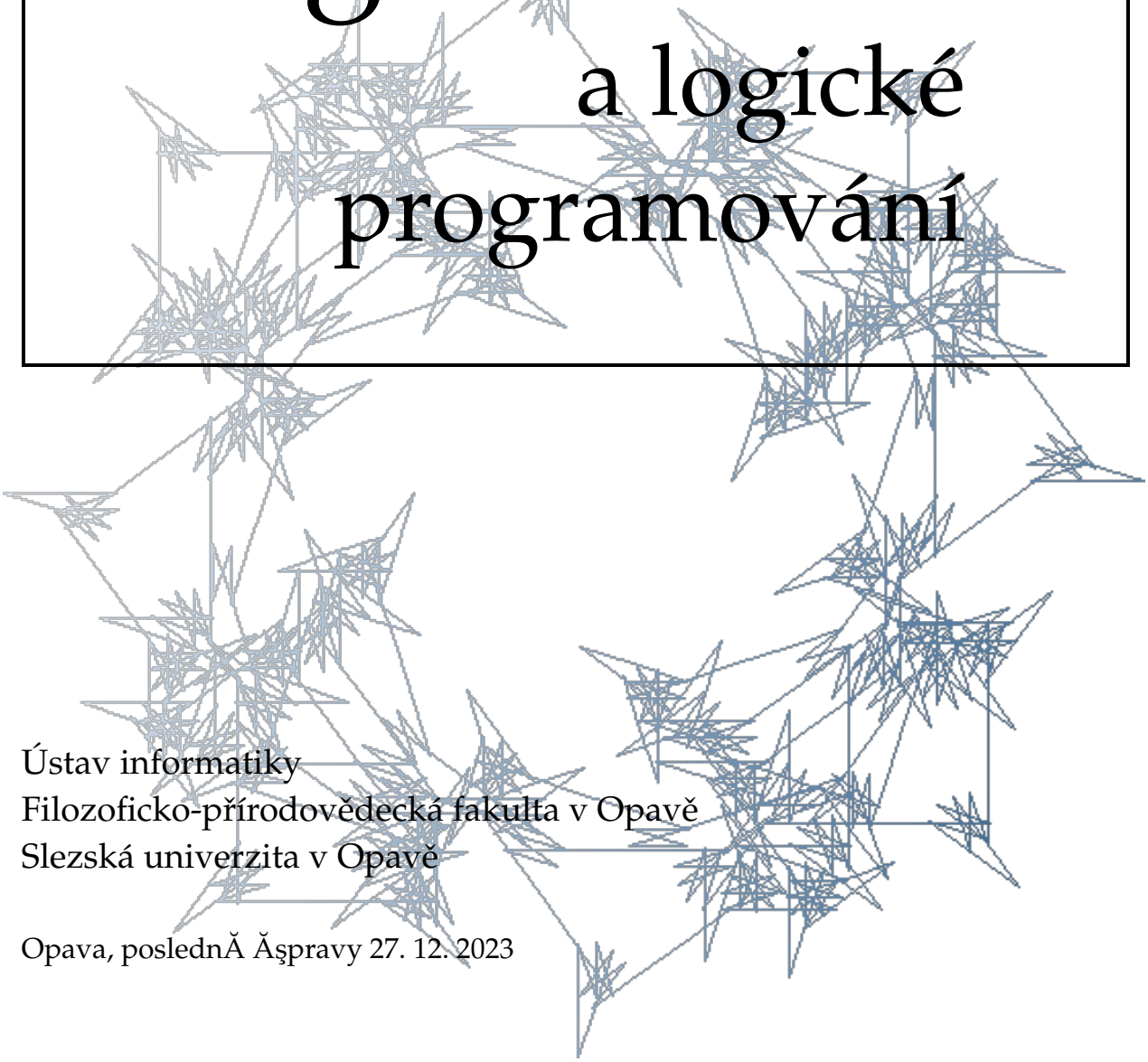
SLEZSKÁ
UNIVERZITA
FILOZOFICKO-
PŘÍRODOVĚDECKÁ
FAKULTA V OPAVĚ

Šárka Vavrečková

Skripta do předmětu

Logika a logické programování



Ústav informatiky
Filozoficko-přírodovědecká fakulta v Opavě
Slezská univerzita v Opavě

Opava, poslednĂ Ăşpravy 27. 12. 2023

Anotace: Tento dokument je určen pro studenty předmětu *Logika a logické programování* vyučovaného na Ústavu informatiky Slezské univerzity v Opavě. První část dokumentu je spíše teoretická, cílem je orientovat se v teoretických základech vědy (zde navazujeme na předmět *Úvod do logiky*). V druhé části dokumentu se dostáváme k základům a principům logického programování, zaměříme se na programovací jazyk *Prolog*.

Logika a logické programování

RNDr. Šárka Vavrečková, Ph.D.

Ústav informatiky
Filozoficko-přírodovědecká fakulta v Opavě
Slezská univerzita v Opavě
Bezručovo nám. 13, Opava

Sázeno v systému LaTeX

Předmluva

Co najdeme v těchto skriptech

 *Rychlý náhled:* Tento text je určen studentům předmětu *Logika a logické programování* na Ústavu informatiky Slezské univerzity v Opavě. Předpokládá znalosti z předmětu *Úvod do logiky*, základní pojmy a metody přednášené v tomto úvodním předmětu jsou připomenuty v kapitole 1.

Předmět Logika a logické programování navštěvují především studenti informatiky a matematiky, proto se zde budeme zabývat logikou spíše matematickou než filozofickou. Protože základy logiky již máme zvládnuté, text nás povede spíše k implementacím logiky ve vědě a především v informatice, konkrétně k logickému programování využívajícímu princip rezoluce.

Existuje mnoho knih, skript a článků s touto tématikou, a to různě orientovaných. Účelem tohoto textu rozhodně není pokrýt celou problematiku, mnohé důkazy nejsou uvedeny celé, některé z vět jsou zde vysloveny pouze s náznakem důkazu (týká se triviálních důkazů patřících do předchozího kurzu, ale i příliš složitých důkazů). Pro hlubší seznámení s problematikou proto odkazuji na literaturu uvedenou na konci knihy před přílohami. Z důvodu názornosti je výklad doprovázen mnoha příklady a některé problémy jsou také na příkladech vysvětlovány.

Některé oblasti jsou „navíc“ (jsou označeny ikonami fialové barvy), ty nejsou probírány a ani se neobjeví na zkoušce – jejich úkolem je motivovat k dalšímu samostatnému studiu či pokusům nebo pomáhat v budoucnu při získávání dalších informací. Pokud je fialová ikona před názvem kapitoly (sekce), platí pro vše, co se v dané kapitole či sekci nachází.

Značení

Ve skriptech se používají následující barevné ikony:

-  *Rychlý náhled* (skript, kapitoly), ve kterém se dozvíme, o čem to bude.
-  *Klíčová slova* kapitoly.
-  *Cíle studia* pro kapitolu nám řeknou, co nového se v dané kapitole naučíme.
-  Nové *pojmy*, značení apod. jsou značeny modrým symbolem, který vidíme zde vlevo.
-  Konkrétní *postupy* a nástroje, způsoby řešení různých situací, do kterých se může správce počítačového vybavení dostat, atd. jsou značeny také modrou ikonou.

-  ,  Některé části textu jsou označeny fialovou ikonou, což znamená, že jde o *nepovinné úseky*, které nejsou probírány (většinou; studenti si je mohou podle zájmu vyžádat nebo sami prostudovat). Jejich účelem je dobrovolné rozšíření znalostí studentů o pokročilá témata, na která obvykle při výuce nezbývá moc času.
-  Žlutou ikonou jsou označeny odkazy, na kterých lze získat *další informace* o tématu. Nejčastěji u této ikony najdeme webové odkazy na stránky, kde se dané tématice jejich autoři věnují podrobněji.
-  Červená je ikona pro *upozornění* a poznámky.

Pokud je množství textu patřícího k určité ikoně větší, je celý blok ohraničen prostředím s ikonami na začátku i konci, například pro definování nového pojmu:

 **Definice 0.1**

V takovém prostředí definujeme pojem či vysvětlujeme sice relativně známý, ale komplexní pojem s více významy či vlastnostmi.

 **Věta 0.1**

V teoretické části najdeme věty, teorémy a lemmata, také pro ně existuje vlastní prostředí. Za tímto prostředím obvykle následuje důkaz.

Podobně může vypadat prostředí pro delší postup nebo delší poznámku či více odkazů na další informace. Mohou být použita také jiná prostředí:

 **Příklad 0.1**

Takto vypadá prostředí s příkladem. Příklady jsou obvykle komentovány, aby byl jasný postup jejich řešení.

 **Úkol**

Otázky a úkoly, náměty na vyzkoušení, které se doporučuje při procvičování učiva provádět, jsou uzavřeny v tomto prostředí. Pokud je v prostředí více úkolů, jsou číslovány.

Obsah

Předmluva **iii**

Úvod **1**

1 Základní znalosti **4**
1.1 Výroková logika . . . 4
1.2 Predikátová logika . . . 8
1.3 Důkazové metody sémantické analýzy . . . 11
1.4 Formální systémy . . . 16
1.4.1 Důkazy . . . 16
1.4.2 Logické systémy . . . 16
1.4.3 Vlastnosti formálních důkazových metod a systémů . . . 18

2 Systém přirozené dedukce **21**
2.1 Systém přirozené dedukce výrokové logiky . . . 21
2.2 Formální důkazy . . . 23
2.2.1 Přímý a nepřímý důkaz . . . 24
2.2.2 Hypotézy . . . 29
2.2.3 Větvené důkazy . . . 30
2.3 Vlastnosti Systému přirozené dedukce výrokové logiky . . . 32
2.3.1 Korektnost . . . 33
2.3.2 Úplnost a bezespornost . . . 34
2.4 Systém přirozené dedukce predikátové logiky . . . 37
2.5 Vlastnosti Systému přirozené dedukce predikátové logiky . . . 39

3 Klauzulární logika **42**
3.1 Hornovy klauzule . . . 42
3.2 Syntaxe jazyka klauzulární logiky . . . 43
3.2.1 Jazyk klauzulární logiky . . . 43
3.2.2 Univerzální tvrzení . . . 45
3.2.3 Existenční tvrzení . . . 47
3.3 Sémantika jazyka klauzulární logiky . . . 49
3.4 Vlastnosti klauzulí . . . 55
3.4.1 Prázdná množina antecedentu nebo konsekventu . . . 55

3.4.2 Konjunkce a disjunkce atomů v klauzuli . . . 56
3.4.3 Negace atomů . . . 58
3.4.4 Negace klauzule . . . 60
3.4.5 Predikát rovnosti a jiné predikáty podle relací . . . 61
3.5 Substituce . . . 63
3.6 Vztahy mezi klauzulemi . . . 65
3.6.1 Odvození důsledku dvojice klauzulí . . . 65
3.6.2 Unifikace klauzulí . . . 67
3.6.3 Znalostní báze . . . 72

4 Klauzulární axiomatický systém **76**
4.1 Definice systému . . . 76
4.2 Formální důkazy . . . 78
4.3 Korektnost systému . . . 80

5 Logické programování **82**
5.1 Pár slov k programovacím jazykům . . . 82
5.2 Logické programování v Prologu . . . 83
5.2.1 Zápis klauzulí v Prologu . . . 84
5.2.2 Ovládání aplikace SWI Prolog a webové aplipace SWISH . . . 86
5.2.3 Nápověda . . . 93
5.2.4 Základní práce s databází . . . 95
5.2.5 Anonymní proměnná . . . 96
5.3 Rezoluce v logickém programování . . . 97
5.3.1 Nepřímá rezoluce . . . 98
5.3.2 Lineární výpočetní strom . . . 99
5.4 Průběh výpočtu v Prologu . . . 102
5.5 Řízení výpočtu v Prologu . . . 108
5.5.1 Predikáty popření, selhání a řezu . . . 109
5.5.2 Krabičkový model . . . 111

Literatura **115**

A Příklady **118**

B Řešení příkladů **124**

Rejstřík **139**

Úvod

Co je to vlastně logika? Pod tímto pojmem většina lidí chápe souhrn obecných (myšlenkových) postupů, které slouží k vyvozování závěrů či vyslovování myšlenek neodporujících myšlenkám již přijatým za správné, a také postupů, které analyzují korektnost již vyslovených myšlenek. Existuje i kratší definice: logika je věda o správném usuzování. Podobnou definici najdeme ve většině učebnic logiky.

Rozlišujeme také různé druhy logiky, které však nemusí být navzájem úplně odděleny – například *formální logiku*, která se zaměřuje výhradně na formu tvrzení (odtud název), ne již na jejich obsah, stavěnou jako protiklad neformální (intuitivní) logiky, *matematickou logiku*, která pro analýzu (převážně matematických problémů) používá matematické nástroje (a je důsledně formální), *filozofickou logiku* zaměřenou na formální aplikaci logiky na filozofické problémy.

Absolventi středních škol znají alespoň v základu *výrokovou logiku* a *predikátovou logiku prvního řádu*. Zatímco výroková logika dokáže analyzovat tvrzení pouze do úrovně jednoduchých (atomických) výroků – výrokových proměnných a logických konstant (obdoba vět v souvětí spojených příslušnými spojkami), predikátová logika prvního řádu již rozlišuje subjekt (zastoupený individuovou konstantou) a vlastnost subjektu či vztah mezi subjekty (predikát). V predikátové logice používáme kvantifikaci individuových proměnných, za které lze dosadit individuové konstanty (určujeme aplikovatelnost na nejméně jedno nebo všechna individua daného oboru). Existují také *predikátové logiky vyšších řádů*, které zacházejí ještě dále, například kvantifikují také predikátové proměnné, pracují s vlastnostmi vlastností a vztahů, a také se vztahy mezi vlastnostmi a vztahy.

Různé typy logik jsou často děleny na *klasické* (dvouhodnotové) a *neklasické*. K neklasickým logikám řadíme například *vícehodnotové logiky*, kde kromě hodnot $true$ a $false$ mohou formule nabývat i jiných hodnot „mezi“ těmito dvěma základními hodnotami. To je například *tříhodnotová logika* přidávající prostřední hodnotu *možná* nebo *fuzzy logika* pracující s celým intervalem od 0 ($false$) do 1 ($true$).

K neklasickým logikám patří také *pravděpodobnostní logika*, která podobně jako fuzzy logika používá číselné hodnoty z daného intervalu, různé typy *modálních logik* přidávající ke klasické logice modality[1] možnosti a nutnosti, a *temporální logika* (logika času, vychází z modální logiky) pokoušející se vyjádřit časové hledisko při charakterizování prvků jazyka (kdy který děj nastal, kdy měl daný objekt zkoumanou vlastnost apod.).

[1] Modality jsou prostředky pro vyjádření pravděpodobnostních atributů logických tvrzení, například dané tvrzení platí nutně (vždy), a nebo možná (někdy; je možné, že…).

Jak a kde logika vznikla? Logika vznikla ve starověkém Řecku (alespoň podle dochovaných nálezů z historie). Náznaky logiky se objevují už u Sókrata a Platóna, za zakladatele logiky je však považován jejich pokračovatel Aristotelés. Aristotelova logika obohacená o metody teorie množin se dodnes používá při vyhodnocování nepříliš složitých výroků nazývaných *sylogismy*. Velký význam měla (a mají) také díla Euklidova. Euklides jako první definoval axiomatický systém. Známá je jeho axiomatika geometrie, která ještě donedávna zaměstnávala matematiky a inspirovala vznik neeuklidovských geometrií (hyperbolické, eliptické apod.) tím, že se tito matematici pokoušeli dokázat závislost jednoho z axiomů, pátého postulátu, na ostatních.

V následujících stoletích logiku přibližovali dnes běžnému pojetí George Boole (známe pojem Booleova algebra, který se úzce váže také k výrokové logice), John Venn (zabýval se logikou a teorií pravděpodobnosti, známé jsou také Vennovy diagramy), Gottlob Fregge (zavedl predikátovou logiku), Bertrand Russel a další. Na přelomu 19. a 20. století se v souvislosti s teorií množin a pokusy o co největší formalizaci vědy do popředí zájmu mnoha vědců dostaly tzv. *antinomie* (paradoxy, spory; některé se dochovaly již od starověku). Známý je například Russelův paradox (volně podle [3]):

> *Definujme normální množinu jako množinu, která neobsahuje samu sebe (tj. není svým vlastním prvkem). Je množina $\mathcal{M}$ všech normálních množin normální množinou?*
>
> *Kdyby byla odpověď ANO, pak by množina $\mathcal{M}$ nemohla obsahovat samu sebe. Když ale neobsahuje samu sebe, pak nemůže být množinou* všech *normálních množin, ona sama v sobě chybí. Proto odpověď nemůže být ANO.*
>
> *Kdyby byla odpověď NE, a tedy množina $\mathcal{M}$ by obsahovala samu sebe, pak by nemohla být množinou všech normálních množin, protože by obsahovala nejméně jednu množinu, která není normální (samu sebe). Proto odpověď nemůže být NE.*

Princip Russelova paradoxu vychází z vlastností teorie množin definované Cantorem tak, aby byla co nejjednodušší a snadno použitelná. Po zveřejnění paradoxu došlo k přeformulování teorie množin tak, aby formálně k paradoxům (nejen tomuto) nemohlo dojít, byla vytvořena teorie množin založená na axiomech.

Na počátku 20. století jeden z nejvýznamnějších matematiků té doby David Hilbert vytýčil program formalizace vědy (především matematiky), který měl kromě jiného i zamezit těmto paradoxům. Účelem bylo vytvořit formální systémy, které by zaručily *bezespornost* (při odvozování nových tvrzení nelze dojít ke sporu s předpoklady) a pokud možno *úplnost* (každé tvrzení platné v dané logice je dokazatelné) a *rozhodnutelnost* (o tvrzení bychom měli vždy mít možnost rozhodnout v konečném počtu kroků, zda je pravdivé v dané logice) odvozování.

Ukázalo se, že problémem je především úplnost, což dokázal Kurt Gödel ve své disertační práci. Zatímco výroková a predikátová logika prvního řádu jsou úplné, o teoriích na nich postavených to nemusí platit, například teorie aritmetiky. Bylo také dokázáno, že výroková logika je rozhodnutelná, predikátová logika prvního řádu rozhodnutelná není.

Hilbertův program formalizace vědy sice nebyl a ani nemohl být splněn, jak dokázal Gödel, ale přesto přinesl i užitečné výsledky a v současné logice se tento trend projevuje maximální snahou o formalizaci myšlenek a postupů jejich zpracování. S důsledky se setkáváme v mnoha vědních disciplínách, samozřejmě také v informatice.

Logika a informatika. Logika tedy bývá spojována s lidským myšlením. Od minulého století však s pojmy a postupy pracují nejen lidé, ale také stroje, počítače, proto nastala potřeba metody logické analýzy co nejvíce zautomatizovat a přizpůsobit potřebám a schopnostem počítačů.

V informatice logiku používáme často i tehdy, když to vůbec netušíme, například v expertních a databázových systémech může být používána pravděpodobnostní nebo fuzzy logika. Logika se také prosazuje obecně v umělé inteligenci, kde pomáhá ze zjištěných faktů a definovaných pravidel vytvářet nové úsudky a tím řídit chování umělé inteligence i v „nenaprogramovaných“ případech. Pro tyto účely se používají nejen klasické logiky, ale i logiky neklasické.

Základem pro využití logiky v informatice je logické programování. Nejde jen o to, abychom při programování mysleli logicky (to je ostatně naprosto nezbytné), ale jde o zavedení metod logické dedukce do programování a jejich přímou podporu v programovacím jazyce. *Logické programovací jazyky* byly původně založeny na klasických logikách, z nejznámějších je programovací jazyk Prolog. Postupně se však začaly objevovat logické programovací jazyky založené na neklasických logikách (například pro fuzzy logiku Fuzzy Prolog nebo pro temporální logiku Templog, Chronolog, Temporal Prolog), dalším (již ne zcela novým) zajímavým projektem je programovací jazyk Merkur.

Další možnost využití logiky je v analýze přirozeného jazyka, kde jsou kromě jiného vytvářeny nástavby pro obecné logické programovací jazyky (především pro varianty Prologu), ale také například v Lispu nebo v C.[2]

Co bude v textu následovat? Velká část následujícího textu je věnována formalizaci logiky. S některými formálními systémy jsme se již seznámili v předmětu Úvod do logiky – známe Hilbertovský a Gentzenovský systém. Nyní je zařadíme do obecného formálního rámce a podíváme se na další, trochu odlišný, formální systém – Systém přirozené dedukce. Účelem této kapitoly je především důkladně si osvojit význam logických spojek, syntaktických pravidel a logické dedukce, protože na těchto základech stojí všechny následující kapitoly a při jejich nepochopení se nedokážeme v dalším textu orientovat.

Dále definujeme nový typ logiky – klauzulární logiku, a pak na této logice postavíme další formální systém. Tento formální systém již můžeme použít pro popis základů logického programování (především v Prologu), což bude obsahem poslední kapitoly textu.

Do příloh byly zařazeny příklady týkající se klauzulární logiky a programování v Prologu. V příloze A najdeme zadání příkladů, v příloze B je jejich řešení.

V textu je odlišeno značení logických spojek $\rightarrow$ a $\leftrightarrow$ od metaznaků $\Rightarrow$ a $\Leftrightarrow$.

[2] Přehled najdeme například na http://nlp.fi.muni.cz/projekty/grammar_workbench/prehled.html.

Kapitola 1

Základní znalosti

Rychlý náhled: Tato kapitola především uspořádává znalosti získané v předchozím předmětu, Úvod do logiky. Jejím účelem je připomenout základní pojmy a metody, které budou používány v následujících kapitolách.

V poslední sekci této kapitoly se budeme zabývat dokazovacími systémy, které pro dedukci využívají výhradně syntaktické důkazové metody. Výhodou těchto systémů je především to, že syntaktické metody se snadno programují, a proto je lze využít pro logické programování.

Klíčová slova: Výroková logika, predikátová logika, formule, ohodnocení, model, interpretace, logická spojka, sémantická tabulka, literál, konjunkt, disjunkt, atom, term, arita, struktura, univerzum diskurzu, sémantické tablo, rezoluce, důkaz, důkaz sporem, nepřímý důkaz, logický systém, formální systém, teorie, sémantická korektnost, sémantická úplnost, bezespornost.

Cíle studia: Cílem je upevnit si základní pojmy a postupy matematické logiky, a také naučit se orientovat ve formálních systémech a jejich vlastnostech.

1.1 Výroková logika

V následujícím textu budeme používat tyto základní pojmy:

Výroková proměnná, logická konstanta

Složitost formule odpovídá počtu logických spojek formule, formule se složitostí 0 je tvořena pouze jedinou výrokovou proměnnou a žádnou logickou spojkou.

Ohodnocení (valuace) výrokové proměnné p je zobrazení v, které proměnné p přiřadí hodnotu $true$ nebo $false$. V tabulce 1.1 na straně 5 jsou jednotlivá ohodnocení proměnných v prvních dvou sloupcích.

Interpretace I výrokové formule F při zvoleném ohodnocení v je zobrazení přiřazující formuli F hodnotu $true$ nebo $false$ podle ohodnocení v uplatněného na všechny proměnné nacházející se ve formuli F a logické spojky dle tabulky 1.1. Značíme $I(F)$.

Model formule F je takové ohodnocení (valuace) v, ve kterém je formule interpretována hodnotou *true*.

Formule F je pravdivá při ohodnocení v, jestliže je v tomto ohodnocení interpretována hodnotou $true$ (tj. ohodnocení v je modelem formule F).

Formule F je splnitelná, jestliže má alespoň jeden model.

Formule F je tautologie (logicky platná, splněna, logický zákon), jestliže je pravdivá ve všech valuacích, tedy všechny valuace jsou modely formule F.

Formule F je kontradikce (nesplnitelná), když nemá žádný model.

Množina formulí $\mathcal{M}$ je splnitelná, jestliže všechny formule této množiny mají alespoň jeden společný model, tedy když existuje alespoň jedna valuace, která je modelem pro všechny formule množiny. Takovou valuaci nazýváme *model množiny formulí $\mathcal{M}$.*

Formule F logicky vyplývá z množiny formulí $\mathcal{M}$ (tedy je jejich logickým důsledkem), pokud je pravdivá v každém modelu množiny $\mathcal{M}$ (tj. každý model množiny $\mathcal{M}$ je zároveň modelem formule F). Značíme $\mathcal{M} \models F$.

Interpretace formule závisí nejen na zvolené valuaci, ale také na syntaktické struktuře formule dané především logickými spojkami. Nebudeme zde probírat syntaxi do důsledků, ale podíváme se na definici logických spojek.

Logické spojky používané ve výrokové logice mohou být unární (například negace, spojka má jeden argument) nebo binární (například konjunkce, spojka má dva argumenty) a nebo nulární ($true$, $false$ – nemají žádný argument). Při interpretaci formule se složitostí 1, tedy obsahující právě jednu logickou spojku, máme celkem $2^{2^2} = 2^4 = 16$ možností:

p	q	$false$	$p \,\&\, q$	$\neg(p \to q)$	p	$\neg(q \to p)$	q	$p + q$	$p \vee q$
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

p	q	$p \downarrow q$	$p \leftrightarrow q$	$\neg q$	$q \to p$	$\neg p$	$p \to q$	$p \uparrow q$	$true$
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

Tabulka 1.1: Sémantická tabulka logických spojek výrokové logiky

Ve výrokové logice nevyužíváme všechny výše uvedené spojky, ale obvykle jen následující:

- *nulární:* $false$, $true$, o nich hovoříme jako o *logických konstantách*
- *unární:* $\neg$ (negace)
- *základní binární:* $\&$ (konjunkce), $\vee$ (disjunkce), $\to$ (implikace), $\leftrightarrow$ (ekvivalence)
- *další binární:*
  - $+$ (XOR) – vylučovací nebo
  - $\uparrow$ (NAND, Schefferův operátor) – negace konjunkce („Not AND“)
  - $\downarrow$ (NOR, Pierceův operátor) – negace disjunkce („Not OR“)

V souvislosti s logickými spojkami používáme také další pojmy:

Funkčně úplná množina logických spojek: Je zřejmé, že některé logické spojky lze zapsat pomocí kombinace jiných (často se tak zvýší složitost formule). Funkčně úplná množina logických spojek je taková množina logických spojek, pomocí nichž lze vyjádřit všechny logické spojky uvedené v tabulce 1.1. Jsou to například tyto množiny: $\{\,\&\,,\vee,\neg\}$, $\{\,\&\,,\neg\}$, $\{\vee,\neg\}$, $\{\rightarrow,\neg\}$, $\{\uparrow\}$, $\{\downarrow\}$.

Literál výrokové proměnné p je buď tato proměnná nebo její negace, pro proměnnou p tedy existují dva literály: $p, \neg p$.

Disjunkt je formule, která je konjunkcí konečně mnoha literálů.

Formule je v disjunktivní normální formě (DNF), jestliže je disjunkcí konečně mnoha disjunktů, například formule $(p\,\&\,q)\vee(r\,\&\,\neg p\,\&\,\neg q)\vee\neg r$.

Konjunkt je formule, která je disjunkcí konečně mnoha literálů. Konjunkty se také nazývají *klauzule* (stejně jako v predikátové logice).

Formule je v konjunktivní normální formě (KNF), jestliže je konjunkcí konečně mnoha konjunktů, například formule $(p\vee q)\,\&\,(r\vee\neg p\vee\neg q)\,\&\,\neg r$. Této formě se také říká *klauzulární normální forma*.

Při úpravách logických výrazů často používáme některé tautologie. Pro výrokovou logiku jsou to například

- DeMorganovy zákony: $\neg(a\,\&\,b)\Leftrightarrow(\neg a\vee\neg b)$
  $\neg(a\vee b)\Leftrightarrow(\neg a\,\&\,\neg b)$
- Distributivita: $a\,\&\,(b\vee c)\Leftrightarrow(a\,\&\,b)\vee(a\,\&\,c)$
  $a\vee(b\,\&\,c)\Leftrightarrow(a\vee b)\,\&\,(a\vee c)$
- Přepis implikace: $(a\rightarrow b)\Leftrightarrow(\neg a\vee b)$
  $\neg(a\rightarrow b)\Leftrightarrow(a\,\&\,\neg b)$
- Přepis ekvivalence: $(a\leftrightarrow b)\Leftrightarrow((a\rightarrow b)\,\&\,(b\rightarrow a))$
- Zavedení a odstranění dvojí negace: $a\Leftrightarrow\neg\neg a$
  $\neg\neg a\Leftrightarrow a$

Také víme, že logické spojky konjunkce a disjunkce jsou komutativní, kdežto implikace není.

Disjunktivní normální formu výrokové formule získáme buď ekvivalentními úpravami této formule a nebo využitím modelů zjištěných ze sémantické tabulky. Konjunktivní normální formu výrokové formule získáme buď opět ekvivalentními úpravami, a nebo tak, že formuli znegujeme, převedeme do disjunktivní normální formy, opět znegujeme a pak při ekvivalentní úpravě využijeme vztah mezi konjunkcí a negací disjunkce (ale jde to i jednodušeji).

Příklad 1.1

Zjistíme disjunktivní a konjunktivní normální formu formule
$\mathcal{F}=(B\,\&\,\neg A)\vee\neg(B\rightarrow\neg A)$.

Nejdřív zjistíme modely této formule. Vytvoříme sémantickou tabulku a najdeme modely formule $\mathcal{F}$.

A	B	$\neg A$	$\neg B$	$X = B \,\&\, \neg A$	$Y = B \to \neg A$	$\mathcal{F} = X \vee \neg Y$
0	0	1	1	0	1	0
0	1	1	0	1	1	1
1	0	0	1	0	1	0
1	1	0	0	0	0	1

Našli jsme dva modely pro výrokové proměnné A, B: $\{A = 0, B = 1\}$ a $\{A = 1, B = 1\}$. Disjunktivní normální forma vychází z těchto dvou modelů, je

$$\mathcal{F} \Leftrightarrow (\neg A \,\&\, B) \vee (A \,\&\, B)$$

Nyní převedeme formuli do konjunktivní normální formy, použijeme nejdřív „delší“ cestu. K tabulce připojíme další sloupec s negací formule a zjistíme modely.

A	B	$\ldots$	$\mathcal{F}$	$\neg\mathcal{F}$
0	0		0	1
0	1		1	0
1	0		0	1
1	1		1	0

Modely negace formule jsou také dva – $\{A = 0, B = 0\}$ a $\{A = 1, B = 0\}$. Disjunktivní normální forma negace formule je

$$\neg\mathcal{F} \Leftrightarrow (\neg A \,\&\, \neg B) \vee (A \,\&\, \neg B)$$

Využijeme tautologie o dvojí negaci a formuli $\mathcal{F}$ takto upravíme:

$$\begin{aligned}
\mathcal{F} \;\Leftrightarrow\; \neg\neg\mathcal{F} \;\Leftrightarrow\; \neg(\neg\mathcal{F}) \;&\Leftrightarrow\; \neg\Big((\neg A \,\&\, \neg B) \vee (A \,\&\, \neg B)\Big) \;\Leftrightarrow \\
&\Leftrightarrow\; \neg(\neg A \,\&\, \neg B) \,\&\, \neg(A \,\&\, \neg B) \;\Leftrightarrow \\
&\Leftrightarrow\; (\neg\neg A \vee \neg\neg B) \,\&\, (\neg A \vee \neg\neg B) \;\Leftrightarrow \\
&\Leftrightarrow\; (A \vee B) \,\&\, (\neg A \vee B)
\end{aligned}$$

Získali jsme konjunktivní normální formu zadané formule $\mathcal{F}$.

Jak to jde jednodušeji (zkratkou): není třeba vytvářet sloupec tabulky s negací formule, stačí si uvědomit, co jsme vlastně dělali v předchozím postupu. Vybereme právě ta ohodnocení, která nejsou modely formule (tj. ve sloupci s formulí najdeme 0). Všechny literály proměnných určené stejně jako u disjunktivní formy navíc znegujeme (případně odstraníme dvojí negaci) a vytvoříme konjunkty.

Výše popsaný postup slouží k vytvoření *úplné disjunktivní a konjunktivní normální formy formule*. To znamená, že ve všech disjunktech/konjunktech jsou zastoupeny literály všech výrokových proměnných, které ve formuli máme. Jednodušší a kratší však bývají normální formy, které nejsou úplné, získáváme je obvykle ekvivalentními úpravami původní formule.

Úkoly

1. Podle tabulky 1.1 si vypište sémantickou tabulku pro implikaci.
2. Najděte všechny modely formule $(A \vee (B \to \neg A)) \to \neg(A \,\&\, B)$. Modely porovnejte s tabulkou, kterou jste vytvořili v předchozím úkolu. Jaká možnost úpravy formule ze srovnání vyplývá?

3. Najděte všechny modely množiny formulí $\{(p \,\&\, \neg q) \rightarrow false,\ p \vee q\}$.
4. Pomocí sémantické tabulky dokažte, že DeMorganovy zákony pro konjunkci a disjunkci jsou tautologie.
5. Dokažte, že platí $(\neg a \vee \neg b) \Leftrightarrow \neg(a \,\&\, b)$. Použijte nejdřív sémantickou tabulku a potom ekvivalentní úpravy podle tautologií pro přepis implikace.
6. Prohlédněte si sémantickou tabulku formule $\mathcal{F}$ vytvořenou v příkladu na str. 6. Lze tuto formuli podstatně zjednodušit?
7. Vytvořte disjunktivní a konjunktivní normální formu (nejdřív ekvivalentními úpravami, pak úplné formy pomocí sémantické tabulky) formule
$$\neg\Big((p \rightarrow q) \rightarrow (p \vee q)\Big).$$
8. Lze vytvořit disjunktivní normální formu kontradiktorické formule? Lze vytvořit konjunktivní normální formu tautologie? Pokud nelze, zdůvodněte (když nevíte, zkuste je vytvořit).

1.2 Predikátová logika

 V predikátové logice budeme kromě dříve uvedených pojmů, formulí a postupů z výrokové logiky používat následující:

Atom je nejmenší část formule, které lze přiřadit pravdivostní hodnotu $true$ nebo $false$, tedy je to predikát včetně argumentů (parametrů) nebo logická konstanta. Také hovoříme o atomické formuli.

Literál je atom nebo negace atomu, literály jsou například $p(x)$ a $\neg p(x)$.

Výskyt individuové proměnné x je *vázaný*, pokud je (tento výskyt) součástí některé podformule $\forall x\ A$ (univerzálně vázaná proměnná) nebo $\exists x\ A$ (existenčně vázaná proměnná). Výskyt individuové proměnné, který není vázaný, je *volný*.

Uzavřená formule je taková formule, ve které jsou všechny výskyty všech proměnných vázané (případně neobsahuje žádné proměnné). Formule, která není uzavřená, je *otevřená*.

Term t je substituovatelný za proměnnou x ve formuli F, jestliže je splněno:

1. proměnná x je ve formuli F volná (není vázaná kvantifikátorem),
2. term t nesmí obsahovat proměnnou, která je ve formuli vázaná (tj. původně volná proměnná se po substituci nesmí stát vázanou proměnnou).

Substituce termu za proměnnou se vždy provádí přes všechny výskyty této proměnné.

Arita predikátu je počet parametrů tohoto predikátu, podobně se určuje arita funktoru, relace, funkce. Například predikát $p(x, y)$ má aritu 2, což zapisujeme jako $p/2$. Na následujících stranách se s pojmem arita setkáme znovu.

Struktura $\mathcal{S}$ je uspořádaná trojice množin $\mathcal{S} = (W, \mathcal{F}, \mathcal{R})$, kde W nazýváme *univerzum diskurzu* (množina individuí), tato množina určuje svět, ve kterém se při interpretaci pohybujeme, dále $\mathcal{F} = \{F_1, F_2, \ldots, F_u\}$ je množina funkcí, $\mathcal{R} = \{R_1, R_2, \ldots, R_v\}$ je množina relací. Obvykle se předpokládá $W \neq \emptyset$.

Denotační zobrazení D je zobrazení přiřazující každému funktoru a predikátu formule včetně nulárních (funktoru, individuové konstantě, predikátu, logické konstantě) některý prvek struktury, tedy individuové konstaně prvek univerza diskurzu, funktoru funkci, predikátu relaci, logické konstantě sémantickou hodnotu $true$ nebo $false$).

Struktura aplikovatelná na formuli je struktura, pro kterou existuje denotační zobrazení použitelné (aplikovatelné) na tuto formuli, tedy platí

- jestliže je ve formuli alespoň jeden predikát, pak množina relací struktury je neprázdná (denotační zobrazení každému predikátu může přiřadit některý prvek množiny relací), navíc pro každý predikát ve formuli existuje v množině relací alespoň jedna relace se stejnou aritou,
- lze předpokládat, že univerzum diskurzu je neprázdné,
- jestliže jsou ve formuli funktory, je množina funkcí struktury neprázdná, také s ohledem na aritu funktorů.

Ohodnocení (valuace) individuové proměnné x je zobrazení e, které každé individuové proměnné přiřadí prvek z univerza diskurzu, tedy pro každou proměnnou x platí $e(x) \in W$. Pokud toto zobrazení přiřazuje svým argumentům pouze prvky univerza diskurzu struktury $\mathcal{S}$, mluvíme o *valuaci aplikovatelné na strukturu $\mathcal{S}$*.

Ohodnocení (valuace) termu t je zobrazení e' definované rekurzívně:

- $e'(c) = D(c)$, kde c je individuová konstanta (tj. jako denotační zobrazení),
- $e'(x) = e(x)$, x je individuová proměnná (tj. jako zobrazení ohodnocení individuové proměnné),
- $e'(f(t_1, t_2, \ldots, t_n)) = F(e'(t_1), e'(t_2), \ldots, e'(t_n))$, kde f je funktor, denotát tohoto funktoru je $D(f) = F$, $t_1, t_2, \ldots, t_n$ jsou termy.

Interpretace formule F je zobrazení I, které v dané struktuře $\mathcal{S}$ a valuaci e, což značíme $I(F)[\mathcal{S}, e]$, přiřadí formuli hodnotu $true$ nebo $false$ takto (rekurzivní definice):

- logické konstantě přiřadí hodnotu jejího denotátu,
- atomu s n-árním predikátem p ve tvaru $p(t_1, t_2, \ldots, t_n)$ je přiřazena hodnota $true$, pokud pro n-tici individuí vzniklou ohodnocením e' termů v jeho argumentech platí $(e'(t_1), e'(t_2), \ldots, e'(t_n)) \in R$, kde $R = D(p)$ (tj. relace R je denotátem predikátu p), jinak je přiřazena hodnota $false$,
- formuli se složitostí větší než 0 ve tvaru $A \circ B$, kde $\circ$ je některá ze spojek v tabulce 1.1 na str. 5, je přiřazena hodnota $(I(A)[\mathcal{S}, e]) \circ (I(B)[\mathcal{S}, e])$, a spojka $\circ$ je interpretována také podle tabulky 1.1,
- formuli ve tvaru $\forall x A$ je přiřazena hodnota $true$, pokud pro všechna individua $a \in W$ platí $I(A)[e(x/a)] = true$ (po dosazení a za všechny výskyty x je podformule A interpretována $true$), jinak je formule interpretována hodnotou $false$,
- formuli ve tvaru $\exists x A$ je přiřazena hodnota $true$, pokud pro nejméně jedno individuum $a \in W$ platí $I(A)[e(x/a)] = true$, jinak je formule interpretována hodnotou $false$.

Formule F je pravdivá ve struktuře $\mathcal{S}$ při ohodnocení e, jestliže v této struktuře a ohodnocení interpretována hodnotou $true$, zapisujeme $I(F)[\mathcal{S}, e] = true$.

Formule F je splnitelná ve struktuře $\mathcal{S}$, jestliže existuje ohodnocení, ve kterém je v této struktuře pravdivá.

Formule F je platná (splněna) ve struktuře $\mathcal{S}$, jestliže je v této struktuře interpretována hodnotou $true$ pro jakékoliv ohodnocení, zapisujeme $I(F)[\mathcal{S}] = true$.

Formule F je logicky platná (logický zákon, tautologie), jestliže je interpretována hodnotou $true$ v jakékoliv struktuře a ohodnocení, zapisujeme $I(F) = true$.

Stejně jako u výrokové logiky, i u té predikátové si ukážeme několik užitečných tautologií.

- DeMorganovy zákony pro kvantifikátory: $\neg(\forall x A(x)) \Leftrightarrow (\exists x \neg A(x))$
  $\neg(\exists x A(x)) \Leftrightarrow (\forall x \neg A(x))$
- Distribuce kvantifikátorů: $\forall x(A(x)\ \&\ B(x)) \Leftrightarrow (\forall x A(x)\ \&\ \forall x B(x))$
  $\exists x(A(x) \vee B(x)) \Leftrightarrow (\exists x A(x) \vee \exists x B(x))$

  (Zde si musíme dát pozor, pro jiné kombinace kvantifikátorů a logických spojek nemusí ekvivalence platit!)

Když zapisujeme funktory, funkce, predikáty a relace, máme možnost jejich argumenty reprezentovat dvojím způsobem:

- $f(x, y, z)$ – argumenty vypíšeme v závorce; tento způsob používáme, když je důležité jednotlivým argumentům přiřadit název pro další použití či význam nebo je vzájemně odlišit,
- $f/3$ – napíšeme lomítko a počet argumentů; používáme, když není nutné rozlišit jednotlivé argumenty. Číslo za lomítkem, jak už víme, se nazývá *arita*.

Například relaci $\mathrm{rodice}\left(\langle \mathrm{dite} \rangle, \langle \mathrm{matka} \rangle, \langle \mathrm{otec} \rangle\right)$ lze zapsat jako $\mathrm{rodice}/3$.

Příklad 1.2

Vezměme jednoduchou formuli $\mathcal{F} = \forall x\Big((p(x) \vee p(m(x))\Big)$. Vytvoříme strukturu pro interpretaci $\mathcal{S}_1 = (\mathcal{N},\ \{\mathrm{inc}/1, \mathrm{mocnina}/1\},\ \{\mathrm{sude}/1\})$.

Univerzum diskurzu je množina přirozených čísel, v množině funkcí máme funkci $\mathrm{inc(x)} = x + 1$ a funkci $\mathrm{mocnina}(x) = x^2$, v množině relací je unární relace $\mathrm{sude}/1$ vracející $true$, jestliže je její argument sudé číslo.

Struktura je aplikovatelná na zadanou formuli, protože funktoru m lze přiřadit některou funkci s vhodnou aritou a také pro predikát p existuje relace se stejným počtem argumentů.

Použijeme uvedenou strukturu a denotační zobrazení D_1:
$D_1(p/1) = \mathrm{sude}/1$, $D_1(m/1) = \mathrm{mocnina}/1$.

Po použití denotačního zobrazení D_1 na formuli bude $\forall x\Big(\mathrm{sude}(x)\ \vee\ \mathrm{sude}(x^2)\Big)$. Toto denotační zobrazení není zrovna ideálně zvoleno, což si můžeme vyzkoušet například valuací individuové proměnné x hodnotami $e(x) = 4$ a $e(x) = 3$.

U struktury $\mathcal{S}_1$ vyzkoušíme jinou denotaci:
$D_2(p(x)) = \mathrm{sude}(x)$, $D_2(m(x)) = x + 1$.

Opět použijeme na formuli: $\forall x\Big(\mathrm{sude}(x)\ \vee\ \mathrm{sude}(x+1)\Big)$. Při jakkoliv zvolené valuaci (za x dosazujeme přirozená čísla z univerza diskurzu) je výsledkem $true$. Protože v definici struktury aplikovatelné na formuli je „…pro kterou *existuje* denotační zobrazení …“, můžeme zvolit denotaci D_2. Daná formule $\mathcal{F}$ je ve struktuře $\mathcal{S}_1$ *pravdivá* pro jakékoliv ohodnocení individuových proměnných (zde jen x), proto je *v této struktuře platná* (splněna). Není to však tautologie, s čímž souvisí i výše naznačená role volby denotačního zobrazení.

Příklad 1.3

Při interpretaci formule $\forall x\Big((p(x) \vee p(m(x))\Big)$ z předchozího příkladu použijeme jinou strukturu: $\mathcal{S}_2 = (\{\text{jan}, \text{eva}, \text{karel}, \text{iva}, \ldots\},\ \{\text{otec}/1\},\ \{\text{je\_zena}/1\})$.

Univerzum diskurzu obsahuje jména, funkce $\text{otec}/1$ vrací jméno otce svého argumentu, relace $\text{je\_zena}/1$ vrací $true$, jestliže je argumentem individuum určující ženské jméno. Konkrétní formu funkce a relace nebudeme uvádět, je zřejmá.

Tato struktura je také aplikovatelná na danou formuli, zvolíme denotaci D_3:
$D_3(p) = \text{je\_zena}$, $D_3(m) = \text{otec}$.

Opět použijeme na formuli $\forall x\Big(\text{je\_zena}(x) \vee \text{je\_zena}(otec(x))\Big)$.

Pokud při valuaci dosadíme za x ženské jméno, je celá formule interpretována hodnotou $true$, ale pokud použijeme mužské jméno, je formule interpretována hodnotou $false$. Proto formule sice *je splnitelná* ve struktuře $\mathcal{S}_2$ (pro $e(x)$ dosazující ženská jména), ale *není v této struktuře platná*.

Zjistili jsme, že formule $\mathcal{F}$ je ve struktuře $\mathcal{S}_1$ platná, ale není platná ve struktuře $\mathcal{S}_2$. Proto se nejedná o *tautologii* (logický zákon), není logicky platná – to by musela být platná ve všech aplikovatelných strukturách.

Úkoly

1. Při znalosti všech výrokových a predikátových tautologií dosud v textu uvedených zjistěte, která z následujících formulí je tautologie, a dokažte to pomocí ekvivalentních úprav:
   - $\exists x\,(A(x) \to B(x)) \Leftrightarrow (\forall x A(x) \to \forall x B(x))$
   - $\exists x(A(x) \to B(x)) \Leftrightarrow (\forall x A(x) \to \exists x B(x))$
   - $\exists x(A(x) \to B(x)) \Leftrightarrow (\exists x A(x) \to \exists x B(x))$
2. Pro interpretaci následujících formulí vymyslete dvě různé struktury S_1 a S_2 s neprázdným univerzem diskurzu (společné pro všechny formule). Ukažte, že jsou v těchto strukturách splnitelné/platné/nesplnitelné.
   - $\exists x\ \Big(p(x) \vee \neg p(x)\Big)$
   - $\exists x\ \Big(p(x)\ \&\ \neg p(x)\Big)$
   - $\forall x\ \exists y\ \Big(q(x,y) \to q(f(x),y)\Big)$
   - $\forall x\ q(x, f(x)) \to \exists x\ q(x,x)$

1.3 Důkazové metody sémantické analýzy

Sémantická analýza má za úkol určit, zda je analyzovaná formule tautologie, kontradikce či splnitelná, stanovujeme tedy pravdivostní (sémantickou) hodnotu formule.

