. Zvýrazněné výrazy nahraďte výrazy v závorkách a rozhodněte, zda je pravdivostní hodnota celého výroku zachována. Poté rozhodněte, zda jsou zvýrazněné výrazy použity v intenzionálním nebo extenzionálním kontextu.

a) Honza chce být **Batmanem**. (Bruce Wayne)

b) **Země je kulatá** a Merkur je měsíc. (Slunce je hvězda.)

c) Honza ví, že **Země je kulatá**. (Slunce je hvězda.)

a) Honza chce být Bruce Waynem.
Pravdivostní hodnota tvrzení není zachována. Honza chce být superhrdinou, jakého představuje Batman, ale ne boháčem Bruce Waynem. Jedná se tedy o intenzionální kontext.

b) Slunce je hvězda a Merkur je měsíc.
Pravdivostní hodnota tvrzení je zachována. Zachovala by se také, kdybychom tuto větu nahradili jakoukoli jinou větou, která je pravdivá. Jedná se tedy o extenzionální kontext.

c) Honza ví, že Sluce je hvězda.
Pravdivostní hodnota tvrzení není zachována. Honza může vědět, že Země je kulatá, a nevědět, že Slunce je hvězda. Jedná se o intenzionální kontext.

Skutečnost, že klasické logiky nerozlišují mezi extenzionálním a intenzionálním kontextem, je jedním z důvodů, proč nedokážou správně pracovat s přirozeným jazykem. Intenzionální logiky zavádějí formalismy, které pracovat s výrazy v intenzionálním kontextu umožňují, a díky tomu jsou schopny přirozený jazyk reprezentovat lépe. Mezi intenzionální logiky patří například TIL a modální logiky.

Příklad 8.1.1. Rozhodněte, zda jsou zvýrazněné výrazy použity v intenzionálním, nebo extenzionálním kontextu. (Hint: pokud se neumíte rozhodnout, zkuste výrazy substituovat a rozmyslet si, jestli se tím změní význam věty.)

a) **Zmrzlinu** znali už ve starověkém Řecku.
b) Michalovi spadla **zmrzlina** na zem.
c) Albert chce být **běžcem**.
d) **Běžec** doběhl do cíle.
e) **Sousedův pes** umí udělat salto.
f) Michal by chtěl mít **psa**.
g) Dáša řekla, že **na Marsu žijí Marťané**.
h) Pokud **na Marsu žijí Marťané**, pak tam jsou budovy.
i) Policie zadržela **hledaného žháře**.

Příklad 8.1.2. Použijte uvedená slova ve dvou větách – v intenzionálním a extenzionálním kontextu.

a) stromy
b) vítěz turnaje
c) tužka

8.2 Možné světy

Svět v logice představuje určitý ohraničený prostor, nad kterým s logikou pracujeme. Stav, ve kterém se daný svět může nacházet, nazýváme *možný svět* a všechny možné stavy reprezentuje *množina možných světů*. Mezi možnými světy je jeden *aktuální svět*, který odpovídá skutečnému stavu světa. Pomocí možných světů zavádíme sémantiku modálních logik.

Možné světy mohou být propojeny pomocí *relace dostupnosti*.

Definice 48: *Relace dostupnosti* je binární relace $S \subseteq W^2$ na množině možných světů W. Říkáme, že *svět v je dostupný ze světa w* právě tehdy, když platí $(w, v) \in S$.

Obecně platí, že relace dostupnosti představuje určitou formu možnosti. Za možné světy ve světě $w \in W$ považujeme pouze ty světy, které jsou z w dostupné.

Možným světům a vztahům přístupnosti přiřazujeme konkrétnější význam v závislosti na modální logice, v níž jsou použity. Například v temporální logice představují možné světy okamžiky v čase a dostupnost mezi světy představuje změnu stavu světa v čase.

Příklad. Uvažte svět, který obsahuje pouze výroky p, q, r, a v němž platí $p \Rightarrow q$.

a) Kolik takových možných světů existuje?

b) Přidejme podmínku, že ve světě platí také $p \vee \neg r$. Kolik možných světů existuje v takovém případě?

a) 6 možných světů. Světy $p, \neg q, r$ a $p, \neg q, \neg r$ nejsou možné.

b) 4 možné světy. Z další podmínky vyplývá, že dva světy, v nichž platí $\neg p \wedge r$, také nejsou možné.

Příklad 8.2.1. Dáša má dvě rostliny, fialku a zelenec, které zalévá vždy současně. Zelenec uvadne 4 až 6 dní po zalití. Fialka uvadne 10 až 15 dní po zalití. Uvažme zjednodušený svět, který obsahuje pouze tyto dvě rostliny, a v němž je stav rostliny binární – buď je uvadlá, nebo není.

a) Uveďte, kolik existuje možných světů.

b) Dáša řekla, že rostliny naposledy zalévala před dvěma týdny. Uveďte, kolik možných světů existuje, pokud tuto informaci znáte, a které to jsou.

Příklad 8.2.2. V bufetu se prodává káva, bagety a croissanty. Ne vždy ale mají kompletní nabídku. Mějme svět, který reprezentuje stav bufetové nabídky.

a) Uveďte, kolik existuje možných světů.

b) Zjistíte, že v bufetu mají vždy croissanty nebo bagety, ale nemají je současně. Uveďte, kolik možných světů existuje, pokud tuto informaci znáte, a které to jsou.

Příklad 8.2.3. Do bufetu z předchozího zadání (neberte v úvahu omezení zavedené v bodě b)) přijde student. Chce si koupit kávu, nebo, pokud ji nemají, cokoli jiného. Jak se změní stav bufetu po jeho odchodu? Předpokládejte, že student si koupí pouze jednu věc a káva se může vyprodat také.

Možné stavy bufetu reprezentujte pomocí možných světů. Změnu stavu bufetu reprezentujte pomocí relace dostupnosti. Možné světy a relaci graficky znázorněte.

8.3 Modální logiky a K-logika

Modální logiky jsou intenzionální logiky, které se zabývají různými modalitami neboli módy pravdy. Mezi tyto modality patří:

- *Aletická* modalita – vyjadřuje nutnost – nutně platí, možná platí.
- *Temporální* modalita – vyjadřuje platnost v určitém časovém okamžiku – vždy platilo, bude platit.
- *Deontická* modalita – vyjadřuje, jak by věci měly být – je povinné, je povoleno.
- *Epistemická* modalita – vyjadřuje znalost a přesvědčení – Albert ví, že platí, Albert věří, že platí.

Naproti tomu v klasické logice uvažujeme pouze o tom, zda je tvrzení pravdivé nebo nepravdivé.

Modality reprezentujeme pomocí *modálních operátorů*. Ty se od logických operátorů liší tím, že pravdivostní hodnota na jejich výstupu závisí nejen na pravdivostní hodnotě vstupních tvrzení (extenzi), ale také na významu samotných tvrzení (intenzi). Jelikož klasické logiky nepracují s intenzemi výrazů, modality správně reprezentovat nedokážou.

K-logika je formální systém pracující s modalitou nutnosti, kterou reprezentuje pomocí modálních operátorů *nutně* $\Box$ a *možná* $\Diamond$. K-logika je navíc základní modální logika – její úpravou můžeme zavést logiky dalších modalit.

V této kapitole zavádíme K-logiku s využitím výrokové logiky. Nejprve rozšíříme abecedu výrokové logiky o symboly modálních operátorů:

Definice 49: *Abeceda K-logiky* zahrnuje:
- abecedu výrokové logiky,
- symboly $\Box, \Diamond$.

Do syntaxe výrokové logiky následně přidáme pravidlo pro tvorbu formulí, které obsahují modální operátory:

Definice 50: Je-li φ formule, pak také $(\Box\varphi)$ a $(\Diamond\varphi)$ jsou formule.

Dále zavedeme *sémantiku K-logiky*. Interpretace K-logiky, nazývaná *Kripkeho interpretace* nebo *Kripkeho rámec*, nepředstavuje jediný stav světa (jako interpretace v klasických logikách), ale více stavů. Každý stav je reprezentován jako možný svět a odpovídající interpretace elementárních výroků.

Definice 51: *Kripkeho interpretace* (též *Kripkeho rámec*) pro jazyk L je struktura C zahrnující:
- množinu možných světů W,
- relaci dostupnosti $S \subseteq W^2$,

- interpretaci výrokových proměnných I_w, pro každý svět $w \in W$.

Relace dostupnosti pro každý možný svět w určuje, které světy jsou z něj dostupné. Dostupné světy ze světa w chápeme jako světy, které jsou ve světě w možné. Interpretace I_w pro každý možný svět w určuje, které elementární výroky v něm jsou nebo nejsou pravdivé.

Jelikož různé světy vyhodnocují elementární výroky různě, i složitější formule může v různých světech nabývat různých pravdivostních hodnot.

Definice 52: Zda formule φ *platí ve světě* $w \in W$, píšeme $w \models \varphi$, definujeme následovně:

- $w \models p$, právě když atomická formule p je pravdivá v I_w,
- $w \models (\varphi \Rightarrow \psi)$, právě když neplatí $w \models \varphi$ nebo platí $w \models \psi$ (a podobně pro ostatní logické spojky),
- $w \models \Box\varphi$, právě když $v \models \varphi$ pro všechny světy $v \in W$ takové, že $(w, v) \in S$,
- $w \models \Diamond\varphi$, právě když existuje svět $v \in W$ takový, že $(w, v) \in S$ a $v \models \varphi$.

Je-li formule atomická, platí ve světě w právě tehdy, když je pravdivá v I_w. V případě logických spojek zůstává sémantika stejná jako ve výrokové logice.

Formule φ je *nutně pravdivá* ve světě w, píšeme $\Box\varphi$, právě když je pravdivá ve všech světech dostupných z w. Formule φ *možná platí* ve světě w, píšeme $\Diamond\varphi$, právě když platí alespoň v jednom světě dostupném ze světa w.

Všimněte si, že sémantika modálních operátorů $\Box$ a $\Diamond$ je podobná sémantice kvantifikátorů $\forall$ a $\exists$. Stejně jako u nich platí i u modálních operátorů, že jeden lze odvodit z druhého, tedy: $\Box\varphi \Leftrightarrow \neg\Diamond\neg\varphi$.

Nakonec zbývá definovat pravdivost v interpretaci a logickou pravdivost.

Definice 53:

- Formule φ je *pravdivá v Kripkeho interpretaci* C, píšeme $\models_C \varphi$, jestliže platí ve všech světech $w \in W$.
- Formule φ je *logicky pravdivá (tautologie)*, jestliže platí ve všech Kripkeho interpretacích.

Vyhodnocení K-logické formule v Kripkeho interpretaci si ukážeme na příkladu.

Příklad. Uvažte Kripkeho rámec C, kde $W = \{w_1, w_2, w_3\}$, $S = \{(w_1, w_2), (w_1, w_3), (w_2, w_2)\}$ a elementární výroky jsou p, q. Ve světech platí následující tvrzení:

- $w_1 : p, q$,

- $w_2 : p$,
- $w_3 : q$.

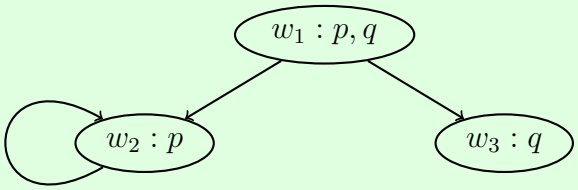
Pomocí grafu znázorněte možné světy a relaci dostupnosti. Pak rozhodněte, zda je formule $\Diamond\Box(\neg q \vee p)$:

a) platná ve světě w_1,

b) platná v Kripkeho interpretaci C,

c) logicky pravdivá.

Grafické znázornění možných světů a vztahů mezi nimi může vypadat následovně:



Nejprve vytvoříme pravdivostní tabulku, ve které vyhodnotíme formuli pro všechny světy. Ta je užitečná i v případě, že máme vyhodnotit platnost formule pouze v jednom světě. Chceme-li například rozhodnout, zda $\Box\Diamond\varphi$ platí v nějakém světě, musíme nejdříve vyhodnotit formuli $\Diamond\varphi$ pro všechny světy, které jsou z tohoto světa dostupné.

	w_1	w_2	w_3
p	1	1	0
q	1	0	1
$\neg q$	0	1	0
$\neg q \vee p$	1	1	0
$\Box(\neg q \vee p)$	0	1	1
$\Diamond\Box(\neg q \vee p)$	1	1	0

Pomocí tabulky pak odpovíme na otázky ze zadání:

a) Ano, formule $\Diamond\Box(\neg q \vee p)$ platí ve světě w_1.

b) Ne, formule neplatí v interpretaci C, protože neplatí ve světě w_3.

c) Ne, formule není logicky pravdivá, protože neplatí v interpretaci C.

Příklad 8.3.1. Určete modalitu věty.

a) Včera pršelo.

b) Hana je přesvědčena, že káva není zdravá.

c) Možná má můj pes chřipku.

d) Kočky nevědí, že Merkur je planeta.

e) Na tomto místě se nesmí skákat do vody.

Příklad 8.3.2. Pomocí K-logiky formálně zapište následující věty:

a) Možná prší nebo možná svítí slunce.

b) Možná prší a svítí slunce zároveň.

c) Pokud je duha, pak nutně prší a svítí slunce zároveň.

Příklad 8.3.3. Uvažte Kripkeho rámec C, kde $W = \{w_1, w_2, w_3\}$, $S = \{(w_1, w_3), (w_2, w_2), (w_2, w_1), (w_2, w_3)\}$ a elementární výroky jsou p, q. Ve světech platí následující tvrzení:

- $w_1 : q$,
- $w_2 : \neg p$,
- $w_3 : q, p$.

Rozhodněte, zda je formule $\Box(\neg\Diamond q \vee p) \wedge (p \Rightarrow \Box p)$:

a) platná ve světě w_1,

b) platná v Kripkeho interpretaci C,

c) logicky pravdivá.

Příklad 8.3.4. Uvažte Kripkeho rámec C, kde $W = \{w_1, w_2, w_3\}, S = \{(w_1, w_1), (w_1, w_2), (w_2, w_1), (w_3, w_2)\}$ a elementární výroky jsou p, q, r. Ve světech platí následující tvrzení:

- $w_1 : p$,
- $w_2 : q$,
- $w_3 : p, r$.

Bez použití pravdivostní tabulky rozhodněte, zda v dané interpretaci platí následující formule:

a) $\Box p$

b) $\neg\Diamond\neg q$

c) $\neg\Diamond r$

d) $p \vee \Diamond p$

e) $\Box r \Rightarrow \Diamond r$

Příklad 8.3.5. Uvažte Kripkeho rámec C, kde $W = \{w_1, w_2, w_3, w_4\}$, $S = \{(w_2, w_2), (w_3, w_4), (w_4, w_3), (w_4, w_1)\}$, a elementární výrok p, který platí ve světech w_2 a w_3. Pomocí pravdivostní tabulky určete, ve kterých světech platí následující formule:

a) $\Box\neg\Box p$

b) $\Box\Diamond p$

c) $\Diamond\Diamond\Diamond p$

Příklad 8.3.6. Znegujte následující formule a upravte je tak, aby se symboly negace nacházely pouze přímo před elementárními výroky:

a) $\Diamond((\neg p \Rightarrow \Diamond q) \vee \Box\neg q)$

b) $(\Box\neg p \wedge q) \vee \Diamond\neg p$

c) $\Box\Diamond(p \Rightarrow \Diamond\Box q)$

Příklad 8.3.7. Pro každou následujících formulí rozhodněte, zda je tautologie. Pokud ne, určete nějakou podmínku relace dostupnosti tak, aby formule platila ve všech interpretacích, které podmínku splňují.

a) $\Box p \Rightarrow p$

b) $\Box(p \Rightarrow q) \Rightarrow (\Box p \Rightarrow \Box q)$

c) $\Box p \Rightarrow \Diamond p$

d) $p \Rightarrow \Box\Diamond p$

Příklad 8.3.8. Uvažte Kripkeho rámec C, kde $W = \{w_1, w_2, w_3\}$ a elementární výroky jsou p, q, r. Ve světech platí následující tvrzení:

- $w_1 : p$,
- $w_2 : q$,
- $w_3 : p, r$.

Doplňte relaci dostupnosti Kripkeho rámce C tak, aby v ní platila formule:

a) $\Box p \Rightarrow r$

b) $\Box(q \wedge r)$

c) $\Diamond\Box p$

8.4 Vícehodnotové logiky

Vícehodnotové logiky jsou podobné klasickým logikám v tom, že pravdivostní hodnota celého výrazu se skládá z pravdivostních hodnot dílčích výrazů, jsou tedy extenzionální. Nicméně se od nich liší tím, že pracují s více než dvěma pravdivostními hodnotami. Pravdivostní hodnoty se většinou pohybují v intervalu od 0 do 1, přičemž 0 obecně představuje absolutní nepravdu a 1 absolutní pravdu, avšak konkrétní interpretace závisí na oblasti použití logiky.

