Kapitola 3

Základy logiky a logického programování

Vědní disciplina nazývaná *logika* je naukou, která se již po více než dvě tisíciletí zabývá studiem lidského uvažování. Své kořeny má již v antickém Řecku, "logiké techne" znamená v řečtině umění uvažovat. Mezi základní úlohy logiky patří nalézání *metod správného usuzování*, tedy postupů, které dovolují přecházet od poznatků, jejichž pravdivost byla ověřena, k poznatkům novým, vyplývajícím a také pravdivým.

Existují dva způsoby *poznávání skutečnosti*: *přímý* a *nepřímý*. *Přímé* poznávání je *empirické* a spočívá v aplikování příslušného poznávacího postupu na objekty reálného světa, jichž se týká. Patří sem např. pozorování, měření a experimentování. Druhý způsob poznávání – *nepřímý* – je založen na *vyplývání* nových poznatků z poznatků již dříve získaných. Většinou mu předchází empirické poznávání a usuzování vyjadřuje závislosti mezi poznatky původními. Někdy se tento způsob poznávání nazývá také *teoretický*. Metody teoretického poznávání – *vyplývání* – zkoumá logika.

Zjištěné poznatky jsou zaznamenávány vhodným jazykem. Logika nabízí jazyky navržené právě k těmto účelům. Lze prověřovat jejich syntax, tedy zabývat se správnou tvorbou jejich výrazů. Symboly logických jazyků však většinou odkazují k nějakým významům, je tedy třeba se zabývat i zpracováním sémantiky.

Jazyky logických počtů (někdy používáme pojmu *kalkulů*) umožňují jednak přesně a úsporně vyjadřovat poznatky, jednak formalizovat usuzování. Každý *symbol abecedy jazyka* zastupuje část poznávané reality ve světě, o kterém jazyk vypovídá, je mu tedy přisouzen význam. Ze symbolů jazyka vytváříme podle syntaktických pravidel *slova*. Správně vytvořená slova jazyků logických kalkulů nazýváme *formule*.

Nové poznatky můžeme odvodit dvěma způsoby: dedukcí a indukcí.

Dedukce je způsob usuzování, kdy z poměrně malého počtu výchozích formulí odvozujeme nebo dokazujeme formule další. Přitom je třeba zachovávat *logickou pravdivost*, tj. přecházet od pravdivých formulí opět jenom k pravdivým. *Důkazem* nějaké formule nazveme při deduktivním usuzování konečnou posloupnost formulí, jejíž posledním členem je dokazovaná formule a v níž pravdivost každé formule byla náležitě prokázána. To lze provést dvěma způsoby:

1. Formule tvořící jistý výchozí soubor jako pravdivé definujeme. Tyto formule nazýváme *axiómy*. Axióm potom může být libovolným členem důkazu (důkaz = posloupnost formulí) s výjimkou posledního; axióm totiž dedukcí nedokazujeme, je pravdivý podle definice.

2. Pravdivost formulí v posloupnosti lze dokázat pomocí pravidel správného usuzování z formulí, které nazýváme *premisy* (předpoklady), přičemž pravidla usuzování zaručují, že jsou-li premisy pravdivé, je pravdivý i *závěr*.

Za axiómy můžeme vzít např. formule, které vyjadřují základní výsledky našeho pozorování či experimentu (v případě, že deduktivně zkoumáme např. nějaký fyzikální systém), nebo formule, které pokládáme za evidentní. V podstatě lze říci, že výběrem axiómů vymezujeme tu část reality, kterou budeme dedukcí poznávat. Je to ta část, ve které axiómy platí.

Jedním ze základních požadavků kladených na soubor axiómů je požadavek jejich bezespornosti. Soubor axiómů prohlásíme za *bezesporný*, pokud existují pouze formule, které z něj lze pravidly správného usuzování odvodit. V opačném případě (kdy lze odvodit libovolnou formuli) hovoříme o *sporném* souboru axiómů. Bezespornost lze také definovat požadavkem, aby ze souboru axiómů nebylo možno odvodit nějaké formule a současně jejich negaci. Lze ukázat, že obě definice bezespornosti jsou ekvivalentní, první z nich však nepoužívá pojem "negace".

Formule, které dokazujeme ze souboru axiómů, nazýváme *teorémy* (někdy též *věty*). Důkaz teorému (věty), tak jak jsme jej popsali, má některé významné vlastnosti: Teorém je odvoditelný podle pravidel správného usuzování z předpokladů. Předpoklady jsou buď axiómy nebo teorémy, které z axiómů vyplývají a lze je z nich dokázat. Teorém nemůže být svým vlastním předpokladem nebo předpokladem svých předpokladů.

Při budování axiomatizovaného deduktivního systému se můžeme také setkat s pojmem definice. *Definice* pojmenovávají složitější poznatky a slouží k větší úspornosti zápisu. Teoreticky vzato se lze bez definic obejít, jejich praktický význam je však značný.

Indukce spočívá v odvozování obecného poznatku z řady poznatků speciálních. Induktivní metody jsou spojovány s empirickým poznáváním, kterým speciální požadavky často získáváme. Chceme-li např. dokázat obecný poznatek "Všichni sourozenci pana X mají krevní skupinu A", provedeme každému sourozenci pana X zkoušku krve a jestliže ve všech případech zjistíme skupinu A, je poznatek dokázán. Takováto metoda vychází z posouzení *všech možných* speciálních případů a nazývá se *úplná indukce* (nezaměňovat s matematickou indukcí). Je zřejmé, že úplná indukce je pravidlem správného usuzování. Důkaz úplnou indukcí je však proveditelný jen v případě konečného počtu alternativ, které je třeba posoudit, a přijatelný, pokud tento počet není příliš velký.

Často se stává, že všechny speciální případy není možné vyhodnotit, protože jejich počet je nekonečný nebo je konečný, ale případů je příliš mnoho. Pak se uchylujeme k prozkoumání pouze omezeného konečného počtu případů. Získaný úsudek však není pravidlem správného odvozování a nazývá se *neúplná indukce*. Ačkoliv neúplná indukce nemůže zaručit pravdivost závěrů, bývá poměrně často využívána, protože je jediným dostupným způsobem usuzování (odvozování). Např. většina poznatků z fyziky je odvozena neúplnou indukcí. Závěry takto získané pak nenazýváme teorémy (větami), nýbrž *zákony*.

3.1 Výroková logika

Nejjednodušší logikou je *výroková logika*. Tvrzení, o kterých má smysl rozhodnout, zda jsou pravdivá, nazýváme *výroky*. Věty typu "tři plus čtyři je sedm", "Jiří má sestru" jsou tzv. *elementární výroky*. Jejich elementárnost spočívá v tom, že je nelze dále rozložit na výroky jednodušší. Výrazy typu "tři plus čtyři" nebo "má sestru" již nejsou výroky, protože nemá smysl hovořit o jejich pravdivosti. Výroková logika se nezabývá vnitřní strukturou elementárních výroků, na elementárních výrocích posuzujeme pouze jejich pravdivost nebo nepravdivost. Výrokovou logiku zajímá pouze, zda výroky nabývají jednu ze dvou možných *pravdivostních hodnot* – *pravda* nebo *nepravda* (true nebo false). Elementární výroky lze proto nahradit (zastoupit) libovolně zvolenými symboly, kterým přiřadíme stejnou pravdivostní hodnotu jako výrokům zastoupeným.

Tiskacími písmeny velké latinské abecedy (popř. s indexem) budeme označovat *výrokové proměnné*. Výroková proměnná zastupuje elementární výrok (viz výše), kterému můžeme přiřadit pravdivostní hodnotu. Obdobně můžeme přiřadit pravdivostním hodnotám *pravda* a *nepravda* jednodušší symboly – např. 1 a 0 nebo T , F . Potom zastupuje-li např. výroková proměnná A výrok "číslo 25 je sudé", budeme říkat že A *má hodnotu* (nebo raději A *nabývá hodnoty*) 0 , resp. F místo toho, abychom říkali, že výrok "číslo 25 je sudé" je nepravdivý. Přitom však budeme neustále mít na mysli, že A zastupuje uvedený výrok a symbol 0 , resp. F znamená hodnotu "nepravda". Přiřazení pravdivostní hodnoty elementárním výrokům výroková logika neřeší, pravdivostní hodnotu získáme aplikováním vhodného poznávacího postupu na zkoumanou realitu. Elementární výroky tedy vyjadřují výsledky přímého poznávání.

Z elementárních výroků můžeme tzv. *logickými spojkami* a v případě nutnosti též závorkami vytvářet *složené výroky*. Ve složených výrocích budeme používat logické spojky $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\Leftrightarrow$ a jimi budeme definovat *logické funkce* negace, logický součin (konjunkce), logický součet (disjunkce), podmíněný soud (implikace), a logická rovnost (ekvivalence). V dalším budeme předpokládat, že význam logických spojek a jimi vyjádřených logických funkcí je dostatečně znám z předchozího studia. Pokud ne, lze jej nalézt v libovolné učebnici základů logiky (např. [Brabec80], [Manna81], [Štěpánek82]).

3.1.1 Jazyk výrokové logiky

Zavedením výrokových proměnných, logických spojek a závorek jsme vymezili symboly používané *jazykem výrokové logiky*. Výrazy tohoto jazyka v dalším využijeme k popisování našich poznatků z reálného světa a k odvozování poznatků dalších. Každá posloupnost těchto symbolů však nepatří do jazyka výrokové logiky (např. $A \wedge \rightarrow B$, $\rightarrow B$ atd.). Správně vytvořené výrazy jazyka výrokové logiky budeme nazývat *výrokové formule*. Jejich syntax vymezuje následující definice:

Označme $V = \{\neg, \wedge, \vee, \rightarrow, \Leftrightarrow, (,), A, B, C, ...\}$ abecedu jazyka výrokové logiky, kde $A, B, C, ...$ označují výrokové proměnné.

1. Každá výroková proměnná je výrokovou formulí.
2. Jestliže $\mathcal{A}$ a $\mathcal{B}$ jsou výrokové formule, pak také $(\neg\,\mathcal{A})$, $(\mathcal{A} \wedge \mathcal{B})$, $(\mathcal{A} \vee \mathcal{B})$, $(\mathcal{A} \rightarrow \mathcal{B})$ a $(\mathcal{A} \Leftrightarrow \mathcal{B})$ jsou výrokové formule.
3. Žádné jiné výrokové formule než podle bodů 1 a 2 neexistují.

Jazykem výrokové logiky nad abecedou V potom nazýváme množinu všech výrokových formulí správně vytvořených ze symbolů abecedy V . Abeceda V může být konečná nebo nekonečná, takto definované jazyky budou vždy nekonečné.

Velká psací (kaligrafická) písmena latinské abecedy, která jsme v definici použili, nepatří mezi symboly jazyka výrokové logiky. Jsou to metasymboly, které jsme zavedli, abychom mohli o výrazech jazyka výrokové logiky vypovídat. Výrazy jazyka dostaneme, když ve výrokové formuli za všechny metasymboly dosadíme výrokové proměnné. Tyto metasymboly mají podobný význam jako neterminální proměnné u gramatiky.

Každá obecná výroková formule zastupuje nekonečně mnoho výroků nebo formulí. Např. jestliže A, B a C jsou výrokové proměnné, pak výroková formule $((\mathcal{A} \Leftrightarrow \mathcal{B}) \rightarrow \mathcal{A})$ zastupuje formule $((A \Leftrightarrow B) \rightarrow A)$, $((B \Leftrightarrow C) \rightarrow B)$ atd. Protože však $\mathcal{A}$ a $\mathcal{B}$ mohou zastupovat též výrokové formule tvořené podle bodu 2 z dalších formulí, zastupuje uvedená formule také např. formuli $(((A \wedge B) \Leftrightarrow C) \rightarrow (A \wedge B))$ a podobně.

Existují výrokové formule, které jsou *identicky pravdivé*, tj. pravdivé pro jakékoli pravdivostní hodnoty přiřazené proměnným vyskytujícím se ve výrokové formuli. Takovéto formule nazýváme *výrokové tautologie*. Pravdivostní tabulka výrokové tautologie bude mít ve všech řádcích pravdivostní hodnoty 1 , resp. T . Příkladem výrokových tautologií jsou výrokové formule

$$((\neg\,(\mathcal{A} \wedge \mathcal{B})) \Leftrightarrow ((\neg\,\mathcal{A} \;\vee \neg\,\mathcal{B}))) \;,$$
$$((\neg\,(\mathcal{A} \vee \mathcal{B})) \Leftrightarrow ((\neg\,\mathcal{A} \;\wedge \neg\,\mathcal{B}))) \;,$$

které jsou známy jako *de Morganova pravidla* pro úpravy výrokových formulí.

Vyhodnocením pravdivosti výše uvedených výrokových formulí (de Morganových pravidel) pravdivostní tabulkou snadno ověříme, že formule jsou pravdivé pro všechna možná přiřazení pravdivostních hodnot výrokovým proměnným, tj. že formule skutečně jsou tautologiemi. Takové vyhodnocení je vlastně důkaz pravdivosti metodou úplné indukce, kterou jsme popsali na počátku kapitoly. Podobně můžeme vyhodnocením a zápisem do pravdivostní tabulky posoudit libovolnou výrokovou formuli. Existuje tedy algoritmus, který o libovolné výrokové formuli je schopen rozhodnout, zda formule je či není tautologií. Výroková logika je tedy *rozhodnutelná*.

Formule, která je identicky nepravdivá, se nazývá *kontradikce*. Dále platí, že je-li formule $\mathcal{A}$ kontradikce, potom $\neg\mathcal{A}$ je výroková tautologie.

Jestliže formule $\mathcal{A} \Leftrightarrow \mathcal{B}$ je výrokovou tautologií, říkáme, že formule $\mathcal{A}$ a $\mathcal{B}$ jsou *logicky ekvivalentní*.

Nyní si ukažme, jak pravidla správného usuzování souvisejí s výrokovými tautologiemi:

Mějme dány výrokové formule $\mathcal{C} \Leftrightarrow \mathcal{D}$ a $\mathcal{C} \rightarrow \mathcal{D}$. Budeme zkoumat, zda z platnosti jedné z nich můžeme usoudit na platnost druhé. Zapíšeme-li obě formule formou pravdivostní tabulky (viz tabulka 3.1), zjistíme, že formule $\mathcal{C} \Leftrightarrow \mathcal{D}$ nabývá hodnoty 1 v prvním a čtvrtém řádku, zatímco formule $\mathcal{C} \rightarrow \mathcal{D}$ v prvním, druhém a čtvrtém řádku. Formule $\mathcal{C} \rightarrow \mathcal{D}$ je pravdivá pro obě možnosti, pro které je pravdivá $\mathcal{C} \Leftrightarrow \mathcal{D}$ a navíc je ještě pravdivá pro jednu další kombinaci pravdivostních hodnot formulí $\mathcal{C}$ a $\mathcal{D}$. Když je pravdivá formule $\mathcal{C} \Leftrightarrow \mathcal{D}$, je jistě pravdivá i $\mathcal{C} \rightarrow \mathcal{D}$, čili z pravdivosti $\mathcal{C} \Leftrightarrow \mathcal{D}$ lze oprávněně usuzovat na pravdivost formule $\mathcal{C} \rightarrow \mathcal{D}$. Tuto skutečnost můžeme zapsat formulí $(\mathcal{C} \Leftrightarrow \mathcal{D}) \rightarrow (\mathcal{C} \rightarrow \mathcal{D})$, která je výrokovou tautologií.