 V předchozím semestru jsme se seznámili se základními metodami sémantické analýzy pro provádění důkazů. Probírali jsme

1. *sémantické metody* (pracujeme s pravdivostními hodnotami jednotlivých částí formule):
   - sémantická tabulka
   - sémantický strom, Quinův algoritmus
2. *syntaktické metody* (při analýze nepracujeme s pravdivostními hodnotami částí formule, používáme pouze syntaktickou strukturu formule – strukturou podformulí, umístěním jednotlivých logických spojek, kvantifikátory, ...):
   - sémantické tablo (slovo „sémantické“ v názvu znamená pouze ten fakt, že účelem metody je zjistit sémantickou hodnotu formule)
   - rezoluce (přímá a nepřímá)

Dále se soustředíme především na syntaktické metody a možnosti jejich použití při dokazování. Výhodou syntaktických metod je jejich snadnější naprogramování a obecně automatizace, také díky tomu, že důkaz logicky platné formule syntaktickou metodou bývá konečný. Princip důkazu vychází z deduktivního úsudku.

Definice 1.1 (Deduktivní úsudek)

Deduktivní úsudek zapisujeme schématem $P_1, P_2, \ldots, P_n \models Z$, kde

- $P_1, P_2, \ldots, P_n, Z$ jsou tvrzení,
- $P_1, P_2, \ldots, P_n$ nazýváme *předpoklady* (premisy),
- Z je *závěr*.

Aby se jednalo o platný deduktivní úsudek, musí platit:

- Z platnosti předpokladů usuzujeme na platnost závěru (tj. to, zda je závěr platný, vyplývá z toho, zda jsou či nejsou platné předpoklady).
- Ve všech případech, kdy jsou platné zároveň všechny předpoklady, musí být platný i závěr (tedy nesmí nastat situace, ve které by všechny předpoklady byly pravdivé a závěr nepravdivý). „Ve všech případech“ může znamenat například ve výrokové logice „při všech možných ohodnoceních výrokových symbolů“.

Jinými slovy: zjistíme modely množiny předpokladů; kdyby závěr nebyl v některém z těchto modelů pravdivý (při daném ohodnocení by měl hodnotu $false$), nejednalo by se o deduktivní úsudek. Všimněte si, že v definici se nepíše nic o tom, jak má být ohodnocen závěr.

Větu „nesmí nastat situace, ve které by všechny předpoklady byly pravdivé a závěr nepravdivý“ můžeme symbolicky přepsat jako

$$\neg\left(P_1 \& P_2 \& \ldots \& P_n \& \neg Z\right) \tag{1.1}$$

$$\neg\left(\left(P_1 \& P_2 \& \ldots \& P_n\right) \& \neg Z\right) \tag{1.2}$$

$$\left(P_1 \& P_2 \& \ldots \& P_n\right) \rightarrow Z \tag{1.3}$$

Zapisujeme $P_1, P_2, \ldots, P_n \models Z$.

Odvozovací pravidlo je metoda, kterou lze použít pro odvození jednoho tvrzení z jiných. Musí splňovat vlastnosti deduktivního úsudku (definice na str. 12). Existuje mnoho odvozovacích pravidel, v následujícím textu se s některými seznámíme. Obvykle jsou reprezentována formou deduktivního úsudku, například odvozovací pravidlo pro rezoluci ve výrokové logice zapisujeme jako

$$A \vee B,\ \neg B \vee C \ \models\ A \vee C \tag{1.4}$$

Odvozovací pravidla, která používáme v sémantickém tablu, najdeme v tabulce 1.2.

Pravidlo	*Význam*	*Větvení v tablu*
α-pravidla	$A \,\&\, B$ $\neg(\neg A \vee \neg B)$ $\neg(A \to \neg B)$	$\cdots$ $\downarrow$ A, B
β-pravidla	$A \vee B$ $\neg(\neg A \,\&\, \neg B)$ $\neg A \to B$	$\cdots$ $A \qquad B$
γ-pravidla	$\forall x\, A(x)$ $\neg\exists x\, \neg A(x)$	$\cdots$ $\downarrow$ $\forall x\, A(x),\ A(a)$
δ-pravidla	$\exists x\, B(x)$ $\neg\forall x\, \neg B(x)$	$\cdots$ $\downarrow$ $B(b)$

Tabulka 1.2: Odvozovací pravidla v sémantickém tablu

Definice 1.2 (Definice důkazu)

Důkaz tvrzení T z předpokladů $P_1, P_2, \ldots, P_n$ je taková posloupnost tvrzení $B_1, B_2, \ldots, B_m$, kde $B_m = T$ a každý její člen $B_i, 1 \leq i \leq m$, je:

- jeden z předpokladů P_j nebo
- vznikl uplatněním některého odvozovacího pravidla na předchozí členy posloupnosti.

Věta 1.1 (Věta o nepřímém důkazu a důkazu sporem)

Následující formule jsou tautologie:

$$(A \to B) \ \Leftrightarrow\ (\neg B \to \neg A\,) \tag{1.5}$$

$$(F \leftrightarrow true) \ \Leftrightarrow\ (\neg F \leftrightarrow false) \tag{1.6}$$

$$((A \to B) \leftrightarrow true) \ \Leftrightarrow\ ((A \,\&\, \neg B) \leftrightarrow false) \tag{1.7}$$

Důkaz: Všechny tyto vztahy jsou dokazatelné například sémantickou tabulkou, což všichni ovládáme, důkazy tedy necháme na čtenáři.

□

Poznámka:

Vztah (1.5) se nazývá *nepřímý důkaz formule F*; provádíme ho tak, že znegujeme podformuli dané formule nacházející se vpravo od implikace (zde B) a pokoušíme se odvodit negaci podformule vlevo od implikace (zde A).

Vztah (1.6) se nazývá *důkaz sporem*; provádíme ho tak, že formuli znegujeme a odvozujeme tak dlouho, dokud nedojdeme ke sporu – kontradiktorické formuli ($false$). Vztah (1.7) je konkretizace postupu na implikaci.

Pojmy nepřímého důkazu a důkazu sporem často splývají, protože vztah (1.6) pro důkaz sporem lze pro potřeby dokazování zjednodušit na

$$(true \to F) \Leftrightarrow (\neg F \to false),$$

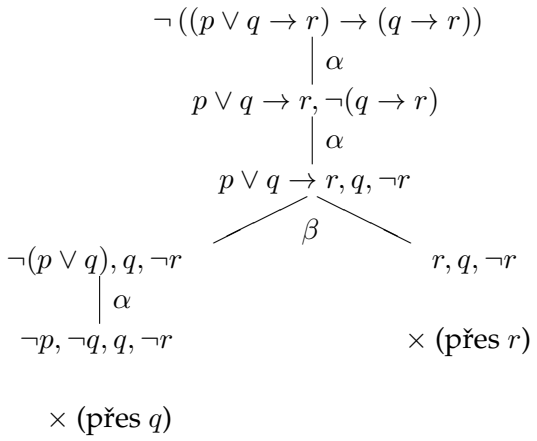
což je vlastně nepřímý důkaz po dosazení F za B a $true$ za A ve vztahu (1.5).

Příklad 1.4

Ověříme sémantickým tablem (sporem), zda platí $p \vee q \to r \models q \to r$

Ověřujeme logickou platnost formule $(p \vee q \to r) \to (q \to r)$. Vytvoříme sémantické tablo pro *negaci* této formule a pokud se uzavře, znamená to, že negace formule je kontradikce, tedy původní formule je tautologie.

Protože jde o formuli výrokové logiky, budeme používat pouze α a β pravidla. Výsledné tablo najdeme na obrázku 1.1.



Obrázek 1.1: Příklad sémantického tabla pro výrokovou logiku

Sémantické tablo se uzavřelo, proto negovaná formule je kontradikce a původní formule tautologie, tedy ověřovaný vztah platí.

U sémantického tabla pro formuli predikátové logiky si musíme dát pozor především na to, abychom při uplatňování δ-pravidla dosadili za individuovou proměnnou vždy jen takovou konstantu, která v důkazu (tablu) dosud nebyla použita. Pokud ve větvi uplatňujeme γ- i δ-pravidla, nejdřív použijeme δ-pravidla a potom teprve γ-pravidla, protože v těch nejsme nijak omezeni výběrem konstanty pro dosazení.

Příklad 1.5

Ověříme nepřímou rezolucí platnost vztahu $\{p \vee q, p \to r, q \to s\} \models r \vee s$.

Použijeme nepřímou rezoluci, dokazujeme, že následující formule (negace původního vztahu) je kontradikce:
$(p \vee q)\&(p \to r)\&(q \to s)\&\neg(r \vee s)$.

Abychom mohli rezoluční pravidlo používat (je ve vzorci (1.4) na straně 13), musíme mít formuli v konjunktivní normální formě. Jednotlivé podformule rozepíšeme a postupně uplatňujeme rezoluční odvozovací pravidlo.

Ekvivalentní formule v konjunktivní normální formě je
$(p \vee q)\ \&\ (p \to r)\ \&\ (q \to s)\ \&\ \neg(r \vee s) \Leftrightarrow (p \vee q)\ \&\ (\neg p \vee r)\ \&\ (\neg q \vee s)\ \&\ \neg r\ \ \&\ \neg s$

Řádkový důkaz:

1.	$p \vee q$	
2.	$\neg p \vee r$	
3.	$\neg q \vee s$	
4.	$\neg r$	*první konjunkt negace závěru*
5.	$\neg s$	*druhý konjunkt negace závěru*
6.	$\neg p$	R(2,4)
7.	q	R(1,6)
8.	s	R(3,7)
9.	$\square$	R(5,8)

Protože jsme dospěli k prázdné formuli ($\square$, *false*), původní vztah je platný.

Úkoly

1. Dokažte, že daná formule je tautologie. Použijte všechny výše zmíněné základní metody – sémantickou tabulku, sémantický strom, sémantické tablo i rezoluci. Nezapomeňte, že za určitých okolností je nutné použít nepřímý důkaz.

$$(p \to q) \leftrightarrow (\neg q \to \neg p)$$

2. Zjistěte, zda je daná formule tautologie. Použijte sémantické tablo.

$$(A\ \&\ \forall x B(x))) \to \forall x(A\ \&\ B(x))$$

3. Dokažte sémantickým tablem (pozor na posloupnost γ a δ pravidel) formuli

$$\forall x(A(x) \to B(x)) \to (\exists x\, A(x) \to \exists x\, B(x))$$

1.4 Formální systémy

1.4.1 Důkazy

Jaké důkazy obvykle provádíme?

1. Zajímá nás, zda daná formule je tautologií.
2. Chceme vědět, jestli daný závěr vyplývá ze zadaných předpokladů.
3. Zjišťujeme, co by mohlo vyplývat ze zadaných předpokladů (nemáme konkrétní představu o tom, co dokazujeme, generujeme formule vyplývající z předpokladů).

Je zřejmé, že první dva případy jsou vzájemně převeditelné (viz vztahy (1.1)–(1.3) na straně 12).

 Důkazy dělíme na:

- *Přímé*
  - v případě dokazování logické platnosti formule je na konci důkazu právě tato formule,
  - v případě dokazování vyplývání závěru z množiny předpokladů pak je na konci důkazu dokazovaný závěr.
- *Nepřímé*
  - v případě dokazování logické platnosti formule tuto formuli popřeme a dokážeme, že tato negace je kontradikce,
  - v případě dokazování vyplývání závěru z množiny předpokladů závěr znegujeme, přidáme k množině předpokladů a postupujeme stejně jako v předchozím bodě (jen pracujeme s množinou formulí), opět je třeba dojít ke sporu s některou formulí v množině.

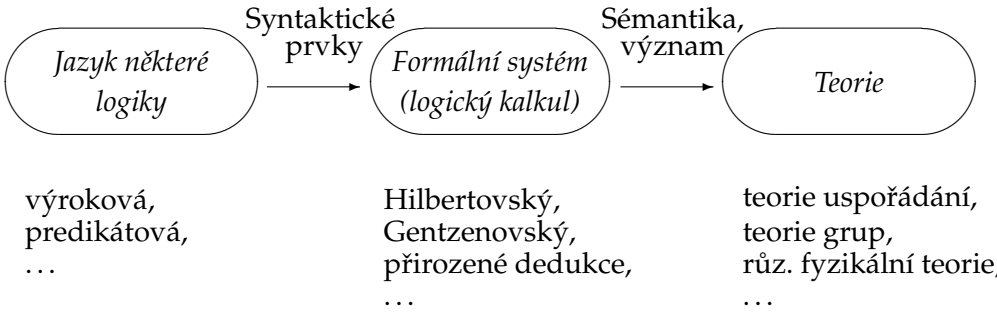
Dosud jsme se zabývali spíše nepřímými důkazy. Pomocí sémantického tabla a rezoluce jsme dokazovali, že negace formule je kontradikce, a proto původní formule je tautologie.

1.4.2 Logické systémy

Ve výrokové a predikátové logice existují pro ověřování logické platnosti formulí sémantické i syntaktické postupy (zmíněné v kapitole 1.3), pokud však chceme využít logiku pro budování teorií, nebude nám to stačit. Proto vytváříme systémy s předem nadefinovanými stavebními kameny – axiomy, a metodami – odvozovacími pravidly. V rámci takto nadefinovaného systému pak pomocí odvozovacích pravidel odvozujeme další tvrzení, která nazýváme větami nebo teorémy.

Odvozovací pravidla ve formálních systémech bývají syntaktická, při jejich používání se tedy nezabýváme přímo ohodnocením jednotlivých prvků jazyka, ale pracujeme se syntaktickou strukturou formulí. Tato pravidla obvykle vycházejí ze syntaktických metod sémantické analýzy nebo jsou odvozena z vlastností operátorů zvoleného jazyka (konjunkce, implikace, ...). Znak $\models$ značí logické vyplývání, s ním jsme se už setkali. U formálních systémů budeme používat znak $\vdash$ – symbol pro *dokazatelnost*.

 *Formální systém* nad daným formálním jazykem je množina tvrzení formulovaných v tomto jazyce.



Obrázek 1.2: Postup vytváření logických systémů

✎ *Syntaktické prvky* jsou buď formule (nazýváme je obvykle *logické axiomy*) nebo *odvozovací pravidla* (ta určují, jak odvozovat další formule). Logické axiomy musí být vždy tautologie nad zvoleným jazykem formálního systému (např. nad výrokovou logikou), odvozovací pravidla musí splňovat vlastnosti deduktivního úsudku. Celý důkazový postup musí zachovávat pravdivost ve struktuře.

✎ U *sémantických prvků* již tento požadavek není. Přidáváme formule (nazývané obvykle *speciální axiomy*), které nemusí být tautologiemi, stačí, když jsou platné v dané struktuře (tj. za určitých okolností, v dané interpretaci). Když začneme brát v úvahu interpretaci, systém přestává být univerzálně použitelný, ale přesto je užitečný, protože existují taková tvrzení, která platí v jedné teorii, ale v jiné nikoliv.

Například v teorii grup platí speciální axiom o neutrálním prvku (symbolem $\circ$ označme operaci grupy), v komutativní grupě také speciální axiom komutativity operace:

$$\exists n \forall x (x \circ n = x \ \ \& \ \ n \circ x = x) \tag{1.8}$$

$$\forall x \forall y (x \circ y = y \circ x) \tag{1.9}$$

Tyto axiomy však neplatí v mnoha jiných teoriích včetně některých algebraických (existují algebraické struktury, které nemají neutrální prvek, případně operace, které nejsou komutativní). *Speciální axiomy* jsou tvrzení, která považujeme za platná v rámci této teorie (tj. v zamýšlené interpretaci), ale nemusí být platná v každé interpretaci (jako třeba tvrzení o neutrálním prvku (1.8)).

Formální systémy dělíme na axiomatické a předpokladové.

✎ *Axiomatický systém* je určen

- jazykem (obvykle výroková nebo predikátová logika, příp. s omezením spojek),
- logickými axiomy,
- odvozovacími pravidly.

Důkaz začíná axiomy (případně dokázanými větami), vytváříme přímé důkazy. Typickým příkladem axiomatického formálního systému je Hilbertovský axiomatický systém.

✎ *Předpokladový formální systém* je určen

- jazykem,
- dedukčními pravidly.

Dedukční pravidla (jsou obdobou odvozovacích pravidel, musí splňovat jejich vlastnosti) mohou mít předpoklady (tj. jsou ve tvaru „jestliže byly již dokázány věty XXX, pak lze do důkazu přidat

větu YYY“), ale nemusejí. Dedukční pravidla bez předpokladů nazýváme axiomy. V důkazu můžeme používat různé druhy předpokladů, tedy přípustný je i nepřímý důkaz (důkaz sporem).

Zatímco axiomatické systémy se vyznačují rozsáhlejší množinou základních tvrzení, axiomů, a jen několika odvozovacími pravidly, předpokladové systémy jsou konstruovány tak, aby umožňovaly „přirozené“ odvozování vycházející z definice logických spojek a dalších prvků jazyka, proto počet dedukčních pravidel bývá přímo úměrný počtu těchto prvků.

 *Teorii* vytvoříme tak, že k vybranému formálnímu systému přidáme speciální axiomy. Stejně jako u formálních systémů se i v teoriích používají syntaktické metody odvozování (z logických a speciálních axiomů odvozujeme pomocí syntaktických – odvozovacích či dedukčních – pravidel další věty teorie).

1.4.3 Vlastnosti formálních důkazových metod a systémů

Aby byla metoda (nebo formální systém) použitelná pro dokazování, musí mít vlastnosti, které nám zaručí, že při používání této metody získáváme opravdu správné (korektní) výsledky. U metod sémantické analýzy (tablo, rezoluce) a u formálních systémů vyžadujeme tyto vlastnosti:

- sémantická korektnost,
- sémantická úplnost,
- bezespornost.

Užitečnou vlastností může být také minimálnost. U formálních systémů a teorií se vyžaduje především korektnost a bezespornost.

 **Definice 1.3 (Sémantická korektnost a sémantická úplnost)**

Důkazová metoda je sémanticky korektní, pokud

- každá pomocí ní dokazatelná formule je logicky platná (teorém) – pokud lze formuli dokázat touto metodou, je logicky platná, resp.
- formule dokazatelná z daných předpokladů logicky vyplývá z těchto předpokladů.

Důkazová metoda je sémanticky úplná, pokud

- všechny logicky platné formule lze touto metodou dokázat, resp.
- jestliže formule logicky vyplývá z daných předpokladů, pak je z nich dokazatelná.

V druhém případě hovoříme o vlastnosti *silné úplnosti*.

Jaký je vztah mezi korektností a úplností? Označme M metodu, jejíž korektnost a úplnost zkoumáme, $\mathcal{F}_M$ množinu všech formulí dokazatelných metodou M a $\mathcal{P}$ množinu všech logicky platných formulí. Pak platí:

Korektnost: $\mathcal{F}_M \subseteq \mathcal{P}$
Úplnost: $\mathcal{F}_M \supseteq \mathcal{P}$

Pro metodu, která je zároveň korektní a úplná, platí $\mathcal{F}_M \cong \mathcal{P}$, což znamená, že metodou lze dokázat právě ty formule, které jsou logicky platné.

Definice sémantické korektnosti a úplnosti pro formální systémy jsou obdobné definicím pro metody, proto je zde nebudeme uvádět.

Pojem sporné množiny formulí již známe z předchozího semestru a víme, že při dokazování důsledku množiny formulí vždy musíme ověřit, zda je tato množina formulí bezesporná. Dále definujeme sporný a bezesporný systém.

Definice 1.4 (Sporný a bezesporný systém)

Formální systém je sporný, jestliže je v něm dokazatelná jakákoliv formule. Formální systém je bezesporný, když není sporný.

Věta 1.2 (O sporném systému)

Je-li $\mathcal{U}$ sporná množina formulí, pak platí $\mathcal{U} \vdash F$ i $\mathcal{U} \vdash \neg F$. Slovy: Ve sporném systému existuje formule F taková, že v tomto systému je dokazatelná jak formule F, tak i její negace.

Důkaz: je zřejmý, vyplývá z předchozí definice.

□

Definice 1.5 (Minimální systém)

Nechť $\mathcal{M}$ je množina obsahující logické axiomy formálního systému. Formální systém je minimální, pokud je množina $\mathcal{M}$ minimální, tedy žádný prvek této množiny není dokazatelný z ostatních jejích prvků.

U syntaktických formálních systémů i teorií je vyžadována korektnost. S úplností se běžně setkáváme u formálních systémů, ale u teorií až tak obvyklá není. Například mnohé matematické teorie jsou sice korektní, ale nejsou úplné. Bezespornost je u jakýchkoliv logických systémů naprostou nutností, množina axiomů a odvozovacích pravidel musí být vždy bezesporná.

Příklad 1.6

Pokusme se odvodit vztahy mezi korektností, úplností a bezesporností (týká se všech formálních systémů). Označme K korektnost, U úplnost a B bezespornost.

- $K \Rightarrow B$
  Když je systém korektní, je také bezesporný.
- $\neg B \Rightarrow U$
  Sporný systém je vždy úplný (všechny logicky platné formule v něm dokážeme).
- $U \,\&\, \neg K \Rightarrow \neg B$
  Úplný systém, který není korektní, je sporný.
- $\neg B \Rightarrow \neg K$
  Sporný systém není korektní.

- $U \,\&\, B \Rightarrow K$
  Úplný a bezesporný systém je korektní.
- $\neg(U \Rightarrow B)$
  Úplný systém nemusí být bezesporný (ale může být).
- $\neg(B \Rightarrow U)$
  Bezesporný systém nemusí být úplný.

Důkazy těchto tvrzení vyplývají přímo z definice korektnosti, úplnosti a bezespornosti. Nejsou náročné, proto je přenecháme čtenáři.

Úkoly

1. Dokažte korektnost a úplnost metody nepřímé rezoluce ve výrokové logice. U korektnosti pracujte s rezolučním pravidlem a postupem při jeho používání, u úplnosti si všimněte formy, do které je nutné předem převést formuli. Zamyslete se také nad tím, zda do této formy některé formule nelze převést, a co z toho vyplývá.
2. Zdůvodněte vztahy uvedené v příkladu na straně 19. Zvláště si všimněte posledních dvou vztahů.

Kapitola 2

Systém přirozené dedukce

Rychlý náhled: V této kapitole se budeme zabývat Systémem přirozené dedukce, který je zástupcem *předpokladových formálních systémů*. Systém postavíme nejdřív na výrokové logice a potom na predikátové logice prvního řádu. Vycházíme především z literatury [3].

Systém definujeme pro výrokovou logiku, dále probereme typy důkazů, které zde lze provádět a dokážeme korektnost a úplnost tohoto systému, a pak provedeme totéž pro predikátovou logiku. Systém přirozené dedukce predikátové logiky je vlastně rozšířením Systému přirozené dedukce výrokové logiky, proto ty části, důkazy a věty, které jsou stejné, nebudeme uvádět.

Klíčová slova: Systém přirozené dedukce, dedukční pravidlo, věta o substituci, formální důkaz, přímý a nepřímý důkaz, hypotéza, přímá a nepřímá hypotéza, větvený důkaz.

Cíle studia: Po prostudování této kapitoly porozumíte tomu, jak se vytváří a používá předpokladový formální systém, konkrétně Systém přirozené dedukce. Cílem je pochopit, že ať už cokoliv dokazujeme nebo třeba programujeme (v logickém programovacím jazyce), vždy se pohybujeme v uceleném logickém systému, jehož možnosti a hranice jsou přesně dány.

2.1 Systém přirozené dedukce výrokové logiky

Jazyk Systému přirozené dedukce výrokové logiky přejímáme z výrokové logiky. *Dedukční pravidla* jsou uvedena v tabulce 2.1. Pravidlo eliminace implikace (EI) je také nazýváno *Modus Ponens* a značí se MP.

Následující věta nám umožňuje používat dedukční pravidla jako metapravidla, tj. za „písmena“ A, B lze dosazovat jakoukoliv dobře utvořenou formuli.

Věta 2.1 (Věta o substituci)

Nechť F je formule výrokové logiky, která obsahuje právě výrokové proměnné $p_1, p_2, \ldots, p_n$ (a žádné další). Nechť dále $A_1, A_2, \ldots, A_n$ jsou jakékoliv formule výrokové logiky.

Formuli F' vytvoříme tak, že pro každé i, $1 \leq i \leq n$, nahradíme všechny výskyty výrokové proměnné p_i ve formuli F formulí (A_i). Potom jestliže F je tautologie, pak také F' je tautologie.

1.	$\vdash A \vee \neg A$	A	axiom
2.	$\vdash A \to A$	A	axiom
3.	$A, B \vdash A \,\&\, B$	ZK	zavedení konjunkce
4.	$A \,\&\, B \vdash A$ nebo $A \,\&\, B \vdash B$	EK	eliminace konjunkce
5.	$A \vdash A \vee B$ nebo $B \vdash A \vee B$	ZD	zavedení disjunkce
6.	$A \vee B, \neg A \vdash B$ nebo $A \vee B, \neg B \vdash A$	ED	eliminace disjunkce
7.	$B \vdash A \to B$	ZI	zavedení implikace
8.	$A \to B, A \vdash B$	EI	eliminace implikace
9.	$A \to B, B \to A \vdash A \leftrightarrow B$	ZE	zavedení ekvivalence
10.	$A \leftrightarrow B \vdash A \to B$ nebo $A \leftrightarrow B \vdash B \to A$	EE	eliminace ekvivalence

Tabulka 2.1: Dedukční pravidla Systému přirozené dedukce výrokové logiky

Důkaz: Každá výroková proměnná p_i může nabývat hodnot z množiny $\{0, 1\}$, každá formule A_i taktéž. Jestliže je F tautologie, pak je pravdivá pro jakékoliv ohodnocení, tedy ať za p_i dosadíme cokoliv, vždy po vyhodnocení formule F dostaneme hodnotu 1 (*true*).

Po dosazení (A_i) za p_i můžeme totéž tvrdit o F' – ať jsou formule A_i jakkoliv interpretovány, formule F' je vždy vyhodnocena s výsledkem 1 (*true*).

□

Poznámka:

Všimněte si, že při nahrazování výrokových proměnných formule A_i závorkujeme. Není to vždy nutné, ale měli bychom mít na paměti, že pro logické spojky platí podobné pravidlo, které známe z aritmetiky – vyhodnocování zprava. U komutativních logických spojek není problém, ale například u implikace je třeba zajistit, aby vkládaná formule A_i byla ve výsledné formuli F' podformulí.

Příklad 2.1

Vezměme formuli $F = X \to (Y \to X)$ a nahraďme všechny výskyty proměnné X formulí $p \to q$. Bez uzávorkování získáme formuli $F' = p \to q \to (Y \to p \to q)$.

Zatímco formule F je tautologie, formule F' tautologie není, protože není pravdivá pro ohodnocení $v(Y) = 0,\ v(p) = 0,\ v(q) = 0$.

Při použití uzávorkování vytvoříme formuli $F'' = (p \to q) \to (Y \to (p \to q))$. Tato formule je již tautologie (ověřte). Formule $(p \to q)$ je podformulí formule F''.

Věta 2.2 (Rozšířená věta o substituci)

Nechť F je formule výrokové logiky, která obsahuje podformuli A. Dále nechť B je formule výrokové logiky ekvivalentní s formulí A.

Formuli F' vytvoříme tak, že v F nahradíme podformuli A formulí B. Jestliže F je tautologie, pak i F' je tautologie.

Důkaz: Formule A a B jsou ekvivalentní, tedy při vyhodnocení s použitím stejné valuace dávají stejné výsledky, proto jestliže F je tautologie, pak po záměně B za A je také formule F' tautologie. □

Zatímco *Věta o substituci* byla o tom, že můžeme libovolně dosazovat za výrokové proměnné, *Rozšířená věta o substituci* nám říká, že lze dosazovat nejen za proměnné, ale také můžeme zaměnit podformuli za jinou s ní ekvivalentní. Musíme však vždy dát pozor, aby šlo opravdu o podformuli.

Příklad 2.2

Víme, že platí ekvivalence $(A \to B) \leftrightarrow (\neg B \to \neg A)$. Kdy ji lze použít podle Věty o rozšířené substituci? Vezměme si následující formule:

1. $F_1 = (p \to q)\ \&\ (\neg r \to \neg q) \to (p \to r)$
2. $F_2 = (p\ \&\ \neg q \to \neg p) \to (\neg p \vee q)$
3. $F_3 = \neg(p \vee q) \to \neg p\ \&\ \neg q$

V prvním případě můžeme ekvivalenci upravit na $(q \to r) \leftrightarrow (\neg r \to \neg q)$, to můžeme díky větě o substituci (zde pouze „přejmenováváme“ proměnné), a dosadit do formule F_1. Získáme formuli $F_1' = (p \to q)\ \&\ (q \to r) \to (p \to r)$. Původní formule byla tautologie, po úpravě jsme také získali tautologii (Kdo nevěří, může si ověřit například sémantickou tabulkou; ostatně, nepřipomíná to náhodou tranzitivitu?).

V druhém případě je sice ve formuli logická spojka implikace a před i za ní najdeme negované výrokové proměnné, ale pozor, nejedná se o podformuli! Implikace má mezi logickými spojkami menší prioritu než konjunkce (podobně jako například sčítání má menší prioritu než násobení u aritmetických operátorů), proto úprava formule F_2 na $(p\ \&\ p \to q) \to (\neg p \vee q)$ by byla chybou. Jak původní, tak i upravená formule jsou sice tautologie, přesto však jde o chybu.

Třetí formule ze seznamu, F_3, je tautologie. Opět si však můžeme všimnout, že není možné tuto formuli upravit podle ekvivalence uvedené na začátku příkladu – kdybychom formuli $\neg(p \vee q) \to \neg p$ považovali za podformuli a provedli bychom výše naznačenou úpravu, získali bychom formuli $p \to (p \vee q)\ \&\ \neg q$, která není s původní formulí ekvivalentní a není to tautologie (pozor na priority spojek).

Co z toho vyplývá? Nepodceňujme závorky a hojně je používejme, spoléhání na priority operátorů může způsobit nedorozumění nebo přehlédnutí.

2.2 Formální důkazy

Systém přirozené dedukce je předpokladový formální systém, proto nejsme omezeni pouze na přímé důkazy. Zde definujeme nejdřív základní typy důkazů – přímý a nepřímý, a pak se podíváme na jejich modifikace využívající hypotézy.

2.2.1 Přímý a nepřímý důkaz

Přímý důkaz je postaven na tom, že je dána množina předpokladů a naším úkolem je ověřit, zda z těchto předpokladů vyplývá zadaná formule s tím, že tato formule (cíl) bude posledním bodem důkazu (k ní celý důkaz směřuje). Důkaz je řádkový, tedy *posloupnost formulí* (obvykle každá na jednom řádku, proto hovoříme o řádkovém důkazu), mezi nimiž platí předem určené vztahy.

Definice 2.1 (Přímý důkaz z předpokladů)

Přímý důkaz formule F z daných předpokladů $P_1, P_2, \ldots, P_n$ je posloupnost formulí $A_1, A_2, \ldots, A_m$, kde $F = A_m$ a pro každé $i \in \{1, \ldots, m\}$ platí pro A_i některá z těchto možností:

- $A_i \in \{P_1, \ldots, P_n\}$ (tj. je to některý z předpokladů),
- A_i je axiom (dedukční pravidlo bez předpokladů),
- A_i vznikla použitím některého dedukčního pravidla na předchozí členy posloupnosti.

Přímý důkaz formule F (tj. bez předpokladů) je definován stejně, jen $n = 0$.

Věta 2.3 (Věta o dedukci)

Nechť P a Z jsou formule. Pak platí následující vztah:

$$P \vdash Z \quad \Leftrightarrow \quad \vdash P \to Z \tag{2.1}$$

Důkaz: Protože už známe definici důkazu, vytvoříme důkaz „uvnitř“ Systému přirozené dedukce s využitím této definice.

1) „$\Rightarrow$“: Předpokládáme, že platí $P \vdash Z$, chceme dokázat, že platí $\vdash P \to Z$.

1. P	Předpoklad	V důkazu je pouze pravidlo zavedení implikace.
2. Z	Odvozeno z předpokladu	
3. $P \to Z$	ZI(2)	

2) „$\Leftarrow$“: Předpokládejme, že platí $P \to Z$ a P, chceme odvodit Z.

1. $P \to Z$	Předpoklad 1	Použijeme pravidlo eliminace implikace na předchozí dva členy důkazu.
2. P	Předpoklad 2	
3. Z	EI(1,2)	

□

Poznámka:

Pokud větu o dedukci použijeme rekurzívně n-krát, získáme

$$\begin{aligned} P_1, P_2, \ldots, P_{n-1}, P_n \vdash Z \quad &\Leftrightarrow \quad P_1, P_2, \ldots, P_{n-1} \vdash P_n \to Z \\ &\vdots \\ P_1, P_2, \ldots, P_{n-1}, P_n \vdash Z \quad &\Leftrightarrow \quad \vdash P_1 \to (P_2 \to \ldots (P_n \to Z) \ldots) \end{aligned} \tag{2.2}$$

Všimněte si uzávorkování, které vzniklo při postupném uplatňování věty. Kdybychom některé závorky zapomněli, šlo by o jinou formuli než původní, nebyla by to ekvivalentní úprava.

V následujících příkladech najdeme věty, které budeme používat v dalších důkazech podle Věty o substituci. Jestliže se nám podaří dokázat ekvivalenci mezi předpokladem a závěrem (tj. provedeme důkaz „v obou směrech"), lze takovýto vztah použít i podle Věty o rozšířené substituci.

Příklad 2.3

Dokážeme platnost vztahu $A \to B,\ B \to C,\ A \ \vdash\ C$

1. $A \to B$ Př1
2. $B \to C$ Př2
3. A Př3
4. B EI(1,3)
5. C EI(2,4)

Podle věty o dedukci jsme tím dokázali také například vztahy

$$A \to B,\ B \to C \ \vdash\ A \to C \tag{2.3}$$
$$A \to B \ \vdash\ (B \to C) \to (A \to C) \tag{2.4}$$
$$B \to C \ \vdash\ (A \to B) \to (A \to C) \tag{2.5}$$
$$\vdash\ (A \to B) \to ((B \to C) \to (A \to C)) \tag{2.6}$$
$$\vdash\ (B \to C) \to ((A \to B) \to (A \to C)) \tag{2.7}$$
$$A \to B,\ A \ \vdash\ (B \to C) \to C \tag{2.8}$$
$$B \to C,\ A \ \vdash\ (A \to B) \to C \tag{2.9}$$
$$A \ \vdash\ (A \to B) \to ((B \to C) \to C) \tag{2.10}$$
$$A \ \vdash\ (B \to C) \to ((A \to B) \to C) \tag{2.11}$$

Vztah (2.3) se nazývá *věta o tranzitivitě implikace*. Často se používá v důkazech jako pomocné pravidlo, jeho použití značíme zkratkou TI.

Poznámka:

Věta o dedukci (str. 24) se často používá k úpravě formule, kterou chceme dokázat – formuli pomocí této věty rozdělíme na předpoklady a závěr, který pak dokazujeme z předpokladů.

Dále větu o dedukci použijeme, pokud chceme již dokázanou formuli použít jako pomocné pravidlo, nebo dokázané pravidlo jako logicky platnou formuli a také zařadit do důkazu.

Příklad 2.4

Dokážeme platnost vztahu $A, \neg A \vdash B$.

1.	A	Př1
2.	$\neg A$	Př2
3.	$A \vee B$	ZD(1)
4.	B	ED(2,3)

Ze sporné množiny vyplývá cokoliv, toto pomocné pravidlo nazýváme *pravidlo sporné množiny* a značíme SM.

Definice 2.2 (Nepřímý důkaz)

Nepřímý důkaz formule F z předpokladů $P_1, P_2, \ldots, P_n$ je posloupnost formulí $A_1, A_2, \ldots, A_m$, kde

- pro každé $i \in \{1, \ldots, n\}$ je $A_i = P_i$ (tj. nejdřív do důkazu zařadíme předpoklady),
- $A_{n+1} = \neg F$,
- pro každé $i > (n+1)$ pro A_i platí některá z těchto možností:
  - $A_i \in \{P_1, \ldots, P_n\}$ (tj. je to některý z předpokladů),
  - A_i je axiom (dedukční pravidlo bez předpokladů),
  - A_i vznikla použitím někt. dedukčního pravidla na předchozí členy posloupnosti,
- $A_m = \neg A_j$ pro některé $j < m$ (spor).

Nepřímý důkaz formule F (tj. bez předpokladů) je definován stejně, jen $n = 0$ a prvním členem posloupnosti je negace formule F.

Příklad 2.5

Dokážeme větu $\neg(A \vee B) \rightarrow (\neg A \,\&\, \neg B)$.

Použijeme větu o dedukci, tedy předpokladem pro nás bude část formule před implikací. Důkaz rozdělíme do tří částí. Nejdřív z předpokladu dokážeme závěr $\neg A$, potom $\neg B$, a v třetí části vše spojíme pravidlem zavedení konjunkce.

1.	$\neg(A \vee B)$	Př1
2.	A	NZ (negace závěru)
3.	$A \vee B \Rightarrow \neg A$	ZD(2), spor s 1.

(1.)	$\neg(A \vee B)$	Př1
4.	B	NZ (negace závěru)
5.	$A \vee B \Rightarrow \neg B$	ZD(4), spor s 1.

6.	$\neg A$	Př1
7.	$\neg B$	Př2
8.	$\neg A \,\&\, \neg B$	ZK(6,7)

Příklad 2.6

Dokážeme platnost vztahu $(\neg A \to \neg B) \vdash (B \to A)$

1.	$\neg A \to \neg B$	Př1
2.	B	Př2
3.	$\neg A$	NZ
4.	$\neg B$	EI(1,3), spor s 2.

Toto pomocné pravidlo budeme nazývat *pravidlo kontrapozice*, značíme PK.

Příklad 2.7

Dokážeme platnost vztahu $\neg\neg A \vdash A$.

1.	$\neg\neg A$	Př1
2.	$\neg\neg A \to (\neg A \to \neg\neg\neg A)$	VD na SM
3.	$\neg A \to \neg\neg\neg A$	EI(1,2)
4.	$(\neg A \to \neg\neg\neg A) \to (\neg\neg A \to A)$	VD na PK
5.	$\neg\neg A \to A$	EI(3,4)
6.	A	EI(1,5)

Toto pomocné pravidlo nazýváme *pravidlo eliminace negace*, značíme EN. Zkratka „VD“ znamená „Věta o dedukci“, „SM“ je pravidlo sporné množiny.

Dále již budeme předpokládat platnost pravidla eliminace negace, jeho důkazové kroky nemusíme uvádět v posloupnosti důkazu.

Příklad 2.8

Nepřímý důkaz téhož vztahu s využitím pravidla kontrapozice (příklad na str. 27):

1.	$\neg\neg A$	Př1
2.	$\neg A$	NZ
3.	$\neg\neg\neg\neg A \to \neg\neg A$	ZI(1)
4.	$\neg A \to \neg\neg\neg A$	PK(3)
5.	$\neg(\neg\neg A)$	EI(2,4), spor s 1.

Příklad 2.9

Dokážeme platnost vztahu $A \vdash \neg\neg A$.

1.	A	Př1
2.	$\neg(\neg\neg A)$	NZ
3.	$\neg A$	EN(2), spor s 1

Toto je pomocné *pravidlo zavedení negace*, značíme ZN. Taktéž ho nemusíme uvádět v posloupnosti důkazu.

Příklad 2.10

Dokážeme platnost vztahu $A \to B \vdash \neg B \to \neg A$.

1. $A \to B$	Př1	Toto je pomocné *pravidlo transpozice*, značíme PT.
2. $(\neg\neg A \to \neg\neg B)$	ZN(1)	
3. $\neg B \to \neg A$	PK(2)	

Poznámka:

Dvojice pravidel ZN a EN (str. 27) určuje ekvivalenci, a proto podle Rozšířené věty o substituci (str. 22) lze tato pravidla používat i „uvnitř" formulí – na podformule. Totéž platí pro pravidla PK (str. 27) a PT (str. 27).

Příklad 2.11

Dokážeme platnost vztahu $A \,\&\, B \;\vdash\; \neg(\neg A \vee \neg B)$

1. $A \,\&\, B$	Př1	Toto pomocné pravidlo nazýváme *pravidlo převodu konjunkce na disjunkci*, značíme KD.
2. $\neg A \vee \neg B$	NZ	
3. A	EK(1)	
4. B	EK(1)	
5. $\neg B$	ED(2,3), spor s 4	

Úkoly

1. Dokažte přímým důkazem v Systému přirozené dedukce větu
$$((p \to q) \,\&\, p) \to q$$
2. Dokažte větu
$$(p \to q) \,\&\, (q \to r) \to (p \to r)$$
Použijte Větu o dedukci, postupujte přímým důkazem.
3. Dokažte přímým důkazem vztah
$$A \vee B \;\vdash\; \neg A \to B$$
Tento vztah nazýváme *pravidlo převodu disjunkce na implikaci* a značíme DI.
4. Dokažte přímým důkazem větu
$$(p \to q) \to ((q \to r) \to (p \to r))$$
Dávejte pozor na uzávorkování. Tato věta se nazývá *hypotetický sylogismus* – to, co lze odvodit ze závěru, lze odvodit i z jeho předpokladu (resp. množiny předpokladů).
5. Proveďte nepřímý důkaz vztahu
$$A \to B \;\vdash\; \neg(A \,\&\, \neg B)$$
Tento vztah nazýváme *pravidlo převodu implikace na konjunkci*, značíme IK.

6. Proveďte nepřímý důkaz vztahu

$$A \to B, \neg B \ \vdash \ \neg A$$

(bez použití Věty o dedukci). Jedná se o pravidlo *Modus Tollens*, značíme MT.

2.2.2 Hypotézy

V systému přirozené dedukce můžeme s výhodou používat také hypotetické předpoklady – *hypotézy*. Důležité je si uvědomit, že platnost důsledku, ke kterému pomocí hypotézy dojdeme, je podmíněna platností této hypotézy.

 **Přímá hypotéza.** Mimo hlavní posloupnost důkazu může být uvedena hypotéza H. Jestliže na základě této hypotézy a předchozích členů posloupnosti důkazu lze odvodit formuli A, pak formuli $H \to A$ můžeme připojit k hlavní posloupnosti důkazu (tj. k řádnému důkazu) jako větu.

 **Příklad 2.12**

Dokážeme platnost věty $(p \,\&\, q \to r) \,\&\, q \to ((p \to r) \vee (q \to r))$, použijeme důkaz s hypotézou.

1. $(p \,\&\, q \to r) \,\&\, q$	Př1
2. $p \,\&\, q \to r$	EK(1)
3. q	EK(1)
(a) p	H1
(b) $p \,\&\, q$	ZK(3,a)
(c) r	EI(2,b)
4. $p \to r$	(a) $\to$ (c)
5. $(p \to r) \vee (q \to r)$	ZD(4)

Jak vidíme na příkladu, s hypotézou pracujeme zásadně mimo hlavní větev důkazu (vytvoříme si pomocnou větev značenou třeba písmeny místo číslic, ve které postupně odvodíme závěr hypotézy), pak se opět vracíme do hlavní větve důkazu.

Na pravdivost či platnost hypotézy neklademe žádné požadavky, může dokonce jít i o kontradikci. Výsledná implikace je již samozřejmě formulí odvozenou z uvedených předpokladů.

 Přímou hypotézu používáme tehdy, když do důkazu potřebujeme zařadit implikaci, kterou lze tímto způsobem získat.

 **Věta 2.4 (Věta o přímé hypotéze)**

V důkazu lze použít přímou hypotézu, tj. jestliže z hypotézy H lze odvodit formuli A, pak do posloupnosti důkazu můžeme přidat formuli $H \to A$. Formálně:

$$P, H \ \vdash \ A \ \Rightarrow \ P \ \vdash \ H \to A$$

Důkaz: Důkaz s přímou hypotézou převedeme na přímý důkaz. Uvedený vztah lze jednoduše chápat jako použití Věty o dedukci. Znamená to, že pokud stanovíme hypotézu (jakkoliv podle potřeby) a odvodíme z této hypotézy a dalších předpokladů závěr A, pak můžeme uvedenou implikaci přidat do posloupnosti důkazu (Hypotézu nelze nechat v množině předpokladů!). □

Nepřímá hypotéza. Mimo hlavní posloupnost důkazu může být uvedena hypotéza H. Jestliže na základě této hypotézy a předchozích členů posloupnosti důkazu lze odvodit formuli A, která je ve sporu s některým členem posloupnosti řádného důkazu, pak k hlavní posloupnosti důkazu můžeme jako větu připojit formuli $\neg H$.

Příklad 2.13

Dokážeme platnost věty $\neg(p \vee q) \rightarrow (\neg p \,\&\, \neg q)$, použijeme nepřímou hypotézu (a dokonce na dvou místech v důkazu).

1. $\neg(p \vee q)$	Př1		
(a) p	H1	(a) q	H2
(b) $p \vee q$	ZD(a), spor s 1	(b) $p \vee q$	ZD(a), spor s 1
2. $\neg p$	¬ H1	3. $\neg q$	¬ H2
		4. $\neg p \,\&\, \neg q$	ZK(2,3)

Nepřímá hypotéza je vlastně pomocný nepřímý důkaz provedený mimo hlavní posloupnost důkazu. S hypotézou zacházíme jako s negací závěru, kterou chceme popřít.

Typicky se tento typ důkazu používá, pokud máme v závěru dokazované formule negaci nebo konjunkci negací (stejně jako u nepřímého důkazu).

Úkoly

1. Srovnejte důkaz pravidla DK (převod disjunkce na konjunkci v příkladu na straně 26 (nepřímým důkazem) a důkaz téhož pravidla v příkladu na této straně (použití nepřímé hypotézy). Které kroky přímého důkazu odpovídají použití hypotéz?
2. Dokažte pravidlo $A \,\&\, B \;\vdash\; \neg(A \rightarrow \neg B)$ (*pravidlo převodu konjunkce na implikaci*, značíme KI). Nejdřív proveďte nepřímý důkaz, pak nepřímý důkaz s hypotézou.

2.2.3 Větvené důkazy

Princip větveného důkazu vychází ze zpracování disjunkce podformulí, která je již součástí důkazu. Každou z podformulí použijeme jako hypotézu, z každé hypotézy odvodíme stejný typ závěru.

Přímý větvený důkaz s hypotézami. Jestliže se v posloupnosti důkazu nachází formule ve tvaru $B_1 \vee B_2 \vee \ldots \vee B_m$ a formuli A lze dokázat ze všech hypotéz – podformulí B_i, $i \in \{1 \ldots m\}$, pak k hlavní posloupnosti důkazu můžeme připojit formuli A.

Příklad 2.14

Dokážeme platnost věty $((p \,\&\, q) \vee (p \,\&\, r)) \;\rightarrow\; (p \,\&\, (q \vee r))$

1. $(p \,\&\, q) \vee (p \,\&\, r)$ Př1

(a) $p \,\&\, q$	H1	(a) $p \,\&\, r$	H2
(b) p	EK(a)	(b) p	EK(a)
(c) q	EK(a)	(c) r	EK(a)
(d) $q \vee r$	ZD(c)	(d) $q \vee r$	ZD(c)
(e) $p \,\&\, (q \vee r)$	ZK(b,d)	(e) $p \,\&\, (q \vee r)$	ZK(b,d)

2. $p \,\&\, (q \vee r)$

Poznámka:

Jak vidíme na příkladu na straně 31, pro větvení si vybereme vždy některou formuli z důkazu, která je disjunkcí podformulí. Zde se jedná o disjunkci podformulí $p \,\&\, q$ a $p \,\&\, r$. Jestliže v posloupnosti důkazu žádná vhodná formule není, můžeme použít jeden z axiomů (dedukčních pravidel bez předpokladů), a to $A \vee \neg A$ (se substitucí některé formule za A).