Mezi vícehodnotové logiky patří tříhodnotová *Lukasiewiczova logika*. Kromě pravdivostních hodnot 0 (nepravda) a 1 (pravda) zavádí hodnotu 1/2 (nevím). Následující definice je obdobou definice interpretace z klasické výrokové logiky.

Definice 54: *Interpretace I* v Lukasiewiczově logice je zobrazení $I : \mathcal{P} \to \{0, 1/2, 1\}$ přiřazující pravdivostní hodnoty 0 (nepravda), 1 (pravda) a 1/2 (nevím) jednotlivým výrokovým proměnným množiny $\mathcal{P}$.

Kvůli této změně zavádí Lukasiewiczova logika také novou *valuační funkci val* (obdobu valuace), která umí pracovat s hodnotou 1/2 a zachovává sémantiku logických spojek.

Definice 55: *Valuační funkce val_I* příslušící interpretaci I je definována indukcí ke struktuře formule:

- $val_I(p) = I(p)$, pro atomickou formuli $p \in \mathcal{P}$,

- $val_I(\neg\varphi) = 1 - val_I(\varphi)$,
- $val_I(\varphi \wedge \psi) = min(val_I(\varphi), val_I(\psi))$,
- $val_I(\varphi \vee \psi) = max(val_I(\varphi), val_I(\psi))$,
- $val_I(\varphi \Rightarrow \psi) = min(1, 1 - val_I(\varphi) + val_I(\psi))$.

V Lukasiewiczově logice nelze implikaci obecně vyjádřit negací a disjunkcí, protože rovnost $val(p \Rightarrow q) = val(\neg p \vee q)$ neplatí vždy. V případě $val(p) = 1/2$, $val(q) = 1/2$ totiž $val(p \Rightarrow q) = 1$, ale $val(\neg p \vee q) = 1/2$.

V Lukasiewiczově logice je *modelem formule* každá interpretace, v níž je hodnota formule větší než 0.

Definice 56: *Modelem formule* φ v Lukasiewiczově logice je interpretace I taková, že pro jí příslušící valuační funkci platí $val_I(\varphi) > 0$.

Příklad. Uvažte formuli $p \Rightarrow \neg(q \vee p)$ Lukasiewiczovy logiky. Rozhodněte, zda je interpretace $I(p) = 1/2$, $I(q) = 0$ modelem formule.

Ano, daná interpretace je modelem formule, protože $val_I(p \Rightarrow \neg(q \vee p)) = 1$. Formuli vyhodnotíme postupně podle definice funkce val_I. Mezivýpočty jsou uvedeny v tabulce.

p	$\Rightarrow$	$\neg$	$(q$	$\vee$	$p)$
$1/2$	1	$1/2$	0	$1/2$	$1/2$

Další vícehodnotovou logikou je *fuzzy logika*, která umožňuje vyjádřit míru pravdivosti nejednoznačných výroků, jako je například výrok Petr je mladý. Fuzzy logika nepracuje s diskrétními pravdivostními hodnotami, ale s intervalem hodnot $[0, 1]$.

Definice 57: *Interpretace* I ve fuzzy logice je zobrazení $I : \mathcal{P} \to [0, 1]$ přiřazující pravdivostní hodnoty z intervalu $[0, 1]$ jednotlivým výrokovým proměnným množiny $\mathcal{P}$.

Model formule a valuační funkce jsou definovány stejně jako v Lukasiewiczově logice s tím rozdílem, že valuační funkce pracuje nad spojitou doménou.

Definice 58: *Modelem formule* φ ve fuzzy logice je interpretace I taková, že pro jí příslušící valuační funkci platí $val_I(\varphi) > 0$.

Příklad. Uvažte formuli $(p \vee \neg q) \wedge \neg p$ fuzzy logiky. Rozhodněte, zda je interpretace $I(p) = 0.2$, $I(q) = 0.9$ modelem formule.

Ano, daná interpretace je modelem formule, protože $val_I((p \vee \neg q) \wedge \neg p) = 0.2$. Formuli vyhodnotíme postupně podle definice funkce val_I. Mezivýpočty jsou uvedeny v tabulce.

$(p$	$\vee$	$\neg$	$q)$	$\wedge$	$\neg$	p
0.2	0.2	0.1	0.9	0.2	0.8	0.2

Příklad 8.4.1. Uvažte interpretaci Lukasiewiczovy logiky $I(p) = 1/2$, $I(q) = 0$, $I(r) = 1/2$. Rozhodněte, zda je modelem následujících formulí:

a) $\neg(p \wedge r)$

b) $p \Rightarrow (q \vee \neg r)$

c) $\neg(r \Rightarrow q)$

d) $(r \vee \neg r) \wedge q$

Příklad 8.4.2. Uvažte formuli $(p \Rightarrow q) \wedge (p \vee \neg q)$ Lukasiewiczovy logiky. Vytvořte pravdivostní tabulku, která zahrnuje pouze interpretace obsahující hodnotu $1/2$, a rozhodněte, které z nich jsou modelem formule.

Příklad 8.4.3. Uvažte interpretaci fuzzy logiky $I(p) = 0.3$, $I(q) = 0.6$, $I(r) = 0.2$. Rozhodněte, zda je modelem následujících formulí:

a) $(\neg p \wedge q) \vee r$

b) $q \Rightarrow \neg r$

c) $\neg r \Rightarrow p$

d) $(r \wedge \neg q) \vee \neg(q \wedge p)$

Příklad 8.4.4. Uvažte formuli $(p \Rightarrow q) \wedge (p \wedge \neg q)$ fuzzy logiky. Nalezněte interpretaci, ve které je valuace formule větší než 0.5.

Příklad 8.4.5. Nalezněte interpretaci fuzzy logiky, která je modelem všech následujících formulí:

- $p \Rightarrow r$
- $\neg q$
- $q \vee \neg r$

Příklad 8.4.6. Pro každou z následujících formulí uveďte interpretaci fuzzy logiky, která není její modelem.

a) $\neg p$

b) $p \wedge \neg q$

c) $(p \vee q) \Rightarrow r$

Příklad 8.4.7. Uvažte interpretaci $I(p) = 0$, $I(q) = 0.5$, $I(r) = x$. Nalezněte množinu všech hodnot x takových, že valuace formule $\neg(q \wedge \neg r) \vee (r \Rightarrow p)$ fuzzy logiky se rovná hodnotě:

a) 0

b) 0.5

c) 0.8

d) 1

9 Reprezentace a vyvozování znalostí

K tomu, aby inteligentní systém správně fungoval, mu potřebujeme dodat dostatek informací o světě, ve kterém pracuje, a naučit ho, jak tyto informace využívat. Oblast *reprezentace znalostí* se zabývá tím, jak znalosti vyjádřit ve formě, jež je vhodná pro počítačové zpracování, zatímco oblast *vyvozování znalostí* se zabývá tím, jak s uloženými znalostmi pracovat a odvozovat z nich znalosti nové. V této kapitole si několik přístupů k reprezentaci a vyvozování znalostí představíme.

Důležitou součástí reprezentování znalostí je organizování *objektů* do *kategorií* neboli tříd. Ve fyzické realitě sice přicházíme do kontaktu hlavně s objekty, ale na úrovni myšlení a vyvozování pracujeme ve velkém s kategoriemi. Například pokud si někdo chce koupit stejnou klávesnici, jako má jeho kamarád, nechce ten samý objekt, ale model klávesnice. Stejně tak zařazení objektu do kategorie nám často umožní odvodit o něm nové znalosti. Například vidíme-li v sekci ovoce v obchodě zelený, velký a kulatý objekt, zařadíme jej do kategorie meloun, z čehož můžeme následně odvodit informaci, že vevnitř má červenou dužinu.

V predikátové logice bychom mohli kategorie reprezentovat jako predikát, ale při tomto přístupu nedokážeme správně reprezentovat vlastnosti třídy. Kategorii proto reprezentujeme jako konstantu. Příslušnost do kategorie, která je reprezentována konstantou ověříme použitím predikátu, například *Member(m, meloun)*, jenž se vyhodnotí jako pravdivý, pokud je m instancí kategorie meloun. Takové modelování abstrakce jako objektu se nazývá *reifikace* neboli zvěcnění.

Objekty a třídy organizujeme do *hierarchie tříd*, třída může být *podtřídou* či nadtřídou jiné třídy, objekt je *instancí* nějaké třídy. Kromě hierarchie tříd se budeme zbývat i tím, jak reprezentovat další znalosti o pojmech, ty nazýváme i *fakta*, týkají-li se objektů a *pravidla*, týkají-li se tříd.

9.1 Ontologie

Ontologie je formální systém, který popisuje svět, případně jeho část, skrze pojmy, vztahy mezi pojmy a axiomy. S použitím ontologie můžeme potom vyjádřit konkrétní znalosti o daném světě. Ontologie můžeme zavést pomocí různých formalismů, v této sbírce budeme používat predikátovou logiku.

Ontologie dělíme na doménové a všeobecné (upper ontologies). Všeobecná ontologie se snaží popsat celý svět a poskytnout tak nástroj na reprezentování znalostí z libovolné oblasti. Naproti tomu doménová ontologie je specifičtější například v zavedených pojmech a uzpůsobená pro reprezentaci nějaké konkrétní domény.

Jednoduchou ontologii si ukážeme na příkladu.

Příklad. Aby student získal zápočet, potřebuje získat alespoň 10 bodů z domácích úkolů a 15 bodů z testu. Zaveďte ontologii, která popisuje uvedenou situaci.

Nejprve pomocí predikátové logiky zavedeme jazyk ontologie, tj. všechny pojmy, které jsou nezbytné k popisu situace. Můžeme používat konstanty, funkce a predikáty.

- Úkoly(s, b): predikát, pravdivý, pokud má student s z domácích úkolů b bodů.
- Test(s, b): predikát, pravdivý, pokud má student s z testu b bodů.
- Zápočet(s): predikát, pravdivý, pokud byl studentovi s udělen zápočet.
- $+$, $\leq$: funkce a predikát, oba se standardní sémantikou.
- 10, 15: konstanty, reprezentují minimální nutný počet bodů.

Dále zavedeme obecně platné axiomy:

- A1: $\forall s\Big(\text{Zápočet}(s) \iff \exists b\big(10 \leq b \wedge \text{Úlohy}(s, b)\big) \wedge \exists b\big(15 \leq b \wedge \text{Test}(s, b)\big)\Big)$

V dalším příkladu si ukážeme, jak lze ontologii použít k reprezentaci konkrétní situace a odvození znalostí.

Příklad. Albert získal 12 bodů za domácí úkoly a 20 bodů z testu. Pomocí ontologie z předchozího příkladu reprezentujte situaci a odvoďte, zda Albert dostane zápočet.

Nejprve zavedeme tvrzení specifická pro danou situaci:

- Úlohy(Albert, 12)
- Test(Albert, 20)

Z axiomu A1 potom plyne, že platí predikát Zápočet(Albert), a tudíž Albert zápočet získal.

Příklad 9.1.1. Uvažujme následující neúplný jazyk ontologie, který reprezentuje hru piškvorky:

- Player(p): predikát, p je hráč
- Mark(m): predikát, m je značka
- Square(q): predikát, q je pole
- $p_\times, p_\bigcirc$: konstanty, hráči „křížek“, resp. „kolečko“
- $q_{11}, q_{12}, ..., q_{33}$: konstanty, pole herního plánu
- s_0: konstanta, počáteční situace, situace reprezentuje stav hry
- WinningPosition(q_1, q_2, q_3): predikát, pravdivý pro pole, která tvoří výherní pozici
- Opponent(p): funkce, reprezentuje protihráče hráče p
- Wins(p, s): predikát, pravdivý, pokud v situaci s vyhrál hráč p
- Play(p, q): funkce, reprezentuje akci, že hráč p označí pole q

- Poss(a, s): predikát, je pravdivý, pokud je možné vykonat akci a v situaci s
- Result(a, s): funkce, reprezentuje situaci, jež nastane po provedení akce a v situaci s

Jazyk doplňte o pojmy, které budou reprezentovat:

a) značky, které mohou být umisťovány na pole,

b) informaci o tom, která značka patří zadanému hráči,

c) informaci o tom, který hráč je na tahu v zadané situaci,

d) informaci o tom, která značka je umístěna na zadaném poli.

Příklad 9.1.2. V ontologii vytvořené v předchozím příkladu chybí axiomy, které by určovaly sémantiku zavedených predikátů a funkcí. Doplňte ontologii o následující axiomy:

a) Nikdo jiný kromě $p_\times$ a $p_\bigcirc$ není hráčem.

b) Oponent hráče $p_\times$ je $p_\bigcirc$.

c) Značka hráče $p_\bigcirc$ je $\bigcirc$.

d) Vítězné pozice se skládají pouze z polí, která se nacházejí v jednom řádku, sloupci nebo diagonále (formuli nepište celou, repetitivní části můžete vynechat).

e) Axiom určující, kdy platí Win(p, s).

f) Axiom určující, kdy platí Poss(a, s).

Příklad 9.1.3. Využijte nástroj, který pro slovo přirozeného jazyka najde odpovídající pojmy ontologie SUMO (pro více informací o nalezených pojmech je rozklikněte) a určete:

a) pojmy související se slovem *coffee*,

b) pojem p, který reprezentuje zimní roční období,

c) s čím je pojem p ekvivalentní,

d) pojem, jehož podtřídou je pojem p.

e) Nalezněte výraz, který obsahuje pojem p a pojem MediterraneanClimateZone, a určete jeho význam.

Příklad 9.1.4. Určete význam následujících výrazů ontologie SUMO. Pokud neumíte odvodit význam nějakého pojmu, můžete si ho vyhledat zde.

a) (⟹
   (*attribute* ?AREA *MountainousTerrain*)
   (*exists*(?MTN)
      (*and*
         (*instance* ?MTN *Mountain*)
         (*located* ?MTN ?AREA))))

b) (⟹
   (*and*
      (*instance* ?SMILE *Smiling*)

(*agent* ?SMILE ?AGENT))
(*holdsDuring*
(*WhenFn* ?SMILE)
(*attribute* ?AGENT *Happiness*)))

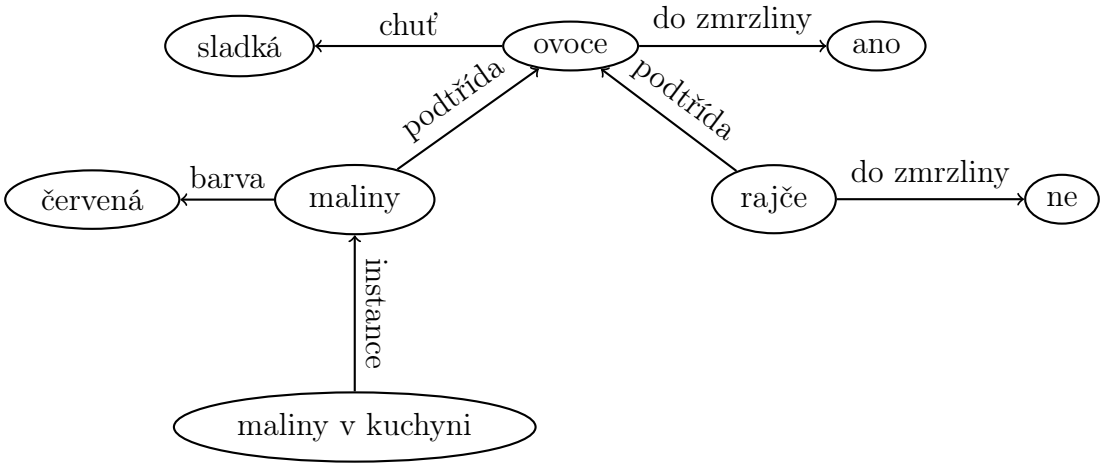
c) (⟹
(*instance* ?O *OlympicGames*)
(*or*
(*exists*(?W)
(*and*
(*instance* ?W *WinterSeason*)
(*temporalPart*
(*WhenFn* ?O) ?W)))
(*exists*(?S)
(*and*
(*instance* ?S *SummerSeason*)
(*temporalPart*
(*WhenFn* ?O) ?S)))))

9.2 Sémantické sítě a rámce

Sémantické sítě reprezentují pojmy a vztahy mezi pojmy. Mají formu grafu, přičemž uzly grafu představují pojmy a hrany vztahy. Nejdůležitější vztahy jsou *instance* a *podtřída*, které tvoří hierarchii pojmů. Další vztahy reprezentují *vlastnosti* pojmů, například *Barva(beruška, červená)*, nebo příslušnost části k celku, *Část(noha, kráva)*. Vztahy, jež souvisí s hierarchií pojmů zapisujeme ve vertikálním směru, ostatní vztahy zapisujeme horizontálně.

