

České Vysoké Učení Technické v Praze
Fakulta elektrotechnická

Velmi jemný úvod do matematické logiky

Doplňkový text k přednáškám Y01MLO a X01DML

Jiří Velebil
katedra matematiky
Praha, 2007

velebil@math.feld.cvut.cz
<http://math.feld.cvut.cz/velebil>



Obsah

Úvod	3
Co v textu je	3
Co v textu není	4
Poznámka ke zkouškám	5
1 Neformální pojmy matematiky	6
1.1 Definice	6
1.2 Věta a důkaz	8
1.3 Jazyk a meta-jazyk	10
Doplňující literatura	10
2 Výroková logika	11
2.1 Syntaxe a sémantika výrokové logiky	11
2.2 DNF a CNF, Karnaughovy mapy	17
2.3 Sémantický důsledek ve výrokové logice	32
2.4 Cvičení	34
Revize kapitoly	37
Doplňující literatura	37
3 Predikátová logika	39
3.1 Syntaxe a sémantika predikátové logiky	39
3.2 Sémantický důsledek v predikátové logice	50
3.3 Cvičení	51
Revize kapitoly	52
Doplňující literatura	52
4 Resoluční algoritmy	53
4.1 Resoluční algoritmus ve výrokové logice	53
4.2 Resoluční algoritmus v predikátové logice	56
4.3 Cvičení	64
Revize kapitoly	64
Doplňující literatura	65
5 Formalizace českých vět	66
5.1 Formalizace jednoduchých vět	66
5.2 Obtížnost formalizací	69
5.3 Cvičení	71
Revize kapitoly	71
Doplňující literatura	71
A Naivní teorie množin	72
A.1 Pojem množiny	72
A.2 Paradox pana Bertranda Russella	73
A.3 Cvičení	74
Revize kapitoly	74

Doplňující literatura	74
B Mohutnosti	75
B.1 Funkce jako nálepkovací schéma	75
B.2 Konečnost a nekonečnost	79
B.3 Spočetnost a nespočetnost	79
B.4 Cvičení	82
Revize kapitoly	82
Doplňující literatura	82
Rejstřík	83

Úvod

Existuje teorie, která tvrdí, že kdyby jednou někdo přišel na to, k čemu vesmír je a proč tu je, vesmír by okamžitě zmizel a jeho místo by zaujalo něco ještě mnohem bizarnějšího a nepravděpodobnějšího.

Existuje jiná teorie, která tvrdí, že už se to stalo.

Douglas Adams, *Stopařův průvodce po Galaxii*

Text, který začínáte číst, by měl posloužit jako doplněk pro předměty *Matematická logika* pro první ročník a *Diskrétní matematika a logika* pro druhý ročník. Cílem předmětu je podat základy matematické logiky, potřebné ke studiu *computer science*. Samozřejmě, že nebylo možné postihnout všechny aspekty moderní matematické logiky, bylo nutné se řídit sylabem přednášky. V této úvodní kapitole je tudíž stručně popsáno, co lze v textu nalézt a co všechno v něm chybí. Toto je první varianta textu. Proto přivítám jakékoli reakce.

Text je zveřejněn jako „klikací PDF“, tj. jako soubor s hypertextovými odkazy. Přesto nejde o „klikací knihu“, takový text bych psal a řadil úplně jinak. Hypertextové odkazy jsou vytvořeny jen pro zpříjemnění prohlížení elektronické podoby dokumentu.

Poděkování. Rád bych poděkoval studentkám a studentům *Diskrétní matematika a logika* zimního semestru 2006 za připomínky k některým částem textu. Za případné chyby, které v textu přesto zůstaly, nesu ovšem plnou odpovědnost já sám. K napsání byl použit systém \LaTeX Leslieho Lamporta, hypertextové odkazy byly vytvořeny balíkem `hyperref` Sebastiana Rahtze a Heiko Oberdiek, obrázky byly nakresleny pomocí maker `Xy-pic` Kristoffera H. Rose a Rosse Moorea.

Přeji Vám příjemnou četbu.

Jiří Velebil
V Praze, v červenci 2007

Co v textu je

I když je text psaný systémem definice, věta, důkaz, je v textu řada příkladů, které teoretické pojmy ilustrují na praktických problémech. Na konci každé kapitoly potom je řada cvičení. Pokud při čtení narazíte na značku



zvolněte: tak je označeno něco, co si vyžaduje Vaši pozornost. V každé kapitole na konci je také seznam možné doplňující četby. *V žádném případě nejde o literaturu povinnou.* Všechny definice, tvrzení apod. jsou průběžně číslovány a značkou ■ je označen konec důkazu. Na konci textu je rejstřík.

V textu jsou často i historické odbočky a poznámky, které bezprostředně s probíranou látkou nesouvisí. Účelem bylo ukázat, že matematika je jedna velká stavba, kterou staví lidé z masa a kostí, není to jen sbírka pouček z přednášek.

V **kapitole 1** se seznámíme velmi neformálně s pojmy jako *definice*, *věta* a *důkaz*. Tyto pojmy jsou základními prvky každého matematického textu. Vysvětlíme jak definice psát a co je věta a její důkaz.

Text z matematické logiky začíná v **kapitole 2** zavedením *syntaxe a sémantiky výrokové logiky*. Výroková logika je zavedena jako formální jazyk. Budeme studovat *syntaktickou analýzu* formulí a otázky *sémantické ekvivalence a sémantického důsledku*.

V **kapitole 3** zavedeme další formální jazyk: jazyk *predikátové logiky*. Predikátová logika má větší vyjadřovací schopnost než výroková logika. Toho je dosaženo především prací s *proměnnými a kvantifikátory*. Sémantika predikátové logiky je pak podána způsobem, který velmi připomíná standardní denotační *sémantiku imperativního programovacího jazyka*.

Kapitola 4 je věnována *resolučním algoritmům* ve výrokové a predikátové logice. Resoluční princip je algoritmus, který *syntakticky* zpracovává sémantické důsledky. Vysvětlíme myšlenky základních resolučních algoritmů, rafinovanější resoluční algoritmy v textu zmíněny nejsou.

Konečně v **kapitole 5** se zmíníme o poměrně složitém problému *formalizace českých vět* v jazycích výrokové a predikátové logiky. Formalizace totiž vyžaduje dokonale rozumět *přirozenému jazyku*. Některé jednoduché české věty však umíme formalizovat snadno a to se v textu naučíme. Tato dovednost je užitečná tehdy, když hodláme komunikovat s formálními systémy (v programování a umělé inteligenci).

V textu o logice je nutné zmínit i některé pojmy *naivní teorie množin*. To je uděláno v **dodacích A a B**. Zmíníme se o *naivním pojmu množiny* a základech *teorie mohutností množin*.

Co v textu není

V textu nejsou zmíněny pojmy **syntaktického důsledku** ve výrokové a predikátové logice. Syntaktický důsledek formalizuje **pojem důkazu** v logice. Syntaktický důsledek vysvětlený pomocí pravidel **přirozené dedukce** je podán ve skriptu

☞ M. Demlová a B. Pondělíček, *Matematická logika*, FEL ČVUT, Praha 1997

V knize

☞ A. Nerode a R. A. Shore, *Logic for Applications*, Springer, New York, 1997

je sémantický důsledek realizován systémem **úspěšných tabel**.

Se syntaktickým důsledkem jsou úzce spjaty pojmy **úplnosti** a **neúplnosti** logik. O těch se je možno dočíst ve vynikajících knihách

☞ D. R. Hofstadter, *Gödel, Escher, Bach: An Eternal Golden Braid*, Basic Books, New York, 1979

☞ R. Smullyan, *Gödel's Incompleteness Theorem*, Oxford University Press, 1992

O životě brilantního logika Kurta Gödela (a samozřejmě o jeho nejznámějším výsledku, **větě o neúplnosti**)¹ pojednává kniha

☞ R. Goldsteinová, *Neúplnost — Důkaz a paradox Kurta Gödela*, Argo + Dokořán, Praha, 2006

Logika souvisí s formálním myšlením. Otázku, zda mohou **počítače myslet** (a co by to vůbec mělo znamenat), probírají například knihy

☞ R. Penrose, *Emperor's New Mind: Concerning Computers, Minds and The Laws of Physics*, Oxford University Press, 1989

☞ R. Penrose, *Shadows of the Mind: A Search for the Missing Science of Consciousness*, Vintage Books, 1995

V těchto dvou knihách je důraz kladen na analýzu **Churchovy-Turingovy teze**² a **Gödelovy věty o neúplnosti**. Pokud máte raději knihy více filosofické, pak kniha

¹Věta brněnského rodáka Kurta Gödela (1906–1978), dokázaná v roce 1931, patří k nejzajímavějším výsledkům v matematice 20. století. Zhruba řečeno říká, že každý složitý systém (složitý=umožňující aritmetiku přirozených čísel) obsahuje věty, které jsou pravdivé, ale nedokazatelné. Zajímavé je, že pro důkaz své věty Gödel vlastně důvtipně formalizoval analogii *Epimenidova paradoxu* popisovaného například v Bibli (List Titovi, 1, verše 12–13): *Jeden z nich, jejich vlastní prorok, řekl: „Křesťané jsou sami lháři, zlá zvířata, lenivá břicha.“ A je to pravdivé svědectví.* Gödelův důkaz umožňuje sestavit analogické tvrzení: *Jsem nedokazatelná věta.*

²Tato teze zhruba tvrdí, že všechny modely pojmu *algoritmus* jsou navzájem ekvivalentní. Pojmenovaná je podle matematiků Alonza Churcha (1903–1995) a Alana Mathisona Turinga (1912–1954). Oba pány je právem možno považovat za jedny z otců moderní computer science. Alonzo Church vymyslel *λ -kalkulus*, který je teoretickým základem *funkcionálních programovacích jazyků*, jakým je například LISP, MIRANDA nebo HASKELL. Alan Turing v konceptu *Turingova stroje* popsal abstraktně, co je to *programovatelný počítač*. Obě práce vyšly v roce 1936.

☞ D. C. Dennet, *Consciousness Explained*, Penguin Books, 1993

je pro Vás tím pravým.

Poznámka ke zkouškám

Pro Y01MLO: Tento text *nejde* do všech podrobností, které budou na přednáškách odpředneseny. Informujte se o tématech a hloubce probrání těchto témat u přednášející/ho. Standardní učebnicí pro předmět Y01MLO je skriptum

☞ M. Demlová a B. Pondělíček, *Matematická logika*, FEL ČVUT, Praha 1997

Jak zkouška z X01DML, tak zkouška z Y01MLO nebude jen z početní techniky. Měli byste na jisté úrovni být schopni popsat i teoretické důvody, které zaručují správnost Vašeho řešení. Proto si zvykněte ke každému příkladu, který řešíte, přidat i drobný vysvětlující komentář.

U zkoušky budete definovat pojmy a vyslovovat a dokazovat tvrzení. Přečtěte si kapitolu **1**.

Kapitola 1

Neformální pojmy matematiky

Možná to nebylo důležité. Určitě to nebylo důležité. Nicméně to přesto představovalo zdroj Hollyho zmatků.

Je to zdroj zmatků, pomyslel si. Potom přemýšlel, jestli takový obrat jako „zdroj zmatků“ vůbec existuje, nebo jestli si ho právě vymyslel. Ani tohle nevěděl. Ach ano, bylo to beznadějně.

Rob Grant a Doug Naylor, *Červený trpaslík*

V této kapitole se neformálně seznámíme s pojmy, které budeme v dalším textu používat; totiž s pojmy *definice*, *věta* a *důkaz*. Znovu podotýkáme, že tyto pojmy neformalizujeme. Následující odstavce se tedy netýkají jen matematické logiky, týkají se jakékoli oblasti lidského zkoumání, ve které:

- hodláme pracovat s jasně zavedenými pojmy,
- hodláme o těchto pojmech pronášet nějaká tvrzení,
- hodláme o pravdivosti těchto tvrzení korektním způsobem přesvědčit své oponenty.

1.1 Definice

Definice je *vymezení pojmu*, se kterým hodláme v budoucnu pracovat. Jakožto každé vymezení pojmu, by každá definice měla obsahovat dvě části:

1. *Název* zaváděného pojmu.
2. *Charakteristickou vlastnost* zaváděného pojmu.

V tomto odstavci vybudujeme pojem *sudosti* přirozeného čísla. Záměrně budeme postupovat metodou pokusů a omylů. Pokusíme se tak ukázat, jakých chyb se lze při definování pojmu dopustit. Ukážeme také, jaké *špatné* otázky si můžeme při definování klást.

1.1.1 Definice Řekneme, že *číslo je sudé*, když je dělitelné dvěma.

Definice 1.1.1 má téměř všechny náležitosti, jediným nedostatkem je, že jsme neřekli, jaká čísla máme na mysli. Znamená to, že chceme zkoumat sudost například čísla π ? K zodpovězení této otázky je nutné si uvědomit, co myslíme slovem *číslo* v definici 1.1.1. Pokud tím myslíme reálné číslo, proč jsme to explicitně neřekli? Definici 1.1.1 tedy opravíme:

1.1.2 Definice Řekneme, že *reálné číslo je sudé*, když je dělitelné dvěma.

První námitku jsme tedy odstranili, zbývá však ještě porozumět charakteristické vlastnosti z definice 1.1.2. Co pro reálné číslo x znamená, že je dělitelné dvěma? Zřejmě fakt, že x umíme napsat jako $2z$, kde z je reálné číslo. Pak to ale znamená, že *každé* reálné číslo je sudé podle definice 1.1.2 (zkuste přesně zformulovat příslušnou větu a dokázat ji).

Zjistili jsme, že definice 1.1.2 je tedy formálně v pořádku, ve skutečnosti však žádný nový pojem *nezavádí*. Sudost reálných čísel jsme nejspíš opravdu definovat nechtěli. Proto definici znovu opravíme:

1.1.3 Definice Řekneme, že *přirozené číslo je sudé*, když je dělitelné dvěma.

Tato definice již smysl má. Všimneme si totiž, že jsme skutečně zavedli nový pojem. Některá přirozená čísla totiž sudá jsou a některá ne:

1.1.4 Příklad Je přirozené číslo 202 442 sudé? K odpovědi použijeme definici 1.1.3. Musíme rozhodnout, zda 202 442 je dělitelné dvěma. Ano je, protože $202\,442 = 2 \cdot 101\,221$ a dělitelnost dvěma znamená přesně existenci takového rozkladu.

Je přirozené číslo 187 299 sudé? K odpovědi použijeme opět definici 1.1.3. Musíme rozhodnout, zda 187 299 je dělitelné dvěma. To zcela jistě není, protože platí $187\,299 = 2 \cdot 93\,649 + 1$.



1.1.5 Poznámka V tento okamžik matematik zajásá: byl zaveden netriviální pojem a má tedy cenu začít se tímto pojmem zabývat, dávat jej do souvislostí s jinými pojmy a začít tak pracovat.

Přesto je v praxi časová posloupnost opačná: nové pojmy se zavádějí především pro *usnadnění vyjadřování*. Pracujeme-li v oblasti matematiky, kde se velmi často vyskytují přirozená čísla dělitelná dvěma, pak je velmi vhodné dát této vlastnosti nějaké jméno. Zatímco bez pojmu sudosti bychom nejspíš vydrželi (charakteristická vlastnost tohoto pojmu není tak komplikovaná), například bez pojmu *konjunktivní normální formy* by se naše vyjadřování v tomto textu stalo velmi těžkopádným.



1.1.6 Poznámka Všimněme si, že v příkladu 1.1.4 jsme museli rozumět vlastnosti *být dělitelný dvěma*. To znamená, že bychom tento pojem také měli přesně zavést. To se v matematice skutečně děje, každý zaváděný pojem by měl být pochopitelný a redukovatelný na pojmy jednodušší tak dlouho, až dojdeme k axiomatice (v našem případě k axiomatice přirozených čísel). V praxi se znalost některých pojmů předpokládá. Tak například nikde v tomto textu nevysvětlujeme, co je množina přirozených čísel, jak je definováno násobení přirozených čísel, a podobně. Předpokládáme, že čtenář(ka) už má něco z matematiky za sebou.



1.1.7 Poznámka Pozor! Častou chybou jsou tvrzení typu: *Definice sudosti platí.*, a podobné. Taková tvrzení jsou *zcela nesmyslná*. Definice je zavedení pojmu: jediné, co si u definic můžeme klást za cíl, je jejich adekvátnost a srozumitelnost. Definovat můžeme cokoli, co se nám líbí. Musíme při tom dbát jen na dvě věci:

1. Název zaváděného pojmu by se neměl shodovat s názvy již použitými.

V ideálním případě by název zaváděného pojmu měl být zapamatovatelný a měl by zaváděný pojem vysvětlovat. Výjimkou bývají názvy na počest matematiků, kteří se zaváděnými pojmy pracovali: z analýzy například znáte pojem *Riemannův integrál*. Je pojmenován po německém matematikovi Georgu Friedrichovi Bernhardu Riemannovi (1826–1866), který s tímto pojmem pracoval jako první a dokázal o něm řadu užitečných tvrzení.

Samozřejmě: můžete namítnout, že název *finitární signatura na lokálně konečně presentované kategorii* nic známého nepřipomíná, ale to jen proto, že jsme o teorii signatur nemluvili.

Matematika je budována od jednodušších pojmů ke složitějším a pokud projdeme celou cestu, tak nás výše uvedený název svojí názorností příjemně potěší.

V tomto textu vybudujeme část klasické matematické logiky. Všimněte si, jak je názvosloví tvořeno: u naprosté většiny pojmů zjistíte, že jejich názvy jsou velmi názorné.

2. Charakteristická vlastnost pojmu by měla být naprosto přesně popsána.

Definice totiž v matematice používáme především k tomu, abychom se zavedenými pojmy mohli dále pracovat. Jakákoli nejednoznačnost může mít fatální důsledky. Podívejte se do dodatků na definici množiny v **A.1.1** a na potíže, které z vágnosti charakteristické vlastnosti plynou: Russellův paradox **A.2.1**.

Situace, na kterou jsme narazili u definice **1.1.2** nám pak ukazuje, že bychom se po každém zavedení pojmu měli podívat, jaký má naše definice *smysl*. To znamená zeptat se, zda vůbec nějaký objekt, který definujeme, existuje, nebo zda není nově zaváděný pojem zbytečný.

Příkladem první situace je následující definice:

1.1.8 Definice Řekneme, že *přirozené číslo n je podivné*, pokud je n^2 záporné celé číslo.

Bližším zkoumáním snadno zjistíme, že žádné podivné přirozené číslo *neexistuje*. Ačkoli je tedy definice **1.1.8** formálně bezchybná, zavádí (velmi pravděpodobně) nesmyslný pojem.

Příkladem druhého extrému je definice **1.1.2**. Podle této definice je každé reálné číslo sudé. Pojem sudosti reálného čísla tedy splývá s pojmem reálného čísla. Nejspíš tedy nemá smysl sudost reálných čísel vůbec zavádět.



1.1.9 Poznámka V matematice se poměrně striktně dodržuje princip, nazývaný *Occamova břitva*:¹ zavádějte nové pojmy pouze tehdy, když je to vhodné pro další práci a množství zavedených pojmů minimalizujte.