U přímého větveného důkazu se vlastně jedná o zkrácení postupu použití přímé hypotézy, kdy bychom dostali v hlavní posloupnosti důkazu věty $B_i \rightarrow A$ (*„formuli A lze dokázat ze všech hypotéz – podformulí B_i“*). Větu lze zformulovat takto:

Věta 2.5 (Věta o přímém větveném důkazu)

Z platnosti formule $B_1 \vee B_2 \vee \ldots \vee B_m$ a formulí $B_1 \rightarrow A,\ B_2 \rightarrow A, \ldots, B_m \rightarrow A$ plyne platnost formule A.

Důkaz: Větu dokážeme pro $m = 2$, a to nepřímým důkazem (předpoklady $B_1 \rightarrow A$ a $B_2 \rightarrow A$ byly získány uplatněním přímé hypotézy).

1. $B_1 \vee B_2$ Př1
2. $B_1 \rightarrow A$ Př2
3. $B_2 \rightarrow A$ Př3
4. $\neg A \rightarrow \neg B_1$ PT(2)
5. $\neg A \rightarrow \neg B_2$ PT(3)
6. $\neg A$ NZ (negace závěru)
7. $\neg B_1$ EI(4,6)
8. $\neg B_2$ EI(5,6)
9. B_2 ED(1,7), spor s 8.

V důkazu využíváme pravidla transpozice, eliminace implikace a eliminace dedukce.
Protože negace závěru $\neg A$ byla popřena, dokázali jsme platnost závěru A.

□

Nepřímý větvený důkaz s hypotézami. Jestliže se v posloupnosti nepřímého důkazu nachází formule ve tvaru $B_1 \vee B_2 \vee \ldots \vee B_m$ a ze všech hypotéz – podformulí B_i pro $i \in \{1 \ldots m\}$ docházíme ke sporu s některým členem hlavní posloupnosti důkazu (tedy všechny B_i jsou vyvráceny), pak jsme dospěli ke sporu i v hlavní posloupnosti důkazu a původní formule je dokázána.

Příklad 2.15

Dokážeme platnost věty $(\neg(r \to p) \,\&\, (q \to \neg s) \,\&\, s) \to \neg\,(p \vee q \vee (\neg s \,\&\, p))$

1. $\neg(r \to p)$ Př1
2. $(q \to \neg s)$ Př2
3. s Př3
4. $p \vee q \vee (\neg s \,\&\, p)$ NZ NZ = H1 ∨ H2 ∨ H3

(a) p	H1	(a) q	H2	(a) $\neg s \,\&\, p$	H3
(b) $r \to p$	ZI(a)	(b) $\neg s$	EI(2,a)	(b) $\neg s$	EK(a)
	spor s 1.		spor s 3.		spor s 3.

$\Rightarrow$ *spor* (ve všech větvích), věta je dokázána.

Jak vidíme, nepřímý větvený důkaz je vlastně nepřímý důkaz (protože formule $B_1 \vee B_2 \vee \ldots \vee B_m$ často bývá v závěru), kde zároveň používáme větvení. Popřením všech větví důkazu přidáváme k hlavní posloupnosti důkazu formuli, která je ve sporu s některým předchozím členem posloupnosti.

Tento typ důkazu používáme, pokud v závěru dokazované formule je negace disjunkce, tak jak to bylo v příkladu na str. 32.

Úkoly

1. Přímým větveným důkazem dokažte rezoluční pravidlo. Větvení založte na axiomu.
2. Dokažte přímým větveným důkazem pravidlo $A \to B \quad \vdash \quad \neg A \vee B$ (*pravidlo převodu implikace na disjunkci*, ID).

2.3 Vlastnosti Systému přirozené dedukce výrokové logiky

Jak již víme, formální systémy mohou mít různé vlastnosti – korektnost, úplnost, bezespornost, minimálnost. V případě Systému přirozené dedukce VL si ukážeme, že splňuje vlastnosti korektnost a úplnost. Protože každý korektní systém je bezesporný, bude tím dána i bezespornost tohoto systému.

2.3.1 Korektnost

Věta 2.6 (Věta o korektnosti Systému přirozené dedukce výrokové logiky)
Každá formule dokazatelná v Systému přirozené dedukce výrokové logiky je logicky platná, tedy pro každou formuli A výrokové logiky platí:

$$\vdash A \quad \Longrightarrow \quad \models A \tag{2.12}$$

Důkaz: Dokazujeme, že důkaz provedený v tomto systému je korektní, tedy že všechna dedukční pravidla zachovávají pravdivost (pro každé ohodnocení, při kterém jsou pravdivé předpoklady, je pravdivý i závěr), a dále že samotná konstrukce důkazu (řazení posloupnosti důkazu) je korektní.

Korektnost dedukčních pravidel lze dokázat sémantickými tabulkami (zde například pro jeden axiom a dedukční pravidla ZK a EI):

$\vdash A \vee \neg A$

A	$\neg A$	$A \vee \neg A$
0	1	1
1	0	1

$A, B \vdash A \,\&\, B$

A	B	$A \,\&\, B$
0	0	0
0	1	0
1	0	0
1	1	1

$A \to B, A \vdash B$

A	B	$A \to B$	A	B
0	0	1	0	0
0	1	1	0	1
1	0	0	1	0
1	1	1	1	1

Tabulka 2.2: Sémantické tabulky pro některá dedukční pravidla

V případě pravidel bez předpokladů (axiomů) musí být v posledním sloupci pouze hodnoty 1, u dedukčních pravidel s předpoklady musí být v posledním sloupci (závěru pravidla) hodnoty 1 v těch řádcích, kde jsou hodnoty 1 ve všech sloupcích označených předpoklady pravidla (říkáme, že pravidlo zachovává pravdivost – závěr je pravdivý v každém ohodnocení, ve kterém jsou pravdivé předpoklady).

Pokud takto vyjdou tabulky pro všechna dedukční pravidla, pak je dokázána korektnost dedukčních pravidel systému.

Korektnost přímého důkazu vyplývá z jeho definice. Protože se jedná o konečnou *posloupnost* formulí, použijeme důkaz matematickou indukcí.

Jestliže jde o *důkaz z předpokladů* (je dán alespoň jeden předpoklad), pak dokazujeme, že pravdivost závěru je podmíněna pravdivostí předpokladů, tedy v každém ohodnocení, ve kterém jsou předpoklady pravdivé, musí být pravdivý i závěr.

Báze indukce: Z definice vyplývá, že první člen posloupnosti je axiom nebo předpoklad. Platnost axiomů jsme dokázali výše, pravdivostí předpokladů je podmíněna pravdivost závěru pro dané ohodnocení, proto nás dále zajímají pouze ta ohodnocení, ve kterých jsou všechny předpoklady pravdivé.

Předpoklad indukce: předpokládejme, že je věta dokázána až ke k-tému členu posloupnosti důkazu, tedy až k této formuli včetně jsou členy posloupnosti buď předpoklady nebo logicky platné formule (nebo formule, jejichž pravdivost je podmíněna pravdivostí předpokladů pro dané ohodnocení).

Krok indukce: podívejme se na člen s indexem $k+1$. Jestliže se jedná o axiom, pak je jeho logická platnost již dokázána.

Pokud tento člen vznikl aplikací některého dedukčního pravidla na předchozí členy posloupnosti, pak toto pravidlo bylo použito na logicky platné formule nebo na formule pravdivé pro všechna ohodnocení, ve kterých jsou pravdivé všechny předpoklady (formule z členů důkazu do indexu k). Výše bylo ukázáno, že všechna dedukční pravidla zachovávají pravdivost, tedy v posloupnosti důkazu na místě $k+1$ je formule, která je logicky platná nebo pravdivá ve všech ohodnoceních, ve kterých jsou pravdivé všechny předpoklady až dosud v důkazu uvedené.

Pokud krok indukce použijeme postupně na celou posloupnost důkazu, zjistíme, že poslední člen posloupnosti je formule pravdivá ve všech ohodnoceních, ve kterých jsou pravdivé všechny předpoklady.

□

Princip nepřímého důkazu vychází z věty na straně 13. Je podložen zdůvodněním pomocí sémantické tabulky, tedy není třeba tento typ důkazu zvlášť dokazovat.

Důkaz s využitím hypotéz a větvení jsme odvodili z přímého důkazu v předchozích kapitolách (věty na str. 29 a 31), proto je taktéž nemusíme dokazovat.

Souhrn: korektnost Systému přirozené dedukce výrokové logiky se dokazuje takto:

1. předpokládáme, že formule A je dokazatelná v SPD VL (tj. $\vdash A$), chceme dokázat, že je logicky platná (tj. $\models A$),
2. dokážeme korektnost *všech* dedukčních pravidel, například sémantickou tabulkou,
3. dokážeme korektnost postupu konstrukce přímého důkazu – jde o posloupnost, tedy použijeme důkaz matematickou indukcí.

Úkol

Vytvořte sémantické tabulky pro zbývající dedukční pravidla.

2.3.2 Úplnost a bezespornost

Protože již máme dokázánu korektnost, lze následující důkazy bez problémů provádět uvnitř Systému přirozené dedukce. Než dokážeme úplnost Systému přirozené dedukce výrokové logiky, budeme potřebovat dvě pomocné věty (lemmata).

Lemma 2.7 (Lemma o neutrální formuli)

$$B \vdash A, \quad \neg B \vdash A \quad \Longrightarrow \quad \vdash A \tag{2.13}$$

Tedy jestliže lze formuli dokázat z některé formule i její negace, pak formule z předpokladů již nemusíme do posloupnosti důkazu zapisovat.

Důkaz: Při dokazování tohoto lemmatu použijeme metodu větvení.

1. $B \to A$ VD na Př1
2. $\neg B \to A$ VD na Př2
3. $B \vee \neg B$ A (H1 $\vee$ H2)
   (a) B H1
   (b) A EI(1,a)

   (a) $\neg B$ H2
   (b) A EI(2,a)
4. A (z větveného důkazu)

□

Lemma 2.8 (Lemma o sémantice a dokazatelnosti)

Nechť F je formule obsahující právě výrokové proměnné $p_1, p_2, \ldots, p_n$. Ve zvolené valuaci v označme:

$F' = F$, pokud $I(F, v) = 1$, *$p'_i = p_i$, pokud $v(p_i) = 1$,*
$F' = \neg F$, pokud $I(F, v) = 0$ *$p'_i = \neg p_i$, pokud $v(p_i) = 0$ pro každé $i \in \{1, \ldots, n\}$*

Pak platí

$$p'_1, p'_2, \ldots, p'_n \vdash F' \tag{2.14}$$

Důkaz: Důkaz provedeme matematickou indukcí podle složitosti formule (podle počtu logických spojek).

Báze indukce: $F = p$ (0 logických spojek)
$p' \vdash p'$ platí, protože jde o substituci do dedukčního pravidla A2 ($\vdash A \to A$).

Předpoklad indukce: předpokládejme, že věta platí pro formule B, C o složitosti nejvýše k:
$p'_1, p'_2, \ldots, p'_n \vdash B'$
$p'_1, p'_2, \ldots, p'_n \vdash C'$

Krok indukce: dokážeme, že věta platí pro formuli F hloubky $k + 1$. Důkaz povedeme pouze pro dvě logické spojky – negaci a implikaci.

a) $F = \neg B$, B je formule o složitosti k:

   Máme dokázat, že platí $p'_1, p'_2, \ldots, p'_n \vdash (\neg B)'$. Protože platí předpoklad indukce, dokazujeme $B' \vdash (\neg B)'$. Z interpretace vyplývají dvě možnosti:

   1) $I(B) = 0 \Rightarrow$ dokazujeme $\neg B \vdash \neg B$, což je tautologie (axiom),
   2) $I(B) = 1 \Rightarrow$ dokazujeme $B \vdash \neg\neg B$, to je dokázáno na str. 27 jako pomocné pravidlo zavedení negace (ZN).

b) $F = B \to C$, kde B, C jsou formule složitosti nejvýše k:

   Vytvoříme tabulku (viz následující strana) pro jednotlivé valuace proměnných (resp. interpretace podformulí) B a C, třetí sloupec tabulky přísluší formuli F, do čtvrtého sloupce pak dosadíme (interpretujeme zobrazení „čárka“). Jednotlivé řádky pak určují věty, které je třeba dokázat.

B	C	$F = B \to C$	*dokazujeme* $(B', C' \vdash F')$*:*	
0	0	1	$\neg B, \neg C \vdash B \to C$	①
0	1	1	$\neg B, C \vdash B \to C$	②
1	0	0	$B, \neg C \vdash \neg(B \to C)$	③
1	1	1	$B, C \vdash B \to C$	④

Tabulka 2.3: Určení tvrzení, která je třeba dokázat

①, ②: Dokazujeme $\neg B \vdash B \to C$, neboli $\neg B, B \vdash C$ (podle věty o dedukci)
$\Rightarrow$ dokázáno, pravidlo sporné množiny (viz str. 26).

③: Dokážeme nepřímým důkazem:

1. B Př1
2. $\neg C$ Př2
3. $B \to C$ NZ
4. C EI(1,3), spor s 2.

④: Dokážeme přímým důkazem:

1. C Př1
2. $B \to C$ ZI(1)

Platnost indukčního kroku jsme dokázali pro negaci a implikaci. Dále můžeme buď totéž provést pro ostatní logické spojky (konjunkci, disjunkci a ekvivalenci) nebo dokázat věty o existenci ekvivalentních formulí, které obsahují pouze spojky negace a implikace.

□

Věta 2.9 (Věta o úplnosti Systému přirozené dedukce VL)

Každá logicky platná formule výrokové logiky je dokazatelná v Systému přirozené dedukce výrokové logiky, tedy platí

$$\models A \quad \Longrightarrow \quad \vdash A \tag{2.15}$$

Důkaz: Předpokládejme, že A je tautologie
$\Rightarrow$ A je pravdivá při každém ohodnocení ($A \Leftrightarrow A'$), pro všechna ohodnocení platí

$$p'_1, p'_2, \ldots, p'_n \vdash A$$

$\Rightarrow$ Pak platí zároveň tyto věty:

$$p_1, p'_2, \ldots, p'_n \vdash A$$
$$\neg p_1, p'_2, \ldots, p'_n \vdash A$$

$\Rightarrow$ Podle lemmatu o neutrální formuli (na straně 34) pak platí

$$p'_2, \ldots, p'_n \vdash A$$

Stejně budeme postupovat i pro další výrokové proměnné, po n uplatněních tohoto postupu získáme větu $\vdash A$. Tím je vztah (2.15) dokázán.

□

Věta 2.10 (Věta o bezespornosti Systému přirozené dedukce VL)
Systém přirozené dedukce výrokové logiky je bezesporný.

Důkaz: Větu dokážeme sporem. Kdyby byl tento systém sporný, pak by existovala formule A taková, že $\vdash A$ a zároveň $\vdash \neg A$. Potom ale podle věty o korektnosti Systému přirozené dedukce výrokové logiky (na straně 33) platí $\models A$ a zároveň $\models \neg A$, což ale není možné. Proto je Systém přirozené dedukce výrokové logiky bezesporný. □

Úkol
Dokončete důkaz Lemmatu o sémantice a dokazatelnosti – proveďte důkazy pro zbývající logické spojky.

2.4 Systém přirozené dedukce predikátové logiky

Jazyk Systému přirozené dedukce predikátové logiky přejímáme z predikátové logiky prvního řádu (PL_1).

Dedukční pravidla přejímáme ze Systému přirozené dedukce výrokové logiky a přidáváme další čtyři pravidla, která jsou uvedena v tabulce 2.4.

1.-10.	Dedukční pravidla přejatá ze Systému přirozené dedukce VL, viz str. 21		
11.	$A(x) \vdash \forall x A(x)$	Z∀	Zavedení obecného kvantifikátoru
12.	$\forall x A(x) \vdash A(x/t)$	E∀	Eliminace obecného kvantifikátoru
13.	$A(x/t) \vdash \exists x A(x)$	Z∃	Zavedení existenčního kvantifikátoru
14.	$\exists x A(x) \vdash A(c)$	E∃	Eliminace existenčního kvantifikátoru

Tabulka 2.4: Dedukční pravidla Systému přirozené dedukce predikátové logiky

Na rozdíl od pravidel pro výrokovou logiku tato pravidla mají svá *omezení*:

1) Pro pravidla E∀ a Z∃: term t musí být *substituovatelný* za x.
2) Pro pravidlo E∃: c je individuová konstanta, která v posloupnosti důkazu dosud nebyla použita.

   Zde si můžeme vzpomenout na *Herbrandovu proceduru*, kterou známe z předchozího semestru; c pro nás ve skutečnosti neznamená konkrétní individuovou konstantu jako je například jméno člověka, ale je to pouze označení některé z konstant obsažených v univerzu. Musíme vybrat takové označení, o kterém jsme si jisti, že neukazuje na tentýž prvek univerza (to si můžeme představit jako množinu objektů) jako předchozí použité konstanty.
3) Pro pravidlo E∃: jestliže A obsahuje volné proměnné $y_1, \ldots, y_n$ (a žádné jiné), pak má pravidlo formu
$$\exists x A(x, y_1, \ldots, y_n) \vdash A(f(y_1, \ldots, y_n), y_1, \ldots, y_n),$$

tedy volné proměnné ponecháme a místo vázané proměnné dosadíme funkční symbol o n proměnných (n je počet volných proměnných), který v posloupnosti důkazu dosud nebyl použit. Pokud je vázaných proměnných více, dosadíme funkční symbol za každou z nich, ale pokaždé jiný. To odpovídá procesu skolemizace (opět odkazujeme na předmět Úvod do logiky).

Další věty a definice přejímáme ze Systému přirozené dedukce výrokové logiky, a to včetně různých typů důkazů (přímý, nepřímý, hypotézy, větvený důkaz).

Příklad 2.16

Dokážeme větu $\forall x\,(A(x) \to B(x)) \to (\forall x A(x) \to \forall x B(x))$, použijeme přímý důkaz.

1.	$\forall x\,(A(x) \to B(x))$	Př1
2.	$\forall x A(x)$	Př2
3.	$A(x) \to B(x)$	E∀(1)
4.	$A(x)$	E∀(2)
5.	$B(x)$	EI(3,4)
6.	$\forall x B(x)$	Z∀(5)

Větu o dedukci uplatníme na formuli dvakrát, tak získáme dva předpoklady a závěr omezíme na $\forall x B(x)$.

Proměnná x je substituovatelná za x, i když se po uplatnění pravidla eliminace kvantifikátoru stává volnou proměnnou.

Příklad 2.17

Dokážeme větu $\neg\forall x A(x) \to \exists x \neg A(x)$ nepřímým důkazem s přímou hypotézou.

1.	$\neg\forall x A(x)$	Př1
2.	$\neg\exists x \neg A(x)$	NZ
	(a) $\neg A(x)$	H1
	(b) $\exists x \neg A(x)$	Z∃(a)
3.	$\neg A(x) \to \exists x \neg A(x)$	(a → b)
4.	$\neg\exists x \neg A(x) \to A(x)$	PT(3)
5.	$A(x)$	EI(2,4)
6.	$\forall x A(x)$	Z∀(5), spor s 1.

Příklad 2.18

Dokážeme větu $\exists x \neg A(x) \to \neg\forall x A(x)$ nepřímým důkazem.

1.	$\exists x \neg A(x)$	Př1
2.	$\forall x A(x)$	NZ
3.	$\neg A(c)$	E∃(1)
4.	$A(c)$	E∀(2), spor s 3.

Poznámka:

Z důkazů v předchozích dvou příkladech vyplývá věta

$$\exists x \neg A(x) \quad \leftrightarrow \quad \neg \forall x A(x),$$

tedy ekvivalence, proto podle věty o rozšířené substituci můžeme vztahy dokázané ve větách používat pro nahrazování podformulí.

Příklad 2.19

Dokážeme vztah $\exists x A(x) \rightarrow \exists x B(x) \quad \vdash \quad \forall x(A(x) \rightarrow B(m))$, kde m je individuová konstanta, přímým důkazem s hypotézou.

1. $\exists x A(x) \rightarrow \exists x B(x)$ Př1
   (a) $A(x)$ H1
   (b) $\exists x A(x)$ Z∃(a)
   (c) $\exists x B(x)$ EI(1,b)
   (d) $B(m)$ E∃(c)
2. $A(x) \rightarrow B(m)$ (a → d)
3. $\forall x(A(x) \rightarrow B(m)$ Z∀(2)

m je individuová konstanta, která v důkazu dosud nebyla použita.

Úkoly

1. Dokažte vztah $\forall x(A(x) \rightarrow B(x)) \quad \vdash \quad (\exists x A(x) \rightarrow \exists x B(x))$ přímým důkazem. Můžete použít Větu o dedukci. Vezměte také v úvahu, který z kvantifikátorů lze eliminovat jako první.
2. Dokažte vztah $\forall x(A \vee B(x)) \quad \vdash \quad A \vee \forall x B(x)$, kde x není volnou proměnnou v A. Použijte větvený důkaz (větvení podle axiomu).
3. Dokažte vztah $A \vee \forall x B(x) \quad \vdash \quad \forall x(A \vee B(x))$, kde x není volnou proměnnou v A. Použijte větvený důkaz (větvení podle předpokladu).

2.5 Vlastnosti Systému přirozené dedukce predikátové logiky

Věta 2.11 (Věta o korektnosti Systému přirozené dedukce predikátové logiky)

Každá formule dokazatelná v Systému přirozené dedukce predikátové logiky je logicky platná, tedy pro každou formuli A predikátové logiky platí:

$$\vdash A \quad \Longrightarrow \quad \models A \tag{2.16}$$

Důkaz: Oproti důkazu věty o korektnosti Systému přirozené dedukce výrokové logiky stačí dokázat korektnost přidaných dedukčních pravidel, zbytek důkazu můžeme přejmout. Korektnost těchto pravidel již nelze dokázat sémantickou tabulkou, místo toho použijeme přímý důkaz (úvahou). Využijeme toho, že během dokazování se vždy pohybujeme v rámci jedné struktury.

Pravidlo č. 11, $A(x) \vdash \forall x A(x)$ *(Z∀):*

Provedeme důkaz logickou úvahou podle interpretace ve struktuře $\mathcal{S}$: pokud platí $\models A(x)[\mathcal{S}])$ (formule je splněna ve struktuře $\mathcal{S}$ při kterékoliv valuaci e), znamená to, že formule A je platná (splněna) pro jakékoliv ohodnocení a interpretaci (ať za x dosadíme cokoliv, výsledkem je vždy formule platná v dané struktuře $\mathcal{S}$), zapisujeme $I(\forall x A[\mathcal{S}, e(x/a)] = true$ pro kterýkoliv prvek univerza diskurzu a, tedy platí také $\models \forall x A(x)[\mathcal{S}]$.

Pravidlo č. 12, $\forall x A(x) \vdash A(x/t)$ *(E∀):*

Předpoklad $\forall x A(x)$ znamená, že po dosazení libovolného prvku univerza diskurzu do formule A za x dostaneme vždy formuli interpretovanou jako *true*. Proto když za x dosadíme term t, který je substituovatelný za x, je také formule $A(x/t)$ interpretována s výsledkem *true*.

Pravidlo č. 13, $A(x/t) \vdash \exists x A(x)$ *(Z∃):*

Jestliže po dosazení t za x je formule A interpretována s výsledkem *true* v dané struktuře, pak existuje alespoň jeden prvek univerza diskurzu ($e'(t)$), kterým lze formuli $A(x)$ interpretovat, což znamená $\exists x A(x)$.

Pravidlo č. 14, $\exists x A(x) \vdash A(c)$ *(E∃):*

Jestliže existuje prvek univerza diskurzu, který můžeme dosadit za x a formule A je pak ve zvolené struktuře $\mathcal{S}$ interpretována jako *true*, pak tento prvek můžeme „nějak nazvat“. Víme, že tento prvek existuje, ale nevíme, který to je, proto musíme zvolit název, který jsme dosud nepoužili.

Toto pravidlo ve skutečnosti *nezachovává pravdivost* – jde především o konstantu c, protože přímo v předpisu pravidla není stanoveno, že konstanta c nesmí být použita v předchozí posloupnosti důkazu.

Dále bychom měli dokázat korektnost celého důkazového postupu. Definice důkazů jsme přejali ze Systému přirozené dedukce výrokové logiky a máme dokázánu korektnost dedukčních pravidel (vlastně až na pravidlo č. 14). Můžeme přejmout také důkaz korektnosti důkazového postupu s doplněním o důkazy korektnosti přidaných dedukčních pravidel s tím, že dedukční pravidlo č. 14 (E∃) sice nezachovává pravdivost, ale při dodržení podmínky výběru takové konstanty c, která v posloupnosti důkazu ještě nebyla použita, zachovává pravdivost alespoň důkazový postup jako celek a výsledná formule je pravdivá ve všech modelech, ve kterých jsou pravdivé předpoklady důkazu.

□

Věta 2.12 (Věta o úplnosti Systému přirozené dedukce predikátové logiky)

Každá logicky platná formule predikátové logiky je dokazatelná v Systému přirozené dedukce predikátové logiky, tedy platí

$$\models A \quad \Longrightarrow \quad \vdash A \tag{2.17}$$

Důkaz: Pokud rozšíříme důkaz vztahu (2.14) na straně 35 o kvantifikátory, můžeme přejmout celý důkaz věty o úplnosti Systému přirozené dedukce výrokové logiky ze strany 36. □

Poznámka:

Větu o bezespornosti můžeme také přejmout včetně celého důkazu.

Kapitola 3

Klauzulární logika

Rychlý náhled: Až dosud jsme formální systémy stavěli na výrokové logice nebo predikátové logice prvního řádu. Později budeme definovat další formální systém, ale tentokrát nad klauzulární logikou.

Klauzulární logika vychází z predikátové logiky, přejímá mnohé prvky jejího jazyka a některé mechanismy vyhodnocování pravdivosti, ale některé syntaktické prvky nepřejímá a nahrazuje je jinými, které jsou sice obdobné, ale přizpůsobené pro snadnější automatizaci odvozování.

Klauzulární logika a na ní postavený Klauzulární axiomatický systém budou přímo navrženy tak, aby byly použitelné pro logické programování. Pro nás bude klauzulární logika mezistupněm mezi predikátovou logikou a logickým programováním.

Klíčová slova: Klauzulární logika, klauzule, hornova klauzule, univerzální a existenční tvrzení, struktura, interpretace, pravdivá klauzule, platná klauzule, popírající množina, substituce, unifikace, množina neshod, znalostní báze, fakt, aplikovatelná struktura, model znalostní báze.

Cíle studia: Po prostudování této kapitoly zvládnete práci s klauzulemi klauzulární logiky, převod vět přirozeného jazyka do formy klauzulí, vytvoření znalostní báze a její používání při vyvozování důsledků.

3.1 Hornovy klauzule

V klauzulární logice budeme pracovat s klauzulemi ve speciálním tvaru, které budeme nazývat Hornovy klauzule. Nejdřív si připomeneme definici klauzule ve výrokové logice.

Definice 3.1 (Klauzule)

Klauzule jsou definovány induktivně:

- *Báze:* Literály jsou klauzule (literál – viz str. 8).
- *Indukce:* Jestliže A a B jsou klauzule, pak $A \vee B$ je také klauzule.
- *Zobecnění:* Všechny formule, které jsou utvořeny použitím konečného počtu pravidel v bázi a indukci, jsou klauzule, žádná jiná formule není klauzule.

Definice 3.2 (Hornovy klauzule)

Hornovy klauzule jsou klauzule s nejvýše jedním pozitivním literálem (tj. literálem bez negace). Můžeme je psát v následujících tvarech:

$$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \vee B \tag{3.1}$$

$$(A_1, A_2, \ldots, A_n) \rightarrow B \tag{3.2}$$

V klauzulární logice (ale ne ve všech logických jazycích) používáme obecné klauzule s jakýmkoliv počtem pozitivních literálů, jen je zapisujeme podobně jako Hornovy klauzule:

$$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \vee B_1 \vee B_2 \vee B_m \tag{3.3}$$

$$(A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_n) \rightarrow (B_1 \vee B_2 \vee \ldots \vee B_m) \tag{3.4}$$

3.2 Syntaxe jazyka klauzulární logiky

Syntaxe jazyka klauzulární logiky je podobná syntaxi jazyka predikátové logiky prvního řádu. Hlavním rozdílem je používání jediné logické spojky – implikace. Nepoužíváme dokonce ani negaci, samotnou negaci reprezentujeme jiným způsobem. Nejsou definovány také žádné kvantifikátory, pro reprezentaci univerzálních a existenčních tvrzení opět budeme používat jiné mechanismy.

3.2.1 Jazyk klauzulární logiky

Definujeme nejdřív abecedu klauzulární logiky, pak z prvků abecedy utvoříme termy, atomy a klauzule.

Definice 3.3 (Abeceda jazyka klauzulární logiky)

Abeceda jazyka klauzulární logiky zahrnuje následující symboly:

- *Proměnné* – začínají velkým písmenem, například X, Prom, Matka, Pocet, Soucet, Zvire, Smer_cesty
- *Individuové konstanty* – čísla nebo začínají malým písmenem, například 28, 3.14159, jaguár, lenka (pozor, i vlastní jména a jiné názvy musí začínat malým písmenem, pokud jsou to konstanty)
- *Logické konstanty* – true (t, 1), false (f, 0)
- *Existenční (Skolemovy) konstanty* – začínají vždy symbolem @, například $@a$, $@e$, $@x$, $@neco$
- *Funkční symboly* (funktory) – začínají obvykle malým písmenem, každý funktor má přiřazenu *aritu* – přirozené číslo (včetně nuly) určující počet argumentů funktoru, např. soucet(X,Y,S) má aritu 3, zapisujeme soucet/3
- *Existenční (Skolemovy) funktory* – také mají aritu, například $@f$ / 3
- *Predikátové symboly* – začínají písmenem, taktéž mají aritu
- *Logická spojka* – implikace ($\rightarrow$)
- *Pomocné symboly* – čárky, závorky, případně složené závorky

Definice 3.4 (Termy a atomy)

Termy jazyka klauzulární logiky definujeme induktivně:

- *Báze:* Každá proměnná, individuová konstanta a existenční konstanta je term.
- *Indukce:* Každý funkční symbol, jehož argumenty jsou termy, je term. Každý existenční funktor, jehož argumenty jsou termy, je term.
- *Zobecnění:* Každý term vznikne pouze konečným počtem použití pravidel stanovených v bázi a indukci, nic jiného není term.

Atom jazyka klauzulární logiky může být v jednom z těchto tvarů:

- logická konstanta, nebo
- jestliže p je n-ární predikát a $t_1, \ldots, t_n$ jsou termy, pak $p(t_1, \ldots, t_n)$ je atom. Termy $t_1, \ldots, t_n$ jsou parametry (argumenty) tohoto atomu.

Bázový term je term, v němž se nevyskytuje žádná proměnná ani existenční term. *Bázový atom* je atom, jehož parametry jsou bázové termy.

Definice 3.5 (Klauzule klauzulární logiky)

Klauzuli reprezentujeme předpisem

$$\underbrace{\{p_1, p_2, \ldots, p_n\}}_{antecedent(A)} \rightarrow \underbrace{\{q_1, q_2, \ldots, q_m\}}_{konsekvent(K)} \tag{3.5}$$

kde $p_i,\ i \in \{1, 2, \ldots, n\}$ a $q_j,\ j \in \{1, 2, \ldots, m\}$ jsou atomy jazyka klauzulární logiky. Mezi atomy v antecedentu je vztah konjunkce, mezi atomy v konsekventu je vztah disjunkce. Je to přepis zobecněných Hornových klauzulí definovaných na str. 43 vzorcem (3.4).

Protože podle vzorce (3.5) jsou množiny antecedentu a konsekventu jednoznačně odděleny, je zvykem zapisovat klauzule bez množinových závorek, tedy vzorec bude

$$\underbrace{p_1, p_2, \ldots, p_n}_{antecedent(A)} \rightarrow \underbrace{q_1, q_2, \ldots, q_m}_{konsekvent(K)} \tag{3.6}$$

Speciální formy klauzulí:

$$\rightarrow \quad q_1, q_2, \ldots q_m \tag{3.7}$$

$$p_1, p_2, \ldots p_n \quad \rightarrow \tag{3.8}$$

(3.6): obecná forma,

(3.7): $A = \emptyset \ \Rightarrow$ *Fakt*, pravdivost tvrzení v konsekventu není ničím podmíněna,

(3.8): $K = \emptyset \ \Rightarrow$ odpovídá implikaci $A \rightarrow false$; jak později zjistíme, používá se k reprezentaci *negativních* tvrzení, v logickém programování k reprezentaci dotazu (protože vycházíme z nepřímého důkazu, ve kterém se *neguje* závěr a přidává se k množině předpokladů).

3.2.2 Univerzální tvrzení

Pod pojmem univerzální tvrzení rozumíme taková tvrzení, kde by v přepisu do predikátové logiky všechny proměnné byly vázány univerzálně, a tedy nevyskytují se zde žádné existenční konstanty ani existenční funktory. Přepis z predikátové do klauzulární logiky je ukázán na následujících příkladech.

Příklad 3.1

V přirozeném jazyce:	V létě mají všichni školáci prázdniny.
V predikátové logice:	$\forall x\,(rocni\_obdobi(leto)\ \&\ skolak(x)) \rightarrow ma(x, prazdniny))$
V klauzulární logice:	$rocni\_obdobi(leto),\ skolak(X)) \rightarrow ma(X, prazdniny)$

Příklad 3.2

Převedeme z přirozeného jazyka do klauzulární logiky následující věty:

1. Psi štěkají (všichni psi štěkají; každý pes štěká; všechno, co je pes, štěká).
   $pes(X) \rightarrow steka(X)$
2. V létě má listí zelenou barvu, zatímco na podzim má listí žlutou nebo červenou barvu.
   $rocni\_obdobi(leto),\ listi(X) \rightarrow barva(X, zelena)$
   $rocni\_obdobi(podzim),\ listi(X) \rightarrow barva(X, zluta),\ barva(X, cervena)$
3. Stůl je tvrdý.
   $stul(X) \rightarrow tvrdy(X)$ ..... „(všechny) stoly jsou tvrdé“
   nebo
   $\rightarrow tvrdy(stul)$ ..... „(konkrétní) stůl je tvrdý“, jde o takto nazvaný objekt
4. Když je hezké počasí, děti jdou na procházku.
   $pocasi(hezky),\ dite(X) \rightarrow jde(X, prochazka)$
5. Píšu písemku, takže musím do školy.
   $pise(ja, pisemka) \rightarrow musi\_do(ja, skola)$
   nebo
   $ja(X),\ pise(X, pisemka) \rightarrow musi\_do(X, skola)$

Na příkladech je vidět, že obvykle stačí „předsunout“ kvantifikátory do prefixu formule, formuli převést do tvaru, kdy hlavní spojkou je implikace, nalevo od ní jsou konjunkce a napravo disjunkce, a pak konjunkce a disjunkce nahradit čárkami. Obecný postup známe z předchozího semestru, používali jsme ho při skolemizaci.

Pokud se nedaří uspořádat konjunkce a disjunkce tak, aby odpovídaly schématu (3.6) ze strany 44, je třeba formuli rozdělit na více formulí (viz bod 2 předchozího příkladu). Postup si lépe ukážeme v sekcích o sémantice.

Jak postupovat při vytváření formulí:

1. Když vytváříme množinu klauzulí, ze které budeme odvozovat, měli bychom předem stanovit konstanty a predikáty včetně typu parametrů. Odvozovací pravidla často pracují se

dvěma různými klauzulemi, tedy dbáme na to, aby v klauzulích byly pro stejný typ informace používány stejné predikáty (například pro vyjádření, že určitý objekt má konkrétní barvu, vytvoříme jediný predikát).

2. Jaké predikáty již jsou nadefinovány, těch se držíme. Například v bodu 3 příkladu na straně 45 jsou dvě možnosti, jak predikáty zvolit. Jestliže jsme v předchozích klauzulích již použili predikát $stul/1$, zvolíme první možnost a nebudeme řetězec $stul$ používat pro konstantu. Je to důležité především proto, abychom později mohli z těchto klauzulí odvozovat.
3. Pokud definujeme nové predikáty, zachováváme určitou syntaxi a názvy predikátů necháváme v jednotném čísle, první osobě, čase přítomném (ale například pokud se celý popisovaný děj odehrává v minulosti, můžeme u všech použít minulý čas – rozhodně by měl být jednotný pro tentýž predikát):
   - když vyjadřujeme vlastnost nějakého objektu, název predikátu bude právě tato vlastnost, první parametr bude objekt, který vlastnost má, případné další parametry tuto vlastnost upřesňují, například
     $barva(\langle$čeho$\rangle, \langle$jaká$\rangle)$
     $dite(\langle$kdo$\rangle)$ nebo $je_dite(\langle$kdo$\rangle)$
     $dospely(\langle$kdo$\rangle)$
     $vysoky(\langle$kdo$\rangle, \langle$výška$\rangle)$
     $adresa(\langle$kdo$\rangle, \langle$ulice$\rangle, \langle$číslo$\rangle, \langle$město$\rangle, \langle$psč$\rangle)$
     $zamestnani(\langle$osoba$\rangle, \langle$název zaměstnání$\rangle)$
   - fakt, že se něco děje, někdo provádí určitou činnost apod., vyjádříme názvem činnosti (často slovesem), první parametr (jeden nebo více) pak bývá subjekt nebo objekt, kterého se činnost týká, další parametry opět činnost upřesňují:
     $jde(\langle$kdo$\rangle, \langle$kam$\rangle)$
     $ma(\langle$kdo$\rangle, \langle$co$\rangle)$
     $pohybuje_se(\langle$kdo$\rangle, \langle$jak$\rangle)$
     $nese(\langle$kdo$\rangle, \langle$co$\rangle)$
     $boji_se(\langle$kdo$\rangle, \langle$koho$\rangle)$
4. Predikáty (a také případné funktory) navrhujeme tak, aby byly co nejvíc vypovídající a aby měly spíše méně parametrů (většinou si vystačíme se dvěma nebo třemi parametry).
5. Když se rozhodujeme, zda určitý název použijeme pro konstantu nebo predikát, bereme v úvahu, jestli takto označený objekt nebo subjekt má nějaké další vlastnosti, které by mohly být využity při odvozování. Například u bodu 3 v příkladu zvolíme konstantu $stul$, pokud máme jen jeden stůl nebo u stolu nemíníme zkoumat jeho tvrdost, výsku apod., kdežto $stul$ bude predikát, jestliže je jeden stůl ve vlastnictví jedné osoby, další ve vlastnictví jiné osoby či v jiné místnosti, každý ze stolů má jinou výšku, ubrus, je z jiného materiálu apod.

   Vodítkem mohou být například slova „každý“, „některý“, „všechny“, nebo uvedení podstatných jmen v množném čísle, které obvykle znamenají, že bychom daný objekt nebo subjekt měli popsat spíše predikátem (například „Stůl je tvrdý.“ bude spíše s konstantou, kdežto „Stoly jsou tvrdé.“ znamená použití predikátu $stul/1$).

Na tom, jak navrhneme konstanty a predikáty, závisí, zda a jak jednoduše bude možné odvozovat další klauzule.

Úkoly

1. Uvedené klauzule převeďte do češtiny (slovenštiny):
   - $jidlo(X) \rightarrow chut(X, slany),\ chut(X, sladky),\ chut(X, kysely),$ $chut(X, horky)$
   - $\rightarrow chut(mrkev, sladky)$
   - $\rightarrow chut(okurka, kysely)$
   - $jidlo(X) \rightarrow je\_kde(X, talir),\ je\_kde(X, hrnec)$
   - $jidlo(X),\ je\_kde(X, talir) \rightarrow vola(matka, dite)$
   - $clovek(X),\ vek(X, Y),\ Y >= 18 \rightarrow dospely(X)$
   - $dite(X),\ jidlo(Y),\ chut(Y, sladky) \rightarrow ma\_rad(X, Y)$
2. Následující věty převeďte na klauzule klauzulární logiky, pokuste se volit konstanty a predikáty tak, aby bylo možné s klauzulemi dále pracovat (odvozovat). Nejdřív stanovte názvy a parametry predikátů.
   - Psi mají 4 nohy.
   - Azor je pes. Pánem Azora je Honza. *(Pozor, konstanty píšeme malým počátečním písmenem, i když jde o jména!)*
   - Azor vodí svého pána na procházku do parku nebo k řece.
   - Na procházce má Azor vodítko.
   - Jestliže má Azor hlad a je doma, škrábe na ledničku.
   - Když Azor škrábe na ledničku a Honza je doma, dá Honza Azorovi žrádlo.
   - Pokud má Azor hlad a je na procházce v parku, kouše klacek nebo vodítko.

3.2.3 Existenční tvrzení

Existenční tvrzení jsou tvrzení obsahující existenční termy, tedy existenční konstanty nebo existenční funktory. Ve formě formule predikátové logiky jsou tyto termy vázány existenčním kvantifikátorem. Tento typ tvrzení používáme, když se jedná o určitý počet objektů/subjektů, ale ne nutně o všechny (nebo nevíme, zda se tvrzení týká všech).

Příklad 3.3

Následující klauzule přepíšeme do češtiny:

$$\rightarrow\ vlastni(jana, @neco) \tag{3.9}$$

$$\rightarrow\ jde\_kudy(@nekdo, les) \tag{3.10}$$

$$\rightarrow\ bavi(@nekdo, logika) \tag{3.11}$$

$$modry(@c) \rightarrow\ vidi(ja, @c) \tag{3.12}$$

(3.9): Jana něco vlastní (existuje něco, co vlastní Jana).
(3.10): Někdo jde lesem.
(3.11): Existuje někdo, koho baví logika (někoho baví logika).
(3.12): Vidím něco modrého.

Příklad 3.4

Převedeme do jazyka klauzulární logiky větu „Každý zaměstnanec má svého (nějakého) nadřízeného“.

Když se pokusíme větu převést jako univerzální tvrzení, dostaneme klauzuli
$zamestnanec(X) \rightarrow nadrizeny(Y, X)$,
což ale znamená „Každý zaměstnanec má všechny jako své nadřízené“, to není dobře. Proto je zřejmé, že nadřízený je „vázán“ existenčně.

Když použijeme existenční konstantu, dostaneme klauzuli
$zamestnanec(X) \rightarrow nadrizeny(@c, X)$,
což znamená „Existuje nadřízený všech zaměstnanců“ neboli „Existuje někdo, kdo je nadřízený každého zaměstnance“, což opět není správně. Potřebujeme, aby se nadřízený vztahoval k určitému zaměstnanci.

Správně je klauzule s existenčním funktorem:
$zamestnanec(X) \rightarrow nadrizeny(@f(X), X)$
Denotátem funktoru $@f(X)$ bude některá funkce, která přiřazuje vždy konkrétnímu zaměstnanci konkrétního nadřízeného, to, že je existenční, znamená, že existuje alespoň jedna taková funkce (bude určena procesem denotace).

Poznámka:

Jak poznat, kdy použít existenční konstantu a kdy existenční funktor? Pomůckou může být pozice existenčního kvantifikátoru v ekvivalentní formuli predikátové logiky. V našem příkladě je proměnná X vázána univerzálním kvantifikátorem (každý zaměstnanec, všichni zaměstnanci), nadřízený tohoto zaměstnance je vázán existenčním kvantifikátorem.

Z toho vyplývá, že je třeba nejdřív přesunout všechny kvantifikátory do prefixu formule, následující postup a rozhodování mezi existenční konstantou a existenčním funktorem vpodstatě odpovídá procesu skolemizace.

Jestliže je v predikátové formuli nejdřív univerzální a pak existenční kvantifikátor (pro každého zaměstnance existuje nadřízený), použijeme existenční funktor, když je nejdřív existenční a pak univerzální (existuje nadřízený všech zaměstnanců), použijeme existenční konstantu.

Příklad 3.5

Převedeme do jazyka klauzulární logiky následující věty:

1. Každé muzeum má nějaký bezpečnostní systém.
   $muzeum(X) \rightarrow bezpecnostni\_system(X, @f(X))$,
   protože v predikátové logice je
   $\forall X \exists F(muzeum(X) \rightarrow bezpecnostni\_system(X, F))$
2. Každý student nosí nějakou čepici.
   $student(X) \rightarrow nosi(X, @cepice(X))$,
   kde $@cepice$ je existenční funktor přiřazující lidem čepice, protože v predikátové logice je
   $\forall X \exists C(student(X) \rightarrow nosi(X, C))$

Úkoly

1. Následující klauzule převeďte do češtiny (slovenštiny):
   (a) $cast\_dne(noc), pritomen(@c, skola) \rightarrow poplach(skola)$
   (b) $poplach(skola), pritomen(X, skola), clovek(X) \rightarrow podezrely(X, vloupani)$
   (c) $\rightarrow poplach(@c)$
   (d) $clovek(X), ma(X, motiv, Y) \rightarrow podezrely(X, Y)$
   (e) $clovek(X), ma(X, prilezitost, Y) \rightarrow podezrely(X, Y)$
   (f) $podezrely(X), trestny\_cin(Y), dukaz(@d(Y), X) \rightarrow vinen(X, Y)$
   (g) $umi(X, Y) \rightarrow podezrely(X, ucil\_se(Y))$
   (h) $clovek(X) \rightarrow podezrely(X, @p(X))$
   (i) $clovek(X) \rightarrow podezrely(X, @p)$
2. Podle následujících vět vytvořte klauzule. Předem navrhněte predikáty a konstanty.
   (a) Čokoláda je sladká.
   (b) Mrkev je sladká.
   (c) Petra má ráda čokoládu.
   (d) Některé sladké věci jsou zdravé.
   (e) Ve skříni jsou pouze zdravé věci (vše, co je ve skříni, je zdravé).
   (f) Ve skříni něco je.
   (g) Každou zdravou věc má někdo rád (tj. když je to zdravé, ...).
   (h) Některé zdravé věci mají rádi všichni.
   (i) Každý, kdo jí něco sladkého, mlsá.
   (j) Když někdo mlsá a špatně si čistí zuby, bolí ho zuby. *(Tady pozor, slovo „někdo“ nemusí nutně znamenat existenční tvrzení!)*
3. Zamyslete se nad pátou větou z předchozího úkolu („Ve skříni jsou ...“). Připouští tato věta situaci, ve které by skříň byla prázdná?

3.3 Sémantika jazyka klauzulární logiky

Sémantika jazyka klauzulární logiky vychází ze sémantiky v predikátové logice. Struktura pro interpretaci je definována naprosto stejně, určité rozdíly jsou definici denotace a valuace především z důvodu odlišností v reprezentaci existenčních tvrzení.