Sémantické sítě podporují dědičnost a mechanismus *vzoru* a *výjimky*. Pojmy dědí vzorové hodnoty vlastností z nadtříd, přičemž vždy platí ta hodnota, která je k pojmu nejblíže. Vzorovou hodnotu může přepsat výjimka uvedená u uvažovaného pojmu. Princip dědičnosti neplatí nutně v horizontálních vztazích. Například noha, jako část krávy, zdědí barvu krávy, ale nedědí oblíbené jídlo krávy.

Příklad. Uvažte následující sémantickou síť.



a) Je rajče sladké?

b) Je rajče vhodné do zmrzliny?

c) Jaké znalosti lze vyvodit ze sítě o pojmu *maliny v kuchyni*?

a) Ano.

b) Ne.

c) Maliny v kuchyni jsou instancí kategorie maliny, jsou ovoce, mají červenou barvu, jsou sladké a vhodné do zmrzliny.

Rámec je univerzální struktura, v níž jsou uloženy veškeré relevantní informace o pojmech. Stejně jako sémantické sítě, rámce reprezentují taxonomii pojmů, jejich vlastnosti, avšak mají textovou podobu. Pojem je v rámci reprezentovaný pomocí následujících hodnot:

- *objekt* – samotný pojem
- *sloty* – vztahy daného pojmu
- *hodnoty slotů* - pojmy, které jsou v uvedeném vztahu s daným pojmem.

Rámce též podporují dědičnost a výjimky. Hvězdičkou jsou označeny sloty, které nesou vzorové hodnoty a mohou být při dědění přepsány výjimkou.

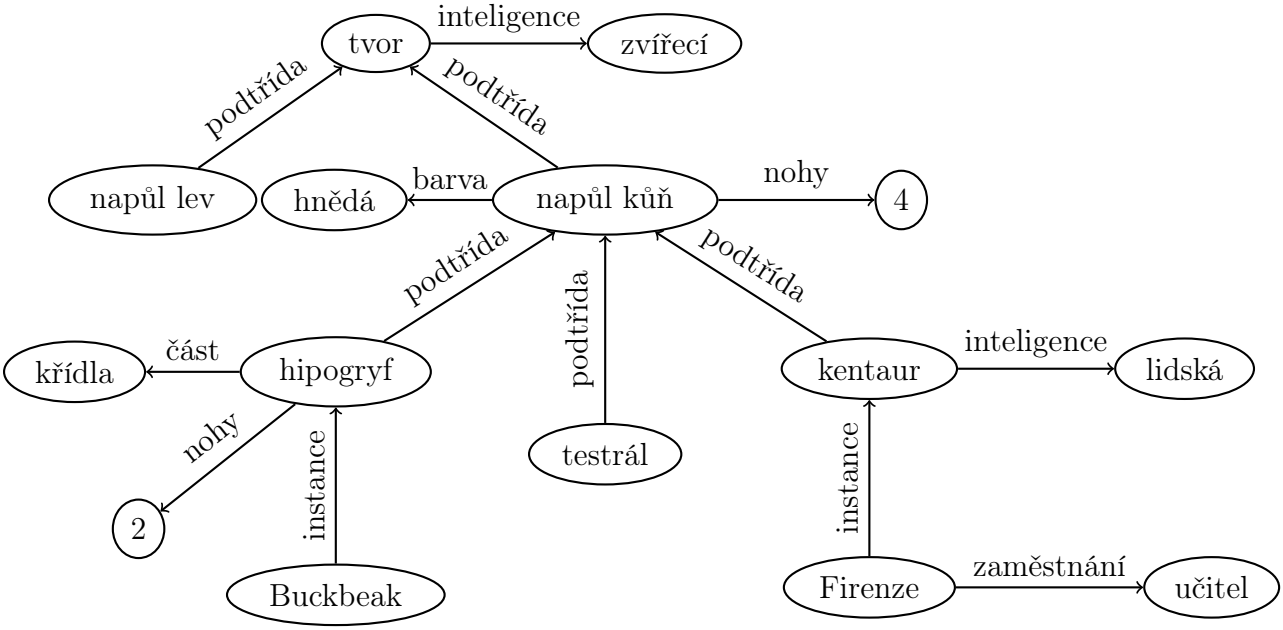
Příklad. Pomocí rámce reprezentujte znalosti ze sémantické sítě z předchozího příkladu.

objekty	**sloty**	**hodnoty**
ovoce		
	*chuť:	sladká
	*do_zmrzliny:	ano
rajče		
	podtřída:	ovoce
	do_zmrzliny:	ne
maliny		
	podtřída:	ovoce
	*barva:	červená
maliny v kuchyni		
	instance:	maliny

Objekty a hodnoty slotů jsou ekvivalentní s uzly grafu sémantické sítě, sloty jsou ekvivalentní s hranami grafu.

Příklad 9.2.1. Vytvořte sémantickou síť, která bude reprezentovat aspoň 4 libovolné kategorie domácích zvířat a jejich vlastnosti.

Příklad 9.2.2. Uvažte následující sémantickou síť, která částečně reprezentuje magické tvory.



Odpovězte na následující otázky.

a) Mají hipogryfové lidskou inteligenci?

b) Má Firenze lidskou inteligenci?

c) Kolik noh mají testrálové?

d) Jakou barvu má hipogryf?

Příklad 9.2.3. Podle informací vyplývajících ze sémantické sítě z předchozího příkladu popište, co víte o následujících pojmech:

1. Buckbeak,
2. Firenze,
3. napůl lev.

Příklad 9.2.4. Pomocí rámce reprezentujte znalosti o pojmech *napůl kůň* a *Firenze*, které jsou vyznačeny v sémantické síti v Příkladu 9.2.2.

Příklad 9.2.5. Uvažte následující rámec:

akční film		
	podtřída:	film
	**exploze*	ano
Drive		
	instance:	akční film
	exploze:	ne
Skyfall		
	instance:	akční film
	série:	James Bond
komedie		
	zábavná	ano
	podtřída:	film
černá komedie		
	podtřída:	komedie
	**země původu:*	Velká Británie
	humor:	černý
Horší než smrt		
	instance:	černá komedie
Big Lebowski		
	instance:	černá komedie
	země původu:	USA
romantická komedie:		
	podtřída:	komedie
	**konec:*	šťastný
Annie Hall		
	instance:	romantická komedie

Rozhodněte, zda z rámce vyplývají následující tvrzení:

a) Všechny akční filmy obsahují exploze.

b) Skyfall obsahuje exploze.

c) Akční film má více instancí než černá komedie.

d) Horší než smrt je z Británie.

e) Annie Hall je z USA.

f) Big Lebowski je zábavný film.

Příklad 9.2.6. Rámec z předchozího příkladu převeďte na sémantickou síť.

9.3 Nejisté znalosti

V běžném životě se často setkáváme s nejistými znalostmi. K reprezentaci neurčitých znalostí se používají například nemonotónní logiky, které předpokládají, že výchozí výroky jsou pravdivé, dokud nejsou vyvráceny. Dalším přístupem je práce s pravděpodobností, které se budeme věnovat v tomto oddílu.

Při definování pravděpodobnosti pracujeme s množinou *(všech) elementárních jevů* Ω, která zahrnuje všechny možné výsledky náhodného pokusu. V případě hodu kostkou by bylo $\Omega = \{1, 2, ..., 6\}$, v případě hodu mincí $\Omega = \{\text{rub}, \text{líc}\}$. Každá podmnožina množiny všech elementárních jevů se nazývá *jev*. V případě hodu kostkou může být jevem například „padné sudé číslo“, čili množina $\{2, 4, 6\}$. Množinu všech jevů značíme 2^Ω (známá též obecně jako *potenční množina* či *množina všech podmnožin*).

Definice 59: Uvažme konečnou množinu elementárních jevů Ω. *Pravděpodobnost* je funkce $P : 2^\Omega \to [0, 1]$ přiřazující jevům hodnoty z intervalu $[0, 1]$, která splňuje

- $P(\Omega) = 1$, čili pravděpodobnost celé množiny jevů je 1,
- $P(A \cup B) = P(A) + P(B)$ pro disjunktní množiny $A \subseteq \Omega$, $B \subseteq \Omega$, čili pravděpodobnost disjunktních množin je aditivní.

Množina elementárních jevů Ω vybavená pravděpodobností P se nazývá *pravděpodobnostní prostor* (Ω, P).

Intuitivní chápání definice pravděpodobnosti je, že hodnota $P(A)$ určuje pravděpodobnost, že při náhodném pokusu nastane nějaký z jevů $\omega \in A$. Pokud je $A = \Omega$, pochopitelně je pravděpodobnost 1, jelikož nějaký z jevů nastat musí. Vyplývá z axiomů, že $P(\varnothing) = 0$? Vyplývá z axiomů, že $P(\Omega - A) = 1 - P(A)$? Rozmyslete si.

Příklad. Jsou následující funkce *pravděpodobnosti* nad množinou $\Omega = \{\text{rub}, \text{líc}\}$? U těch, které nejsou, dokažte.

a) $P_a = \{\varnothing \mapsto 0, \quad \{\text{rub}\} \mapsto 0.2, \quad \{\text{líc}\} \mapsto 0.7, \quad \{\text{rub}, \text{líc}\} \mapsto 1\}$,

b) $P_b = \{\varnothing \mapsto 0, \quad \{\text{rub}\} \mapsto -0.2, \quad \{\text{líc}\} \mapsto 1.2, \quad \{\text{rub}, \text{líc}\} \mapsto 1\}\}$,

c) $P_c = \{\varnothing \mapsto 0, \quad \{\text{rub}\} \mapsto 0, \quad \{\text{líc}\} \mapsto 1, \quad \{\text{rub}, \text{líc}\} \mapsto 1\}\}$,

d) $P_d = \{\varnothing \mapsto 0.3, \quad \{\text{rub}\} \mapsto 0.3, \quad \{\text{líc}\} \mapsto 0.7, \quad \{\text{rub}, \text{líc}\} \mapsto 1\}\}$,

a) Ne, protože $P_a(\{\text{rub}, \text{líc}\}) = 1 \neq 0.9 = P_a(\{\text{rub}\}) + P_a(\{\text{líc}\})$.

b) Ne, protože $P_b(\{\text{rub}\}) = -0.2 \notin [0, 1]$.

c) Ano.

d) Ne, protože $P_d(\{\text{rub}\}) = 0.3 \neq 0.6 = P_d(\{\text{rub}\}) + P_d(\varnothing)$.

Elementární jevy často vykazují nějaké číselné charakteristiky, jako například číslo hozené na kostce či množství srážek v mm. K popisu číselných charakteristik slouží pojem *náhodné proměnné*.

Definice 60: *Náhodná proměnná* X nad pravděpodobnostním prostorem (Ω, P) je libovolná (měřitelná) funkce $X : \Omega \to \mathbb{R}$.

Náhodná proměnná tedy přiřazuje každému elementárnímu jevu číselnou hodnotu. Speciálním typem je *booleovská* náhodná proměnná, která své číselné hodnoty omezuje na 0 (nepravda) a 1 (pravda).

Příklad. Uvažme pokus, ve kterém 4 krát hodíme mincí. Dále mějme následující výsledek pokusu: líc, líc, rub, líc. Uvažte následující náhodné proměnné:

a) V každém hodu padl líc.

b) Počet padlých líců.

Jakých hodnot nabudou tyto proměnné pro uvedený pokus? Jakých hodnot mohou nabývat všeobecně?

a) 0 (nepravda). Náhodná proměnná nabývá hodnot z množiny {0, 1} v závislosti na tom, zda v náhodném pokusu padly jenom líce.

b) 3. Náhodná proměnná nabývá hodnot z množiny {0, 1, 2, 3, 4} v závislosti na počtě padlých líců.

V případě booleovských proměnných budeme dále jejich hodnoty značit pro proměnnou A rovněž jako a a $\neg a$ (čili název proměnné začínající malým písmenem) vedle klasického 1 (pravda) a 0 (nepravda).

Když máme definovánu náhodnou proměnnou, můžeme začít uvažovat o pravděpodobnostech, s nimiž bude nabývat jednotlivých hodnot.

Definice 61: Uvažujme náhodnou proměnnou X nad pravděpodobnostním prostorem (Ω, P). *Pravděpodobnost, že proměnná* X *nabude hodnoty* a, ozn. $P(X = a)$, je definována

jako

$$P(X = a) = P(\{\omega \in \Omega \mid X(\omega) = a\}),$$

neboli pravděpodobnost množiny elementárních jevů, pro něž nabývá náhodná proměnná hodnoty a.
Pokud je v uvažovaném kontextu zápis jednoznačný, $P(X = a)$ můžeme značit zkráceně jako $P(a)$.

Rozložení pravděpodobnosti mezi jednotlivými hodnotami náhodné proměnné v souladu s uvedenou definicí definuje funkci, která se nazývá *rozdělení pravděpodobností náhodné proměnné*.

Definice 62: *Rozdělení* (či *distribuce*) *pravděpodobnosti náhodné proměnné* X, značeno $P(X)$, je funkce $P(X) : \mathbb{R} \to [0, 1]$, která přiřadí každé hodnotě $a \in \mathbb{R}$ pravděpodobnost, že X nabude hodnoty a podle Definice 61.

Rozložení pravděpodobnosti obsahuje pravděpodobnosti všech hodnot náhodné proměnné. Jelikož náhodná proměnná vždy nabývá nějaké hodnoty, pravděpodobnost, že nabude nějaké z hodnot, je 1. Proto platí následující tvrzení.

Věta 63: Součet hodnot rozložení pravděpodobnosti náhodné proměnné je roven 1.

Příklad. Uvažte booleovskou náhodnou proměnnou Prší ve světě, kde pravděpodobnost toho, že prší, je 0.3. Uveďte, jakým hodnotám se rovnají následující výrazy:

a) $P(\neg\text{prší})$,

b) $P(\text{Prší})$.

a) Výraz označuje pravděpodobnost toho, že neprší. Jelikož víme, že náhodná proměnná má pouze dvě hodnoty, potom $P(\neg\text{prší}) = 1 - P(\text{prší}) = 0.7$.

b) Výraz označuje distribuci pravděpodobnosti, tedy

$$P(\text{Prší})(x) = \begin{cases} 0.3 & \text{pro hodnotu prší, neboli } x = 1 \text{ (pravda)}, \\ 0.7 & \text{pro hodnotu } \neg\text{prší, neboli } x = 0 \text{ (nepravda)}, \\ 0 & \text{jinak, neboli } x \neq 1 \wedge x \neq 0. \end{cases}$$

Distribuce pravděpodobností náhodných proměnných lze navzájem kombinovat. Vznikne tak složená distribuce pravděpodobností.

Definice 64: *Složené rozložení pravděpodobnosti náhodných proměnných* $X_1, \ldots, X_n$ určuje pravděpodobnost toho, že náhodné proměnné $X_1, \ldots, X_n$ nabudou (v tom samém pokusu) hodnot $a_1, \ldots, a_n$. Složené rozložení značíme $P(X_1 \wedge \ldots \wedge X_n)$ nebo $P(X_1, \ldots, X_n)$. Pravděpodobnost konkrétního přiřazení hodnot značíme obdobně, $P(a_1 \wedge \ldots \wedge a_n)$ nebo $P(a_1, \ldots, a_n)$, kde a_i je hodnota náhodné proměnné X_i.

Příklad. Mějme složenou distribuci booleovských náhodných proměnných Prší a Sněží.

a) Kolik pravděpodobnostních hodnot obsahuje jejich složená distribuce?

b) Co reprezentuje výraz $P(\text{sněží}, \neg\text{prší})$?

a) 4. Jsou to $P(\text{sněží}, \text{prší})$, $P(\neg\text{sněží}, \text{prší})$, $P(\text{sněží}, \neg\text{prší})$ a $P(\neg\text{sněží}, \neg\text{prší})$.

b) Výraz reprezentuje pravděpodobnost, že sněží a neprší zároveň.

Složené rozložení pravděpodobnosti všech náhodných proměnných v uvažovaném světě obsahuje pravděpodobnosti všech možných přiřazení hodnot proměnným. Zahrnuje tedy všechny situace, které mohou v daném světě nastat (z pohledu hodnot náhodných proměnných), nazývané též *atomické události*. Pomocí této distribuce pak lze vypočítat libovolnou pravděpodobnost, a to tak, že sečteme pravděpodobnosti všech atomických událostí, které splňují podmínku jevu, jehož pravděpodobnost nás zajímá.

Příklad. Uvažte náhodné proměnné Počasí s hodnotami {oblačno, sněží, slunečno} a Úterý s hodnotami {pravda, nepravda}. Pomocí hodnot složeného rozložení pravděpodobnosti vyjádřete $P(\text{oblačno} \vee \text{slunečno})$.

$P(\text{oblačno} \vee \text{slunečno}) = P(\text{úterý} \wedge \text{oblačno}) + P(\neg\text{úterý} \wedge \text{oblačno}) + P(\text{úterý} \wedge \text{slunečno}) + P(\neg\text{úterý} \wedge \text{slunečno})$.