Tabulka 3.1: Pravdivostní tabulka složeného výroku $(A \Leftrightarrow B) \rightarrow (A \rightarrow B)$:

A	B	$A \Leftrightarrow B$	$A \rightarrow B$	$(A \Leftrightarrow B) \rightarrow (A \rightarrow B)$
0	0	1	1	1
0	1	0	1	1
1	0	0	0	1
1	1	1	1	1

Řekneme, že výroková formule $\mathcal{B}$ *vyplývá* z formule $\mathcal{A}$, když formule $\mathcal{B}$ je pravdivá vždy, když je pravdivá formule $\mathcal{A}$. V příkladu uvedeném v předchozím odstavci formule $\mathcal{C} \rightarrow \mathcal{D}$ vyplývá z $\mathcal{C} \Leftrightarrow \mathcal{D}$. Jestliže formule $\mathcal{B}$ vyplývá z formule $\mathcal{A}$ (zapíšeme $\mathcal{A} \models \mathcal{B}$ – viz dále), pak formule $\mathcal{A} \rightarrow \mathcal{B}$ je výrokovou tautologií. Formuli $\mathcal{A}$ nazýváme *antecendent* nebo *premisa* a formuli $\mathcal{B}$ *konsekvent* nebo také *konkluse (závěr)*. Z definice logického vyplývání a z vlastností implikace můžeme odvodit dvě formulace věty významné pro dokazování:

1. Formule $\mathcal{B}$ logicky vyplývá z formulí $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_k$ tehdy a jen tehdy, když formule $(\mathcal{A}_1 \wedge \mathcal{A}_2 \wedge ... \wedge \mathcal{A}_k) \rightarrow \mathcal{B}$ je výrokovou tautologií.

2. Formule $\mathcal{B}$ logicky vyplývá z formulí $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_k$ tehdy a jen tehdy, když formule $(\mathcal{A}_1 \wedge \mathcal{A}_2 \wedge ... \wedge \mathcal{A}_k \wedge \neg \mathcal{B})$ je výrokovou kontradikcí.

Z logického vyplývání lze také odvodit další významné pravidlo správného usuzování, a to *pravidlo odloučení* neboli pravidlo *modus ponens*, které říká, že jsou-li formule $\mathcal{A}$ a $\mathcal{A} \rightarrow \mathcal{B}$ výrokovými tautologiemi, pak i formule $\mathcal{B}$ je výrokovou tautologií.

Dosud jsme dokazovali, že nějaká formule je výrokovou tautologií, úplnou indukcí s použitím sémantiky logických funkcí. Pravidlo modus ponens však dovoluje odvozovat dedukcí. Zvolíme některé výrokové formule za axiómy jazyka výrokové logiky. Pokud vybereme jako axiómy výrokové tautologie, umíme pravidlem modus ponens odvozovat další výrokové tautologie. Jedním z takových možných systémů axiómů je následující soubor formulí:

$$(\mathcal{A} \rightarrow (\mathcal{B} \rightarrow \mathcal{A})) \tag{3.1}$$

$$((\mathcal{A} \rightarrow (\mathcal{B} \rightarrow \mathcal{C})) \rightarrow ((\mathcal{A} \rightarrow \mathcal{B}) \rightarrow (\mathcal{A} \rightarrow \mathcal{C}))) \tag{3.2}$$

$$((\neg \mathcal{B} \rightarrow \neg \mathcal{A}) \rightarrow ((\neg \mathcal{B} \rightarrow \mathcal{A}) \rightarrow \mathcal{B})) \tag{3.3}$$

Snadno se lze přesvědčit, že všechny tři formule jsou tautologiemi. Stačí sestrojit příslušné pravdivostní tabulky.

V axiómech (3.1) až (3.3) se vyskytují pouze dvě základní logické funkce: negace a implikace. Zbývající tři lze pomocí nich definovat takto:

$$\mathcal{A} \wedge \mathcal{B} \quad \text{definujeme jako} \quad \neg(\mathcal{A} \rightarrow \neg \mathcal{B})) \tag{3.4}$$

$$\mathcal{A} \vee \mathcal{B} \quad \text{definujeme jako} \quad ((\neg \mathcal{A}) \rightarrow \mathcal{B}) \tag{3.5}$$

$$\mathcal{A} \Leftrightarrow \mathcal{B} \quad \text{definujeme jako} \quad (\mathcal{A} \rightarrow \mathcal{B}) \wedge (\mathcal{B} \rightarrow \mathcal{A}) \tag{3.6}$$

Formule získané dedukcí z axiómů nazveme *formálně dokazatelné*. Ke každé formálně dokazatelné formuli však musí existovat důkaz její logické pravdivosti, jak bylo uvedeno v úvodu této kapitoly. Pravidlo modus ponens umožňuje plně abstrahovat od sémantiky jednotlivých logických spojek a dovoluje pracovat s formulemi jako s posloupnostmi symbolů, tedy z hlediska syntaktického. Jestliže tedy najdeme v dosud provedené části důkazu formule $\mathcal{A} \rightarrow \mathcal{B}$ a $\mathcal{A}$, které již byly dokázány, můžeme do důkazu přepsat formuli $\mathcal{B}$ a považovat ji za dokázanou, aniž bychom zkoumali pravdivostní hodnoty. Korektnost takového kroku zaručuje pravidlo modus ponens.

Každá formálně dokazatelná formule jazyka výrokové logiky je výrokovou tautologií, protože axiómy (3.1) až (3.3) jsou výrokové tautologie a pravidlo modus ponens vede od tautologií opět k tautologiím. Otázkou ovšem je, zda každá výroková tautologie je formálně dokazatelnou formulí z axiómu (3. 1) až (3. 3) použitím pravidla modus ponens. Lze dokázat, že uvedený soubor axiómů (3. 1) až (3. 3) je *úplný* v širokém smyslu či podle Gödela, tj. každá formule, která je tautologií, je z něj formálně dokazatelná [Štěpánek82].

Formule (3. 1) až (3. 3) nejsou jedinou možností, jak zvolit axiómy výrokové logiky. Jiným používaným souborem axiómů jazyka výrokové logiky jsou např. formule

$$\mathcal{A} \rightarrow (\mathcal{B} \rightarrow \mathcal{A}) \tag{3.7}$$

$$(\mathcal{A} \rightarrow (\mathcal{B} \rightarrow \mathcal{C})) \rightarrow ((\mathcal{A} \rightarrow \mathcal{B}) \rightarrow (\mathcal{A} \rightarrow \mathcal{C})) \tag{3.8}$$

$$(\mathcal{A} \wedge \mathcal{B}) \rightarrow \mathcal{A} \tag{3.9}$$

$$(\mathcal{A} \wedge \mathcal{B}) \rightarrow \mathcal{B} \tag{3.10}$$

$$\mathcal{A} \rightarrow (\mathcal{B} \rightarrow (\mathcal{A} \wedge \mathcal{B})) \tag{3.11}$$

$$\mathcal{A} \rightarrow (\mathcal{A} \vee \mathcal{B}) \tag{3.12}$$

$$\mathcal{B} \rightarrow (\mathcal{A} \vee \mathcal{B}) \tag{3.13}$$

$$(\mathcal{A} \rightarrow \mathcal{C}) \rightarrow ((\mathcal{B} \rightarrow \mathcal{C}) \rightarrow ((\mathcal{A} \vee \mathcal{B}) \rightarrow \mathcal{C})) \tag{3.14}$$

$$(\mathcal{A} \rightarrow \mathcal{B}) \rightarrow ((\mathcal{A} \rightarrow \neg \mathcal{B}) \rightarrow \neg \mathcal{A}) \tag{3.15}$$

$$\neg\neg \mathcal{A} \rightarrow \mathcal{A} \tag{3.16}$$

Logickou funkci ekvivalence $\mathcal{A} \Leftrightarrow \mathcal{B}$ opět definujeme jako $(\mathcal{A} \rightarrow \mathcal{B}) \wedge (\mathcal{B} \rightarrow \mathcal{A})$. Oba uvedené soubory axiómů, tj. formule (3. 1) až (3. 3) a (3. 7) až (3. 16) jsou ekvivalentní v tom smyslu, že formule formálně dokazatelná z jednoho souboru axiómů je formálně dokazatelná i z druhého. To lze dokázat tím, že odvodíme jeden soubor axiómů dedukcí z druhého. Protože jsme řekli, že libovolnou výrokovou tautologii lze formálně odvodit z axiómů (3. 1) až (3. 3), lze odvodit i formule (3. 7) až (3. 16), všechny tyto formule jsou totiž výrokové tautologie. Lze ukázat, že platí i opačný vztah. Podobně existuje i mnoho dalších ekvivalentních souborů axiómů jazyka výrokové logiky.

Prohlášení výrokové tautologie za pravdivý výrok je jistě oprávněné, nicméně výroková tautologie není žádným poznatkem o zkoumané skutečnosti. Je totiž identicky pravdivá, tedy pro libovolné přiřazení pravdivostních hodnot výrokovým proměnným a její pravdivostní hodnota nezávisí na tom, jaké skutečnosti jednotlivé proměnné popisují. *Pravdivost logické tautologie vyplývá z její struktury.*

Empirické poznatky zapíšeme formulemi, které budou také pravdivé, ale nebudou výrokovými tautologiemi. To znamená, že pro některé kombinace pravdivostních hodnot budou tyto formule nepravdivé. Tím, že takovouto formuli prohlásíme za pravdivou, říkáme, že tyto kombinace nemohou nastat. Vylučujeme je na základě pozorování apod. V tomto případě jsou však výrokovým proměnným přiřazeny *vždy* určité konkrétní hodnoty, které popisují zkoumanou realitu.

Předpokládejme, že realitu, kterou chceme poznávat, popíšeme k elementárními výroky, z nichž každý může být pravdivý nebo nepravdivý. Existuje tedy $N_0 = 2^k$ přiřazení jejich pravdivostních hodnot. Pokusem nebo pozorováním vyloučíme některé z těchto možností a připustíme, že může nastat pouze $N_1 < N_0$ možností. Každé poznávání má charakter takovéhoto vylučování některých alternativ (stavů) jako nemožných. Logika nabízí jazyk k zaznamenávání takových poznatků (např. ve tvaru výroků, formulí výrokové logiky) a dále poskytuje formální aparát, kterým lze získat další poznatky odvozováním. Formule, které zachycují tyto poznatky, nejsou však výrokovými tautologiemi. Nemůžeme tedy např. za výrokové proměnné dosazovat libovolné hodnoty, ale jen ty, které z daného poznávacího postupu obdržíme. Tyto formule budeme nazývat *pravdivé formule* (podmíněně, mimologicky), zatímco identicky pravdivé formule výrokové logiky budeme vždy nazývat výrokovými tautologiemi. V odvození pravidla modus ponens jsme předpokládali, že $\mathcal{A}$ a $\mathcal{A} \to \mathcal{B}$ jsou identicky pravdivé formule (výrokové tautologie). Podobnou úvahou se však lze přesvědčit, že platí i slabší tvrzení: jestliže formule $\mathcal{A}$ a $\mathcal{A} \to \mathcal{B}$ jsou pravdivé pro nějaké přiřazení pravdivostních hodnot výrokovým proměnným, které se v nich vyskytují, je pro stejné přiřazení i formule $\mathcal{B}$ pravdivá. Tím získáme pravidlo modus ponens platné pro pravdivé formule i pro výrokové tautologie.

3.1.2 Teorie a modely jazyka výrokové logiky

Zvolme si nějaký soubor axiómů výrokové logiky, např. axiómy (3.1) až (3.3). Přidáme-li k těmto axiómům některé další pravdivé formule – tzv. *mimologické axiómy*, můžeme dedukcí odvozovat další formule (poznatky). Ty budou pravdivé ve všech světech, ve kterých jsou pravdivé dané mimologické axiómy.

Označme T nějakou množinu formulí jazyka výrokové logiky. Tyto formule vezmeme jako mimologické axiómy. Množinu T budeme nazývat *teorií jazyka výrokové logiky*. Důkazem formule $\mathcal{A}$ z teorie T je konečná posloupnost formulí taková, že její poslední člen je formule $\mathcal{A}$ a každá z předchozích formulí je buď axióm výrokové logiky, patří do T nebo se získá z předchozích formulí odvozením (aplikací pravidla modus ponens). Potom řekneme, že formule $\mathcal{A}$ je *formálně dokazatelná z teorie* T, resp. $\mathcal{A}$ je *teorém* v T a píšeme $T \vdash \mathcal{A}$ (čteme "T dává $\mathcal{A}$"). Pokud $T = \emptyset$, píšeme $\vdash \mathcal{A}$ a formule $\mathcal{A}$ je formálně dokazatelnou formulí jazyka výrokové logiky (a výrokovou tautologií, jak jsme ukázali).

Teorie T je *sporná*, pokud lze najít nějakou formuli $\mathcal{A}$ takovou, že $T \vdash \mathcal{A}$ a $T \vdash \neg\mathcal{A}$. V opačném případě se teorie nazývá *bezesporná*.

Důležitou vlastnost odvozování popisuje tzv. *věta o dedukci*:
Jestliže $T \cup (\mathcal{A}) \vdash \mathcal{B}$, pak $T \vdash (\mathcal{A} \to \mathcal{B})$, kde $\mathcal{A}$ a $\mathcal{B}$ jsou formule výrokové logiky. Jestliže tedy z teorie T a nějaké formule $\mathcal{A}$ je formálně dokazatelná formule $\mathcal{B}$, pak z teorie T je formálně dokazatelná formule $\mathcal{A} \to \mathcal{B}$. Důkaz věty o dedukci vychází z dané soustavy axiómů a lze ho najít prakticky v každé literatuře zabývající se matematickými základy logiky (např. [Brabec80], [Manna81], [Štěpánek82]).

Použití věty o dedukci si ilustrujme na následujících příkladech:

Příklad 3. 1: Když $T = \emptyset$, pak $\mathcal{A} \vdash \mathcal{B}$ a podle věty o dedukci $\vdash (\mathcal{A} \to \mathcal{B})$. Tedy, když z formule $\mathcal{A}$ lze formálně dokázat $\mathcal{B}$, pak $\mathcal{A} \to \mathcal{B}$ je výrokovou tautologií ($\mathcal{B}$ implikuje $\mathcal{A}$).

Příklad 3. 2: Dokážeme, že formule $(\mathcal{A} \to \mathcal{B}) \to ((\mathcal{B} \to \mathcal{C}) \to (\mathcal{A} \to \mathcal{C}))$ je formálně dokazatelnou formulí jazyka výrokové logiky. Jednotlivé kroky dedukce (jednotlivé formule) budeme číslovat a vpravo od formulí budeme vyznačovat "ospravedlnění" formule v důkazu. Mimologické axiómy (formule teorie) budeme značit MA a číslem, formule získané pravidlem modus ponens pak budeme označovat MP a čísly formulí, které byly při aplikaci pravidla použity.