Definice 3.6 (Struktura pro interpretaci)
V klauzulární logice je sémantika dána strukturou $\mathcal{S} = (W, \mathcal{F}, \mathcal{R})$, kde jsou $\mathcal{F} = \{F_1, F_2, \ldots, F_u\}$,

$\mathcal{R} = \{R_1, R_2, \ldots, R_v\}$. Prvek W nazýváme *univerzum diskurzu*, $\mathcal{F}$ je množina funkcí, $\mathcal{R}$ je množina relací.

Struktura je aplikovatelná na množinu klauzulí, jestliže všechny prvky, které se vyskytují v této množině klauzulí, lze interpretovat některým z prvků struktury (tedy individuovým konstantám můžeme přiřadit některý prvek univerza diskurzu, funktorům funkci, predikátům relaci).

Definice 3.7 (Denotace)

Označme $\mathcal{F}_n$ množinu všech n-árních funkcí, tedy funkcí arity n, $\mathcal{F}_n \subseteq \mathcal{F}$. Denotační zobrazení D je definováno následovně:

- $D(c) = a \in W$, kde c je některá individuová konstanta, každé individuové konstantě je přiřazen některý prvek univerza diskurzu,
- $D(@c) = W$, existenční konstantě přiřazujeme celé univerzum diskurzu, konkrétní prvek univerza bude vybrán až během interpretace,
- $D(f_k) = F_k,\ 1 \leq k \leq u$, každému funktoru přiřadíme funkci z množiny $\mathcal{F}$,
- $D(@f/n) = \mathcal{F}_n$, existenčnímu funktoru přiřadíme množinu všech funkcí dané arity, konkrétní funkce bude určena až během interpretace,
- $D(p_k) = R_k$, každému predikátu přiřadíme některou relaci z množiny $\mathcal{R}$.

Definice 3.8 (Valuace proměnné a termu)

Ohodnocení (valuace) proměnné X je zobrazení e, které každé proměnné přiřadí prvek z univerza diskurzu, tedy pro každou proměnnou X je $e(X) \in W$.

Pokud ohodnotíme všechny proměnné nacházející se v některém termu t, potom se tento term stane *bázovým termem*.

Pokud toto zobrazení přiřazuje svým argumentům pouze prvky univerza diskurzu struktury $\mathcal{S}$, mluvíme o *valuaci aplikovatelné na strukturu* $\mathcal{S}$.

Ohodnocení (valuace) termu t je zobrazení e' definované následovně:

- $e'(c) = D(c)$, c je individuová konstanta,
- $e'(X) = e(X)$, X je proměnná,
- $e'(f(t_1, t_2, \ldots, t_n)) = F(e'(t_1), e'(t_2), \ldots, e'(t_n))$, kde f je funktor, jeho denotát je $D(f) = F$, $t_1, t_2, \ldots, t_n$ jsou termy.

Definice 3.9 (Interpretace)

Interpretace atomu je zobrazení I, které v dané struktuře $\mathcal{S}$ a pro danou valuaci e přiřadí každému atomu hodnotu $true$ (t, 1) nebo $false$ (f, 0) takto:

- logické konstantě je vždy přiřazena hodnota jejího denotátu,

- n-árnímu atomu p/n, v jehož argumentech se nevyskytují žádné exitenční termy (ve tvaru $p(t_1, t_2, \ldots, t_n)$), je přiřazena hodnota $true$, pokud pro n-tici vzniklou ohodnocením e' termů v jeho argumentech platí $(e'(t_1), e'(t_2), \ldots, e'(t_n)) \in R$, kde $R = D(p)$ (tj. relace R je denotátem predikátu p), jinak je přiřazena hodnota $false$,
- n-árnímu atomu p/n, v jehož argumentech se nachází existenční term $@t$ (atom ve tvaru $p(t_1, \ldots, @t, \ldots, t_n)$), je přiřazena hodnota $true$, pokud se v denotátu existenčního termu $D(@t)$ nachází prvek t' (funkce v případě existenčního funktoru nebo prvek univerza v případě existenční konstanty), po jehož dosazení do argumentů predikátu místo existenčního termu platí
  - $(e'(t_1), \ldots, e'(t'), \ldots, e'(t_n)) \in R$, pokud $@t$ je existenční funktor,
  - $(e'(t_1), \ldots, t', \ldots, e'(t_n)) \in R$, pokud $@t$ je existenční konstanta,

  kde $R = D(p)$, jinak je přiřazena hodnota $false$.

Atom je pravdivý ve struktuře $\mathcal{S}$ při ohodnocení e, jestliže v této struktuře a ohodnocení interpretován hodnotou $true$. Zapisujeme $I(p)[\mathcal{S}, e] = true$. Fakt, že atom p je interpretován hodnotou $true$ ve struktuře $\mathcal{S}$ (při jakémkoliv ohodnocení), zapisujeme $I(p)[\mathcal{S}] = true$.

Dále si definujeme pravdivou a platnou klauzuli. Je to vlastně už jen formalita – tyto termíny jsou obdobné těm, které už známe z predikátové logiky. Podobně by to bylo například i s pojmem kontradikce.

Definice 3.10 (Pravdivost klauzule)

Klauzule je nepravdivá ve struktuře $\mathcal{S}$ při ohodnocení e, jestliže v tomto ohodnocení je její antecedent pravdivý a konsekvent nepravdivý, tedy když jsou všechny atomy antecedentu interpretovány jako $true$ a všechny atomy konsekventu jako $false$.

V opačném případě je *klauzule pravdivá ve struktuře a daném ohodnocení*.

A	K	A → K
0	0	1
0	1	1
1	0	0
1	1	1

Poznámka:

Tato definice je pouhým přepisem pravdivosti formule v predikátové logice. To zjistíme, když si uvědomíme způsob interpretace formule s logickou spojkou implikace, konjunkce nebo disjunkce. Klauzule je pravdivá v dané struktuře a ohodnocení, pokud je alespoň jeden atom v antecedentu nepravdivý nebo alespoň jeden atom v konsekventu pravdivý.

Definice 3.11 (Platnost klauzule)

Klauzule je platná (splněna) ve struktuře $\mathcal{S}$, když je pravdivá pro jakoukoliv valuaci aplikovatelnou ve struktuře $\mathcal{S}$.

Jestliže klauzule není pravdivá v žádné valuaci aplikovatelné v dané struktuře, pak je *nesplnitelná* (není platná) v dané struktuře.

Klauzule je logicky platná (logický zákon), jestliže je platná v jakékoliv struktuře.

Sémantika vymezuje svět, ve kterém s klauzulemi pracujeme. Určuje konstanty, které lze dosadit za proměnné, Stanoví funkce a relace, které dávají konkrétní význam funktorům a predikátům. Když odvozujeme nové tvrzení, toto tvrzení je zasazeno do světa určeného kromě jiného také sémantikou, proto nás především zajímají klauzule platné v dané struktuře, obvykle nevyžadujeme, aby byly logicky platné.

Příklad 3.6

Chceme interpretovat klauzuli $C = p(X), q(X,a) \rightarrow r(X,f(b)), r(X,c)$. Pro interpretaci použijeme strukturu $\mathcal{S}_1$.

$\mathcal{S}_1 = (W_1, \mathcal{F}_1, \mathcal{R}_1)$, kde

$W_1 = \{listi, zluta, hneda, zelena, modra, kuratko, jasan, dub, buk\}$, $\mathcal{F}_1 = \{barva/1\}$,

$\mathcal{R}_1 = \{strom/1, ma/2, barva\_listi/2\}$,

Funkce $barva/1$:

$barva(kuratko) = zluta,\ barva(nebe) = modra,\ barva(zeme) = hneda$, atd. pro další prvky univerza,

Relace:

$strom/1 = \{(jasan), (dub), (buk)\}$,

$ma/2 = \{(jasan, listi), (buk, listi)\}$,

$barva\_listi = \{(jasan, zluta), (dub, hneda), (buk, zelena)\}$

Denotace:

$D(a) = listi,\ D(b) = kuratko,\ D(c) = modra,$

$D(f) = barva,$

$D(p) = strom,\ D(q) = ma,\ D(r) = barva\_listi$

Uplatníme denotační zobrazení D na atomy v klauzuli:

$strom(X), ma(X, listi) \rightarrow barva\_listi(X, barva(kuratko)), barva\_listi(X, modra)$

Funkce $barva$ má konstantní argument, proto tuto funkci můžeme vyhodnotit:

$strom(X), ma(X, listi) \rightarrow barva\_listi(X, zluta), barva\_listi(X, modra)$

Interpretujeme se zvolenou valuací $e_1(X) = jasan$:

$I(strom(jasan))[\mathcal{S}_1, e_1] = true$ $\quad$ $I(barva\_listi(jasan, zluta))[\mathcal{S}_1, e_1] = true$

$I(ma(jasan, listi))[\mathcal{S}_1, e_1] = true$ $\quad$ $I(barva\_listi(jasan, modra))[\mathcal{S}_1, e_1] = false$

Interpretace celé klauzule ve zvolené struktuře a ohodnocení:

$I(C)[\mathcal{S}_1, e_1] = I(true, true \rightarrow true, false) = true$

Pro valuaci $e_2(X) = buk$ je $I(C)[\mathcal{S}_1, e_2] = false$.

Příklad 3.7

Při interpretaci téže klauzule nyní použijeme strukturu $\mathcal{S}_2$:

$\mathcal{S}_2 = (W_2, \mathcal{F}_2, \mathcal{R}_2)$, kde

$W_2 = \{skola, index, sesit, student, kladivko, jana, pepa, karel, 0\}$, $\mathcal{F}_2 = \{prukaz/1\}$,

$\mathcal{R}_2 = \{je\_student/1, jde\_do/2, ma/2\}$,

Funkce $prukaz/1$:
$prukaz(student) = index,\ prukaz(karel) = index,\ prukaz(kladivko) = 0$, atd. pro další prvky univerza,

Relace:
$je\_student/1 = \{(pepa), (jana)\}$,
$jde\_do/2 = \{(pepa, skola), (jana, kino), (karel, skola)\}$,
$ma/2 = \{(pepa, sesit), (jana, index), (karel, kladivko)\}$

Denotace:
$D(a) = skola,\ D(b) = student,\ D(c) = sesit,$
$D(f) = prukaz,$
$D(p) = je\_student,\ D(q) = jde\_do,\ D(r) = ma$

Uplatníme denotační zobrazení D na atomy v klauzuli:
$je\_student(X), jde\_do(X, skola) \rightarrow ma(X, prukaz(student)), ma(X, sesit)$

Po vyhodnocení funkce $prukaz$:
$je\_student(X), jde\_do(X, skola) \rightarrow ma(X, index), ma(X, sesit)$

Interpretujeme se zvolenou valuací $e_3(X) = pepa$:
$I(je\_student(pepa))[\mathcal{S}_2, e_3] = true$ $\quad$ $I(ma(pepa, index))[\mathcal{S}_2, e_3] = false$
$I(jde\_do(pepa, skola))[\mathcal{S}_2, e_3] = true$ $\quad$ $I(ma(pepa, sesit))[\mathcal{S}_2, e_3] = true$

Interpretace celé klauzule ve zvolené struktuře a ohodnocení:
$I(C)[\mathcal{S}_2, e_3] = I(true, true \rightarrow false, true) = true$

Na rozdíl od struktury $\mathcal{S}_1$, ve struktuře $\mathcal{S}_2$ je formule C interpretována vždy jako $true$, proto můžeme psát $I(C)[\mathcal{S}_2] = true$.

Příklad 3.8

Vezměme klauzuli $\mathcal{C} = p(@c), p(X) \rightarrow q(a(X), @c)$. Vytvoříme dvě struktury pro interpretaci:

- $\mathcal{S}_1 = (\mathcal{R}, \{naslednik/1\}, \{prirozene\_cislo/1, vetsi\_nez/2\})$
- $\mathcal{S}_2 = (\mathcal{R}, \{odmocnina/1\}, \{prirozene\_cislo/1, rovna\_se/2\})$

V obou případech je univerzum tvořeno množinou všech reálných čísel. Funkce:

- funkce $naslednik/1$ vrací číslo o 1 větší než je jeho argument,
- funkce $odmocnina/1$ vrací druhou odmocninu svého argumentu.

Relace jsou definovány tak, jak je známe z matematiky:

- relace $prirozene\_cislo/1$ vrací $true$, pokud je její argument přirozené číslo,
- relace $vetsi\_nez/2$ vrací $true$, jestliže je její první argument větší než druhý,
- relace $rovna\_se/2$ vrací $true$, pokud se oba její argumenty rovnají.

Nejdřív klauzuli interpretujeme v první struktuře. Denotace pro strukturu $\mathcal{S}_1$:
$D_1(a) = naslednik$
$D_1(p) = prirozene\_cislo$, $D_1(q) = vetsi\_nez$
$D_1(@c) = \mathcal{R}$ (celé univerzum)

Uplatníme denotační zobrazení D_1 na atomy v klauzuli (kromě existenční konstanty):
$prirozene\_cislo(@c), prirozene\_cislo(X) \rightarrow vetsi\_nez(naslednik(X), @c)$

Je zřejmé, že při interpretaci stačí v denotaci existenční konstanty zvolit z univerza diskurzu prvek 1, a klauzule bude při jakémkoliv ohodnocení proměnné X interpretována hodnotou $true$. Prvek 1 je přirozené číslo, atom $prirozene\_cislo(@c)$ bude interpretován hodnotou $true$.

- Jestliže za X dosadíme přirozené číslo (tj. celé číslo větší než 0), jeho následník (číslo o 1 vyšší) bude vždy větší než 1. Proto antecedent i konsekvent jsou interpretovány hodnotou $true$, klauzule je interpretována hodnotou $true$.
- Pokud za X dosadíme číslo, které není přirozené (0 nebo záporné číslo), je antecedent vyhodnocen jako $false$ a klauzule je interpretována hodnotou $true$.

To znamená, že ve struktuře $\mathcal{S}_1$ interpretujeme klauzuli $\mathcal{C}$ takto:
$I(\mathcal{C})[\mathcal{S}_1] = true$, klauzule je platná (splněna) ve struktuře $\mathcal{S}_1$.

Zaměřme se nyní na strukturu $\mathcal{S}_2$. Denotační zobrazení bude následující:
$D_2(a) = odmocnina$
$D_2(p) = prirozene\_cislo$, $D_2(q) = rovna\_se$
$D_2(@c) = \mathcal{R}$ (celé univerzum)

Uplatníme denotační zobrazení D_2 na klauzuli $\mathcal{C}$:
$prirozene\_cislo(@c), prirozene\_cislo(X) \rightarrow rovna\_se(odmocnina(X), @c)$

Vezměme nyní dvě různá ohodnocení: $e_1(X) = 1$, $e_2(X) = 2$. Nejdřív zvolíme za $@c$ některý prvek univerza diskurzu, který je přirozeným číslem, a pak zjistíme, jak je klauzule v těchto ohodnoceních interpretována. Pokud je například za $@c$ dosazen prvek 1, platí
$I(true, true \rightarrow true) = true$, při ohodnocení e_1
$I(true, true \rightarrow false) = false$, při ohodnocení e_2

Pokud však za $@c$ dosadíme číslo, které není přirozené (třeba -8), vypadá interpretace jinak. Antecedent je pro všechna ohodnocení vyhodnocen jako $false$, a tedy
$I(\mathcal{C})[\mathcal{S}_2] = I(false \rightarrow \ldots) = true$

Z toho vyplývá, že klauzule $\mathcal{C}$ je platná také ve struktuře $\mathcal{S}_2$. Všimněte si, že kdybychom z antecedentu odstranili atom $p(@c)$, výsledek by byl jiný.

Jak vidíme, při interpretaci klauzule, která obsahuje existenční konstanty, dosazujeme za existenční konstantu některé individuum z univerza diskurzu tak, aby byla celá klauzule při zvoleném ohodnocení interpretována hodnotou $true$ (pokud je to ovšem možné). Podobně postupujeme při interpretaci klauzule, která obsahuje existenční funktory – volíme z množiny funkcí arity shodné s aritou existenčního funktoru.

Úkoly

1. Projděte si příklad na straně 53 a zjistěte, zda existuje struktura, ve které by daná klauzule nebyla platná.
2. Je dána tato klauzule: $\mathcal{C} = \rightarrow q(a(X, 2), @f(X))$ (antecedent je prázdný). Vezměme strukturu $\mathcal{S} = (\mathcal{N}, \{deleni/2, dvojnasobek/1\}, \{rovna\_se/2\})$. Univerzum diskurzu je množina

přirozených čísel. První z funkcí struktury, $deleni/2$, svůj první argument *celočíselně* vydělí druhým argumentem, druhá funkce svůj jediný argument vynásobí dvěma. Relace $rovna\_se/2$ vrací hodnotu $true$, pokud se její argumenty rovnají.

Stanovte vhodně denotační zobrazení a zjistěte, zda je klauzule C v dané struktuře pravdivá či dokonce platná.

3.4 Vlastnosti klauzulí

3.4.1 Prázdná množina antecedentu nebo konsekventu

Antecedent i konsekvent jsou množiny, a množiny mohou být také prázdné. Klauzuli pak interpretujeme následovně:

1. $\{\} \rightarrow \{q_1, q_2, \ldots, q_m\}$ (antecedent je prázdná množina): protože mezi atomy v antecedentu je vztah konjunkce, pravdivostní hodnota klauzule se nezmění, když do antecedentu přidáme atom $true$. V případě prázdné množiny antecedentu proto interpretujeme formuli $\{true\} \rightarrow \{q_1, q_2, \ldots, q_m\}$.

   Klauzuli s prázdnou množinou antecedentu nazýváme *fakt*. Pokud je klauzule v dané struktuře a ohodnocení považována za pravdivou, je pravdivá i disjunkce atomů v konsekventu.

2. $\{p_1, p_2, \ldots, p_n\} \rightarrow \{\}$ (konsekvent je prázdná množina): protože mezi atomy v konsekventu je vztah disjunkce, pravdivostní hodnota klauzule se nezmění, když do konsekventu přidáme atom $false$. V případě prázdné množiny konsekventu proto interpretujeme formuli $\{p_1, p_2, \ldots, p_n\} \rightarrow \{false\}$.

   Jestliže je klauzule v dané struktuře a ohodnocení považována za pravdivou, je konjunkce atomů v antecedentu nepravdivá, tedy alespoň jeden atom je interpretován jako $false$. Toho se využívá k reprezentaci negace atomů.

3. $\{\} \rightarrow \{\}$ (antecedent i konsekvent jsou prázdná množina, nazýváme *prázdná klauzule*): interpretujeme klauzuli $\{true\} \rightarrow \{false\}$, která podle definice implikace je vždy interpretována jako $false$, tedy jde o kontradiktorickou (nesplnitelnou) klauzuli. Toho využíváme při důkazu sporem (dokazovaný závěr znegujeme a odvozováním se snažíme získat prázdnou klauzuli).

Příklad 3.9

Vytvoříme několik klauzulí s prázdnou množinou antecedentu nebo konsekventu:

- $\rightarrow kulaty(zeme)$
  (Země je kulatá.)
- $\rightarrow pocet\_nohou(clovek, 2), pocet\_nohou(clovek, 4)$
  (Člověk má dvě nebo čtyři nohy.)
- $barva(pisek, fialova) \rightarrow$
  (Písek není fialový.)

- $vrah(zahradnik), vrah(domovnik) \rightarrow$
  (Buď zahradník není vrah nebo domovník není vrah (nebo žádný z nich není vrah).)

3.4.2 Konjunkce a disjunkce atomů v klauzuli

Vycházíme z obecného tvaru klauzule $p_1, p_2, \ldots, p_n \rightarrow q_1, q_2, \ldots, q_m$, která je do predikátové logiky překládána na formuli $p_1 \,\&\, p_2 \,\&\, \ldots \,\&\, p_n \rightarrow q_1 \vee q_2 \vee \ldots \vee q_m$ s univerzálním vázáním proměnných.

Konjunkce v antecedentu je tam, kde má být, takže ji není třeba řešit. Atomy spojené konjunkcemi prostě oddělíme čárkou.

Příklad 3.10

Když to má pruhy a kopyta, je to zebra.
$ma(X, pruhy), ma(X, kopyta) \rightarrow zebra(X)$

Disjunkci v antecedentu, tedy formuli $F_1 \vee F_2 \rightarrow K$, řešíme podle věty

Věta 3.1 (Disjunkce v antecedentu)

$$(F_1 \vee F_2 \rightarrow K) \iff ((F_1 \rightarrow K) \,\&\, (F_2 \rightarrow K)) \tag{3.13}$$

Důkaz: lze provést například sémantickým tablem, je triviální. □

Poznámka:

V množině klauzulí je mezi klauzulemi stav konjunkce, proto je tento přepis použitelný. Znamená to, že vytvoříme dvě klauzule, které budou mít stejný konsekvent, a antecedent se do klauzulí rozdělí v místě disjunkce.

Atomů spojených disjunkcí může být v antecedentu jakýkoliv počet, pak samozřejmě vytvoříme tolik klauzulí, kolik je atomů antecedentu spojených disjunkcemi.

Příklad 3.11

Kdo nosí index nebo skripta, je student.
$nosi(X, index) \rightarrow student(X)$
$nosi(X, skripta) \rightarrow student(X)$

Konjunkce v konsekventu nemá co dělat, proto formuli $A \rightarrow F_1 \,\&\, F_2$ řešíme podle věty

Věta 3.2 (Konjunkce v konsekventu)

$$(A \rightarrow F_1 \,\&\, F_2) \iff ((A \rightarrow F_1) \,\&\, (A \rightarrow F_2)) \tag{3.14}$$

Důkaz: lze opět provést například sémantickým tablem. □

Stejně jako v případě disjunkce v antecedentu, i zde vytvoříme tolik klauzulí, kolik je atomů v konsekventu spojených konjunkcí, všechny klauzule budou mít stejný antecedent.

Příklad 3.12

Kočky mají ostré zuby, ostré drápy a dobrý zrak.

$kocka(X) \rightarrow ma(X, ostre\_zuby)$
$kocka(X) \rightarrow ma(X, ostre\_drapy)$
$kocka(X) \rightarrow vidi(X, dobre)$

Disjunkce v konsekventu je tam, kde má být, proto ji pouze nahradíme čárkami:

Příklad 3.13

Kdo je v nemocnici, je buď nemocný nebo přišel někoho navštívit.

$nemocnice(N), je\_kde(X, N) \rightarrow nemocny(X), navstivil(X, @f(X))$

Úkol

Následující věty převeďte do jazyka klauzulární logiky.

1. Všechno sladké obsahuje cukr nebo umělé sladidlo.
2. Mrkev je sladká a zdravá, rajská jablka jsou zdravá, okurka je kyselá.
3. Mrkev i čokoláda jsou sladké a dobré. *(Pozor, tato věta je složená – říká nám více faktů, z nichž první je, že mrkev je sladká, další, že čokoláda je sladká, atd.)*
4. Myši i lidé mají rádi sýr.
5. Jerry je myš a Pepa je člověk.
6. Všechno, co je sladké nebo zdravé, jedí lidé.
7. Vše, co jedí lidé, je sladké nebo zdravé.
8. Některé kyselé věci jsou dobré a zdravé.

Aby z vět číslo 2 a 8 bylo možné odvodit, že okurka je dobrá a zdravá, měli bychom si uvědomit, že větu číslo 2 nutně musíme rozdělit do více klauzulí. Mezi jednotlivými klauzulemi je vždy vztah konjunkce, proto také věta 2 bude přepsána na konjunkci nejméně tří (že by čtyř?) klauzulí.

3.4.3 Negace atomů

Jestliže chceme v klauzuli použít negativní literál pro *bázový atom*, nemůžeme použít atom s negací (protože symbol negace nepatří do syntaxe jazyka klauzulární logiky), ale použijeme jednu z následujících formulí:

Věta 3.3 (Negace bázového atomu)

$$p \,\&\, \neg q \to r \iff p \to q \vee r \tag{3.15}$$

$$p \to \neg q \vee r \iff p \,\&\, q \to r \tag{3.16}$$

Důkaz: Logická platnost těchto formulí je snadno dokazatelná například Quinovým algoritmem nebo sémantickým tablem.

□

To znamená, že v případě *bázových atomů* (bez existenčních termů a proměnných!!!) stačí odstranit negaci a převést atom na opačnou stranu implikace.

Příklad 3.14

Převeďte do jazyka klauzulární logiky následující věty:

1. Když nefouká vítr, drak spadne.
   $\to pocasi(vitr), pozice(drak, pada)$ podle (3.15)
2. V neděli bratr nejde do školy.
   $den\_v\_tydnu(nedele), jde(bratr, skola) \to$ podle (3.16)
3. Když na mě zaútočí medvěd a nemám zbraň, neutíkám.
   $utok(medved, ja), utika(ja) \to ma(ja, zbran)$ podle (3.15) a (3.16)

V případě *atomů s proměnnými a existenčními konstantami* vycházíme opět z predikátové logiky, především z DeMorganových zákonů:

$$\exists c(\neg A(c)) \iff \neg(\forall Y\, A(Y)) \tag{3.17}$$

$$\forall X(\neg A(X)) \iff \neg(\exists m A(m)) \tag{3.18}$$

$$\forall x(p(x) \to \neg(\exists u\; q(x,u))) \iff \forall x \forall u(p(x) \to \neg q(x,u)) \tag{3.19}$$

Tedy podle vztahu (3.17) negujeme atom s existenční konstantou tak, že existenční konstantu nahradíme *novou* proměnnou (která se v klauzuli nevyskytuje) a pak stejně jako bázový atom převedeme na druhou stranu implikace.

Podle (3.18) negujeme atom s proměnnou tak, že tuto proměnnou nahradíme *novou* existenční konstantou (která se v klauzuli nevyskytuje), případně existenčním funktorem, a převedeme na druhou stranu implikace.

Podle příkladu ve vztahu (3.19) řešíme negaci atomu, kde se popření týká existenčního funktoru. Atom převedeme na druhou stranu implikace a existenční funktor nahradíme novou proměnnou.

Poznámka:

Negace atomu, který obsahuje proměnné nebo existenční termy, je věc ošidná. V predikátové logice to můžeme demonstrovat na příkladu, kdy se před predikátem s více proměnnými nachází několik kvantifikátorů (na různých úrovních, v různých vzdálenostech od predikátu, třeba až před celou formulí) a my negaci umístíme někam mezi ně (například mezi druhý a třetí kvantifikátor). Pak se negace projeví pouze na proměnných vázaných kvantifikátory, které jsou až za negací.

Proto když si nejsme jisti, zda máme při negaci zaměnit proměnné a existenční termy, můžeme si pomoci přepisem do predikátové logiky.

Postup:

- sestavíme formuli predikátové logiky,
- negace posuneme co nejblíž atomům (směrem doprava do podformulí), kvantifikátory naopak co nejdál vlevo (do prefixu formule),
- s negací u atomů s proměnnými pak zacházíme stejně jako s negací u bázových atomů, záměna kvantifikátorů už byla vyřešena v předchozích krocích.

Příklad 3.15

Vyjádříme v jazyce klauzulární logiky tyto věty:

1. Někdo nemá jedničku z logiky. (negujeme „Všichni mají jedničku z logiky.“)

   predikátová logika: $\neg(\forall c\ znamka(c, logika, 1)) \iff \exists c\ \neg znamka(c, logika, 1)$
   klauzulární logika: $znamka(@c, logika, 1) \rightarrow$

2. Nikdo nemá jedničku z logiky. (negujeme „Někdo má jedničku z logiky.“)

   predikátová logika: $\neg(\exists x\ znamka(x, logika, 1)) \iff \forall x\ \neg znamka(x, logika, 1)$
   klauzulární logika: $znamka(X, logika, 1) \rightarrow$

3. Nikdo v neděli nechodí do školy.

   predikátová logika: $den\_v\_tydnu(nedele) \rightarrow \neg\exists x\ jde(x, skola)$
   klauzulární logika: $den\_v\_tydnu(nedele), jde(X, skola) \rightarrow$

4. Rostliny, které nejsou byliny ani keře, jsou stromy.

   predikátová logika: $\forall x\ (rostlina(x)\ \&\ \neg bylina(x)\ \&\ \neg ker(x) \rightarrow strom(x))$
   klauzulární logika: $rostlina(X) \rightarrow bylina(X), ker(X), strom(X)$

   Jak vidíme, tato věta je vlastně ekvivalentní i s několika dalšími větami, například „Rostliny mohou být byliny, keře nebo stromy.“.

Poznámka:

Negace vyžaduje (nejen) v logickém programování poněkud specifické zacházení. Například pokud znegujeme univerzálně vázanou proměnnou, pak ji tzv. „uvolníme“, což může mít nečekané důsledky. Později se na tuto problematiku ještě zaměříme.

Úkol

Následující věty převeďte do jazyka klauzulární logiky.

1. Čokoláda není zdravá.
2. To, co je kyselé, není sladké.
3. Některé sladké věci nejsou zdravé.
4. Kdo co nemá rád, to odmítá.
5. Když je něco sladké a zdravé, pak to děti neodmítají.
6. Nic sladkého neexistuje.
7. Honza nemá nic sladkého, ale má okurku.
8. Honza je člověk, ale Micka ne.
9. Když je něco sladké a neobsahuje to cukr, pak to obsahuje umělé sladidlo.
10. Když má (kterýkoliv) člověk nemoc cukrovku, pak je pro něj umělé sladidlo zdravé, ale když člověk nemá cukrovku, pak pro něj umělé sladidlo není zdravé.
11. Některé kočky nemají rády mléko, ale Micka ano.
12. Kdokoliv má rád mléko a myši a nemá rád okurky a mrkev, je kočka.

3.4.4 Negace klauzule

Když chceme popřít klauzuli, použijeme již dříve naznačené postupy pro řešení konjunkcí a disjunkcí atomů a negace atomu. Označme A antecedent, K konsekvent,
$A = p_1 \,\&\, p_2 \,\&\, \ldots \,\&\, p_n$,
$K = q_1 \vee q_2 \vee \ldots \vee q_m$.
Pak platí tato posloupnost ekvivalencí:

$$\neg(A \to K) \iff (A \,\&\, \neg K) \iff \begin{pmatrix} \to & A \\ \to & \neg K \end{pmatrix} \iff \begin{pmatrix} & \to & p_1 \\ & \to & p_2 \\ & \ldots & \\ & \to & p_n \\ q_1 & \to & \\ q_2 & \to & \\ & \ldots & \\ q_m & \to & \end{pmatrix}$$

Znamená to, že všechny atomy negujeme do důsledků (všechny proměnné zaměníme za existenční konstanty atd.) s přehozením na druhou stranu implikace a osamostatníme do jednotlivých klauzulí. Opět může pomoci převod do predikátové logiky s tím, že mezi klauzulemi je vztah konjunkce.

Definice 3.12 (Popírající množina klauzule)

Nechť F je klauzule klauzulární logiky. Pak negací klauzule F je množina klauzulí F' vytvořená výše popsaným způsobem. Množinu klauzulí F' nazýváme popírající množinou klauzule F. Je zřejmé, že mezi klauzulemi v množině F' je vztah konjunkce.

Příklad 3.16

Vyjádříme negace těchto vět (klauzulí):

1. Každé vadné zboží je reklamováno.
   klauzule: $zbozi(X), vadny(X) \rightarrow reklamace(X)$
   negace klauzule: $\rightarrow zbozi(@c)$
   $\rightarrow vadny(@c)$
   $reklamace(@c) \rightarrow$
   negovaná věta: Některé vadné zboží není reklamováno.
2. Některé hračky mají rády všechny děti.
   klauzule: $dite(X), hracka(@h) \rightarrow rad(X, @h)$
   predikátová logika: $\neg(\exists h \forall d(dite(d)\ \&\ hracka(h) \rightarrow rad(d,h)) \Leftrightarrow$
   $\Leftrightarrow \forall h \exists d(dite(d)\ \&\ hracka(h)\ \&\ \neg rad(d,h))$
   negace klauzule: $\rightarrow dite(@f(Y))$
   $\rightarrow hracka(Y)$
   $rad(@f(Y), Y) \rightarrow$
   negovaná věta: Pro každou hračku existuje dítě, které ji nemá rádo.

Negace druhé věty je zároveň ukázkou toho, jak postupujeme při přepisu do klauzulární logiky při negaci formule s posloupností kvantifikátorů odpovídající existenčnímu funktoru.

Úkol

Znegujte následující klauzule (můžete si pomoci predikátovou logikou). Vyjádřete slovně jak původní klauzuli, tak i výslednou klauzuli po negaci.

1. $clovek(X) \rightarrow ma\_rad(X, mrkev), ma\_rad(X, dort)$
2. $kocka(X), zdravy(X) \rightarrow pocet\_nohou(X, 4)$
3. $dite(@c), hodny(@c) \rightarrow$
4. $\rightarrow sviti(@s), tma$ (kde „tma“ je nulární predikát)
5. $mys(X), kocka(Y) \rightarrow boji\_se(X, Y)$
6. $kocka(@k), mys(X) \rightarrow boji\_se(@k, X)$
7. $kocka(X) \rightarrow majitel(X, @m(X))$

3.4.5 Predikát rovnosti a jiné predikáty podle relací

Predikát rovnosti pracuje podobně jako jiné predikáty – jeho atom je interpretován hodnotou $true$ nebo $false$. Jestliže jsou jeho oba argumenty shodné, je výsledek $true$, jinak $false$. Podobné predikáty lze vytvořit i pro jiné relační operátory ($<$, $>$, $\leq$, $\geq$, $\neq$).

Rozdíl oproti jiným predikátům je ve způsobu zápisu. Zatímco predikáty obecně se zapisují prefixově (nejdřív název, pak argumenty v závorce), většina logických programovacích jazyků

umožňuje používat predikáty rovnosti a nerovnosti v infixovém zápisu (tj. nemusíme psát v prefixu „$=(X,Y)$“, ale můžeme napsat „$X = Y$“).

Příklad 3.17

Ukážeme si použití predikátů rovnosti a nerovnosti.

- Psi mají čtyři nohy a dvě uši.

  $pes(X) \rightarrow pocet(X, noha) = 4$
  $pes(X) \rightarrow pocet(X, ucho) = 2$
- Slepic je méně než kachen.

  $\rightarrow mnozstvi(slepice) < mnozstvi(kachna)$
- Čísla větší než nula jsou kladná.

  $cislo(X), X > 0 \rightarrow kladne(X)$
- Listí má na podzim hnědou, červenou nebo žlutou barvu.

  $obdobi(podzim) \rightarrow barva(listi) = hneda, barva(listi) = cervena,$
  $barva(listi) = zluta$

U predikátu rovnosti je třeba dávat pozor především na to, aby byl u funktorů použit jen pro případy testování jednoznačného přiřazení.

Příklad 3.18

Uvedené věty vyjádříme v jazyce klauzulární logiky.

1. Slepic je 25.

   $\rightarrow pocet(slepice, 25)$
   nebo $\rightarrow pocet(slepice) = 25$
2. Barva zralých jahod je červená.

   $jahoda(X), zraly(X) \rightarrow barva(X) = cervena$
   nebo $jahoda(X), zraly(X) \rightarrow barva(X, cervena)$
   nebo $X = jahoda, zraly(X) \rightarrow barva(X) = cervena$
3. Kuchařka potřebuje vařechu.

   $kucharka(X) \rightarrow potrebuje(X, varecha)$
   nebo $X = kucharka \rightarrow potrebuje(X, varecha)$
   špatně: $kucharka(X) \rightarrow potrebuje(X) = varecha$ (protože kuchařka potřebuje i jiné věci než vařechu)
4. Pes má uši.

   $pes(X) \rightarrow ma(X, usi)$
   nebo $X = pes \rightarrow ma(X, usi)$
   špatně: $pes(X) \rightarrow ma(X) = usi$ (pes má taky oči, čumák, ocásek, ...)

V příkladu je většina vět přepsána několika způsoby. Pokaždé jsou použity buď predikáty nebo funktory s různou aritou. Vhodný způsob volíme tak, abychom ve všech klauzulích používali predikáty a funktory stejné arity a se stejným významem argumentů.

Například pokud v případě 2 použijeme název $barva$ pro funktor s jedním argumentem vracející barvu svého argumentu, nemůžeme v jiné klauzuli použít tento název pro predikát.

Úkol

Vyjádřete v klauzulární logice matematický vztah
„Jestliže je nějaké číslo dvojnásobkem jiného celého čísla, pak je sudé.“

V predikátové logice lze tuto větu zapsat jako

$\forall X(\exists N(cele(N)\ \&\ X = 2 \cdot N) \rightarrow sude(X))$

Upravte tuto formuli predikátové logiky (je třeba převést všechny kvantifikátory do prefixu, pozor na ekvivalentní úpravy implikace) a vytvořte ekvivalentní klauzuli klauzulární logiky.

3.5 Substituce

V klauzulích se obvykle nacházejí proměnné a existenční termy. Při používání těchto klauzulí pro účely odvozování důsledků budeme chtít s těmito prvky dále pracovat, a to se dělá pomocí substituce.

Definice 3.13 (Substituce)

Substituce termů $t_1, t_2, \ldots, t_n$ za proměnné $X_1, X_2, \ldots, X_n$ do klauzule $\mathcal{C}$, kde každý term t_i je substituovatelný za proměnnou X_i, $1 \leq i \leq n$, je dána množinou $\varphi = \{t_1/X_1, t_2/X_2, \ldots, t_n/X_n\}$, značíme $\mathcal{C}[\varphi]$.

Substituce je tedy definována prakticky stejně jako v predikátové logice, za proměnnou dosazujeme term, který může být funktor, konstanta, existenční term nebo opět proměnná.

Definice 3.14 (Existenční substituce)

Existenční substituce dosud nepoužitého existenčního termu $@m$ za *bázový term* t se zapisuje $\varphi = (@m/t;\ k_1, k_2, \ldots, k_r)$ a představuje nahrazení $k_1.$, $k_2.$, … a $k_r.$ výskytu termu t existenčním termem $@m$ v dané klauzuli.

Na rozdíl od běžné substituce termů v existenční substituci *nahrazujeme individuové konstanty* a nikoliv proměnné, navíc můžeme substituci omezit pouze na některé výskyty konstanty, nemusíme nutně nahrazovat všechny výskyty konstanty v klauzuli.

Umožňuje nám to fakt, že když je v některém parametru dosazena konstanta, znamená to, že existuje alespoň jedno individuum z univerza diskurzu, které lze dosadit (je to denotát této konstanty). Proto lze tvrzení zobecnit na existenční term. Protože však musíme vyloučit případ-

nou kolizi s jinými existenčními termy, které se v klauzuli již vyskytují (u nich může být vztah existence spojen s úplně jiným prvkem univerza či funkcí), je třeba volit nový, dosud nepoužitý existenční term.

V zápisu existenční substituce máme poněkud nekorektně v posloupnosti první prvek oddělen středníkem, je to z důvodu větší přehlednosti zápisu (středník odděluje předpis pro provedení nahrazení od předpisu pro umístění nahrazení).

Příklad 3.19

U zadaných klauzulí provedeme uvedené substituce.

1. $\mathcal{C}_1 \;=\; p(a,X), q(f(X),b,Z) \to q(f(a),g(b,a),Y)$

   $\varphi_1 = \{a/X, g(a,Z)/Y, Z/Z\}$
   $\varphi_2 = (@c/a;\ 1,3)$

   Provedeme substituce:

   $\mathcal{C}_1[\varphi_1] \;=\; p(a,a), q(f(a),b,Z) \to q(f(a),g(b,a),g(a,Z))$
   $\mathcal{C}_1[\varphi_2] \;=\; p(@c,X), q(f(X),b,Z) \to q(f(a),g(b,@c),Y)$
   $\mathcal{C}_1[\varphi_1][\varphi_2] \;=\; p(@c,a), q(f(@c),b,Z) \to q(f(a),g(b,a),g(a,Z))$

2. $\mathcal{C}_2 \;=\; dum(@f(Y)), vlastni(@f(Y),Y) \to ma(Y,bydliste)$
   (Kdo vlastní nějaký dům, má kde bydlet.)

   $\varphi_3 = \{X/X, pavel/Y\}$
   $\varphi_4 = (@c/pavel;\ 4)$

   Provedeme substituce:

   $\mathcal{C}_2[\varphi_3] \;=\; dum(@f(pavel)), vlastni(@f(pavel),pavel) \to ma(pavel,bydliste)$
   (Jestliže Pavel vlastní nějaký dům, má kde bydlet.)
   $\mathcal{C}_2[\varphi_3][\varphi_4] \;=\; dum(@f(pavel)), vlastni(@f(pavel),pavel) \to ma(@c,bydliste)$
   (Pokud Pavel vlastní nějaký dům, někdo má kde bydlet.)

3. $\mathcal{C}_3 \;=\; clovek(X) \to umi(X,zpivat), hraje\_na(X,@nastroj(X))$
   (Každý člověk umí zpívat nebo hrát na nějaký hudební nástroj.)

   $\varphi_5 = \{ucitel(X,hudebni\_v)/X\}$
   $\varphi_6 = \{honza/X\}$
   $\varphi_7 = (@c/ucitel(honza,hudebni\_v);\ 1\text{–}4)$

   Provedeme substituce:

   $\mathcal{C}_3[\varphi_5] \;=\; clovek(ucitel(X,hudebni\_v)) \to umi(ucitel(X,hudebni\_v),zpivat),$
   $hraje\_na\Big(ucitel(X,hudebni\_v),@nastroj(ucitel(X,hudebni\_v))\Big)$
   (Pro každého člověka, který je něčí učitel v předmětu Hudební výchova, platí, že buď umí zpívat nebo hraje na nějaký hudební nástroj.)

   $\mathcal{C}_3[\varphi_5][\varphi_6] \;=\; clovek(ucitel(honza,hudebni\_v)) \to umi(ucitel(honza,hudebni\_v),zpivat),$
   $hraje\_na(ucitel(honza,hudebni\_v),@nastroj(ucitel(honza,hudebni\_v)))$
   (Každý člověk, kdo je Honzův učitel v předmětu Hudební výchova, buď umí zpívat nebo hraje na nějaký hudební nástroj.)

$\mathcal{C}_3[\varphi_5][\varphi_6][\varphi_7] = clovek(@c) \rightarrow umi(@c, zpivat), hraje\_na(@c, @nastroj(@c))$
(Existuje člověk, který umí zpívat nebo hraje na nějaký hudební nástroj.)

Poznámka:

Existenční substituci ve skutečnosti můžeme uplatnit také na termy, ve kterých se vyskytují proměnné (takový funktor však nesmí obsahovat jako argumenty žádné existenční termy, opravdu pouze proměnné!). Je to možné z toho důvodu, že předpokládáme neprázdné univerzum diskurzu, a tedy vždy existuje prvek univerza, kterým lze ohodnotit proměnnou.

Úkol

Je dána klauzule C, substituce φ_1, φ_2 a φ_3 a existenční substituce φ_4 a φ_5:

$C = p\Big(f(X,a),b\Big),\ q(a,f(f(X,Y),Y),g(b))\Big) \rightarrow r\Big(f(Y,a),Y,g(a),b\Big)$

$\varphi_1 = \{g(a)/X,\ a/Y\}$
$\varphi_2 = \{Y/X,\ Y/Y\}$
$\varphi_3 = \{f(g(Z))/X,\ f(Z)/Y\}$
$\varphi_4 = (@m/a\,;\ 1\text{–}3)$
$\varphi_5 = (@c/b\,;\ 2,3)$

Vytvořte následující klauzule:

1. $C[\varphi_1] =$
2. $C[\varphi_3] =$
3. $C[\varphi_2][\varphi_1] =$
4. $C[\varphi_4] =$
5. $C[\varphi_2][\varphi_4] =$
6. $C[\varphi_4][\varphi_5] =$

3.6 Vztahy mezi klauzulemi

Pro logické programování budeme potřebovat možnost odvozování závěru, proto definujeme odvozovací pravidlo a ukážeme si způsob jak toto pravidlo použít.

3.6.1 Odvození důsledku dvojice klauzulí

V klauzulární logice používáme *rezoluční odvozovací pravidlo*, které lze odvodit z rezolučního odvozovacího pravidla predikátové logiky. Předpis pravidla je následující:

$$A_1, p \rightarrow K_1, \quad A_2 \rightarrow p, K_2 \quad \vdash \quad A_1, A_2 \rightarrow K_1, K_2 \tag{3.20}$$

Znamená to, že ve dvojici klauzulí najdeme tentýž atom (stejný *včetně argumentů*) p, a to v jedné klauzuli v antecedentu, v druhé v konsekventu, a výslednou klauzuli utvoříme z antecedentů a konsekventů obou klauzulí s výjimkou společného atomu p. Tento atom „odřízneme“, proto se toto pravidlo také nazývá *odvozovací pravidlo rezolučního řezu*.

Odvození z rezolučního pravidla predikátové logiky je následující:

$$\begin{array}{rcl} C \vee \neg p, \quad p \vee D & \models & C \quad \vee \quad D \\ (\neg A_1 \vee K_1) \vee \neg p, \quad p \vee (\neg A_2 \vee K_2) & \models & (\neg A_1 \vee K_1) \quad \vee \quad (\neg A_2 \vee K_2) \\ \neg A_1 \vee \neg p \vee K_1, \quad \neg A_2 \vee p \vee K_2 & \models & \neg A_1 \vee \neg A_2 \quad \vee \quad K_1 \vee K_2 \\ A_1 \,\&\, p \to K_1, \quad A_2 \to p \vee K_2 & \models & A_1 \,\&\, A_2 \quad \to \quad K_1 \vee K_2 \end{array}$$

Všechny zde použité operace vycházejí z ekvivalentních úprav pro formule predikátové logiky, je zde také použita substituce (pro C a D).

Příklad 3.20

Z uvedených klauzulí odvodíme závěr *Monika je matkou Honzy.*, a to tak, že pravidlo rezoluce uplatníme nejdřív na první a druhou klauzuli, pak na výsledek a třetí klauzuli.

1. $otec(pavel, honza), manzele(pavel, monika) \to matka(monika, honza)$
2. $\to otec(pavel, honza)$
3. $\to manzele(pavel, monika)$

4. $manzele(pavel, monika) \to matka(monika, honza)$ R(1,2)
5. $\to matka(monika, honza)$ R(3,4)

Při prvním použití odvozovacího pravidla rezolučního řezu je $p = otec(pavel, honza)$, při druhém použití je $p = manzele(pavel, monika)$.

Úkol

Je dána následující množina klauzulí. Odvoďte z ní pomocí odvozovacího pravidla rezolučního řezu odpovědi na tyto otázky:

1. Je Martin otcem Ely?
2. Je Jana Pavlovou babičkou?
3. Je Roman Honzovým pravnukem?

Množina klauzulí:

- $\to matka(jana, petr)$
- $\to otec(honza, dana)$
- $\to matka(jitka, pavel)$
- $\to otec(petr, pavel)$
- $\to manzele(dana, martin)$
- $\to matka(dana, ela)$
- $\to matka(ela, roman)$
- $matka(jana, petr), otec(petr, pavel) \to babicka(jana, pavel)$
- $otec(honza, dana), matka(dana, ela), matka(ela, roman) \to pravnuk(roman, honza)$
- $manzele(dana, martin), matka(dana, ela) \to otec(martin, ela)$

3.6.2 Unifikace klauzulí

Rezoluční odvození můžeme použít jen tehdy, pokud dvojice atomů, které chceme vyříznout, je absolutně stejná (včetně parametrů).