1.2 Věta a důkaz

Pojmy samozřejmě zavádíme proto, abychom s nimi dále pracovali. Příklad takové práce s pojmem sudosti jsme viděli v příkladu **1.1.4**. Obecnějším příkladem práce je *formulování vět* a *dokazování vět*. Slovo *věta* je technický termín. Označujeme jím útvar, který má formu *pravdivého úsudku* o již zavedených pojmech.

Co je (ne nutně pravdivý) úsudek? Je to soubor oznamovacích vět, ve kterém jsou jasně odděleny *předpoklady* od *závěru*. Všechny pojmy v úsudku již musí být definovány.

Příkladem je (předpokládáme, že značce $n + 1$ rozumíme):

1.2.1 Úsudek *Ať n je libovolné sudé přirozené číslo. Potom přirozené číslo $n + 1$ je opět sudé.*

Útvar **1.2.1** má všechny náležitosti úsudku: oznamovací věta *Ať n je libovolné sudé přirozené číslo*. je jeho (jediným) předpokladem, oznamovací věta *Přirozené číslo $n + 1$ je opět sudé*. je závěrem úsudku, a slovo *potom* je oddělovací značkou (srovnejte s pojmem sémantického důsledku v odstavci **2.3**), která odděluje předpoklady od závěru.

Úsudek **1.2.1** však není pravdivý a není tudíž matematickou větou: číslo 0 je sudé (podle definice **1.1.3**), ale číslo 1 sudé není (opět podle definice **1.1.3**).

Příkladem věty je (předpokládáme, že značce n^2 rozumíme):

1.2.2 Věta *Ať n je libovolné sudé přirozené číslo. Potom přirozené číslo n^2 je opět sudé.*

Útvar **1.2.2** má opět tvar úsudku, a to dokonce *pravdivého úsudku*, proto jsme jej označili jako větu. O pravdivosti přesvědčujeme *důkazem*.

DŮKAZ. Ať n je libovolné sudé přirozené číslo. Potom (podle definice **1.1.3**) a podle toho, co znamená dělitelnost dvěma, existuje přirozené číslo k tak, že $n = 2k$. Chceme ukázat, že přirozené číslo n^2 je opět sudé.

S číslem k můžeme dále pracovat. Upravíme n^2 takto: $n^2 = (2k)^2 = 4k^2 = 2 \cdot (2k^2)$. Protože $2k^2$ je přirozené číslo, je důkaz ukončen: číslo n^2 je dělitelné dvěma, a tudíž jde o sudé číslo. ■



1.2.3 Poznámka Předchozí důkaz je možná příliš pedantický. V praxi se důkazy píší tak, že některá fakta se předkládají jako zřejmá. Přesto musí mít každý korektní důkaz vlastnost „nafukovatelnosti“: kdykoli si to oponent vyžádá, musíme umět nejasná místa našeho důkazu dovysvětlit.

Není jisté příliš korektní prohlásit, že důkaz následující věty je zřejmý:

¹William of Occam (1285–1349) byl anglickým scholastickým filosofem.

Věta. Pro přirozené číslo $n \geq 3$ neexistují přirozená čísla a, b, c splňující rovnici $a^n + b^n = c^n$.

Pokusíte-li se o důkaz tohoto tvrzení, zjistíte, že to není jednoduché. Jde totiž o *velkou Fermatovu větu* (také se jí říká *poslední Fermatova věta*), která byla zformulována francouzem Pierre de Fermatem (1601–1665). Problém odolával více než 300 let (zůstal tak posledním z nedokázaných Fermatových tvrzení, odtud pochází název věty). Po Fermatově smrti se objevilo mnoho špatných důkazů a pokusy o důkaz velké Fermatovy věty zruinovaly nejednu vědeckou kariéru.

Teprve 23. června 1993 na přednášce v Cambridge předvedl angličan Andrew Wiles důkaz, který je korektní. Intenzivně na něm pracoval sedm let a využil v něm nejmodernějšího aparátu matematiky.² Počet stran článku

☞ A. Wiles, Modular Elliptic Curves and Fermat's Last Theorem, *Ann. Math.* 142 (1995), 443–551

o jednoduchosti Wilesova důkazu leccos napovídá.

Co je a co není zřejmé je opět otázka matematické praxe. Snažte se proto slovem *zřejmý* ve svých důkazech šetřit.



1.2.4 Poznámka Je těžké říci neformálně, co důkaz je. Obecně by mělo jít o konečnou korektní argumentaci, která nás od předpokladů věty dovede bezpečně k jejímu závěru.

Techniky důkazů v matematice jsou nejrůznější: znáte důkaz indukci, důkaz přímý, důkaz nepřímý, důkaz sporem, a podobně.

Jak se dokazování naučit? Čtením již existujících důkazů a pokoušením se psát důkazy vlastní. Budete-li číst důkazy v tomto textu, záhy zjistíte, že dokazování je vlastně snadné: jakmile člověk zvládne základní techniku, dostane po spatření znění věty pocit, že by se „na to šlo asi takhle“ a tento pocit je v naprosté většině případů správný. Jen překvapivě málo důkazů v matematice používá takové triky,³ na které by člověk okamžitě nepřišel. Matematici ale takové triky samozřejmě milují, protože (poté, co příslušný trik zvládnou) obohacují škálu technik, se kterými jde pracující matematik do boje. Příkladem takového triku je *Cantorův diagonální trik* z věty B.3.9, jehož aplikace v moderní matematice již pomohly dokázat celou řadu zajímavých výsledků.



1.2.5 Poznámka V textu narazíte i na jiné názvy, než jen *věta*: jsou to *lemma* (obecně: pomocná věta malé důležitosti), *tvrzení* (obecně: pravdivý úsudek menší důležitosti než věta) a *důsledek* (název snad nepotřebuje vysvětlení).

Tyto další pojmy odrážejí logickou strukturu textu. Ovšem, jako ve všem na tomto světě, jsou i zde jisté výjimky: například *Zornovo lemma* je velmi důležitým tvrzením axiomatické teorie množin a název *lemma* si drží z historických důvodů.

Budete-li psát sami někdy nějaký matematický text, budete již mít za sebou četbu řady textů, které matematici psali, a rozlišování vět, tvrzení a lemmat Vám nebude činit nejmenší potíže.



1.2.6 Poznámka Matematické texty jsou psány *systémem DVD*, tj. definice-věta-důkaz. Jde o systém, který se historicky vyvinul v nejmocnější zbraň racionálního uvažování.

Je vhodné poznamenat, že při matematické práci není většinou časová posloupnost *DVD* dodržována. Důkazy se většinou tvoří jako první, v nich se mohou objevit pojmy, které stojí za to definovat, je napsána definice a pak je teprve formulována věta. Matematik si totiž s pojmy „hraje“ a zjišťuje, co o nich platí. Málokdy to vypadá takto: teď si sednu, něco definuju, pak o tom zformuluju nějakou větu a odpoledne tu větu dokážu. Definice, věty a důkazy se rodí studiem jiných definic, vět a důkazů, jejich zobecňováním a hledáním analogií.

Časová posloupnost *DVD*, kterou znáte z přednášek a nejrůznějších textů, je otázkou presentace již udělané práce: nejprve sdělíme s jakými pojmy budeme pracovat, pak ve větě sdělujeme nová fakta a poté přesvědčujeme případné oponenty o tom, že máme pravdu.

Přesto si povšimněte, že slušný matematický text není jen čistě *DVD*, v takovém textu by neměla chybět řada příkladů a spojujícího textu, který vysvětluje proč a kam míříme.

²Historii pokusů o důkaz velké Fermatovy věty (a samozřejmě Wilesův triumf) popisuje velmi čtivá kniha S. Singh, *Fermat's Last Theorem*, Fourth Estate Ltd., London, 1997.

³Nemyslíme samozřejmě kouzla, ale rafinované techniky.

1.3 Jazyk a metajazyk

V tomto textu budeme studovat matematickou logiku, a proto narazíme mnohokrát na problém *rozlišení jazyka a metajazyka*.

Metajazykem budeme v tomto textu vždy rozumět češtinu: text je napsán česky, o jednotlivých pojmech mluvíme česky, atd. *Jazykem*, který zkoumáme a o kterém mluvíme pomocí metajazyka, je tu logika.

Musíme si tudíž dávat velký pozor na zápisy tvaru

$$\varphi \text{ je tautologie} \Leftrightarrow \varphi \text{ je pravdivá v každé interpretaci.} \quad (1.1)$$

Takový zápis totiž míchá jazyk s metajazykem: symbol \Leftrightarrow je logická spojka a *patří do jazyka*, který zkoumáme. Proto budeme místo výše uvedeného psát

$$\varphi \text{ je tautologie právě tehdy, když } \varphi \text{ je pravdivá v každé interpretaci.} \quad (1.2)$$

I když se řádky (1.1) a (1.2) čtou stejně, řádek (1.1) je napsán naprosto *špatně*. Pamatujte si: metajazykem je (matematická) čeština.

Matematická čeština ovšem používá celou řadu obrátů, kterým je nutné správně rozumět: *...právě tehdy, když... , jestliže... , potom...* a tak dále. Nelze poradit nic jiného, než číst matematický text a uvědomovat si význam těchto speciálních slovních spojení. Po troše praxe zjistíte, že jejich použití je velmi jednoduché.

Můžete namítnout, že matematická čeština je jazyk podivný: to je způsobeno tím, že se snažíme vyjadřovat jednoznačně a přesně. Skutečný přirozený jazyk je daleko bohatší a připouští nejednoznačnosti a skryté významy, které dělají skutečný život tak zajímavým.

Doplňující literatura

Neformální zavádění pojmů *definice*, *věta* a *důkaz* je častým námětem filosofických teorií. Podívejte se například do knih

☞ P. Kolář, *Argumenty filosofické logiky*, Filosofia, Praha 1999

☞ B. Russell, *Zkoumání o smyslu a pravdivosti*, Academia, Praha 1975

Kapitola 2

Výroková logika

Pravdivost tvrzení nemá s jeho uvěřitelností nic společného. A naopak.

The Notebooks of Lazarus Long

V této kapitole zavedeme základní pojmy výrokové logiky. Vysvětlíme, že tato logika je *formálním jazykem*: má tedy svoji syntaxi a sémantiku. Budeme řešit následující problémy:

1. Syntaktická analýza: jak rozpoznat, že daný řetězec je formulí?
2. Sémantická ekvivalence: jak rozpoznat, že dvě formule „znamenají“ totéž?
3. Sémantický důsledek: jak rozpoznat, že ze zadaných formulí plyne další formule?

Podstatným rysem výrokové logiky je existence dobře známých pravdivostních tabulek. V predikátové logice, zavedené v kapitole 3, nic jako pravdivostní tabulka neexistuje.

2.1 Syntaxe a sémantika výrokové logiky

Připomeňme, že potřeba *výrokové logiky* tak, jak je vyložena například ve skriptu

☞ M. Demlová a B. Pondělíček, *Matematická logika*, skriptum FEL ČVUT, Praha, 1997

vznikla z nejasností, které jsou způsobeny používáním přirozeného jazyka pro popis nejrůznějších (i velmi jednoduchých) situací, jakou je například výrok

Mám hlad a jdu spát.

Tyto problémy lze odstranit tak, že přesně vymezíme, co je výrok a jakým způsobem lze nové výroky vytvářet.

1. Některé věty deklarativně za výroky prohlásíme. Těmto větám budeme říkat *atomické výroky*, protože je hodláme považovat za dále nedělitelné.
2. Další výroky budeme z již zkonstruovaných výroků vytvářet předem dohodnutým způsobem. Konstrukce, které budeme k vytváření nových výroků používat, nazveme *logické spojky*.
3. Žádným jiným způsobem než výše popsány *nelze* výrok zkonstruovat.

Při určování pravdivosti výroků budeme postupovat následovně:

1. Zjistíme, zda daný výrok je atomický nebo ne.
2. Pro atomické výroky musíme jejich pravdivost či nepravdivost *znát*.
3. Není-li výrok atomický, pak vznikl spojením jednodušších výroků. Známe-li pravdivost jednodušších výroků a *rozumíme-li* jednotlivým spojkám, zjistíme pravdivost složitějšího výroku.

Snadno zjišťujeme, že přirozený jazyk lze v úvahách z předchozích odstavců nahradit *symbols* a zavést tak výrokovou logiku čistě formálně. Namísto *výrok* budeme říkat *formule*.

1. Zvolíme množinu At (smí být i prázdná). Jejím prvkům budeme říkat *atomické formule*.¹
2. Zvolíme množinu S disjunktní s At . Prvkům množiny S budeme říkat *logické spojky*. Každé logické spojce přiřadíme kladné přirozené číslo — její *aritu*. Arita spojky udává, kolik formulí daná spojka spojí v novou formuli. Běžným přístupem² je volba spojek

- | | |
|--|---|
| (a) <i>True</i> : \mathbf{tt} arity 0. | (d) <i>Disjunkce</i> : \vee arity 2. |
| (b) <i>Negace</i> : \neg arity 1. | (e) <i>Implikace</i> : \Rightarrow arity 2. |
| (c) <i>Konjunkce</i> : \wedge arity 2. | (f) <i>Ekvivalence</i> : \Leftrightarrow arity 2. |

Obecná formule je pak buď atomická formule, nebo \mathbf{tt} nebo řetězec $(\neg\varphi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \Rightarrow \psi)$, $(\varphi \Leftrightarrow \psi)$, kde φ i ψ jsou formule.

3. Žádným jiným způsobem než konečným použitím výše popsaných pravidel *nelze* formuli zkonstruovat.



2.1.1 Poznámka Výše uvedená syntaxe se nejkratším způsobem popíše *Backusovou-Naurovou formou*.³ Zápis by vypadal takto:

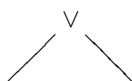
$$\varphi ::= a \mid \mathbf{tt} \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \Rightarrow \varphi) \mid (\varphi \Leftrightarrow \varphi) \quad a \in At$$



2.1.2 Příklad Je řetězec $(a \vee (E \Rightarrow (\neg b)))$ formulí výrokové logiky?

Na danou otázku odpoví algoritmus, který se pro daný řetězec pokusí sestavit *syntaktický strom*. Pro daný řetězec bude algoritmus postupovat následovně:

1. Přečteme první závorku zleva a hledáme odpovídající závorku (je to závorka nejvíce napravo). Nalezneme *hlavní* spojku v této závorce: jde o spojku \vee . Přečteme si aritu této spojky (je to spojka arity 2) a vytvoříme zárodek syntaktického stromu:



2. Dále se rekurzivně snažíme vytvořit syntaktické stromy řetězců, které spojka \vee váže: jedná se o řetězce a a $(E \Rightarrow (\neg b))$.
3. Při pokusu o sestavení syntaktického stromu pro řetězec a se algoritmus zastavuje: *nemáme žádnou informaci o tom, zda a je atomická formule*.

Závěr: daný řetězec *není* formulí výrokové logiky.

Situace se změní, pokud změním původní zadání na následující problém: Ať $a, b, E \in At$. Je řetězec

$$(a \vee (E \Rightarrow (\neg b)))$$

formulí výrokové logiky?

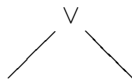
Opět spustíme algoritmus pro hledání syntaktického stromu. Budeme postupovat rychleji:

¹Množina At smí obsahovat jakékoli prvky s výjimkou konstanty \mathbf{tt} , logických spojek a závorek — to lze ovšem vždy bez újmy na obecnosti předpokládat.

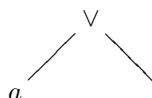
²Pro minimalistické volby množiny spojek viz cvičení 2.4.1 a 2.4.2.

³Tuto notaci vymyslel John Warner Backus (nar. 1924) v roce 1959 pro popis syntaxe jazyka ALGOL 60. Původně se jí říkalo Backusova normální forma, na Backusovu-Naurovu formu byla přejmenována později (Peter Naur se podílel na zprávě o ALGOLU v roce 1963). Je zajímavé, že pravidla stejné vyjadřovací schopnosti (metapřavidla, transformace a rekurze) použil hindský gramatik Pāṇini (cca 6. století př.n.l.) pro popis *sanskrtu*. Jeho 3959 veršů díla *Astādhyāyī* tak představuje první formální popis přirozeného jazyka.

1. Přečteme první závorku zleva a hledáme odpovídající závorku (je to závorka nejvíce napravo). Nalezneme *hlavní* spojku v této závorce: jde o spojku \vee . Přečteme si aritu této spojky (je to spojka arity 2) a vytvoříme zárodek syntaktického stromu:



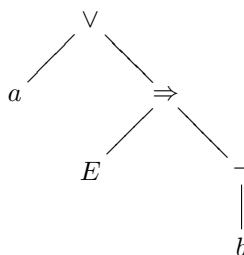
2. Dále se rekursivně snažíme vytvořit syntaktické stromy řetězců, které spojka \vee váže: jedná se o řetězce a a $(E \Rightarrow (\neg b))$.
3. Řetězec a neobsahuje žádné závorky. Prohlédneme si deklaraci množiny At a hledáme, zda a je jejím prvkem. Ano, je. Dosud vytvořený syntaktický strom vypadá takto:



a tvorba levé části stromu je ukončena, snažíme se vytvořit syntaktický strom pro řetězec $(E \Rightarrow (\neg b))$.

⋮

4. Algoritmus se zastavuje úspěšně sestavením syntaktického stromu



Závěr: Ať $a, b, E \in At$. Pak daný řetězec je formulí výrokové logiky.



2.1.3 Poznámka Zavedeme následující *syntaktické konvence*:

1. Nebudeme psát nejvíce vnější závorky. Například místo $(E \Rightarrow (\neg b))$ budeme psát $E \Rightarrow (\neg b)$.
2. Budeme předpokládat, že spojka \neg váže nejsilněji. Například místo $E \Rightarrow (\neg b)$ budeme psát $E \Rightarrow \neg b$.
3. Zavedeme *false*: formule **ff** je *syntaktická zkratka* za formuli $\neg \mathbf{tt}$.

Nebude-li uvedeno jinak, budeme v dalším vždy předpokládat, že syntaxe je relaxována podle těchto pravidel.

Sémantika vytvořených formulí výrokové logiky je dána jejich *ohodnocením*. Ohodnocení formule nabývá hodnoty buď 1 (pravda) nebo 0 (nepravda). Přitom chceme, aby byla ohodnocena *každá* vytvořená formule. Ohodnocení je tedy funkce u z množiny všech formulí do množiny $\{0, 1\}$ taková, že jsou splněny následující požadavky:

1. $u(\mathbf{tt}) = 1$.
2. $u(\neg\varphi) = 1$ právě tehdy, když $u(\varphi) = 0$.
3. $u(\varphi \wedge \psi) = 1$ právě tehdy, když $u(\varphi) = u(\psi) = 1$.
4. $u(\varphi \vee \psi) = 0$ právě tehdy, když $u(\varphi) = u(\psi) = 0$.
5. $u(\varphi \Rightarrow \psi) = 0$ právě tehdy, když $u(\varphi) = 1$ a $u(\psi) = 0$.

6. $u(\varphi \Leftrightarrow \psi) = 1$ právě tehdy, když $u(\varphi) = u(\psi)$.

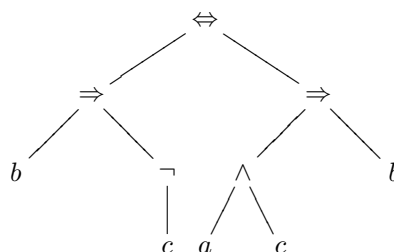
Je zřejmé, že pokud pro každou atomickou formuli a zadáme hodnotu $u(a)$, pak výše uvedená pravidla nám umožní zjistit hodnotu $u(\varphi)$ pro libovolnou formuli φ výrokové logiky.