Předpokládejme, že teorie T je tvořena třemi formulemi:

$$T = \{ \mathcal{A} \to \mathcal{B},\ \mathcal{B} \to \mathcal{C},\ \mathcal{A} \}$$

1. $\mathcal{A} \to \mathcal{B}$	MA 1
2. $\mathcal{B} \to \mathcal{C}$	MA 2
3. $\mathcal{A}$	MA 3
4. $\mathcal{B}$	MP 1, 3
5. $\mathcal{C}$	MP 2, 4

Tedy $\{\mathcal{A} \to \mathcal{B},\ \mathcal{B} \to \mathcal{C},\ \mathcal{A}\} \vdash \mathcal{C}$. Podle věty o dedukci dostaneme v prvním kroku $\{\mathcal{A} \to \mathcal{B},\ \mathcal{B} \to \mathcal{C}\} \vdash (\mathcal{A} \to \mathcal{C})$, dalším aplikováním téže věty dostaneme $(\mathcal{A} \to \mathcal{B}) \vdash (\mathcal{B} \to \mathcal{C}) \to (\mathcal{A} \to \mathcal{C})$ a jejím třetím použitím konečně získáme $\vdash (\mathcal{A} \to \mathcal{B}) \to ((\mathcal{B} \to \mathcal{C}) \to (\mathcal{A} \to \mathcal{C}))$. Formule $(\mathcal{A} \to \mathcal{B}) \to ((\mathcal{B} \to \mathcal{C}) \to (\mathcal{A} \to \mathcal{C}))$ je formálně odvoditelná z axiómů výrokové logiky, ačkoli jsme ji z nich bezprostředně neodvozovali. Použili jsme větu o dedukci, která z těchto axiómů vychází.

Ukážeme si také, že dedukcí lze dokázat, že obsahuje-li teorie T nějakou formuli a současně i její negaci (teorie T je sporná), lze z této teorie odvodit každou formuli. Předpokládejme tedy, že $T = \{ \mathcal{A}, \neg \mathcal{A} \}$, a dokážeme formuli $\mathcal{B}$ (libovolnou):

1. $\mathcal{A}$	MA 1
2. $\neg \mathcal{A}$	MA 2
3. $\mathcal{A} \rightarrow (\neg \mathcal{B} \rightarrow \mathcal{A})$	Axióm (3.1) dosazením $\neg \mathcal{B}$ za $\mathcal{B}$
4. $\neg \mathcal{A} \rightarrow (\neg \mathcal{B} \rightarrow \neg \mathcal{A})$	Axióm (3.1) dosazením $\neg \mathcal{A}$ za $\mathcal{A}$, $\neg \mathcal{B}$ za $\mathcal{B}$
5. $\neg \mathcal{B} \rightarrow \mathcal{A}$	MP 1, 3
6. $\neg \mathcal{B} \rightarrow \neg \mathcal{A}$	MP 2, 4
7. $(\neg \mathcal{B} \rightarrow \neg \mathcal{A}) \rightarrow ((\neg \mathcal{B} \rightarrow \mathcal{A}) \rightarrow \mathcal{B})$	Axióm (3.3)
8. $(\neg \mathcal{B} \rightarrow \mathcal{A}) \rightarrow \mathcal{B}$	MP 6, 7
9. $\mathcal{B}$	MP 5, 7

O formuli $\mathcal{B}$ nebyly učiněny žádné předpoklady, lze tedy ze sporných axiómů $\neg \mathcal{A}$, $\mathcal{A}$ odvodit libovolnou formuli $\mathcal{B}$.

Všimněme si, že formální odvozování je postupem, který se opírá výhradně o syntaktickou stránku formulí. Avšak i když budeme brát v úvahu sémantiku logických spojek, výroková logika sleduje na výrokových proměnných pouze jejich pravdivostní hodnoty nezávisle na poznatcích, které reprezentují. Zkoumáme-li tedy sémantiku jazyka výrokové logiky, můžeme abstrahovat od konkrétních poznatků a nahradit je jejich pravdivostní hodnotou. Výsledky potom lze dodatečně interpretovat v reálném světě, který zkoumáme. Přiřazení pravdivostních hodnot všem symbolům abecedy jazyka výrokové logiky nazveme *modelem tohoto jazyka*. Stejný model potom zastupuje všechny soubory elementárních výroků, které přiřazují jednotlivým výrokovým proměnným tutéž pravdivostní hodnotu.

Má-li abeceda jazyka k symbolů, existuje 2^k různých modelů (2^k řádků v pravdivostních tabulkách). Označíme-li M nějaký model jazyka výrokové logiky a $\mathcal{A}$ je formulí tohoto jazyka, potom model M přiřazuje pravdivostní hodnotu všem výrokovým proměnným v $\mathcal{A}$ a pomocí tabulek lze vyhodnotit pravdivostní hodnotu formule $\mathcal{A}$. Pokud model M vede k vyhodnocení formule s pravdivostní hodnotou `true`, řekneme, že formule $\mathcal{A}$ *je pravdivá v modelu* M a píšeme $M \models \mathcal{A}$. Formule $\mathcal{A}$, která je pravdivá ve všech možných modelech, je výrokovou tautologií, formule $\mathcal{A}$, která není pravdivá ve všech modelech, je výrokovou kontradikcí.

Označme T teorii jazyka výrokové logiky a M model tohoto jazyka. Když $M \models \mathcal{A}$ pro všechny formule $\mathcal{A} \in T$, potom řekneme, že M je modelem teorie T. Jestliže v každém modelu M teorie T je formule $\mathcal{A}$ pravdivá, potom formule $\mathcal{A}$ *logicky vyplývá* z teorie T a píšeme $T \models \mathcal{A}$ (s logickým vyplýváním jsme se už setkali a zde se k němu znovu vracíme a definujeme ho modelem).

Důležitá je souvislost formální (syntaktické) dokazatelnosti a logického (sémantického) vyplývání. Lze ukázat, že $T \vdash \mathcal{A}$ právě tehdy, když $T \models \mathcal{A}$, tj. formule $\mathcal{A}$ je formálně odvoditelná z T právě tehdy, když tato formule z T logicky vyplývá. Formální dokazování a logické vyplývání lze ilustrovat následujícím příkladem (volně podle [Clocksin81]):

Příklad 3. 3: V detektivním příběhu došlo ke zločinu v domě pana X a policejní komisař vyšetřující zločin zjistil na místě činu následující fakta:

a) Soused pana X řekl, že pana X viděl v obývacím pokoji.

b) Obývací pokoj sousedí s kuchyní. Zločinec vystřelil v kuchyni a ránu bylo slyšet ve všech sousedních místnostech. Pan X, který má dobrý sluch, řekl, že ránu neslyšel.

Dokážeme A) formálním důkazem a B) logickým vyplýváním tvrzení, že pokud soused mluvil pravdu, pan X lhal.

Zavedeme výrokové proměnné A, B, C, D a E a přiřadíme jim elementární výroky:

A . . . Soused pana X mluvil pravdu.
B . . . Pan X byl v obývacím pokoji.
C . . . Pan X byl blízko kuchyně.
D . . . Pan X slyšel výstřel.
E . . . Pan X mluvil pravdu.

Poznatky, které vyšetřovatel získal na místě činu, můžeme v jazyce výrokové logiky zapsat takto:

$$A \to B,\ \ B \to C,\ \ C \to D,\ \ D \to \neg E$$

Chceme odvodit tvrzení teorému, které je vyjádřeno formulí $A \to \neg E$. Tedy teorie $T = \{A \to B, B \to C, C \to D, D \to \neg E\}$ a máme dokázat, že $T \vdash (A \to \neg E)$, resp. $T \models (A \to \neg E)$.

ad A) Formální důkaz (odvození) formule $A \to \neg E$ z teorie T :
Pro postupné odvození použijeme axiómy (3.1) až (3.3) a pravidlo modus ponens:

1. $A \to B$	MA 1
2. $B \to C$	MA 2
3. $C \to D$	MA 3
4. $D \to \neg E$	MA 4
5. $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$	Axióm (3.2)
6. $(B \to C) \to (A \to (B \to C))$	Axióm (3.1)
7. $(A \to (B \to C))$	MP 2, 6
8. $((A \to B) \to (A \to C))$	MP 5, 7
9. $A \to C$	MP 1, 8
10. $(A \to (C \to D)) \to ((A \to C) \to (A \to D))$	Axióm (3.2)
11. $(C \to D) \to (A \to (C \to D))$	Axióm (3.1)
12. $(A \to (C \to D))$	MP 3, 11
13. $((A \to C) \to (A \to D))$	MP 10, 12
14. $A \to D$	MP 13, 9
15. $(A \to (D \to \neg E)) \to ((A \to D) \to (A \to \neg E))$	Axióm (3.2)
16. $(C \to \neg E) \to (A \to (D \to \neg E))$	Axióm (3.1)
17. $(A \to (D \to \neg E))$	MP 16, 4
18. $((A \to D) \to (A \to \neg E))$	MP 15, 17
19. $A \to \neg E$	MP 18, 14

Závěr:
Formule $A \to \neg E$ je formálně dokazatelná v teorii T .

ad B) Důkaz, že formule $A \to \neg E$ logicky vyplývá z teorie T:
Podle druhé věty o logickém vyplývání stačí dokázat, že formule

$$((A \to B) \wedge (B \to C) \wedge (C \to D) \wedge (D \to \neg E) \ \wedge \ \neg(A \to \neg E))$$

je kontradikcí. Upravíme-li poslední závorku podle (3.4), dostaneme

$$((A \to B) \wedge (B \to C) \wedge (C \to D) \wedge (D \to \neg E) \ \wedge (A \wedge E)) \ .$$

O tom, že tato formule je kontradikcí, se lze přesvědčit pomocí pravdivostní tabulky, ve které dostaneme hodnotu `false` pro všechna možná přiřazení pravdivostních hodnot proměnným.

Oběma způsoby (formálním důkazem i logickým vyplýváním) jsme tak dokázali tvrzení, že "*jestliže soused mluvil pravdu, pak pan X lhal*" .

3.2 Predikátová logika

Ve výrokové logice jsme detailně zkoumali vlastnosti logických spojek a způsoby zápisu jednoduchých i složených výroků. Nyní budeme pracovat s jazykem predikátové logiky, který kromě spojek obsahuje ještě proměnné, funkční symboly, predikátové symboly a symboly kvantifikátorů. *Predikátová logika* je logický formalismus sloužící především k symbolickému zápisu logického odvozování v matematice. Skutečnosti (poznatky) zapsané symbolikou predikátové logiky – *jazykem predikátové logiky* – vyjádříme podobně jako ve výrokové logice *formulemi* jazyka predikátové logiky. Interpretací symbolů (viz dále), které ve formulích vystupují, získáme výroky, jimž lze přiřadit pravdivostní hodnotu. Danou formuli lze interpretovat mnoha různými způsoby a dostat tak celou třídu výroků, z nichž každý je buď pravdivý nebo nepravdivý. Zvláštním předmětem našeho zájmu bude velmi omezená podtřída formulí, které dávají pravdivé výroky při všech možných interpretacích.

3.2.1 Jazyk predikátové logiky prvního řádu

Ve druhé kapitole tohoto skripta pojednávající o řešení úloh jsme se setkali s pojmem *stav* úlohy, který jsme podle povahy úlohy popisovali datovými strukturami numerické (pole či matice reprezentující stav řešení hlavolamu "8") nebo nenumerické povahy (kotouče A, B, C na kolících hanojských věží). Problém přechodu z jednoho stavu úlohy do jiného jsme však řešili různými algoritmy (viz strategie hledání řešení), čili programově. Jazyk predikátové logiky poskytuje vyjadřovací prostředky nejen pro vyjádření stavů, ale i k popisu pravidel a znalostí. Svými vyjadřovacími schopnostmi je výrazně bohatší než jazyk výrokové logiky, který jsme probrali v předchozím odstavci. Tam jsme elementární výroky chápali jako nedělitelné jednotky, nezkoumali jejich vnitřní stavbu, zajímalo nás pouze, zda jsou pravdivé či nepravdivé. My však poměrně často potřebujeme vyjádřit, že všechny objekty mají nějakou vlastnost (např. všichni lidé jsou smrtelní), nebo že existuje (alespoň jeden) objekt, který má nějakou sledovanou vlastnost (např. že existuje sudé prvočíslo). V jazyce predikátové logiky máme proto k dispozici symboly pro pojmenování popisovaných objektů a také symboly, které pojmenovávají vlastnosti objektů a vztahy mezi objekty.

Pro vyjádření (zápis) vlastností objektů a relací mezi nimi zavádíme tzv. *predikáty* , což ve smyslu jazyka predikátové logiky jsou symboly pro vyjádření obecně n-árních relací. Pro popis vlastností objektů používáme jednomístné (unární) predikáty, pro popis relací mezi objekty predikáty vícemístné (binární, n-ární). Pro množstevní vyjádření pak používáme *kvantifikaci objektů* a pro její symbolický zápis symboly tzv. *kvantifikátorů*. Pro všeobecnou kvantifikaci (platící pro všechny objekty, všechny vlastnosti, všechny relace atd.) používáme symbol *obecného* (někdy říkáme též všeobecného) kvantifikátoru $\forall$, pro vyjádření, že existuje alespoň jeden objekt, alespoň jedna vlastnost,... pak *existenční* kvantifikátor $\exists$. Kvantifikujeme-li v popisech našich poznatků (faktů, situací) pouze objekty, potom hovoříme o použití *predikátové logiky prvního řádu*, kvantifikujeme-li i vlastnosti a relace, resp. predikáty popisující dané vlastnosti a relace, pak podle "hloubky" kvantifikace hovoříme o použití predikátové logiky druhého, resp. vyšších řádů. V našich úlohách se většinou spokojíme pouze s kvantifikací objektů, a proto se v dalším výkladu budeme zabývat pouze predikátovou logikou prvního řádu.

Syntax jazyka predikátové logiky prvního řádu popíšeme *abecedou symbolů*, které bude jazyk používat, a postupem, jak jsou symboly skládány, aby tvořily *slova* jazyka. Tato slova budeme stejně jako u jazyka výrokové logiky nazývat *formulemi jazyka predikátové logiky prvního řádu*. Tam, kde nemůže dojít k nedorozumění, je budeme nazývat jen *formulemi*.

Abecedu jazyka predikátové logiky prvního řádu tvoří:

1. *individuové proměnné*, které budeme značit malými písmeny x, y, $z, \dots$,
2. *individuové konstanty*, které zapisujeme velkými písmeny A, B, $C, \dots$,
3. *funkční symboly*, pro něž budeme používat malá písmena f, g, $h, \dots$; každý funkční symbol má stanoven počet argumentů (četnost), které přijímá,
4. *predikátové symboly*, které označujeme velkými písmeny P, Q, $R, \dots$; každý predikátový symbol má stanoven počet argumentů (místnost), které přijímá,
5. *symboly logických spojek* $\neg, \wedge, \vee, \rightarrow, \Leftrightarrow$ (viz odstavec 3.1),
6. *symboly kvantifikátorů* $\forall$ (obecný kvantifikátor) a $\exists$ (existenční kvantifikátor) – zápis $(\forall x)$ čteme "pro všechna x" nebo "pro každé x", $(\exists x)$ čteme "existuje x" nebo "existuje alespoň jedno x".