Pravděpodobnost, kterou jsme se zabývali doteď, vyjadřuje pravděpodobnost jevu bez ohledu na další informace, které o světě máme. Při výpočtu podmíněné pravděpodobnosti naopak do výpočtu zapojujeme znalosti o už odehraných událostech.

Například ze statistiky známek z minulého roku student ví, že pravděpodobnost hodnocení A na zkoušce je 1/10. Pokud se ovšem učil na zkoušku týden a před zkouškou spal 9 hodin, pak může předpokládat, že pravděpodobnost A je vyšší. Čím více znalostí máme, tím lépe bude hodnota pravděpodobnosti odpovídat realitě.

Máme-li dva jevy A a B, podmíněná pravděpodobnost $P(A \mid B)$, čili A za podmínky B, je pravděpodobnost jevu A, pokud nastal jev B.

Definice 65: Uvažme jevy $A, B \subseteq \Omega$, kde $P(B) \neq 0$ *Podmíněná pravděpodobnost A za předpokladu B* je definována vztahem

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}.$$

Nemá-li výskyt jevu B vliv na pravděpodobnost jevu A, říkáme, že jevy A a B jsou *stochasticky nezávislé*. Pro nezávislé jevy platí, že pravděpodobnost toho, že nastanou oba, je násobkem pravděpodobnosti jednotlivých jevů.

Definice 66: Dva jevy $A, B \subseteq \Omega$ jsou *stochasticky nezávislé*, pokud

$$P(A \cap B) = P(A) \cdot P(B).$$

Jsou-li jevy A, B nezávislé, pak platí $P(A \mid B) = P(A)$, neboli zjištěním, zda jev B nastal, se nedozvíme žádnou další informaci, o pravděpodobnosti jevu A.
Z pravidla o podmíněné pravděpodobnosti lze odvodit Bayesovu větu:

Věta 67 (jednoduchá Bayesova věta): Uvažme dva náhodné jevy $A, B \subseteq \Omega$, přičemž $P(B) \neq 0$. Pak platí

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}.$$

Bayesovu větu lze dobře aplikovat v případech, kdy uvažujeme o pravděpodobnostech příčiny a následku a známe hodnotu jedné podmíněné pravděpodobnosti. Můžeme tak například vyjádřit pravděpodobnost, že člověk má určitou nemoc, má-li její příznak (přičemž pravděpodobnost příznaku, pokud člověk nemoc má, známe), či pravděpodobnost, že výsledek testu je korektní, vyšel-li pozitivně.

Příklad. Víme, že pravděpodobnost slunečného počasí je 0.3 a pravděpodobnost horkého počasí (alespoň 30 °C) je 0.05. Dále víme, že pokud je horko, pravděpodobnost slunečního počasí stoupne na 0.9.

a) Jaká je pravděpodobnost, že svítí slunce a je horko?

b) Jaká je pravděpodobnost, že svítí slunce a horko není?

c) Jaká je pravděpodobnost, že je venku horko, pokud z okna vidíme, že svítí slunce?

Pravděpodobnost slunečného počasí budeme značit $P(s) = 0.3$, pravděpodobnost horkého počasí $P(h) = 0.05$ a pravděpodobnost slunečného počasí za předpokladu, že je horko, $P(s|h) = 0.9$.

a) Pravděpodobnost, že svítí slunce a je horko můžeme označit jako $P(s \wedge h)$. Tuto pravděpodobnost umíme vypočítat pomocí pravidla o podmíněné pravděpodobnosti, tedy $P(s \wedge h) = P(s \mid h) \cdot P(h) = 0.9 \cdot 0.05 = 0.045$.

b) Pravděpodobnost, že svítí slunce a horko není, tedy $P(s \wedge \neg h)$, vypočítáme s využitím principu složené distribuce pravděpodobnosti. Víme, že $P(s) = P(s \wedge h) + P(s \wedge \neg h)$, potom $P(s \wedge \neg h) = P(s) - P(s \wedge h) = 0.3 - 0.045 = 0.255$.

c) Pro výpočet pravděpodobnosti horkého počasí za předpokladu, že je slunečno, $P(h|s)$ použijeme Bayesovu větu. Platí $P(h|s) = \frac{P(s|h) \cdot P(h)}{P(s)} = \frac{0.9 \cdot 0.05}{0.3} = 0.15$. Pravděpodobnost, že je horko, pokud svítí slunce, je tedy 0.15.

Příklad 9.3.1. Zaveďte náhodnou proměnnou, která reprezentuje následující jevy:

a) Prší.

b) Venkovní teplota naměřená digitálním teploměrem.

c) Počet hodin, které strávil studiem PB016 absolvent předmětu.

Příklad 9.3.2. Uvažte následující tabulku složených pravděpodobností náhodné proměnné Počasí s hodnotami $\{0$ (slunečno), 1 (déšť), 2 (sníh)$\}$ a booleovských proměnných Zácpy a Zpoždění.

	zpoždění		¬zpoždění	
	zácpy	¬zácpy	zácpy	¬zácpy
slunečno	0.15	0.10	0.05	0.30
déšť	0.15	0.05	0.02	0.08
sníh	0.07	0.01	0.01	0.01

Určete hodnoty následujících výrazů.

a) P(zácpy)

b) P(Zácpy)

c) P((zácpy $\vee$¬zpoždění) $\wedge$ slunečno)

d) P(Zácpy | déšť)

e) P(sníh | ¬zácpy $\vee$ zpoždění)

Příklad 9.3.3. Uvažte tabulku složeného rozložení pravděpodobnosti z předchozího příkladu. Určete, kolik hodnot z tabulky potřebujeme použít pro výpočet následujících pravděpodobností.

a) P(déšť $\wedge$ ¬zpoždění $\wedge$ zácpy)

b) P(slunečno $\wedge$ zpoždění)

c) P(slunečno $\vee$ zpoždění)

d) P($\neg$sníh)

Příklad 9.3.4. Doplňte následující tabulku složených pravděpodobností o hodnoty a a b tak, aby jevy X a Y byly na sobě nezávislé.

X	Y	$P(X \cap Y)$
1	1	$3/5$
1	0	$1/5$
0	1	a
0	0	b

Příklad 9.3.5. Vyjádřete následující pravděpodobnosti:

a) $P(a \mid b)$ pomocí $P(b)$ a $P(\neg a \wedge b)$,

b) $P(a \mid b)$ pomocí $P(a \wedge b)$ a $P(\neg a \wedge b)$,

c) $P(b)$ pomocí $P(a \mid b)$ a $P(a \wedge b)$.

Příklad 9.3.6. Dokažte, že

a) jsou-li dva jevy A, B nezávislé, pak $P(A \mid B) = P(A)$,

b) jsou-li A, B dva jevy a $P(B) \neq 0$, platí Bayesův vzorec

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}.$$

Vyjděte z Definice 65 a Definice 66.

Příklad 9.3.7. Následující situace modelujte pomocí náhodných proměnných a odpovězte na otázky.

a) Mimozemšťané mohou být přátelští, ale nemusí. 75 % jich přátelských je. Přátelští mimozemšťané přistávají na zemi v 90 % případů přes den, nepřátelští výhradně v noci. Pokud mimozemšťan přistane v noci, jaká je pravděpodobnost, že je přátelský?

Příklad 9.3.8. Alenka je odbornice na cvrčky. Většinu druhů dokáže určit podle jejich zvuku. Cvrček zpěvavý však dokáže napodobovat zvuky jiných cvrčků, takže je velmi obtížné ho rozeznat. Navíc je velmi vzácný – tvoří pouze 2 % populace cvrčků. Přesto má Alenka výborné statistiky při rozpoznávání tohoto cvrčka, s pravděpodobností 73 % správně určí, zda se jedná o cvrčka zpěvavého, či nikoli. Pokud se nejedná o cvrčka zpěvavého, dokáže ho určit pouze v 87 % případů.

a) Jaká je pravděpodobnost, že Alenka pozná cvrčka zpěvavého?

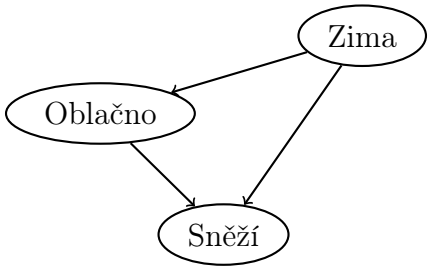
b) Pokud slyšíme cvrčka, jaká je pravděpodobnost, že Alenka řekne, že jde o cvrčka zpěvavého?

Příklad 9.3.9. Alenčin kamarád Bořek tvrdí, že jeho statistiky jsou ještě lepší. Zda se jedná o cvrčka zpěvavého, či nikoli, prý určí správně s pravděpodobností 98 %. K tomu si přisadí Cyril s tvrzením, že on má ještě lepší výsledky. Pokud slyší cvrčka zpěvavého, správně ho prý určí s pravděpodobností 100 %. Bořek ani Cyril nejsou experti na cvrčky, ale jejich tvrzení jsou pravdivá. Jak je to možné?

9.4 Bayesovské sítě

Bayesovská síť je datová struktura používaná k reprezentaci pravděpodobnosti nejistých jevů a závislostí mezi nimi. Skládá se z acyklického orientovaného grafu, v němž uzly představují jednotlivé náhodné proměnné a hrany označují, které jevy jsou na sobě závislé. Pokud hrana vede z A do B, říkáme, že A je rodičem B, což znamená, že náhodná proměnná B je přímo závislá na náhodné proměnné A. Pro každou náhodnou proměnnou jsou uvedeny pravděpodobnosti jejích hodnot, které jsou podmíněny tím, jakých hodnot nabývají nadřazené jevy.

Bayesovská síť může vypadat například takto:



Přitom pravděpodobnostní rozdělení jednotlivých proměnných jsou uvedena v tabulce (pravděpodobnosti hodnot *nepravda* nejsou uvedeny, lze je snadno vypočítat):

Oblačno:

Z	$P(O)$
1	0.6
0	0.5

Zima:

$P(Z)$
0.2

Sněží:

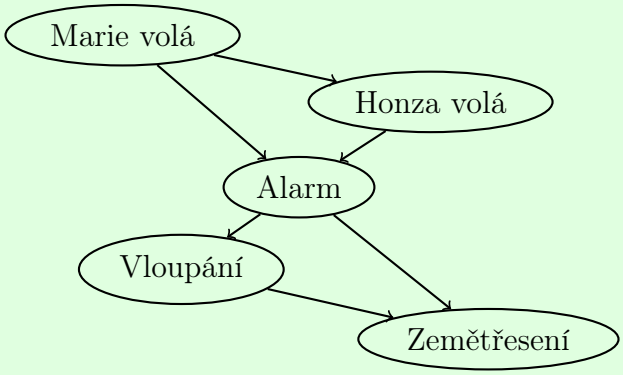
O	Z	$P(S)$
1	1	0.8
1	0	0.001
0	1	0.01
0	0	0

První řádek tabulky Sněží například určuje pravděpodobnost sněžení za předpokladu, že je zataženo a chladno.

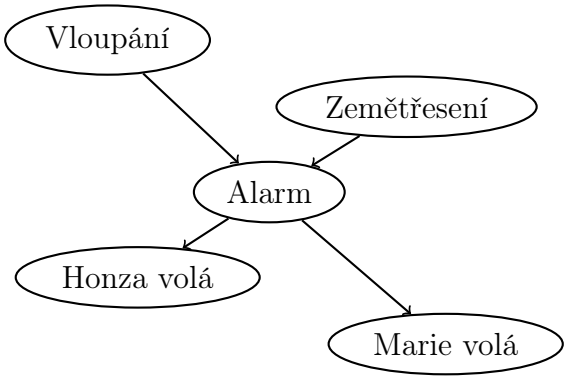
Bayesovskou síť tvoříme postupným přidáváním jevů jako uzlů. Do nového uzlu zavádíme hrany ze stávajících uzlů, které přímo ovlivňují jeho pravděpodobnost. Formálně to znamená, že pro uzel X_i hledáme minimální podmnožinu $\text{Parents}(X_i)$ z množiny již přidaných uzlů $X_1, ..., X_{i-1}$ takovou, že $P(X_i \mid \text{Parents}(X_i)) = P(X_i \mid X_1, ..., X_{i-1})$. Poté zopakujeme postup přidáním dalšího uzlu. Pořadí přidávání lze zvolit libovolně, ale přidávání příčin před následky obvykle vede ke kompaktnější síti.

Příklad. V domě máme nainstalován alarm, který se spustí v případě vloupání, ale také v případě zemětřesení. Požádali jsme sousedy Marii a Honzu, aby nám zavolali, pokud uslyší náš alarm. Uvažujme náhodné proměnné Vloupání, Zemětřesení, Alarm, Honza volá, Marie volá. Vytvořte síť přidáním uzlů v pořadí M, H, A, V, Z.

1. Přidáme uzel Marie volá. Jelikož v síti nejsou žádné uzly, uzel Marie volá nemá rodiče.
2. Přidáme uzel Honza volá. Pokud volá Marie, pravděpodobně se spustil alarm, a tudíž i pravděpodobnost, že zavolá Honza se zvyšuje – jev Honza volá je v tomto případě přímo závislý na jevu Marie volá. Vedeme tedy hranu z M do H.
3. Přidáme uzel Alarm. Pravděpodobnost, že zvoní alarm se přímo zvyšuje, pokud volá Marie, a ještě více, volá-li i Honza – z uzlů M a H vedeme hranu do A.
4. Přidáme uzel Vloupání. Pokud víme, zda alarm zvonil, informace o tom, zda volala Marie nebo Honza, nám o pravděpodobnosti vloupání nic nového neřekne (předpokládáme, že sousedé nesledují, zda někdo vloupává, pouze volají, pokud slyší alarm). Uzel vloupání je tedy přímo závislý pouze na alarmu.
5. Přidáme uzel Zemětřesení. Pokud víme, že zvonil alarm, pravděpodobnost zemětřesení se zvyšuje. Na druhou stranu, pokud víme, že došlo k vloupání a byl spuštěn alarm, pravděpodobnost je menší (alarm byl pravděpodobně spuštěn lupičem). Pravděpodobnost zemětřesení je tedy přímo ovlivněna spuštěním alarmu i vloupáním.



Z uvedeného příkladu je zřejmé, že dané pořadí přidávání uzlů nebylo zvoleno nejvhodněji a v síti vznikly neintuitivní závislosti. Síť však stále správně reprezentuje situaci. Pokud bychom přidávali uzly v pořadí od příčiny k následku, např. V, Z, A, H, M, vznikla by následující síť, která reprezentuje závislosti intuitivněji a je také kompaktnější:



Bayesovská síť reprezentuje složené rozdělení pravděpodobnosti všech náhodných proměnných, a to často kompaktněji než tabulka složené distribuce. Ze sítě tedy můžeme vyjádřit jakoukoli pravděpodobnost jevu v uvažovaném světě.

Příklad 9.4.1. Uvažte situaci popsanou v úvodu s náhodnými proměnnými Vloupání, Zemětřesení, Alarm, Honza volá, Marie volá. Vytvořte síť přidáváním uzlů v pořadí Z, V, A, M, H.

Příklad 9.4.2. Uvažte náhodné proměnné Bouřka, Zataženo, Léto, Hrom, Teplo a Blesk. Vytvořte bayesovskou síť, která bude zachytávat závislosti jevů co nejkompaktněji.

10 Strojové učení

Metody strojového učení nám umožňují konstruovat systémy, které se svou funkcionalitu postupně učí z dat. Tyto systémy využívají získávané znalosti k tomu, aby vylepšovaly svou výkonnost. Jednou ze základních vlastností takových systémů je schopnost generalizovat, tedy schopnost aplikovat naučené znalosti na doposud neviděná data.

Algoritmy strojového učení můžeme podle způsobu učení rozdělit do třech hlavních kategorií:

- *Učení s učitelem*, kde známe pro učicí data i očekávané výstupy.
- *Učení bez učitele*, kde se model učí vzory v datech bez dodatečných informací.
- *Zpětnovazebné učení* („reinforcement learning“), kde za provedené akce následuje zpětná vazba v podobě odměny nebo trestu.

Pomocí metod strojového učení můžeme řešit spoustu různých úloh. *Klasifikace* má za cíl přidělit vstupním objektům jednu z konečně mnoha předem určených tříd. Může jít například o rozhodování, zda počasí pro daný den ohodnotit jako `slunečno`, `polojasno`, nebo `deštivo`. Cílem *regrese* je na základě vstupů modelovat spojitou proměnnou, například odhadnout venkovní teplotu v určitém čase. *Shlukování*, které je obvykle příkladem učení bez učitele, se snaží vstupní data rozdělit do několika skupin, které shlukují vzájemně podobné příklady. V této sbírce se budeme věnovat pouze učení s učitelem. Zaměříme se hlavně na úlohu klasifikace, setkáme se ale i s regresí.