Pokud jsou parametry individuové konstanty (to znamená, že jde o bázový atom), není problém. Jestliže jsou parametry proměnné nebo existenční konstanty a my ještě nechceme provést bázovou substituci zvlášť pro každou z klauzulí, musíme počítat s tím, že obecně v různých klauzulích se jedná o různé termy, i když jsou stejně pojmenované (například když je „X“ ve dvou klauzulích, ve skutečnosti se jedná o dvě různé proměnné – je to totéž jako lokální proměnné funkcí při programování).

Postup (Unifikace dvou klauzulí)

Jak to pro dvojici klauzulí řešit:

1. pokud je to nutné, přejmenujeme v jedné klauzuli proměnné a existenční konstanty, které mají v obou klauzulích stejné názvy,
2. najdeme vhodnou substituci, kterou lze aplikovat na obě klauzule,
3. aplikujeme tuto substituci,
4. provedeme rezoluční odvození.

Substituce vhodná pro obě klauzule, po jejíž aplikaci lze použít rezoluční odvození (tj. mají stejný atom, jedna v antecedentu, druhá v konsekventu) se nazývá *unifikátor*. Pro jednu dvojici klauzulí může existovat více různých unifikátorů. Pokud unifikátor zvolíme nevhodně, lze sice rezoluční řez provést, ale výsledná klauzule může být pro další odvozování nevhodná (např. dosadíme konstantu někam, kam bychom potřebovali později dosazovat za proměnnou).

Unifikátor je tedy substituce aplikovatelná na dvě klauzule zároveň, tedy vždy definuje výrazy pro dosazení za všechny proměnné z obou formulí v jediné společné množině. Proto je před unifikací nutné odlišit názvy proměnných v různých klauzulích.

Příklad 3.21

Jsou dány tyto klauzule:

$p(f(X,a),b),\ q(@c,Y) \rightarrow r(X,Y,f(Y))$

$m(X,@c,Z) \rightarrow p(Z,X)$

Klauzule obsahují společný predikát p, ale zatím nejde o společný atom, protože v každé klauzuli má tento predikát odlišné parametry.

V obou klauzulích se vyskytuje proměnná X a existenční konstanta $@c$. Pokud chceme klauzule unifikovat a následně použít pravidlo rezolučního řezu, je třeba v jedné z klauzulí obojí přejmenovat:

$p(f(X,a),b),\ q(@c,Y) \rightarrow r(X,Y,f(Y))$

$m(A,@d,Z) \rightarrow p(Z,A)$

Zbývá najít vhodnou unifikaci (tedy substituci aplikovatelnou na obě klauzule) a provést rezoluci, což se naučíme v následujícím textu.

Unifikátor φ je obecnější (tj. šíře použitelný) než unifikátor σ, pokud σ lze dostat tak, že na klauzuli po aplikaci φ uplatníme ještě některou další substituci – unifikaci (zjednodušeně lze říci, že čím méně konstant – konkrétních hodnot, tím obecnější).

Definice 3.15 (Obecnost unifikátorů)

Nechť σ a φ jsou unifikátory dvojice klauzulí.

φ je *obecnější* než σ, jestliže existuje substituce λ taková, že platí $\varphi \circ \lambda = \sigma$, tedy je možno σ odvodit z φ.

φ je *nejobecnější* unifikátor, jestliže pro každý unifikátor γ existuje substituce λ taková, že platí $\varphi \circ \lambda = \gamma$ – jakýkoliv unifikátor pro stejnou dvojici klauzulí je možno odvodit z φ.

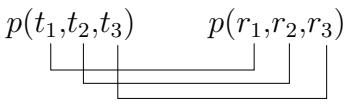
Z hlediska dalšího odvozování je samozřejmě nejvýhodnější použít nejobecnější unifikátor, protože se v něm vyskytuje nejméně konstant (za proměnnou je možné dále dosazovat, za konstantu už ne). Dále se seznámíme s algoritmem, který lze použít pro nalezení nejobecnějšího unifikátoru dvojice klauzulí.

Postup (Algoritmus pro zjištění nejobecnějšího unifikátoru)

Pokud potřebujeme najít nejvhodnější unifikátor, postupujeme následovně:

A) Vytvoříme *množinu neshod*
Z obou klauzulí, které chceme unifikovat, vezmeme atomy, přes které má být provedeno rezoluční odvození: $p(t_1, t_2, \ldots, t_n),\ p(r_1, r_2, \ldots, r_n)$, z jejich parametrů vytvoříme rovnice (to je právě ta množina neshod):

$t_1 = r_1$
$t_2 = r_2$
$\ldots$
$t_n = r_n$



B) Následující body algoritmu provádíme na tuto množinu neshod tak dlouho, dokud rovnice nejsou ve tvaru *Proměnná* = *výraz*, kde

- *Proměnná* se nevyskytuje na pravé straně žádné rovnice množiny,
- pro každou *Proměnnou* je zde nejvýše jeden *výraz*.

Nejobecnější unifikátor je pak
$\varphi = \{vyraz_1/Prom_1, vyraz_2/Prom_2, \ldots, vyraz_k/Prom_k\}$.

Opakujeme následující kroky:

1. Vyřadíme rovnice, které mají stejnou levou a pravou stranu (např. $X = X$ nebo $f(a, Y) = f(a, Y)$).
2. Rovnice ve tvaru *výraz* = *Proměnná* nahradíme rovnicemi *Proměnná* = *výraz* (přehodíme proměnnou doleva).
3. Rovnice ve tvaru $t_1 = t_2$, kde t_1 a t_2 nejsou proměnné:
   (a) jsou to různé funkční symboly (např. $f(X) = g(Y)$)
   $\Rightarrow$ KONEC, množina neshod nemá řešení.

(b) jinak: tuto rovnici nahradíme množinou neshod pro tyto termy.
Například rovnici $p(X, a, Z) = p(f(Y, b), Y, K)$ nahradíme rovnicemi
$X = f(Y, b),\ a = Y,\ Z = K$.

4. Rovnice ve tvaru *Proměnná* = *t*, kde *t* je jakýkoliv term:
   (a) jestliže se *Proměnná* vyskytuje v *t* (např. jako parametr funktoru)
   ⇒ KONEC, množina neshod nemá řešení.
   (b) jinak: všechny výskyty *Proměnné* v ostatních rovnicích nahradíme termem *t* (podobně jako v běžné aritmetické soustavě rovnic – dosadíme), samotnou rovnici neměníme.

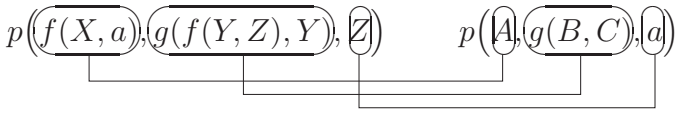
Pokud větev algoritmu končí tvrzením „⇒ KONEC, množina neshod nemá řešení.“, znamená to nejen, že pro zadanou dvojici klauzulí (přesněji atomů) neexistuje nejobecnější unifikátor, ale dokonce to, že pro ně neexistuje žádný unifikátor.

Příklad 3.22

Unifikujeme dvojici klauzulí:

$$\mathcal{C}_1 = p\Big(f(X,a), g(f(Y,Z),Y), Z\Big),\ r\Big(X, f(X,Y)\Big) \rightarrow m\Big(X, a\Big)$$
$$\mathcal{C}_2 = r\Big(A, b\Big),\ m\Big(f(C,B), b\Big) \rightarrow p\Big(A, g(B,C), a\Big)$$

Ve formulích jsou celkem tři predikáty – p, r a m (ostatní jsou funktory, proměnné a konstanty). Predikát r nelze použít pro rezoluci, protože se atomy s tímto predikátem vyskytují v obou formulích na stejné straně implikace. Predikát m je sice v atomech na různých stranách, ale z jeho druhého parametru bychom do množiny neshod zařadili rovnici $a = b$, která není přípustná (výpočet by skončil podle bodu 3a) výše uvedeného algoritmu).



Zaměříme se na predikát p. Žádná proměnná se nevyskytuje v obou klauzulích. Kdyby například proměnná X byla i v druhé klauzuli, museli bychom ji přejmenovat, protože všechny proměnné jsou lokální v rámci jedné klauzule. Množinu neshod vytvoříme podle odpovídajících si parametrů obou atomů:

$$\begin{aligned} f(X,a) &= A \\ g(f(Y,Z),Y) &= g(B,C) \\ Z &= a \end{aligned}$$

Na množinu neshod budeme rekurzívně uplatňovat výše uvedený algoritmus. V prvním kroku pouze přehodíme levou a pravou stranu v první rovnici.

1) $A = f(X,a)$ ..... podle bodu 2 algoritmu
$g(f(Y,Z),Y) = g(B,C)$
$Z = a$

2) Druhou rovnici zpracujeme podle bodu 3a) algoritmu (z parametrů funktoru g vytvoříme množinu neshod – jsou dva parametry, proto z jedné původní rovnice získáme dvě nové).

$A = f(X, a)$
$f(Y, Z) = B$ ..... podle bodu 3a) algoritmu
$Y = C$ ..... podle bodu 3a) algoritmu
$Z = a$

3) V obou přehodíme levou a pravou stranu podle bodu 2 algoritmu, protože proměnné B a C se v jiných rovnicích nevyskytují (narozdíl od Y a Z).

$A = f(X, a)$
$B = f(Y, Z)$ ..... podle bodu 2 algoritmu
$C = Y$ ..... podle bodu 2 algoritmu
$Z = a$

4) V druhé rovnici se na pravé straně vyskytuje proměnná Z, která je v jiné rovnici vlevo, tedy je třeba dosadit.

$A = f(X, a)$
$B = f(Y, a)$ ..... podle bodu 4b) algoritmu
$C = Y$
$Z = a$

Podle množiny rovnic získané v posledním bodu vytvoříme unifikátor (proměnné, které se nevyskytují na levé straně rovnic, budou dosazeny samy za sebe):

$$\varphi = \{X/X,\ Y/Y,\ a/Z,\ f(X, a)/A,\ f(Y, a)/B,\ Y/C\}$$

Tento unifikátor použijeme jako substituci u obou klauzulí:

$$\mathcal{C}_1[\varphi] \ = \ p\Big(f(X, a), g(f(Y, a), Y), a\Big),\ r\Big(X, f(X, Y)\Big) \ \rightarrow \ m\Big(X, a\Big)$$
$$\mathcal{C}_2[\varphi] \ = \ r\Big(f(X, a), b\Big),\ m\Big(f(Y, f(Y, a)), b\Big) \ \rightarrow \ p\Big(f(X, a), g(f(Y, a), Y), a\Big)$$

Po uplatnění rezolučního odvozovacího pravidla získáme klauzuli

$$\mathcal{C} = \ r\Big(X, f(X, Y)\Big), r\Big(f(X, a), b\Big),\ m\Big(f(Y, f(Y, a)), b\Big) \ \rightarrow \ m\Big(X, a\Big)$$

Poznámka:

Za určitých okolností lze pro unifikaci použít existenční substituci. Pak bychom měli použít nový (dosud nepoužitý) existenční term. Po provedení existenční substituce však mohou nastat problémy při odvozování, protože pro existenční termy platí totéž co pro proměnné – jsou lokální v rámci dané klauzule a pro jejich unifikaci vlastně ani nemáme v algoritmu předpis.

Může však nastat situace, kdy je třeba unifikovat dvojici klauzulí s univerzálně vázanou proměnnou s tím, že na daném místě potřebujeme existenční konstantu (například tehdy, když výsledkem odvození má být existenční tvrzení). Protože za univerzálně vázanou proměnnou lze dosadit cokoliv z univerza diskurzu, lze v tomto případě použít existenční substituci s jakoukoliv existenční konstantou včetně některé již použité.

Příklad 3.23

1. $rozbije(ferda, okno) \rightarrow provede(otec(ferda), zaskleni(okno))$
   (Když Ferda rozbije okno, Ferdův otec zařídí zasklení tohoto okna.)
2. $\rightarrow rozbije(@c, okno)$ (Někdo rozbil okno.)
3. $rozbije(@d, okno) \rightarrow provede(otec(@d), zaskleni(okno))$
   (existenční substituce do 1.)

   ... při substituci jsme museli použít dosud nepoužitou existenční konstantu, proto klauzule nejsou unifikovatelné.

Příklad 3.24

1. $rozbije(X, okno) \rightarrow provede(otec(X), zaskleni(okno))$
   (Když někdo rozbije okno, jeho otec zařídí zasklení tohoto okna.)
2. $\rightarrow rozbije(ferda, okno)$
   (Ferda rozbil okno.)
3. $rozbije(ferda, okno) \rightarrow provede(otec(ferda), zaskleni(okno))$ (subst. do 1.)
4. $\rightarrow provede(otec(ferda), zaskleni(okno))$ (rezoluce na 2., 3.)
5. $\rightarrow provede(otec(@c), zaskleni(okno))$ (existenční substituce na 4.)

Pokud nutně musí být provedena existenční substituce (chceme méně konkrétní existenční tvrzení v závěru), pak ji provádíme pokud možno až nakonec, protože může zkomplikovat další odvozování.

Příklad 3.25

1. $rozbije(X, okno) \rightarrow provede(otec(X), zaskleni(okno))$
   (Když někdo rozbije okno, jeho otec zařídí zasklení tohoto okna.)
2. $\rightarrow rozbije(@c, okno)$
   (Někdo rozbil okno.)
3. $rozbije(@c, okno) \rightarrow provede(otec(@c), zaskleni(okno))$
   (existenční substituce do 1.)
4. $\rightarrow provede(otec(@c), zaskleni(okno))$ (rezoluce na 2., 3.)

Existenční substituce v kroku 3 je unifikací pro rezoluci z následujícího kroku. Rezoluci pak můžeme provést jen z toho důvodu, že v unifikaci byla existenční konstanta dosazena za univerzálně vázanou proměnnou a univerzum diskurzu není prázdné.

Úkoly

1. Unifikujte dané klauzule tak, aby bylo možné provést rezoluční řez (použijte algoritmus pro nalezení nejobecnějšího unifikátoru):

   $p(f(X), g(Y)) \rightarrow r(Y, Z)$
   $\rightarrow p(f(f(c)), g(Z))$

2. Podle následujících vět sestavte množinu klauzulí klauzulární logiky. Použijte tyto predikáty:

   $snehulak(\langle kdo \rangle)$
   $nos(\langle kdo \rangle, \langle z\ ceho \rangle)$

   Věty:

   (a) Lucka je ze sněhu a má nos z mrkve.
   (b) Sněhulák Pepík nemá nos z mrkve. *(tady pozor, ve větě jsou dva fakty)*
   (c) Každý sněhulák má nějaký nos.
   (d) Který sněhulák nemá nos z mrkve, má ho z větvičky.

   Z množiny klauzulí odvoďte klauzuli „$\rightarrow nos(pepa, vetvicka)$“, tj. zjistěte, zda má Pepa nos z větvičky.

3.6.3 Znalostní báze

Znalostní báze pro nás bude množina tvrzení o *uzavřeném* modelovaném světě. To, že je modelovaný svět uzavřený, znamená, že při odvozování budeme brát v úvahu pouze obsah báze; co v ní nebude, to se neobjeví ani v odvozovaných závěrech (jako by to neexistovalo).

Definice 3.16 (Znalostní báze)
Znalostní báze (báze znalostí) je neprázdná množina klauzulí jazyka klauzulární logiky, tyto klauzule nazýváme *speciální axiomy* báze nebo předpoklady důkazu. Mezi klauzulemi báze je vztah konjunkce.

Poznámka:
Znalostní bázi si můžeme představit jako formuli

$$(A_1 \rightarrow K_1) \quad \& \quad (A_2 \rightarrow K_2) \quad \& \quad \ldots \quad \& \quad (A_r \rightarrow K_r) \tag{3.21}$$

(jak již víme, v množině klauzulí je mezi jednotlivými klauzulemi vztah konjunkce).

Znalostní báze nám především slouží v logickém programování ke stanovení prostředí, ve kterém chceme odvozovat a zjišťovat odpovědi na otázky, je to jakási množina předpokladů důkazu. Je to obdoba báze znalostních systémů, které pracují na podobném principu.

V sekci 3.3 o sémantice byl svět, ve kterém provádíme odvozování, určen nejen klauzulemi, ale také strukturou – univerzem diskurzu, funkcemi a relacemi. Aby bylo možné bázi co nejjednodušeji naprogramovat a dále používat, budeme také jednotlivé prvky struktury (univerzum, funkce, relace) reprezentovat klauzulemi.

Postup (Vytvoření znalostní báze)

Bázi vytvoříme takto:

1. zavedeme vhodné predikáty,
2. zavedeme vztahy mezi predikáty, tedy vytvoříme klauzule,
3. přidáme fakty o konkrétní situaci podle interpretační struktury,
4. můžeme začít testovat pravdivost některé klauzule.

Ve znalostní bázi se budou nacházet dva druhy klauzulí: pravidla a fakty.

Pravidla nám říkají, co platí, když platí určité předpoklady, jsou to takové klauzule, se kterými jsme až dosud pracovali. Obvykle obsahují proměnné nebo existenční termy (nebo obojí).

Fakty o konkrétní situaci jsou většinou klauzule reprezentující funkce a relace struktury a valuaci, vztahují se tedy k interpretaci klauzulí.

Postup (Přepis funkce do znalostní báze)

V případě *přepisu funkce* s předpisem $f : (X_1, X_2, \ldots, X_n) \longrightarrow Y$ použijeme predikát rovnosti. Pokud je tato funkce dána tabulkou 3.1, vytvoříme množinu klauzulí

$\rightarrow f(a_{11}, a_{12}, \ldots, a_{1n}) = b_1$
$\rightarrow f(a_{21}, a_{22}, \ldots, a_{2n}) = b_2$
$\vdots$

Předpokladem je, že n-tice $(a_{i1}, a_{i2}, \ldots, a_{in})$ z řádků tabulky jsou vždy po dvou různé (jinak by ani nešlo o funkci). Pokud tento předpoklad není splněn, musíme vytvořit novou relaci (arity $n + 1$), kterou použijeme pro definování faktů, a vhodně upravíme predikáty a klauzule báze.

X_1	X_2	$\ldots$	X_n	$f(X_1, X_2, \ldots, X_n)$
a_{11}	a_{12}	$\ldots$	a_{1n}	b_1
a_{21}	a_{22}	$\ldots$	a_{2n}	b_2
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$

Tabulka 3.1: Funkce ze struktury pro interpretaci klauzulí

Postup (Přepis relace do znalostní báze)

Při *přepisu relace* dosadíme do argumentů příslušného predikátu všechny řádky relace a z každého takového atomu vytvoříme jednu klauzuli. Pro relaci R z tabulky 3.2 vzniknou klauzule

$\rightarrow R(a_{11}, a_{12}, \ldots, a_{1n})$
$\rightarrow R(a_{21}, a_{22}, \ldots, a_{2n})$
$\vdots$

Tento postup lze použít také pro přepis valuace (ohodnocení) proměnných, ale pouze tehdy, když stejně pojmenované

R :

X_1	X_2	$\ldots$	X_n
a_{11}	a_{12}	$\ldots$	a_{1n}
a_{21}	a_{22}	$\ldots$	a_{2n}
$\ldots$	$\ldots$	$\ldots$	$\ldots$

Tabulka 3.2: Relace ze struktury pro interpretaci klauzulí

proměnné v různých klauzulích jsou považovány za tytéž proměnné. Většina logických programovacích jazyků však takové proměnné obecně považuje za různé, o tutéž proměnnou jde pouze v rámci jedné klauzule.

Příklad 3.26

Vytvoříme znalostní bázi obsahující klauzule podle následujících vět a v této bázi odvodíme odpověď na otázku „Má Hanička dobrou náladu?“

- O vánocích děti zpívají všechny (známé) koledy.
- Kdo zpívá radostnou písničku, má dobrou náladu, kdo zpívá smutnou písničku, je smutný.
- Někdo není smutný.
- Dětské písničky jsou radostné, a taky všechny písničky, které jsou stejného typu jako „Nesem vám noviny“, jsou radostné.
- Jsou vánoce, Hanička je dítě a písnička „Nesem vám noviny“ je koleda.

Navrhneme predikáty a funktory (zde vlastně jen predikáty):

$doba(\langle jaka \rangle)$	$typ\_pisne(\langle nazev\_pisne \rangle, \langle typ \rangle)$
$dite(\langle jmeno \rangle)$	$pisnicka(\langle nazev\_pisne \rangle, \langle nalada\_pisne \rangle)$
$zpiva(\langle kdo \rangle, \langle co \rangle)$	$nalada(\langle kdo \rangle, \langle jaka \rangle)$

Sestavíme bázi a odvodíme potřebnou klauzuli:

1. $doba(vanoce), dite(X), typ\_pisne(Y, koleda) \rightarrow zpiva(X, Y)$
2. $zpiva(X, Y), pisnicka(Y, radostna) \rightarrow nalada(X, dobra)$
3. $zpiva(X, Y), pisnicka(Y, smutna) \rightarrow nalada(X, smutna)$
4. $nalada(@c, smutna) \rightarrow$
5. $typ\_pisne(X, detska) \rightarrow pisnicka(X, radostna)$
6. $typ\_pisne(nesem\_vam\_noviny, X), typ\_pisne(Y, X) \rightarrow pisnicka(Y, radostna)$
7. $\rightarrow doba(vanoce)$
8. $\rightarrow dite(hanicka)$
9. $\rightarrow typ\_pisne(nesem\_vam\_noviny, koleda)$
10. rezoluce na 1 a 7:
    $dite(X), typ\_pisne(Y, koleda) \rightarrow zpiva(X, Y)$
11. substituce do 10, $\{hanicka/X, nesem\_vam\_noviny/Y\}$:
    $dite(hanicka), typ\_pisne(nesem\_vam\_noviny, koleda) \rightarrow zpiva(hanicka, nesem\_vam\_noviny)$
12. rezoluce na 8 a 11:
    $typ\_pisne(nesem\_vam\_noviny, koleda) \rightarrow zpiva(hanicka, nesem\_vam\_noviny)$
13. rezoluce na 9 a 12:
    $\rightarrow zpiva(hanicka, nesem\_vam\_noviny)$

14. substituce do 6, $\{koleda/X, nesem\_vam\_noviny/Y\}$:
$typ\_pisne(nesem\_vam\_noviny, koleda)$,
$typ\_pisne(nesem\_vam\_noviny, koleda)$
$\rightarrow\ pisnicka(nesem\_vam\_noviny, radostna)$
15. rezoluce na 9 a 14 (dvakrát):
$\rightarrow\ pisnicka(nesem\_vam\_noviny, radostna)$
16. substituce do 2, $\{hanicka/X, nesem\_vam\_noviny/Y\}$:
$zpiva(hanicka, nesem\_vam\_noviny), pisnicka(nesem\_vam\_noviny, radostna)$
$\rightarrow\ nalada(hanicka, dobra)$
17. rezoluce na 13 a 16:
$pisnicka(nesem\_vam\_noviny, radostna)\ \rightarrow\ nalada(hanicka, dobra)$
18. rezoluce na 15 a 17:
$\rightarrow\ nalada(hanicka, dobra)$

Většinu pojmů souvisejících s interpretací znalostní báze (struktura, denotace, ohodnocení, atd.) jsme probrali v kapitole 3.3 od strany 49. V následující definici nalezneme ostatní potřebné pojmy.

Definice 3.17 (Aplikovatelná struktura, logický důsledek báze)

Struktura aplikovatelná na znalostní bázi je taková struktura, která umožňuje přiřadit každé individuové konstantě vyskytující se v klauzulích báze některý prvek univerza diskurzu struktury, každému funktoru funkci z množiny funkcí struktury a každému predikátu relaci z množiny relací struktury.

Model znalostní báze je taková struktura aplikovatelná na bázi, ve níž jsou všechny klauzule báze platné.

Klauzule je *logickým důsledkem* znalostní báze, jestliže je platná ve všech modelech této báze.

V předchozím příkladu je aplikovatelná struktura přímo součástí znalostní báze. Trochu to sice komplikuje případné změny struktury a tím i interpretace, ale ulehčuje to odvozování a tento postup odpovídá metodám používaným v logickém programování.

Kapitola 4

Klauzulární axiomatický systém

Rychlý náhled: Klauzulární axiomatický systém je založen na klauzulární logice. Provádění důkazů v tomto systému odpovídá postupům používaným v logických programovacích jazycích. V tomto systému můžeme používat přímé i nepřímé důkazy.

V této kapitole definujeme formální systém obdobně jako v kapitole věnované Systému přirozené dedukce, ale pouze v základech, velmi stručně. Naznačíme také důkaz korektnosti tohoto systému.

Klíčová slova: Klauzulární axiomatický systém, logický axiom, speciální axiom, znalostní báze, odvozovací pravidlo substituce, existenční substituce a rezoluční, Modus Ponens, Modus Tolens, přímý a nepřímý důkaz.

Cíle studia: Cílem této kapitoly je ukázat vytvoření uceleného formálního systému na bázi klauzulární logiky. Účelem je osvětlit si formální význam znalostní báze, na kterémžto principu stojí deklarativní programovací jazyky včetně logických.

4.1 Definice systému

Jazyk Klauzulárního axiomatického systému přejímáme z klauzulární logiky. Syntaxe staví na logických a speciálních axiomech, a pak na několika odvozovacích pravidlech.

Logické axiomy (A) jsou takové klauzule klauzulární logiky, ve kterých se tentýž atom (včetně argumentů) vyskytuje v antecedentu i v konsekventu, tedy formule

$$p_1, p_2, \ldots, p_n, p \quad \rightarrow \quad p, q_1, q_2, \ldots, q_m \tag{4.1}$$

Speciální axiomy (SA) jsou klauzule nacházející se ve znalostní bázi. Znalostní báze nesmí být sporná množina.

Účelem speciálních axiomů je tedy konkretizovat „svět“, ve kterém se budeme při práci v systému pohybovat (tedy znalostní bázi). Pokud změníme speciální axiomy, změníme i svět, ve kterém chceme dojít k závěrům. Výběr speciálních axiomů tedy má přímý vliv na to, k jakým závěrům dojdeme.

Odvozovací pravidla jsou tři:

1. *Odvozovací pravidlo substituce* (S): Jestliže $\varphi = \{t_1/X_1, t_2/X_2, \ldots, t_n/X_n\}$ je substituce a $\mathcal{C}$ je klauzule, pak platí
$$\mathcal{C} \vdash \mathcal{C}[\varphi] \tag{4.2}$$
2. *Rezoluční odvozovací pravidlo* (odvozovací pravidlo rezolučního řezu, R): Označme p atom a A_1, A_2, K_1, K_2 množiny atomů. Pak platí
$$A_1, p \to K_1, \quad A_2 \to p, K_2 \quad \vdash \quad A_1, A_2 \to K_1, K_2 \tag{4.3}$$
3. *Odvozovací pravidlo existenční substituce* (ES): Jestliže $\varphi = (@c/a; k_1, k_2, \ldots, k_n)$ je existenční substituce a $\mathcal{C}$ je klauzule, pak platí
$$\mathcal{C} \vdash \mathcal{C}[\varphi] \tag{4.4}$$

V klauzulární logice používáme pro usnadnění odvozování také tato *pomocná odvozovací pravidla:*

1. *Logické zákony pro predikát rovnosti:* Jestliže t, r, s jsou termy a A je predikát, pak platí
   - $\to \ t = t$ — reflexivita predikátu rovnosti (RR)
   - $t = r \ \to \ r = t$ — symetrie predikátu rovnosti (SR)
   - $t = r, \ r = s \ \to \ t = s$ — tranzitivita predikátu rovnosti (TR)
   - $t = r, \ A(t) \ \to \ A(r)$ — bázová substituce (BS)

     ($A(t), A(r)$ jsou bázové atomy)
2. *Pomocné odvozovací pravidlo kontrakce* (KK):
   - $p, p, A \to K \quad \vdash \quad p, A \to K$
   - $A \to p, p, K \quad \vdash \quad A \to p, K$

   Více výskytů téhož atomu (shodných včetně argumentů) v antecedentu lze zredukovat na jeden výskyt, totéž platí o konsekventu. Pravidlo lze rovněž použít na odstranění atomu $true$ z antecedentu nebo $false$ z konsekventu.
3. *Pomocné odvozovací pravidlo přeuspořádání atomů* (PA): Atomy v antecedentu lze přeuspořádat (změnit jejich pořadí). Atomy v konsekventu lze přeuspořádat.

Příklad 4.1

Odvodíme pravidla *Modus Ponens* (MP) a *Modus Tolens* (MT).

$$\text{MP}: \quad \to P\,, \ P \to Q \quad \vdash \quad \to Q \tag{4.5}$$
$$\text{MT}: \quad Q \to\,, \ P \to Q \quad \vdash \quad P \to \tag{4.6}$$

Modus Ponens:

1. $\to P$	SA_1
2. $P \to Q$	SA_2
3. $\to Q$	R(1,2)

Modus Tolens:

1. $Q \to$	SA_1
2. $P \to Q$	SA_2
3. $P \to$	R(1,2)

V obou případech máme dva speciální axiomy (tj. předpoklady) a používáme jednou rezoluční odvozovací pravidlo.

Poznámka:

Substituci můžeme provést v těchto případech:

- substituce bázového termu (například konstanty) za proměnnou,
- substituce nové (v klauzuli se dosud nevyskytující) proměnné za proměnnou,
- substituce termu, v němž se vyskytují pouze nové proměnné, za proměnnou,
- substituce existenční konstanty za bázový term.

Poslední bod se týká existenční substituce. Podmínky dané zbylými body splňuje mj. nejobecnější unifikátor, který lze nalézt podle algoritmu na str. 68 v kap. 3.6.2.

4.2 Formální důkazy

V Klauzulárním axiomatickém systému můžeme provádět jak přímé, tak i nepřímé důkazy. Logické programovací jazyky obvykle pracují na principu nepřímého důkazu, protože tento způsob je jednodušší, lépe technicky realizovatelný.

Definice 4.1 (Přímý důkaz)

Přímý důkaz klauzule $\mathcal{C}$ z ze znalostní báze (množiny speciálních axiomů) $\mathcal{B}$ je posloupnost klauzulí $A_1, A_2, \ldots, A_m$, kde $\mathcal{C} = A_m$ a pro každé $i \in \{1, \ldots, m\}$ platí pro A_i některá z těchto možností:

- $A_i \in \mathcal{B}$ (tj. je to některý ze speciálních axiomů báze),
- A_i je logický axiom,
- A_i vznikla použitím některého odvozovacího pravidla na předchozí členy posloupnosti.

Příklad 4.2

Vyjádříme v klauzulích klauzulární logiky znalostní bázi a odvodíme podle ní odpověď na dotaz „Jede Honza lodí?“ *přímým důkazem*. Znalostní báze:

- Když země leží za mořem, každý, kdo do ní cestuje, jede lodí nebo letadlem.
- Německo neleží za mořem, ale Anglie ano.
- Honza cestuje do Anglie. Honza nejezdí letadlem.

Řešení:

1. $lezi(X, za\_morem), cestuje(Y, X) \rightarrow jede(Y, lod), jede(Y, letadlo)$ SA_1
2. $lezi(nemecko, za\_morem) \rightarrow$ SA_2
3. $\rightarrow lezi(anglie, za\_morem)$ SA_3
4. $\rightarrow cestuje(honza, anglie)$ SA_4
5. $jede(honza, letadlo) \rightarrow$ SA_5

6. $lezi(anglie, za\_morem), cestuje(honza, anglie)$
   $\rightarrow jede(honza, lod), jede(honza, letadlo)$ S(1)$\{anglie/X, honza/Y\}$
7. $cestuje(honza, anglie) \rightarrow jede(honza, lod), jede(honza, letadlo)$ R(3,6)
8. $\rightarrow jede(honza, lod), jede(honza, letadlo)$ R(4,7)
9. $\rightarrow jede(honza, lod)$ R(5,8)

Prvních pět klauzulí je ze znalostní báze, další klauzule jsme odvodili s použitím odvozovacích pravidel substituce a rezoluce.

Definice 4.2 (Nepřímý důkaz klauzule)

Nepřímý důkaz klauzule $\mathcal{C}$ ze znalostní báze $\mathcal{B} = \{SA_1, SA_2, \ldots, SA_n\}$ je posloupnost klauzulí $A_1, A_2, \ldots, A_m$, kde $A_m = \rightarrow$ (prázdná klauzule, vždy interpretovná jako $false$) a pro každé $i \in \{1, \ldots, m\}$ platí pro A_i některá z těchto možností:

- A_i patří do popírající množiny klauzule $\mathcal{C}$,
- $A_i \in \mathcal{B}$ (tj. je to některý ze speciálních axiomů báze),
- A_i je logický axiom,
- A_i vznikla použitím některého odvozovacího pravidla na předchozí členy posloupnosti.

Postup je následující:

1. vytvoříme popírající množinu pro klauzuli, kterou chceme dokázat (viz kapitolu 3.4.4 na straně 60),
2. tuto popírající množinu (PM) přidáme k znalostní bázi,
3. odvozujeme s použitím odvozovacích pravidel,
4. jestliže odvodíme prázdnou klauzuli ($\rightarrow$), klauzule je dokázána (vyplývá ze znalostní báze).

Příklad 4.3

Zadání předchozího příkladu vyřešíme *nepřímým důkazem*.

1. $lezi(X, za\_morem), cestuje(Y, X) \rightarrow jede(Y, lod), jede(Y, letadlo)$ SA_1
2. $lezi(nemecko, za\_morem) \rightarrow$ SA_2
3. $\rightarrow lezi(anglie, za\_morem)$ SA_3
4. $\rightarrow cestuje(honza, anglie)$ SA_4
5. $jede(honza, letadlo) \rightarrow$ SA_5
6. $jede(honza, lod) \rightarrow$ PM
7. $lezi(anglie, za\_morem), cestuje(honza, anglie)$
   $\rightarrow jede(honza, lod), jede(honza, letadlo)$ S(1)$\{anglie/X, honza/Y\}$
8. $cestuje(honza, anglie) \rightarrow jede(honza, lod), jede(honza, letadlo)$ R(3,7)

9. $\rightarrow jede(honza, lod), jede(honza, letadlo)$ R(4,8)
10. $\rightarrow jede(honza, lod)$ R(5,9)
11. $\rightarrow$ R(6,10)

Prvních pět klauzulí je stejných jako v předchozím příkladu (znalostní báze), následuje popírající množina dokazované klauzule (popírající množina obsahuje pouze jednu klauzuli, protože dokazovaná klauzule obsahuje jen jeden atom). Poslední člen důkazu je prázdná klauzule, čímž jsme dokončili nepřímý důkaz.

Úkol

Podle následujících vět sestavte znalostní bázi, vyberte vhodně predikáty pro vyjádření faktu, že někdo je lev či zajíc, někdo má konkrétní barvu, je silný, atd.

1. Lvi jsou žlutí a velcí. Každý lev je silný.
2. Zajíci jsou rychlí, ale nejsou silní. Mají hnědou nebo béžovou barvu a jsou malí.
3. Kdo není velký, je malý.
4. Lvi, kteří nejsou zranění, jsou rychlí.
5. Králové zvířat jsou právě a jenom ti, kteří jsou velcí, silní a rychlí. *(pozor, ekvivalence!)*
6. Hektor je lev a Zulejda je zajíc. Žádný z nich není zraněný.

Pro kontrolu: celkový počet klauzulí ve znalostní bázi by měl být 17. Zjistěte, zda ze znalostní báze vyplývají tato tvrzení, proveďte poznačený typ důkazu:

1. Hektor je velký. *(přímý i nepřímý důkaz, porovnejte jejich délku)*
2. Hektor je králem zvířat. *(přímý důkaz)*
3. Zulejda není králem zvířat. *(nepřímý důkaz)*
4. Existuje někdo malý. *(nepřímý důkaz, pozorně vytvořte popírající množinu)*

4.3 Korektnost systému

Korektnost Klauzulárního axiomatického systému budeme dokazovat převedením problému do predikátové logiky.

Věta 4.1 (Korektnost Klauzulárního axiomatického systému)

Klauzulární axiomatický systém je korektní.

Důkaz: *Logický axiom* je každá klauzule, jejíž množiny antecedentu a konsekventu mají neprázdný průnik (tedy existuje alespoň jeden atom, který je v antecedentu i v konsekventu). Je to tedy klauzule ve tvaru $A, p \ \rightarrow p, K$. Po převodu do predikátové logiky platí tato posloupnost ekvivalencí:

$$(A \,\&\, p) \rightarrow (p \vee K) \ \Leftrightarrow \ (\neg A \vee \neg p) \vee (p \vee K) \ \Leftrightarrow \ \neg A \vee (\neg p \vee p) \vee K \ \Leftrightarrow \ \neg A \vee 1 \vee K \ \Leftrightarrow \ 1$$

Tedy logický axiom je logicky platná formule.

Rezoluční odvozovací pravidlo přepsané do predikátové logiky je odvoditelné z rezolučního pravidla pro predikátovou logiku, jak je uvedeno v kapitole 3.6.1 na straně 65. To je snadno dokazatelné s použitím sémantických metod predikátové logiky, dokazovali jsme také ekvivalent pro Systém přirozené dedukce v kapitole 1 na straně 32 (máme dokázáno, že Systém přirozené dedukce je korektní, tedy důkaz v tomto systému má stejnou váhu jako důkaz sémantickými metodami predikátové logiky).

Odvozovací pravidla substituce a existenční substituce jsou také převeditelná do predikátové logiky, v definici substituce pro klauzulární logiku na straně 63 je dán požadavek na substituovatelnost termů za proměnné.

V případě existenční substituce se dá korektnost dokázat s použitím interpretace klauzule: jestliže něco platí pro nějaký konkrétní prvek univerza diskurzu (tj. tento prvek je použit jako term v klauzuli), pak existuje takový prvek univerza diskurzu, pro který to platí (tento prvek můžeme ve všech nebo některých jeho výskytech nahradit novou existenční konstantou).

Korektnost posloupnosti důkazu se dokazuje stejně jako v případě dříve probíraných formálních systémů.

□

Kapitola 5

Logické programování

Rychlý náhled: V této kapitole se budeme zabývat jedním z programovacích jazyků pro logické proramování, Prologem. Naučíme se s Prologem pracovat, tedy vytvářet programy a používat je. V druhé části kapitoly probereme možné způsoby implementace rezoluce v logických programovacích jazycích a podíváme se na způsob implementace použitý v Prologu.

Klíčová slova: Logické programování, Prolog, program, pravidlo, fakt, dotaz, cílová klauzule, anonymní proměnná, predikát rovnosti, rekurze, rezoluce, rezoluční uzávěr množiny klauzulí, lineární výpočetní strom, zásobník.

Cíle studia: Po prostudování této kapitoly se seznámíte se základy programování v programovacím jazyce Prolog. Naučíte se sestavit program, tedy znalostní bázi, a pokládat dotazy, které mají být podle programu vyhodnoceny. Dále nahlédnete do vnitřního fungování jak samotného Prologu, tak i dalších deklarativních programovacích jazyků.

5.1 Pár slov k programovacím jazykům

V tomto předmětu se předpokládá, že studenti alespoň tuší, jak je to s programovacími jazyky. Nicméně alespoň pár slov na úvod. Při programování existují dvě základní paradigmata (tj. principy): procedurální (také imperativní) a deklarativní (také logické). Každé paradigma je vhodné pro trochu jiné typy zadání.

Běžné programovací jazyky jako C, C++, Java, C#, Python a další jsou procedurální, což znamená, že program tvoříme jako posloupnost kroků, které je třeba udělat (tj. přímo sdělujeme, jak se má postupovat). Deklarativní programování spočívá v tom, že netvoříme přímo postup, ale v programu sdělujeme, čeho má být dosaženo.

Mimochodem, třetím používaným paradigmatem je funkcionální programování, které si bere z obou předchozích něco: program je zde jedna velká funkce, ve které jsou vnořeny další funkce, v nich opět může být vnořeno něco dalšího, atd. Účelem je maximálně se vyhnout používání proměnných (pokud funkce vrátí vypočtenou hodnotu nadřízené funkci, tak dotyčné místo existuje jen chvíli v zásobníku funkcí), což je praktické zejména u Big Data (kde šetříme pamětí co to jde). K nejznámějším funkcionálním jazykům patří Haskell, Scala, F# a Dart, ale funkcionálně

lze programovat i v „běžných“ jazycích jako je třeba Python nebo Java. Další paradigmata jsou objektové či stavové programování.

Dále se budeme věnovat logickému programování používajícímu deklarativní paradigma, konkrétně jazyku Prolog (dalším známým deklarativním jazykem je SQL). Protože nezadáváme postup, musí být nějaký způsob, jak si překladač ten postup odvodí sám.

5.2 Logické programování v Prologu

Prolog je jedním z jazyků pro logické programování. Vznikl ve Francii v roce 1973 (prof. A. Colmerauer) a jeho název je zkratka z francouzského PROgrammation à LOGique („programování v logice“). Je to deklarativní (neimperativní) jazyk. Většina překladačů je pouze interpretační nebo umožňují obojí, kompilace Prologu (GNU Prolog) obvykle znamená vytvoření jakéhosi mezikódu (například GNU Prolog generuje kód WAM – Warren Abstract Machine), který je pak přeložen na binární (spustitelný) soubor.

Existuje mnoho implementací Prologu. K nejznámějším patří

- SWI Prolog šířený pod licencí GPL, používaný v unixových systémech včetně Linuxu a MacOS, a Windows, nejnovější verze má také integrován editor a jednoduchého správce kódu, pro (obousměrné) přemostění mezi Javou a Prologem lze použít JPL,
- SWISH (SWI Prolog for Sharing) je online dostupná varianta Prologu, hodí se tehdy, když nechceme nic instalovat,
- LPA Win Prolog je komerční program pro Windows,
- GNU Prolog pro unixové systémy včetně Linuxu, přes Cygwin lze provozovat i ve Windows, existuje také GNU Prolog for Java ve formě knihovny do Javy (gnu.prolog),
- InterProlog umožňuje výpočetní část implementovat v Prologu a grafické rozhraní řešit v Javě,
- Amzi! Prolog, Visual Prolog, DGKS Prolog (implementován v Javě), Strawbery Prolog, OpenProlog, Mercury, SICStus Prolog, Quintus Prolog (oba komerční), Temporal Prolog, atd.

Jednotlivé Prology se liší nejen licení, vzhledem a vybaveností svého editoru (většina Prologů má vlastní editor, se kterým je provázán), ale bohužel v některých případech také syntaxí programovacího jazyka. Rozdíly jsou například v práci se soubory. Jak plyne z výše uvedeného seznamu, existují i možnosti propojení s jinými programovacími jazyky.

Zde budeme používat volně šiřitelný SWI Prolog, který najdeme ke stažení (včetně manuálu) například na adrese http://www.swi-prolog.org/. Prostředí SWI Prologu vidíme na obrázku 5.1.

Úkol

Pokud jste tak ještě neučinili, najděte si vhodnou implementaci Prologu. Můžete použít také online verzi SWI Prologu (SWISH), která je dostupná na https://swish.swi-prolog.org/.

5.2.1 Zápis klauzulí v Prologu

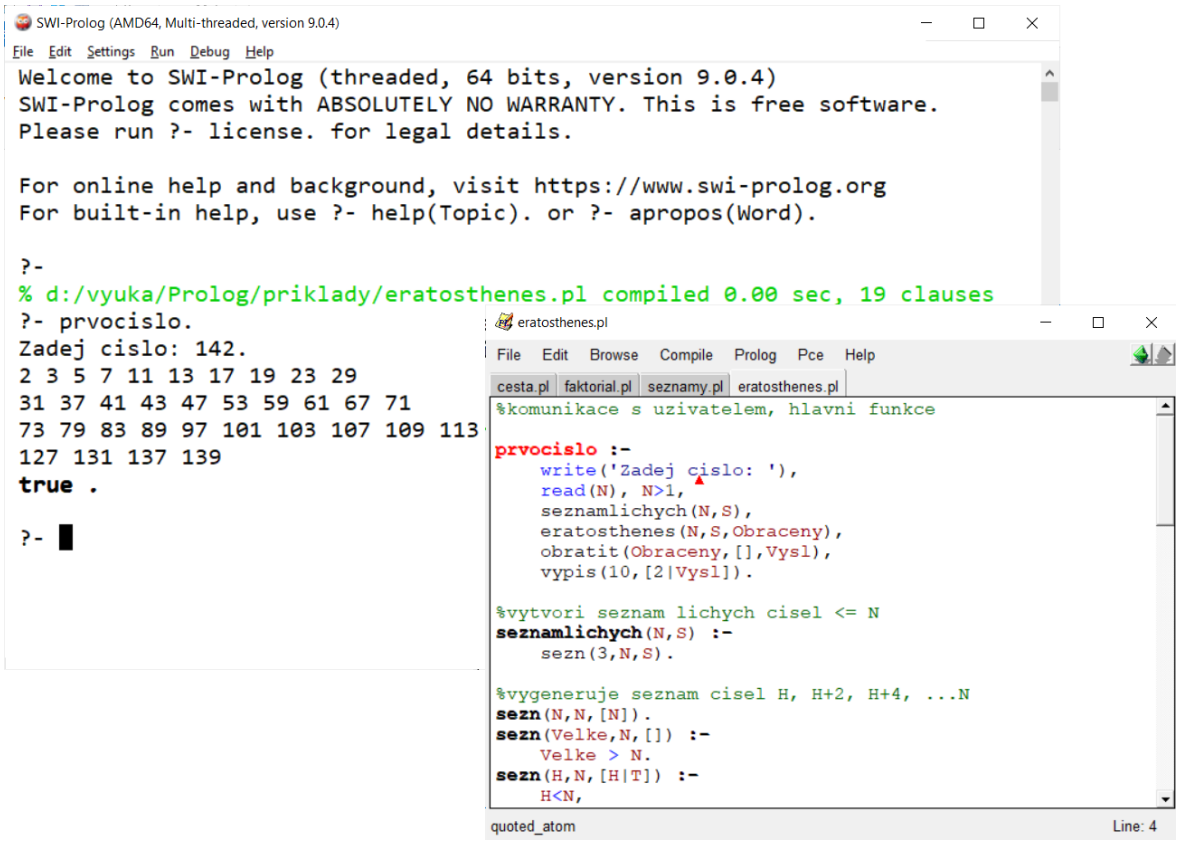
Definice 5.1 (Program v Prologu)

Program v Prologu je konečná neprázdná množina Hornových klauzulí (viz kapitolu 3.1 na straně 43). Je to ekvivalent znalostní báze klauzulární logiky. V programu lze použít dva druhy klauzulí:

- *pravidla* – obecná tvrzení ve tvaru „Závěr platí, pokud platí všechny jeho předpoklady zároveň.“
- *fakty* – tvrzení bez předpokladů, je to obdoba toho, co jsme měli v předchozí kapitole jako speciální axiomy.

Používání programu spočívá v zadávání *dotazů* (cílových klauzulí) – Hornových klauzulí bez pozitivních literálů. Prolog dotazy vyhodnocuje podle programu a podle vnitřních pravidel (obdoba logických axiomů).