Kromě těchto symbolů budeme používat závorky v běžném významu a někdy budeme též používat predikátové symboly nebo individuové konstanty tvořené více písmeny tak, aby byl zřejmý jejich význam – např. predikátový symbol *SMRTELNÝ*, individuovou konstantu $KOSTKA$ apod.

Ze symbolů uvedených v bodech 4 až 6 tvoříme *výrazy* jazyka predikátové logiky prvního řádu:

- *Term* je *(i)* individuová konstanta nebo proměnná,
  (ii) výraz tvaru $f(t_1, t_2, ..., t_n)$, kde f je n-četný funkční symbol (přijímá n argumentů) a $t_1, t_2, ..., t_n$ jsou termy.
- *Atomická formule* je výraz tvaru $P(t_1, t_2, ..., t_m)$, kde P je m-místný predikátový symbol a $t_1, t_2, ..., t_m$ jsou termy.
- *Formule* je *(i)* atomická formule,
  (ii) jeden z výrazů
  $\neg\mathcal{A}$, $\mathcal{A}\wedge\mathcal{B}$, $\mathcal{A}\vee\mathcal{B}$, $\mathcal{A}\rightarrow\mathcal{B}$, $\mathcal{A}\Leftrightarrow\mathcal{B}$, $(\forall x)\,\mathcal{A}$, $(\exists x)\,\mathcal{A}$,
  kde $\mathcal{A}$, $\mathcal{B}$ jsou formule, x je individuová proměnná.
- Individuová proměnná se nazývá *vázaná*, když se vyskytuje v oblasti působnosti (v dosahu) obecného nebo existenčního kvantifikátoru. Individuová proměnná, která není vázaná, se nazývá *volná*.
- *Uzavřená formule* je formule, ve které se nevyskytují volné proměnné.
- *Literál* je atomická formule nebo její negace.
- Disjunkci literálů nazýváme *klauzulí*, někdy též (správněji) *disjunktem*.

Poznámka 3. 1: Jak již bylo řečeno v úvodu odstavce, jazyk predikátové logiky, který nepřipouští kvantifikované predikátové symboly, se nazývá *jazyk predikátové logiky prvního řádu.* Takový jazyk tudíž nemůže obsahovat výrazy jako $(\forall P)\,P(A)$, $(\exists\,Q)\,(P(x)\ \wedge\ Q(y))$ apod.

3.2.2 Interpretace formulí predikátové logiky 1. řádu

Jestliže má jazyk predikátové logiky prvního řádu vypovídat o reálném světě, musíme přiřadit jeho výrazům *významy.* K tomu použijeme relační struktury, které zobrazují zkoumanou skutečnost. Označíme $\mathcal{M}$ relační strukturu, která je tvořena nosičem D , relacemi $R_i^n \subset D^n$ a operacemi $O_i^n : D^n \rightarrow D$. Horní indexy, vyjadřující četnost (aritu) relace nebo operace, budeme tam, kde nedojde k nedorozumění, vynechávat.

Symboly jazyka predikátové logiky prvního řádu interpretujeme takto:

a) Každé individuové konstantě A přiřadíme prvek $A_{\mathcal{M}}$ nosiče D, $A_{\mathcal{M}} \in D$.

b) Každému funkčnímu symbolu f_i přiřadíme operaci O_i stejné četnosti. Termům, které neobsahují proměnné, odpovídají elementy nosiče, které získáme provedením příslušné operace O_i s elementy nosiče přiřazenými podle bodu a).

c) Každému predikátovému symbolu P_i přiřadíme relaci R_i o stejné místnosti.

Relační strukturu $\mathcal{M}$ pak nazveme *interpretační strukturou.*

Někdy volíme symboly jazyka tak, aby vyjadřovaly určitou standardní interpretaci. Například jako individuové konstanty zavedeme obvyklá jména lidí – $KAREL, JOSEF, ...$, jako predikátové konstanty pojmenování vztahů mezi nimi – $BRATR\,(x,y)$, $SYN\,(otec, matka, potomek)$ atd.

Pravdivostní hodnoty přiřazujeme při interpretaci formulí takto:

A) Uzavřená atomická formule $P(t_1, t_2, ..., t_n)$ (atomická formule neobsahující volné proměnné) je *pravdivá v interpretační struktuře* $\mathcal{M}$, jestliže predikátovému symbolu P je přiřazena relace R a prvky nosiče D odpovídající termům $t_1, t_2, ..., t_n$ jsou v relaci R. Pokud atomická formule obsahuje volné proměnné, přiřazení (interpretace) $I(x) \in D$ volných proměnných x prvkům nosiče D, pro které jsou termy $t_1, t_2, ..., t_n$ v relaci, *splňují* P v interpretační struktuře $\mathcal{M}$. Atomická formule P je *pravdivá v interpretační struktuře* $\mathcal{M}$, jestliže ji v této relační struktuře *všechny* interpretace I splňují. Takovou skutečnost zapíšeme $\mathcal{M} \models P$.

B) Uzavřená formule, tvořená atomickými formulemi, logickými spojkami a kvantifikátory, má pravdivostní hodnotu určenou pravdivostními hodnotami atomických formulí, interpretací logických spojek podle obecně známých pravidel a interpretací kvantifikátorů.

Kvantifikovaná formule $(\forall\, x)\, P(x)$ znamená *konjunkci* formulí $P(x)$ přes všechna možná přiřazení symbolu x prvkům nosiče D.

Kvantifikovaná formule $(\exists\, x)\, P(x)$ znamená *disjunkci* formulí $P(x)$ přes všechna možná přiřazení symbolu x prvkům nosiče D.

Následující definice jsou analogií podobných definic, které jsme uvedli pro výrokovou logiku:

- Řekneme, že *formule* $\mathcal{B}$ *logicky vyplývá z formule* $\mathcal{A}$, když formule $\mathcal{B}$ je pravdivá ve všech interpretačních strukturách, ve kterých je pravdivá formule $\mathcal{A}$.
- Formule $\mathcal{A}$, která je pravdivá v každé interpretační struktuře $\mathcal{M}$, se nazývá *logicky pravdivá (tautologie)*.
- Formule $\mathcal{A}$ a $\mathcal{B}$ nazýváme *logicky ekvivalentní*, jestliže $\mathcal{A}$ logicky vyplývá z $\mathcal{B}$ a $\mathcal{B}$ logicky vyplývá z $\mathcal{A}$.

Jazyk predikátové logiky prvního řádu lze, podobně jako jazyk výrokové logiky, vybudovat dedukcí z axiómů. Za axiómy můžeme vzít axiómy výrokové logiky (3. 1) až (3. 3) spolu s dalšími logicky pravdivými formulemi:

$$(\forall\, x)\, \mathcal{A}(x) \;\rightarrow\; \mathcal{A}(t)\ , \tag{3.17}$$

kde term t neobsahuje proměnné vázané v $\mathcal{A}$,

$$(\forall\, x)\, (\mathcal{A} \rightarrow \mathcal{B}) \;\rightarrow\; (\mathcal{A} \rightarrow (\forall\, x)\, \mathcal{B})\ , \tag{3.18}$$

kde formule $\mathcal{A}$ neobsahuje volnou proměnnou x.

Odvozovacími pravidly jsou:

- *pravidlo modus ponens:* $\mathcal{B}$ lze odvodit z $\mathcal{A}$ a z $\mathcal{A} \rightarrow \mathcal{B}$, (3.19)
- *pravidlo generalizace:* $(\forall\, x)\, \mathcal{A}$ lze odvodit z $\mathcal{A}$. (3.20)

Pravidlo generalizace říká, že jestliže formule $\mathcal{A}$ je pravdivá pro blíže nespecifikovanou proměnnou x, je pravdivá pro každou hodnotu této proměnné.

Formule $\mathcal{A}$ predikátové logiky 1. řádu je formálně dokazatelná z axiómů, jestliže existuje důkaz (viz odst. 3. 1) využívající uvedená dvě odvozovací pravidla.

Podobně jako v případě výrokové logiky lze dokázat, že formule jazyka predikátové logiky prvního řádu je formálně dokazatelná z axiómů právě tehdy, jestliže je logicky pravdivá. K uvedeným axiómům můžeme opět přidat další formule, které podobně jako u výrokové logiky nazveme *mimologické axiómy*. Množinu T mimologických axiómů T pak nazveme *teorií* jazyka predikátové logiky 1. řádu. Dedukcí, aplikací pravidel správného usuzování, můžeme z teorie T dokázat další formule. Jestliže formule $\mathcal{A}$ je *formálně dokazatelná z teorie* T (je teorémem v T), píšeme $T \vdash \mathcal{A}$.

Nechť $\mathcal{M}$ je interpretační struktura. Jestliže všechny axiómy teorie T jsou v $\mathcal{M}$ pravdivé, říkáme, že $\mathcal{M}$ je *modelem* teorie T a píšeme $\mathcal{M} \models T$.

Řekneme, že formule $\mathcal{A}$ je *pravdivá v teorii* T, jestliže je pravdivá v každém jejím modelu. Píšeme $T \models \mathcal{A}$. Lze dokázat (podle Gödela, 1930 [Štěpánek82]), že formule jazyka predikátové logiky je formálně dokazatelná z teorie T právě tehdy, když je pravdivá v T. Tedy $T \vdash \mathcal{A}$ právě tehdy, když $T \models \mathcal{A}$.

Jazyk predikátové logiky prvního řádu je podstatně bohatším vyjadřovacím prostředkem než jazyk výrokové logiky, který je v jazyce predikátové logiky obsažen jako speciální podtřída (podmnožina jazyka). Atomické uzavřené formuli v příslušné interpretaci lze totiž přisoudit jednu z pravdivostních hodnot a lze ji tudíž chápat jako výrok.

3.3 Rezoluční metoda a dokazování teorémů

Dokazování formulí jazyka výrokové logiky či jazyka predikátové logiky prvního řádu z axiómů příslušného logického kalkulu výše uvedenými odvozovacími pravidly je sice korektní (říkáme, že je postupem *správného usuzování*), ale nehodí se ke strojovému dokazování, protože z hlediska výpočetní složitosti je neefektivní. Proto v roce 1965 přišel J. A. Robinson s postupem odvozování, který vede k návrhu algoritmů použitelných pro automatické dokazování formulí výrokové a predikátové logiky. Tento postup byl později nazván *rezoluční metodou*, resp. *rezolučním principem* [Manna81].

Důkaz, že vyšetřovaný teorém logicky vyplývá z axiómů (z dané teorie T), je při použití rezoluční metody založen na tvrzení, že *vyplývá-li teorém z dané teorie, pak neexistuje model teorie T, v němž by byla pravdivá negace dokazovaného teorému.* Axiómy z teorie T a negace dokazovaného teorému tak musí vést ke sporu.

3.3.1 Rezoluční metoda ve výrokové logice

Aplikace rezoluční metody na formule *výrokové logiky* vyžaduje, aby formule teorie T byly ve tvaru klauzulí, tj. disjunktů literálů. *Literálem* v jazyce výrokové logiky však nazveme jen výrokové proměnné nebo jejich negace (srovnej s definicí literálu v jazyce predikátové logiky 1. řádu v odst. 3. 2. 1). Jestliže formule teorie T nejsou klauzulemi, nahradíme je logicky ekvivalentními formulemi, které jsou klauzulemi, nebo postupujeme podle následujícího algoritmu: Není-li nějaká formule $\mathcal{A}$ z teorie T klauzulí, upravíme ji na tvar

$$\mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \ldots \wedge \mathcal{C}_{\mathrm{n}} \;,$$

kde $\mathcal{C}_i$ jsou klauzule. Z teorie T vynecháme formuli $\mathcal{A}$ a přidáme klauzule $\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_\mathrm{n}$. Nově vzniklou teorii označíme T' a bude mít tvar

$$T' \;=\; (T - \{\mathcal{A}\}) \;\cup\; \{\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_\mathrm{n}\} \;\; . \tag{3.21}$$

Každý model teorie T' je také modelem teorie T a naopak. V modelu, v němž je formule $\mathcal{A}$ pravdivá, je pravdivá také formule $\mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \ldots \wedge \mathcal{C}_\mathrm{n}$, a proto musí být v tomto modelu pravdivé klauzule $\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_\mathrm{n}$.

V dalším mějme dány dvě klauzule ve tvaru

$$A \vee L_1 \vee L_2 \vee \ldots \vee L_n \;\text{ a }\; \neg A \vee M_1 \vee M_2 \vee \ldots \vee M_m \;,$$

kde A, L_i, M_j jsou literály, $L_i \neq M_j$ pro $i = 1, \ldots, n$, $j = 1, \ldots, m$. Tyto klauzule nazveme *rodičovské*. Rodičovské klauzule určené k rezoluci se vyznačují tím, že obsahují tzv. *pár komplementárních literálů* – jedna z rodičovských klauzulí obsahuje literál A, druhá pak jeho negaci $\neg A$. Rezolucí odvodíme z těchto dvou rodičovských klauzulí novou klauzuli, kterou nazveme jejich *rezolventou*. Rezolventa vznikne disjunkcí rodičovských klauzulí s vynecháním páru komplementárních literálů, čili

$$L_1 \vee L_2 \vee \ldots \vee L_n \vee M_1 \vee M_2 \vee \ldots \vee M_m \;\; .$$

Jestliže k množině klauzulí (teorii T, resp. T') neexistuje žádný model, v němž by negace teorému byla pravdivá, lze odvodit postupným opakováním rezoluční metody prázdnou klauzuli (budeme ji označovat $\square$), která je nepravdivá a představuje spor.