2.1.4 Příklad Předpokládejme, že $a, b, c \in At$. Pak řetězec $\varphi = (b \Rightarrow \neg c) \Leftrightarrow ((a \wedge c) \Rightarrow b)$ je formule výrokové logiky a my můžeme obvyklým způsobem spočítat pravdivostní tabulku pro formuli φ .

a	b	c	φ
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Algoritmus pro vyplňování pravdivostní tabulky je opět rekurzivní: připomeňme, jak postupujeme například při vyplňování pátého řádku, tj. při ohodnocení $u(a) = 1$, $u(b) = 0$, $u(c) = 0$.

1. Syntaktický strom dané formule je



Dané ohodnocení ohodnocuje listy syntaktického stromu: $u(a) = 1$, $u(b) = 0$, $u(c) = 0$.

2. Daný strom nyní procházíme od listů ke kořeni a postupně ohodnocujeme vnitřní vrcholy stromu tak, jak nám přikazují sémantická pravidla. Například vrchol \wedge je ohodnocen 0, protože $u(a \wedge c) = 0$.
3. Ohodnocení celé formule φ je hodnota, kterou má kořen syntaktického stromu: $u(\varphi) = 1$.



2.1.5 Poznámka Konstrukce pravdivostní tabulky je klasickým příkladem doktríny formálních jazyků: *syntaxe řídí sémantiku*. Na to, abychom byli schopni říci význam komplikovaného řetězce, musíme znát význam jednodušších částí.

S aparátem *induktivních definic* bychom byli schopni tuto doktrínu vyjádřit lépe: syntaxe formulí je definována induktivně z množiny atomických formulí a proto můžeme sémantiku (tj. ohodnocení *všech* formulí) definovat rekurzivně ze znalosti ohodnocení

$$u : At \longrightarrow \{0, 1\}$$

atomických formulí (přesně to jsme dělali v minulém příkladu). Proto v dalším textu budeme často ohodnocení popisovat jeho hodnotou na atomických formulích z At . Pro vysvětlení induktivních definic odkazujeme na text:

☞ J. Velebil, *Diskrétní matematika*, <http://math.feld.cvut.cz/velebil/>

Povšimněme si triviálního, ale důležitého faktu:

2.1.6 Lemma *At formule φ obsahuje s různých atomických formulí, $s \geq 0$. Potom pravdivostní tabulka formule φ obsahuje 2^s řádků.*

DŮKAZ. Je-li $s = 0$, pak musí formule φ být vytvořena z formule \mathbf{tt} pomocí logických spojek. Její pravdivostní tabulka pak obsahuje jediný řádek (promyslete proč).

At $s > 0$. Označme atomické formule, které jsou ve formuli φ obsaženy jako a_1, \dots, a_s . Potom existuje 2^s různých ohodnocení těchto atomických formulí (protože jde o počet zobrazení z množiny $\{a_1, \dots, a_s\}$ do množiny $\{0, 1\}$). Tudíž existuje 2^s různých ohodnocení. ■



2.1.7 Poznámka Z předchozího lemmatu plyne, že jakoukoli sémantickou otázku o konečném počtu formulí výrokové logiky lze vyřešit (obecně velmi neefektivním) algoritmem, který sestaví a prohledá konečnou (obecně ale velmi rozměrnou) pravdivostní tabulku (srovnejte s poznámkou 3.1.23).

Připomeňme znovu, jak pravdivostní tabulka vypadá. Označme jako $s \geq 0$ počet atomických formulí, které jsou ve formuli φ použity.

1. V případě $s = 0$ má pravdivostní tabulka tvar

	φ
--	-----------

to jest: záhlaví pro atomické formule je prázdné a pravdivostní hodnotu formule φ počítáme v jediném (prázdném) ohodnocení u , kdy $u(\mathbf{tt}) = 1$.

2. V případě $s > 0$ má pravdivostní tabulka 2^s řádků. Budeme používat standardní pořadí řádků, například tabulka s osmi řádky bude vypadat takto ($a, b, c \in At$):

a	b	c	φ
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Jako první instanci algoritmu, prohledávajícího jistým způsobem pravdivostní tabulku, zmíníme *problém synonym*: jak poznat, že dvě formule φ a ψ znamenají totéž? Nejprve musíme ovšem říci, co tím myslíme.

2.1.8 Definice Formule φ a ψ jsou *sémanticky ekvivalentní* (jsou *synonyma*), pokud platí rovnost $u(\varphi) = u(\psi)$ pro všechna pravdivostní ohodnocení u . Tento fakt značíme $\varphi \models \psi$.

2.1.9 Příklad At $a, b \in At$. Pak platí $a \Rightarrow b \models \neg a \vee b$.

Sestavíme pravdivostní tabulku obou formulí (má $2^2 = 4$ řádky):

a	b	$a \Rightarrow b$	$\neg a \vee b$
0	0	1	1
0	1	1	1
1	0	0	0
1	1	1	1

Definice sémantické ekvivalence pak přikazuje projít všechny řádky dané pravdivostní tabulky (protože se v definici říká *pro všechna pravdivostní ohodnocení*). Pokud se hodnoty $u(a \Rightarrow b)$ a $u(\neg a \vee b)$ budou shodovat, ukážeme, že $a \Rightarrow b \models \neg a \vee b$ platí.

Ano, sloupce ohodnocení obou formulí se shodují, tudíž $a \Rightarrow b \models \neg a \vee b$ platí.

Sémantická ekvivalence má všechny potřebné vlastnosti, které umožňují „slepovat“ formule, tj. jde o *relaci ekvivalence*.

2.1.10 Věta Platí:

1. Pro libovolnou formuli α platí: $\alpha \models \alpha$ (relace \models je reflexivní).
2. Pro libovolné formule α a β platí: jestliže platí $\alpha \models \beta$, pak platí $\beta \models \alpha$ (relace \models je symetrická).
3. Pro libovolné formule α , β a γ platí: jestliže platí $\alpha \models \beta$ a současně $\beta \models \gamma$, pak platí $\alpha \models \gamma$ (relace \models je transitivní).

DŮKAZ. Ukážeme pouze transitivitu relace \models , ostatní vlastnosti se dokáží analogicky. Ať tedy α , β a γ jsou libovolné formule výrokové logiky a ať platí $\alpha \models \beta$ a současně $\beta \models \gamma$. Chceme ukázat, že platí $\alpha \models \gamma$.

Zvolíme tedy libovolné ohodnocení u a chceme ukázat, že platí rovnost $u(\alpha) = u(\gamma)$. Protože platí $u(\alpha) = u(\beta)$ a současně $u(\beta) = u(\gamma)$, plyne vztah $u(\alpha) = u(\gamma)$ z vlastností rovnosti. ■

Synonyma formulí **tt** a **ff** si zaslouží zvláštní jméno.

2.1.11 Definice Formule φ je *tautologie*, pokud platí $\varphi \models \mathbf{tt}$ a formule φ je *kontradikce*, pokud platí $\varphi \models \mathbf{ff}$.

Následující věta má jednoduchý (ale zdlouhavý) důkaz. Pokuste se ji dokázat.

2.1.12 Věta Ať α , β , γ jsou formule výrokové logiky. Pak platí:

1. $\alpha \wedge \beta \models \beta \wedge \alpha$ (spojka \wedge je sémanticky komutativní).
2. $\alpha \wedge (\beta \wedge \gamma) \models (\alpha \wedge \beta) \wedge \gamma$ (spojka \wedge je sémanticky asociativní).
3. $\alpha \wedge \alpha \models \alpha$ (spojka \wedge je sémanticky idempotentní).
4. $\alpha \vee (\alpha \wedge \beta) \models \alpha$ (platí sémantický absorpční zákon spojky \wedge).
5. $\alpha \vee \beta \models \beta \vee \alpha$. (spojka \vee je sémanticky komutativní).
6. $\alpha \vee (\beta \vee \gamma) \models (\alpha \vee \beta) \vee \gamma$ (spojka \vee je sémanticky asociativní).
7. $\alpha \vee \alpha \models \alpha$. (spojka \vee je sémanticky idempotentní).
8. $\alpha \wedge (\alpha \vee \beta) \models \alpha$ (platí sémantický absorpční zákon spojky \vee).
9. $\alpha \wedge (\beta \vee \gamma) \models (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$ (platí sémantický distributivní zákon spojek \wedge a \vee).
10. $\alpha \wedge \mathbf{tt} \models \alpha$ (formule **tt** je nejvíce pravdivá).
11. $\alpha \wedge \mathbf{ff} \models \mathbf{ff}$ (formule **ff** je nejméně pravdivá).
12. $\alpha \wedge \neg \alpha \models \mathbf{ff}$ (platí sémantický zákon kontradikce).
13. $\alpha \vee \neg \alpha \models \mathbf{tt}$ (platí sémantický zákon vyloučeného třetího).
14. $\neg(\alpha \wedge \beta) \models \neg \alpha \vee \neg \beta$ a $\neg(\alpha \vee \beta) \models \neg \alpha \wedge \neg \beta$ (platí sémantické De Morganovy zákony).
15. $\neg \neg \alpha \models \alpha$ (platí sémantický zákon idempotence negace).



2.1.13 Poznámka V předchozí větě jsme úzkostlivě dbali u všech zákonů na přídatné jméno *sémantický*. Má to velký význam. Jsou-li $a, b \in At$, pak platí $a \wedge b \models b \wedge a$, protože spojka \wedge je *sémanticky komutativní*, ale *neplatí* rovnost $a \wedge b = b \wedge a$, protože jde o různé řetězce. Spojka \wedge ve výrokové logice tedy na syntaktické úrovni *není komutativní*.

Přesto dovolíme další relaxaci syntaxe a dovolíme (v úvahách o sémantice) psát například $a \vee b \vee c$ jako zkratku za kterýkoli ze správných zápisů $a \vee (b \vee c)$ nebo $(a \vee b) \vee c$.

2.2 DNF a CNF, Karnaughovy mapy

Karnaughovy⁴ mapy jsou přepisem pravdivostní tabulky a slouží jako nástroj k rychlému zápisu konjunktivních a disjunktivních normálních forem (CNF a DNF) formulí výrokové logiky. V tomto odstavci se zaměříme na následující problémy:

1. Je-li dána pravdivostní tabulka formule, jak napsat tuto formuli v úplné CNF a úplné DNF?
2. Je-li dána pravdivostní tabulka formule, jak napsat tuto formuli v CNF a DNF, které jsou v ireducibilním tvaru, tzn. tyto formy již nelze dále redukovat pomocí pravidel výrokové logiky?

Obě úlohy lze vyřešit poměrně snadno pomocí Karnaughových map.

Nejprve připomeneme základní definice: ať φ je formule výrokové logiky. Řekneme, že formule ψ je

- *disjunktivní normální formou* (též *DNF*) formule φ , pokud platí $\psi \models \varphi$ a formule ψ je buď formule

tt

nebo je napsána ve tvaru

$$\psi_1 \vee \dots \vee \psi_n$$

pro nějaké přirozené číslo $n \geq 1$, kde každá formule ψ_i ($i \in \{1, \dots, n\}$) je buď rovna formuli **ff**, tj. kontradikci, nebo je napsána ve tvaru

$$l_1 \wedge \dots \wedge l_{k_i}$$

pro nějaké přirozené číslo $k_i \geq 1$ a každé l_j ($j \in \{1, \dots, k_i\}$) je buď atomická nebo negace atomické formule. V tomto kontextu každé formulí ψ_i ($i \in \{1, \dots, n\}$) říkáme *klausule pro DNF* (používá se i termín *minterm*) a každému l_j ($j \in \{1, \dots, k_i\}$) říkáme *literál*.

Číslo n budeme říkat *počet klausulí v DNF* a každému z čísel k_1, \dots, k_n budeme říkat *délka klausulí* ψ_1, \dots, ψ_n .

V případě, že ψ je formule **tt**, říkáme, že DNF formule φ má nulový počet klausulí pro DNF.

Poznamenejme ještě, že klausule **ff** má délku 0.

- *konjunktivní normální formou* (též *CNF*) formule φ , pokud platí $\psi \models \varphi$ a formule ψ je buď formule

ff

nebo je napsána ve tvaru

$$\psi_1 \wedge \dots \wedge \psi_n$$

pro nějaké přirozené číslo $n \geq 1$, kde každá formule ψ_i ($i \in \{1, \dots, n\}$) je buď rovna formuli **tt**, tj. tautologii, nebo je napsána ve tvaru

$$l_1 \vee \dots \vee l_{k_i}$$

pro nějaké přirozené číslo $k_i \geq 1$ a každé l_j ($j \in \{1, \dots, k_i\}$) je buď atomická nebo negace atomické formule. V tomto kontextu každé formulí ψ_i ($i \in \{1, \dots, n\}$) říkáme *klausule pro CNF* (používá se i termín *maxterm*) a každému l_j ($j \in \{1, \dots, k_i\}$) říkáme *literál*.

Číslo n budeme říkat *počet klausulí v CNF* a každému z čísel k_1, \dots, k_n budeme říkat *délka klausulí* ψ_1, \dots, ψ_n .

V případě, že ψ je formule **ff**, říkáme, že CNF formule φ má nulový počet klausulí pro CNF.

Poznamenejme ještě, že klausule **tt** má délku 0.

⁴Maurice Karnaugh (nar. 1924) navrhl tyto mapy jako způsob rychlého návrhu logických obvodů v roce 1953.

Jistě jste zjistili, že definice DNF a CNF formule jsou si velmi podobné. Tato podobnost (jde o jistou dualitu) v praxi umožňuje zformulovat postupy a výsledky například pouze pro disjunktivní normální formy a potom se odvolat na dualitu mezi DNF a CNF. Protože tento text chce být elementární, nebudeme žádnou dualitu zavádět. Za tento elementární postup platíme jistou zdlouhavostí textu — naprostou většinu výsledků uvádíme zvlášť pro DNF a CNF.

Nejprve definici DNF rozebereme podrobněji. Formule ψ je disjunktivní normální formou formule φ , pokud jsou splněny následující dva požadavky:

1. Formule ψ je *synonymem* pro φ , tj. platí $\psi \models \varphi$.
2. Formule ψ má speciální *syntaktickou stavbu*: tj. ψ je disjunkcí „stavebních kamenů“ pro disjunktivní normální formu. Těmto stavebním kamenům říkáme *klausule pro DNF*. Každá klausule pro DNF má opět speciální syntaktickou stavbu: klausule pro DNF je konjunkcí literálů. Literál je buď atomická formule nebo negace atomické formule a slouží jako „stavební kámen“ pro tvorbu klausule pro DNF.

Při tvorbě DNF je možné nepoužít žádnou klausuli (v případě, že $\varphi \models \mathbf{tt}$) nebo je třeba použít alespoň jednu klausuli (potom počet klausulí je číslo n z uvedené definice). Lze však použít klausule kladné délky nebo klausule délky 0.

Analogicky lze rozebrat definici konjunktivní normální formy formule.

2.2.1 Příklady Uveďme nyní nějaké příklady. Předpokládáme, že symboly a, b, c jsou atomické formule.

1. Pro formuli $\varphi = a \Rightarrow b$ je formule $\psi = \neg a \vee b$ její disjunktivní normální formou. Především zřejmě platí $\psi \models \varphi$. Dále je formule ψ sestavena ze dvou klausulí pro DNF: $\neg a$ a b . Jak $\neg a$, tak b jsou klausule pro DNF, protože a a b jsou atomické formule. Abychom zdůraznili, že ψ má dvě klausule pro DNF, budeme psát $\psi = (\neg a) \vee (b)$. Každá z těchto klausulí má délku jedna, protože každá obsahuje pouze jeden literál.

Na formuli ψ se lze ovšem dívat i jako na konjunktivní tvar formule φ : $\neg a \vee b$ je klausule pro CNF, $\psi = (\neg a \vee b)$. Tato jediná klausule má délku dva — obsahuje totiž dva literály.

Tento příklad ukazuje, že stejný řetězec může být chápán jako disjunktivní i konjunktivní normální forma. Taková situace ale není typická.

2. Pro formuli $\varphi = a \Rightarrow a$ je možné nalézt DNF například takto:

$$a \Rightarrow a \models (a) \vee (\neg a)$$

Použili jsme dvě klausule pro DNF: a (klausule délky 1) a $\neg a$ (klausule délky 1).

Lze však napsat i

$$a \Rightarrow a \models \mathbf{tt}$$

a to je podle definice opět DNF formule φ . Tato DNF má nulový počet klausulí.

Hledáme-li CNF pro formuli $\varphi = a \Rightarrow a$, využijeme opět vztahu

$$a \Rightarrow a \models (a \vee \neg a)$$

Použili jsme jednu klausuli pro CNF: $a \vee \neg a$ (klausule délky 2).

3. Uvažujme ještě jednou o formuli $\varphi = a \Rightarrow b$. Pojem DNF, který jsme zavedli, nám dovoluje za DNF pro φ považovat i formuli $\psi = (\neg a) \vee (b) \vee (\mathbf{ff})$. Na tvorbu DNF jsme tentokrát použili tři klausule, třetí klausule je délky nula.

Analogicky zkuste zapsat CNF pro formuli ψ použitím tří klausulí pro CNF.

Zjišťujeme tedy, že úloha *najděte DNF a CNF* nemá jednoznačné řešení.

4. Pro formuli $\varphi = \neg a \Rightarrow (b \wedge c)$ nalezneme DNF snadno:

$$\neg a \Rightarrow (b \wedge c) \models a \vee (b \wedge c),$$

Tedy DNF má dvě klausule: a (klausule délky jedna) a $b \wedge c$ (klausule délky dva).

Použitím distributivního zákona na formuli $a \vee (b \wedge c)$ pak dostáváme

$$a \vee (b \wedge c) \models (a \vee b) \wedge (a \vee c)$$

a formule napravo je CNF formule φ . Obsahuje dvě klausule: $a \vee b$ a $a \vee c$, obě klausule mají délku dvě.

Než budeme definovat Karnaughovy mapy, vyřešíme úkol najít *jakoukoli* DNF a CNF pro danou formuli elementárními prostředky. Metoda Karnaughových map bude zobecněním následujícího postupu.

2.2.2 Příklad Předpokládejme, že symboly a, b, c jsou atomické formule. Nalezněme jakoukoli DNF a jakoukoli CNF pro formuli $\varphi = (b \Rightarrow \neg c) \Leftrightarrow ((a \wedge c) \Rightarrow b)$. Nejprve spočítáme obvyklým způsobem pravdivostní tabulku pro formuli φ .

a	b	c	φ
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Začněme hledat jakoukoli DNF formule φ . Nejprve si uvědomme, že hledáme formuli ψ , která je synonymem formule φ , tj. platí $\psi \models \varphi$ a navíc formule ψ je disjunkcí nějakých dalších formulí (které jsou klausulemi pro DNF). Především požadavek $\psi \models \varphi$ vyžaduje, aby pravdivostní tabulka pro ψ byla stejná jako pravdivostní tabulka pro formuli φ . Díváme-li se na poslední sloupec tabulky pro φ jako na vektor, pak sloupec pro ψ musí být vytvořen „logickým součtem“ nějakých dalších vektorů nul a jedniček délky osm. Každý z těchto vektorů musí být pravdivostní tabulkou klausule pro DNF.