Konkrétních metod nebo algoritmů, které lze zařadit pod strojové učení, existuje opět mnoho. V této kapitole se budeme postupně věnovat dvěma základním typům. V první části to bude jednoduchý (ale velmi používaný) model *rozhodovacích stromů*. Druhá část se pak bude týkat algoritmů okolo *lineárních modelů*. Na tyto modely poté navážeme v další kapitole, která představí složitější architektury *neuronových sítí*.

10.1 Rozhodovací stromy

Model rozhodovacího stromu nám poskytuje přirozený pohled na klasifikační úlohy. Větve takového stromu reprezentují sekvence testů na atributy vstupu, pomocí kterých postupně dojdeme k výsledku. Není-li řečeno jinak, budeme pro jednoduchost předpokládat, že atributy vstupních příkladů nabývají diskrétních hodnot.

Definice 68: *Rozhodovací strom* je speciální typ kořenového stromu, který má dva typy uzlů:

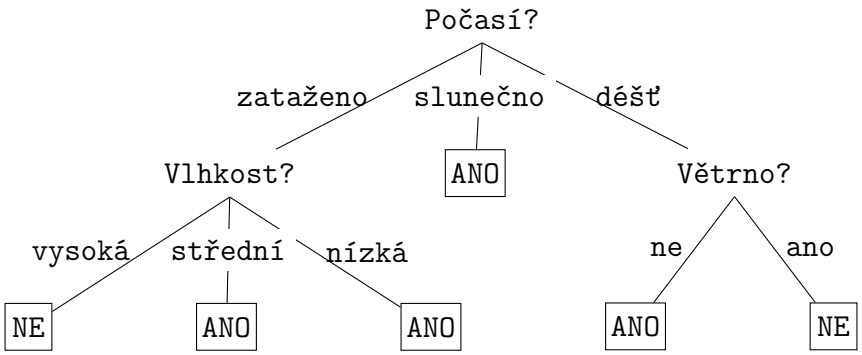
- vnitřní (rozhodovací) uzly,
- listové uzly.

Vnitřní uzel stromu odpovídá testu na hodnotu některého atributu vstupu. Hrany vedoucí z uzlu do jeho potomků poté reprezentují možné hodnoty testovaného atributu. Listové uzly specifikují výslednou třídu.

Definice 69: Říkáme, že rozhodovací strom je *konzistentní* s konkrétním datasetem, právě když správně oklasifikuje všechny příklady z datasetu.

Ilustrujme si klasifikaci pomocí rozhodovacího stromu na elementárním příkladu. Pro jednoduchost uvažujme případ binární klasifikace, kde jsou listové uzly označeny pouze *ANO*/*NE*. Obecně ale samozřejmě rozhodovací stromy takto omezeny nejsou.

Příklad. Mějme následující strom reprezentující rozhodovací problém, zda jsou venkovní podmínky vhodné pro hraní tenisu (rozhodujeme se pomocí veličin `Vlhkost`, `Počasí`, `Větrno`).



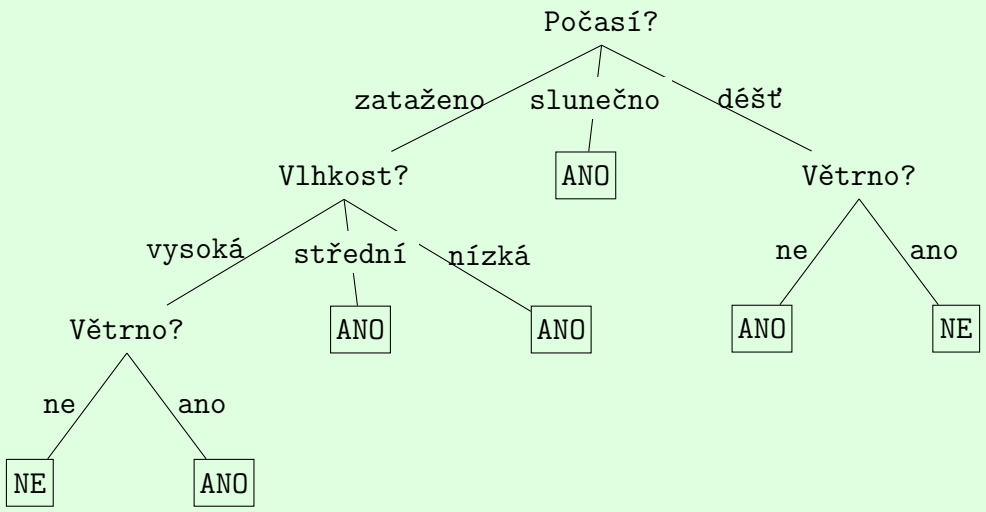
a) Klasifikujte pomocí daného stromu následující situace (příklady obsahují atributy v pořadí `Vlhkost`, `Počasí`, `Větrno`):
   1. `[vysoká, slunečno, ne]`
   2. `[střední, déšť, ano]`
   3. `[vysoká, zataženo, ne]`

b) Upravte strom přidáním jednoho rozhodovacího uzlu tak, aby výstupem pro příklad `[vysoká, zataženo, ano]` bylo `ANO`. Ostatní příklady by nový strom měl klasifikovat stejně jako původní.

c) Mohli bychom původní strom použít i ke klasifikaci příkladů z datasetu čtveřic s atributy `Vlhkost`, `Počasí`, `Větrno` a `Teplota`? Vstupy by tedy kromě původních tří atributů obsahovaly ještě dodatečný atribut `Teplota`.

a)
   1. ANO – z prvního rozhodovacího uzlu se dostaneme po hraně `slunečno` rovnou do listového uzlu
   2. NE
   3. NE

b) Například následující strom:



c) Ano, mohli. Výsledná třída by ale na novém atributu `Teplota` nikdy nezáležela, jelikož se ve stromě podle tohoto atributu nikde nerozhoduje.

Příklad 10.1.1. Mějme dataset D zadaný následující tabulkou (řádky reprezentují jednotlivé příklady, sloupce atributy, závislá proměnná je *Tenis*).

Vlhkost	*Počasí*	*Větrno*	*Tenis*
vysoká	zataženo	ano	NE
střední	slunečno	ne	ANO
nízká	slunečno	ano	ANO
střední	déšť	ne	NE

a) Vytvořte (ručně) rozhodovací strom, který je konzistentní s tímto datasetem. Snažte se o co nejmenší strom.

b) Jak by (konzistentní) strom vypadal, pokud by v D byl navíc ještě příklad `[střední, slunečno, ne]` s třídou NE?

Příklad 10.1.2. Je pravda, že v žádné větvi rozhodovacího stromu nemá cenu testovat stejný atribut vícekrát? Svou odpověď odůvodněte.

10.2 Učení rozhodovacích stromů

K vytvoření stromu využíváme (jak je tomu ve strojovém učení obvyklé) množinu trénovacích dat. Trénovací data se v tomto případě skládají z dvojic příkladů a jejich očekávaných tříd.

Často lze ovšem vytvořit spoustu různých stromů konzistentních s danými trénovacími daty. Tyto stromy se mohou lišit jak ve své velikosti, tak ve schopnosti generalizace. Cílem obvykle je vytvořit co nejmenší konzistentní strom, který co nejlépe generalizuje na nové příklady.

Jelikož je ale problém nalezení nejmenšího konzistentního rozhodovacího stromu *NP-úplný*, musíme si vystačit s heuristickými postupy. Tyto postupy budují strom od kořene, a v každém uzlu se snaží najít (v nějakém smyslu) co *nejlepší* rozhodovací atribut. Představíme si postup, který jako metriky k nalezení takového atributu využívá *entropii* a *informační zisk* (*information gain*). Nejdříve si tyto metriky zadefinujme.

Entropie je obecně mírou neuspořádanosti, nejistoty, nebo informace. V původním smyslu jde o míru nejistoty náhodné proměnné – my budeme uvažovat jako tuto náhodnou proměnnou třídu. Uvažme pro ilustraci případ binární klasifikace (proměnná třídy má 2 hodnoty). Nechť máme dataset tvořený příklady pouze jedné třídy. Pokud bychom náhodně vytáhli jeden příklad, už dopředu máme jistotu, jaká bude jeho třída. Proto výsledek tohoto pokusu nenese žádnou informační hodnotu a entropie (míra informace) je nulová. Uvažme naopak dataset, jenž je z poloviny tvořen příklady jedné třídy, a z druhé poloviny příklady třídy druhé. V tomto případě nemáme dopředu žádnou jistotu, jakou bude mít náhodně zvolený příklad třídu, a proto je entropie nejvyšší (pro binární klasifikaci rovna 1).

Definice 70: Uvažujme množinu příkladů D. Nechť jsou příklady rozděleny do n tříd a P_i je pravděpodobnost, že náhodně vybraný příklad bude mít třídu i. *Entropii* pro dataset D spočítáme jako

$$E(\langle P_1, \ldots, P_n \rangle) = \sum_{i=1}^{n} -P_i \cdot \log_2(P_i)$$

Máme-li třídy pouze 2, vzorec se obvykle zapisuje následovně:

$$E(\langle P_1, P_2 \rangle) = -P_1 \cdot \log_2(P_1) - (1 - P_1) \cdot \log_2(1 - P_1)$$

Entropii můžeme využít k výpočtu informačního zisku, jenž nám dává jakýsi heuristický odhad, jak je který atribut „dobrý“. Informační zisk pro určitý atribut nám intuitivně říká, jak moc se liší „neuspořádanost“ množiny před a po rozdělení podle daného atributu – tedy kolik informace získáme testem na hodnotu onoho atributu. Informační zisk je vyšší, pokud se entropie („neuspořádanost“) rozdělením sníží.

Definice 71: Uvažujme množinu příkladů D a atribut A, který může nabývat celkem k hodnot. A tedy rozděluje množinu D na podmnožiny $D_1, \ldots, D_k$. Nechť p je celkový počet pozitivních příkladů v D a n počet negativních (uvažujeme binární klasifikaci). Dále nechť p_i je počet pozitivních příkladů v D_i a n_i počet negativních v D_i.
Informační zisk atributu A spočítáme jako

$$Gain(A) = E(\langle \tfrac{p}{p+n}, \tfrac{n}{p+n} \rangle) - \sum_{i=1}^{k} \tfrac{p_i+n_i}{p+n} \cdot E(\langle \tfrac{p_i}{p_i+n_i}, \tfrac{n_i}{p_i+n_i} \rangle)$$

Příklad. Mějme dataset daný tabulkou níže (řádky reprezentují jednotlivé příklady, sloupce

kromě prvního atributy, závislá proměnná je *Tenis*). Spočítejte

a) entropii celého setu,

b) informační zisk pro atribut *Počasí*.

	Vlhkost	*Počasí*	*Větrno*	*Teplota*	*Tenis*
1	vysoká	zataženo	ano	nižší	NE
2	střední	zataženo	ne	vyšší	ANO
3	nízká	slunečno	ano	nižší	ANO
4	střední	déšť	ne	nižší	NE
5	nízká	slunečno	ano	vyšší	ANO
6	vysoká	déšť	ne	vyšší	NE
7	střední	slunečno	ano	nižší	ANO

Zamyslete se, co nám spočítaná entropie říká o datasetu, a zda-li může být atribut *Počasí* vhodným kandidátem pro rozhodovací uzel.

a) $E(\langle P_{ANO}, P_{NE}\rangle) = -\frac{4}{7} \cdot log_2(\frac{4}{7}) - (\frac{3}{7}) \cdot log_2(\frac{3}{7}) \approx \frac{4}{7} \cdot 0.807 + (\frac{3}{7}) \cdot 1.222 \approx 0.985$

b) $Gain(Počasí) \approx 0.985 - (\frac{2}{7} \cdot E(\langle\frac{1}{2}, \frac{1}{2}\rangle) + \frac{3}{7} \cdot E(\langle\frac{3}{3}, \frac{0}{3}\rangle) + \frac{2}{7} \cdot E(\langle\frac{0}{2}, \frac{2}{2}\rangle)) \approx 0.985 - (\frac{2}{7} \cdot 1 + \frac{3}{7} \cdot 0 + \frac{2}{7} \cdot 0) \approx 0.699$

Původní entropie je vysoká, což znamená, že dataset je poměrně vyvážený. Informační zisk atributu *Počasí* je vysoký, zdá se tedy být vhodným kandidátem. Můžeme si všimnout, že například hodnoty *déšť* i *slunečno* již plně určují výsledek.

Samotná procedura vytváření stromu probíhá rekurzivně od kořene. V každém uzlu vybereme pomocí informačního zisku rozhodovací atribut. Podle různých hodnot vybraného atributu rozdělíme trénovací množinu a rekurzivně pokračujeme. Uzel automaticky prohlásíme za listový, pokud pro něj uvažovaná množina obsahuje jen příklady jedné třídy (jeho label bude tato třída). Uzel také prohlásíme za listový, pokud jsme v dané větvi již otestovali všechny atributy – jeho třída se zvolí podle převažující třídy mezi příklady (v tomto případě nebude strom kompletně konzistentní).

Příklad 10.2.1. Uvažujme opět dataset daný tabulkou z předchozího příkladu. Nalezněte atribut, který by učicí algoritmus zvolil jako rozhodovací atribut pro kořen stromu.

Příklad 10.2.2. Nyní konečně aplikujte i zbytek učicího algoritmu a vybudujte pomocí něj kompletní rozhodovací strom. Dataset zůstává stejný jako v předchozích příkladech. První krok jste již spočítali v minulém příkladu.

Příklad 10.2.3. Mějme následující množiny hodnot tříd (p/n). Pro každou z nich spočítejte hodnotu entropie. Co jste z těchto výsledků vypozorovali?

a) $\{p, p, p, p\}$

b) $\{p, p, p, n\}$

c) $\{p, p, n, n\}$

d) $\{p, n, n, n\}$

e) $\{n, n, n, n\}$

Příklad 10.2.4. Uvažujte následující dataset.

A	*B*	*C*	*Třída*
0	0	0	T
0	1	0	F
1	0	0	F
1	1	1	T

a) Na tento dataset aplikujte učicí algoritmus a vybudujte pomocí něj kompletní rozhodovací strom. Mají-li v některém kroku dva atributy stejný informační zisk, zvolte lexikograficky.

b) Existuje konzistentní rozhodovací strom s menší hloubkou, než má strom vytvořený pomocí heuristického algoritmu? Pokud ano, vytvořte jej. Pokud ne, odůvodněte proč tomu tak je.

Příklad 10.2.5. Nechť k-hodnotový atribut A rozdělí množinu příkladů D na podmnožiny D_i, pro které platí, že obsahují p_i pozitivních a n_i negativních příkladů. Ukažte, že pokud je pro všechna i poměr $\frac{p_i}{p_i+n_i}$ stejný, atribut má nulový informační zisk.

Příklad 10.2.6. Uveďte, jaké hlavní výhody a nevýhody rozhodovacích stromů znáte. Uvažujte základní trénovací algoritmus.

Příklad 10.2.7. Jak už bylo zmíněno v jednom z předchozích příkladů, základní verze učení rozhodovacích stromů je obecně náchylná k tzv. přeučení (*overfitting*), což může ovlivnit schopnost generalizovat. Jaké vás napadnou způsoby, kterými lze často docílit lepší generalizace?

10.3 Perceptron a lineární klasifikace

V této sekci opustíme rozhodovací stromy a představíme si jiný typ algoritmů. Konkrétně se zaměříme na metody založené na lineárních modelech a představíme si nejjednodušší verze (jednovrstvých) neuronových sítí. Takové modely obvykle pracují nad spojitou doménou, a tedy jejich vstupy bývají vektory (reálných) čísel. Předpokládejme proto dále, že pracujeme s objekty, jejichž atributy mají formu reálných čísel. Toho jde obecně docílit vhodným předzpracováním dat.

Lineární modely už můžete znát ze statistiky, kde je okolo nich vybudována rozsáhlá teorie. V kontextu tohoto kurzu využijeme myšlenku lineárních modelů k tomu, abychom si pomocí nich zavedli takzvané umělé neurony. Neurony uvnitř fungují jako lineární model, na výslednou lineární transformaci vstupů na závěr ještě aplikují takzvanou *aktivační funkci*. Tyto umělé neurony pak tvoří základní stavební bloky neuronových sítí. To, jak můžeme neurony propojovat do nejrůznějších architektur a vytvářet složitější sítě, si podrobněji představíme

v další kapitole.

V této sekci se budeme soustředit na jednoduchý binární klasifikátor, *perceptron*. Aby model vstupy rozdělil do dvou tříd, používá jako aktivaci *prahovou funkci*. V n-rozměrném prostoru vstupů perceptron zadává nadrovinu oddělující příklady různých tříd, jak si konkrétněji ukážeme v následující sekci.