Příklad 3.4: Rezoluční metodou vyřešíme detektivní příběh z příkladu 3.3 (zločin v domě pana X) mnohem rychleji:
Teorie T obsahuje formule

$$T = \{\, A \to B,\, B \to C,\, C \to D,\, D \to \neg E \,\} \quad ,$$

resp.

$$T = \{\, \neg A \vee B,\, \neg B \vee C,\, \neg C \vee D,\, \neg D \vee \neg E \,\}$$

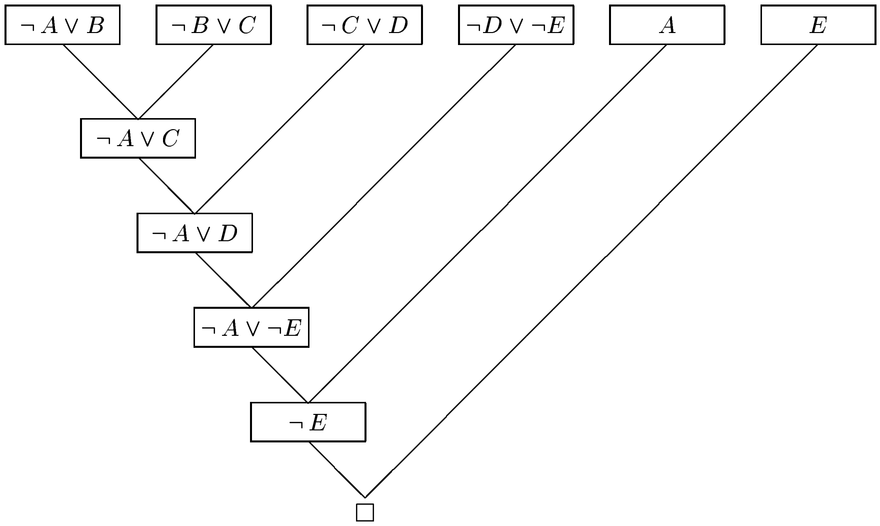
a máme dokázat teorém $A \to \neg E$, resp. $\neg A \vee \neg E$.
Rezoluční metodou ukážeme, že teorie T a negace teorému $\neg\, (\, \neg A \vee \neg E\,)$ vedou ke sporu. Vytvoříme množinu formulí

$$\begin{aligned} T' \;&=\; T \;\cup\; \{\, \neg\, (\, \neg A \vee \neg E\,)\,\} \;= \\ &=\; \{\, \neg A \vee B,\, \neg B \vee C,\, \neg C \vee D,\, \neg D \vee \neg E,\, \neg(\neg A \vee \neg E)\,\} \;. \end{aligned}$$

Poslední formule není klauzulí, a proto ji podle De Morganova pravidla upravíme na $A \wedge E$ a podle (3.21) nahradíme v množině T' samostatnými formulemi A a E , které jsou klauzulemi:

$$T' \;=\; \{\, \neg A \vee B,\, \neg B \vee C,\, \neg C \vee D,\, \neg D \vee \neg E\;,\, A,\;\; E\,\} \;\; .$$

Postup odvození prázdné klauzule $\square$ rezoluční metodou znázorníme *derivačním stromem* – viz obr. 3. 1:



Obr. 3. 1: Derivační strom důkazu pravdivosti teorému $\neg A \vee \neg E$.

3.3.2 Rezoluční metoda v predikátové logice 1. řádu

Použití rezoluční metody ve výrokové logice je poměrně jednoduché – v klauzulích snadno najdeme pár komplementárních literálů, které rezolucí vylučujeme. V jazyce predikátové logiky je tato úloha obtížnější, protože modely tohoto jazyka jsou složitější (relační struktury). K nalezení párů komplementárních literálů je třeba porovnávat odpovídající si termy a hledat vhodné *substituce* v klauzulích. Např. literály $P(x, f(A))$ a $\neg P(B, z)$ se stanou komplementárním párem, pokud proměnná x je rovna konstantě B a proměnná z termu $f(A)$.

Nalezení vhodné substituce s je důležitým krokem rezolučního dokazování formulí jazyka predikátové logiky 1. řádu. Spočívá v nahrazení všech výskytů proměnné x_i termem t_i . Substituci zapíšeme

$$s = \{t_1/x_1, t_2/x_2, ..., t_n/x_n\} .$$

Term t_i , kterým substituujeme, nesmí obsahovat proměnnou x_i , kterou nahrazuje. Substituci s provedenou v klauzuli $\mathcal{C}$ označíme symbolem $\mathcal{C}s$.

Příklad 3.5: Máme dány literály $L_1 = P(x, f(A))$, $L_2 = \neg P(B, z)$. Aby literály L_1 a L_2 tvořily komplementární pár, musíme najít takovou substituci s, pro niž platí $L_1 s = \neg L_2 s$. Tuto podmínku splňuje substituce

$$s = \{ B/x,\ f(A)/z \} .$$

Její aplikací dostaneme

$$L_1 s = \neg L_2 s = P(B, f(A)) .$$

Poznámka 3.2: Pro zjednodušení zápisu (podobně jako v jazyce výrokové logiky) budeme klauzule zapisovat jako množiny literálů. Takže např. klauzuli

$$\mathcal{C} = L_1 \vee L_2 \vee ... \vee L_n \qquad \text{zapíšeme jako} \qquad \mathcal{C} = \{L_1, L_2, ..., L_n\} .$$

Definice 3.1: Literál L' nazveme *instancí* literálu L, jestliže jej získáme nějakou substitucí s v literálu L. Pak platí $L' = Ls$.

Příklad 3.6: Literály $L' = Q(x, A, f(C))$ a $L'' = Q(B, A, f(C))$ jsou instancemi literálu $L = Q(x, A, f(y))$, v němž jsme postupně aplikovali substituce $s_1 = \{C/y\}$ a $s_2 = \{B/x\}$. Čili platí $L' = L\, s_1$ a $L'' = L' s_2 = L\, s_1 s_2$.

Definice 3.2:

Instance, která neobsahuje proměnné, se nazývá *základní instance*.

Definice 3.3: Substituce s se nazývá *unifikátor* klauzule $\mathcal{C} = \{L_1, L_2, ..., L_n\}$, jestliže platí

$$L_1\, s = L_2\, s = ... = L_n\, s .$$

Příklad 3.7: Substituce $s = \{B/x, A/z, C/y, f(C)/w\}$ je unifikátorem klauzule $\mathcal{C} = \{ Q\,(x, A, f(y)),\ Q\,(B, z, w) \}$. Protože oba literály mají po substituci tvar $Q\,(B, A, f(C))$, stávají se základními instancemi. Uvedený unifikátor však není nejjednodušší, substituuje zbytečně mnoho. Jednodušším unifikátorem je substituce $g = \{B/x, A/z, f(y)/w\}$. Její aplikací získáme instance $Q\,(B, A, f(y))$, které nejsou základní, což z hlediska dalšího postupu bývá výhodné (v dalších krocích lze aplikovat další substituce).

Definice 3.4: Unifikátor g klauzule $\mathcal{C}$ se nazývá *nejobecnější unifikátor* této klauzule, jestliže pro libovolný jiný unifikátor s klauzule $\mathcal{C}$ platí, že klauzule $\mathcal{C}s$ je instancí klauzule $\mathcal{C}g$.

Unifikace umožňuje porovnávat literály v klauzulích predikátové logiky prvního řádu. Aby ji však bylo možno provést, musíme formule predikátové logiky 1. řádu na klauzule převést. Převod obecné formule F jazyka predikátové logiky prvního řádu, např.

$$(\forall x)\{P(x) \to \{(\forall y)[P(y) \vee Q(x, y)] \ \wedge \neg (\forall y)[P(y) \to Q(y, x)]\}\},$$

na tvar formule vhodný pro aplikaci rezoluční metody, tj. aby *matice* formule (viz šestý bod) obsahovala pouze konjunkci klauzulí (budeme říkat, že formuli převedeme na *klauzulární tvar*), provedeme v následujících deseti krocích:

1. Není-li formule F uzavřená, tj. obsahuje-li volné proměnné, vytvoříme *existenční uzávěr* formule F tak, že všechny volné proměnné kvantifikujeme existenčním kvantifikátorem [Manna81]. Uzávěr formule zapíšeme $(\exists x)...(\exists z)\ (F)$, kde symboly $x, ..., z$ reprezentují všechny volné proměnné ve formuli F . V našem příkladě formule F neobsahuje žádné volné proměnné, tudíž existenční uzávěr formule není třeba dělat.

2. Ve formuli F se vyskytující ekvivalence a implikace nahradíme ekvivalentními formulemi (viz přehled ekvivalentních formulí). Pro náš příklad dostaneme:

$$(\forall x)\{\neg P(x) \vee \{(\forall y)[P(y) \vee Q(x,y)] \ \wedge \neg (\forall y)[\neg P(y) \vee Q(y,x)]\}\}.$$

3. Upravíme formuli tak, aby logické spojky "negace" ($\neg$) se vztahovaly jen na atomy. To znamená, že spojky $\neg$ přemístíme "dovnitř" jednotlivých logických výrazů, závorek atd. Přitom použijeme pravidel

$$\begin{array}{lll} \neg (\exists x)P(x) & \text{je ekvivalentní} & (\forall x)(\neg P(x)) \ , \\ \neg (\forall x)P(x) & \text{je ekvivalentní} & (\exists x)(\neg P(x)) \ . \end{array}$$

   Kde je to možné, využijeme pro další úpravu De Morganova pravidla.

   Pro náš příklad upravovaná formule přejde do tvaru

$$(\forall x)\{\neg P(x) \vee \{(\forall y)[P(y) \vee Q(x,y)] \wedge (\exists y)[P(y) \ \wedge \neg Q(y,x)]\}\}.$$

4. Přejmenujeme kvantifikované proměnné tak, aby se navzájem lišily, a to proto, aby v rámci platnosti (dosahu) kvantifikátoru nemohlo dojít k záměně kvantifikovaných proměnných, resp. k víceznačnosti. Pro náš příklad tak dostaneme:

$$(\forall x)\{\neg P(x) \vee \{(\forall y)[P(y) \vee Q(x,y)] \wedge (\exists z)[P(z) \ \wedge \neg Q(z,x)]\}\}$$

5. Vyloučíme existenční kvantifikátory. Výraz $(\forall x)(\exists z)\ P(z)$ znamená, že pro každé x existuje z takové, že $P(z)$. Tuto závislost můžeme vyjádřit pomocí tzv. *Skolemovy funkce* $z = f(x)$. Výraz $(\exists z)\ P(z)$ nahradíme Skolemovou funkcí bez argumentů $z = A$, kde A je nějaká individuová konstanta. Tedy místo $(\exists z)\ P(z)$ píšeme $P(A)$. Upravovaná formule z našeho příkladu bude mít potom tvar

$$(\forall x)\{\neg P(x) \vee \{(\forall y)[P(y) \vee Q(x,y)] \wedge [P(f(x)) \ \wedge \neg Q(f(x),x)]\}\}.$$

6. Ve formuli nyní zbývají jen všeobecné (univerzální) kvantifikátory, které přesuneme na začátek formule, do tzv. *prefixu* formule. Zbytek formule následující za prefixem (za seznamem všeobecných kvantifikátorů) nazýváme *maticí* formule. Takový tvar formule nazýváme *prenexním* tvarem formule, resp. *prenexní normální formou* logické formule [Manna81]. Popisovaný příklad bude mít tvar:

$$(\forall x)(\forall y)\{\neg P(x) \vee \{[P(y) \vee Q(x,y)] \wedge [P(f(x)) \ \wedge \neg Q(f(x),x)]\}\}$$

7. Všechny proměnné v matici formule jsou všeobecně (univerzálně) kvantifikovány, na pořadí kvantifikátorů nezáleží. Formule je tak splněna pro všechny hodnoty kvantifikovaných proměnných, takže pro jednoduchost zápisu můžeme prefix formule vynechat. Pro náš příklad dostaneme:

$$\neg P(x) \vee \{[P(y) \vee Q(x,y)] \wedge [P(f(x)) \ \wedge \neg Q(f(x),x)]\}$$

8. Matici formule převedeme na konjunkci klauzulí pomocí ekvivalentních formulí $\mathcal{A} \vee (\mathcal{B} \wedge \mathcal{C}) \quad \Leftrightarrow \quad (\mathcal{A} \vee \mathcal{B}) \wedge (\mathcal{A} \vee \mathcal{C})$. V uváděném příkladu dostáváme

$$[\neg P(x) \vee P(y) \vee Q(x,y)] \wedge [\neg P(x) \vee P(f(x))] \wedge [\neg P(x) \vee \neg Q(f(x),x)].$$

9. Výrazy v hranatých závorkách jsou nyní klauzulemi. Konjunkci klauzulí můžeme nahradit [Manna81] množinou klauzulí

$$\{[\neg P(x) \vee P(y) \vee Q(x,y)], [\neg P(x) \vee P(f(x))], [\neg P(x) \vee \neg Q(f(x),x)]\}.$$

   Klauzule dále můžeme vyjádřit také jako množiny literálů, takže pro náš příklad dostaneme následující množinovou formu zápisu klauzulí:

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

10. Proměnné přejmenujeme tak, aby každá proměnná byla obsažena nejvýše v jedné klauzuli. Přitom vycházíme z následujících ekvivalentních formulí $(\forall x)[P(x) \wedge Q(x)] \Leftrightarrow [(\forall x)P(x) \wedge (\forall y)Q(y)]$. Pro náš příklad dostaneme

$$\{\neg P(x_1), P(y), Q(x_1,y)\},$$
$$\{\neg P(x_2), P(f(x_2))\},$$
$$\{\neg P(x_3), \neg Q(f(x_3),x_3)\}.$$

Tím jsme původní logickou formuli převedli na množinu klauzulí neobsahujících stejné proměnné, zapsaných jako množiny literálů.

Rezoluční metodu v predikátové logice prvého řádu formulujeme potom takto:

Nechť $\mathcal{C}_1 = \{L_{\mathrm{i}}\}$ a $\mathcal{C}_2 = \{M_{\mathrm{j}}\}$, $i = 1,2,...,n$, $j = 1,2,...,m$ jsou klauzule s navzájem různými proměnnými (viz výše uvedený bod 10). Dále předpokládejme, že $\mathcal{C}_1^{'}$ a $\mathcal{C}_2^{'}$ jsou klauzule, $\mathcal{C}_1^{'} \subseteq \mathcal{C}_1$ a $\mathcal{C}_2^{'} \subseteq \mathcal{C}_2$, $\mathcal{C}_1^{'} = \{L_{\mathrm{i}}^{'}\}$ a $\mathcal{C}_2^{'} = \{M_{\mathrm{j}}^{'}\}$, g je nejobecnější unifikátor klauzule $\mathcal{C}_1^{'} \cup \{\neg M_{\mathrm{j}}^{'}\}$. Pak klauzule

$$(\mathcal{C}_1 - \mathcal{C}_1^{'})\, g \ \cup \ (\mathcal{C}_2 - \mathcal{C}_2^{'})\, g$$

je *rezolventou* klauzulí $\mathcal{C}_1$ a $\mathcal{C}_2$.

Použití rezoluční metody pro dokazování logické pravdivosti formulí jazyka predikátové logiky prvního řádu a současně použití predikátové logiky k reprezentaci znalostí si ilustrujme pomocí následujícího příkladu:

Příklad 3.8: Pozorováním vymezené problémové oblasti (zde zoologie) jsme zjistili následující poznatky:

1. Každá ryba má žábry.
2. Savci nemají žábry.
3. Někteří savci dovedou plavat.

Úkolem je dokázat tvrzení, že:

4. Někteří živočichové dovedou plavat a přitom nejsou ryby.

V jazyce predikátové logiky prvého řádu vyjádříme tyto poznatky následovně:
– individuová proměnná x bude reprezentovat živočichy,
– predikátový symbol $RYBA$ bude přiřazen vlastnosti "být rybou",
– predikátový symbol $MÁ\_ŽÁBRY$ označuje živočichy, kteří mají žábry,
– predikát $SAVEC(x)$ představuje, že živočich x je savec a
– predikát $PLAVE(x)$ vyjadřuje, že živočich x umí plavat.
Atomická formule $RYBA(x)$ bude pravdivá tehdy a jen tehdy, pokud x označuje rybu, $PLAVE(x)$ bude pravdivá, pokud živočich dovede plavat atd. Např. $RYBA(KAPR)$ je pravdivá, $RYBA(PES)$ nepravdivá, avšak atomické formule $PLAVE(RYBA)$ i $PLAVE(PES)$ jsou obě pravdivé.