Zápis jednotlivých elementů ukazuje tabulka 5.1. Každý element (příkaz, prologovskou klauzuli) vždy ukončíme tečkou. V pravidle rozlišujeme *tělo pravidla* (podle tabulky 5.1 to je B, C, D) a *hlavu pravidla* (A), tedy pravidlo je ve tvaru *hlava* :- *tělo*. Pro přehlednost se v delším pravidle cíl s oddělujícím dvojznakem zapisuje na samostatný řádek, tělo může být na více řádcích.



Obrázek 5.1: SWI Prolog ve verzi 9.04 pro Windows

	Klauzulární logika	**Predikátová klauzule**	**Zápis v Prologu**
Pravidlo	$B, C, D \rightarrow A$	$A \vee \neg B \vee \neg C \vee \neg D$	A :- B, C, D.
Fakt	$\rightarrow A$	A	A.
Dotaz	$B, C, D \rightarrow$	$\neg B \vee \neg C \vee \neg D$	?- B, C, D.

Tabulka 5.1: Zápis elementů v Prologu

V případě dotazu dvojznak `?-` nezapisujeme, jde o prompt (výzvu) Prologu.

Příklad 5.1

Proměnné zapisujeme velkým počátečním písmenem, konstanty malým počátečním písmenem. Konstantou je i číslo nebo řetězec v uvozovkách či apostrofech. To před závorkou je predikát, v závorce jsou parametry (argumenty).

Příklady faktů:

```
kocka(micka).     % Micka je kočka.
mys(jerry).      % Jerry je myš.
pes(zoubek).     % Zoubek je pes.
otec(jan, klara).   % Jan je otcem Kláry.
```

V pravidle píšeme nejdřív závěr a pak za dvojznakem `:-` předpoklady. Příklady pravidel:

```
lovi(Kdo, Koho) :-     % Kočky loví myši (když je „Kdo" kočka a „Koho" myš)
kocka(Kdo), mys(Koho).

prcha(Kdo, PredKym) :-      % Kočky prchají před psy.
kocka(Kdo), pes(PredKym).
```

Jako implikaci v predikátové logice bychom tato pravidla zapsali takto:

```
kocka(Kdo) & mys(Koho) -> lovi(Kdo, Koho)
kocka(Kdo) & pes(PredKym) -> prcha(Kdo, PredKym)
```

Pokud bychom se chtěli na něco dotázat, zápis bude následující:

```
prcha(micka, zoubek).      % Prchá Micka před Zoubkem?
prcha(X, zoubek).      % Kdo prchá před Zoubkem?
```

Pokud by nám bylo proti mysli, aby jména začínala malým písmenem, můžeme je napsat velkým, ale v uvozovkách či apostrofech (pozor, není to totéž, takže když se pro konkrétní zápis konstanty rozhodneme, musíme se ho držet). Takže pro Micku můžeme vybrat ze tří možností:

```
micka    "Micka"    'Micka'
```

Sestavený program se uloží do textového souboru s příponou .pl a konzultuje, tedy předá se překladači jazyka Prolog.

Příklad 5.2

Převedeme zadané klauzule klauzulární logiky na prologovské klauzule:

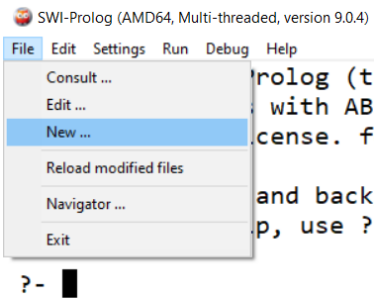
1. „Jahoda je červená."
   Klauzulární logika: $\rightarrow barva(jahoda, cervena)$
   Prolog: `barva(jahoda,cervena).`

2. „Ferda je dítě.“
   Klauzulární logika: $\rightarrow dite(ferda)$
   Prolog:
   ```
   dite(ferda).
   ```
3. „Psi mají čtyři nohy.“
   Klauzulární logika: $pes(X) \rightarrow pocet\_nohou(X, 4)$
   Prolog:
   ```
   pocet_nohou(X,4) :- pes(X).
   ```
4. „Školáci mají v létě prázdniny.“
   Klauzulární logika: $skolak(X), obdobi(leto) \rightarrow ma\_prazdniny(X)$
   Prolog:
   ```
   ma_prazdniny(X) :-
     skolak(X),
     obdobi(leto).
   ```
5. „Děti mají rády sladká jídla.“
   Klauzulární logika: $dite(X), jidlo(Y), chut(Y, sladky) \rightarrow ma\_rad(X, Y)$
   Prolog:
   ```
   ma_rad(X,Y) :-
     dite(X),
     jidlo(Y),
     chut(Y,sladky).
   ```

5.2.2 Ovládání aplikace SWI Prolog a webové aplipace SWISH

Nejdřív se zaměříme na (instalovanou) aplikaci. Po spuštění SWI Prologu se objeví konzola, což je okno, do kterého zadáváme dotazy (včetně požadavků na nápovědu nebo dotazů na dosud načtené klauzule).

V menu nás ze začátku budou zajímat především položky v části *File*, a to *New* (pro vytvoření nového programu), *Edit* (pro otevření existujícího programu v textovém editoru) a *Consult* (pro konzultování – načtení – programu do vnitřní databáze Prologu, jak vidíme na obrázku vpravo.



Obrázek 5.2: Menu *File* ve SWI Prologu

 **Vytvoření programu.** Pokud zvolíme *New* nebo *Edit*, otevře se další okno, což je editor. Na obrázku 5.1 je vidět vpravo dole. Jde o jednoduchý editor, který umí vysvicovat syntaxi Prologu:

- jasně červené jsou predikáty, které jsou použity pouze v klauzulích typu fakt nebo v hlavách klauzulí odpovídajících pravidlům (v predikátové nebo kauzulární logice by byly jen „za implikací“),
- tučné černé jsou predikáty nacházející se v hlavách klauzulí, které se vyskytují i v tělech jiných klauzulí,
- netučné černé jsou predikáty v tělech klauzulí,
- modré jsou operátory a některé vestavěné predikáty (například write/1),
- červenohnědé jsou proměnné,
- tučné tmavé červené jsou chyby,
- zelené jsou komentáře (komentářovým symbolem je procento).

Proč Prolog zabarvuje predikáty v hlavách klauzulí (a ve faktech) některé jasně červenou a jiné černou barvou? Protože ty černé se v jiných klauzulích nacházejí na opačné straně implikace,

a tedy je možné použít je jako „spojovací materiál“ při unifikaci a následné rezoluci, když z takových dvou klauzulí chceme něco odvodit. Atomy s predikáty zabarvenými jasně červenou barvou *nejsou chybné*, jen se Prologu moc nelíbí, že nebude možné využít je pro rezoluci s jinou klauzulí.

✎ **Načtení/konzultování programu.** Pokud už máme program sestaven (například s využitím zmíněného editoru, nebo v jakémkoliv jiném textovém editoru), uložíme soubor s příponou .pl a pak v konzoli zvolíme v menu *File* → *Consult*, v běžném dialogovém okně najdeme soubor a potvrdíme. Alternativně můžeme v konzoli použít predikát `consult`:

```
consult("D:/Prolog/priklady/rodina.pl").
```

Tímto se soubor načte do interní databáze Prologu, Prolog si ho předzpracuje do binární podoby, aby se mu s klauzulemi lépe a rychleji pracovalo. Na výzvu Prologu (prompt, je to dvojznak `?-`, znamená „zadej dotaz“) zadáváme dotazy, Prolo vypisuje odpovědi.

Načtení (přeložení, konzultování) programu je nutné, protože Prolog si program udržuje v interním kódu (databázi) v binární podobě, se kterým se mu pracuje jednodušeji a především rychleji, navíc interní kód bývá obvykle bez syntaktických chyb (kontrola se provádí při načítání a sestavování interní reprezentace programu).

Při každé změně v souboru programu musíme (samozřejmě po uložení těchto změn) program znovu načíst, aby si Prolog mohl tento interní kód obnovit. Znovunačtení je jednodušší, nemusíme soubor hledat, stačí v konzoli napsat:

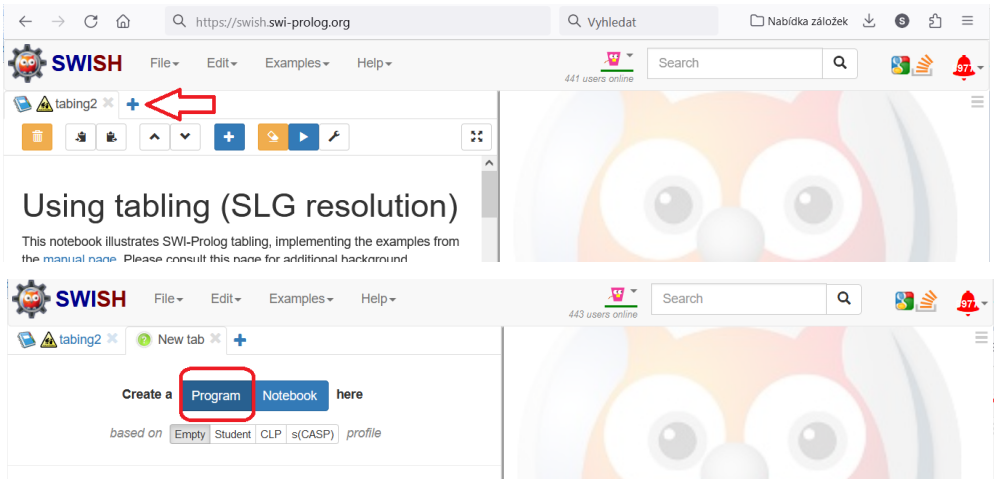
```
make.
```

(nezapomeňte tečku na konci).

V Prologu totiž obvykle vše končí tečkou: fakty, pravidla, dotazy, odpovědi. Všimněte si, že i dotaz (defacto příkaz) `make.` končí tečkou.

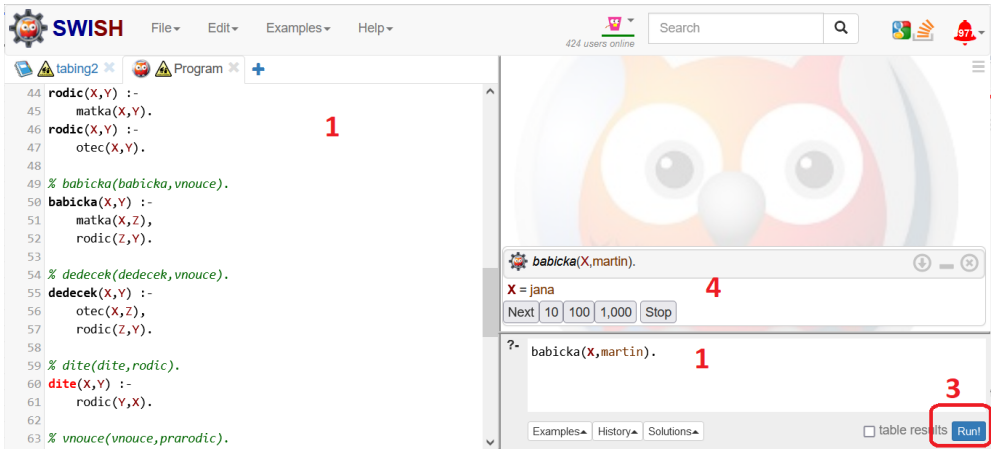
✎ **Zadávání dotazů.** Dotazy píšeme do konzoly, každý dotaz taktéž končí tečkou. Pokud na dotaz Prolog vyhledává více než jednu odpověď, pak je vypisuje jednotlivě a po každé čeká na pobídnutí, jestli chceme další možnou odpověď: pokud má hledat další, stiskneme klávesu ;, pokud ne, tak cokoliv jiného, třeba tečku nebo Enter.

✎ **Používání online varianty SWISH.** Na adrese https://swish.swi-prolog.org/ najdeme online variantu SWI Prologu. Obrázek 5.3 ukazuje prostředí po otevření odkazu a níže po otevření



Obrázek 5.3: Prostředí webové aplikace SWISH

nové záložky pro vlastní program (můžeme mít otevřeno více záložek s různými programy), klepneme na tlačítko Program.



Obrázek 5.4: Program a dotazy ve SWISH

Na obrázku 5.4 je pak situace po napsání nebo vkopírování programu (bod 1). Zde není nutno konzultovat, program je ve stejném okně jako konzola. Zadáme dotaz (vpravo dole, bod 2) a klepneme na tlačítko Run (zcela dole, bod 3). Dotaz je vyhodnocen a výsledek se zobrazí o něco výše (bod 4).

Rozdíl oproti instalované aplikaci je hlavně v tom, že program se zde nekonzultuje a za dotazem se neklepe na Enter, ale na tlačítko Run, a odpovědi na dotaz se zobrazují nahoře, nikoliv pod dotazem. Jinak je to podobné.

Příklad 5.3

Sestavíme program obsahující zadané klauzule.

Zadání programu: Petr má rád květiny, Ivanu a televizi.
Jan má rád jitrnice a televizi.
Věra má ráda všechno, co má rád Jan.

V klauzulární logice:
$\rightarrow ma\_rad(petr, kvetiny)$
$\rightarrow ma\_rad(petr, ivana)$
$\rightarrow ma\_rad(petr, televize)$
$\rightarrow ma\_rad(jan, jitrnice)$
$\rightarrow ma\_rad(jan, televize)$
$ma\_rad(jan, X) \rightarrow ma\_rad(vera, X)$

Program v Prologu:

```
ma_rad(petr,kvetiny).
ma_rad(petr,ivana).
ma_rad(petr,televize).
ma_rad(jan,jitrnice).
ma_rad(jan,televize).
ma_rad(vera,X) :- ma_rad(jan,X).
```

Program, který jsme takto vytvořili, uložíme do souboru s příponou PL. Tento soubor pak načteme (konzultujeme) do Prologu a můžeme zadávat dotazy.

Dotazy mohou obsahovat jeden nebo více atomů, argumenty predikátů mohou být i proměnné. Pokud jsou všechny atomy dotazu bázové, Prolog odpoví pouze `yes` nebo `no`, podle toho, zda klauzule dotazu vyplývá z programu (znalostní báze), jestliže však jsou použity proměnné, Prolog vypíše postupně všechny možné hodnoty proměnné (kombinace proměnných), pro které je dotaz splnitelný.

Příklad 5.4

Po načtení programu z předchozího příkladu zadáme následující dotazy a získáme uvedené odpovědi.

```
?- ma_rad(petr,televize).                          Má rád Petr televizi?
    yes
?- ma_rad(jan,kvetiny).                            Má rád Jan květiny?
    no
?- ma_rad(jan,X).                                  Co má rád Jan?
    X = jitrnice ;
    X = televize ;
    no
?- ma_rad(X,jitrnice).                             Kdo má rád jitrnice?
    X = jan ;
    X = vera ;
    no
?- ma_rad(petr,X),ma_rad(jan,X)                    Co má rád Petr a zároveň Jan?
    X = televize ;
    no
?- ma_rad(Kdo,Co).                                 Kdo má co (koho) rád?
    Kdo = petr , Co = kvetiny ;
    Kdo = petr , Co = ivana ;
    Kdo = petr , Co = televize ;
    Kdo = jan , Co = jitrnice ;
    Kdo = jan , Co = televize ;
    Kdo = vera , Co = jitrnice ;
    Kdo = vera , Co = televize ;
    no
```

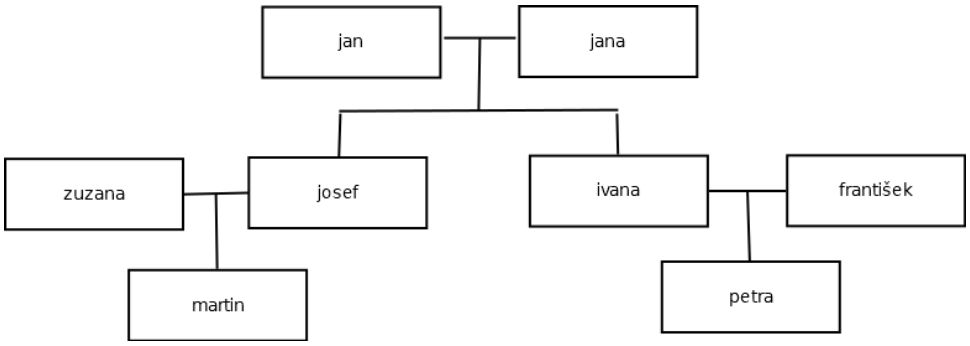
Příklad 5.5

Napíšeme program v Prologu, který ve faktech a pravidlech zachytí rodinnou strukturu, která je znázorněna na obrázku 5.5 (část klauzulí v dále sestaveném programu chybí).

Nejdřív pomocí predikátů `muz/1` a `zena/1` vytvoříme fakty o tom, kdo je muž a kdo žena:

- jan, josef, František a martin jsou muži,
- jana, zuzana, ivana a petra jsou ženy.

```
muz(jan).
zena(jana).
...
```



Obrázek 5.5: Rodinná struktura pro příklad

Připíšeme predikát `manzele/2`, který určí, kdo jsou manželé. Pozor, musí fungovat „obousměrně“, tedy nesmí záležet na tom, jestli jako první parametr uvedeme ženu nebo muže. Rekurzi se vyhneme použitím pomocného predikátu (třeba `manzelepom/2`).

```
manzelepom(jan,jana).
manzelepom(josef,zuzana).
manzelepom(ivana,frantisek).
manzele(X,Y) :-
    manzelepom(X,Y).
manzele(X,Y) :-
    manzelepom(Y,X).
```

Následně dodáme pravidla určující predikáty `manzelka/2` a `manzel/2`.

```
% manzelka(zena,muz).
manzelka(Zena,Muz) :-
    manzele(Zena,Muz),
    zena(Zena),
    muz(Muz).

% manzel(muz,zena).
manzel(Muz,Zena) :-
    manzele(Muz,Zena),
    muz(Muz),
    zena(Zena).
```

Vytvoříme množinu faktů určující, kdo je čí matkou a otcem, a obecně rodičem a dítětem:

```
% matka(matka,dite).
matka(jana,josef).
matka(jana,ivana).
matka(zuzana,martin).
matka(ivana,petra).

% otec(otec,dite).
otec(X,Y) :-
    manzel(X,Z),
    matka(Z,Y).

%rodic(rodic,dite).
rodic(X,Y) :-
    matka(X,Y).
rodic(X,Y) :-
    otec(X,Y).
```

```
% dite(dite,rodic).
dite(X,Y) :-
    rodic(Y,X).
```

Další bude babička, dědeček, vnuk, vnučka a sourozenec, přičemž musíme zajistit, aby nikdo nebyl sourozencem sám sobě:

```
% babicka(babicka,vnouce).
babicka(X,Y) :-
    matka(X,Z),
    rodic(Z,Y).

% dedecek(dedecek,vnouce).
dedecek(X,Y) :-
    otec(X,Z),
    rodic(Z,Y).

vnuk(X,Y) :-
    muz(X),
    babicka(Y,X).
vnucka(X,Y) :-
    zena(X),
    dedecek(Y,X).
vnucka(X,Y) :-
    zena(X),
    babicka(Y,X).
vnouce(X,Y) :-
    vnuk(X,Y).
vnouce(X,Y) :-
    vnucka(X,Y).

% sourozenec(X,Y).
sourozenec(X,Y) :-
    matka(Z,X),
    matka(Z,Y),
    X\=Y.
```

Program konzultujeme přes *File* a pak se můžeme dotazovat. Položíme následující dotazy:

- Jsou Zuzana a Josef manželé?
- Jsou Jan a Ivana manželé?
- Kdo je Martinův dědeček?
- Jak se jmenují Janova vnoučata?

Celá dotazovací komunikace bude vypadat takto (píšeme to, co je **za promptem** „`?-`“):

```
?- manzele(zuzana,josef).
true.

?- manzele(jan,ivana).
false.

?- dedecek(X,martin).
X = jan ;
false.
```

```
?- vnouce(X,jan).
X = martin ;
X = petra ;
false.
```

Zatím jsme v parametrech predikátů používali buď konstanty (začínající malým písmenem) nebo proměnné (začínající velkým písmenem). Do proměnné Prolog postupně doplňuje a vypisuje nalezené možnosti. Ale co když nás ty možnosti ve skutečnosti nezajímají? Můžeme se zeptat třeba takto:

- Má Jan nějaká vnoučata?
- Je Jana něčí babičkou?

V dotazech pak nepoužijeme běžnou proměnnou, ale místo ní tzv. anonymní proměnnou. Anonymní proměnná je vázaná existenčně (v predikátové logice bychom před ní použili symbol zavináče). Tady používáme podtržítko:

```
?- vnouce(_,jan).
true .

?- babicka(jana,_).
true .
```

Úkol

Podle následujících vět sestavte soubor s příponou .PL s klauzulemi:

- Hektor a Lea jsou lvi, Zulejda je zajíc.
- Lvi jsou silní a velcí, mají žlutou barvu.
- Zajíci jsou rychlí a mají hnědou barvu.
- Kdo je silný a velký, je králem zvířat.

Toto zadání je podobné zadání úkolů na straně 80. Dbejte na to, aby opravdu šlo o Hornovy klauzule a nezapomeňte, v jakém pořadí je antecedent/konsekvent v prologovské klauzuli. Nezapomeňte, že konstanty začínají malým písmenem.

.PL soubor uložte a konzultujte (načtěte) do některého Prologu. Dále zadávejte dotazy:

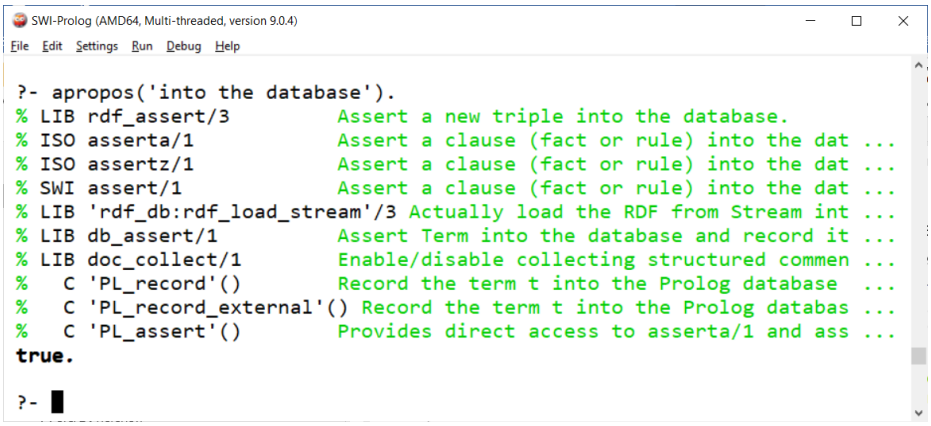
- Je Hektor lev?
- Je Zulejda lev?
- Je Lea králem zvířat?
- Kdo je lev? (tj. chceme, aby Prolog vypsal všechny lvy)
- Kdo je rychlý?
- Kdo je hnědý?
- Kdo má jakou barvu? (tj. chceme uspořádané dvojice [*jméno*, *barva*])

5.2.3 Nápověda

Nápovědu k vestavěným predikátům a dalším mechanismům získáme příkazem `help`, případně pokud neznáme název predikátu, ale známe klíčové slovo z jeho popisu, tak `apropos` (pracujeme v hlavním okně Prologu, tedy tam, kde zadáváme i dotazy).

```
apropos('into the database').
```

Zadaný dotaz žádá o seznam vestavěných predikátů, které ve svém popisu v nápovědě mají vkládání faktů a pravidel do databáze. Na obrázku 5.6 je výstup, kde chceme zjistit, které predikáty (příkazy) slouží k přístupu do databáze.

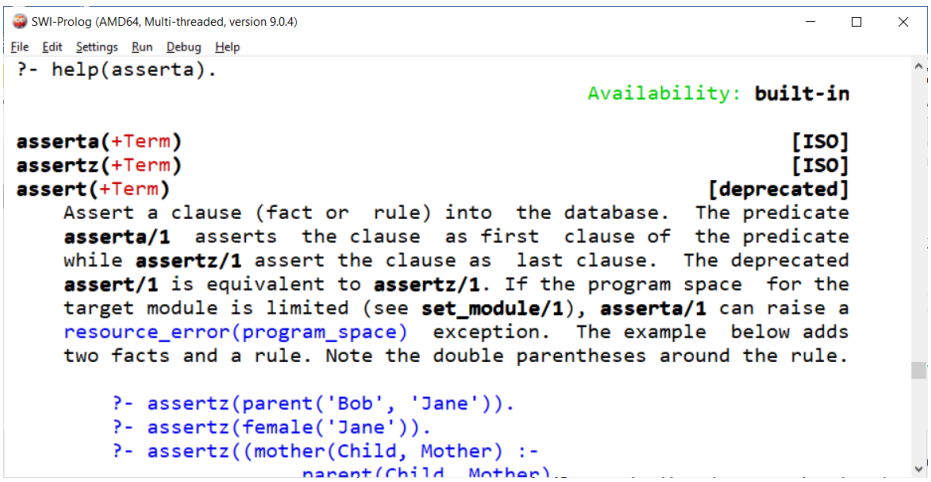


Obrázek 5.6: Nápověda pro případ, že hledám název příkazu

Zajímá nás druhý a třetí nalezený řádek, jejichž popis začíná stejně, tedy použijeme příkaz help, abychom se dozvěděli podrobnosti:

```
help(asseta).
help(assetz).
```

Vlastně bude stačit jeden z nich, protože oba tyto predikáty mají společnou nápovědu:



Obrázek 5.7: Nápověda pro konkrétní predikát/příkaz

Takže je jasné, že takto můžeme vkládat do báze ručně další klauzule, aniž bychom je museli vepisovat do programu. Zatímco `asserta/1` vloží novou klauzuli před již načtené klauzule

pro daný predikát, `assertz/1` vloží novou klauzuli za již načtené klauzule pro daný predikát. Tyto vestavěné predikáty používáme v dotazovacím prostředí, pokud se dodatečně rozhodneme načtenou bázi obohatit.

Úkol

Zjistěte pomocí příkazu `apropos/1`, jestli existuje funkce (predikát) zjišťující podřetězec z daného řetězce (podřetězec se anglicky řekne substring). Dále zjistěte, jestli existuje funkce (predikát) zřetězující, tedy spojující řetězce (zřetězit se anglicky řekne concatenate).

V nápovědě (či referenční příručce a různých manuálech) se u argumentů predikátů či termů používá toto značení – zajímá nás symbol před označením argumentu:

- `+argument` musí být instanciovaný, tedy mít už předem přiřazenu nějakou hodnotu
- `-argument` zde má být proměnná
- `?argument` instanciovaný parametr nebo proměnná
- `@argument` parametr nebude vázán unifikací
- `:argument` jedná se o název predikátu

Příklad 5.6

V předchozím úkolu jsme měli zjistit, jak se nazývá predikát pro nalezení podřetězce. Pokud jsme postupovali správně, zjistili jsme, že to je predikát `sub_string/5` nebo `sub_string/3` (a pár dalších možností). Zobrazíme si k němu nápovědu:

```
?- help(sub_string).
sub_string(+Table, +Sub, +String)
    Succeeds if Sub is a substring of String using the named Table.

                                                    Availability: built-in

sub_string(+String, ?Before, ?Length, ?After, ?SubString)
    This predicate  is functionally  equivalent to  sub_atom/5, but  operates on
    strings. Note that this  implies the  string input  arguments can  be either
    strings or atoms.  If SubString  is unbound  (output) it  is unified  with a
    string. The following  example splits  a string  of the  form <name>=<value>
    into the name part (an atom) and the value (a string).

        name_value(String, Name, Value) :-
            sub_string(String, Before, _, After, "="),
            !,
            sub_atom(String, 0, Before, _, Name),
            sub_string(String, _, After, 0, Value).
```

Jak vidíme, před každým argumentem v závorkách je některý z výše zmíněných znaků. Tento predikát můžeme v dotazu využít třeba takto:

```
?- sub_string("zadanretezec123", 5, 7, Zbytek, Sub).
Zbytek = 3,
Sub = "retezec".
```

První argument je konstantní řetězec v uvozovkách, může být i bez uvozovek (protože začíná malým písmenem a neobsahuje mezery) nebo v apostrofech – fungovalo by to stejně. Musí jít

o konstantu nebo unifikovanou proměnnou, protože v předpisu máme před tímto argumentem znak +.

Druhý argument je pozice, od které chceme získat podřetězec (pozor, počítá se od nuly), následuje počet požadovaných znaků podřetězce, potom kolik má zbývat za podřetězcem, poslední parametr je ten podřetězec. Za poslední dva parametry jsme dosadili proměnnou, což můžeme, to odpovídá symbolu ?.

Kdyby nás nezajímal předposlední parametr, mohli bychom použít anonymní proměnnou:

```
?- sub_string("zadanretezec123", 5, 7, _, Sub).
Sub = "retezec".
```

Ale tento predikát může být použit i jinak než na zjištění podřetězce na dané pozici a v dané délce. Například můžeme vyhledat pozici a délku zadaného podřetězce:

```
?- sub_string("zadanretezec123", Poz, Delka, _, "adan").
Poz = 1,
Delka = 4
```

V tomto případě by byla irelevantní i délka, tedy místo proměnné Delka bychom klidně mohli dát také anonymní proměnnou.

Pokud bychom chtěli najít všechny výskyty určitého podřetězce v daném řetězci, můžeme predikát použít takto:

```
?- sub_string("vstup123 x2123yt", Poz, _, _, "123").
Poz = 5 ;
Poz = 11 ;
false.
```

Prolog najde postupně všechny výskyty, pokud po každém nalezeném budeme poctivě zadávat středník. Když už žádný další nenajde, vypíše false.

Úkol

Vypište si nápovědu k predikátu `between/3`. Zjistěte, jak se dá používat, kde lze použít proměnnou a kde musí být instanciovaná hodnota, vyzkoušejte různé varianty.

5.2.4 Základní práce s databází

V Prologu máme i „manažerské“ nástroje – speciální predikáty či příkazy sloužící pro práci s interní databází. Již dříve byl zmíněn postup konzultování programu přes menu, totéž se dá provést pomocí predikátu `consult/1`, kterému jako parametr dodáme název souboru, ideálně s celou cestou k němu.

Protože obvykle s tímtéž zdrojovým souborem pracujeme opakovaně (hlavně když ladíme program) a potřebujeme ho často načítat, může se hodit také predikát `make/0`, který jednoduše znovu konzultuje veškeré programy, které zatím byly konzultovány. Stačí napsat:

```
make.
```

Dále se nám může hodit například predikát `listing/1` – jako parametr uvedeme predikát, jehož definici chceme vypsat. Například podle předchozího příkladu můžeme napsat:

```
listing(manzele).
```

Tím zjistíme definici predikátu `manzele/2`.

5.2.5 Anonymní proměnná

Existenční termy v Prologu nepoužíváme, místo nich máme k dispozici *anonymní proměnnou* (zapisuje se znakem podtržítka). Pro argument, ve kterém je použita, existuje hodnota, kterou tam lze dosadit, ale tato hodnota nás nezajímá (kdyby nás zajímala, pak bychom použili proměnnou).

Příklad 5.7

Máme následující program:

```
lovi(liska,zajic).                    Liška loví zajíce.
lovi(orel,mys).                       Orel loví myš.
lovi(orel,vrabec).                    Orel loví vrabce.
lovi(honza,ryba).                     Honza loví rybu.
dravec(X) :- lovi(X,_).               Kdo někoho loví, je dravec.
```

Budeme zadávat dotazy:

```
?- dravec(_).                         Existují nějací dravci?
    yes
?- lovi(liska,_).                     Loví někoho liška?
    yes
?- lovi(X,_).                         Kdo někoho loví?
    X = liska ;
    X = orel ;
    X = honza ;
    no
```

Anonymní proměnnou také použijeme místo „běžné“ proměnné, pokud se tato proměnná vyskytuje v těle pravidla pouze jednou.

Obecně platí, že anonymní proměnná *začíná* symbolem podtržítka. To znamená, že za podtržítkem může následovat řetězec písmen a číslic. Pokud do Prologu přepisujeme klauzuli, ve které se některá existenční konstanta vyskytuje více než jednou, měli bychom za podtržítko přidat ještě identifikační řetězec, který zajistí vazbu mezi těmito různými výskyty.

Příklad 5.8

V úkolu na straně 92 jsme vytvořili .PL soubor s klauzulemi o dvou lvech a jednom zajíci. Podle této báze je král zvířat každý, kdo je zároveň silný a velký. Do báze přidáme tyto dvě klauzule:

```
existuje_rychly_kral1 :- rychly(_),kral_zvirat(_).
existuje_rychly_kral2 :- rychly(_1),kral_zvirat(_1).
```

Po uložení a rekonzultování zadáme dotaz „Existuje rychlý král?“:

1. Nejdřív použijeme první přidaný predikát:

   ```
   ?- existuje_rychly_kral1.
       yes
   ```

   Ovšem tato odpověď ve skutečnosti není správná. V definici predikátu jsme použili dvě anonymní proměnné a nedefinovali jsme mezi nimi žádnou vazbu, tedy Prolog pouze zjistil, zda existují hodnoty, které lze na tato místa dosadit, považoval je za různé proměnné.

2. Použijeme druhý přidaný predikát:

```
?- existuje_rychly_kral2.
   no
```

Dostali jsme správnou odpověď, protože přidáním „`1`“ za podtržítko jsme určili vazbu a Prolog do obou míst dosazuje tutéž hodnotu.

Úkol

Vytvořte soubor s bází prologovských klauzulí podle těchto vět:

- Pepa jedl mrkev, Jana jedla zákusek, Pepa pil pivo, Honza pil vodu.
- Každý, kdo něco jedl, je sytý.
- Vše, co někdo jedl, je snědeno.
- Vše, co někdo pil, je vypito.
- Co bylo snědeno *nebo* vypito, se musí koupit.
- Každý, kdo něco jedl *a* pil (obojí zároveň), odchází.

Použijte následující predikáty:

`jedl(`$\langle kdo, co\rangle$`)`	`snedeno(`$\langle co\rangle$`)`	`koupit(`$\langle co\rangle$`)`
`pil(`$\langle kdo, co\rangle$`)`	`vypito(`$\langle co\rangle$`)`	`odchazi(`$\langle kdo\rangle$`)`
`syty(`$\langle kdo\rangle$`)`		

Uložte, konzultujte do Prologu a položte následující dotazy (s využitím stejných predikátů):

- Kdo je sytý?
- Je někdo systý? (chceme odpověď Yes/No)
- Co je třeba koupit?
- Je třeba něco koupit? (chceme odpověď Yes/No)
- Kdo zároveň jedl i pil?
- Kdo odchází?
- Odchází někdo?

5.3 Rezoluce v logickém programování

Rezoluce je základním odvozovacím pravidlem (zároveň se substitucemi), na kterém je založena funkcionalita logických programovacích jazyků. Používá se nepřímá rezoluce, tedy dotaz je znegován a přidán k bázi (programu), odvozovací mechanismus se pak pokouší dostat k prázdné klauzuli znamenající popření znegovaného dotazu.

Projdeme si funkcionalitu odvozovacího mechanismu jazyka Prolog.

5.3.1 Nepřímá rezoluce

Definice 5.2 (Rezoluční uzávěr množiny klauzulí)

Nechť M je množina klauzulí klauzulární logiky. Označíme $\mathcal{R}(M)$ množinu klauzulí, pro kterou platí:

- $M \subseteq \mathcal{R}(M)$ (zařadíme zde všechny klauzule původní množiny).
- Jestliže klauzule C vznikne uplatněním rezolučního odvozovacího pravidla na klauzule C_i a C_j, kde $C_i \in M,\ C_j \in M$, pak $C \in \mathcal{R}(M)$ (na každou unifikovatelnou dvojici klauzulí z M uplatníme rezoluční pravidlo a výslednou rezolventu zařadíme do $\mathcal{R}(M)$).
- Klauzule lze do množiny $\mathcal{R}(M)$ zařadit pouze způsoby z předchozích bodů postupu.

Rezoluční uzávěr množiny klauzulí M n-tého stupně je množina klauzulí $\mathcal{R}_n(M)$ definovaná rekurzívně:

$$\begin{aligned} \mathcal{R}_0(M) &= M, \\ \mathcal{R}_i(M) &= \mathcal{R}\left(\mathcal{R}_{i-1}(M)\right),\ 1 \leq i \leq n \end{aligned} \tag{5.1}$$

Věta 5.1 (Robinsonův rezoluční princip)

Množina klauzulí klauzulární logiky M je nesplnitelná, jestliže existuje přirozené číslo n takové, že množina $\mathcal{R}_n(M)$ obsahuje prázdnou klauzuli $\rightarrow$.

Důkaz: Důkaz provedeme zpětnou matematickou indukcí s převodem do predikátové logiky.

Množinu klauzulí klauzulární logiky lze převést na formuli predikátové logiky v klauzulární normální formě (klauzule převedeme do predikátové logiky a spojíme konjunkcí). Z množiny M klauzulí klauzulární logiky tak sestrojíme ekvivalentní formuli $C(M)$ predikátové logiky.

Již dříve bylo dokázáno, že

- prázdná klauzule má vždy hodnotu $false$ (je kontradiktorická, nepravdivá), přepisujeme ji do predikátové logiky jako $true \rightarrow false$,
- rezoluční pravidlo zachovává pravdivost (je korektní), a to i po převodu do predikátové logiky,
- jestliže množina klauzulí obsahuje nesplnitelnou podmnožinu, je nesplnitelná (mezi klauzulemi je vztah konjunkce), tedy po převodu do predikátové logiky platí vztah $X \,\&\, false \Leftrightarrow false$.

Rezoluční odvozovací pravidlo převedeme do predikátové logiky takto:

$$F_1 \,\&\, (A_1 \,\&\, p \rightarrow K_1) \,\&\, (A_2 \rightarrow p \vee K_2) \,\&\, F_2 \ \vdash\ F_1 \,\&\, (A_1 \,\&\, A_2 \rightarrow K_1 \vee K_2) \,\&\, F_2 \tag{5.2}$$

Již dříve jsme si ukázali, že také toto pravidlo zachovává pravdivost, je přímo odvoditelné z rezolučního odvozovacího pravidla pro predikátovou logiku.

Báze indukce: Množina klauzulí $\mathcal{R}_n(M)$ obsahuje prázdnou klauzuli, proto formule predikátové logiky $C(\mathcal{R}_n(M))$ je nesplnitelná.

Předpoklad indukce: Předpokládejme, že formule $C(\mathcal{R}_i(M)),\ i > 0$ je nesplnitelná.

Krok indukce: Jestliže formule $C(\mathcal{R}_i(M)),\ i > 0$, je nesplnitelná a byla získána z formule $C(\mathcal{R}_{i-1}(M))$ uplatněním pravidla (5.2), pak také $C(\mathcal{R}_{i-1}(M))$ musí být nesplnitelná, protože použité pravidlo zachovává pravdivost. Když krok indukce uplatníme n-krát, zjistíme, že $\mathcal{R}_0(M) = M$ je nesplnitelná. □

Důsledek 5.2

Důsledkem předchozí věty je princip nepřímého rezolučního odvozování *v klauzulární logice, v predikátové logice a také v logických programovacích jazycích – závěr znegujeme, přidáme k množině klauzulí (programu) a dokazujeme, že takto rozšířená množina je nesplnitelná, a pro důkaz nesplnitelnosti stačí, když odvodíme prázdnou klauzuli.*

Účelem logického programování založeného na nepřímém rezolučním odvozování je tedy při uplatňování rezolučního odvozovacího pravidla odvodit prázdnou klauzuli. Pokud volíme postup naznačený výše uvedenou větou (metoda *generování do šířky*, v každém kroku vytvoříme rezolventy pro všechny dvojice klauzulí, které lze unifikovat), je často již pro celkem malé číslo i množina $\mathcal{R}_i(M)$ hodně velká, proto tento postup není moc vhodný.

Jinou možností jsou metody *generování do hloubky*, kdy generujeme pouze ty klauzule, které potřebujeme pro postupné odvození prázdné klauzule.

Metody generování do hloubky mají výhodu větší efektivnosti, ale jejich úspěšnost závisí na vhodné volbě klauzulí, na které uplatníme rezoluční odvozovací pravidlo. Různé metody používají pro tento výběr různé strategie. Nejpoužívanější metodou v logickém programování je *lineární metoda* – rezoluční odvozovací pravidlo uplatňujeme vždy na poslední klauzuli přidanou k důkazu a některou další klauzuli; klauzuli, která je důsledkem uplatnění pravidla, pak použijeme v dalším kroku pro další odvození.

5.3.2 Lineární výpočetní strom

Lineární výpočetní strom klauzule popisuje možnosti průběhu výpočtu dané klauzule (všimněte si, není zde uvedeno, že popisuje samotný výpočet). Výpočet je pak průchod tímto výpočetním stromem.

Definice 5.3 (Lineární výpočetní strom)

Lineární výpočetní strom klauzule pro znalostní bázi je takový strom, kde:

- všechny uzly jsou ohodnoceny klauzulemi,
- kořen je ohodnocen cílovou klauzulí,
- pro všechny uzly stromu platí: jestliže je uzel ohodnocen klauzulí C, pak každý jeho potomek je ohodnocen klauzulí vzniklou uplatněním rezolučního odvozovacího pravidla na klauzuli C a některou klauzuli znalostní báze.

Příklad 5.9

Vytvoříme lineární výpočetní strom pro nepřímý důkaz důkaz klauzule $D \rightarrow$ ze znalostní báze

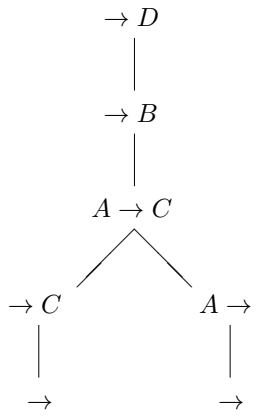
1. $A, B \rightarrow C$
2. $D \rightarrow B$
3. $\rightarrow A$
4. $C \rightarrow$

Při konstrukci nepřímého důkazu budeme vždy rezoluční odvozovací pravidlo používat na poslední klauzuli, kterou jsme přidali k posloupnosti důkazu (v prvním kroku to je klauzule popírající množiny) a některou další klauzuli, tedy budeme postupovat *lineární metodou*.

V tomto příkladu existují dvě takové posloupnosti důkazu:

5.	$\rightarrow D$	PM	5.	$\rightarrow D$	PM
6.	$\rightarrow B$	R(2,5)	6.	$\rightarrow B$	R(2,5)
7.	$A \rightarrow C$	R(1,6)	7.	$A \rightarrow C$	R(1,6)
8.	$\rightarrow C$	R(3,7)	8.	$A \rightarrow$	R(4,7)
9.	$\rightarrow$	R(4,8)	9.	$\rightarrow$	R(3,8)

Výpočetní strom konstruujeme tak, že kořen stromu ohodnotíme první klauzulí popírající množiny a dále podle každé vytvořené důkazní posloupnosti vytvoříme větev stromu. Strom pro uvedený příklad je na obrázku 5.8.



Obrázek 5.8: Výpočetní strom nepřímého důkazu klauzule

Výpočetní strom může být hodně rozvětvený a navíc i nekonečný (některá větev představuje nekonečný výpočet). Účelem je najít prázdnou klauzuli. Hledání můžeme provádět dvěma způsoby:

- *prohledávání do hloubky* – prohledáme nejdřív první větev, pak druhou, ...,
- *prohledávání do šířky* – prohledáváme strom „po patrech“ shora, tedy pracujeme se všemi větvemi stromu najednou.

Výhodou druhé metody je spolehlivost (tj. metoda prohledávání do šířky *je úplná*), pokud někde ve stromě je prázdná klauzule, najdeme ji. Její nevýhodou je však pomalost a výpočetní složitost (musíme si udržovat informace o všech prováděných důkazech zároveň). Tuto nevýhodu řeší

první metoda, její nevýhodou je však riziko možnosti nekonečného výpočtu (v případě, že větev nejvíc vlevo neobsahuje prázdnou klauzuli, například z důvodu zacyklení výpočtu) – tj. metoda prohledávání do hloubky *není úplná*.

Programovací jazyky pro logické programování obvykle používají lineární metodu s prohledáváním do hloubky. Nebezpečí zacyklení výpočtu lze pak předejít vhodným pořadím klauzulí ve znalostní bázi a pořadím atomů v jednotlivých klauzulích. Zacyklení většinou předejdeme, pokud dodržujeme tato pravidla:

1. Ve znalostní bázi *nejdřív uvádíme fakty, pak pravidla*. Protože překladač při prohledávání báze postupuje shora dolů, docílíme tím používání takových unifikací klauzulí pro rezoluci, při kterých nedojde ke zbytečnému opakování výpočtu a tím někdy i k rekurzivnímu vyhodnocování klauzulí.

2. Jestliže je v těle klauzule (v antecedentu) *atom se stejným predikátem jako atom v hlavě klauzule*, třeba i s jinými argumenty, pak takový atom *umístíme až na konec těla klauzule*. Opět tím zamezíme nadbytečnému rekurzivnímu volání klauzule.

Zopakujme si nyní všechna pravidla, která bychom měli dodržovat při sestavování znalostní báze (programu v Prologu):

- předem si promyslíme názvy predikátů a konstant tak, aby byly čitelné pro uživatele, případně můžeme přidávat komentáře,
- pokud se proměnná vyskytuje v klauzuli pouze jednou, použijeme anonymní proměnnou,
- funktory používáme jen tehdy, když je to opravdu nutné a bereme na vědomí, že funktor lze používat spíše jen jako argument predikátu,
- v bázi uvedeme nejdřív fakty a pak pravidla, zvláště v případě, že některá pravidla mají v hlavě shodný predikát s příslušným faktem,
- jestliže se v pravidle vyskytuje rekurze, toto pravidlo uvedeme jako poslední ze všech klauzulí, které mají v hlavě tentýž predikát,
- pokud je v klauzuli atom negovaný predikátem `not`, pak tento atom uvádíme v klauzuli jako poslední,
- můžeme si také všímat nepřímé rekurze, nad tou se zamyslíme v následujícím úkolu.

Tento seznam rozhodně není vyčerpávající. Při používání pokročilejších programovacích technik například můžeme predikáty navrhovat tak, aby jejich argumenty mohly být zároveň vstupní i výstupní, apod. Mnohé nedostatky také zjistíme při ladění programu, kdy do dotazů dosazujeme různé možné hodnoty.

Úkol

Je dána znalostní báze šesti klauzulí klauzulární logiky:

1. $A \rightarrow B$
2. $E \rightarrow D$
3. $B, C \rightarrow D$
4. $\rightarrow C$
5. $\rightarrow A$
6. $A, D \rightarrow E$

Napište dvě různé posloupnosti nepřímého důkazu pro tvrzení $\rightarrow D$ (jde o nepřímý důkaz, nezapomeňte toto tvrzení předem negovat).

Dále vytvořte výpočetní strom pro toto tvrzení s délkami větví nejvýše 5. Zjistěte, která větev je rekurzivní a které klauzule rekurzi (nepřímou) způsobují. Zamyslete se nad tím, jak by bylo vhodné uspořádat klauzule v bázi, aby díky této větvi nedošlo k zacyklení už při hledání prvního řešení (používáme prohledávání do hloubky).