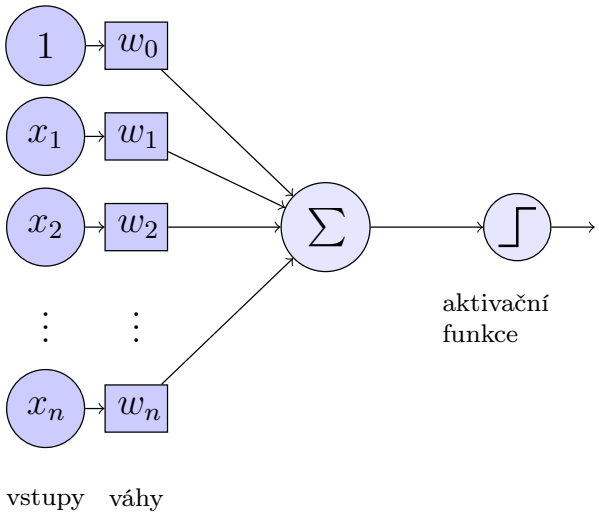
Definice 72: Uvažujme vstupní vektor $\vec{x} = \langle x_1, \ldots, x_n \rangle$. Pak *rozšířený vstupní vektor* je $\tilde{x} = \langle 1, x_1, \ldots, x_n \rangle$.

Rozšířený vstupní vektor se nám bude hodit pro zkrácený zápis některých výrazů pomocí vektorových operací.

Definice 73: Uvažujme příklad $\vec{x} = \langle x_1, \ldots, x_n \rangle$ a vektor vah $\vec{w} = \langle w_0, w_1, \ldots, w_n \rangle$. Výstup perceptronové jednotky s váhami $\vec{w}$ pro příklad $\vec{x}$ je

$$C[\vec{w}](\vec{x}) = \begin{cases} 1 & \text{pokud } w_0 + \sum_{i=1}^{n} w_i \cdot x_i = \vec{w} \cdot \tilde{x} \geq 0, \\ 0 & \text{jinak.} \end{cases}$$

Funkcionalitu perceptronu si můžeme ilustrovat následujícím diagramem.



Příklad. Mějme perceptron s váhovým vektorem $\vec{w} = \langle 1, 1, 2.5, 0, -1 \rangle$.

a) Určete, jak tento model oklasifikuje následující příklady.

1) $\vec{x}_1 = \langle 0, 1, 1, 2 \rangle$
2) $\vec{x}_2 = \langle -1, 1.5, 3, 4 \rangle$

3) $\vec{x}_3 = \langle -3, 1, 3, 0.5 \rangle$

b) Jak bychom museli *co nejméně* změnit některou váhu tak, aby byl výstup pro příklad $\vec{x}_4 = \langle -2, 1, 2.5, 3 \rangle$ roven 1?

a) 1) 1, protože $1 + 1 \cdot 0 + 2.5 \cdot 1 + 0 \cdot 1 + (-1) \cdot 2 = 1.5 > 0$
2) 0, protože $1 + (-1) + 3.75 + 0 + (-4) = -0.25 < 0$
3) 1, protože $1 + (-3) + 2.5 + 0 + (-0.5) = 0$. Tento bod je zajímavý tím, že leží přímo na „dělící hranici“ perceptronu. To má pěkný geometrický význam, který si stručně představíme v další sekci.

b) Pro původní váhy máme $1 + (-2) + 2.5 + 0 + (-3) = -1.5$. Potřebujeme aby byla vážená suma rovna (alespoň) 0. Nejmenší změna, kterou toho lze docílit, je změna w_4 z hodnoty -1 na -0.5. Zamyslete se, proč volíme právě w_4.

Příklad 10.3.1. Pro každou z následujících logických operací definujte perceptron, který ji implementuje. Jako množinu vstupů uvažujte pouze vektory nad $\{0, 1\}$.

a) binární NAND

b) binární implikace

c) n-ární disjunkce

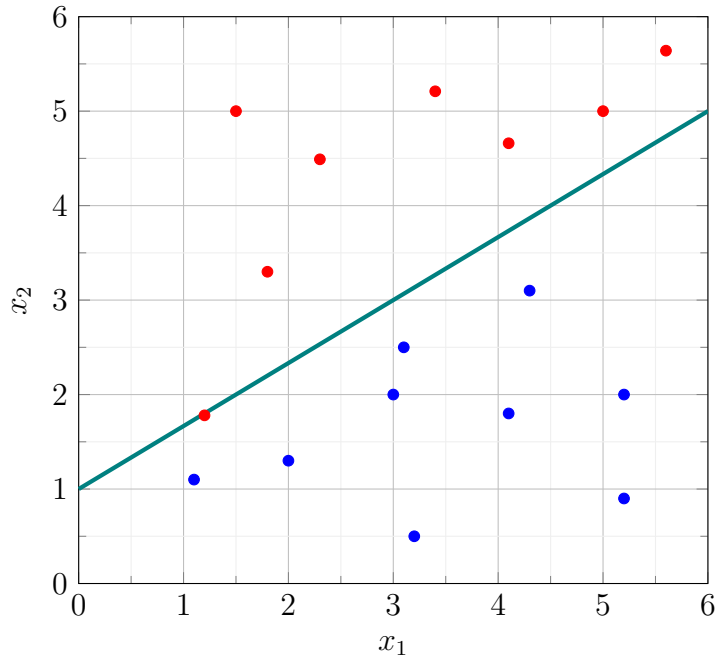
Příklad 10.3.2. Kolik Booleovských funkcí pro dva vstupy jde reprezentovat perceptronem?

10.4 Geometrický význam a perceptronový algoritmus

Perceptron má i svůj geometrický význam, je to takzvaný *lineární separátor*. Váhy perceptronu zadávají v prostoru nadrovinu (v dvourozměrném prostoru je to přímka), která rozděluje daný prostor vstupů na dvě části. Body z jedné části model klasifikuje jako 1, z druhé jako 0.

Definice 74: *Dělící nadrovina* daná perceptronem s váhami $\vec{w} = \langle w_0, w_1, \ldots, w_n \rangle$ je právě množina bodů $\{\vec{x} = \langle x_1, \ldots, x_n \rangle \mid w_0 + \sum_{i=1}^{n} w_i \cdot x_i = \vec{w} \cdot \tilde{x} = 0\}$.

Pro ilustraci uvažujme případ dvourozměrného prostoru, kde perceptron zadává oddělující přímku. Příklad grafické reprezentace takového klasifikátoru můžete vidět na následujícím obrázku (body jsou obarveny podle výsledku klasifikace).



Příklad. Uvažte model a data reprezentované předchozí ilustrací.

a) Kterou barvou jsou v obrázku vykresleny body oklasifikované jako 1?

b) Zamyslete se, kolik daný model potřebuje vah, a jaký je jejich geometrický význam.

c) Určete z obrázku hodnoty vah pro daný model, víme-li, že $w_0 = -3$.

a) Červeně. Z definice víme, že body na dělící přímce jsou oklasifikovány jako 1.

b) Celkem máme tři váhy $\vec{w} = \langle w_0, w_1, w_2 \rangle$. Váhy w_1, w_2 nám zadávají sklon přímky, zatímco w_0 ovlivňuje posun od počátku. Váze w_0 se proto také někdy říká *bias*.

c) Rovnice přímky je $w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 = 0$. Víme-li, že $w_0 = -3$, pak (pomocí bodů, kterými prochází) dostaneme $w_1 = -2$ a $w_2 = 3$.

Základní idea učení perceptronu je snaha manipulovat s dělící nadrovinou („otáčet ji“, resp. „posunovat“) tak, aby nakonec správně oddělovala trénovací příklady patřící do různých tříd.

Definice 75: Uvažujme tréninkový set $D = \{(\vec{x}_1, c_1), \ldots, (\vec{x}_p, c_p))\}$, kde c_i je opravdová třída příkladu $\vec{x}_i$. Perceptron C s váhami $\vec{w}$ je konzistentní s D pokud pro všechna k z $1, \ldots, p$ platí $C[\vec{w}](\vec{x}_k) = c_k$.

Definice 76: *Online perceptronový algoritmus* je procedura, která iterativně upravuje váhy perceptronu. Postupně prochází přes jednotlivé trénovací příklady, a kdykoliv je příklad špatně oklasifikován, pootočí nadrovinou tak, aby byl příklad blíže správnému poloprostoru.

Uvažujme tréninkový set $D = \{(\vec{x}_1, c_1), \ldots, (\vec{x}_p, c_p))\}$, kde c_i je opravdová třída příkladu $\vec{x}_i$. Sekvence vektorů vah $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$ je počítána následovně:

- $\vec{w}^{(0)}$ je inicializován náhodně hodnotami okolo 0,
- $\vec{w}^{(t+1)} = \vec{w}^{(t)} - \alpha \cdot (C[\vec{w}^{(t)}](\vec{x}_k) - c_k) \cdot \tilde{x}_k$,

kde $k = (t \mod p) + 1$, což vyjadřuje jen to, že příklady jsou uvažovány postupně cyklicky. Konstanta $0 < \alpha \leq 1$ je tzv. *učicí konstanta* (*learning rate*).

Pokud se nad algoritmem zamyslíte, pokaždé když narazí na špatně klasifikovaný vektor, tak jej přičte k aktuálnímu vektoru vah nebo ho od něj odečte (po pronásobení ϵ).

Příklad. Mějme trénovací množinu $D = \{(\langle -1, 0\rangle, 1), (\langle 0, 1\rangle, 1), (\langle 3, 0\rangle, 0)\}$. Proveďte 3 kroky perceptronového algoritmu, jestliže $\vec{w}^{(0)} = \langle 0, 1, -1\rangle$ a $\alpha = 1$.

- $\vec{w}^{(0)} = \langle 0, 1, -1\rangle$
- $\vec{w}^{(0)} \cdot \tilde{x}_1 = -1$ $\quad C[\vec{w}^{(0)}](\vec{x}_1) = 0$ $\quad \vec{w}^{(1)} = \vec{w}^{(0)} - (0 - 1) \cdot \tilde{x}_1 = \langle 1, 0, -1\rangle$
- $\vec{w}^{(1)} \cdot \tilde{x}_2 = 0$ $\quad C[\vec{w}^{(1)}](\vec{x}_2) = 1$ $\quad \vec{w}^{(2)} = \vec{w}^{(1)} = \langle 1, 0, -1\rangle$
- $\vec{w}^{(2)} \cdot \tilde{x}_3 = 1$ $\quad C[\vec{w}^{(2)}](\vec{x}_3) = 1$ $\quad \vec{w}^{(3)} = \vec{w}^{(2)} - (1 - 0) \cdot \tilde{x}_3 = \langle 0, -3, -1\rangle$

Příklad 10.4.1. Platí, že perceptronový algoritmus vždy zkonverguje a nalezne perceptron, který správně klasifikuje všechny příklady z datasetu? Odůvodněte svou odpověď.

Příklad 10.4.2. Mějme trénovací sadu $D = \{(\langle 3, -1\rangle, 1), (\langle 2, 1\rangle, 1), (\langle 0, 3\rangle, 0)\}$. Aplikujte perceptronový algoritmus, dokud nenalezne separující nadrovinu. Uvažujte $\vec{w}^{(0)} = \langle 0, -2, 1\rangle$ a $\alpha = 1$. Na závěr načrtněte dělící přímku.

Příklad 10.4.3. Mějme trénovací sadu $D = \{(\langle 1, 1\rangle, 1), (\langle 0, 1\rangle, 0), (\langle 0, 0\rangle, 1), (\langle 1, 0\rangle, 0)\}$. Uvažujte $\vec{w}^{(0)} = \langle 0, 0, 0\rangle$ a $\alpha = 1$. Kolik iterací perceptronového algoritmu je třeba, aby nalezl separující nadrovinu?

10.5 Základy lineární regrese

Myšlenku lineárních modelů můžeme samozřejmě využít i k řešení jiných úloh, než jen ke klasifikaci. Asi nejznámější úlohou je regrese, v níž se snažíme hledat funkční závislosti mezi numerickými vlastnostmi. V našem případě budeme uvažovat *lineární regresi* – cílem tedy bude lineárně aproximovat neznámou funkci. Jedná se v podstatě o prokládání trénovacích

dat „co nejlepším“ lineárním modelem. Díky tomu poté můžeme predikovat funkční hodnoty i pro dosud neviděné vstupy.

Definice 77: Uvažujme n-rozměrné příklady tvaru $\vec{x} = \langle x_1, \ldots, x_n \rangle$ a vektor vah $\vec{w} = \langle w_0, w_1, \ldots, w_n \rangle$. Regresní model R s váhami $\vec{w}$ je dán jako

$$R[\vec{w}](\vec{x}) = w_0 + \sum_{i=1}^{n} w_i \cdot x_i = \vec{w} \cdot \tilde{x}$$

To, jak je konkrétní model dobrý (resp. „špatný“), definujeme pomocí chybové funkce (*error function*). Obvykle chybová funkce nějakým způsobem vyjadřuje, jak daleko jsou původní funkční hodnoty příkladů od těch aproximovaných modelem. Idea je taková, že lepší model by měl mít menší chybu. Mezi nejznámější typy chyb patří například kvadratická chyba (*squared error*) nebo absolutní chyba (*absolute error*), a jejich varianty. V této sekci budeme nejčastěji pracovat s kvadratickou chybou.

Definice 78: Uvažujme dataset $D = \{(\vec{x}_1, f_1), \ldots, (\vec{x}_p, f_p))\}$, kde f_i je opravdová funkční hodnota příkladu $\vec{x}_i$. *Kvadratická chyba* pro model s váhami $\vec{w} = \langle w_0, w_1, \ldots, w_n \rangle$ je

$$E(\vec{w}) = \tfrac{1}{2} \cdot \sum_{k=1}^{p} (R[\vec{w}](\vec{x}_k) - f_k)^2 = \tfrac{1}{2} \cdot \sum_{k=1}^{p} (\vec{w} \cdot \tilde{x}_k - f_k)^2$$

Příklad. Mějme dataset jednorozměrných příkladů a jejich očekávaných funčních hodnot $D = \{(43, 41), (44, 45), (45, 49), (46, 47), (47, 44)\}$ a lineární model R s váhami $\vec{w} = \langle 9.2, 0.8 \rangle$.

a) Spočítejte pro R kvadratickou chybu na množině D.

b) Určete pomocí modelu R aproximaci funkční hodnoty pro $x = 40$.

a) Spočítejme si nejdříve kvadratickou chybu pro jednotlivé trénovací příklady.

$$(9.2 + 0.8 \cdot 43 - 41)^2 = 6.76$$
$$(9.2 + 0.8 \cdot 44 - 45)^2 = 0.36$$
$$(9.2 + 0.8 \cdot 45 - 49)^2 = 14.44$$
$$(9.2 + 0.8 \cdot 46 - 47)^2 = 1$$
$$(9.2 + 0.8 \cdot 47 - 44)^2 = 7.84$$

$E(\vec{w}) = \frac{1}{2} \cdot (6.76 + 0.36 + 14.44 + 1 + 7.84) = 15.2$

b) $R[\vec{w}](40) = 9.2 + 0.8 \cdot 40 = 41.2$

Učení regresního modelu je založeno na trochu jiném principu než u perceptronu. Jelikož máme chybovou funkci, můžeme hledat takové váhy, které minimalizují její hodnotu. Řešíme tedy jednoduchý optimalizační problém. Algoritmus, který se k tomu obvykle používá již nejspíše znáte – jedná se o gradientní sestup (*gradient descent*). Tato metoda pomocí odečítání gradientu chybové funkce postupně upravuje váhy tak, aby docílila minimální chyby. Podrobněji se s ní seznámíte buď v následující kapitole, nebo v předmětu IB031.

Příklad 10.5.1. Mějme množinu jednorozměrných příkladů a jejich očekávaných funčních hodnot $D = \{(1, 0), (4, 2), (5, 3), (6, 3)\}$ a lineární model R s váhami $\vec{w} = \langle \frac{2}{3}, \frac{1}{3} \rangle$.

a) Spočítejte pro R kvadratickou chybu na množině D.

b) Nalezněte nějaký model, který bude mít pro množinu D menší kvadratickou chybu než R.

Příklad 10.5.2. Mějme množinu se dvěma příklady $D = \{(\langle 3, 5 \rangle, 13), (\langle 6, 8 \rangle, 22)\}$ a lineární model R s váhami $\vec{w} = \langle 3, 1, 2 \rangle$.

a) Spočítejte pro R kvadratickou chybu na množině D.

b) Aproximujte (predikujte) pomocí modelu R funkční hodnotu pro $\vec{x} = \langle 4, 7 \rangle$.