Formule predikátové logiky prvého řádu pak zapíšeme:

(1) $(\forall x)(RYBA(x) \rightarrow MÁ\_ŽÁBRY\,(x))$
(2) $(\forall x)(SAVEC(x) \rightarrow \neg\; MÁ\_ŽÁBRY\,(x))$
(3) $(\exists x)(SAVEC(x)\ \wedge PLAVE(x))$

Úkolem je dokázat tvrzení teorému

(4) $(\exists x)(PLAVE(x)\ \wedge \neg\, RYBA(x))$.

V odstavci 3. 2 bylo řečeno, že tvrzení teorému je pravdivé, pokud formule

$$[(\forall x)(RYBA(x) \rightarrow MÁ\_ŽÁBRY\,(x))\ \wedge\ (\forall x)(SAVEC(x) \rightarrow \neg\ MÁ\_ŽÁBRY\,(x)) \wedge \\ \wedge\ (\exists x)(SAVEC(x)\ \wedge PLAVE(x))]\ \rightarrow\ (\exists x)(PLAVE(x)\ \wedge \neg\, RYBA(x))$$

je logicky pravdivá, resp. formule

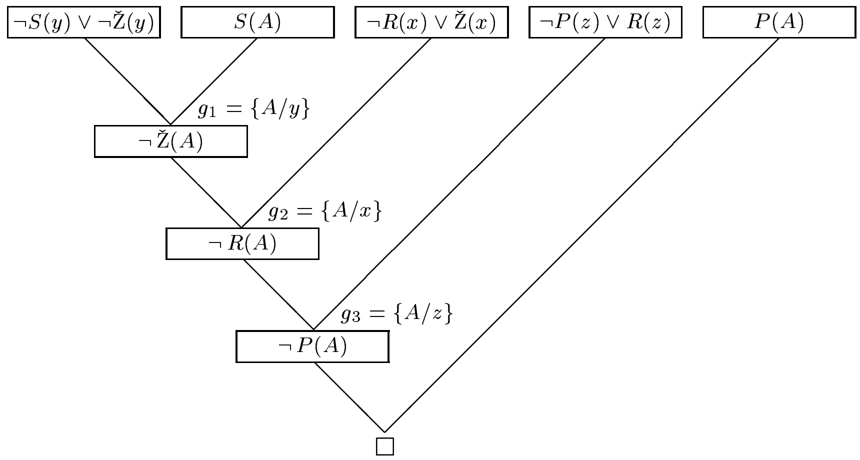
$$[(\forall x)(RYBA(x) \rightarrow MÁ\_ŽÁBRY\,(x))\ \wedge\ (\forall x)(SAVEC(x) \rightarrow \neg\ MÁ\_ŽÁBRY\,(x)) \wedge \\ \wedge\ (\exists x)(SAVEC(x)\ \wedge PLAVE(x))]\ \wedge\ \neg\,(\exists x)(PLAVE(x)\ \wedge \neg\, RYBA(x))$$

je nesplnitelná pro jakékoli x (není logicky pravdivá pro žádnou hodnotu x).

Pro aplikaci rezoluční metody využijeme druhý tvar formule, tzn. budeme dokazovat její nesplnitelnost (nepravdivost). Převedením formulí (1) až (4) do klauzulárního tvaru pomocí výše uvedeného postupu dostaneme následující množinu klauzulí:

(1) $\neg\, RYBA(x)\ \vee\ MÁ\_ŽÁBRY\,(x)$
(2) $\neg\, SAVEC(y)\ \vee\ \neg\ MÁ\_ŽÁBRY\,(y)$
(3a) $SAVEC(A)$
(3b) $PLAVE(A)$
(4) $\neg\, PLAVE(z)\ \vee\ RYBA(z)$

Důkaz odvodíme opakovaným použitím rezoluční metody na klauzule (1) až (4) a rozšiřováním této množiny klauzulí o rezolventy, dokud neodvodíme prázdnou klauzuli $\Box$. Jeden z možných derivačních stromů aplikace rezoluční metody je uveden na obr. 3. 2.



Obr. 3. 2: Derivační strom důkazu pravdivosti teorému z příkladu 3. 8

Rezoluční metoda umožňuje dokazování teorémů z dané výchozí teorie. Není však obecným předpisem k řešení zadaného problému, protože např. jednoznačně neurčuje, které klauzule v daném kroku vzít (vybrat) pro rezoluci jako rodičovské. V minulém kroku jsme jako výchozí rodičovské klauzule zvolili klauzule (2) a (3a). Stejně tak jsme ale mohli zahájit klauzulemi (1) a (2), (3b) a (4) nebo (1) a (4). Generování všech možných rezolucí vede ke kombinatorické explozi a proto je třeba zvolit vhodnou řídicí strategii.

Nejjednodušší možností je volba prohledávání do šířky. V první úrovni vytvoříme všechny možné rezolventy z klauzulí ve výchozí množině, v druhé úrovni rezolventy, jejichž alespoň jedna rodičovská klauzule je z první úrovně, ve třetí úrovni musí být alespoň jedna rodičovská klauzule z druhé úrovně atd. Už z tohoto slovního popisu algoritmu hledání dvojic klauzulí určených k rezoluci je zřejmé, že strategie prohledávání do šířky je značně neefektivní (což ostatně již bylo konstatováno v předchozí kapitole). Proto jiná strategie např. požaduje, aby jako rodičovská byla přednostně vybrána klauzule, kterou tvoří jediný literál, neboť rezolventa bude obsahovat méně literálů než druhá z rodičovských klauzulí. V [Nilsson82] lze nalézt strategii, která požaduje, aby alespoň jedna z rodičovských klauzulí byla negací odvozovaného teorému nebo byla z této klauzule v některém z předchozích kroků odvozena. Mezi efektivnější postupy pak patří strategie, které minimalizují prohledávaný derivační strom [Nilsson82], [Rich83], [Schalkoff90].

Rezoluční metoda je teoretickým základem programovacího jazyka Prolog, který reprezentuje poznatky vyjádřené (zapsané) ve tvaru klauzulí s nejvýše jedním pozitivním (nenegovaným) literálem a dnes patří k nejrozšířenějším prostředkům pro logické programování.

3.4 Základy logického programování

Logické programování používá logiky jako výpočtového formalismu; formule predikátové logiky je sama o sobě programem k vyhodnocení relace, kterou popisuje. Programovací jazyk *PROLOG* (PROgramming in LOGic) je implementací tohoto formalismu.

Rezoluční metoda dokazování logické pravdivosti formulí pracuje s libovolnými formulemi jazyka predikátové logiky, ale často se potýká s kombinatorickou explozí možných rodičovských párů rezolvent. Výpočtová složitost obecné rezoluční metody je superexponenciální. Výkonné algoritmy dokazování teorémů se proto opírají o heuristiky založené na rozsáhlých znalostech z problémové oblasti. Takový postup je však nevhodný pro konstrukci programovacího jazyka. Při návrhu Prologu bylo proto přijato řešení spočívající v omezení tvaru klauzulí vstupujících do rezoluce. Vhodným tvarem klauzulí jsou tzv. *Hornovy klauzule*.

3.4.1 Hornovy klauzule

Hornovy klauzule jsou klauzule, které obsahují nejvýše jeden pozitivní literál. Jsou-li P, $Q_1, \ldots, Q_n$ atomické formule (dále budeme říkat jen *atomy*) jazyka predikátové logiky 1. řádu ($n \geq 0$), potom formule

$$\{P, \neg Q_1, \ldots, \neg Q_n\} \qquad \text{a} \qquad \{\neg Q_1, \ldots, \neg Q_n\}$$

jsou *Hornovy klauzule* . Zapíšeme-li první z nich jako disjunkci literálů, dostaneme formuli

$$P, \vee \neg Q_1 \vee \ldots \vee \neg Q_n ,$$

což je formule ekvivalentní s implikací

$$P \leftarrow Q_1 \wedge \ldots \wedge Q_n ,$$

ve které všechny atomy vystupují v aserci ("pozitivně") a symbol ' $\leftarrow$ ' odděluje negativní a pozitivní část klauzule. V tomto tvaru klauzule vyjadřuje postačující podmínku "P platí, jestliže platí $Q_1, \ldots, Q_n$". Proto se často symbol konjunkce $\wedge$ nahrazuje čárkou. Pak získáme zápis

$$P \leftarrow Q_1, \ldots, Q_n .$$

Použijeme-li stejný přepis pro druhou klauzuli, dostaneme poněkud záhadné vyjádření ve tvaru

$$\Box \leftarrow Q_1, \ldots, Q_n ,$$

které lze chápat jako posloupnost dotazů na atomy uvedené v posloupnosti $Q_1, \ldots, Q_n$.

Zápis, který používá jazyk Prolog, nahrazuje šipku v uvedených klauzulích symbolem `:-` , resp. `?-` v klauzuli bez pozitivního literálu. Tím je de facto naznačeno, že klauzule neobsahující pozitivní literál je vlastně *dotazem*. Výše uvedené klauzule pak v Prologu zapíšeme jako

$$P \;:-\; Q_1, \ldots, Q_n. ,$$
$$?-\; Q_1, \ldots, Q_n. .$$

Příklad 3.8: Důkaz skutečnosti, že je-li číslo 3 dělitelem čísla 6 a číslo 6 dělitelem čísla 18, pak také číslo 3 je dělitelem čísla 18, zapíšeme v Prologu následující posloupností příkazů:

```
dělitel(3,6).
dělitel(6,18).
dělitel(X,Z) :- dělitel(X,Y), dělitel(Y,Z).
```

V prvních dvou klauzulích (příkazech) vynecháváme znak `:-` , protože jde o konkluzivní úsudek bez premis (neobsahují negativní literál). Chceme-li dokázat, že 3 je dělitelem 18, položíme dotaz ve tvaru

```
?- dělitel(3,18).
```

Výše uvedené tři klauzule použijeme jako program pro dokázání tvrzení, resp. pro odvození odpovědi. Substitucí $\sigma_1 = \{3/X,\ 18/Z\}$ unifikujeme atom v dotazu s pozitivním atomem v třetím řádku (příkazu) teorie (unifikace s předcházejícími dvěma příkazy (řádky programu) není možná) a jako rezolventu dostaneme klauzuli

```
?- dělitel(3,Y), dělitel(Y,18).
```

K odvození prázdné klauzule □ musíme rezolvovat oba atomy získané klauzule s prvními dvěma příkazy výše zapsaného programu; začneme levým atomem. Při použití substituce $\sigma_2 = \{ 6 / Y \}$ dostaneme rezolucí levého atomu s prvním řádkem programu novou rezolventu

```
?- dělitel(6,18). ,
```

která další rezolucí s druhým řádkem teorie dává □ jako rezolventu. Od Prologu pak dostaneme kladnou odpověď na položený dotaz v podobě řetězce `yes` .

3.4.2 Logické programy

Pokud Hornovy klauzule obsahují pozitivní atom, záleží dále na počtu negativních atomů. Je-li tento počet nulový, dostáváme klauzuli

P .

Symbol '.' (tečka) je symbolem ukončujícím každý příkaz Prologu a má funkci podobnou jako ';' v Pascalu; nesmíme tedy zapomenout každý příkaz, včetně dotazů, ukončit tečkou.

Je-li dále např. $n = 2$, dostáváme

$P \ :- \ Q_1 \ , \ Q_2 \, .$

První typ klauzule (příkazu) vyjadřuje jednoduchý fakt a říkáme mu *nepodmíněný příkaz* nebo někdy také *axióm*. Druhý typ vyjadřuje skutečnost, že P vyplývá z premis Q_1 a Q_2, a říkáme mu *podmíněný příkaz* nebo také *procedura*. Nepodmíněné příkazy deklarují fakta, která jsou pravdivá bez ohledu na další předpoklady, podmíněné příkazy jsou úsudky, která deklarují pravdivost určitého faktu, jsou-li pravdivé zapsané podmínky (předpoklady) – viz dále.

Logický program je potom libovolná množina podmíněných a / nebo nepodmíněných příkazů.

Klauzule, které neobsahují pozitivní literál, se vztahují k provádění logických programů (iniciují spuštění procesu rezolučního dokazování). Je-li $n > 0$, klauzule

$? - \ Q_1 \ \ldots , \ Q_n \, .$

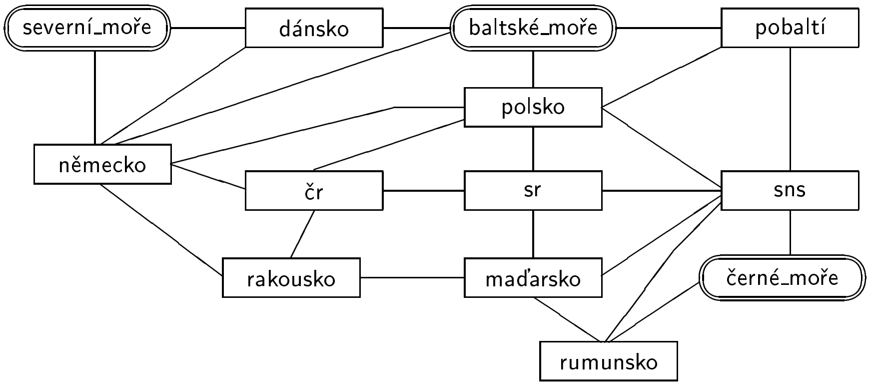
určuje posloupnost dotazů. Říkáme jí *cílová klauzule*, její atomy nazýváme *cíle*. Je-li $n = 0$, dostáváme prázdnou klauzuli, která znamená konec výpočtu.

Postup vytváření logického programu v jazyce Prolog si ukážeme na následujícím zeměpisném příkladě:

Příklad 3.9: Vycházejme ze zjednodušené mapy několika Evropských zemí z obrázku 3.3. Skutečnost, že Německo má s Českou republikou společnou hranici, vyjádříme v Prologu nepodmíněným příkazem

```
sousedí (německo, čr).
```

Pro vztah (relaci) "mají společnou hranici" jsme použili predikát (přesně název predikátu) `sousedí`, `německo` a `čr` jsou jeho argumenty. Protože se zde jedná o konstantní argumenty (v terminologii predikátové logiky o individuové konstanty), zapisujeme je *malými písmeny*, resp. musí v Prologu začínat malým písmenem (pozor – odlišnost od definice jazyka predikátové logiky !).



Obr. 3.3: Zjednodušená struktura evropských států použitá v příkladu 3.9.