5.4 Průběh výpočtu v Prologu

Prolog je deklarativní jazyk, tedy programátor určuje, *co* se má provést a ne *jak* se to má provést a kam ukládat mezivýsledky výpočtu. Data a program splývají, nerozlišují se. Řízení výpočtu je tedy na Prologu samotném, my mu pouze sdělíme, co má zjistit (odvodit, dokázat, vypsat).

Přesto je užitečné mít alespoň základní přehled o tom, jak výpočet probíhá, a to proto, abychom se dokázali vyhnout zbytečné, třeba i nekonečné rekurzi, optimalizovat program, a také co nejlépe využít prostředky, které nám jazyk nabízí.


Prolog při uplatňování rezoluce používá *lineární metodu s prohledáváním do hloubky*, která byla popsána v předchozí sekci. Aby bylo možné používat rezoluční odvozovací pravidlo, musí být obvykle klauzule upraveny substitucí – unifikací. Pro unifikaci je použit algoritmus podobný algoritmu pro hledání nejobecnějšího unifikátoru, který je uveden v kapitole 3.6.2 na straně 67.

Informace o použitých unifikačních substitucích se ukládají. Protože proměnné v Prologu jsou lokální pro danou klauzuli (dvě stejně pojmenované proměnné v různých klauzulích jsou ve skutečnosti různé proměnné) a globální proměnné neexistují, musí být v údaji o unifikaci explicitně odlišeny stejně pojmenované proměnné z různých klauzulí. Prolog toto zajišťuje připojením čísla klauzule k proměnné, a tím je odstraněno nebezpečí kolize.

Aniž si to většinou uvědomujeme, zadáváme dotaz v negovaném tvaru. Pokud jsou v dotazu použity proměnné, v původním (nenegovaném) tvaru jsou ve skutečnosti vázány existenčně, tedy ptáme se, zda existují nějaké hodnoty proměnných takové, že platí formule dotazu. V predikátové logice můžeme negovaný dotaz vyjádřit takto:

$$\begin{aligned} & \neg\exists u_1 \exists u_2 \ldots \exists u_r (A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_p) \\ \Leftrightarrow \quad & \forall u_1 \forall u_2 \ldots \forall u_r (\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_p) \\ \Leftrightarrow \quad & \forall u_1 \forall u_2 \ldots \forall u_r (true \rightarrow (\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_p)) \end{aligned}$$

Poslední uvedená formule se do klauzulární logiky přepisuje jako

$$A_1, A_2, \ldots, A_p \rightarrow \tag{5.3}$$

Všechny proměnné u_i jsou vázány univerzálně, proto lze na klauzuli uplatňovat rezoluční pravidlo. Pokud jsme na některých místech zadali anonymní proměnné, je s nimi zacházeno jako s volnými proměnnými.

Samotný výpočet je rekurzivní proces, který se provádí tak dlouho, dokud je co počítat (u zacykleného výpočtu teoreticky i do nekonečna, prakticky se výpočet zastaví s chybovým hlášením o přetečení zásobníku).


V každém kroku zpracováváme klauzuli, kterou nazýváme *cílová klauzule*, její atomy nazýváme *cíle*, a hledáme klauzuli takovou, aby bylo možné na ni a na cílovou klauzuli uplatnit

pravidlo rezoluce. Když se nám to podaří, rezolventa (výsledek uplatnění pravidla rezoluce) se stává novou cílovou klauzulí pro další krok výpočtu. Z toho, že v programu jsou pouze Hornovy klauzule (v konsekventu je nejvýše jeden atom), vyplývá, že všechny cílové klauzule, které během výpočtu získáváme, mají prázdný konsekvent (jsou to prologovské klauzule bez hlavy).

 Při výpočtu používáme *zásobník*, do kterého při každém použití unifikace a rezoluce ukládáme údaje o této operaci. Uložíme vždy číslo klauzule, na kterou bylo spolu s cílovou klauzulí uplatněno pravidlo rezoluce, a údaje o unifikační substituci použité pro přípravu na rezoluci, tedy údajem je uspořádaná dvojice $[i, \varphi]$, kde i je číslo klauzule a φ je unifikace.

Údaje se ze zásobníku vyjímají při každém ukončení výpočtu větve (ať úspěšném – `yes` nebo neúspěšném – `no`). Když byl tento údaj do zásobníku uložen, byla unifikace φ a rezoluce použita na klauzuli s číslem i. V případě úspěchu jsme již cíl, ke kterému se takto dalo dostat, zpracovali a potřebujeme najít další cíl, v případě neúspěchu tato cesta zklamala a potřebujeme najít další cestu ke splnění cíle, proto budeme pokračovat následující klauzulí (číslo $i + 1$) s tím, že jako cílovou klauzuli budeme mít tu, která byla cílovou klauzulí před krokem určeným údajem $[i, \varphi]$. Tato klauzule se dá zjistit „zpětným provedením“ operací daných těmito údaji.

Vyjmutí údajů ze zásobníku je vlastně navracení se k předchozímu cíli, tato operace se nazývá *navracení (backtracking)*.

Postup (Výpočet v Prologu)

Protože pracujeme s Hornovými klauzulemi a navíc výpočet začíná u dotazu, který má všechny atomy v antecedentu, je *algoritmus postupu výpočtu* poměrně jednoduchý:

1. Na začátku výpočtu se cílovou klauzulí stane (negovaný) dotaz. Prvním cílem je nejlevější atom této klauzule. Od bodu 2 následuje rekurzivní algoritmus zpracování cílové klauzule.

2. Pokud cílová klauzule není prázdná klauzule, tento bod přeskočíme a pokračujeme bodem 3. Jestliže cílovou klauzulí je prázdná klauzule, *končíme výpočet větve s úspěchem* (`yes`). Jsou dvě možnosti:

   (a) V dotazu jsou proměnné: vypíšeme nalezené hodnoty těchto proměnných (poslední hodnoty ze zásobníku příslušející těmto proměnným), čekáme na stisk klávesy a pokud je to klávesa  , provedeme navracení a pokračujeme bodem 3.

   (b) V dotazu nejsou proměnné: vypíšeme `yes`, ukončíme celý výpočet a smažeme obsah zásobníku.

3. Vezmeme nejlevější atom (cíl) cílové klauzule a hledáme v programu klauzuli, která

   - ještě pro tento cíl nebyla použita,
   - má ve své hlavě (tj. v konsekventu) tentýž predikát jako testovaný cíl
   - a je možné provést unifikaci přes atom v hlavě klauzule a testovaný cíl cílové klauzule.

   Jsou tři možnosti:

   (a) Takovou klauzuli najdeme: pokračujeme bodem 4.

   (b) Takovou klauzuli se nepodaří najít a zásobník není prázdný: provedeme navracení (vše, co bylo až do této pozice v programu provedeno, zrušíme a zkoušíme další cestu) a pokračujeme bodem 3.

(c) Takovou klauzuli se nepodaří najít a zásobník je prázdný: nelze pokračovat jinak, než jak se dosud postupovalo (tj. zásobník je prázdný, ale cílová klauzule je neprázdná), *končíme výpočet s neúspěchem* (vypíšeme `no`).

4. Unifikujeme cílovou klauzuli a nalezenou klauzuli, uplatníme pravidlo rezoluce a rezolventu (výsledek rezoluce) použijeme jako *novou cílovou klauzuli*. Je zřejmé, že nejlevější cíl, který jsme zpracovávali v původní cílové klauzuli, se v nové cílové klauzuli neobjeví, je „odříznut".

   Do zásobníku je uloženo číslo klauzule, která je unifikována s cílem, a údaje o použité substituci. Pokračujeme bodem 2.

Příklad 5.10

Odvození odpovědi na dotaz „Jde Pepa do restaurace?" z uvedeného programu provedeme nejdřív v klauzulární logice a pak v Prologu.

V klauzulární logice:

1. $\rightarrow turista(pepa)$ SA_1
2. $\rightarrow cestuje(pepa, dlouho)$ SA_2
3. $nudi\_se(X), spolecensky(X) \rightarrow jde\_do(X, restaurace)$ SA_3
4. $turista(X), ma\_hlad(X) \rightarrow jde\_do(X, restaurace)$ SA_4
5. $cestuje(X, dlouho) \rightarrow ma\_hlad(X)$ SA_5
6. $jde\_do(pepa, restaurace) \rightarrow$ PM (výchozí cílová klauzule)
7. $nudi\_se(pepa), spolecensky(pepa) \rightarrow jde\_do(pepa, restaurace)$ S(3)$\{pepa/X\}$
8. $nudi\_se(pepa), spolecensky(pepa) \rightarrow$ R(6,7)
   nelze pokračovat, vyjmeme ze zásobníku $[3, pepa/X]$, pokračujeme 4. klauzulí
9. $turista(pepa), ma\_hlad(pepa) \rightarrow jde\_do(pepa, restaurace)$ S(4)$\{pepa/X\}$
10. $turista(pepa), ma\_hlad(pepa) \rightarrow$ R(6,9)
11. $ma\_hlad(pepa) \rightarrow$ R(1,10)
12. $cestuje(pepa, dlouho) \rightarrow ma\_hlad(pepa)$ S(5)$\{pepa/X\}$
13. $cestuje(pepa, dlouho) \rightarrow$ R(11,12)
14. $\rightarrow$ R(2,13)

Všimněte si, že všechny klauzule, které jsme vytvořili uplatněním rezolučního pravidla (od bodu 7), mají prázdný konsekvent.

V Prologu:

```
turista(pepa).                                  (1)
cestuje(pepa,dlouho).                           (2)
jde_do(X,restaurace) :- nudi_se(X),spolecensky(X).   (3)
jde_do(X,restaurace) :- turista(X),ma_hlad(X).       (4)
ma_hlad(X) :- cestuje(X,dlouho).                (5)
?- jde_do(pepa,restaurace).                     (dotaz)
```

Ve skutečnosti bychom museli přidat ještě nějaké klauzule s predikátem `nudi_se` a `spolecensky` v hlavě (není s čím unifikovat), protože jinak by při pokusu o vyhodnocení cíle `jde_do` Prolog vypsal chybové hlášení `Predicate Not Defined`.

Postup výpočtu:

1. Výchozí cílová klauzule je
   ```
   :- jde_do(pepa,restaurace).
   ```
   Hledáme v hlavách klauzulí predikát `jde_do`.

2. Nalezli jsme klauzuli číslo 3, hlava klauzule souhlasí s prvním (momentálně jediným) cílem cílové klauzule. Provedeme unifikaci, výsledkem unifikace jsou klauzule
   ```
   :- jde_do(pepa,restaurace).
   jde_do(pepa,restaurace) :- nudi_se(pepa),spolecensky(pepa).
   ```
   Do zásobníku uložíme `{[3,{pepa/X_3}]`, číslo 3 určuje klauzuli.

   Cílová klauzule (po unifikaci a rezoluci) je
   ```
   :- nudi_se(pepa),spolecensky(pepa).
   ```
   Hledáme v hlavách klauzulí predikát `nudi_se`.

3. Navracení, predikát `nudi_se` není v hlavě žádné unifikovatelné klauzule. Ze zásobníku vyjmeme `[3,{pepa/X_3}]`, tyto údaje pomohou zjistit původní cílovou klauzuli, která se opět stává cílovou klauzulí. Prohledáváme program od klauzule číslo 4 (tj. 3+1).

   Cílová klauzule je
   ```
   :- jde_do(pepa,restaurace).
   ```

4. Nalezli jsme klauzuli číslo 4. Provedeme unifikaci, výsledkem unifikace jsou klauzule
   ```
   :- jde_do(pepa,restaurace).
   jde_do(pepa,restaurace) :- turista(pepa),ma_hlad(pepa).
   ```
   Do zásobníku uložíme `[4,{pepa/X_4}]`.

   Cílová klauzule je
   ```
   :- turista(pepa),ma_hlad(pepa).
   ```
   Hledáme v hlavách klauzulí predikát `turista`.

5. Nalezli jsme klauzuli číslo 1. Unifikace je prázdná množina (není třeba unifikovat, v obou klauzulích jsou pouze konstanty).

   Do zásobníku uložíme `[1,∅]`.

   Cílová klauzule je
   ```
   :- ma_hlad(pepa).
   ```
   Hledáme v hlavách klauzulí predikát `ma_hlad`.

6. Nalezli jsme klauzuli číslo 5. Provedeme unifikaci, výsledkem jsou klauzule
   ```
   :- ma_hlad(pepa).
   ma_hlad(pepa) :- cestuje(pepa,dlouho).
   ```
   Do zásobníku uložíme `[5,{pepa/X_5}]`.

Cílová klauzule je
`:- cestuje(pepa,dlouho).`

Hledáme v hlavách klauzulí predikát `cestuje`.

7. Nalezli jsme klauzuli číslo 2. Unifikace je prázdná množina.

   Do zásobníku uložíme `[2,∅]`.

   Cílová klauzule je
   `:- .`

   Protože je to prázdná klauzule, končíme výpočet větve s výsledkem `yes`. V dotazu nejsou žádné proměnné, proto nemusíme pokračovat dál. Vypíšeme `yes` a vyprázdníme zásobník.

Příklad 5.11

Podle stejné báze provedeme odvození odpovědi na dotaz „Kdo je hladový?“ (tj. tážeme se „Existuje někdo hladový?“ a chceme zároveň vědět, kdo to je).

V klauzulární logice:

1. $\rightarrow turista(pepa)$ SA_1
2. $\rightarrow cestuje(pepa, dlouho)$ SA_2
3. $nudi\_se(X), spolecensky(X) \rightarrow jde\_do(X, restaurace)$ SA_3
4. $turista(X), ma\_hlad(X) \rightarrow jde\_do(X, restaurace)$ SA_4
5. $cestuje(X, dlouho) \rightarrow ma\_hlad(X)$ SA_5
6. $ma\_hlad(X) \rightarrow$ PM (výchozí cílová klauzule)
7. $cestuje(X, dlouho) \rightarrow ma\_hlad(X)$ S(5)$\{X/X\}$ (unifikace)
8. $cestuje(X, dlouho) \rightarrow$ R(6,7)
9. $cestuje(pepa, dlouho) \rightarrow$ S(8)$\{pepa/X\}$ (unifikace s 2)
10. $\rightarrow$ R(2,9)

V Prologu:

```
turista(pepa).
cestuje(pepa,dlouho).
jde_do(X,restaurace) :- nudi_se(X),spolecensky(X).
jde_do(X,restaurace) :- turista(X),ma_hlad(X).
ma_hlad(X) :- cestuje(X,dlouho).

?- ma_hlad(X).
```

Aby Prolog mohl pracovat korektně, ve skutečnosti budeme potřebovat také klauzule, které mají v hlavě predikáty `nudi_se` a `spolecensky`, jako u předchozího příkladu.

Postup výpočtu:

1. Výchozí cílová klauzule je
   `:- ma_hlad(X).`
   Hledáme v hlavách klauzulí predikát `ma_hlad`.

2. Nalezli jsme klauzuli číslo 5. Výsledkem unifikace jsou klauzule
   `:- ma_hlad(X).`
   `ma_hlad(X) :- cestuje(X,dlouho).` (tj. klauzule se nemění)
   Do zásobníku uložíme `[5,{X/X,X_5/X}]`.
   Cílová klauzule je
   `:- cestuje(X,dlouho).`
   Hledáme v hlavách klauzulí predikát `cestuje`.

3. Nalezli jsme klauzuli číslo 2. Výsledkem unifikace jsou klauzule
   `:- cestuje(pepa,dlouho).`
   `cestuje(pepa,dlouho).`
   Do zásobníku uložíme `[2,{pepa/X}]`.
   Cílová klauzule je
   `:- .`

4. Je to prázdná klauzule, proto vypíšeme poslední hodnotu, která byla substituována za X (tj. to co je na zásobníku nejvýše):
   ```
   X = pepa
   ```
   a čekáme na stisk klávesy.

5. Byla stisknuta klávesa ;.

6. Navracení: vyjmeme ze zásobníku poslední uložené údaje – `[2,{pepa/X}]`.
   Cílová klauzule je
   `:- cestuje(X,dlouho).`
   Hledáme v hlavách klauzulí predikát `cestuje`, a to až od 3. klauzule.

7. Navracení, predikát `cestuje` není v hlavě žádné klauzule od klauzule č. 3. Ze zásobníku vyjmeme `[5,{X/X,X_5/X}]`.
   Cílová klauzule je
   `:- ma_hlad(X).`
   Hledáme v hlavách klauzulí predikát `ma_hlad`, a to až od 6. klauzule.

8. V bázi však už šestá klauzule není, navracení nelze provést (zásobník je prázdný), proto končíme výpočet větve s neúspěchem (vypíšeme `no`) a ukončíme celý výpočet.

Během výpočtu byl vygenerován výstup

```
X = pepa ;          (v bodu 4 tohoto postupu)
no                  (v bodu 8 tohoto postupu)
```

S anonymními proměnnými zachází Prolog při unifikaci poněkud volněji, lze je unifikovat s kteroukoliv konstantou a také proměnnou (Prolog si hlídá, aby univerzum diskurzu nebylo prázdné).

Příklad 5.12

Jsou dány tyto prologovské klauzule:

```
vlk(akela).
vyje(X) :- vlk(X).
hluk :- vyje(_).
```

Pokud se dotážeme, zda je hluk (dotaz `hluk.`, cílová klauzule je `:- hluk.`), je třeba unifikovat atom `vyje(_)` s hlavou druhé klauzule, která v příslušném argumentu obsahuje proměnnou. Unifikace nemůže znamenat zobecnění, ale buď konkretizaci nebo zachování původního stupně obecnosti. Tedy výsledkem unifikace nemůže být dosazení proměnné za anonymní proměnnou, ale naopak dosazení anonymní proměnné za proměnnou.

Zjednodušeně chápeme stupnici obecnosti jako *konstanta — anonymní proměnná — proměnná*.

Po uplatnění unifikace a rezoluce získáme cílovou klauzuli `:- vyje(_).`, v tomto kroku je úkolem zjistit, zda někdo vyje, postupujeme podobně jako v předchozím kroku.

Další cílovou klauzulí je `:- vlk(_).`, nyní unifikujeme anonymní proměnnou s konstantou `akela`. Konstanta má menší stupeň obecnosti než anonymní proměnná, proto je výsledkem unifikace dosazení konstanty.

Úkol

Máme tento program v Prologu:

```
lev(hubert).
dite(zita, hubert).
vlk(azor).
selma(X, kockovita) :- lev(X).
selma(X, psovita) :- vlk(X).
lev(X) :- dite(X, Y), lev(Y).
vlk(X) :- dite(X, Y), vlk(Y).
nebezpecny(X) :- selma(X,_).
```

Zjistěte, jak jsou vyhodnocovány dotazy

- `?- lev(X).` (Vyjmenuj všechny lvy.)
- `?- selma(azor, S).` (Jaký typ šelmy je Azor?)
- `?- nebezpecny(zita).` (Je Zita nebezpečná?)

5.5 Řízení výpočtu v Prologu

Prolog má mnoho vestavěných predikátů, z nichž některé slouží k řízení výpočtu. Všechny tyto možnosti jsou podrobněji probírány ve volitelném předmětu *Praktikum z logického programování*.

5.5.1 Predikáty popření, selhání a řezu

Nás budou zajímat především predikáty `fail` a `!`. Predikát `fail` je vždy vyhodnocen jako $false$ (také se nazývá „predikát selhání“), predikát `!` se nazývá *predikát řezu* a slouží k „odříznutí“ dalších možných generovaných řešení (je vyhodnocen jako $true$, tedy uspěje, ale znemožní navracení).

 Predikát `fail` v kombinaci s predikátem řezu je použit například při definování predikátu `not` pro negaci atomu:

```
not(nějaký_atom) :- call(nějaký_atom),!,fail.
not(nějaký_atom).
```

Nejdřív je zavolán cíl `nějaký_atom` pomocí vestavěného predikátu `call/1` (ten jenom zavolá – „spustí“ – svůj argument). Dále se pokračuje podle toho, jakou pravdivostní hodnotu vrátí vyhodnocení `call(nějaký_atom)`:

$true$: díky predikátu `fail` první klauzule vrátí hodnotu $false$ a díky predikátu `!` vyhodnocování predikátu `not(X)` nepokračuje další klauzulí, tedy `not(nějaký_atom)` je vyhodnoceno jako $false$,

$false$: v první klauzuli vyhodnocování nedojde až k predikátu řezu `!` a tedy při navracení pokračujeme k druhé klauzuli, kterou lze s cílem unifikovat vždy a protože je to fakt bez předpokladů, cíl je vyhodnocen jako $true$.

 **Příklad 5.13**

Můžeme vyzkoušet na těch nejjednodušších atomech, zadáme následující dotazy:

```
?- not(true).
false.

?- not(fail).
true.
```

Dále vytvoříme program, ve kterém se bude vyskytovat predikát `not`:

```
strom(jedle).
strom(jablon).
strom(topol).
strom(moruse).

vyska(jedle,8).
vyska(jablon,2).
vyska(topol,10).
vyska(moruse,3).

velkyStrom(X) :- strom(X),vyska(X,Y),Y>5.

malyStrom(X)  :- strom(X),not(velkyStrom(X)).
```

Vyzkoušíme pár dotazů:

```
?- velkyStrom(jedle).
true.
```

```
?- velkyStrom(jablon).
false.

?- velkyStrom(S).
S = jedle ;
S = topol ;
false.

?- malyStrom(S).
S = jablon ;
S = moruse.
```

Program funguje podle očekávání. Všimněte si však odlišného chování Prologu u posledních dvou dotazů: zatímco při výpisu velkých stromů hledal Prolog poctivě všechna řešení a po nenalezení dalšího vypsal `false`, u malých stromů výstup false na konci chybí (není to tím, že by uživatel nestiskl středník, ostatně všimněte si té tečky na konci). Je to z toho důvodu, že predikát `not`, který zde přichází ke slovu, má ve své definici predikát řezu.

Příklad 5.14

Vyjádříme v Prologu větu „Všichni kromě Honzy jsou přítomni.“

```
pritomen(X) :- X=honza,!,fail.
pritomen(X).
```

Vyhodnocování probíhá takto: když chceme zjistit, zda je určitý člověk (například Honza) přítomen, začneme nejdřív první klauzulí (za `X` je dosazen dotyčný zjišťovaný).

Jestliže atom `X=honza` je vyhodnocen jako $true$, pokračujeme dalšími atomy první formule. Atom `!` je vyhodnocen jako $true$, atom `fail` je však vyhodnocen jako $false$, a proto dojde k navracení. Protože však před atomem `fail` je predikát řezu, k navracení nedojde a je vrácena hodnota vrácená atomem `fail`, tedy $false$. Jinými slovy – pokud je argumentem atomu `pritomen(X)` Honza, zjistíme, že není přítomen.

Jestliže však je při unifikaci za `X` dosazeno něco jiného než Honza, pak atom `X=honza` je vyhodnocen jako $false$ a hned dojde k navracení (backtrackingu), které tentokrát není „zamezeno“ predikátem řezu, proto pokračujeme k druhé klauzuli. Ta je vyhodnocena jako $true$, ať už je za její argument dosazeno cokoliv (pokud je co dosazovat), takže celkově dojdeme k závěru, že dotyčný je přítomen.

Tento postup má jednu vadu: predikát `pritomen` je možné použít pro zjištění, zda někdo konkrétní (zadaný konstantou) je přítomen, ale při dosazení proměnné (ptáme se, kdo je přítomen) predikát selže (Prolog vypíše chybové hlášení).

Příklad 5.15

Vyjádříme v Prologu větu „Honza není přítomen.“

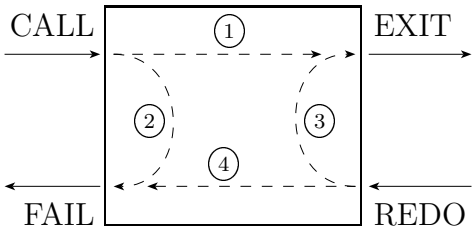
```
pritomen(X) :- X=honza,!,fail.
```

Poznámka:

V Prologu však ve skutečnosti *není třeba explicitně uvádět, že daný subjekt či objekt nemá určitou vlastnost*, protože Prolog pracuje v uzavřeném světě (o čem mu neřekneme, že je pravda, to podle něho není pravda), za určitých okolností však takto můžeme vyřešit některé problémy vyplývající ze způsobu práce Prologu. Proto predikát `not` používáme spíše v podmínkách (jestliže daný objekt či subjekt nemá danou vlastnost...).

5.5.2 Krabičkový model

Zpracování atomů a klauzulí se dá vizualizovat pomocí blokového schématu, kterému se familiárně říká „krabičnový model“ (někteří autoři zůstávají jednoduše u názvu „blokové schéma“, případně bez zdrobnění „krabicový model“).



Obrázek 5.9: Krabičkový model běžného atomu v klauzuli

Na obrázku 5.9 vidíme krabičkový model běžného atomu. Předpokládejme, že se právě vyhodnocuje klauzule s několika atomy, konkrétně jeden z těchto atomů (tedy cíl pro daný krok). Právě se nacházíme ve výpočetním stromě na určitém místě. Cíl je zavolán (vnitřně predikátem `call`) a dál lze pokračovat dvěma způsoby:

① atom je vyhodnocen jako *true*: do zásobníku vložíme informaci o vyhodnocení (klauzule a unifikátor), opouštíme krabičku pro tento atom směrem vpravo (EXIT) a přecházíme do krabičky pro následující atom (např. v cílové klauzuli za čárkou), ve výpočetním stromě pokračujeme dolů,

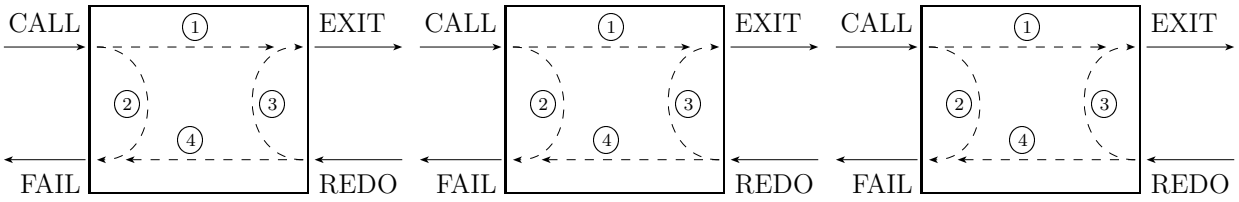
② atom je vyhodnocen jako *false*: opouštíme krabičku směrem vlevo (FAIL), tedy zpět, probíhá backtracking (ve větvi nelze dál pokračovat).

V případě ① tedy přecházíme do „krabičky“ vpravo příslušející dalšímu atomu v pořadí, kde proběhne podobné vyhodnocení jako u této, atd. Někde dál ve větvi dojde k backtrackingu, tedy navracení směrem nahoru (podobně jako by byl případ ②). Pak se do naší krabičky dostaneme znovu, a to zprava (REDO). Co potom? Vyndáme ze zásobníku prvek, který jsme v bodu ① vložili (klauzule a unifikátor) a opět máme dvě možnosti:

③ najdeme další alternativu (unifikaci nebo další klauzuli pro pravidlo rezolučního řezu): vložíme tuto informaci do zásobníku a pokračujeme do další vhodné „krabičky“ vpravo (EXIT), tedy začneme tvořit novou větev,

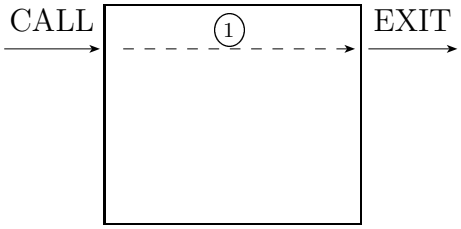
④ v této větvi nelze pokračovat, proto provedeme backtracking směrem vlevo (FAIL), tedy ve stromě nahoru, a hledáme další řešení.

Obrázek 5.10 na straně 112 ukazuje, jak na sebe navazuje zpracování několika následujících atomů (cílů), což odpovídá tomu, že ve výpočetním stromě jdeme v některé větvi směrem dolů (a při navracení pak směrem nahoru, zde do krabiček vlevo).



Obrázek 5.10: Krabičky pro několik po sobě následujících vyhodnocovaných atomů

Jak to však bude vypadat při použití predikátu řezu? To vidíme na obrázku 5.11. Tento predikát vždy uspěje, tedy jdeme přímo ke konci EXIT, a pokud je nějaká „krabička“ (atom) dál vpravo, pokračuje se ve vyhodnocování. Pokud však někde vpravo dojde k backtrackingu, zpět do této „krabičky“ už se nedostaneme, tedy ani není cesta ve výpočetním stromě nahoru, nehledá se další řešení a výpočet skončí.



Obrázek 5.11: Krabičkový model predikátu řezu

Příklad 5.16

Mějme tento jednoduchý program:

```
a :- b, c, !, d, e.
a :- f, g.
```

Zadáme tento dotaz:

```
?- a.
```

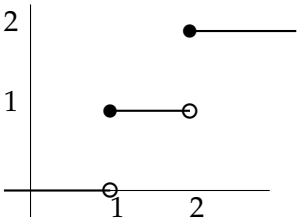
Jak proběhne vyřešení tohoto dotazu? Záleží samozřejmě na hodnotách jednotlivých atomů.

- pokud budou b, c splněny, přecházíme k predikátu řezu, který nás pustí dál vpravo, pokračuje se vyhodnocením d, e (použije se substituce, kterou jsme pro b, c, atd.), jdeme po větvi výpočetního stromu dolů, na konci větve se sice „otočíme“ a začne backtracking, ale když při navracení znovu narazíme na predikát řezu, ukončíme výpočet a nehledáme další řešení pro b, c,
- pokud b, c nebudou splněny, na řez nedojde a není nutno řešit, jestli bude nebo nebude backtracking.

Predikát řezu tedy pustí výpočet směrem vpravo, ale při návratu směrem doleva postaví bariéru (ukončí výpočet, odřízne další větve výpočtu generující další řešení).

Příklad 5.17

Je dána tato funkce:



Naším úkolem je reprezentovat tuto funkci v Prologu.

První řešení:

```
f(X,Y) :- X < 1, Y=0.
f(X,Y) :- 1=<X, X<2, Y=1.
f(X,Y) :- 2=<X, Y=2.
```

Program konzultujeme a položíme několik dotazů:

```
?- f(0,0).
true  .

?- f(0,1).
false.

?- f(0,Y),Y>1.
false.
```

Klauzule jsme sestavili prostě podle tří navzájem disjunktních možností. Program bude fungovat, jen se zbytečně hledají další řešení, když už je jedno správné nalezeno. Projevuje se to u prvního položeného dotazu. Proto použijeme predikáty řezu:

```
f(X,Y) :- X < 1, !, Y=0.
f(X,Y) :- 1=<X, X<2, !, Y=1.
f(X,Y) :- 2=<X, Y=2.
```

Důsledkem je, že po nalezení správného řešení dostaneme první nalezené řešení a další se už nehledá. Třetí klauzuli můžeme zkrátit do formy „když neplatí nic z výše uvedeného, bude platit i toto“, protože k ní se stejně dostaneme jen v případě, že předchozí dvě klauzule neuspěly:

```
f(X,Y) :- X < 1, !, Y=0.
f(X,Y) :- 1=<X, X<2, !, Y=1.
f(_,Y) :- Y=2.
```

Ještě drobná úprava, aby program fungoval optimálněji:

```
f(X,0) :- X<1,!.
f(X,1) :- X>=1,X<2,!.
f(_,2).
```

Rozlišujeme dva druhy řezu:

- zelený řez – nemění deklarativní sémantiku programu, tedy po jeho odstranění dostaneme stejné výsledky (jen odřízneme neúspěšné větve),

- červený řez – zasahuje do sémantiky programu, ovlivňuje vypisované výsledky, po jeho odstranění se vypíšou i jiné výsledky než před odstraněním.

Příklad 5.18

Podívejme se na program v předchozím příkladě. Co se stane, když odstraníme predikáty řezu z programu po první úpravě?

```
f(X,Y) :- X < 1, Y=0.
f(X,Y) :- 1=<X, X<2, Y=1.
f(X,Y) :- 2=<X, Y=2.
```

Program bude podávat stejné výsledky, jen zbytečně procházíme i ty větve, které nedávají správné řešení. Proto jde v obou případech o zelený řez. Ale podívejme se na finální program a odstraňme z něj predikáty řezu:

```
f(X,Y) :- X < 1, !, Y=0.
f(X,Y) :- 1=<X, X<2, !, Y=1.
f(_,Y) :- Y=2.
```

Zadáme několik dotazů:

```
?- f(0,0).
true ;
false.

?- f(0,Y),Y>0.
Y = 2.

?- f(0,Y).
Y = 0 ;
Y = 2.
```

Už výsledek prvního dotazu působí zvláštně. Ovšem u druhého a třetího přímo dostáváme chybné řešení. Pro X=0 má být přece řešení Y=0 a žádné jiné. To znamená, že jsme odstranili červený řez. Kromě správného řešení Prolog pokračoval našel další řešení v jiných větvích, ale ta nejsou správná, přesto výpočet nebyl zastaven.

Zelený řez má za úkol zoptimalizovat výpočet, rozhodně je v programu užitečný. Červenému řezu je lepší se vyhnout, protože za určitých okolností může mít vedlejší nečekané důsledky.

Příklad 5.19

Pokračujme v příkladu: pokud budeme chtít využít pouze zelený řez (a vyhnout se červenému), bude program vypadat takto:

```
f(X,0) :- X<1, !.
f(X,1) :- X>=1, X<2, !.
f(X,2) :- X>=2.
```

Literatura

[1] BEN-ARI, Mordechai. *Mathematical Logic for Computer Science* (second edition). Springer-Verlag (2001). Dostupné také na: https://link.springer.com/book/10.1007/978-1-4471-4129-7

[2] BLACKBURN, Patrick, Johan BOS, Kristina STRIEGNITZ. Learn Prolog Now! [online]. *SWI-Prolog.org* [cit. 2023-03-02]. Dostupné na: https://www.let.rug.nl/bos/lpn/

[3] DUŽÍ, Marie. *Matematická logika* [online]. Skripta VŠB-TU Ostrava [cit. 2009-02-20]. Dostupné na: http://www.cs.vsb.cz/duzi/Matlogika.pdf

[4] Expert Systems in Prolog [online]. *Amzi!* [cit. 2023-12-27]. Dostupné na: http://www.amzi.com/ExpertSystemsInProlog/xsiptop.php

[5] GAHÉR, František. *Logické hádanky, hlavolamy a paradoxy.* Iris Bratislava (1997).

[6] JIRKŮ, Petr. *Logika: Neformální výklad základů formální logiky*. Skriptum VŠE Praha (2000). ISBN 80-245-0054-X.

[7] LUKASOVÁ, Alena. *Formální logika v umělé inteligenci.* Computer Press (2003). ISBN 80-251-0023-5.

[8] MANNA, Zohar. *Matematická teorie programů.* Praha: SNTL (1981). 467 stran.

[9] PAVLÍČEK, Jiří, Peter MIKULECKÝ, Josef HYNEK. *Logické programování a Prolog.* Gaudeamus Hradec Králové (1995). ISBN 80-7041-253-4.

[10] ŠTĚPÁN, Jan. *Logika a logické systémy.* Votobia Olomouc (1992). ISBN 80-85619-29-6.

[11] SUBER, Peter. *Logical Systems*. Skripta Earlham College Richmond, Indiana. Související materiály jsou dostupné na: http://legacy.earlham.edu/~peters/courses/logsys/lshome.htm [cit. 2023-12-27].

[12] SWI Prolog Reference manual [online]. *SWI-Prolog.org* [cit. 2023-03-02]. Dostupné na: https://www.swi-prolog.org/pldoc/doc_for?object=manual

Přílohy

příklady
a jejich řešení

Příloha A

Příklady

V této příloze jsou především příklady na procvičení převodu klauzulí do klauzulární logiky a odvozování v Klauzulárním axiomatickém systému. V některých příkladech je také úkolem převod báze a dotazu do programovacího jazyka Prolog. Řešení příkladů zde není uvedeno, najdeme je v příloze B.

Příklad 1 Vyjádřete v klauzulích klauzulární logiky znalostní bázi a podle ní zjistěte, zda platí „Kdo je okřídlený a má lehkou kostru, létá.“ přímým důkazem a zda platí „Existuje někdo, kdo nemá lehkou kostru.“ nepřímým důkazem.

- Motýl, který odpočívá, je na květině (odpočívající motýl je na květině).
- Motýl je okřídlený a má lehkou kostru, pštros je okřídlený, ale nemá lehkou kostru.
- Kdo je okřídlený a má lehkou kostru, umí létat.
- Ti, kdo umí létat a zrovna nelétají, odpočívají.
- Kdo je okřídlený a je na něčem, potřebuje to, na čem je.
- Existuje někdo, kdo neumí létat.
- Kdo umí létat, je okřídlený (tj. má křídla).
- Zrovna nikdo neodpočívá.

Použijte tyto predikáty: $okridleny(\langle kdo \rangle)$ $odpociva(\langle kdo \rangle)$
$lehka\_kostra(\langle kdo \rangle)$ $je(\langle kdo \rangle, \langle kde \rangle)$
$umi(\langle kdo \rangle, \langle co \rangle)$ $potrebuje(\langle kdo \rangle, \langle co \rangle)$
$leta(\langle kdo \rangle)$

Motýl a pštros pro nás budou konstanty.

Příklad 2 Vyjádřete v klauzulích klauzulární logiky znalostní bázi a odvoďte podle ní odpověď na dotaz „Zlomila si Jana nohu?“ přímým i nepřímým důkazem.

- Jana je dobrá hospodyňka.
- Jana skočila přes plot pro pírko, ale neudělala si bouli.
- Dobrá hospodyňka pro pírko přes plot skočí.

- Kdo skáče přes plot (pro cokoliv), zlomí si nohu nebo si udělá bouli.

Použijte tyto predikáty: $hospodynka(\langle kdo \rangle, \langle jaka \rangle)$
$skace(\langle kdo \rangle, \langle pro\_co \rangle, \langle kam \rangle)$
$zraneni(\langle kdo \rangle, \langle jake \rangle)$

Příklad 3 Vyjádřete v klauzulích klauzulární logiky znalostní bázi a podle ní odvoďte odpověď na dotaz „Bylo nějaké dítě potrestáno zákazem večerníčka?“ nepřímým důkazem.

Znalostní bázi přepište na program v Prologu a pak také do Prologu přepište následující dotazy.

- Mráz kreslí na okno.
- Jana a Pepík jsou děti, Patrik je dospělý.
- Jana kreslí na zeď, ale nebyla potrestána zákazem zákusku.
- Pepík kreslí na papír, Patrik kreslí na zeď.
- Kdo kreslí na zeď, je potrestán.
- Dospělý je potrestán vězením, dítě zákazem večerníčka.
- Dítě, které je potrestáno zákazem večerníčka, pláče.

Dotazy pro vyjádření v Prologu:

- Kdo kam kreslí?
- Kdo pláče?
- Kreslí někdo na zeď?
- Které děti jsou potrestány?
- Jaké tresty mohou být uděleny? (nezajímá nás, kdo byl jak potrestán)

Použijte tyto predikáty: $kresli(\langle kdo \rangle, \langle kam \rangle)$ $potrestan(\langle kdo \rangle)$
$dite(\langle kdo \rangle)$ $trest(\langle kdo \rangle, \langle jaky \rangle)$
$dospely(\langle kdo \rangle)$ $place(\langle kdo \rangle)$

Příklad 4 Vyjádřete v klauzulích klauzulární logiky znalostní bázi a podle ní odvoďte odpověď na dotaz „Pohybuje se někdo rychle?“ nepřímým důkazem.

Znalostní bázi přepište na program v Prologu, pak také do Prologu přepište následující dotazy a vyzkoušejte. Predikáty určete sami podle pravidel, která jste se naučili.
Věty pro znalostní bázi:

- Jerry je myš, Tom a Smoky jsou kočky a Baryk je pes.
- Jerry vidí Toma, Smoky vidí Baryka a Baryk vidí Smokyho.
- Kočky, myši a psi jsou zvířata.
- Žádný pes není kočka.
- Myši se bojí koček, kočky se bojí psů.
- Každý, kdo utíká, se pohybuje rychle.
- Každé zvíře, které vidí někoho, koho se bojí, utíká.
- Pro každého platí: když vidí toho, kdo se ho bojí, honí ho.

- Kdo někoho honí, pohybuje se rychle.

Dotazy pro vyjádření v Prologu:

- Kdo vidí Toma?
- Která zvířata se rychle pohybují?
- Utíká někdo?
- Kdo se koho bojí?
- Kdo koho honí?

Příklad 5 Vyjádřete v klauzulích klauzulární logiky znalostní bázi a odvoďte podle ní odpověď na dotaz „Je javor zelený?“ nepřímým důkazem. Znalostní bázi přepište do Prologu.

- Javor je listnatý strom, borovice a modřín jsou jehličnaté stromy.
- Žádný listnatý strom není jehličnatý, žádný jehličnatý strom není listnatý.
- Listnaté stromy mají v létě listí, jehličnaté stromy mají v létě jehličí.
- Listnaté stromy nemají v zimě listí.
- Borovice má v zimě jehličí.
- Všechno, co má v létě listí nebo jehličí, je zelené.

Berte v úvahu, že Prolog pracuje v uzavřeném světě, tedy některé klauzule není třeba do Prologu přepisovat.

Predikáty je v tomto příkladu možné stanovit více způsoby. Nezapomeňte, že by měly být zvlášť predikáty například pro tvrzení, že strom je listnatý, a tvrzení, že má listí (první tvrzení je obecná vlastnost stromu, druhé se vztahuje obvykle k ročnímu období).

Příklad 6 Vyjádřete v klauzulích klauzulární logiky znalostní bázi (žralok, delfín a kapr jsou konstanty) a podle ní odvoďte odpověď na dotaz „Potřebuje kapr vodu?“ nepřímým důkazem. Znalostní bázi také přepište do Prologu. Stanovte si několik různých dotazů pro Prolog a vyzkoušejte.
Věty pro znalostní bázi:

- Žralok a delfín žijí ve slané vodě, kapr žije ve sladké vodě.
- Všichni, kdo žijí ve vodě (slané i sladké), jsou vodní živočichové.
- Každý vodní živočich potřebuje vodu.
- Někdo žije ve slané vodě.

Příklad 7 Podle znalostní báze v předchozím příkladu odvoďte odpověď na otázku „Existuje někdo, kdo žije ve slané vodě a potřebuje vodu?“ nepřímým důkazem. Při formulování dotazu si můžete pomoci přepisem z predikátové logiky.

Příklad 8 Vyjádřete v klauzulích klauzulární logiky znalostní bázi (jména zvířat jsou konstanty) a podle ní odvoďte odpověď na dotaz „Utekl Ferda?“ nepřímým důkazem. Znalostní bázi přepište na program v Prologu.

- Lišky a zajíci žijí v lese, králíci žijí na dvoře.
- Bystrouška je liška, Ferda je králík a Ušák je zajíc.
- Když se tvor žijící na dvoře dostane do lesa, potká nějakou lišku (jakoukoliv) a neuteče, je sežrán.
- Ferda se dostal do lesa, potkal Bystroušku, ale nebyl sežrán.

Při vytváření znalostní báze dbejte na správnou posloupnost klauzulí v bázi (už v bázi pro klauzulární logiku seřaďte klauzule podle požadavků na bázi Prologu).

Příklad 9 Vyjádřete v klauzulích klauzulární logiky znalostní bázi a podle ní odvoďte odpověď na dotaz „Je Pepa na moři?“ nepřímým důkazem.

- Pepa a Honza jsou námořníci, Rudolf není námořník.
- Někteří námořníci umí plavat.
- Námořníci jsou na lodi, ale ne na řece.
- Loď může být na moři nebo na řece.
- Vztah „být na něčem“ splňuje vlastnost tranzitivity, tedy „každý je také na tom, na čem je to, na čem je“ (například Kdo je na něčem, co je na moři, je také na moři).

Příklad 10 Vyjádřete v klauzulích klauzulární logiky znalostní bázi a podle ní odvoďte odpověď na dotaz „Je zajíc býložravec?“ nepřímým důkazem.

- Liška je šelma.
- Šelmy jsou masožravci.
- Všichni masožravci jsou šelmy.
- Zajíc není šelma, a není ani všežravec.
- Kdo jí rostliny, je býložravec nebo všežravec.
- Kdo není masožravec, jí rostliny. (Každý je buď masožravec, nebo jí rostliny.)
- Existují nějaké šelmy.

Příklad 11 Vyjádřete v klauzulích klauzulární logiky znalostní bázi a podle ní odvoďte odpověď na dotaz „Existuje někdo, kdo dýchá vzduch?“ nepřímým důkazem. Znalostní bázi přepište na program v Prologu.

- Kdo žije ve vodě a nemá žábra, má plíce.
- Kytovci, ryby i obojživelníci žijí ve vodě.
- Kytovci nemají žábra.
- Delfín a vorvaň jsou kytovci, štika je ryba a žába je obojživelník.
- Kdo má plíce, dýchá vzduch.
- Obojživelníci mají žábra i plíce, ryby mají žábra.

Příklad 12 Vyjádřete v klauzulích klauzulární logiky znalostní bázi a podle ní odvoďte odpověď na dotaz „Existuje pták, který dál doskáče?“ nepřímým důkazem.

Znalostní bázi přepište na program v Prologu a pak také do Prologu přepište následující dotazy.

- Skřivan a sova jsou ptáci, ale jen skřivan je ranní ptáče.
- Pepa je člověk a zároveň ranní ptáče.
- Lída je sportovec, vlastní trampolínu.
- Člověk Honza vlastní papouška a trampolínu, ale není sportovec.
- Každý sportovec je člověk.
- Ranní ptáče dál doskáče.
- Sportovec, který vlastní trampolínu, dál doskáče.

Použijte tyto predikáty: $ranni\_ptace(\langle kdo \rangle)$ $clovek(\langle kdo \rangle)$
$doskace(\langle kdo \rangle, \langle kam \rangle)$ $vlastni(\langle kdo \rangle, \langle co \rangle)$
$ptak(\langle kdo \rangle)$ $sportovec(\langle kdo \rangle)$

Dotazy pro vyjádření v Prologu:

- Vyjmenuj všechny lidi.
- Kdo dál doskáče?
- Kdo je ranní ptáče?
- Jsou nějací sportovci?
- Který pták není ranní ptáče?

Příklad 13 Přepište následující věty na program v Prologu a formulujte a vyzkoušejte uvedené dotazy.

- Je -5 stupňů a sněží (budete potřebovat predikát o teplotě a predikát o počasí).
- Lůďa a Honza jsou cestáři.
- Ferda je sněhulák.
- Klapka má místo nosu mrkev.
- Honza právě vypráví pohádky.
- Kdo má mrkev místo nosu nebo uhlíky místo očí, je sněhulák.
- Když je méně stupňů než 0, je mráz.
- Když je mráz a sněží, každý cestář odhrnuje sníh.