Uvažujme o následujících osmi formulích f_1, f_2, \dots, f_8 , které zadáme pravdivostními tabulkami:

a	b	c	f_1	f_2	\dots	f_8
0	0	0	1	0		0
0	0	1	0	1		0
0	1	0	0	0		0
0	1	1	0	0		0
1	0	0	0	0		0
1	0	1	0	0		0
1	1	0	0	0		0
1	1	1	0	0		1

Zřejmě $\varphi \models f_1 \vee f_2 \vee f_3 \vee f_5 \vee f_7$. Úloha bude vyřešena, pokud ukážeme, že každá z formulí f_1, f_2, f_3, f_5, f_7 odpovídá klausuli pro DNF. Platí však více: všechny formule f_1 až f_8 lze chápat jako klausule pro DNF:

$$\begin{array}{ll}
 f_1 \models (\neg a \wedge \neg b \wedge \neg c) & f_5 \models (a \wedge \neg b \wedge \neg c) \\
 f_2 \models (\neg a \wedge \neg b \wedge c) & f_6 \models (a \wedge \neg b \wedge c) \\
 f_3 \models (\neg a \wedge b \wedge \neg c) & f_7 \models (a \wedge b \wedge \neg c) \\
 f_4 \models (\neg a \wedge b \wedge c) & f_8 \models (a \wedge b \wedge c)
 \end{array}$$

DNF pro formuli φ je tedy formule

$$(\neg a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge \neg c) \vee (a \wedge b \wedge \neg c).$$

Analogicky, hledáme-li jakoukoli CNF formule φ , uvědomme si, že hledáme formuli ψ , která je synonymem formule φ , tj. platí $\psi \models \varphi$ a navíc formule ψ je konjunkcí nějakých dalších formulí (které jsou klausulemi pro CNF). Opět požadavek $\psi \models \varphi$ vyžaduje, aby pravdivostní tabulka pro ψ byla stejná jako pravdivostní tabulka pro formuli φ . Díváme-li se na poslední sloupec tabulky pro φ jako na vektor, pak sloupec pro ψ musí být vytvořen logickým součinem nějakých dalších vektorů nul a jedniček délky osm. Každý z těchto vektorů musí být pravdivostní tabulkou klausule pro CNF.

Uvažujme o následujících osmi formulích g_1, g_2, \dots, g_8 , které zadáme pravdivostními tabulkami:

a	b	c	g_1	g_2	\dots	g_8
0	0	0	0	1		1
0	0	1	1	0		1
0	1	0	1	1		1
0	1	1	1	1		1
1	0	0	1	1		1
1	0	1	1	1		1
1	1	0	1	1		1
1	1	1	1	1		0

Zřejmě $\varphi \models g_4 \wedge g_6 \wedge g_8$. Úloha bude vyřešena, pokud ukážeme, že každá z formulí g_4, g_6, g_8 odpovídá klausuli pro CNF. Opět platí více: všechny formule g_1 až g_8 lze chápat jako klausule pro CNF:

$$\begin{aligned} g_1 &\models (a \vee b \vee c) & g_5 &\models (\neg a \vee b \vee c) \\ g_2 &\models (a \vee b \vee \neg c) & g_6 &\models (\neg a \vee b \vee c) \\ g_3 &\models (a \vee \neg b \vee c) & g_7 &\models (\neg a \vee \neg b \vee c) \\ g_4 &\models (a \vee \neg b \vee \neg c) & g_8 &\models (\neg a \vee \neg b \vee \neg c) \end{aligned}$$

CNF pro formuli φ je tedy formule

$$(a \vee \neg b \vee \neg c) \wedge (\neg a \vee b \vee c) \wedge (\neg a \vee \neg b \vee \neg c).$$

Předchozí způsob výpočtu DNF a CNF nebyl možná ten nejefektivnější. Vyzkoušejte si, že např. formule

$$(\neg b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge b \wedge \neg c)$$

je též DNF pro formuli φ , navíc obsahuje jako řetězec méně znaků než

$$(\neg a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge \neg c) \vee (a \wedge b \wedge \neg c).$$

Z další části bude patrné, že způsob hledání DNF a CNF, který jsme předvedli v příkladu 2.2.2, vždy vede na řetězec, který obsahuje maximální počet znaků.

Zavedme následující pojem:

2.2.3 Definice At φ je formule výrokové logiky. At $\{a_1, \dots, a_s\}$ je seznam všech navzájem různých atomických formulí obsažených ve formuli φ . Řekneme, že formule ψ je

- *úplnou DNF pro formuli φ* , pokud ψ je DNF formule φ a navíc každá klausule pro DNF v ψ obsahuje přesně s různých literálů. V žádné klausuli však nesmí být obsaženy dva literály obsahující tutéž atomickou formuli (tj. například a a $\neg a$ současně).
- *úplnou CNF pro formuli φ* , pokud ψ je CNF formule φ a navíc každá klausule pro CNF v ψ obsahuje přesně s různých literálů. V žádné klausuli však nesmí být obsaženy dva literály obsahující tutéž atomickou formuli (tj. například a a $\neg a$ současně).

2.2.4 Příklad Zřejmě formule

$$(\neg a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge \neg c) \vee (a \wedge b \wedge \neg c)$$

je úplná DNF formule φ z příkladu 2.2.2. Formule

$$(\neg b \wedge \neg c) \vee (\neg a \wedge c) \vee (a \wedge b \wedge \neg c)$$

je samozřejmě DNF formule φ , ovšem nejde o úplnou DNF, protože například klausule $(\neg b \wedge \neg c)$ obsahuje pouze dva literály.

2.2.5 Příklad Protože pro formuli $\varphi = a \Rightarrow a$ platí $\varphi \models \neg a \vee a$, obsahuje CNF pro φ jedinou klausuli $(\neg a \vee a)$.

Úplná CNF pro φ však *neexistuje*. Klausule $(\neg a \vee a)$ totiž obsahuje dva literály se stejnou atomickou formuli. Formule φ však samozřejmě má úplnou DNF: jde o formuli $(\neg a) \vee (a)$, která obsahuje dvě klausule.

Příklad 2.2.2 lze snadno zobecnit a dát tak obecný návod na hledání úplné DNF a úplné CNF dané formule.



2.2.6 Jak najít úplnou DNF a úplnou CNF dané formule Ať je dána formule φ výrokové logiky. Ať $\{a_1, \dots, a_s\}$ je seznam všech navzájem různých atomických formulí obsažených ve formuli φ . Předpokládejme, že ve všech pravdivostních tabulkách, o kterých v následujícím budeme mluvit, dodržujeme stejné pořadí řádků.

1. Sestavte pravdivostní tabulku formule φ . Protože je $\{a_1, \dots, a_s\}$ seznam všech navzájem různých atomických formulí obsažených ve formuli φ , má tato tabulka $r = 2^s$ řádků.
2. **Úplná DNF.** Zaveďte formule f_1 až f_r , které mají následující pravdivostní tabulky: formule f_i ($i \in \{1, \dots, r\}$) má v pravdivostní tabulce pouze jedinou jedničku a to právě na i -tém řádku pravdivostní tabulky tak, aby každá z formulí f_1 až f_r byla klausule pro DNF.

Pokud na žádném řádku pravdivostní tabulky formule φ není jednička, úplná DNF formule φ *neexistuje*. Tento případ nastává právě tehdy, když formule φ je kontradikce. *Žádná kontradikce tedy nemá úplnou DNF.*

Pokud v pravdivostní tabulce formule φ je alespoň jedna jednička, označte jako $\{r_1, \dots, r_k\}$ ($k \geq 1$) seznam všech různých řádků pravdivostní tabulky formule φ , na kterých je φ ohodnocena jedničkou.

Úplnou DNF formule φ dostaneme jako disjunkci

$$f_{r_1} \vee \dots \vee f_{r_k}.$$

3. **Úplná CNF.** Zaveďte formule g_1 až g_r , které mají následující pravdivostní tabulky: formule g_i ($i \in \{1, \dots, r\}$) má v pravdivostní tabulce pouze jedinou nulu a to právě na i -tém řádku pravdivostní tabulky tak, aby každá z formulí g_1 až g_r byla klausule pro CNF.

Pokud na žádném řádku pravdivostní tabulky formule φ není nula, úplná CNF formule φ *neexistuje*. Tento případ nastává právě tehdy, když formule φ je tautologie. *Žádná tautologie tedy nemá úplnou CNF.*

Pokud v pravdivostní tabulce formule φ je alespoň jedna nula, označte jako $\{r_1, \dots, r_k\}$ ($k \geq 1$) seznam všech různých řádků pravdivostní tabulky formule φ , na kterých je φ ohodnocena nulou.

Úplnou CNF formule φ dostaneme jako konjunkci

$$g_{r_1} \wedge \dots \wedge g_{r_k}.$$

Neefektivita (tj. možná zbytečně velký počet znaků), která se projevila při výpočtu DNF a CNF metodou příkladu 2.2.2, byla způsobena následujícím (popíšeme situaci pouze pro DNF, pro CNF je situace analogická):

- Při hledání DNF jsme se zaměřili na *jednotlivé* jedničky v pravdivostní tabulce dané formule φ . Příslušné formule f_i (jejich pravdivostní tabulka obsahovala pouze jednu jedničku) sice měly tu výhodu, že jsme byli každé f_i schopni okamžitě zapsat jako klausuli, ovšem možná by šlo uvažovat o jiných formulích, které by v pravdivostní tabulce měly více jedniček. My bychom pak mohli „vyřídít“ více jedniček v pravdivostní tabulce formule φ najednou.

Výše uvedená myšlenka je velmi přitažlivá, má však svá úskalí, na která upozorňuje následující příklad.

2.2.7 Příklad Vezměme opět pravdivostní tabulku pro formuli φ z příkladu 2.2.2.

a	b	c	φ
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

a uvažujme o prvních dvou jedničkách v posledním sloupci. Kdybychom ukázali, že existuje formule f , která je klausule pro DNF a která má následující pravdivostní tabulku

a	b	c	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

mohli bychom psát DNF pro φ jako $f \vee f_3 \vee f_5 \vee f_7$. Formule f skutečně je klausulí pro DNF, a sice $(\neg a \wedge \neg b)$. Navíc DNF ve tvaru $f \vee f_3 \vee f_5 \vee f_7$ obsahuje zřejmě méně znaků než DNF ve tvaru $f_1 \vee f_2 \vee f_3 \vee f_5 \vee f_7$ z příkladu 2.2.2. Zdá se, že čím více jedniček budeme mít ve formuli, kterou „pokrýváme“ jedničky v pravdivostní tabulce formule φ , tím méně znaků na vyjádření DNF budeme potřebovat.

Uvažujme tedy o formuli h jejíž pravdivostní tabulka vypadá následovně:

a	b	c	h
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Pokud h je klausule pro DNF, budeme pro vyjádření φ potřebovat ještě méně znaků. Narážíme však na následující problém: h není (žádnou) klausulí pro DNF! Ukažme to: postupujeme sporem, ať h je klausule pro DNF. Může nastat právě jeden z následujících čtyř případů:

1. h obsahuje tři literály. Pak je buď $h = (a \wedge l_1 \wedge l_2)$ nebo $h = (\neg a \wedge l_1 \wedge l_2)$, kde l_1 a l_2 jsou literály.
 - (a) První případ nemůže nastat — na prvním řádku pravdivostní tabulky by pak nemohla být jednička.
 - (b) V druhém případě: h nemůže obsahovat literál c (první řádek pravdivostní tabulky) ani literál $\neg c$ (druhý řádek pravdivostní tabulky).

Předpoklad, že v h jsou tři literály, vede ke sporu.
2. h obsahuje právě dva literály. Zaměříme se na literál, který h neobsahuje. Může nastat právě jeden z následujících tří případů:
 - (a) h neobsahuje literál, ve kterém je atomická formule a . To ale není možné: klausule h pak totiž nemůže obsahovat ani literál b (viz první řádek pravdivostní tabulky), ani literál $\neg b$ (viz třetí řádek pravdivostní tabulky).
 - (b) h neobsahuje literál, ve kterém je atomická formule b . To ale není možné: klausule h pak totiž nemůže obsahovat ani literál c (viz první řádek pravdivostní tabulky), ani literál $\neg c$ (viz druhý řádek pravdivostní tabulky).
 - (c) h neobsahuje literál, ve kterém je atomická formule c . To ale není možné: klausule h pak totiž nemůže obsahovat ani literál b (viz první řádek pravdivostní tabulky), ani literál $\neg b$ (viz třetí řádek pravdivostní tabulky).

Předpoklad, že v h jsou právě dva literály, vede ke sporu.

- h obsahuje pouze jeden literál l . Pak $h = l$ a nastává jeden z šesti případů: $h = a$, $h = b$, $h = c$, $h = \neg a$, $h = \neg b$, $h = \neg c$. Z pravdivostní tabulky pro h však plyne, že ani jeden z těchto případů nenastává. To je spor — h nemůže obsahovat právě jeden literál.
- h neobsahuje žádný literál. Potom je h klausule pro DNF délky nula, neboli $h = ff$. Z pravdivostní tabulky ihned plyne, že to není možné.

Předchozí příklad naznačuje, že není jednoduché rozhodnout, který vektor nul a jedniček odpovídá přesně jedné klausuli. Uvidíme, že tento problém budeme moci snadno rozhodnout, pokud jednorozměrnou pravdivostní tabulku formule přepíšeme do dvourozměrné tabulky (takové tabulce budeme říkat *Karnaughova mapa*). V Karnaughově mapě pak budeme:

- Pro DNF pokrývat ty plochy jedniček, které odpovídají klausulím pro DNF.
- Pro CNF pokrývat ty plochy nul, které odpovídají klausulím pro CNF.

Karnaughovým mapám je věnována další část textu.

V předchozí části jsme viděli, že jak DNF, tak CNF dané formule nejsou obecně určeny jednoznačně. Navíc metoda výpočtu DNF a CNF, kterou jsme předvedli v příkladu 2.2.2, vede na úplné formy, které jako řetězce obsahují maximální možný počet znaků.

Ukážeme nyní způsob výpočtu DNF a CNF, který má tu výhodu, že získáme kontrolu nad počtem znaků ve formách, které nalezneme. Budeme tak například vědět, zda počet znaků v DNF, kterou jsme spočetli, lze dále zmenšit, či nikoli.

Algoritmus této části je opřen o jiný způsob zápisu pravdivostní tabulky formule. Tomuto způsobu zápisu pravdivostní tabulky říkáme *Karnaughova mapa*. Pozor! V tomto textu se zaměříme na tvorbu Karnaughových map pouze pro formule, ve kterých se vyskytují nanejvýš čtyři atomické formule.

Než řekneme, co je Karnaughova mapa obecně, uvedeme příklad.

2.2.8 Příklad

Připomeňme pravdivostní tabulku pro formuli φ z příkladu 2.2.2.

a	b	c	φ
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Tuto pravdivostní tabulku (její poslední sloupec) nyní přepíšeme do matice. Tuto matici si představíme jako „mapu“ v pravoúhlém systému souřadnic, na její svislou osu budeme nanášet ohodnocení atomické formule a počínaje 0 (matice tedy bude mít dva řádky) a na její vodorovnou osu budeme nanášet ohodnocení *uspořádané dvojice atomických formulí* bc počínaje 00 (matice tedy bude mít čtyři sloupce). Navíc budeme při nanášení souřadnic na osy dodržovat následující pravidlo:

(*) susedí přesně ty souřadnice, které se liší právě v jedné položce.

Položky vzniklé matice pak obsadíme nulami nebo jedničkami tak, jak nám velí pravdivostní tabulka formule φ . Dostaneme následující matici:

$$\begin{array}{c|c|c|c|c}
 a \backslash bc & 00 & 01 & 11 & 10 \\
 \hline
 0 & 1 & 1 & 0 & 1 \\
 1 & 1 & 0 & 0 & 1
 \end{array} \tag{2.1}$$

Této matici říkáme Karnaughova mapa formule φ .

Poznamenejme ještě, že pro situaci, kdy je první souřadnice 0, nám pravidlo (*) nedává žádnou možnost volby pořadí souřadnic ve svislém směru: souřadnice 0 podle pravidla (*) musí sousedit se souřadnicí 1 a jiné souřadnice ve svislém směru nejsou. Jiná je situace ve směru vodorovném. Především pro naše pořadí (zleva doprava) 00, 01, 11, 10 pravidlo (*) říká, že spolu mají sousedit i souřadnice 00 a 10. Požadavek (*) nás tedy nutí dívat se na Karnaughovu mapu tak, jako by byla nakreslena na válci, který obdržíme „přilepením“ pravého okraje sloupce 10 k levému okraji sloupce 00:



Ovšem pořadí vodorovných souřadnic 00, 10, 11, 01 také vyhovuje pravidlu (*). Pro tutéž formuli φ tak dostáváme *jinou* Karnaughovu mapu

$a \backslash bc$	00	10	11	01
0	1	1	0	1
1	1	1	0	0

Pravidlo (*) nám však opět říká, že souřadnice 00 a 01 spolu sousedí. Na Karnaughovu mapu (2.3) se tudíž musíme také dívat jako na válcovou plochu, kde jsme tentokrát „slepili“ pravý okraj sloupce 01 s levým okrajem sloupce 00:



Je jasné, že válcové plochy (2.2) a (2.4) se od sebe liší pouze orientací: pokud „projdeme“ válcovou plochu (2.2) proti směru hodinových ručiček, budeme postupně míjet tytéž souřadnice, jako při průchodu válcovou plochou (2.4) po směru hodinových ručiček.



2.2.9 Jak napsat Karnaughovu mapu libovolné formule, která obsahuje nanejvýš čtyři atomické formule Ať je dána formule φ výrokové logiky. Ať $\{a_1, \dots, a_s\}$ je seznam všech navzájem různých atomických formulí obsažených ve formuli φ . Předpokládejme, že jsme již sestavili pravdivostní tabulku formule φ . Protože je $\{a_1, \dots, a_s\}$ seznam všech navzájem různých atomických formulí obsažených ve formuli φ , má tato tabulka 2^s řádků.

Podle předpokladu je $s \leq 4$. Rozebereme nyní jednotlivé případy.

Pokud je $s = 1$, má naše pravdivostní tabulka 2 řádky a Karnaughovu mapu formule φ nelze sestavit.

Pokud je $s = 2$, má pravdivostní tabulka 4 řádky a příslušná Karnaughova mapa má tvar:

$a_1 \backslash a_2$	0	1
0		
1		

kde prázdná čtyři políčka vyplníme tak, jak nám velí pravdivostní tabulka formule φ .

V případě $s = 3$, má Karnaughova mapa tvar:

$a_1 \backslash a_2 a_3$	00	01	11	10
0				
1				

kde prázdných osm políček vyplníme tak, jak nám velí pravdivostní tabulka formule φ . Pozor! Za sousední přitom považujeme právě ta políčka, jejichž souřadnice se liší v jednom bitu. Například „vodorovné“ souřadnice 00 a 10 spolu sousedí.

V případě $s = 4$, má Karnaughova mapa tvar:

$a_1 a_2 \backslash a_3 a_4$	00	01	11	10
00				
01				
11				
10				

kde prázdných šestnáct políček vyplníme tak, jak nám velí pravdivostní tabulka formule φ . Pozor! Za sousední přitom považujeme právě ta políčka, jejichž souřadnice se liší v jednom bitu. Například „svislé“ souřadnice 00 a 10 spolu sousedí.