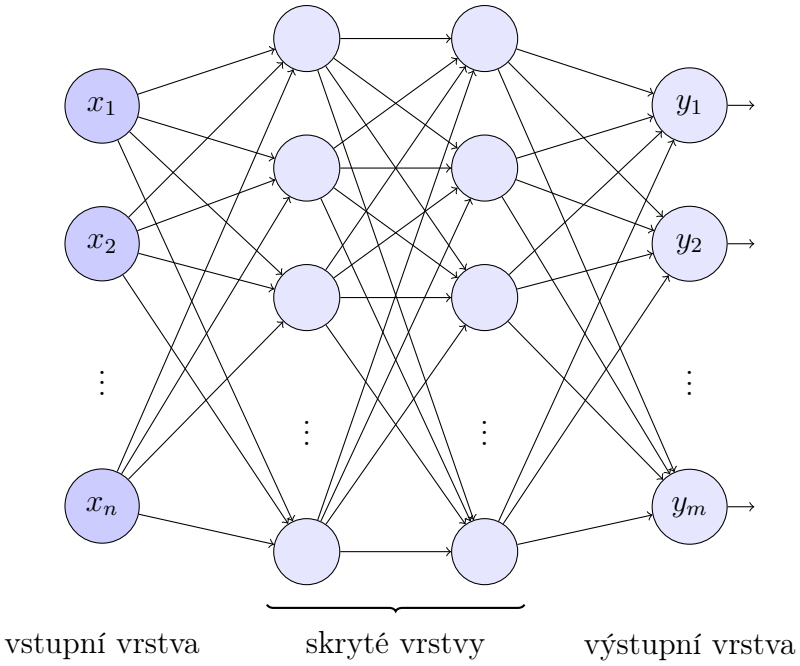
c) Dokážete najít nějaký model minimalizují na D kvadratickou chybu?

11 Neuronové sítě a hluboké učení

V předchozí kapitole byla poprvé představena jednoduchá jednotka umožňující lineární klasifikaci – perceptron. Sdružením více takových jednotek do vrstev a jejich následným propojením pomocí vazeb vznikají *neuronové sítě*, hlavní předmět této kapitoly.

11.1 Struktura a výpočet neuronové sítě

Základní strukturu dopředné neuronové sítě ilustruje následující obrázek.



Na obrázku je znázorněna neuronová síť s n vstupy, 2 skrytými vrstvami a m výstupy. Jednotlivé vrstvy jsou úplně propojené.

Výpočet dopředné neuronové sítě na zadaném vstupu probíhá po vrstvách od vstupní vrstvy k výstupní. Nejprve se jednotky vstupní vrstvy inicializují hodnotami vstupu $\vec{x}$. Na základě hodnot vah mezi vstupní vrstvou a první skrytou vrstvou se spočítají výstupy jednotlivých jednotek skryté vrstvy. Tento proces se postupně aplikuje na všechny vrstvy, dokud nedojde k dosažení vrstvy výstupní. Výstupy jednotek výstupní vrstvy se pak berou jako výstup neuronové sítě $\vec{y}$ na vstupu $\vec{x}$.

Definice 79: *Vážený součet vstupů jednotky i* (též *vnitřní potenciál jednotky i*) je dán jako

$$\xi_i = -w_{0,i} + \sum_j w_{j,i} \cdot a_j,$$

přičemž j iteruje přes všechny jednotky předcházející vrstvy, $w_{0,i}$ je prahová váha jednotky i, $w_{j,i}$ je váha vazby z jednotky j do jednotky i a a_j je výstup jednotky j.

Definice 80: *Výstup jednotky i* je definován vztahem

$$a_i = g_i(\xi_i),$$

kde g_i je aktivační funkce jednotky i.

Příklad. Spočítejte výstup jednotky 4 v druhé skryté vrstvě dopředné neuronové sítě, obsahuje-li první skrytá vrstva jednotky 1, 2, jejichž výstupy jsou $a_1 = 0,84$, $a_2 = 0,02$. Váhy hran v síti jsou $w_{1,4} = -0,15$, $w_{2,4} = 12,4$, prahová váha jednotky 4 je $w_{0,4} = 3,8$. Jednotka 4 používá aktivační funkci sigmoida $g_4(x) = \frac{1}{1+e^{-x}}$.

Vážená váha vstupů po odečtení prahové váhy je

$$-w_{0,4} + w_{1,4} \cdot a_1 + w_{2,4} \cdot a_2 = -3,68,$$

a výstup jednotky je

$$a_4 = g_4(-3,68) = 0,025.$$

Lze tedy konstatovat, že její reakce na zadané vstupy je poněkud vlažná.

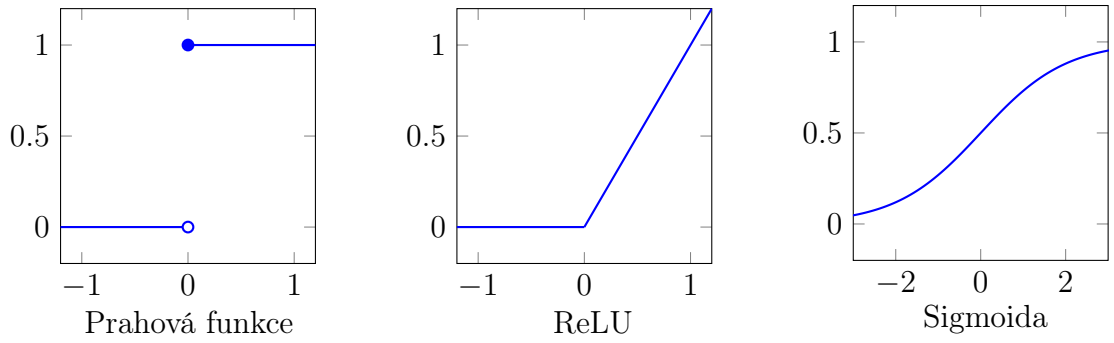
Pro jaké výstupy jednotek 1, 2 dochází k aktivaci jednotky 4?

V rámci neuronových sítí se používají různé aktivační funkce. Jak uvidíme dále, důležitým požadavkem na aktivační funkci je, aby nebyla lineární; pokud navíc chceme neuronovou síť učit, je vhodné, aby bylo možné aktivační funkci derivovat. Některé z používaných aktivačních funkcí uvádí následující definice.

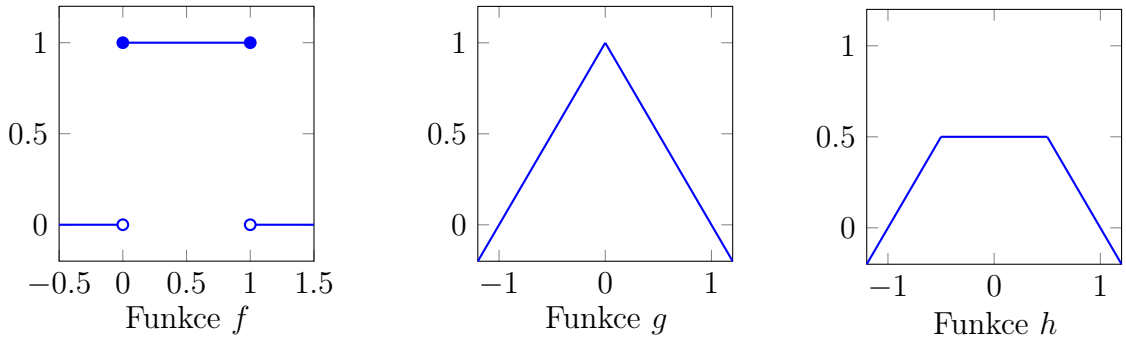
Definice 81: *Aktivační funkce.*

- *Prahová funkce*
$$f(x) = \begin{cases} 1 & x \geq 0, \\ 0 & \text{jinak.} \end{cases}$$
- *ReLU*
$$f(x) = \begin{cases} x & x \geq 0, \\ 0 & \text{jinak.} \end{cases}$$
- *Sigmoida*
$$\sigma(x) = \frac{1}{1+e^{-x}}.$$

Grafy aktivačních funkcí jsou znázorněny na následujících obrázcích.



Příklad 11.1.1. Uvažte následující funkce f, g a h.



Aplikací operací skládání funkcí, násobení konstantou, přičítání konstanty a sčítání

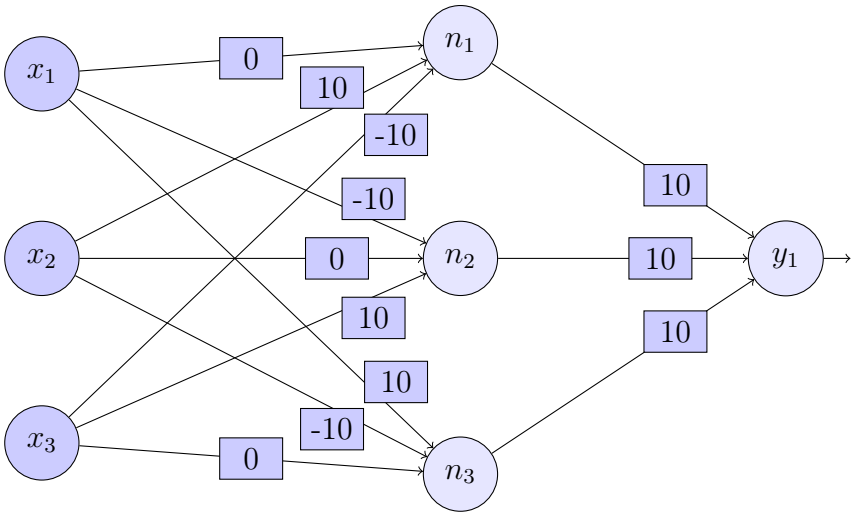
a) a pomocí prahové funkce vyjádřete funkci f,

b) a pomocí ReLU vyjádřete funkci g,

c) a pomocí ReLU vyjádřete funkci h.

Příklad 11.1.2. Uvažte funkci $F : [0, 10] \to \mathbb{R}$. Navrhněte způsob, jak lze pomocí prahové funkce a s využitím operací skládání funkcí, násobení konstantou, přičítání konstanty a sčítání aproximovat funkci F. Požadavek na aproximaci je, aby pro aproximující funkci A platilo, že $A(x) = F(\tilde{x})$ pro nějaké $\tilde{x}$ takové, že $|\tilde{x} - x| \leq 1$, tedy hodnota aproximace se rovná funkční hodnotě v nějakém bodě, který se liší od x maximálně o 1.

Příklad 11.1.3. Uvažujte následující neuronovou síť se vstupy x_1, x_2, x_3, skrytou vrstvou s jednotkami n_1, n_2, n_3 a výstupní vrstvou s jednotkou y_1.



Čísla na obrázku znázorňují hodnoty vah jednotlivých vazeb. Aktivační funkce všech jednotek je sigmoida σ a prahová váha je 5.

a) Jaké budou výstupy sítě pro vstupy $(0,0,0)$, $(1,0,1)$, $(1,1,1)$, $(-1,0,1)$, $(0.5,0.4,0.6)$?

b) Napište jednoduchý program, který pro zadaný vstup spočítá výstup sítě.

c) Co síť počítá? Nalezněte co nejstručnější slovní charakteristiku.

d) Stačila by pro takový výpočet neuronová síť bez skrytých vrstev?

Příklad 11.1.4. Dokažte, že každou booleovskou funkci F, tj. funkci tvaru $F : \{0,1\}^n \to \{0,1\}$, lze vyjádřit pomocí neuronové sítě s jednou skrytou vrstvou, kde každý neuron má prahovou funkci jako aktivační funkci.

Příklad 11.1.5. Uvažujte vícevrstvou neuronovou síť s jedním výstupem, kde každá vnitřní jednotka i má aktivační funkci $f_i(x) = A_i x + B_i$ (kde A_i, B_i jsou reálné konstanty) a výstupní jednotka je aktivována prahovou funkcí.

a) Má neuronová síť stejnou vyjadřovací sílu jako neuronová síť, která pro vnitřní jednotky používá aktivační funkci σ (sigmoidu)? Vyjadřovací silou rozumíme, jaké různé funkce lze neuronovou sítí reprezentovat. Zdůvodněte.

b) Má neuronová síť větší vyjadřovací sílu než perceptron? Pokud ano, nalezněte příklad funkce, kterou lze sítí vyjádřit, ale perceptronem ne. Pokud ne, ukažte, jak takovou síť převést na perceptron počítající stejnou funkci.

11.2 Učení neuronové sítě

V této sekci pronikneme hlouběji do samotné podstaty vícevrstvých neuronových sítí. Konkrétně se zaměříme na *algoritmus zpětného šíření chyby*, který umožňuje trénování neuronových sítí na konkrétní sadě dat.

Učicí sada obsahuje dvojice tvaru

$$(\vec{x}, \vec{y}_{\text{exp}}),$$

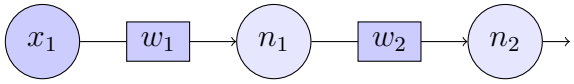
kde $\vec{x}$ je vstup neuronové sítě a $\vec{y}_{\text{exp}}$ je očekávaný výstup. Učení probíhá na principu úpravy hodnot vah v síti, aby došlo k minimalizaci chybové funkce

$$E = E(\vec{y}, \vec{y}_{\text{exp}}),$$

kde $\vec{y}$ je skutečný výstup sítě při vstupu $\vec{x}$. Chybová funkce musí být navržena tak, aby její minima odpovídala minimálnímu rozdílu mezi skutečným a očekávaným výstupem.

Příkladem chybové funkce je kvadratická chyba, kterou jsme představili v předchozí kapitole. Pro použití v neuronových sítích je třeba definici mírně upravit, aby zvládla pojmout vícerozměrný výstup.

Příklad. Uvažujte následující jednoduchou neuronovou síť s jednou skrytou vrstvou.



Symboly ξ_1, ξ_2 označují vážené součty vstupů jednotky 1, resp. 2; symboly a_1, a_2 pak jejich výstupy a g_1, g_2 jejich aktivační funkce. Chybová funkce je

$$E = E(y, y_{\text{exp}}) = E(a_2, y_{\text{exp}}).$$

a) Spočítejte parciální derivace $\partial E/\partial w_2$ a $\partial E/\partial w_1$.

b) Jaký je význam spočítaných parciálních derivací při učení neuronové sítě?

a) Při výpočtu využijeme toho, že platí

$$a_2 = g_2(\xi_2) \qquad \text{a} \qquad \xi_2 = a_1 \cdot w_2,$$

což plyne z definice výstupu jednotky, resp. definice váženého součtu vstupů jednotky. Jelikož E je funkcí a_2 a a_2 je funkcí ξ_2, lze s využitím řetězového pravidla rozepsat

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial a_2}\frac{\partial a_2}{\partial w_2} = \frac{\partial E}{\partial a_2}\frac{\partial a_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial w_2}.$$

S využitím vztahů výše můžeme rozepsat poslední dva součinitele jako

$$\frac{\partial a_2}{\partial \xi_2} = g_2' \qquad \text{a} \qquad \frac{\partial \xi_2}{\partial w_2} = a_1$$

a celkově získáme

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial a_2} \cdot g_2' \cdot a_1 = \frac{\partial E}{\partial y} \cdot g_2' \cdot a_1.$$

Jedlotlivé členy lze v konkrétním případě již získat snadno: parciální derivaci chybové funkce podle skutečného výstupu sítě lze spočítat z konkrétní podoby chybové

funkce, derivaci aktivační funkce spočítáme opět z jejího tvaru (proto je dobré, aby měla derivaci, jak bylo uvedeno dříve při zavádění pojmu aktivační funkce) a hodnotu a_1 získáme prostým provedením výpočtu sítě nad daným vstupem.

Obdobným způsobem spočítáme

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial a_2}\frac{\partial a_2}{\partial w_1} = \frac{\partial E}{\partial a_2}\frac{\partial a_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial w_1}.$$

Jelikož $\xi_2 = a_1 \cdot w_2$ a a_1 dále závisí na w_1, pokračujeme ve výpočtu:

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial a_2}\frac{\partial a_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial a_1}\frac{\partial a_1}{\partial w_1} = \frac{\partial E}{\partial a_2}\frac{\partial a_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial a_1}\frac{\partial a_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial w_1} = \frac{\partial E}{\partial a_2} \cdot g_2' \cdot w_2 \cdot g_1' \cdot x_1.$$

b) Cílem učicího algoritmu je pomocí úprav vah vazeb v síti minimalizovat chybu, tj. rozdíl mezi výstupy, které neuronová síť dává, a očekávanými výstupy. Parciální derivace chybové funkce podle váhy vazby vyjadřuje míru, s jakou se mění velikost chyby při změně váhy.

Je-li například $\partial E/\partial w_1 = 2$, znamená to, že při zvýšení váhy w_1 o 1 se chyba E zvýší přibližně o 2. V takovém případě tedy chceme váhu snížit, což povede ke snížení chyby.

Učicí algoritmus tedy v každé iteraci upraví váhy podle předpisu

$$(w_1, w_2) \leftarrow (w_1, w_2) - \alpha \cdot \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}\right),$$

kde $\alpha > 0$ je tzv. *učicí konstanta* (*learning rate*), která reguluje rychlost učení a s níž jsme se setkali již v předchozí kapitole.