(vyprávění pohádek a odhrnování sněhu vyjádřete jedním predikátem určujícím, kdo jakou práci zrovna dělá). Dotazy pro vyjádření v Prologu:

- Jaké je počasí?
- Co zrovna dělají jednotliví cestáři?
- Vyjmenuj všechny sněhuláky.
- Dělá Lůďa něco?
- Co dělá Honza?

Příklad 14 Přepište následující věty na program v Prologu a formulujte uvedené dotazy.

- Micka je kočka, Karlík a Hvězdička jsou papoušci a Houkalka je sova.
- Pepík je člověk.

- Papoušci a sovy jsou ptáci.
- Kočky a lidé jsou savci.
- Žádný pták není savec.
- Zvířaty jsou ptáci a dále savci kromě lidí.
- Ptáci mají peří a zobák.
- Savci kromě lidí mají čtyři nohy.
- Ptáci a lidé mají dvě nohy.
- Kočky mají rády ptáky.
- Pepík má rád všechna zvířata.
- Karlík má rád všechny savce kromě koček.

Dotazy pro vyjádření v Prologu:

- Kdo má dvě nohy?
- Kdo má koho rád?
- Kdo má rád toho, kdo má jeho rád? (tj. u koho je tato vlastnost obousměrná?)
- Kdo má rád někoho dvounohého?

Příklad 15 Přepište následující věty na program v Prologu a formulujte uvedené dotazy.

- Mája a Vilík jsou včelky.
- Mája je pilná, zatímco Vilík je líný.
- Bzučilka je pilná včelka.
- Honza je včelař, do jeho úlu patří Mája a Vilík.
- Pepa je včelař, do jeho úlu patří Bzučilka.
- Neexistuje nikdo, kdo by byl zároveň pilný i líný.
- Včelky umí létat.
- Kdo nasbírá málo medu, je hladový.
- Pilné včelky nasbírají hodně medu, líné včelky nasbírají málo medu.
- Včelař, do jehož úlu patří nějaké hladové včelky, nasbírá málo medu.
- Včelař, který nenasbírá málo medu, nasbírá hodně medu.

Dotazy pro vyjádření v Prologu:

- Kdo nasbírá málo medu?
- Kdo nasbírá hodně medu?
- Kdo je hladový?
- Kdo vlastní úl, do něhož patří Mája?

Poznámka:

V předposlední větě potřebujeme umístit tutéž anonymní proměnnou na dvě místa v jedné klauzuli (obdoba existenční konstanty v klauzulární logice). To se řeší přidáním znaku za podtržítko, tedy anonymní proměnná je proměnná začínající podtržítkem.

Příloha B

Řešení příkladů

V této příloze je řešení příkladů, jejichž zadání najdeme v příloze A. Při řešení nepřímým důkazem je obvykle použita lineární metoda, stejně jako v Prologu.

U některých příkladů je také přepis do Prologu. V tomto přepisu některé klauzule předlohy v klauzulární logice nejsou přepsány, protože narozdíl postupu výpočtu v znalostní bázi klauzulární logiky Prolog pracuje s uzavřeným světem – nemusíme v programu dávat na vědomí, že určitý objekt nějakou vlastnost nemá, stačí, když není uvedeno, že tuto vlastnost má.

Rovněž je zde ukázáno řešení některých problémů, které přináší trochu jiný způsob výpočtu Prologu, zejména striktní tvar Hornových klauzulí a tím i řešení negace v některých případech, kdy Prolog odmítá převzít tvar klauzule podle klauzulární logiky (negace v hlavě klauzule) nebo podává jiné než očekávané výsledky (například nekonečný výpočet).

Řešení 1

Znalostní báze a přímé odvození prvního tvrzení:

1. $odpociva(motyl) \rightarrow je(motyl, kvetina)$ SA_1
2. $\rightarrow okridleny(motyl)$ SA_2
3. $\rightarrow lehka\_kostra(motyl)$ SA_3
4. $\rightarrow okridleny(pstros)$ SA_4
5. $lehka\_kostra(pstros) \rightarrow$ SA_5
6. $okridleny(X), lehka\_kostra(X) \rightarrow umi(X, letat)$ SA_6
7. $umi(X, letat) \rightarrow leta(X), odpociva(X)$ SA_7
8. $okridleny(X), je(X, Y) \rightarrow potrebuje(X, Y)$ SA_8
9. $umi(@c, letat) \rightarrow$ SA_9
10. $umi(X, letat) \rightarrow okridleny(X)$ SA_{10}
11. $odpociva(X) \rightarrow$ SA_{11}

12. $okridleny(X), lehka\_kostra(X) \rightarrow leta(X), odpociva(X)$ R(6,7)
13. $okridleny(X), lehka\_kostra(X) \rightarrow leta(X)$ R(11,12)

Nepřímé odvození druhého tvrzení:

12. $\rightarrow lehka\_kostra(X)$ PM
13. $\rightarrow lehka\_kostra(pstros)$ S(12)$\{pstros/X\}$
14. $\rightarrow$ R(5,13)

V zadaném dotazu byla existenčně vázaná proměnná. Po negaci je již univerzálně vázaná.

Řešení 2

Znalostní báze a přímé odvození:

1. $\rightarrow hospodynka(jana, dobry)$ SA_1
2. $\rightarrow skace(jana, pirko, pres\_plot)$ SA_2
3. $zraneni(jana, boule) \rightarrow$ SA_3
4. $hospodynka(X, dobry) \rightarrow skace(X, pirko, pres\_plot)$ SA_4
5. $skace(X, Y, pres\_plot) \rightarrow zraneni(X, zlom\_noha), zraneni(X, boule)$ SA_5
6. $hospodynka(jana, dobry) \rightarrow skace(jana, pirko, pres\_plot)$ S(4)$\{jana/X\}$
7. $\rightarrow skace(jana, pirko, pres\_plot)$ R(1,6)
8. $skace(jana, pirko, pres\_plot) \rightarrow zraneni(jana, zlom\_noha)$, $zraneni(jana, boule)$ S(5)$\{jana/X, pirko/Y\}$
9. $\rightarrow zraneni(jana, zlom\_noha), zraneni(jana, boule)$ R(7,8)
10. $\rightarrow zraneni(jana, zlom\_noha)$ R(3,9)

Nepřímé odvození (navazujeme na znalostní bázi):

6. $zraneni(jana, zlom\_noha) \rightarrow$ PM (popírající množina)
7. $skace(jana, pirko, pres\_plot) \rightarrow zraneni(jana, zlom\_noha)$, $zraneni(jana, boule)$ S(5)$\{jana/X, pirko/Y\}$
8. $skace(jana, pirko, pres\_plot) \rightarrow zraneni(jana, boule)$ R(6,7)
9. $hospodynka(jana, dobry) \rightarrow skace(jana, pirko, pres\_plot)$ S(4)$\{jana/X\}$
10. $hospodynka(jana, dobry) \rightarrow zraneni(jana, boule)$ R(8,9)
11. $\rightarrow zraneni(jana, boule)$ R(1,10)
12. $\rightarrow$ R(3,11)

Řešení 3

Znalostní báze a nepřímé odvození:

1. $\rightarrow kresli(mraz, okno)$ SA_1
2. $\rightarrow dite(jana)$ SA_2
3. $\rightarrow dite(pepik)$ SA_3
4. $\rightarrow dospely(patrik)$ SA_4
5. $\rightarrow kresli(jana, zed)$ SA_5
6. $\rightarrow kresli(pepik, papir)$ SA_6

7. $\rightarrow kresli(patrik, zed)$ SA_7
8. $kresli(X, zed) \rightarrow potrestan(X)$ SA_8
9. $dospely(X), potrestan(X) \rightarrow trest(X, vezeni)$ SA_9
10. $dite(X), potrestan(X) \rightarrow trest(X, zakaz\_vecernicka)$ SA_{10}
11. $dite(X), trest(X, zakaz\_vecernicka) \rightarrow place(X)$ SA_{11}
12. $dite(X), trest(X, zakaz\_vecernicka) \rightarrow$ PM
13. $dite(jana), trest(jana, zakaz\_vecernicka) \rightarrow$ S(12)$\{jana/X\}$
14. $trest(jana, zakaz\_vecernicka) \rightarrow$ R(2,13)
15. $dite(jana), potrestan(jana) \rightarrow trest(jana, zakaz\_vecernicka)$ S(10)$\{jana/X\}$
16. $dite(jana), potrestan(jana) \rightarrow$ R(14,15)
17. $potrestan(jana) \rightarrow$ R(2,16)
18. $kresli(jana, zed) \rightarrow potrestan(jana)$ S(8)$\{jana/X\}$
19. $kresli(jana, zed) \rightarrow$ R(17,18)
20. $\rightarrow$ R(5,19)

Pokud nebudeme důsledně používat lineární metodu, bude pro tento příklad řešení kratší:

12. $dite(X), trest(X, zakaz\_vecernicka) \rightarrow$ PM
13. $dite(X), potrestan(X) \rightarrow$ R(10,12)
14. $dite(X), kresli(X, zed) \rightarrow$ R(8,13)
15. $dite(jana), kresli(jana, zed) \rightarrow$ S(14)$\{jana/X\}$
16. $kresli(jana, zed) \rightarrow$ R(2,15)
17. $\rightarrow$ R(5,16)

Program v Prologu:

```
kresli(mraz,okno).
dite(jana).
dite(pepik).
dospely(patrik).
kresli(jana,zed).
kresli(pepik,papir).
kresli(patrik,zed).
potrestan(X) :- kresli(X,zed).
trest(X,vezeni) :- dospely(X),potrestan(X).
trest(X,zakaz_vecernicka) :- dite(X),potrestan(X).
place(X) :- dite(X),trest(X,zakaz_vecernicka).
```

Dotazy:

```
?- kresli(X,Y).
?- place(X).
```

```
?- kresli(_,zed).
?- dite(X),potrestan(X).
?- trest(_,Trest).
```

Řešení 4

Predikáty: $mys(\langle kdo \rangle)$ $honi(\langle kdo \rangle, \langle koho \rangle)$ $boji\_se(\langle kdo \rangle, \langle koho \rangle)$
$kocka(\langle kdo \rangle)$ $vidi(\langle kdo \rangle, \langle koho \rangle)$ $pohybuje\_se(\langle kdo \rangle, \langle jak \rangle)$
$pes(\langle kdo \rangle)$ $zvire(\langle kdo \rangle)$ $utika(\langle kdo \rangle)$

Znalostní báze a nepřímé odvození:

1. $\rightarrow mys(jerry)$ SA_1
2. $\rightarrow kocka(tom)$ SA_2
3. $\rightarrow kocka(smoky)$ SA_3
4. $\rightarrow pes(baryk)$ SA_4
5. $\rightarrow vidi(jerry, tom)$ SA_5
6. $\rightarrow vidi(smoky, baryk)$ SA_6
7. $\rightarrow vidi(baryk, smoky)$ SA_7
8. $kocka(X) \rightarrow zvire(X)$ SA_8
9. $mys(X) \rightarrow zvire(X)$ SA_9
10. $pes(X) \rightarrow zvire(X)$ SA_{10}
11. $pes(X), kocka(X) \rightarrow$ SA_{11}
12. $mys(X), kocka(Y) \rightarrow boji\_se(X, Y)$ SA_{12}
13. $kocka(X), pes(Y) \rightarrow boji\_se(X, Y)$ SA_{13}
14. $utika(X) \rightarrow pohybuje\_se(X, rychle)$ SA_{14}
15. $boji\_se(X, Y), vidi(X, Y), zvire(X) \rightarrow utika(X)$ SA_{15}
16. $boji\_se(Y, X), vidi(X, Y) \rightarrow honi(X, Y)$ SA_{16}
17. $honi(X, @c) \rightarrow pohybuje\_se(X, rychle)$ SA_{17}
18. $pohybuje\_se(X, rychle) \rightarrow$ PM
19. $utika(X) \rightarrow$ R(14,18)
20. $boji\_se(X, Y), vidi(X, Y), zvire(X) \rightarrow$ R(15,19)
21. $vidi(X, Y), zvire(X), mys(X), kocka(Y) \rightarrow$ R(12,20)
22. $vidi(jerry, tom), zvire(jerry), mys(jerry), kocka(tom) \rightarrow$ S(21)$\{jerry/X, tom/Y\}$
23. $zvire(jerry), mys(jerry), kocka(tom) \rightarrow$ R(5,22)
24. $mys(jerry) \rightarrow zvire(jerry)$ S(4)$\{jerry/X\}$
25. $mys(jerry), kocka(tom) \rightarrow$ R(23,24)
26. $kocka(tom) \rightarrow$ R(1,25)
27. $\rightarrow$ R(2,26)

Program v Prologu:

```
mys(jerry).                                              (1)
kocka(tom).                                              (2)
kocka(smoky).                                            (3)
pes(baryk).                                              (4)
vidi(jerry,tom).                                         (5)
vidi(smoky,baryk).                                       (6)
vidi(baryk,smoky).                                       (7)
zvire(X) :- kocka(X).                                    (8)
zvire(X) :- mys(X).                                      (9)
zvire(X) :- pes(X).                                      (10)
boji_se(Kdo,Koho) :- mys(Kdo),kocka(Koho).               (12)
boji_se(Kdo,Koho) :- kocka(Kdo),pes(Koho).               (13)
pohybuje_se(X,rychle) :- utika(X).                       (14)
utika(X) :- boji_se(X,Y),vidi(X,Y),zvire(X).             (15)
honi(X,Y) :- boji_se(Y,X),vidi(X,Y).                     (16)
pohybuje_se(X,rychle) :- honi(X,_).                      (17)
```

Čísla vpravo nejsou ve skutečnosti součástí souboru, uvádíme je zde pouze pro orientaci. Klauzuli číslo 11 nepřepisujeme, protože Prolog pracuje v uzavřeném světě.

Dotazy:

```
?- vidi(X,tom).
?- zvire(X),pohybuje_se(X,rychle).
?- utika(_).
?- boji_se(Kdo,Koho).
?- honi(Kdo,Koho).
```

Řešení 5

Predikáty: $strom(\langle strom\rangle, \langle jaky\rangle)$ $\quad ma\_listi(\langle co\rangle, \langle kdy\rangle)$
$barva(\langle co\rangle, \langle barva\rangle)$ $\quad ma\_jehlici(\langle co\rangle, \langle kdy\rangle)$

Znalostní báze a nepřímé odvození:

1. $\rightarrow strom(javor, listnaty)$ SA_1
2. $\rightarrow strom(borovice, jehlicnaty)$ SA_2
3. $\rightarrow strom(modrin, jehlicnaty)$ SA_3
4. $strom(X, listnaty), strom(X, jehlicnaty) \rightarrow$ SA_4
5. $strom(X, listnaty) \rightarrow ma\_listi(X, leto)$ SA_5
6. $strom(X, jehlicnaty) \rightarrow ma\_jehlici(X, leto)$ SA_6
7. $strom(X, listnaty), ma\_listi(X, zima) \rightarrow$ SA_7
8. $\rightarrow ma\_jehlici(borovice, zima)$ SA_8
9. $ma\_listi(X, leto) \rightarrow barva(X, zelena)$ SA_9
10. $ma\_jehlici(X, leto) \rightarrow barva(X, zelena)$ SA_{10}

11. $barva(javor, zelena) \rightarrow$ PM

12. $ma\_listi(javor, leto) \rightarrow barva(javor, zelena)$ S(9)$\{javor/X\}$
13. $ma\_listi(javor, leto) \rightarrow$ R(11,12)
14. $strom(javor, listnaty) \rightarrow ma\_listi(javor, leto)$ S(5)$\{javor/X\}$
15. $strom(javor, listnaty) \rightarrow$ R(13,14)
16. $\rightarrow$ R(1,15)

Přepis znalostní báze na program v Prologu:

```
strom(javor,listnaty).                          (1)
strom(borovice,jehlicnaty).                     (2)
strom(modrin,jehlicnaty).                       (3)
ma_listi(X,leto) :- strom(X,listnaty).          (5)
ma_jehlici(X,leto) :- strom(X,jehlicnaty).      (6)
ma_jehlici(borovice,zima).                      (8)
barva(X,zelena) :- ma_listi(X,leto).            (9)
barva(X,zelena) :- ma_jehlici(X,leto).          (10)
```

Klauzule číslo 4 a 7 nemusíme přepisovat, Prolog pracuje v uzavřeném světě.

Řešení 6

Predikáty: $zije\_ve\_vode(\langle kdo\rangle, \langle jaka\rangle)$
$vodni\_zivocich(\langle kdo\rangle)$
$potrebuje(\langle kdo\rangle, \langle co\rangle)$

Znalostní báze a nepřímé odvození:

1. $\rightarrow zije\_ve\_vode(zralok, slany)$ SA_1
2. $\rightarrow zije\_ve\_vode(delfin, slany)$ SA_2
3. $\rightarrow zije\_ve\_vode(kapr, sladky)$ SA_3
4. $zije\_ve\_vode(X, slany) \rightarrow vodni\_zivocich(X)$ SA_4
5. $zije\_ve\_vode(X, sladky) \rightarrow vodni\_zivocich(X)$ SA_5
6. $vodni\_zivocich(X) \rightarrow potrebuje(X, voda)$ SA_6
7. $\rightarrow zije\_ve\_vode(@c, slany)$ SA_7
8. $potrebuje(kapr, voda) \rightarrow$ PM
9. $vodni\_zivocich(kapr) \rightarrow potrebuje(kapr, voda)$ S(6)$\{kapr/X\}$
10. $vodni\_zivocich(kapr) \rightarrow$ R(8,9)
11. $zije\_ve\_vode(kapr, sladky) \rightarrow vodni\_zivocich(kapr)$ S(5)$\{kapr/X\}$
12. $zije\_ve\_vode(kapr, sladky) \rightarrow$ R(10,11)
13. $\rightarrow$ R(3,12)

Přepis znalostní báze na program v Prologu:

```
zije_ve_vode(zralok,slany).                     (1)
zije_ve_vode(delfin,slany).                     (2)
zije_ve_vode(kapr,sladky).                      (3)
vodni_zivocich(X) :- zije_ve_vode(X,slany).     (4)
vodni_zivocich(X) :- zije_ve_vode(X,sladky).    (5)
potrebuje(X,voda) :- vodni_zivocich(X).         (6)
```

Sedmou klauzuli již nemusíme přepisovat do Prologu, protože vyplývá z první a druhé klauzule.

Řešení 7 Popírající množinu pro dotaz vytvoříme přepisem negace závěru z predikátové logiky:
$\neg\exists x(zije\_ve\_vode(x, slany)\ \&\ potrebuje(x, voda)) \Leftrightarrow$
$\Leftrightarrow \forall x(\neg zije\_ve\_vode(x, slany) \vee \neg potrebuje(x, voda))$

Nepřímé odvození:

8.	$zije\_ve\_vode(X, slany), potrebuje(X, voda) \rightarrow$	PM
9.	$zije\_ve\_vode(zralok, slany), potrebuje(zralok, voda) \rightarrow$	S(8)$\{zralok/X\}$
10.	$potrebuje(zralok, voda) \rightarrow$	R(1,9)
11.	$vodni\_zivocich(zralok) \rightarrow potrebuje(zralok, voda)$	S(6)$\{zralok/X\}$
12.	$vodni\_zivocich(zralok) \rightarrow$	R(10,11)
13.	$zije\_ve\_vode(zralok, slany) \rightarrow vodni\_zivocich(zralok)$	S(4)$\{zralok/X\}$
14.	$zije\_ve\_vode(zralok, slany) \rightarrow$	R(12,13)
15.	$\rightarrow$	R(1,14)

Pokud nebudeme používat důsledně lineární metodu:

8.	$zije\_ve\_vode(X, slany), potrebuje(X, voda) \rightarrow$	PM
9.	$vodni\_zivocich(X), zije\_ve\_vode(X, slany) \rightarrow$	R(6,8) (unifikace)
10.	$zije\_ve\_vode(X, slany), zije\_ve\_vode(X, slany) \rightarrow$	R(1,9)
11.	$zije\_ve\_vode(X, slany) \rightarrow$	KK(10)
12.	$zije\_ve\_vode(@c, slany) \rightarrow$	S(11)$\{@c/X; 1\}$
13.	$\rightarrow$	R(7,12)

Krok 12 důkazu si můžeme dovolit, protože množina univerza diskurzu je zjevně neprázdná, obsahuje prvky $zralok, delfin, kapr, voda, slany, sladky$.

Řešení 8

Predikáty:

$zije\_kde(\langle kdo\rangle, \langle kde\rangle)$	$sezran(\langle kdo\rangle)$	$zajic(\langle kdo\rangle)$
$je\_v\_lese(\langle kdo\rangle)$	$utika(\langle kdo\rangle)$	$kralik(\langle kdo\rangle)$
$potka(\langle kdo\rangle, \langle koho\rangle)$	$liska(\langle kdo\rangle)$	

Znalostní báze a nepřímé odvození:

1.	$\rightarrow liska(bystrouska)$	SA_1
2.	$\rightarrow kralik(ferda)$	SA_2
3.	$\rightarrow zajic(usak)$	SA_3
4.	$\rightarrow je\_v\_lese(ferda)$	SA_4
5.	$\rightarrow potka(ferda, bystrouska)$	SA_5
6.	$sezran(ferda) \rightarrow$	SA_6
7.	$liska(X) \rightarrow zije\_kde(X, les)$	SA_7

8. $zajic(X) \rightarrow zije\_kde(X, les)$ SA_8
9. $kralik(X) \rightarrow zije\_kde(X, dvur)$ SA_9
10. $zije\_kde(X, dvur), je\_v\_lese(X), potka(X, Y),$
    $liska(Y) \rightarrow sezran(X), utika(X)$ SA_{10}
11. $utika(ferda) \rightarrow$ PM
12. $zije\_kde(ferda, dvur), je\_v\_lese(ferda), potka(ferda, Y),$
    $liska(Y) \rightarrow sezran(ferda), utika(ferda)$ S(10)$\{ferda/X\}$
13. $zije\_kde(ferda, dvur), je\_v\_lese(ferda), potka(ferda, Y),$
    $liska(Y) \rightarrow sezran(ferda)$ R(11,12)
14. $kralik(ferda) \rightarrow zije\_kde(ferda, dvur)$ S(9)$\{ferda/X\}$
15. $kralik(ferda), je\_v\_lese(ferda), potka(ferda, Y),$
    $liska(Y) \rightarrow sezran(ferda)$ R(13,14)
16. $je\_v\_lese(ferda), potka(ferda, Y), liska(Y) \rightarrow sezran(ferda)$ R(2,15)
17. $potka(ferda, Y), liska(Y) \rightarrow sezran(ferda)$ R(4,16)
18. $potka(ferda, bystrouska),$
    $liska(bystrouska) \rightarrow sezran(ferda)$ S(17)$\{bystrouska/Y\}$
19. $liska(bystrouska) \rightarrow sezran(ferda)$ R(5,18)
20. $\rightarrow sezran(ferda)$ R(1,19)
21. $\rightarrow$ R(6,20)

Přepis znalostní báze na program v Prologu:

```
liska(bystrouska).                                    (1)
kralik(ferda).                                        (2)
zajic(usak).                                          (3)
je_v_lese(ferda).                                     (4)
potka(ferda,bystrouska).                              (5)
sezran(X) :- X=ferda,!,fail.                          (6)
zije_kde(X,les) :- liska(X).                          (7)
zije_kde(X,les) :- zajic(X).                          (8)
zije_kde(X,dvur) :- kralik(X).                        (9)
utika(X) :-
       zije_kde(X,dvur),je_v_lese(X),
       potka(X,Y),liska(Y),
       not(sezran(X)).                                (10)
```

Poznámka:

Při přepisu negace v klauzuli č. 6 byl použit postup z kapitoly 5.5 na straně 108. Tato klauzule však obvkle není nutná, Prolog pracuje v uzavřeném světě. Zde jsme ji zařadili jen proto, že každý použitý predikát (včetně predikátu `sezran`) se musí vyskytovat v nejméně jedné hlavě některé klauzule.

Řešení 9

Predikáty: $namornik(\langle kdo \rangle)$
$umi(\langle kdo \rangle, \langle co \rangle)$
$je\_kde(\langle kdo - co \rangle, \langle kde \rangle)$

Znalostní báze a nepřímé odvození:

1. $\rightarrow namornik(pepa)$ SA_1
2. $\rightarrow namornik(honza)$ SA_2
3. $namornik(rudolf) \rightarrow$ SA_3
4. $namornik(@c) \rightarrow umi(@c, plavat)$ SA_4
5. $namornik(X) \rightarrow je\_kde(X, lod)$ SA_5
6. $namornik(X), je\_kde(X, reka) \rightarrow$ SA_6
7. $\rightarrow je\_kde(lod, more), je\_kde(lod, reka)$ SA_7
8. $je\_kde(X, Y), je\_kde(Y, Z) \rightarrow je\_kde(X, Z)$ SA_8
9. $je\_kde(pepa, more) \rightarrow$ PM
10. $je\_kde(pepa, Y), je\_kde(Y, more) \rightarrow je\_kde(pepa, more)$ S(8)$\{pepa/X, more/Z\}$
11. $je\_kde(pepa, Y), je\_kde(Y, more) \rightarrow$ R(9,10)
12. $namornik(pepa) \rightarrow je\_kde(pepa, lod)$ S(5)$\{pepa/X\}$
13. $je\_kde(pepa, lod), je\_kde(lod, more) \rightarrow$ S(11)$\{lod/Y\}$
14. $namornik(pepa), je\_kde(lod, more) \rightarrow$ R(12,13)
15. $je\_kde(lod, more) \rightarrow$ R(1,14)
16. $\rightarrow je\_kde(lod, reka)$ R(7,15)
17. $je\_kde(X, lod), je\_kde(lod, reka) \rightarrow je\_kde(X, reka)$ S(8)$\{lod/Y, reka/Z\}$
18. $je\_kde(X, lod) \rightarrow je\_kde(X, reka)$ R(16,17)
19. $namornik(X) \rightarrow je\_kde(X, reka)$ R(5,18)
20. $namornik(pepa) \rightarrow je\_kde(pepa, reka)$ S(19)$\{pepa/X\}$
21. $\rightarrow je\_kde(pepa, reka)$ R(1,20)
22. $namornik(pepa), je\_kde(pepa, reka) \rightarrow$ S(6)$\{pepa/X\}$
23. $namornik(pepa) \rightarrow$ R(21,22)
24. $\rightarrow$ R(1,23)

Řešení 10

Predikáty: $bylozravec(\langle kdo \rangle)$ $selma(\langle kdo \rangle)$
$vsezravec(\langle kdo \rangle)$ $ji(\langle kdo \rangle, \langle co \rangle)$
$masozravec(\langle kdo \rangle)$

Znalostní báze a nepřímé odvození:

1. $\rightarrow selma(liska)$ SA_1
2. $selma(zajic) \rightarrow$ SA_2
3. $vsezravec(zajic) \rightarrow$ SA_3
4. $selma(X) \rightarrow masozravec(X)$ SA_4
5. $masozravec(X) \rightarrow selma(X)$ SA_5
6. $ji(X, rostliny) \rightarrow bylozravec(X), vsezravec(X)$ SA_6
7. $\rightarrow masozravec(X), ji(X, rostliny)$ SA_7
8. $\rightarrow selma(@c)$ SA_8
9. $bylozravec(zajic) \rightarrow$ PM
10. $ji(zajic, rostliny) \rightarrow bylozravec(zajic), vsezravec(zajic)$ S(6)$\{zajic/X\}$
11. $ji(zajic, rostliny) \rightarrow vsezravec(zajic)$ R(9,10)
12. $\rightarrow masozravec(zajic), ji(zajic, rostliny)$ S(7)$\{zajic/X\}$
13. $\rightarrow masozravec(zajic)$ R(11,12)
14. $selma(zajic) \rightarrow masozravec(zajic)$ S(4)$\{zajic/X\}$
15. $selma(zajic) \rightarrow$ R(13,14)
16. $\rightarrow$ R(2,15)

Řešení 11

Predikáty: $zije(\langle kdo \rangle, \langle kde \rangle)$ $kytovec(\langle kdo \rangle)$
$ma(\langle kdo \rangle, \langle co \rangle)$ $ryba(\langle kdo \rangle)$
$dycha(\langle kdo \rangle, \langle co \rangle)$ $obojzivelnik(\langle kdo \rangle)$

Znalostní báze a nepřímé odvození:

1. $\rightarrow kytovec(delfin)$ SA_1
2. $\rightarrow kytovec(vorvan)$ SA_2
3. $\rightarrow ryba(stika)$ SA_3
4. $\rightarrow obojzivelnik(zaba)$ SA_4
5. $kytovec(X) \rightarrow zije(X, voda)$ SA_5
6. $ryba(X) \rightarrow zije(X, voda)$ SA_6
7. $obojzivelnik(X) \rightarrow zije(X, voda)$ SA_7
8. $kytovec(X), ma(X, zabra) \rightarrow$ SA_8
9. $zije(X, voda) \rightarrow ma(X, zabra), ma(X, plice)$ SA_9
10. $ma(X, plice) \rightarrow dycha(X, vzduch)$ SA_{10}
11. $obojzivelnik(X) \rightarrow ma(X, zabra)$ SA_{11}
12. $obojzivelnik(X) \rightarrow ma(X, plice)$ SA_{12}
13. $ryba(X) \rightarrow ma(X, zabra)$ SA_{13}

14. $dycha(X, vzduch) \rightarrow$ PM

15. $ma(X, plice) \rightarrow$ R(10,14)
16. $zije(X, voda) \rightarrow ma(X, zabra)$ R(9,15)
17. $kytovec(X) \rightarrow ma(X, zabra)$ R(5,16)
18. $kytovec(delfin) \rightarrow ma(delfin, zabra)$ S(17)$\{delfin/X\}$
19. $\rightarrow ma(delfin, zabra)$ R(1,18)
20. $kytovec(delfin), ma(delfin, zabra) \rightarrow$ S(8)$\{delfin/X\}$
21. $kytovec(delfin) \rightarrow$ R(19,20)
22. $\rightarrow$ R(1,21)

Přepis znalostní báze na program v Prologu:

```
kytovec(delfin).                          (1)
kytovec(vorvan).                          (2)
ryba(stika).                              (3)
obojzivelnik(zaba).                       (4)
zije(X,voda) :- kytovec(X).               (5)
zije(X,voda) :- ryba(X).                  (6)
zije(X,voda) :- obojzivelnik(X).          (7)
ma(X,plice) :-
       zije(X,voda),
       not(ma(X,zabra)).                  (9)
dycha(X,vzduch) :- ma(X,plice).           (10)
ma(X,zabra) :- obojzivelnik(X).           (11)
ma(X,plice) :- obojzivelnik(X).           (12)
ma(X,zabra) :- ryba(X).                   (13)
```

Klauzuli číslo 8 nemusíme přepisovat, protože Prolog pracuje v uzavřeném světě.

Řešení 12

Znalostní báze a nepřímé odvození:

1. $\rightarrow ptak(skrivan)$ SA_1
2. $\rightarrow ranni\_ptace(skrivan)$ SA_2
3. $\rightarrow ptak(sova)$ SA_3
4. $ranni\_ptace(sova) \rightarrow$ SA_4
5. $\rightarrow clovek(pepa)$ SA_5
6. $\rightarrow ranni\_ptace(pepa)$ SA_6
7. $\rightarrow sportovec(lida)$ SA_7
8. $\rightarrow vlastni(lida, trampolina)$ SA_8
9. $\rightarrow clovek(honza)$ SA_9
10. $\rightarrow vlastni(honza, papousek)$ SA_{10}
11. $\rightarrow vlastni(honza, trampolina)$ SA_{11}
12. $sportovec(honza) \rightarrow$ SA_{12}

13. $sportovec(X) \rightarrow clovek(X)$ SA_{13}
14. $ranni\_ptace(X) \rightarrow doskace(X, dal)$ SA_{14}
15. $sportovec(X), vlastni(X, trampolina) \rightarrow doskace(X, dal)$ SA_{15}
16. $ptak(X), doskace(X, dal) \rightarrow$ PM
17. $ptak(skrivan), doskace(skrivan, dal) \rightarrow$ S(16)$\{skrivan/X\}$
18. $doskace(skrivan, dal) \rightarrow$ R(1,17)
19. $ranni\_ptace(skrivan) \rightarrow doskace(skrivan, dal)$ S(14)$\{skrivan/X\}$
20. $ranni\_ptace(skrivan) \rightarrow$ R(18,19)
21. $\rightarrow$ R(2,20)

Přepis na program v Prologu:

```
ptak(skrivan).                                  (1)
ranni_ptace(skrivan).                           (2)
ptak(sova).                                     (3)
clovek(pepa).                                   (5)
ranni_ptace(pepa).                              (6)
vlastni(lida,trampolina).                       (7)
sportovec(lida).                                (8)
clovek(honza).                                  (9)
vlastni(honza,papousek).                        (10)
vlastni(honza,trampolina).                      (11)
clovek(X) :- sportovec(X).                      (13)
doskace(X,dal) :- ranni_ptace(X).               (14)
doskace(X,dal) :-
       sportovec(X),
       vlastni(X,trampolina).                   (15)
```

Klauzule číslo 4 a 12 nepřepisujeme, protože Prolog pracuje s uzavřeným světem.

Dotazy:

```
?- clovek(X).
?- doskace(X,dal).
?- ranni_ptace(X).
?- sportovec(_).
?- ptak(X),not(ranni_ptace(X)).
```

Řešení 13

Predikáty: $teplota(\langle stupnu \rangle)$ $cestar(\langle kdo \rangle)$
$pocasi(\langle jake \rangle)$ $snehulak(\langle kdo \rangle)$
$pracuje(\langle kdo \rangle, \langle prace \rangle)$ $misto(\langle kdo \rangle, \langle ceho \rangle, \langle ma\ \ co \rangle)$

Program v Prologu:

```
teplota(-5).
pocasi(snezi).
```

```
cestar(luda).
cestar(honza).
snehulak(ferda).
misto(klapka,nos,mrkev).
prace(honza,vypravi_pohadky).
snehulak(X) :- misto(X,nos,mrkev).
snehulak(X) :- misto(X,,oci,uhliky).
pocasi(mraz) :- teplota(X),X<0.
prace(X,odhrnuje_snih) :- pocasi(mraz),pocasi(snezi),cestar(X).
```

Dotazy:

```
?- pocasi(X).
?- cestar(X),prace(X,Y).
?- snehulak(X).
?- prace(luda,_).
?- prace(honza,Prace).
```

Řešení 14

Predikáty: $kocka(\langle kdo \rangle)$ $clovek(\langle kdo \rangle)$ $zvire(\langle kdo \rangle)$
$papousek(\langle kdo \rangle)$ $ptak(\langle kdo \rangle)$ $ma(\langle kdo \rangle, \langle co \rangle)$
$sova(\langle kdo \rangle)$ $savec(\langle kdo \rangle)$ $pocet\_nohou(\langle kdo \rangle, \langle kolik \rangle)$

Program v Prologu:

```
kocka(micka).
papousek(karlik).
papousek(hvezdicka).
sova(houkalka).
clovek(pepik).
ptak(X) :- papousek(X).
ptak(X) :- sova(X).
savec(X) :- kocka(X).
savec(X) :- clovek(X).
zvire(X) :- ptak(X).
zvire(X) :- savec(X),not(clovek(X)).
ma(X,peri) :- ptak(X).
ma(X,zobak) :- ptak(X).
pocet_nohou(X,4) :- savec(X),not(clovek(X)).
pocet_nohou(X,2) :- ptak(X).
pocet_nohou(X,2) :- clovek(X).
ma_rad(Y,X) :- ptak(X),kocka(Y).
ma_rad(pepik,X) :- zvire(X).
ma_rad(karlik,X) :- savec(X),not(kocka(X)).
```

Dotazy:

```
?- pocet_nohou(X,2).
?- ma_rad(Kdo,Koho).
?- ma_rad(X,Y),ma_rad(Y,X).
?- ma_rad(X,Y),pocet_nohou(Y,2).
```

Řešení 15

Predikáty: $vcelka(\langle kdo \rangle)$ $liny(\langle kdo \rangle)$ $patri\_do\_ulu(\langle kdo \rangle, \langle vlastnik \rangle)$
$vcelar(\langle kdo \rangle)$ $hladovy(\langle kdo \rangle)$ $nasbira(\langle kdo \rangle, \langle co \rangle, \langle kolik \rangle)$
$pilny(\langle kdo \rangle)$ $umi(\langle kdo \rangle, \langle co \rangle)$

Program v Prologu:

```
vcelka(maja).
pilny(maja).
vcelka(vilik).
liny(vilik).
vcelka(bzucilka).
pilny(bzucilka).
vcelar(honza).
vcelar(pepa).
patri_do_ulu(maja,honza).
patri_do_ulu(vilik,honza).
patri_do_ulu(bzucilka,pepa).
umi(X,letat) :- vcelka(X).
hladovy(X) :- nasbira(X,med,malo).
nasbira(X,med,hodne) :- pilny(X),vcelka(X).
nasbira(X,med,malo) :- liny(X),vcelka(X).
nasbira(X,med,malo) :-
       vcelar(X),
       patri_do_ulu(_y,X),
       vcelka(_y),hladovy(_y).
nasbira(X,med,hodne) :-
       vcelar(X),
       not(nasbira(X,med,malo)).
```

Dotazy:

```
?- nasbira(X,med,malo).
?- nasbira(X,med,hodne).
?- hladovy(X).
?- patri_do_ulu(maja,X).
```

Rejstřík

Symbols

+ 5
↑ 5
↓ 5
⊢ 16
⊨ 16
α-pravidlo 13, 14
β-pravidlo 13, 14
γ-pravidlo 13, 14
δ-pravidlo 13, 14
! 109

A

algoritmus
- výpočtu v Prologu 103
- zjištění unifikátoru 68

antecedent 44, 55, 56, 60, 65, 67, 101
antinomie 2
arita 8, 10, 43, 50, 63
atom 8, 44, 50, 55, 65, 67
- bázový 44, 58, 89
- jazyka klauzulární logiky 44
- pravdivý ve struktuře a ohodnocení ... 51

axiom 16, 18, 22
- logický 17, 76, 81, 84
- speciální 17, 72, 76, 78, 79, 84

B

backtracking 103
báze znalostní 72, 76, 78, 79, 84, 89, 99
bezespornost 2, 18
- SPD PL 41
- SPD VL 37

C

`call` 109
cíl 102, 103
Colmerauer, A. 83

D

databáze interní 87
DeMorganovy zákony 6, 10, 58
denotace 9, 10, 50
disjunkce atomů v klauzuli 56
disjunkt 6
distributivita 6
dotaz 84, 87, 102, 103
důkaz 11, 13, 16
- hypotéza nepřímá 30
- hypotéza přímá 29
- matematickou indukcí 33
- nepřímý 13, 14, 16, 26, 79
- přímý 16, 24, 78
- sporem 13, 14
- větvený s hypotézami nepřímý 32
- větvený s hypotézami přímý 30

důkazové metody sém. analýzy 11
důsledek
- dvojice klauzulí 65
- znalostní báze 75

F

`fail` 109
fakt 55, 101
- v klauzulární logice 44
- v Prologu 84
- v znalostní bázi 73

formule
- atomická 8
- logicky platná 5, 10
- nesplnitelná 5
- otevřená 8

platná ve struktuře 10
pravdivá při ohodnocení 5
pravdivá ve struktuře a ohodnocení 9
splněna 5
splněna ve struktuře 10
splnitelná 5
splnitelná ve struktuře 9, 11
uzavřená 8
funkce do znalostní báze 73
funktor 43
existenční 43, 47, 50, 51
Skolemův 43

G

generování
do hloubky 99
do šířky 99

H

hlava
klauzule 84, 103
pravidla 84
hodnota formule pravdivostní 11
hypotéza 29, 30, 32
nepřímá 30
přímá 29
hypotetický sylogismus 28

I

individuum 8, 9, 11, 54, 63
interní databáze 87
interpretace 50
atomu 50
formule 9
v znalostní bázi 73
výrokové formule 4

K

Klauzulární axiomatický systém 76
klauzulární logika 42
klauzule 6, 42, 44
cílová 84, 99, 102, 103
Hornova 43, 84, 103
logicky platná 52
nepravdivá ve struktuře 51
nesplnitelná ve struktuře 51
platná ve struktuře 51
popírající množina 60
pravdivá ve struktuře a ohodnocení 51
prázdná 55, 79, 98, 99, 103, 106
konjunkce atomů v klauzuli 56
konjunkt 6, 15
konsekvent 44, 55, 57, 60, 65, 67, 103
konstanta
existenční 43, 47, 50, 51, 53, 58, 67, 70
individuová 9, 43, 50
logická 5, 8
Skolemova 43
kontradikce 5
korektnost
Klauzulárního ax. systému 80
sémantická 18
SPD PL 39
SPD VL 33

L

lemma
o sémantice a dokazatelnosti 35
o neutrální formuli 34
lineární výpočetní strom 99
literál 8, 42
výrokové proměnné 6
logické programování 82
logický
důsledek množiny formulí 5
zákon 10, 77
logika
filozofická 1
formální 1
fuzzy 1
klasická 1
klauzulární 42
matematická 1
modální 1
neklasická 1
pravděpodobnostní 1
predikátová 1
temporální 1
vícehodnotová 1
výroková 1

M

metapravidlo 21

metoda
důkazová ... 18
lineární ... 99–102
sémantická ... 12
syntaktická ... 12, 18
úplná ... 100
minimálnost ... 18
množina
formulí splnitelná ... 5
klauzulí nesplnitelná ... 98, 99
logických spojek funkčně úplná ... 6
neshod ... 68
předpokladů ... 16, 28
popírající ... 60, 79, 100
sporná ... 19, 76
model
formule ... 4, 6
množiny formulí ... 5
znalostní báze ... 75
Modus
Ponens ... 21, 77
Tollens ... 29, 77

N

NAND ... 5
navracení ... 103
negace
atomu ... 58
klauzule ... 60
NOR ... 5
normální forma
disjunktivní ... 6
klauzulární ... 6
konjunktivní ... 6
úplná ... 7
`not` ... 101, 109

O

ohodnocení ... 4, 73
proměnné ... 9, 50
termu ... 9, 50
operátor
Pierceův (NOR) ... 5
Schefferův (NAND) ... 5

P

prikaz
make ... 87
paradigma ... 82
paradox ... 2
Russelův ... 2
popírající množina klauzule ... 60, 79
pravdivost klauzulí ... 51
pravidlo ... 101
dedukční ... 17, 21, 37
DI (disjunkce–implikace) ... 28
E∃ (eliminace ∃) ... 37, 40
E∀ (eliminace ∀) ... 37, 40
ED (eliminace disjunkce) ... 21
EE (eliminace ekvivalence) ... 21
EI (eliminace implikace) ... 21
EK (eliminace konjunkce) ... 21
EN (eliminace negace) ... 27
ES (existenční substituce) ... 77, 81
ID (implikace–disjunkce) ... 32
IK (implikace–konjunkce) ... 28
KD (konjunkce–disjunkce) ... 28
KI (konjunkce–implikace) ... 30
KK (kontrakce) ... 77
logické zákony ... 77
MP (Modus Ponens) ... 21, 77
MT (Modus Tollens) ... 29, 77
odvozovací ... 13, 16, 17, 77
odvozovací rezoluční ... 65
PA (přeuspořádání atomů) ... 77
PK (kontrapozice) ... 27
PT (transpozice) ... 27
R (rezoluční) ... 65, 67, 77, 81, 98, 99, 102
S (substituce) ... 77, 81
sémantické tablo ... 13
SM (sporná množina) ... 26, 27
TI (tranzitivita implikace) ... 25
v Prologu ... 84
v znalostní bázi ... 73
Z∃ (zavedení ∃) ... 37, 40
Z∀ (zavedení ∀) ... 37, 40
ZD (zavedení disjunkce) ... 21
ZE (zavedení ekvivalence) ... 21
ZI (zavedení implikace) ... 21
ZK (zavedení konjunkce) ... 21
ZN (zavedení negace) ... 27
predikát

! 109
call 109
fail 109
not 101, 109
rovnosti 61, 77
řezu 109
prefix formule 45, 61
premisa 12
procedura Herbrandova 37
program v Prologu 84, 86
programování
deklarativní 82
funkcionální 82
procedurální 82
programovani
logické ..3, 42, 44, 59, 65, 72, 75, 83, 99, 101
programování logické 82
prohledávání
do hloubky 100–102
do šířky 100
Prolog 83
proměnná
anonymní 96, 101, 102, 108, 123
vázaná 8, 102
volná 8, 37, 102
předpoklad 12

R

rekurze v Prologu 101
relace do znalostní báze 73
rezoluce 12, 13, 15, 65, 103, 104
v logickém programování 97
rezoluční
řez 65, 77
uzávěr množiny klauzulí 98
rezolventa 98, 103, 104
Robinsonův rezoluční princip 98
rozhodnutelnost 2

S

sémantika 17
jazyka klauzulární logiky 49
složitost formule 4
spojka logická 5, 43
strom
sémantický 12
výpočetní lineární 99
struktura 8, 49, 73
struktura aplikovatelná
na formuli 9
na množinu klauzulí 50
na znalostní bázi 75
substituce 21, 63, 67, 77, 78
existenční 63, 70, 77
sylogismus 2
hypotetický 28
symbol
dokazatelnosti 16
logického vyplývání 16
syntaxe 17
jazyka klauzulární logiky 43
Prologu 96
systém
formální 16
axiomatický 17, 76
bezesporný 19
minimální 19
předpokladový 17, 21
sporný 19
logický 16
Systém přirozené dedukce PL 37
Systém přirozené dedukce VL 21

T

tělo
klauzule 84, 101
pravidla 84
tablo sémantické 12–14
tabulka sémantická 5, 6, 12
tautologie 5, 10, 11, 16
teorie 17, 18
term
bázový 44, 50
existenční 47, 51, 63
jazyka klauzulární logiky 44
substituovatelný 8, 37
tvrzení
existenční 47
univerzální 45

U

unifikátor 67

algoritmus 68
nejobecnější 68, 78, 102
obecnější 68
unifikace 67, 101–104
univerzum diskurzu 8, 50, 63, 73
úplnost 2
sémantická 18
SPD PL 40
SPD VL 36
úsudek deduktivní 12
uzávěr rezoluční 98

V

valuace 4, 73
aplikovatelná 9, 50
proměnné 9, 50
termu 9, 50
věta
o bezespornosti SPD VL 37
o dedukci 24, 25, 27, 30
o disjunkci v antecedentu 56
o konjunkci v konsekventu 57
o korektnosti SPD PL 39
o korektnosti SPD VL 33
o negaci bázového atomu 58
o přímé hypotéze 29
o přímém větveném důkazu 31
o sporném systému 19
o substituci 21
o tranzitivitě implikace 25
o úplnosti SPD PL 40
o úplnosti SPD VL 36
rozšířená o substituci 22, 28
výzva Prologu 87

W

WAM (Warren Abstract Machine) 83

X

XOR 5

Z

zásobník 102, 103
zacyklení výpočtu 101
zákon logický 5, 10, 52, 77
zákony
DeMorganovy 6, 10, 58
distribuce kvantifikátorů 10
závěr 12
znalostní báze 72, 78, 79, 84, 89, 99
zobrazení denotační 9, 50