Dvourozměrné tabulce vytvořené podle výše uvedených pravidel říkáme *Karnaughova mapa formule φ* .

2.2.10 Poznámka

K definici Karnaughovy mapy nyní přičiníme několik poznámek.

1. V případě, že formule φ obsahuje pouze jednu atomickou formuli, tj. v případě, kdy $s = 1$, Karnaughovu mapu nelze sestavit. Jak v tomto případě vypadá zjednodušování DNF a CNF?

Označme jedinou atomickou formuli, která je ve formuli φ obsažena, písmenem a . Formule φ pak má právě jednu z následujících čtyř pravdivostních tabulek:

a	φ
0	1
1	1

a	φ
0	0
1	1

a	φ
0	1
1	0

a	φ
0	0
1	0

Probereme podrobně pouze tvorbu DNF, jinak odkazujeme čtenáře na cvičení 2.4.3.

V případě první pravdivostní tabulky je zřejmě $\varphi \models \mathbf{tt}$. Formuli φ lze tudíž popsat nulovým počtem klausulí pro DNF. Tato forma zřejmě obsahuje nejmenší možný počet znaků, totiž nulový.

V případě druhé pravdivostní tabulky je $\varphi \models a$. Jde o DNF, která obsahuje jedinou klausuli délky jedna — obsahuje tedy nejmenší počet znaků.

V případě třetí pravdivostní tabulky je zřejmě $\varphi \models \neg a$. Tudíž formule $(\neg a)$ DNF formule φ a tato forma již zřejmě obsahuje nejmenší možný počet znaků.

V případě čtvrté pravdivostní tabulky je zřejmě $\varphi \models \mathbf{ff}$. Formule \mathbf{ff} je klausule pro DNF, která obsahuje nejmenší možný počet znaků (sice nula znaků). Počet znaků této formy již zřejmě nejde zmenšit.

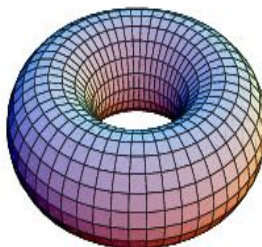
2. Příklad 2.2.8 nás poučil o tom, že v případě tří atomických formulí je třeba chápat Karnaughovu mapu jako „nakreslenou na válci“. Jak chápat Karnaughovy mapy v případě čtyř atomických formulí? Nyní totiž čelíme problému „slepování“ krajů mapy dvakrát: jak ve směru vodorovném, tak ve směru svislém. Například v následující Karnaughově mapě

$ab \backslash cd$	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	1	0
10	0	0	0	0

musíme „slepit“:

- (a) Pravý okraj sloupce 10 s levým okrajem sloupce 00.
- (b) Dolní okraj řádku 10 s horním okrajem sloupce 00.

Výslednou Karnaughovu mapu je tedy třeba představit si jako nakreslenou na povrchu toru. *Torus* (nebo též *anuloid*) je prostorové těleso, tvarem připomínající nafouklou pneumatiku:



Proč jsme do definice Karnaughovy mapy dali požadavky na sousedění políček, jejichž souřadnice se liší právě v jedné položce? Přesně tyto požadavky totiž umožní v Karnaughově mapě detekovat ty plochy jedniček, které odpovídají klausulím pro DNF, a ty plochy nul, které odpovídají klausulím pro CNF.

3. V případě, že formule φ obsahuje pět a více atomických formulí, je situace daleko komplikovanější. Odkazujeme na cvičení 2.4.6.

Zřejmě každou Karnaughovu mapu lze považovat za matici nul a jedniček. Pohled na pravdivostní tabulku jako na (speciálně vytvořenou) matici nám umožní zobecnit postup z příkladu 2.2.2. V daném příkladu jsme totiž například úplnou DNF vytvořili tak, že jsme se zaměřili na jednotlivé jedničky v sloupci pravdivostní tabulky. Tento vektor jsme se pak pokoušeli „nakombinovat“ pomocí logického součtu a sady „bázických“ vektorů. Každý z bázických vektorů přitom odpovídal pravdivostní tabulce jedné klausule pro DNF. Nyní se pokusíme o analogii tohoto postupu pro matice nul a jedniček. Musíme se přitom postarat o následující:

- Musíme vhodně vybrat „bázické“ matice, pomocí nichž jsme schopni nakombinovat danou matici (Karnaughovu mapu). Navíc, každá z těchto bázických matic musí odpovídat klausuli.

To je smyslem následující definice:

2.2.11 Definice *Bázická matice jedniček* je taková Karnaughova mapa (chápaná jako matice nul a jedniček), ve které všechny jedničky tvoří obdélník vytvořený ze sousedících políček. Navíc rozměry tohoto obdélníka musí být mocniny čísla dvě.

Bázická matice nul je taková Karnaughova mapa (chápaná jako matice nul a jedniček), ve které všechny nuly tvoří obdélník vytvořený ze sousedících políček. Navíc rozměry tohoto obdélníka musí být mocniny čísla dvě.

2.2.12 Příklady Uveďme příklady bázických matic a matic, které nejsou bázické. Příslušný obdélník nul nebo jedniček vyznačíme tučně. Předpokládejme, že a, b, c, d jsou atomické formule.

1. Karnaughova mapa

$ab \backslash cd$	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	1	0
10	0	0	0	0

není ani *bázickou maticí jedniček* ani *bázickou maticí nul*. Neobsahuje totiž ani *obdélník jedniček* ani *obdélník nul*.

2. Karnaughova mapa

$ab \backslash cd$	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

je *bázická matice jedniček*. Rozměry *obdélníka jedniček* jsou 1×2 .

3. Karnaughova mapa

$ab \backslash cd$	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

je *bázická matice jedniček*. Rozměry *obdélníka jedniček* jsou 2×2 .

4. Karnaughova mapa

$ab \backslash cd$	00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	1	1	0	0
10	1	1	1	1

je *bázická matice nul*. Rozměry *obdélníka nul* jsou 2×2 .

5. Karnaughova mapa

$ab \backslash cd$	00	01	11	10
00	1	1	1	1
01	1	0	0	0
11	1	0	0	0
10	1	1	1	1

není bázičká matice nul. Rozměry obdélníka nul jsou 2×3 .

6. Karnaughova mapa

$ab \backslash cd$	00	01	11	10
00	0	1	1	1
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

není bázičká matice jedniček. Rozměry obdélníka jedniček jsou 1×3 .

Následující tvrzení o souvislosti bázičkových matic a klausulí uvedeme bez důkazu.

2.2.13 Tvrzení Platí následující:

1. Každá bázičká matice jedniček odpovídá právě jedné klausuli pro DNF.
2. Každá bázičká matice nul odpovídá právě jedné klausuli pro CNF.

2.2.14 Příklady Uvedeme nyní příklady korespondence klausulí a bázičkových matic.

1. Bázičké matice jedniček

$ab \backslash cd$	00	01	11	10
00	0	0	0	1
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

odpovídá následující klausule pro DNF: $(\neg a \wedge \neg b \wedge c \wedge \neg d)$.

2. Bázičké matice nul

$ab \backslash cd$	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

odpovídá následující klausule pro DNF: $(\neg a \wedge \neg b \wedge \neg d)$.

3. Bázičké matice jedniček

$ab \backslash cd$	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

odpovídá následující klausule pro DNF: $(\neg b \wedge \neg d)$.

4. Bázické matice nul

$ab \backslash cd$	00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	1	1	0	0
10	1	1	1	1

odpovídá následující klausule pro CNF: $(\neg b \vee \neg c)$.

Z předchozího příkladu se zdá zřejmé následující pozorování:

Předpokládejme, že studujeme bázické matice jedniček pevných rozměrů. Pak zřejmě čím větší je obdélník jedniček, tím méně atomických formulí je zapotřebí k zapsání příslušné klausule.

Analogické pozorování se zdá platit pro bázické matice nul. Přesné znění a důkaz příslušného tvrzení vynecháme a zapamatujeme si slogan:

K popisu větší plochy na mapě je třeba méně informace.

Pokud tedy budeme chtít pro danou formuli φ nalézt například CNF, která bude mít co možná nejkratší klausule, budeme muset dodržet dvě věci:

1. Karnaughova mapa formule φ musí být „logickým součinem“ bázických matic nul. (Pak bude zaručeno, že formule φ bude konjunkcí klausulí, které odpovídají použitým bázickým maticím nul.)
2. Každá bázická matice nul, kterou použijeme, musí mít co největší obdélník nul. (Pak bude zaručeno, že každá klausule pro CNF, kterou použijeme, bude obsahovat co nejmenší počet literálů.)

Takových bázických matic nul musíme použít co nejmenší počet. (Pak bude zaručeno, že CNF formule φ bude obsahovat co nejmenší počet znaků.)

2.2.15 Definice Ať \mathbb{A} a \mathbb{B} jsou dvě Karnaughovy mapy stejných rozměrů.

1. *Logickým součtem* \mathbb{A} a \mathbb{B} myslíme Karnaughovu mapu označenou $\mathbb{A} \boxplus \mathbb{B}$, kterou obdržíme tak, že příslušné položky v jednotlivých mapách logicky sečteme.

Budeme říkat, že mapa \mathbb{C} je *nakombinována z map \mathbb{A} a \mathbb{B} logickým součtem*, pokud platí $\mathbb{C} = \mathbb{A} \boxplus \mathbb{B}$.

2. *Logickým součinem* \mathbb{A} a \mathbb{B} myslíme Karnaughovu mapu označenou $\mathbb{A} \boxtimes \mathbb{B}$, kterou obdržíme tak, že příslušné položky v jednotlivých mapách logicky vynásobíme.

Budeme říkat, že mapa \mathbb{C} je *nakombinována z map \mathbb{A} a \mathbb{B} logickým součinem*, pokud platí $\mathbb{C} = \mathbb{A} \boxtimes \mathbb{B}$.

2.2.16 Příklad Pro následující dvě Karnaughovy mapy

$ab \backslash cd$	00	01	11	10
00	0	1	1	1
01	0	0	0	0
11	0	0	0	0
10	0	0	1	0

 $\mathbb{A} =$

$ab \backslash cd$	00	01	11	10
00	0	1	1	1
01	0	0	0	1
11	0	0	1	1
10	0	0	0	0

 $\mathbb{B} =$

je

$ab \backslash cd$	00	01	11	10
00	0	1	1	1
01	0	0	0	1
11	0	0	1	1
10	0	0	1	0

 $\mathbb{A} \boxplus \mathbb{B} =$

a

$ab \backslash cd$	00	01	11	10
00	0	1	1	1
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

 $\mathbb{A} \boxtimes \mathbb{B} =$

Zřejmě operace \boxplus odpovídá logické spojce \vee a operace \boxtimes odpovídá logické spojce \wedge . Zformulujme to přesně.

2.2.17 Tvzení *At φ a ψ jsou formule výrokové logiky obsahující stejný počet atomických formulí. At \mathbb{A} je Karnaughova mapa formule φ a \mathbb{B} je Karnaughova mapa formule ψ . Potom platí:*

1. $\mathbb{A} \boxplus \mathbb{B}$ je Karnaughova mapa formule $\varphi \vee \psi$.
2. $\mathbb{A} \boxtimes \mathbb{B}$ je Karnaughova mapa formule $\varphi \wedge \psi$.

2.2.18 Definice Řekneme, že DNF (CNF) dané formule je *ireducibilní*, pokud obsahuje nejmenší možný počet znaků.



2.2.19 Jak najít ireducibilní DNF a ireducibilní CNF dané formule At je dána formule φ výrokové logiky.

1. Sestavte Karnaughovu mapu formule φ . Vzniklou matici nul a jedniček označte \mathbb{A} .
2. **Ireducibilní DNF.** Jestliže mapa \mathbb{A} obsahuje samé nuly, ireducibilní DNF formule φ je formule **ff** (ireducibilní DNF má jedinou klausuli délky nula).

Jestliže mapa \mathbb{A} obsahuje samé jedničky, ireducibilní DNF formule φ je formule **tt** (ireducibilní DNF má nulový počet klausulí).

Pokud nenastane ani jeden z výše uvedených případů, nakombinujte Karnaughovu mapu \mathbb{A} logickým součtem bázičických matic jedniček $\mathbb{A}_1, \dots, \mathbb{A}_n$. Dodržujte přitom následující dvě podmínky:

- (a) Každá matice \mathbb{A}_i musí mít co největší obdélník jedniček.
- (b) Počet matic \mathbb{A}_i (tj. číslo n) musí být co nejmenší.

Nalezněte klausule f_1, \dots, f_n pro DNF, které odpovídají bázičickým maticím $\mathbb{A}_1, \dots, \mathbb{A}_n$. Ireducibilní DNF formule φ je $f_1 \vee \dots \vee f_n$.

3. **Ireducibilní CNF.** Jestliže mapa \mathbb{A} obsahuje samé jedničky, ireducibilní CNF formule φ je formule **tt** (ireducibilní CNF má jedinou klausuli délky nula).

Jestliže mapa \mathbb{A} obsahuje samé nuly, ireducibilní CNF formule φ je formule **ff** (ireducibilní CNF má nulový počet klausulí).

Pokud nenastane ani jeden z výše uvedených případů, nakombinujte Karnaughovu mapu \mathbb{A} logickým součinem bázičických matic nul $\mathbb{A}_1, \dots, \mathbb{A}_n$. Dodržujte přitom následující dvě podmínky:

- (a) Každá matice \mathbb{A}_i musí mít co největší obdélník nul.
- (b) Počet matic \mathbb{A}_i (tj. číslo n) musí být co nejmenší.

Nalezněte klausule f_1, \dots, f_n pro CNF, které odpovídají bázičickým maticím $\mathbb{A}_1, \dots, \mathbb{A}_n$. Ireducibilní CNF formule φ je $f_1 \wedge \dots \wedge f_n$.

2.2.20 Příklad Nalezněme ireducibilní DNF a ireducibilní CNF formule φ , jejíž Karnaughova mapa vypadá následovně:

$$\mathbb{A} = \begin{array}{c|cccc} ab \backslash cd & 00 & 01 & 11 & 10 \\ \hline 00 & 0 & 1 & 1 & 1 \\ 01 & 0 & 0 & 0 & 0 \\ 11 & 0 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 & 0 \end{array}$$

1. **Ireducibilní DNF.** Matici \mathbb{A} lze nakombinovat logickým součtem následujícími bázičnými maticemi jedniček:

$$\mathbb{A}_1 = \begin{array}{c|cccc} ab \backslash cd & 00 & 01 & 11 & 10 \\ \hline 00 & 0 & 0 & 1 & 1 \\ 01 & 0 & 0 & 0 & 0 \\ 11 & 0 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 & 0 \end{array} \quad \mathbb{A}_2 = \begin{array}{c|cccc} ab \backslash cd & 00 & 01 & 11 & 10 \\ \hline 00 & 0 & 1 & 0 & 0 \\ 01 & 0 & 0 & 0 & 0 \\ 11 & 0 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 & 0 \end{array}$$

Matici \mathbb{A} jsme tedy nakombinovali logickým součtem dvou bázičných matic jedniček. Zřejmě nelze matici \mathbb{A} nakombinovat menším počtem bázičných matic.

Přesto nám matice \mathbb{A}_1 a \mathbb{A}_2 nedají ireducibilní DNF formule φ , protože k nakombinování matice \mathbb{A} lze použít i matice

$$\mathbb{B}_1 = \begin{array}{c|cccc} ab \backslash cd & 00 & 01 & 11 & 10 \\ \hline 00 & 0 & 0 & 1 & 1 \\ 01 & 0 & 0 & 0 & 0 \\ 11 & 0 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 & 0 \end{array} \quad \mathbb{B}_2 = \begin{array}{c|cccc} ab \backslash cd & 00 & 01 & 11 & 10 \\ \hline 00 & 0 & 1 & 1 & 0 \\ 01 & 0 & 0 & 0 & 0 \\ 11 & 0 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 & 0 \end{array}$$

kteří mají větší obdélníky jedniček než matice \mathbb{A}_1 a \mathbb{A}_2 . Současně je však vidět, že větší obdélníky jedniček již nenalezneme.

Protože matici \mathbb{B}_1 odpovídá klausule $\neg a \wedge \neg b \wedge c$ a matici \mathbb{B}_2 odpovídá klausule $\neg a \wedge \neg b \wedge d$, je ireducibilní DNF pro formuli φ následující:

$$(\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge \neg b \wedge d).$$

2. **Ireducibilní CNF.** Budeme postupovat trochu rychleji. Matici \mathbb{A} zřejmě nelze logickým součinem nakombinovat jednou bázičnou maticí nul. Lze se přesvědčit, že ji nelze nakombinovat ani s použitím dvou bázičných matic nul. Lze ji však nakombinovat následujícími třemi maticemi:

$$\mathbb{C}_1 = \begin{array}{c|cccc} ab \backslash cd & 00 & 01 & 11 & 10 \\ \hline 00 & 0 & 1 & 1 & 1 \\ 01 & 0 & 1 & 1 & 1 \\ 11 & 0 & 1 & 1 & 1 \\ 10 & 0 & 1 & 1 & 1 \end{array} \quad \mathbb{C}_2 = \begin{array}{c|cccc} ab \backslash cd & 00 & 01 & 11 & 10 \\ \hline 00 & 1 & 1 & 1 & 1 \\ 01 & 0 & 0 & 0 & 0 \\ 11 & 0 & 0 & 0 & 0 \\ 10 & 1 & 1 & 1 & 1 \end{array}$$

$$\mathbb{C}_3 = \begin{array}{c|cccc} ab \backslash cd & 00 & 01 & 11 & 10 \\ \hline 00 & 1 & 1 & 1 & 1 \\ 01 & 1 & 1 & 1 & 1 \\ 11 & 0 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 & 0 \end{array}$$

Tři základní matice nul, které by měly větší obdélníky nul než matice \mathbb{C}_1 , \mathbb{C}_2 a \mathbb{C}_3 , však nenalezneme. Matice \mathbb{C}_1 odpovídá klausuli $c \vee d$, matici \mathbb{C}_2 odpovídá klausuli $\neg b$ a matici \mathbb{C}_3 odpovídá klausuli $\neg a$. Ireducibilní CNF pro formuli φ je:

$$(c \vee d) \wedge (\neg b) \wedge (\neg a).$$

2.2.21 Poznámka Z výše uvedené teorie zřejmě plyne následující:

Pokud Karnaughovu mapu dané formule φ nakombinujeme pomocí základních matic jedniček a *nebudeme* přitom dodržovat požadavky **2a** a **2b** ze strany 30 dostaneme samozřejmě opět nějakou DNF formule φ .

Lze se snadno přesvědčit, že pokud budeme Karnaughovu mapu kombinovat pomocí základních matic jedniček obsahujících *pouze* obdélníky rozměrů 1×1 , dostaneme úplnou DNF formule φ .

Podobnou poznámku lze učinit pro CNF.

2.3 Sémantický důsledek ve výrokové logice

V tomto odstavci zformalizujeme ve výrokové logice důležitý útvar úsudku. Úsudek je v přirozeném jazyce forma zápisu, která odděluje *předpoklady* a *závěr*. Ve formálním pojetí logiky tedy předpoklady nahradíme (konečnou) množinou M formulí výrokové logiky a závěr bude jedna formule φ výrokové logiky. Celý úsudek budeme zapisovat takto

$$M \models \varphi \quad (\text{což čteme: formule } \varphi \text{ je sémantickým důsledkem množiny } M)$$

Poznamenejme, že symbol \models je *oddělovacím znaménkem*, které odděluje předpoklady od závěru (v českých úsudcích je takovým oddělovacím znaménkem slovo *tedy* — viz příklad 5.1.3).