Relace popisující společné hranice mezi všemi státy z obr. 3.3 zapíšeme pak jako následující posloupnost nepodmíněných příkazů:

```
sousedí (dánsko, německo).
sousedí (německo, polsko).
sousedí (německo, čr).
sousedí (německo, rakousko).
sousedí (polsko, pobaltí).
sousedí (pobaltí, sns).
sousedí (polsko, sns).
sousedí (čr, sr).
sousedí (čr, polsko).
sousedí (čr, rakousko).
sousedí (sr, polsko).
```

```
sousedí (sr, sns).
sousedí (sr, maďarsko).
sousedí (rakousko, maďarsko).
sousedí (maďarsko, sns).
sousedí (maďarsko, rumunsko).
sousedí (rumunsko, sns).
```

Tento první jednoduchý program v Prologu vyjadřuje všechna fakta o relacích sousedství vyobrazených evropských států. Nyní se můžeme nejčastěji interpretačního systému Prologu dotazovat, které evropské státy spolu sousedí či nikoli. Takže na dotaz

```
?- sousedí (rakousko, maďarsko).
```

Prolog odpoví `yes` jakmile zjistí, že jde o fakt uvedený v námi zadaném programu. Položíme-li však dotaz

```
?- sousedí (dánsko, maďarsko).
```
,
Prolog odpoví `no` , protože o společné hranici mezi zadanými státy není z posloupnosti nepodmíněných příkazů nic známo. Stejnou odpověď dostaneme na dotaz

```
?- sousedí (francie, maďarsko).
```
,
neboť o objektu popsaném konstantou "`francie`" Prolog dosud "neslyšel". Podobně dostaneme negativní odpověď i na dotaz

```
?- sousedí (maďarsko, rakousko).
```
,
protože skutečnost uvedenou v dotazu nelze z našeho programu odvodit. Mít společnou hranici, resp. spolu sousedit je sice vztah symetrický, ale v posloupnosti nepodmíněných příkazů jsme toto neuvedli. Prolog ctí pořadí argumentů a fakt

```
sousedí (rakousko, maďarsko).
```

je pro něj něco jiného než fakt

```
sousedí (maďarsko, rakousko).
```
,
který není v programu uveden. Náš program definuje relaci `sousedí` jinak, než odpovídá symetrickému vztahu "mít společnou hranici". Tento rozdíl můžeme zatím napravit tím, že ke každému nepodmíněnému příkazu výše uvedeného programu přidáme jeho symetrický protějšek. Dostaneme tak program, který se skládá z dvojnásobného počtu nepodmíněných příkazů. Později si ukážeme, že symetrii relace `sousedí` můžeme daleko snáze vyjádřit jediným podmíněným příkazem.

Můžeme se také zeptat, jaké sousedy má Německo:

```
?- sousedí (německo, X).
```

V tomto případě nečekáme odpověď ano či ne , ale (zatím neznámé) hodnoty proměnné X (jména proměnných v Prologu vždy začínají velkým písmenem!). Prolog na výše položený dotaz odpoví

```
X = polsko  .
```

To však není jediná možná odpověď. Chceme-li získat další řešení, zadáme na klávesnici středník a dostaneme

```
X = čr;
X = rakousko;
no
```

Poslední odpověď nám oznamuje, že byly vyčerpány všechny možnosti.

Dotaz může obsahovat i více proměnných. Lze se např. dotázat, kdo s kým sousedí pomocí příkazu

```
?- sousedí (X, Y).
```

Hledáme všechna X, Y taková, pro něž platí relace *sousedí* (X, Y) . Prolog vyhledá všechny takové dvojice jednu po druhé a odpoví

```
X = dánsko
Y = německo;
X = německo
Y = polsko;
X = německo
Y = čr;
. . . . . .
```

Z našeho programu bychom dostali celkem sedmnáct odpovědí; jejich vyhledávání se dá kdykoli ukončit tím, že místo středníku napíšeme tečku.

Můžeme položit i složitější dotaz, např. "Která země leží mezi Českou republikou a Maďarskem ?" Hledat budeme takovou zemi Z , pro kterou platí relace *sousedí (čr, Z)* a *sousedí (Z, maďarsko)* . Dotaz zapíšeme v Prologu jako konjunkci (posloupnost) dvou jednoduchých dotazů

```
?- sousedí (čr, Z), sousedí (Z, maďarsko).
```

Dostaneme odpověď

```
Z = sr;
```

Napíšeme-li středník, dostaneme ještě jednu odpověď, a to

```
Z = rakousko
```

Náš zeměpisný program rozšíříme dále o další fakta, která přidáme v podobě nepodmíněných příkazů. Přímořské státy mají část své hranice tvořenu mořem; tuto skutečnost vyjádříme příkazovou sekvencí

```
sousedí (dánsko, severní_moře).
sousedí (dánsko, baltské_moře).
sousedí (německo, severní_moře).
sousedí (německo, baltské_moře).
```

```
sousedí (polsko, baltské moře).
sousedí (pobaltí, baltské moře).
sousedí (sns, černé moře).
sousedí (rumunsko, černé_moře).
```

Atomy *severní_moře, baltské_moře* a *černé_moře* nesmějí být podobně jako v jiných programovacích jazycích rozděleny mezerou. Pomůžeme si tedy např. znakem podtržení a nikoli pomlčkou, která se v Prologu používá v jiných souvislostech (zatím jsme se s ní setkali v kombinaci znaků ?−).
Skutečnost, že atomy *severní_moře, baltské_moře* a *černé_moře* jsou v relaci *sousedí* vlastní jména moří a nikoli jména států, musíme Prologu "vysvětlit" přidáním tří dalších nepodmíněných příkazů

```
moře (severní_moře).
moře (baltské_moře).
moře (černé_moře).
```

Přitom malá písmena, kterými atomy začínají, informují o tom, že jde o *vlastní jména* (konstanty) a ne o *proměnné* . Nově zavedená relace *moře* , která má jen jeden argument (je unární), definuje obvykle vlastnosti, které může mít jednotlivý objekt, zatímco binární relace (např. *sousedí*) definují vztahy mezi uspořádanými dvojicemi objektů. Kromě uvedených unárních a binárních relací, které jsou nejčastější, můžeme v Prologu používat i relace vícečetné (obecně n–ární).

Výše jsme uvedli, že vztah "mít společnou hranici" je symetrický, avšak Prolog tím, že respektuje pořadí argumentů, tuto symetrii vztahu "nezná". Platí, že např. nepodmíněný příkaz

```
sousedí (německo, čr).
```

nezahrnuje skutečnost

```
sousedí (čr, německo).
```

Skutečnost, že jestliže země Y má společnou hranici se zemí X , pak také země X má společnou hranici se zemí Y , resp. symetrii relace *sousedí* proto musíme vyjádřit příkazem, který terminologií blízkou jazyku predikátové logiky vyjádříme slovně asi takto:

Pro každé X a každé Y X sousedí s Y ,
jestliže Y sousedí s X .

Takovým příkazem je podmíněný příkaz , který zapíšeme

```
sousedí (X, Y) :- sousedí (Y, X).
```

Z pohledu jazyka predikátové logiky je podmíněný příkaz zápisem úsudku: závěr *sousedí (X, Y)* je pravdivý, jestliže platí předpoklad *sousedí (Y, X)* . Přitom řetězec ':–' zastupuje šipku znázorňující vyplývání, která vede od předpokladu na pravé straně podmíněného příkazu k závěru úsudku, který stojí nalevo od ní, tedy **opačně**, než jsme zvyklí vyplývání zapisovat v jazyce predikátové logiky. V terminologii jazyka Prolog je podmíněný příkaz tvořen *hlavou* příkazu *sousedí (X, Y)* stojící *nalevo* od symbolu ':–' a *tělem* podmíněného příkazu *sousedí (Y, X)* , které stojí napravo od ':–' a které může být tvořeno jednou nebo více premisami. Proměnné v Prologu jsou automaticky *univerzálně kvantifikovány*.

Podmíněnými příkazy se dají deklarovat další vlastnosti relací definovaných výčtem základních faktů (nepodmíněných příkazů), aniž bychom definici relace zdlouhavě doplňovali o další nepodmíněné příkazy. Např. skutečnost, že některé státy leží u moře, můžeme popsat novou relací *u_moře* , kterou definujeme podmíněným příkazem

```
u_moře (Z, M) :- sousedí (Z, M), moře (M).
```

Pak na dotaz

```
?- u_moře (polsko, baltské_moře).
```

dostaneme kladnou odpověď, neboť cíl *u_moře (polsko, baltské_moře)* je splněn, jestliže dostaneme kladnou odpověď na dotaz *?– sousedí (Z, M), moře (M).* Při vyhodnocování této podmínky postupuje Prolog zleva doprava, tzn. že nejprve prověří platnost první podmínky a v případě její logické pravdivosti pokračuje testováním druhé podmínky. Obdobně na dotaz

```
?- u_moře (dánsko, Q).
```

dostaneme odpověď

```
Q = severní_moře;
Q = baltské_moře;
no
```

Binární relace *u_moře* spojuje jméno země s jménem moře, u kterého tato země leží. Zajímají-li nás přímořské státy bez ohledu na to, u jakého moře leží, můžeme definovat unární relaci *přímořský_stát* podmíněným příkazem

```
přímořský_stát (S) :- u_moře (S, M).
```

Na dotaz

```
?- přímořský_stát (S).
```

dostaneme odpověď

```
S = dánsko;
S = německo;
S = polsko;
S = pobaltí;
S = sns;
S = rumunsko;
no
```

Ne vždy je však nutno novou relaci zavádět. Chceme-li v našem příkladu zjistit, zda nějaký stát S leží u moře a na jménu moře nezáleží, použijeme již definovanou relaci *u_moře*, v níž místo proměnné M použijeme tzv. *anonymní proměnnou*, kterou v Prologu zapisujeme izolovaně stojícím znakem podtržení. Anonymní proměnná je proměnná, jejíž hodnoty nejsou dále zpracovávány, nedefinujeme její jméno (zastupuje ho znak '_') a její hodnoty při splnění cíle nevystupují. Čili na dotaz ve tvaru

```
?- u_moře (S, _ ).   ,
```

dostaneme stejnou odpověď jako v případě zavedení relace *přímořský_stát*.

Dva státy, které leží u stejného moře, mohou mít výhodné námořní spojení. Tuto relaci, kterou slovně vyjádříme "pro každé $Z1$ a každé $Z2$ je námořní doprava výhodná, jestliže obě země $Z1$ i $Z2$ leží u stejného moře M ", zapíšeme v Prologu pomocí jediného podmíněného příkazu

```
námořní_doprava (Z1, Z2) :- u_moře (Z1, M), u_moře (Z2, M).
```

V obou podmínkách v těle posledního příkazu musíme uvést stejnou proměnnou M pro jméno moře, i když v dotazech typu

```
?- námořní_doprava (dánsko, pobaltí).
```

jméno moře nevystupuje. Definovali jsme ale podmínku, že námořní spojení je výhodné jen v případě, že oba přímořské státy leží u stejného moře. Kdybychom použili anonymní proměnnou, neobdržíme správnou odpověď, neboť dva výskyty symbolu '_' vždy znamenají dvě různé anonymní proměnné, čímž bychom podmínku polohy států při stejném moři potlačili.

Výrazné rozšíření výrazových možností Prologu přináší možnost *rekurzívní definice relací*. Tuto definici si opět ukážeme na příkladě relace *přechod*, která deklaruje možnost přechodu z jedné země do druhé. Jsou-li $Z1$ a $Z2$ dvě sousední země, jde o nejjednodušší způsob přechodu ze $Z1$ do $Z2$. Slovně můžeme podstatu přechodu z jedné země do druhé vyjádřit

Pro každé $Z1$ a každé $Z2$ je možné přejít ze $Z1$ do $Z2$,
jestliže $Z1$ sousedí se $Z2$.

Jelikož z pohledu predikátové logiky se jedná o jednoduchý úsudek, zapíšeme uvedenou skutečnost v Prologu jedním podmíněným příkazem

```
přechod (Z1, Z2) :- sousedí (Z1, Z2).
```

To však není jediný způsob přechodu ze země do země. Pokud $Z1$ nesousedí se $Z2$, ale obě země sousedí s některou třetí zemí Y, lze ze $Z1$ přejít do $Z2$ tranzitem přes Y. Takovou skutečnost vyjádříme v Prologu příkazem

```
přechod (Z1, Z2) :- sousedí (Z1, Y), sousedí (Y, Z2).
```

Je zřejmé, že řetěz tranzitních zemí, přes které přejdeme ze $Z1$ do $Z2$ může být i delší, např.:

```
přechod(Z1,Z2) :- sousedí(Z1,Y1),sousedí(Y1,Y2),sousedí(Y2,Z2).
```

Jeho délka není předem nijak omezena, avšak jde o velmi nepohodlný způsob zápisu. Proto relaci *přechod* budeme v Prologu definovat zcela korektně a pohodlně bez ohledu na počet zemí, které mezi $Z1$ a $Z2$ leží. Nejprve popíšeme nejjednodušší případ přechodu mezi sousedními zeměmi způsobem uvedeným výše

```
přechod (Z1, Z2) :- sousedí (Z1, Z2).
```

a potom jedním rekurzívním podmíněným příkazem popíšeme všechny složitější případy přechodu:

```
přechod (Z1, Z2) :- sousedí (Z1, Y), přechod (Y, Z2).
```

Ačkoli problém rekurze není určitě čtenáři neznámý, pokusme se nastínit, jak bude Prologovský systém dotaz na přechod mezi zeměmi vyhodnocovat: Cíl, např. `?- přechod (německo, Z).` , rozložíme na dva dílčí cíle (viz pravá strana posledního podmíněného příkazu). První, jednodušší krok uděláme "sami" tím, že vykročíme do některé sousední země Y. Pokud země Y není naší cílovou zemí $Z2$, necháme za nás další krok zdánlivě udělat "někoho jiného". To však je skutečně jen zdání – v dalším kroku, pokud Y nesousedí se $Z2$, opět "sami" vykročíme do další sousední země, např. $Y2$, a z ní budeme dále hledat novou možnost tranzitu do $Z2$ atd. Takže v našem zeměpisném příkladě dostaneme na dotaz `?- přechod (německo, Z).` postupně odpovědi

```
Z = polsko;
Z = čr;
Z = rakousko;
Z = dánsko;
Z = pobaltí;
Z = sns;
Z = sr;
Z = maďarsko;
Z = rumunsko;
no
```

Na závěr odstavce si ukažme použití Prologu pro řešení úlohy uvedené v příkladu 3. 8. Abychom mohli zapsat tam uvedenou posloupnost klauzulí, musíme nejprve definovat některá vždy platná (vždy pravdivá) fakta nepodmíněnými příkazy

```
živočich(pes).
živočich(kočka).
živočich(kapr).
má_žábry(kapr).
plave(pes).
neplave(kočka).
```

Pak teprve můžeme zapsat klauzule podmíněnými příkazy

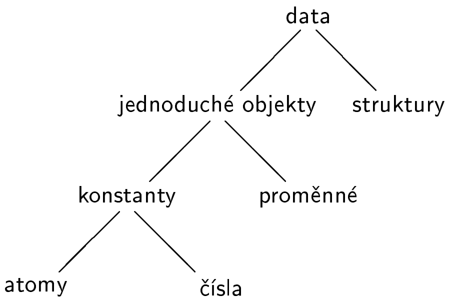
```
neplave(U) :- not(plave(U)).
savec(X) :- živočich(X), not(má_žábry(X)).
ryba(Y) :- živočich(Y), má_žábry(Y).
plave(Z) :- ryba(Z).
```

a platnost tvrzení ověřit položením dotazu

```
?- plave(V), not(ryba(V)).
```

3.4.3 Datové typy a numerické výpočty v Prologu

Datové typy, které lze v Prologu používat, můžeme rozdělit podle schématu na obr. 3. 4:



Obr. 3. 4: Hierarchie datových typů v Prologu

Datové typy v Prologu rozdělujeme na *jednoduché* (tohoto typu jsou všechna data, s nimiž jsme se dosud setkali) a *složené*, kterým říkáme *struktury*. Použité datové typy není nutno v programech deklarovat, Prolog je rozeznává podle jejich syntaxe (způsobu zápisu v programu).