Vektor

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right)$$

se nazývá *gradient chyby* a vyjadřuje, kterým směrem chyba nejrychleji roste. Při postupu opačným směrem, tedy odečtením násobku gradientu od vah sítě, dosahuje síť snížení chyby.

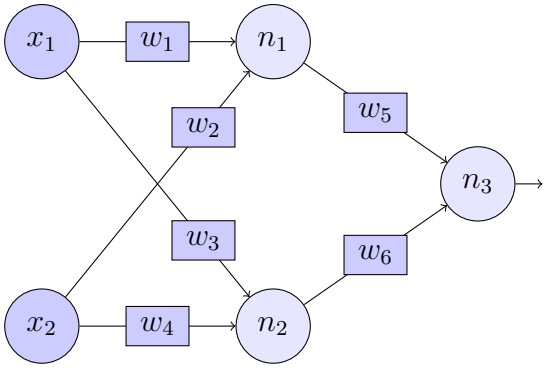
Definice 82: *Online algoritmus pro učení neuronové sítě* je procedura, která iterativně upravuje váhy v neuronové síti. Postupně prochází přes jednotlivé trénovací příklady a odečte od hodnot vah o gradient chyby pronásobený učicí konstantou.

Uvažujme tréninkový set $D = \{(\vec{x}_1, \vec{y}_1), \ldots, (\vec{x}_p, \vec{y}_p))\}$, kde $\vec{y}_i$ je očekávaný výstup při vstupu $\vec{x}_i$. Sekvence vektorů vah $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$ je počítána následovně:

- $\vec{w}^{(0)}$ je inicializován náhodně hodnotami okolo 0,
- $\vec{w}^{(t+1)} = \vec{w}^{(t)} - \alpha \cdot \nabla E(\vec{y}^{(t)}, \vec{y}_k)$,

kde $k = (t \mod p) + 1$ a $\vec{y}^{(t)}$ je výsledek vrácený sítí v iteraci t na vstupu $\vec{y}_k$.

Příklad 11.2.1. Uvažujte následující neuronovou síť s jednou skrytou vrstvou.



Symboly ξ_1, ξ_2, ξ_3 označují vážené součty vstupů jednotky 1, 2, resp. 3; symboly a_1, a_2, a_3 pak jejich výstupy a g_1, g_2, g_3 jejich aktivační funkce. Prahové váhy neuvažujeme. Chybová funkce je

$$E = E(y, y_{\text{exp}}) = E(a_3, y_{\text{exp}}).$$

a) Spočítejte $\partial E/\partial w_5$.

b) Spočítejte $\partial E/\partial w_1$.

c) Jaký je obecný předpis pro $\partial E/\partial w_i$ v obecné dopředné neuronové síti?

Příklad 11.2.2. Nalezněte derivace následujících aktivačních funkcí:

a) prahová funkce,

b) ReLU,

c) sigmoida.

Příklad 11.2.3. Diskutujte význam učicí konstanty α při učení neuronové sítě. Jaký má vliv na průběh učení (pozitivní i negativní), je-li hodnota konstanty velmi velká, resp. velmi malá?

11.3 Zpracování obrazu a klasifikace

Při klasifikaci do n kategorií je vhodné získat výstup ve formě n-tice pravděpodobností. K tomu slouží funkce *softmax*, která zvýrazní rozdíly mezi jednotlivými výstupy pomocí exponenciální funkce a zároveň provede normalizaci, aby součet členů n-tice byl roven jedné.

Definice 83: Funkce *softmax* transformuje n-tici reálných čísel na n-tici pravděpodobností následujícím způsobem

$$\text{softmax}(x_1, \ldots, x_n) = \Big(\frac{e^{x_1}}{\sum_{i=1}^n e^{x_i}}, \frac{e^{x_2}}{\sum_{i=1}^n e^{x_i}}, \ldots, \frac{e^{x_n}}{\sum_{i=1}^n e^{x_i}}\Big).$$

Zamyslete se, zda může po aplikaci funkce softmax vyjít u nějaké kategorie nulová pravděpodobnost.

Příklad 11.3.1.

a) Bez použití kalkulačky odhaděte výsledek aplikace funkce softmax na následující vstupy: $(1, 3, 2)$, $(1, 1, 1)$, $(2, 0, 1)$, $(-5, 0, 0)$.

b) Odhad z předchozí části ověřte výpočtem.

12 Zpracování přirozeného jazyka

Na jazyky formální i přirozené nahlížíme jako na množiny platných vět. Na rozdíl od formálních jazyků (připomeňme třeba jazyk formulí výrokové logiky) s sebou však jazyky přirozené přinášejí řadu komplikací.

Je například poměrně jednoduché určit, že $p \vee \vee\vee$ není platnou formulí výrokové logiky, protože nevyhovuje definici. Je ovšem věta „To byl teda krutopřísnej vejlet.“ platnou větou češtiny? Odpověď není zcela jednoznačná a s ohledem na časový vývoj jazyka a nářeční nebo stylistickou rozmanitost přirozeného jazyka nelze ani přesně definovat, co vše do jazyka ještě patří a co už ne.

Slova přirozeného jazyka mohou být víceznačná – např. slovo *list* může znamenat kus papíru či část rostliny. Přirozený jazyk je navíc nejasný – kdo je činitelem ve větě „Přišel.“? Existuje navíc mnoho dalších komplikací (viz přednášku), které ze zpracování přirozeného jazyka činí nelehkou úlohu.

V této kapitole jsou představeny některé z metod, které se při zpracování přirozeného jazyka používají.

12.1 Předzpracování dat

Textová data, se kterými potřebujeme pracovat, jsou mnohdy dostupná pouze ve velmi neotesané podobě, např. ve formě webových stránek či hrubého textu s nadbytečnými údaji. Cílem předzpracování může být očištění dat od nadbytečných informací (např. různých anotací), jejich segmentace či normalizace, tedy celkově převedení dat do formy, na niž lze již aplikovat samotné metody zpracování přirozeného jazyka.

Příklad 12.1.1. Na adrese `https://web2.mlp.cz/koweb/00/04/34/55/03/bila_nemoc.html` si stáhněte text knihy Karla Čapka Bílá nemoc ve formátu HTML. Formát HTML je textový formát, který obsahuje značky pro internetový prohlížeč, které umožňují dokument zobrazit ve formě webové stránky.

a) Napište skript, který z dokumentu extrahuje pouze text díla, konkrétně ve formě seznamu jednotlivých replik jako řetězců. Výstup tedy bude začínat následujícím způsobem.

```
["Mor je to, mor.  V naší ulici už je v každém domě ...",
 "Žádný mor, malomocenství.  Bílá nemoc tomu říkají,...",
 "Kriste panebože - Kriste panebože - Kriste panebože -",
 ...
]
```

Ve výsledných řetězcích je vypuštěna informace o konkrétní postavě, která danou repliku říká, neboť se jedná o značně repetitivní informaci.

b) Přidejte do skriptu funkce, které provádějí segmentaci textu na věty, resp. tokenizaci na jednotlivá slova. Slova získávejte očištěná od interpunkčních znamének.

c) Vytvořte slovník s frekvencemi jednotlivých slov v díle. Je vhodné mít metodu, která pro každé slovo vrátí jeho základní tvar. Např. slovo „ale“ se v textu může vyskytovat i ve formách „Ale“ nebo „ALE“, ale všechny tyto tvary by se ve slovníku měly započítat pod položku „ale“.

d) Napište funkci, která vypíše seznam n nejčastějších slov, která se v díle (nebo jiném zadaném textu) vyskytují.

12.2 Gramatiky

Gramatiky představují jeden ze způsobů, kterým lze popsat jazyk. Gramatika udává předpis, jakým lze z počátečního symbolu, tzv. kořene gramatiky, postupnou aplikací pravidel gramatiky odvodit platné věty jazyka gramatiky. Speciálním typem gramatik jsou gramatiky bezkontextové, kterým budeme v této sekci věnovat speciální pozornost.

Definice 84: *Bezkontextová gramatika* sestává z

- množiny terminálních symbolů (*slov* jazyka),
- množiny neterminálních symbolů (*syntaktických kategorií*),
- speciálního neterminálního symbolu S reprezentujícího celou větu jazyka (též nazýván *kořen gramatiky*),
- souboru *přepisovacích pravidel* tvaru

$$\text{neterminál} \to \text{libovolný řetězec}.$$

Běžná konvence je psát neterminály s prvním písmenem velkým a veškerá pravidla, která vycházejí ze stejného neterminálu sdružit pomocí symbolu svislítka, tj. např.

$$S \to abbb,$$
$$S \to cac$$

píšeme

$$S \to abbb \mid cac.$$

Gramatika definuje jazyk jako množinu vět sestávajících pouze ze slov jazyka, které lze vygenerovat z kořene gramatiky S postupnou aplikací pravidel gramatiky. Odvození věty v gramatice navíc definuje tzv. *syntaktický strom*. Samotná pravidla gramatiky se často rozdělují na pravidla (pouze mezi syntaktickými kategoriemi) a lexikon (seznamy slov příslušících jednotlivým kategoriím).

Příklad. Uvažte gramatiku s následujícími pravidly:

$$\begin{aligned} S &\to NP\ VP, \\ NP &\to Noun \mid Adj\ NP, \\ VP &\to Verb \end{aligned}$$

a následujícím lexikonem:

$$\begin{aligned} Noun &\to \text{dítě} \mid \text{člověk} \mid \text{kapsa}, \\ Adj &\to \text{starý} \mid \text{cestující} \mid \text{nové}, \\ Verb &\to \text{píše} \mid \text{sedí} \mid \text{mluví}. \end{aligned}$$

a) Rozhodněte, které z následujících vět lze v gramatice vygenerovat.

1. „cestující sedí“
2. „nové nové kapsa píše“
3. „starý člověk mluví“

b) Nalezněte odvození vět v gramatice.

c) Nalezněte ke každému odvození odpovídající syntaktický strom.

a) Gramatika generuje věty, které začínají jmennou frází (*NP*, neboli *noun phrase*), po níž bezprostředně následuje slovesná fráze (*VP*, neboli *verb phrase*). Jmenná fráze sestává z podstatného jména (*Noun*), jemuž předchází libovolný počet přídavných jmen (*Adj*). Slovesná fráze sestává z jediného slovesa.

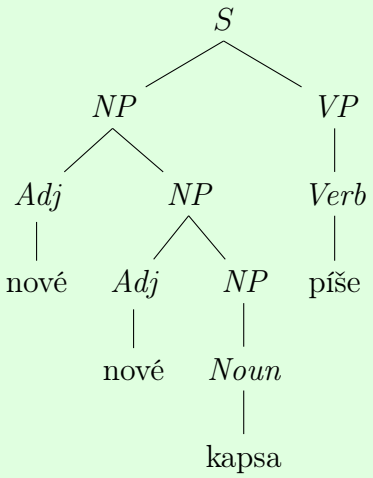
Větu 1 nelze v gramatice vygenerovat, jelikož neobsahuje podstatné jméno. Věty 2 i 3 splňují popis výše a v gramatice je vygenerovat lze.

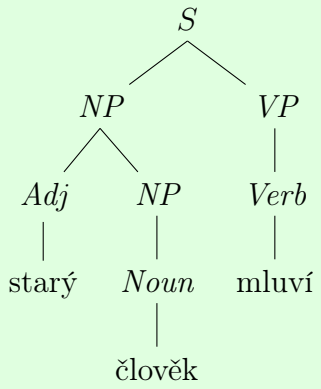
b)

$$\begin{aligned} S &\Rightarrow NP\ VP \\ &\Rightarrow Adj\ NP\ VP \\ &\Rightarrow \text{nové}\ NP\ VP \\ &\Rightarrow \text{nové}\ Adj\ NP\ VP \\ &\Rightarrow \text{nové nové}\ NP\ VP \\ &\Rightarrow \text{nové nové}\ Noun\ VP \\ &\Rightarrow \text{nové nové kapsa}\ VP \\ &\Rightarrow \text{nové nové kapsa}\ Verb \\ &\Rightarrow \text{nové nové kapsa píše} \end{aligned}$$

$$\begin{aligned} S &\Rightarrow NP\ VP \\ &\Rightarrow Adj\ NP\ VP \\ &\Rightarrow \text{starý}\ NP\ VP \\ &\Rightarrow \text{starý}\ Noun\ VP \\ &\Rightarrow \text{starý člověk}\ VP \\ &\Rightarrow \text{starý člověk}\ Verb \\ &\Rightarrow \text{starý člověk mluví} \end{aligned}$$

c) Syntaktické stromy uvedených vět vypadají následovně.





Z příkladu lze vypozorovat několik skutečností. Zaprvé, mohou existovat věty přirozeného jazyka, které nejsou gramatikou popsány – viz věta 1. Zadruhé, gramatika může generovat věty, které do popisovaného přirozeného jazyka nepatří – viz věta 2, kde je porušena shoda v rodě mezi podstatným jménem *kapsa* (ženský rod) a přídavným jménem *nové* (střední rod).

Cílem návrhu gramatik je minimalizovat počet vět, které gramatika generuje a nepatří do jazyka, a zároveň maximalizovat počet vět, které gramatika generuje a do jazyka patří. Kvalitu gramatiky kvantitativně popisují pojmy *pokrytí* a *přesnost*.

Definice 85: Uvažujme zamýšlený jazyk L a gramatiku G generující jazyk $L(G)$.

- *Pokrytí* gramatiky G je $\frac{|L \cap L(G)|}{|L|}$, tedy podíl vět jazyka L, které lze v gramatice vygenerovat.
- *Přesnost* gramatiky G je $\frac{|L \cap L(G)|}{|L(G)|}$, tedy podíl vět generovaných gramatikou, které patří do zamýšleného jazyka.

Příklad 12.2.1. Uvažte gramatiku s následujícími pravidly:

$$\begin{aligned} S &\to NP\ VP \mid VP, \\ NP &\to Noun \mid NP\ Conj\ NP, \\ VP &\to NP\ Esse \end{aligned}$$

a následujícím lexikonem:

$$\begin{aligned} Noun &\to \text{Romulus} \mid \text{Remus} \mid \text{Danubius} \mid \text{fratellus} \mid \text{fratelli} \mid \text{fluvius}, \\ Esse &\to \text{sum} \mid \text{est} \mid \text{sunt} \mid \text{eram} \mid \text{erat} \mid \text{erant}, \\ Conj &\to \text{et}. \end{aligned}$$

a) Rozhodněte, které z následujících vět lze v gramatice vygenerovat.

   1. „Romulus et Remus fratelli erant“
   2. „Remus et Danubius et Romulus sum“
   3. „Danubius est fluvius“

b) Nalezněte ke každé větě její syntaktický strom, pokud existuje.

Příklad 12.2.2. Navrhněte gramatiku pro jednoduché české (resp. slovenské) věty v minulém čase, které dodržují následující schéma:

[určení času / podmět], přísudek v minulém čase, [předmět],

tedy například „včera jsme koupili maso“, „já jsem běžel“, „hráli jste hru“. Nezapomeňte, že pořadí pomocného slovesa *být* a významového slovesa závisí na přítomnosti jiného slova na začátku věty (věta nikdy nezačíná pomocným slovesem). Ošetřete shodu podmětu s přísudkem

a) přímo návrhem gramatiky,

b) přídáním (jednoduše implementovatelných) testů na shodu.

Příklad 12.2.3. Uvažme gramatiku G s následujícími pravidly

$$\begin{aligned} &S \to AAA \mid BA \mid AB \mid C, \\ &A \to a, \\ &B \to bA \mid Ab, \\ &C \to BA \mid AB \mid cB \mid Bc \end{aligned}$$

a jazyk L vět délky 3 sestavených ze slov a, b, c, které obsahují slovo b právě jednou.

a) Je gramatika jednoznačná, neboli existuje pro každou věta jazyka $L(G)$ odvoditelnou v G právě jeden syntaktický strom?

b) Jaké je pokrytí gramatiky G vzhledem k zamýšlenému jazyku L?

c) Jaká je přesnost gramatiky G vzhledem k zamýšlenému jazyku L?

12.3 Syntaktická analýza

Gramatiku popisující jazyk lze použít k provádění *syntaktické analýzy*. Výsledkem syntaktické analýzy je syntaktický strom, který reflektuje strukturu věty a může sloužit jako podklad pro další zpracování.

Příklad 12.3.1. Knihovna NLTK (ze cvičení) umožňuje provádění metod zpracování přirozeného jazyka jako předzpracování textu, značkování či syntaktickou analýzu. Napište gramatiku pro knihovnu NLTK, která bude schopná rozeznat slovesný čas vět v anglickém jazyce. Značky, které slovům NLTK přidělí, nesou vedle slovesného druhu i informaci o tvaru slovesa (gerundium, minulý čas apod.), které lze za tímto účelem využít. Gramatika by měla umět určit alespoň následující časy.

I speak	*present simple*
You are speaking	*present continuous*
He has spoken	*present perfect*
We spoke	*past simple*
They were speaking	*past continuous*
I had spoken	*past perfect*