Kdy je úsudek správný? Tehdy, když platí: kdykoli jsou pravda předpoklady, je pravda i závěr. Jde o sémantickou úvahu, takže nepřekvapí, že prvotní algoritmus pro zjišťování pravdivosti sémantického důsledku využívá prohledávání pravdivostní tabulky.

Nejprve však rozšíříme naši definici ohodnocení pro množiny formulí.

2.3.1 Definice Ať M je množina formulí výrokové logiky a ať u je ohodnocení výrokové logiky.

1. Zápisem $u(M) = 1$ rozumíme: $u(\alpha) = 1$ pro všechna $\alpha \in M$.
2. Zápisem $u(M) = 0$ rozumíme: $u(\alpha) = 0$ pro alespoň jedno $\alpha \in M$.

2.3.2 Příklad Ať $a, b \in At$ a ať $M = \{a \Rightarrow b, a \vee b\}$. Pravdivostní tabulka vypadá následovně

a	b	$a \Rightarrow b$	$a \vee b$
0	0	1	0
0	1	1	1
1	0	0	1
1	1	1	1

Pro ohodnocení $u(a) = 0, u(b) = 1$ (druhý řádek tabulky) platí $u(M) = 1$ (na příslušném řádku jsou jedničky pro všechny formule z množiny M).

Pro ohodnocení $u(a) = 1, u(b) = 0$ (třetí řádek tabulky) platí $u(M) = 0$ (na příslušném řádku formulí z množiny M jsme našli alespoň jednu nulu).

2.3.3 Definice Ať M je množina formulí výrokové logiky a ať φ je formule výrokové logiky. Řekneme, že sémantický důsledek $M \models \varphi$ *platí*, pokud

$$\text{pro všechna ohodnocení } u \text{ platí: } u(M) \leq u(\varphi).$$

2.3.4 Příklad Ať $a, b \in At$, ať $M = \{a \Rightarrow b, a \vee b\}$ a $\varphi = a \wedge b$. Rozhodněte, zda platí sémantický důsledek $M \models \varphi$.

Pravdivostní tabulka vypadá následovně

a	b	$a \Rightarrow b$	$a \vee b$	$a \wedge b$
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	1	1	1

Definice 2.3.3 nám radí: projděte všechny řádky tabulky (máme něco usoudit pro *všechna* ohodnocení) a na každém řádku se zeptejte, zda platí $u(M) \leq u(\varphi)$:

1. První řádek: $0 = u(M) \leq u(\varphi) = 0$.
2. Druhý řádek: $1 = u(M) \not\leq u(\varphi) = 0$.
3. Třetí řádek: $0 = u(M) \leq u(\varphi) = 0$.
4. Čtvrtý řádek: $1 = u(M) \leq u(\varphi) = 1$.

Na druhém řádku požadovaná nerovnost neplatí. Proto neplatí ani $M \models \varphi$.

2.3.5 Poznámka Algoritmus předchozího příkladu můžeme trochu zrychlit, když si uvědomíme následující fakta:

1. V ohodnocení (tj. na řádku) u , kdy $u(M) = 0$ nemusíme nerovnost $u(M) \leq u(\varphi)$ vůbec ověřovat, platí totiž automaticky.
2. V ohodnocení (tj. na řádku) u , kdy $u(M) = 1$ musíme ověřit, zda platí $u(\varphi) = 1$. Pak totiž platí nerovnost $u(M) \leq u(\varphi)$.

Dostáváme tedy standardní (ekvivalentní) definici sémantického důsledku: Řekneme, že sémantický důsledek $M \models \varphi$ platí, pokud

pro všechna ohodnocení u platí: jestliže $u(M) = 1$, pak $u(\varphi) = 1$.

Srovnajte s definicí ve skriptu

☞ M. Demlová a B. Pondělíček, *Matematická logika*, FEL ČVUT, Praha 1997

Než budeme moci vyslovit větu 2.3.10, která je základem resoučnických algoritmů, musíme zavést pojem splnitelné množiny formulí výrokové logiky.

2.3.6 Definice Ať M je množina formulí výrokové logiky. Řekneme, že M je *splnitelná množina* formulí, pokud existuje ohodnocení u takové, že $u(M) = 1$.

Řekneme, že formule φ výrokové logiky je *splnitelná*, pokud je množina $\{\varphi\}$ splnitelná množina formulí výrokové logiky.



2.3.7 Poznámka Pozor na terminologii: je stejná jako terminologie v češtině. Promyslete si rozdíl mezi *splnitelným* a *splněným* přáním.

Podobně: splnitelná množina M je ta, která může být splněna. Splněna je na řádku u , pro který platí $u(M) = 1$. Splnitelná formule φ je ta, která může být splněna. Splněna je na řádku u , pro který platí $u(\varphi) = 1$.

2.3.8 Příklad Ať $a, b \in At$. Pak platí:

1. Množina $\{a \Rightarrow b, a \vee b\}$ je splnitelná. Splněna je například na řádku $u(a) = 0, u(b) = 1$.

2. Množina $\{a \wedge \neg a\}$ není splnitelná. Neexistuje řádek pravdivostní tabulky, na kterém by byla splněna. To znamená, že formule $a \wedge \neg a$ není splnitelná.



2.3.9 Příklad Je množina \emptyset splnitelná nebo ne? Především si uvědomme, že pravdivostní tabulka množiny \emptyset má jediný řádek, protože počet atomických formulí ve všech formulích z množiny \emptyset je nula.

Na tomto jediném řádku (v ohodnocení u) musí být buď 0 nebo 1. Pojďme analyzovat první možnost:

Na řádku je 0 právě tehdy, když *existuje* formule $\alpha \in \emptyset$, pro kterou platí $u(\alpha) = 0$. Množina \emptyset však žádnou formuli neobsahuje, je totiž prázdná. Nemůže tedy existovat $\alpha \in \emptyset$, pro kterou platí $u(\alpha) = 0$.

Proto platí $u(\emptyset) = 1$ (dokonce pro jakékoli ohodnocení). Množina \emptyset je *splnitelná*.



2.3.10 Věta (O sémantickém důkazu sporem) Ať M je množina formulí výrokové logiky a ať φ je formule výrokové logiky. Pak jsou následující tvrzení ekvivalentní:

1. Sémantický důsledek $M \models \varphi$ platí.
2. Množina $M \cup \{\neg\varphi\}$ není splnitelná.

DŮKAZ. Podívejme se nejprve, co znamená mít ohodnocení u tak, že $u(M \cup \{\neg\varphi\}) = 1$: znamená to přesně tolik, že $u(M) = 1$ a současně $u(\varphi) = 0$. (Nakreslete si pravdivostní tabulku.)

Pak ale je tvrzení triviální. ■



2.3.11 Poznámka Proč se věta 2.3.10 jmenuje o sémantickém důkazu sporem? Formulují totiž přesně koncept důkazu sporem: chceme-li dokázat $M \models \varphi$, stačí dokázat, že v množině $M \cup \{\neg\varphi\}$ je „vnitřní spor“, tj. že $M \cup \{\neg\varphi\}$ není splnitelná množina. Množinu $M \cup \{\neg\varphi\}$ si ovšem můžeme představit jako množinu předpokladů M , ke které jsme přidali znegovaný závěr φ . To je přesně to, čím začíná každý důkaz sporem: předpokládáme, že závěr neplatí (a snažíme se odvodit spor).

Věta 2.3.10 (ve spolupráci s CNF) je základem *resolučních algoritmů*, viz například skriptum

☞ M. Demlová a B. Pondělíček, *Matematická logika*, skriptum FEL ČVUT, Praha, 1997

a kapitola 4 tohoto textu.

2.4 Cvičení

2.4.1 Cvičení Dokažte, že pro libovolné formule α a β platí:

$$\text{tt} \models \alpha \vee \neg\alpha \quad \alpha \wedge \beta \models \neg(\neg\alpha \vee \neg\beta) \quad \alpha \Rightarrow \beta \models \neg\alpha \vee \beta \quad \alpha \Leftrightarrow \beta \models \neg(\neg(\neg\alpha \vee \beta) \vee \neg(\neg\beta \vee \alpha))$$

Tomuto výsledku se říká *adekvátnost množiny spojek* $\{\neg, \vee\}$: každou formuli výrokové logiky můžeme nahradit synonymem, které obsahuje pouze spojky \neg a \vee .

Dokažte, že i množina spojek $\{\neg, \wedge\}$ je adekvátní.

2.4.2 Cvičení Definujte *syntaktickou zkratku* $\alpha \text{ nand } \beta = \neg(\alpha \wedge \beta)$. Ukažte, že tato nová spojka je adekvátní, tj., že množina $\{\text{nand}\}$ je adekvátní množina spojek.

2.4.3 Cvičení Podrobně projděte postup pro popis ireducibilních CNF pro formuli obsahující pouze jednu atomickou formuli z poznámky 2.2.10.

2.4.4 Cvičení Více příkladů na toto téma lze nalézt ve cvičeních ke kapitole 8 skriptu

☞ M. Demlová a B. Pondělíček, *Matematická logika*, FEL ČVUT, Praha 1997

Pro následující formule nalezněte úplnou DNF, úplnou CNF, ireducibilní DNF a ireducibilní CNF:

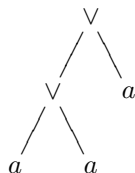
1. $(a \Rightarrow b) \Rightarrow (\neg b \Rightarrow \neg a)$.
2. $(a \text{ xor } b) \text{ xor } c$.
3. $a \Rightarrow (b \Leftrightarrow c)$.
4. $(b \vee d) \Leftrightarrow (\neg a \Leftrightarrow c)$.

kde xor je exklusivní nebo.

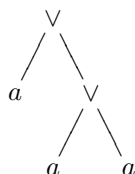
2.4.5 Cvičení Toto cvičení vyžaduje znalost syntaktických stromů formulí výrokové logiky. Ukažte, že DNF formule lze ekvivalentně definovat v řeči syntaktických stromů formule. Postupujte následovně:

1. Připomeňte si možnost jednoznačného přepisu každé formule na její syntaktický strom.

Pozor: víme, že například zápis formule $a \vee a \vee a$ je konvencí za některý z následujících dvou méně přehledných zápisů: $((a \vee a) \vee a)$ nebo $(a \vee (a \vee a))$. Výhodou těchto méně přehledných zápisů je však jednoznačnost přepisu na syntaktický strom. Syntaktický strom formule $((a \vee a) \vee a)$ je:



zatímco formule $(a \vee (a \vee a))$ má *jiny* syntaktický strom:



2. De Morganova pravidla a sémantické distributivní zákony interpretujte jako „posouvání“ logických spojek po syntaktickém stromu.

Například sémantickou ekvivalenci $\neg(a \wedge b) \models (\neg a \vee \neg b)$ (což je jedno z De Morganových pravidel) lze interpretovat jako „posouvání“ logické spojky \neg po syntaktickém stromu směrem dolů. Syntaktický strom formule $\neg(a \wedge b)$ je totiž strom:

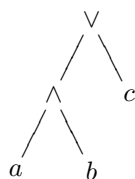


a syntaktický strom formule $(\neg a \vee \neg b)$ je:

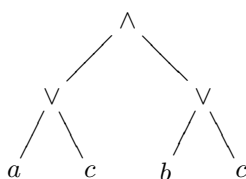


Podobně sémantickou ekvivalenci $(a \wedge b) \vee c \models (a \vee c) \wedge (b \vee c)$ (což je jeden ze sémantických distributivních zákonů) lze interpretovat jako „posouvání“ logické spojky \vee po syntaktickém stromu směrem dolů.

Syntaktický strom formule $(a \wedge b) \vee c$ je totiž strom:



a syntaktický strom formule $(a \vee c) \wedge (b \vee c)$ je:

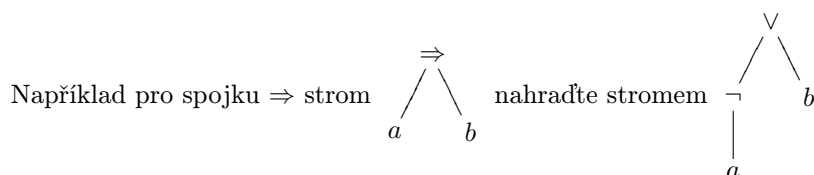


3. Definujte DNF formule jako formuli, jejíž syntaktický strom má jistý tvar.

4. Promyslete proceduru hledání DNF, která se opírá pouze o tvar syntaktických stromů.

Návod: definujte sadu úprav syntaktických stromů:

(a) Odstraňování nepohodlných logických spojek se děje nahrazením stromů, které obsahují nepohodlné spojky, stromy sémanticky ekvivalentních formulí.



(b) Posouvání spojek \neg a \wedge co nejnižší po syntaktickém stromu pomocí De Morganových pravidel a distributivních zákonů.

2.4.6 Cvičení V tomto cvičení vysvětlíme tvorbu Karnaughovy mapy pro formuli, která obsahuje pět nebo šest různých atomických formulí. Pro podrobnosti odkazujeme například na knihu

☞ G. E. Hoernes a M. F. Heilweil, *Úvod do Booleovy algebry a navrhování logických obvodů*, SNTL, Praha 1969

V případě, kdy formule φ obsahuje pět atomických formulí a_1, \dots, a_5 , vypadá Karnaughova mapa takto:

a_5	0				1					
	$a_1 a_2 \backslash a_3 a_4$	00	01	11	10	$a_1 a_2 \backslash a_3 a_4$	00	01	11	10
00										
01										
11										
10										

Povšimněme si, že tato mapa je vlastně „atlasem“ dvou Karnaughových map pro formule obsahující čtyři atomické formule. Podobně situace vypadá v případě, kdy formule φ obsahuje šest atomických formulí a_1, \dots, a_6 :

$a_5 \backslash a_6$	0					1				
	$a_1 a_2 \backslash a_3 a_4$	00	01	11	10	$a_1 a_2 \backslash a_3 a_4$	00	01	11	10
0	00					00				
	01					01				
	11					11				
	10					10				
1	00					00				
	01					01				
	11					11				
	10					10				

V tomto případě jde o „atlas“ čtyř Karnaughových map pro formule se čtyřmi atomickými formullemi.

Zformulujte způsob, kterým je v těchto případech nutné pokrývat Karnaughovy mapy, mají-li výsledné formy obsahovat minimální počet znaků.

Poznamenejme ještě, že pro hledání ireducibilních forem formulí, které obsahují velký počet atomických formulí se používají jiné metody, než je metoda Karnaughových map. O těchto metodách se lze dozvědět například v kapitole 6 knihy

☞ G. Birkhoff a T. O. Bartee, *Aplikovaná algebra*, Alfa, Bratislava 1981

2.4.7 Cvičení Ukažte, že ve výrokové logice platí $\alpha \models \beta$ právě tehdy, když $\alpha \models \beta$ a současně $\beta \models \alpha$.

Revize kapitoly

Dozvěděli jsme se:

- ✓ Výroková logika je *formální* jazyk. Má přesně definovanou *syntaxi* (tj. pravidla jak se věci píší) a *sémantiku* (tj. pravidla co věci znamenají). Syntaxe přitom *řídí* sémantiku, tj. na pochopení významu syntakticky složitých řetězců musíme nejprve porozumět významu syntakticky jednoduchých řetězců.
- ✓ Sémantické otázky ve výrokové logice můžeme vždy zodpovědět prohlížením *pravdivostních tabulek*.

Pro přípravu na zkoušku zkuste zodpovědět následující otázky:

- ✓ Popište syntaxi a sémantiku výrokové logiky.
- ✓ Dejte definici sémantického důsledku ve výrokové logice a vysvětlete algoritmus, kterým můžeme úlohu o sémantickém důsledku (s konečnou množinou předpokladů) vyřešit.

Doplňující literatura

Do větší hloubky je výroková a predikátová logika probrána ve skriptu

☞ M. Demlová a B. Pondělíček, *Matematická logika*, FEL ČVUT, Praha 1997

které jsme několikrát zmínili. Toto skriptum doporučujeme i jako zdroj dalších příkladů.

V této kapitole jsme vůbec nezmínili *syntaktický důsledek*, tj. syntaktickou analogii úsudku. Syntaktický důsledek formalizuje pojem *důkazu* a ve výrokové nebo predikátové logice je realizován *systémem přirozené dedukce* nebo *konstrukcí úspěšných tabel*. Tyto pojmy jsou velmi dobře vysvětleny například v knize

☞ A. Nerode a R. A. Shore, *Logic for Applications*, Springer, New York, 1997

Se syntaktickým důsledkem jsou úzce spjaty pojmy *úplnosti* a *neúplnosti* logik. O těch se je možno dočíst ve vynikajících knihách

☞ D. R. Hofstadter, *Gödel, Escher, Bach: An Eternal Golden Braid*, Basic Books, New York, 1979

☞ R. Smullyan, *Gödel's Incompleteness Theorem*, Oxford University Press, 1992

O životě Kurta Gödela (a samozřejmě o jeho nejznámějším výsledku, *věť o neúplnosti*) pojednává kniha

☞ R. Goldsteinová, *Neúplnost — Důkaz a paradox Kurta Gödela*, Argo + Dokořán, Praha, 2006

O použití formálních logik v programování (*logika Hoarových trojic*) se lze dočíst například ve skriptu

☞ K. Richta a J. Velebil, *Sémantika programovacích jazyků*, Karolinum, Praha 1997

a o zobecnění na logiku komunikujících procesů, takzvané *Hennesyho-Milnerově modální logice*, například v textu

☞ J. Velebil, *Logika programů*, <http://math.feld.cvut.cz/velebil/>

Kapitola 3

Predikátová logika

1. Městská rada rozhodla, že dá postavit nové vězení.
2. Městská rada rozhodla, že nové vězení bude postaveno z materiálu ze starého vězení.
3. Městská rada rozhodla, že dokud nebude nové vězení postaveno, bude se používat vězení staré.

Rozhodnutí městské rady, Canton, Mississippi, USA

V této kapitole zavedeme další formální jazyk: predikátovou logiku. Jde opět o jazyk formální. Popíšeme tedy syntaxi a sémantiku predikátové logiky a zmíníme otázku sémantického důsledku a sémantické ekvivalence. Zdůrazníme, že v predikátové logice není žádná analogie pravdivostních tabulek.

3.1 Syntaxe a sémantika predikátové logiky

Syntaxe a sémantika výrokové logiky, které jsme zavedli v předchozích odstavcích, umožňují formální popis základních logických konstrukcí, jako je například otázka správnosti úsudku.

Nevýhody výrokové logiky spočívají v její malé vyjadřovací bohatosti. Jednou nevýhodou je, že jazyk výrokové logiky nepřipouští práci s proměnnými. Některá důležitá tvrzení, jako například větu

Ne každý, kdo hraje na housle, je Sherlock Holmes.

tak nelze ve výrokové logice adekvátně popsat. V dané větě se totiž vyskytují *dvě* syntaktické kategorie: mluví se v ní o lidech¹ a jejich vlastnostech (hrát na housle, být Sherlockem Holmesem). Viz příklad 5.1.2.