Jednoduché datové typy dělíme dále na *konstantní (atomy)* – v našem příkladě vystupovaly jako *vlastní jména objektů*, např. *čr, černé_moře* a *proměnné*, které vyjadřovaly *obecná jména* tříd objektů (*Stát, Moře*) a zapisovali jsme je s minimálně jedním velkým písmenem na začátku (což je jedno z implicitních syntaktických pravidel, podle nichž Prolog datové objekty rozlišuje). Atomy lze vyjádřit trojím způsobem jako

– řetězce písmen, číslic a znaků _ (podtržení) začínající malým písmenem,
– řetězce speciálních znaků, např. ?–, :–, :–:, \==, =* ap.,
– řetězce znaků uzavřené v apostrofech, např. *'Německo', 'KAREL'*.

Třetí vyjádření je nezbytné v případě, že chceme objekty označovat běžným způsobem, tzn. například používat vlastní jména začínající velkým písmenem.

Jedná se o klasické znakové řetězce, u nichž "zadní" apostrof jednoznačně ukončuje řetězec, takže mezi apostrofy je možné používat i "mezeru", tzn. znakový řetězec *'Karel IV'* můžeme použít pro označení atomu spojeného s významem tohoto vladaře.
Mezi atomy počítáme nejen vlastní jména konstantních datových objektů, nýbrž i symbolická jména relací. Takže i řetězce *sousedí, u_moře, přímořský_stát* jsou také atomy.

Dalším jednoduchým datovým typem v Prologu jsou *čísla*. Tato syntaktická kategorie je silně závislá na použité implementaci Prologu. Standardní implementace umožňují používat pouze *přirozená čísla* z velmi omezeného rozsahu, běžné implementace pak klasická *celá čísla* (tj. včetně záporných čísel) v běžném rozsahu. Racionální čísla ("pascalský" typ *real*) jsou dostupná jen v některých "větších" implementacích.

Proměnné označujeme řetězci znaků začínajícími velkým písmenem. Samotný znak _ (podtržení) označuje *anonymní proměnnou*, o jejímž významu jsme již hovořili. *Oborem platnosti* každé proměnné je *pouze klauzule, ve které vystupuje!* Jestliže se stejné symbolické jméno proměnné vyskytuje ve dvou různých klauzulích, Prolog je interpretuje jako jména *dvou různých proměnných*. Tím se svými vlastnostmi podstatně odlišují od konstant, které v celém programu (ve všech klauzulích) označují stejný datový objekt.

Složené datové objekty neboli *struktury* se skládají z více datových položek a rozdělujeme je na struktury definované *funktory*, které si zpravidla znázorňujeme jako stromové struktury, *seznamy*, které známe z jiných programovacích jazyků, a eventuálně další datové struktury, jejichž sortiment je vždy závislý na konkrétní implementaci Prologu. Popis a použití struktur v Prologu je však nad rámec tohoto skripta, pro jejich studium lze doporučit např. [Jirků91].

Ačkoli Prolog je jazyk především pro symbolické výpočty (symbolic computations), bylo by poměrně nelogické, kdyby neumožňoval provádět s čísly alespoň ty nejjednodušší výpočetní operace. Nad implementovanými číselnými typy jsou proto k dispozici aritmetické operace označené běžnými operátory $+$, $-$, $*$, $/$, mod (jejich sémantika je totožná jako v Pascalu) a množina relačních symbolů umožňujících zápis porovnání hodnot číselných objektů $>$, $<$, $>=$, $=<$, $=$, $\backslash=$, $==$, $\backslash==$. Význam prvních čtyř relátorů je více méně zřejmý; symboly '=', resp. '\ =' použijeme k ověření, zda se symbolický výraz nalevo od něj shoduje, resp. neshoduje s výrazem napravo při případném dosazení za proměnné. Atomy '==' a '\ ==' umozňují testovat shodu (neshodu) dvou výrazů bez možnosti substituovat proměnné. Proto na dotaz

```
?- X = 1 + 4 .
```

Prolog odpoví

```
X = 1 + 4
```

a na dotaz

```
?- X == 1 + 4 .
```

dá zápornou odpověď, protože proměnná X je jiný symbolický výraz než $1 + 4$ a ztotožnění obou výrazů dosazením za X není tentokrát dovoleno.
Oba příklady ukazují, že uvedené dotazy nevedou k aritmetickým výpočtům, s výrazy se pracuje pouze jako s posloupnostmi symbolů. Vyhodnocení aritmetického výrazu je totiž v Prologu zcela jiná operace než test splnitelnosti nějakého cíle. Avšak Prolog umí vyhodnocovat pouze splnitelnost cílů; aritmetický výpočet proto musíme na takovou úlohu převést. Výpočtu hodnoty výrazu $1 + 4$ a jejího uložení do proměnné X docílíme zápisem

```
?- X is 1 + 4 .
```

Nalevo od operátoru *is* stojí vždy proměnná, napravo může být libovolný aritmetický výraz sestavený podle obvyklých pravidel z čísel, operátorů, proměnných a (případně) závorek. Aritmetické operátory mají svoji prioritu, která umožňuje některé závorky vynechávat. Detaily lze opět nalézt v [Jirků91].

3.4.4 Vyhodnocování příkazů aneb jak počítá Prolog

Vyhodnocování posloupnosti příkazů v Prologu spouštíme položením dotazu, říkáme *formulací cíle*. Každý dotaz je tvořen posloupností jednoho nebo více dílčích cílů a kladná odpověď vyžaduje současné splnění všech dílčích cílů. Prolog porovnává jednotlivé dílčí cíle položeného dotazu s hlavami klauzulí (nepodmíněný příkaz chápeme jako hlavu bez těla) zpravidla v pořadí, v jakém byly klauzule zapsány (tedy shora dolů, avšak existují implementace využívající efektivnější prohledávací strategie). Je-li možné dosáhnout shody (ve smyslu predikátové logiky unifikace) dílčího cíle a hlavy některého z nepodmíněných nebo podmíněných příkazů tím, že se vhodně dosadí za proměnné v dílčím cíli a dané klauzuli, Prolog provede stejnou substituci i v ostatních dílčích cílech dotazu a ověřovaný cíl nahradí podmínkami z těla klauzule. Tím vytvoří nový dotaz a v dalším kroku popsaný postup opakuje pro nově odvozenou posloupnost dílčích cílů (nový dotaz). Jsou-li v dotazu obsaženy nějaké proměnné, kladná odpověď obsahuje též hodnoty proměnných, pro které jsou všechny dílčí cíle splněny. Jestliže existuje více řešení (odpovědí na dotaz), Prolog vydá další řešení, je-li k tomu vyzván středníkem. Záporná odpověď (*no*) je vydána pouze tehdy, když k některému z dílčích cílů není nalezena komplementární hlava a postupnou rezolucí jednotlivých dílčích cílů se nepodaří odvodit prázdnou klauzuli.

Shrneme-li hlavní myšlenky algoritmu vyhodnocování programu v Prologu, lze hledání odpovědi na položený dotaz vyjádřit zjednodušeně jako

- *unifikaci proměnných* vhodným dosazením v atomech klauzulí,
- *logickou dedukci*, tj. přechod od prověřovaného cíle k podmánkám, které mohou jeho pravdivost zaručit,
- *návrat k alternativnímu řešení* dříve splněných cílů, pokud zvolené řešení ostatním cílům nevyhovuje.

Příklad 3. 11: Vyhodnocování dotazů v Prologu – výše uvedená fakta si ilustrujme vyhodnocením následujících čtyř dotazů:

a) Položíme-li dotaz

```
?- sousedí (maďarsko, rakousko).
```

, prohledá Prolog nejprve všechny nepodmíněné příkazy popisující relaci "sousedí". Protože nenajde nepodmíněný příkaz s odpovídajícím pořadím argumentů, zkusí v dalším kroku provést unifikaci proměnných v podmíněném příkazu

```
sousedí (X, Y) :- sousedí (Y, X) .
```

Po provedené substituci $\sigma_1 = \{\, \mathit{maďarsko}/X,\ \mathit{rakousko}/Y \,\}$ a rezoluci odvodí nový dotaz

```
?- sousedí (rakousko, maďarsko).
```

, ke kterému je v dalším kroku rezoluční metody nalezen odpovídající nepodmíněný příkaz, resp. jeho hlava, a rezoluční metodou odvozena prázdná klauzule. Prolog pak odpoví `yes` .

b) Položíme-li dotaz

```
?- moře (rakousko).
```

, bude jeho vyhodnocení probíhat stejným způsobem. Cíl *moře (rakousko)* se však neshoduje s hlavou žádné klauzule programu (unární relace *moře* je definována třemi nepodmíněnými příkazy, které připouštějí pouze individuové konstanty *severní_moře*, *baltské_moře* či *černé_moře*). Cíl tedy není možno splnit a Prolog odpoví `no` .

c) Bohužel ani Prolog není chráněn před nebezpečím vzniku nekonečných, či lépe neukončených výpočtů. Položíme-li v našem "zeměpisném" programu dotaz

```
?- sousedí (čr, rumunsko).
```

, pokusí se Prolog ověřit, že cíl *sousedí (čr, rumunsko)* je splněn. Porovnává jej postupně s hlavami klauzulí v našem programu, ale argumenty predikátu *sousedí* se neshodují s žádným z faktů, které jsou deklarovány nepodmíněnými příkazy. Shoda je nalezena až s hlavou podmíněného příkazu *sousedí (X, Y) :- sousedí (Y, X).* po substituci $\sigma_2 = \{\, čr/X,\ rumunsko/Y\,\}$. Dostaneme speciální případ tohoto podmíněného příkazu

`sousedí (čr, rumunsko) :- sousedí (rumunsko, čr). ,`

který podmiňuje splnění výše definovaného cíle pravdivostí podmínky *sousedí (rumunsko, čr)*, což znamená kladnou odpovědí na dotaz

`?- sousedí (rumunsko, čr).   .`

Stejně se Prolog snaží potvrdit pravdivost cíle *sousedí (rumunsko, čr)*, ale žádný nepodmíněný příkaz tomuto cíli nevyhovuje. Opět je tedy nalezen nepodmíněný příkaz *sousedí (X, Y) :- sousedí (Y, X).* , z nějž po substituci $\sigma_3 = \{\, rumunsko/X,\ čr/Y\,\}$ dostaneme speciální případ

`sousedí (rumunsko, čr) :- sousedí (čr, rumunsko). .`

Ten podmiňuje splnění cíle pravdivostí podmínky *sousedí (čr, rumunsko)*, která je totožná s cílem, od něhož jsme vyšli. Oba popsané kroky se budou znovu a znovu opakovat – vznikne neukončený výpočet. Prolog proto obsahuje další prostředky, jimiž lze takovýto výpočet ukončit; jejich detailní popis lze nalézt např. v [Jirků91].

d) Nakonec si ukažme, jak Prolog zpracovává dotazy složené z více cílů a jak prohledává možná alternativní řešení. Zeptáme-li se

`?- přímořský_stát (polsko).   ,`

použije Prolog podmíněný příkaz

`přímořský_stát (S) :- u_moře (S, _). ,`

který jako jediný definuje přímořské státy. Jedním krokem převede tento příkaz po substituci $\sigma_4 = \{\, polsko/S\,\}$ na dotaz

`?- u_moře (polsko, _).   .`

Podle podmíněného příkazu definujícího relaci *u_moře* je kladná odpověď na takto položený dotaz podmíněna kladnou odpovědí na dotaz

`?- sousedí (polsko, M), moře (M).   ,`

který sestává ze dvou dílčích cílů. Prolog je bude řešit jeden po druhém zleva doprava. Začne dílčím cílem *sousedí (polsko, M)* a najde první řešení $M =$ *pobaltí*. Zapamatuje si hodnotu proměnné M a zjišťuje, zda tato hodnota vyhovuje i druhému dílčímu cíli – pokusí se ověřit pravdivost cíle *moře (pobaltí)*, ale neuspěje. Zde by výpočet končil neúspěchem, kdyby cíl *sousedí (polsko, M)* nepřipouštěl žádná další řešení. Prolog se k němu však vrátí a zkusí pro něj nalézt další možné řešení. Po řadě vybere altermativu $M =$ *sns*, která v dalším kroku vede k ověřování pravdivosti dílčího cíle *moře (sns)*, které bude rovněž neúspěšné. Prolog se proto podruhé vrátí k cíli *sousedí (polsko, M)* a vybere další variantu $M =$ *baltské_moře*. Jelikož tato varianta vyhovuje i druhému cíli *moře (baltské_moře)*, do-

staneme odpověď `yes` .
Změnou pořadí dílčích cílů v dotazu

```
?- sousedí (polsko, M), moře (M).
```

,
by bylo možno dosáhnout určitého zkrácení výpočtu (náš program obsahuje deset zemí, ale jen tři moře); postup jeho vyhodnocení však ponecháme čtenáři jako cvičení.

Tolik stručný úvod do problematiky programování v Prologu. Prolog obsahuje samozřejmě celou řadu dalších konstrukcí, které umožňují práci s datovými strukturami, provádění i komplikovanějších výpočtů, úpravy programu (např. přidávání či vypouštění klauzulí) během výpočtu apod. Všechny tyto konstrukce však přesahují rámec skripta a lze se s nimi seznámit v kterékoli obsažnější publikaci zabývající se programováním v Prologu. Závěrem pouze charakterizujme Prolog jako jednoduchý konverzační programovací jazyk, který by měl usnadnit tvorbu logických programů širokému okruhu uživatelů. V Prologu je ponechán dostatek prostoru pro vlastní tvořivou práci uživatele, který má možnost soustředit se spíše na popis vlastních relací, tedy na otázku *CO* se má vyřešit, protože není nucen neustále přemýšlet *JAK* má tu či onu část programu efektivně zapsat, *KAM* má uložit vypočtené výsledky atd. Prolog patří k jazykům umožňujícím tzv. *symbolické programování*, jejichž hlavním charakteristickým rysem je potlačení nutnosti přemýšlet, jak daný problém co nejlépe zapsat.