Zobecnění jazyka výrokové logiky, ve kterém lze výše uvedenou větu popsat formálně a adekvátním způsobem, se nazývá *predikátová logika*.

Jazyk predikátové logiky vytváří dvě syntaktické kategorie:

1. *Termy*: intuitivně je dobré si termy představit jako objekty, o jejichž vlastnostech bude naše logika vypovídat. Termem může být buď *proměnná* nebo je term zkonstruován z jiných termů použitím *funkčních symbolů*.
2. *Formule*: intuitivně jsou to vlastnosti vytvořených objektů. Formule budou vytvářeny dvojím způsobem:
 - (a) *Atomické formule*: to jsou ty formule, které jsou z logického hlediska dále nedělitelné. Atomická formule může být buď vytvořena jako *rovnost termů*² (intuitivně: tvrdíme, že dva objekty jsou totožné) nebo jako aplikace *predikátu* na termy (intuitivně: predikát je nějaká atomická vlastnost objektů).

¹Připustíme-li, že Sherlock Holmes, literární fikce sira Arthura Conana Doylea (1859–1930), je reálně existující člověk. Předlohou Sherlocka Holmese byl Doyleův universitní učitel a průkopník forenzní patologie Joseph Bell (1837–1911), který ve svých přednáškách zdůrazňoval použití dedukce pro určení diagnózy.

²Pro rovnost v predikátové logice budeme používat standardní značku $=$. Značku $=$ pro rovnost zavedl v roce 1557 ve své knize *The Whetstone of Witte* velšán Robert Recorde (1510–1558), protože (jak píše) *žádné dvě věci si nemohou být rovnější než dvojice čar stejné délky*.

(b) *Složité formule*: to jsou formule vytvořené z jiných formulí pomocí známých spojek z výrokové logiky

- | | |
|--|---|
| (a) <i>True</i> : \mathbf{tt} arity 0. | (d) <i>Disjunkce</i> : \vee arity 2. |
| (b) <i>Negace</i> : \neg arity 1. | (e) <i>Implikace</i> : \Rightarrow arity 2. |
| (c) <i>Konjunkce</i> : \wedge arity 2. | (f) <i>Ekvivalence</i> : \Leftrightarrow arity 2. |

ke kterým přidáme ještě dvě speciální značky

\forall (obecný kvantifikátor)

a

\exists (existenční kvantifikátor)

Formule, které jsou vytvořeny *kvantifikováním* jiných formulí, musí mít ovšem velmi speciální tvar: kvantifikovat se mohou pouze standardní proměnné.

Shrnuto: jazyk predikátové logiky je svou podstatou *vícesortový*.³ Obsahuje opět *standardní část*, která sestává ze standardních značek

$$= \mathbf{tt} \quad \neg \quad \wedge \quad \vee \quad \Rightarrow \quad \Leftrightarrow \quad \forall \quad \exists$$

a část *volitelnou*, kterou volíme *vyjadřovací sílu* našeho jazyka. Volitelnou část si můžeme představit jako *deklaraci* proměnných, funkcí a atomických vlastností.

3.1.1 Jazyk predikátové logiky Jazyk \mathcal{L} predikátové logiky je zadán následujícími třemi množinami:

1. Neprázdnou množinou Var *standardních proměnných*.
2. Množinou Pred *predikátů*. U každého predikátu musí být navíc specifikována jeho *arita*, což je přirozené číslo (smí být i nula).⁴ Podotkneme, že množina Pred *smí být prázdná*.
3. Množinou Func *funkčních symbolů*. U každého funkčního symbolu musí být navíc specifikována jeho *arita*, což je přirozené číslo (smí být i nula). Podotkneme, že množina Func *smí být prázdná*.

Zdůrazněme znovu, že definici jazyka \mathcal{L} je vhodné si představit jako soupis *deklarací*.

3.1.2 Příklad Deklarujeme jazyk \mathcal{L} následovně:

1. Množina Var standardních proměnných jazyka \mathcal{L} bude mít tři prvky: x , y a a .
2. Množina Func funkčních symbolů jazyka \mathcal{L} bude mít dva prvky: f arity 0 a H arity 2.
3. Množina Pred predikátů jazyka \mathcal{L} bude mít jeden prvek V arity 1.

Zjevně jsme splnili požadavky na definici jazyka predikátové logiky. Náš jazyk tak obsahuje tři značky x , y , a pro *standardní proměnné*. Dále jazyk \mathcal{L} obsahuje dvě značky f , H pro tvorbu nových objektů ze starých (značka f potřebuje na tvorbu nových objektů 0 argumentů a je tedy dobré si ji představit jako *konstantní objekt*, značka H potřebuje pro tvorbu nového objektu dva objekty, protože má aritu 2). Konečně jsme deklarovali značku V jako *atomickou vlastnost* objektu, protože V má aritu 1.

Jakmile je zadán jazyk \mathcal{L} , jsou vytvořeny množiny $\text{Terms}(\mathcal{L})$ *termů jazyka \mathcal{L}* a $\text{Formulae}(\mathcal{L})$ *formulí jazyka \mathcal{L}* . Tato tvorba je automatická a my nemůžeme formu termů a formulí už nijak ovlivnit, protože termy a formule se tvoří z našich deklarací pomocí standardních značek.

3.1.3 Syntaxe termů a formulí predikátové logiky

Množina termů $\text{Terms}(\mathcal{L})$ jazyka \mathcal{L} je definována takto:

1. Každá standardní proměnná je term jazyka \mathcal{L} .

³Připomíná tak programovací jazyky, viz například text J. Velebil, *Diskrétní matematika*.

⁴Pro praktický význam predikátů arity 0 viz poznámku 3.1.25.

2. Jakmile $t_1, \dots, t_n, n \geq 0$ jsou termy jazyka \mathcal{L} a jakmile je f funkční symbol arity n , je řetězec $f(t_1, \dots, t_n)$ term jazyka \mathcal{L} .
3. Žádným jiným způsobem, než konečným použitím předchozích dvou pravidel, term jazyka \mathcal{L} *nevzniká*.

Množina formulí $\text{Formulae}(\mathcal{L})$ jazyka \mathcal{L} je definována takto:

1. Jakmile t_1, t_2 jsou termy jazyka \mathcal{L} , je řetězec $(t_1 = t_2)$ formule jazyka \mathcal{L} (takzvaná *atomická formule* jazyka \mathcal{L}).
2. Jakmile $t_1, \dots, t_n, n \geq 0$ jsou termy jazyka \mathcal{L} a jakmile je P predikát arity n , je řetězec $P(t_1, \dots, t_n)$ formule jazyka \mathcal{L} (takzvaná *atomická formule* jazyka \mathcal{L}).
3. Jsou-li φ a ψ formule jazyka \mathcal{L} a je-li x standardní proměnná jazyka \mathcal{L} , pak jsou i řetězce

$$\mathbf{tt} \quad (\neg\varphi) \quad (\varphi \wedge \psi) \quad (\varphi \vee \psi) \quad (\varphi \Rightarrow \psi) \quad (\varphi \Leftrightarrow \psi) \quad (\forall x.\varphi) \quad (\exists x.\varphi)$$

formulemi jazyka \mathcal{L} .

4. Žádným jiným způsobem, než konečným použitím předchozích tří pravidel, formule jazyka \mathcal{L} *nevzniká*.

3.1.4 Příklad Pro jazyk \mathcal{L} z příkladu 3.1.2 dostáváme:

1. Jako termy například řetězce $x, y, a, f, H(x, y), H(f, x), H(H(x, a), a)$ atd.
2. Jako formule například řetězce $(x = a), H(y, f) = H(H(x, x), x), V(x), (\forall a.(V(x) \Rightarrow (f = H(a, a))))$ atd.

Povšimněme si, že množiny $\text{Terms}(\mathcal{L})$ a $\text{Formulae}(\mathcal{L})$ jsou *nekonečné*.

3.1.5 Poznámka Syntaxi termů a formulí lze samozřejmě opět přehledněji popsat Backusovou-Naurovou formou:

$$\begin{aligned} t & ::= x \mid f(t_1, \dots, t_n) \\ \varphi & ::= \mathbf{tt} \mid (t_1 = t_2) \mid P(t_1, \dots, t_n) \mid (\neg\varphi) \mid (\varphi \wedge \psi) \mid (\varphi \vee \psi) \mid (\varphi \Rightarrow \psi) \mid (\varphi \Leftrightarrow \psi) \mid (\forall x.\varphi) \mid (\exists x.\varphi) \end{aligned}$$

kde x je standardní proměnná, f funkční symbol arity n a P predikátový symbol arity n .



3.1.6 Poznámka I v predikátové logice zavedeme následující *syntaktické konvence* (srovnejte s poznámkou 2.1.3):

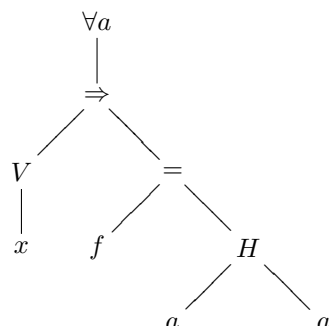
1. Nebudeme psát nejvíce vnější závorky.
2. Budeme předpokládat, že spojka \neg váže nejsilněji.
3. Zavedeme *false*: formule \mathbf{ff} je syntaktická zkratka za formuli $\neg\mathbf{tt}$.

Nebude-li uvedeno jinak, budeme v dalším vždy předpokládat, že syntaxe predikátové logiky je relaxována podle těchto pravidel.

3.1.7 Příklad Syntaktickou analýzu v predikátové logice opět provádíme tvorbou *syntaktických stromů*. V jazyce \mathcal{L} z příkladu 3.1.2 je řetězec

$$\forall a.(V(x) \Rightarrow (f = H(a, a)))$$

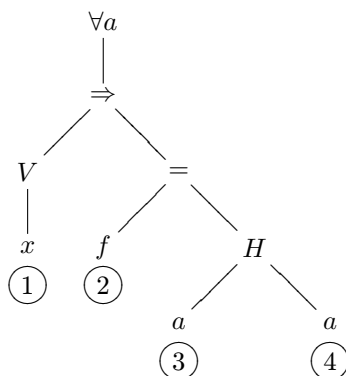
formulí, protože umíme úspěšně sestavit syntaktický strom tohoto řetězce:



Tvorba syntaktického stromu je opět dána rekursivním algoritmem, který v každém kroku ověřuje, zda je značka v jazyce deklarována nebo zda jde o značku standardní. Viz cvičení 3.3.1.

Než budeme schopni popsat sémantiku formulí jazyka predikátové logiky, musíme zavést pojmy *volného* a *vázaného* výskytu standardní proměnné ve formuli.

3.1.8 Příklad V syntaktickém stromu formule z příkladu 3.1.7 označíme jeho jednotlivé listy zleva doprava čísly:



Povšimněme si, že listem syntaktického stromu může být buď standardní proměnná (listy číslo 1, 3 a 4) nebo funkční symbol arity 0 (list číslo 2) nebo predikátovým symbolem arity 0 (v tomto příkladě se to nestalo).

3.1.9 Definice Každý list syntaktického stromu formule φ , který je obsazen standardní proměnnou, nazýváme *výskytem standardní proměnné ve formuli φ* .

Výskyt standardní proměnné x ve formuli φ nazýváme *vázaným*, pokud při cestě od tohoto listu ke kořeni syntaktického stromu narazíme na vrchol označovaný buď $\forall x$ nebo $\exists x$. V opačném případě nazveme tento výskyt *volným*.

Kvantifikátor $\forall x$ nebo $\exists x$ *váže* všechny výskyty standardní proměnné x , které jsou v syntaktickém stromu pod tímto kvantifikátorem.

3.1.10 Příklad Výskyt 1 standardní proměnné ve formuli z příkladu 3.1.8 je volný, protože cestou ke kořeni syntaktického stromu od listu ke kořeni nenarazíme ani na vrchol označovaný $\forall x$ ani na vrchol označovaný $\exists x$.

Ve stejném příkladu jsou oba výskyty 3 a 4 standardní proměnné a vázané, protože cestou od obou listů ke kořeni syntaktického stromu narazíme na vrchol označovaný $\forall a$.

Kvantifikátor $\forall a$ váže oba výskyty 3 a 4 proměnné a .



3.1.11 Poznámka Vázané výskyty proměnných známe velmi dobře z matematické analýzy a z programování.

1. Ve výrazu $\left(\int_0^2 (t^2 - x) dt + x \right)$ je proměnná t *vázaná* (značkou pro určitý integrál $\int_0^2 dt$) a proměnná x má oba výskyty *volné*.

Z analýzy dobře víme, že vázanou proměnnou smíme *přejmenovat*. Musíme ovšem dbát na to, aby se po přejmenování nestala volná proměnná vázanou. Takže přejmenování

$$\left(\int_0^2 (t^2 - x) dt + x \right) [t := x] = \left(\int_0^2 (x^2 - x) dx + x \right)$$

není legální substituce, protože se první (původně volný) výskyt proměnné x stává vázaným výskytem. Přejmenování

$$\left(\int_0^2 (t^2 - x) dt + x \right) [t := z] = \left(\int_0^2 (z^2 - x) dz + x \right)$$

kde z je nová proměnná, *je* legální substituce.

2. V části kódu `for i:=1 to n do z:=z*i ; i:=i+1 endfor` jsou oba výskyty proměnné i *vázané* značkou `for endfor`, výskyt proměnné z je *volný*. Pro přejmenování vázaných proměnných v programování platí stejné podmínky jako pro proměnné v integrálním počtu: po přejmenování se žádná, původně volná proměnná *nesmí* stát vázanou. Přejmenování výše uvedené části kódu na

```
for z:=1 to n do z:=z*z ; z:=z+1 endfor
```

tedy *není* legální.

Problematické mohou být situace, kdy má v nějaké formuli *stejná* standardní proměnná některé výskyty volné a některé vázané. To definice syntaxe formulí nezakazuje, například řetězec

$$V(x) \vee (\exists x.x = x)$$

je formulí v jazyce \mathcal{L} z příkladu 3.1.2 a standardní proměnná x v ní má jak volný, tak vázaný výskyt.

Zavedeme nyní důležitou konvenci, která nám dovolí se smíšeným výskytům standardních proměnných vyhnout.



3.1.12 Poznámka Zavedeme další syntaktickou konvenci, které se říká *α -konverse*:

Dvě formule, které se liší pouze legálním přejmenováním vázaných proměnných, budeme považovat za totožné.

Připomeňme, že *legálním přejmenováním* myslíme přejmenování, které splňuje následující dvě podmínky:

1. Přejmenováváme proměnné, které daný kvantifikátor váže (viz definice 3.1.9).
2. Při přejmenování se žádný, původně volný, výskyt proměnné nesmí stát vázaným.

Díky α -konverzi můžeme tedy bez újmy na obecnosti předpokládat, že v každé formuli má každá standardní proměnná všechny výskyty buď pouze volné nebo pouze vázané.

3.1.13 Definice Řekneme, že formule φ predikátové logiky je *sentence*, pokud je každý výskyt každé proměnné ve formuli φ vázaný.

3.1.14 Příklad Formule $\forall a.(V(x) \Rightarrow (f = H(a, a)))$ z příkladu 3.1.4 není sentencí, protože v ní má standardní proměnná x volný výskyt.

Formule $\exists x.(V(x) \vee (x = f))$ stejného jazyka sentencí je, protože oba výskyty standardní proměnné x jsou vázané.

Sentenci predikátové logiky je dobré si představit jako formuli, která „schovává proměnné před vnějším pozorovatelem“. To bude důležité při definici sémantiky: pravdivostní hodnota sentence nebude záviset na aktuální „hodnotě“ standardních proměnných.

Sémantika formulí predikátové logiky se opět bude řídit doktrínou, že syntaxe řídí sémantiku (viz poznámku 2.1.5). Budeme proto nuceni podat sémantiku i formulí, které obsahují volné standardní proměnné. Protože taková formule standardní proměnné „neschovává“, bude její pravdivost či nepravdivost záviset na *aktuálním kontextu* standardních proměnných. Sémantika formulí se tedy bude sestávat ze dvou částí:

1. Z *interpretace* predikátů a funkčních symbolů. Tato interpretace nám popíše „skutečný“ význam atomických vlastností (tj. predikátů) a funkčních symbolů.
2. Z *kontextu standardních proměnných*, tj. z popisu „aktuálních hodnot“ všech standardních proměnných.

Zformulujeme nyní sémantiku formulí predikátové logiky přesně:

3.1.15 Interpretace predikátů a funkčních symbolů Ať \mathcal{L} je jazyk predikátové logiky s množinou predikátů Pred a množinou funkčních symbolů Func . *Interpretací predikátů a funkčních symbolů* rozumíme následující data:

1. Neprázdna množina U zvaná *universum interpretace*.
Intuitivně: universum je „svět“, jehož prvky popisují termy a o vlastnostech těchto prvků mluví formule. Znovu zdůrazníme, že *universum musí být neprázdna množina*.
2. Přiřazení $\llbracket - \rrbracket$, které
 - (a) Každému predikátu P arity n přiřazuje množinu $\llbracket P \rrbracket$ uspořádaných n -tic prvků universa U .
Intuitivně: pro $n \geq 1$ je $\llbracket P \rrbracket$ seznam n -tic objektů, které mají „vlastnost“ P . Pro $n = 0$ je $\llbracket P \rrbracket$ buď prázdna množina nebo jednoprvková množina.
 - (b) Každému funkčnímu symbolu f arity n přiřazuje funkci $\llbracket f \rrbracket : U^n \rightarrow U$.
Intuitivně: pro $n \geq 1$ je $\llbracket f \rrbracket$ předpis, který z n -tice objektů vytvoří další objekt. Pro $n = 0$ je $\llbracket f \rrbracket$ pevný objekt (konstanta), tj. pevný prvek universa U .

V dalším budeme interpretaci predikátů a funkčních symbolů značit jako uspořádanou dvojici $\langle U, \llbracket - \rrbracket \rangle$.

3.1.16 Příklad Jazyk \mathcal{L} z příkladu 3.1.2 interpretujeme následovně:

1. Jako universum interpretace zvolíme množinu \mathbb{N} přirozených čísel. Tím říkáme, že náš jazyk bude popisovat pouze přirozená čísla a jejich vlastnosti.
2. Přiřazení $\llbracket - \rrbracket$ definujeme následovně:
 - (a) $\llbracket V \rrbracket$ je podmnožina sudých přirozených čísel.
 - (b) $\llbracket f \rrbracket$ je číslo 11 (funkční symbol f má totiž aritu 0, jsme tedy povinni jej interpretovat jako konstantu).
 $\llbracket H \rrbracket$ je funkce součtu dvou přirozených čísel.

Samozřejmě, tato interpretace je jen jednou z mnoha možných interpretací. Uvědomme si ale, že *nemůžeme* interpretovat funkční symbol H například jako dělení: dělení totiž *není* funkce z \mathbb{N}^2 do \mathbb{N} .

3.1.17 Kontext standardních proměnných Ať $\langle U, \llbracket - \rrbracket \rangle$ je interpretace predikátů a funkčních symbolů. *Kontext standardních proměnných* je funkce

$$\rho : \text{Var} \rightarrow U$$

Jestliže ρ je kontext standardních proměnných, x je standardní proměnná a d je prvek universa U , pak symbolem

$$\rho[x := d]$$

označíme kontext standardních proměnných, který má stejné hodnoty jako kontext ρ , *kromě* hodnoty d v x . Kontextu $\rho[x := d]$ říkáme *update kontextu ρ o hodnotu d v x* .

3.1.18 Příklad V interpretaci $\langle U, \llbracket - \rrbracket \rangle$ můžeme jako kontext ρ zvolit například funkci, která má hodnoty $\rho(x) = 231$, $\rho(y) = 0$, $\rho(a) = 12$.

Update kontextu ρ o hodnotu 677 v y je nový kontext ρ' s hodnotami $\rho'(x) = 231$, $\rho'(y) = 677$, $\rho'(a) = 12$.

3.1.19 Sémantika termů predikátové logiky

V interpretaci $\langle U, \llbracket - \rrbracket \rangle$ a kontextu ρ mají termy jazyka \mathcal{L} následující sémantiku:

1. $\llbracket x \rrbracket_\rho = \rho(x)$ pro každou standardní proměnnou.
Neboli: význam standardní proměnné x v kontextu ρ je hodnota $\rho(x)$.
2. Term $f(t_1, \dots, t_n)$ má sémantiku

$$\llbracket f(t_1, \dots, t_n) \rrbracket_\rho = \llbracket f \rrbracket(\llbracket t_1 \rrbracket_\rho, \dots, \llbracket t_n \rrbracket_\rho)$$

Neboli: význam termu $f(t_1, \dots, t_n)$ v kontextu ρ zjistíme tak, že do funkce $\llbracket f \rrbracket$ dosadíme hodnoty $\llbracket t_1 \rrbracket_\rho, \llbracket t_2 \rrbracket_\rho, \dots, \llbracket t_n \rrbracket_\rho$ jako argumenty.

3.1.20 Příklad Značení je z příkladu 3.1.18. Sémantika některých termů v kontextu ρ vypadá následovně:

1. $\llbracket x \rrbracket_\rho = \rho(x) = 231, \llbracket y \rrbracket_\rho = \rho(y) = 0, \llbracket a \rrbracket_\rho = \rho(a) = 12$.
2. $\llbracket H(x, a) \rrbracket_\rho = \llbracket H \rrbracket(\llbracket x \rrbracket_\rho, \llbracket a \rrbracket_\rho) = 231 + 12 = 243$, protože H je interpretováno jako součet.

3.1.21 Sémantika formulí predikátové logiky

V interpretaci $\langle U, \llbracket - \rrbracket \rangle$ a kontextu ρ mají formule jazyka \mathcal{L} následující sémantiku:

1. Atomická formule $t_1 = t_2$ je *pravdivá*, když platí rovnost $\llbracket t_1 \rrbracket_\rho = \llbracket t_2 \rrbracket_\rho$ v universu U .
Neboli: pravdivost atomické formule $t_1 = t_2$ zjistíme tak, že interpretujeme oba termy a hodnoty $\llbracket t_1 \rrbracket_\rho$ a $\llbracket t_2 \rrbracket_\rho$ v universu U porovnáme.
2. Atomická formule $P(t_1, \dots, t_n)$ je *pravdivá*, když platí

$$(\llbracket t_1 \rrbracket_\rho, \dots, \llbracket t_n \rrbracket_\rho) \in \llbracket P \rrbracket$$

Neboli: pro pravdivost atomické formule $P(t_1, \dots, t_n)$ zjistíme, zda uspořádaná n -tice $(\llbracket t_1 \rrbracket_\rho, \dots, \llbracket t_n \rrbracket_\rho)$ prvků universa U leží v množině $\llbracket P \rrbracket$.

3. Formule tt je *pravdivá*.
4. Formule $\neg\varphi$ je *pravdivá* právě tehdy, když formule φ je *nepravdivá*.
5. Formule $\varphi \wedge \psi$ je *pravdivá* právě tehdy, když jsou obě formule φ a ψ *pravdivé* současně.
6. Formule $\varphi \vee \psi$ je *pravdivá* právě tehdy, když je alespoň jedna z formulí φ a ψ *pravdivá*.
7. Formule $\varphi \Rightarrow \psi$ je *nepravdivá* pouze tehdy, když je formule φ *pravdivá* a současně ψ *nepravdivá*.
8. Formule $\varphi \Leftrightarrow \psi$ je *pravdivá* právě tehdy, když jsou buď obě formule φ a ψ *pravdivé* současně nebo když jsou obě formule φ a ψ *nepravdivé* současně.
9. Formule $\forall x.\varphi$ je *pravdivá*, když je formule φ *pravdivá* v *každém* kontextu $\rho[x := d]$, kde d je prvek U .
10. Formule $\exists x.\varphi$ je *pravdivá*, když je formule φ *pravdivá* v *alespoň jednom* kontextu $\rho[x := d]$, kde d je prvek U .

3.1.22 Příklad Značení je z příkladu 3.1.18. Sémantika některých formulí v kontextu ρ vypadá následovně:

1. Atomická formule $x = y$ je *nepravdivá*, protože rovnost $231 = 0$ v přirozených číslech neplatí.
2. Atomická formule $V(x)$ je *nepravdivá*, protože 231 není sudé přirozené číslo.
3. Formule $V(y) \wedge (a = a)$ je *pravdivá*, protože jsou *pravdivé* obě formule $V(y)$ a $(a = a)$ *současně*.

4. Formule $\exists a.V(a)$ je pravdivá, protože formule $V(a)$ je pravdivá (například) v kontextu $\rho[a := 150]$.
5. Formule $\forall a.V(a)$ není pravdivá, protože formule $V(a)$ není pravdivá v každém kontextu tvaru $\rho[a := d]$, kde d je přirozené číslo.



3.1.23 Poznámka Je vidět, že sémantika formulí predikátové logiky je poměrně komplikovaná záležitost. Zdůrazněme především, že v predikátové logice *neexistuje* analogie pravdivostní tabulky. Sémantické otázky tedy *není možné řešit hrubou silou*, jako tomu bylo možné ve výrokové logice.

Sémantiku predikátové logiky jsme zavedli způsobem, kterým se běžně zavádí sémantika *imperativního* programovacího jazyka, viz například skriptum

☞ K. Richta a J. Velebil, *Sémantika programovacích jazyků*, Karolinum, Praha 1997

Připomeňme (definice 3.1.13), že sentence „schovává“ standardní proměnné. Proto pravdivost nebo nepravdivost sentence nemůže záviset na kontextu standardních proměnných.



3.1.24 Definice Řekneme, že sentence φ je *pravdivá* v interpretaci $\langle U, \llbracket - \rrbracket \rangle$, když je pravdivá v libovolném kontextu ρ .



3.1.25 Poznámka Vysvětlíme, jaký je praktický význam predikátových symbolů arity 0. Zavádíme je proto, abychom mohli říci, že *predikátová logika je rozšířením výrokové logiky*.

Přesněji: ať At je množina atomických formulí výrokové logiky. Každou formuli φ výrokové logiky nad množinou At pak můžeme chápat jako formuli jazyka \mathcal{L} predikátové logiky, kde každé $a \in At$ deklaruje jako predikátový symbol arity 0.

Jak vypadá obecná interpretace $\langle U, \llbracket (-) \rrbracket \rangle$ takového jazyka predikátové logiky? Pro každé $a \in At$ musíme definovat $\llbracket a \rrbracket$ jako podmnožinu U^0 . Množina U^0 je ale jednoprvková a má tudíž přesně dvě podmnožiny: \emptyset a U^0 . Pokud $\llbracket a \rrbracket = \emptyset$, je sentence a v interpretaci $\langle U, \llbracket (-) \rrbracket \rangle$ nepravdivá, je-li $\llbracket a \rrbracket = U^0$, je sentence a v interpretaci $\langle U, \llbracket (-) \rrbracket \rangle$ pravdivá. Viz definici 3.1.21.

Při těchto deklaracích můžeme převádět i pravdivost formulí: ať u je ohodnocení formulí výrokové logiky. Interpretaci $\langle U, \llbracket (-) \rrbracket \rangle$ pak definujeme takto: $\llbracket a \rrbracket = U^0$ právě tehdy, když $u(a) = 1$, a to pro každou atomickou formuli $a \in At$.

Pak pro každou formuli φ výrokové logiky nad At platí $u(\varphi) = 1$ právě tehdy, když je φ (jako formule predikátové logiky) pravdivá v právě popsané interpretaci $\langle U, \llbracket (-) \rrbracket \rangle$ jazyka predikátové logiky.

Pozor! Opět zdůrazňujeme, že ani při výše uvedeném ztotožnění nemluvíme v predikátové logice o ohodnoceních.

Máme-li pojem pravdivosti, můžeme zavést pojem sémantického důsledku, sémantické ekvivalence a splnitelnosti formálně *stejným* způsobem, jako ve výrokové logice. Namísto „řádek pravdivostní tabulky“ ovšem musíme říkat „interpretace“. Pojmy zavedeme *pouze* pro sentence.

3.1.26 Definice Řekneme, že množina M sentencí jazyka \mathcal{L} predikátové logiky je *splnitelná*, když existuje interpretace, ve které jsou všechny sentence z množiny M pravdivé současně.

Jestliže množina M je splnitelná a obsahuje jedinou sentenci φ , říkáme, že φ je *splnitelná*.



3.1.27 Příklad Popište jazyk \mathcal{L} predikátové logiky, ve kterém je řetězec

$$P(z) \Leftrightarrow (\forall x.P(x))$$

sentencí a ukažte, že tato sentence je splnitelná.

Syntaktickou analýzou (tj. pokusem o úspěšné sestavení syntaktického stromu) zjišťujeme, že nemáme na výběr: má-li být výše uvedený řetězec sentencí, musí jazyk \mathcal{L} vypadat následovně:

1. Symbol x je standardní proměnná, tj. $\text{Var} = \{x\}$.

2. Symbol P je predikát arity 1.
3. Symbol z je funkční symbol arity 0 (tj. konstanta).

Podle definice máme nalézt alespoň jednu interpretaci $\langle U, \llbracket - \rrbracket \rangle$ jazyka \mathcal{L} , ve které je uvedená sentence pravdivá.

Nejprve si uvědomme, jak musí vypadat *libovolná* interpretace jazyka \mathcal{L} . Tím zjistíme, o čem jsou sentence jazyka \mathcal{L} schopny vypovídat:

Libovolná interpretace jazyka \mathcal{L} sestává z neprázdné množiny U , nějakého pevného prvku $\llbracket z \rrbracket \in U$ (protože z je funkční symbol arity 0) a nějaké podmnožiny $\llbracket P \rrbracket \subseteq U$ (protože P je predikát arity 1).

Nyní musíme zvolit *konkrétní* interpretaci $\langle U, \llbracket - \rrbracket \rangle$ tak, aby v ní byla sentence $P(z) \Leftrightarrow (\forall x.P(x))$ pravdivá. Podle definice 3.1.21 toho můžeme dosáhnout jedním ze dvou způsobů:

1. Obě formule $P(z)$ a $\forall x.P(x)$ jsou v $\langle U, \llbracket - \rrbracket \rangle$ pravdivé *současně*.
2. Obě formule $P(z)$ a $\forall x.P(x)$ jsou v $\langle U, \llbracket - \rrbracket \rangle$ nepravdivé *současně*.

Pokusíme se o splnění první podmínky (pokuste se najít *jinou* interpretaci, která splňuje druhou podmínku).

Má-li $P(z)$ být v $\langle U, \llbracket - \rrbracket \rangle$ pravdivá, musí platit $\llbracket z \rrbracket \in \llbracket P \rrbracket$ (neboli: z má vlastnost P). Má-li $\forall x.P(x)$ být v $\langle U, \llbracket - \rrbracket \rangle$ pravdivá, musí platit $\llbracket P \rrbracket = U$ (neboli: všechny prvky U mají vlastnost P).

Takových interpretací $\langle U, \llbracket - \rrbracket \rangle$ umíme jistě najít celou řadu: zvolíme co *nejjednodušší*. Naši interpretaci zapíšeme do přehledné tabulky

jazyk \mathcal{L}	interpretace
st. proměnné: x	$U = \{u\}$
predikáty: P arity 1	$\llbracket P \rrbracket = U$
funkční symboly: z arity 0	$\llbracket z \rrbracket = u$

ze které je okamžitě vidět, že sentence $P(z) \Leftrightarrow (\forall x.P(x))$ je v interpretaci $\langle U, \llbracket - \rrbracket \rangle$ pravdivá.

Ukázali jsme, že $P(z) \Leftrightarrow (\forall x.P(x))$ je splnitelná sentence.

3.1.28 Definice Řekneme, že sentence φ a ψ jazyka \mathcal{L} predikátové logiky jsou *sémanticky ekvivalentní* (značení $\varphi \models \psi$), pokud mají stejnou pravdivostní hodnotu v každé interpretaci.

3.1.29 Definice Řekneme, že sentence φ jazyka \mathcal{L} predikátové logiky je *tautologie*, pokud platí $\varphi \models \mathbf{tt}$ a sentence φ je *kontradikce*, pokud platí $\varphi \models \mathbf{ff}$.



3.1.30 Příklad Ukažte, že sentence $P(z) \Leftrightarrow (\forall x.P(x))$ z příkladu 3.1.27 *není tautologie*.

Jazyk \mathcal{L} pochopitelně zůstává stejný jako v příkladu 3.1.27. Máme ukázat, že $P(z) \Leftrightarrow (\forall x.P(x)) \not\models \mathbf{tt}$ *neplatí*. Protože \mathbf{tt} je pravdivá v *každé* interpretaci (viz definici 3.1.21), znamená to, že musíme najít interpretaci $\langle U, \llbracket - \rrbracket \rangle$ jazyka \mathcal{L} , ve které sentence $P(z) \Leftrightarrow (\forall x.P(x))$ *není pravdivá*. Toho je možné dosáhnout jedním ze dvou způsobů (viz definici 3.1.21):

1. Formule $P(z)$ je v $\langle U, \llbracket - \rrbracket \rangle$ nepravdivá a *současně* formule $\forall x.P(x)$ je v $\langle U, \llbracket - \rrbracket \rangle$ pravdivá.
2. Formule $P(z)$ je v $\langle U, \llbracket - \rrbracket \rangle$ pravdivá a *současně* formule $\forall x.P(x)$ je v $\langle U, \llbracket - \rrbracket \rangle$ nepravdivá.

Pokusíme se o splnění první podmínky. To znamená, že hledáme interpretaci, pro kterou platí $\llbracket z \rrbracket \notin \llbracket P \rrbracket$ (protože z nemá mít vlastnost P) a současně $\llbracket P \rrbracket = U$ (protože všechny prvky U mají mít vlastnost P). To ale není možné splnit!

Musíme se tedy zaměřit na splnění druhé podmínky: hledáme interpretaci, ve které platí $\llbracket z \rrbracket \in \llbracket P \rrbracket$ (protože z má mít vlastnost P) a současně $\llbracket P \rrbracket \neq U$ (protože ne všechny prvky U mají mít vlastnost P). Takových interpretací je jistě celá řada, popíšeme tu nejjednodušší:

jazyk \mathcal{L}	interpretace
st. proměnné: x	$U = \{u, v\}$
predikáty: P arity 1	$\llbracket P \rrbracket = \{u\}$
funkční symboly: z arity 0	$\llbracket z \rrbracket = u$

V této interpretaci je sentence $P(z) \Leftrightarrow (\forall x.P(x))$ nepravdivá.

Ukázali jsme, že sentence $P(z) \Leftrightarrow (\forall x.P(x))$ není tautologie.



3.1.31 Poznámka Při konstrukci interpretací jazyka predikátové logiky příkladu 3.1.30 se často objevují interpretace typu U =množina všech lidí, $\llbracket P \rrbracket$ =mít vlasy, $\llbracket z \rrbracket$ =Jiří Velebil. Taková interpretace pak „potvrzuje“, že sentence $P(z) \Leftrightarrow (\forall x.P(x))$ není tautologie, protože existuje alespoň jeden člověk, který vlasy nemá.

Taková interpretace je velmi problematická, přinejmenším z následujících důvodů:

1. Není moc jasné, co to je množina všech lidí.

Znamená to množinu všech lidí, žijících 18. července 2007 ve 13:00 středoevropského času? Nebo množinu všech lidí, kteří kdy na zemi žili, žijí a žít budou?

2. Není moc jasné, co to znamená mít vlasy.

Znamená to mít na hlavě alespoň jeden vlas? Nebo člověka s přesně třemi vlasy na hlavě již považujeme za holohlavého?

3. Kterého Jiřího Velebila máme na mysli? Jen v pražském telefonním seznamu jsou uvedeni tři.

Problematickostí této interpretace souvisí s nejednoznačností používání slov v přirozeném jazyce. Více o tom řekneme v kapitole 5.

Výše uvedenou interpretaci však můžeme snadno upravit. Vytvoříme svět, ve kterém bude jeden člověk s vlasy a alespoň jeden člověk, který vlasy nemá. Taková interpretace může vypadat takto:

jazyk \mathcal{L}	interpretace
st. proměnné: x	$U = \{u, v\}$ to jest: v našem světě jsou jen dva „lidé“: u a v
predikáty: P arity 1	$\llbracket P \rrbracket = \{u\}$ to jest: člověk u „má vlasy“
funkční symboly: z arity 0	$\llbracket z \rrbracket = u$ to jest: u je „Jiří Velebil“ v našem světě

V této interpretaci sentence $P(z) \Leftrightarrow (\forall x.P(x))$ pravdivá není: levá strana $P(z)$ ekvivalence se přeloží jako pravdivá věta („člověk“ u v našem světě „má vlasy“) a pravá strana $\forall x.P(x)$ ekvivalence se přeloží jako nepravdivá věta (ne všichni „lidé“ v našem světě „mají vlasy“: takovým je „člověk“ v).

Protože jsme našli interpretaci, ve které je sentence $P(z) \Leftrightarrow (\forall x.P(x))$ nepravdivá, není sentence $P(z) \Leftrightarrow (\forall x.P(x))$ tautologie.

Zjišťujeme, že jsme napsali přesně interpretaci z příkladu 3.1.30. Navíc je tato interpretace naprosto průzračná a jednoznačná.

Pokud se nemůžete vzdát používání interpretací na „množině lidí“, interpretací predikátů jako „mezilidských vztahů“, a tak dále, postupujte takto:

1. Interpretujte nejprve neformálně a uvědomte si, *důvod*, proč se daná sentence překládá jako pravdivá či jako nepravdivá.

2. Poté interpretaci zužte na nezbytně nutnou „množinu lidí“ tak, jak jsme to udělali v případě „vlasatosti“ o několik řádků výše. Všechny nejednoznačnosti přirozeného jazyka tak zmizí.

Vyhnete se tím spoustě nepříjemných dotazů typu: co to přesně znamená „být otcem“, „být bratrem“, a podobně.



3.1.32 Příklad Ukážeme, že sentence $\forall x.(P(x) \vee \neg P(x))$ jazyka \mathcal{L} z příkladu 3.1.16 je tautologie.

Především $\forall x.(P(x) \vee \neg P(x))$ je v daném jazyce skutečně sentence. Nyní musíme ukázat, že $\forall x.(P(x) \vee \neg P(x))$ je pravdivá v každé interpretaci $\langle U, \llbracket - \rrbracket \rangle$ jazyka \mathcal{L} . Napišme tedy *obecnou interpretaci*:

jazyk \mathcal{L}	libovolná interpretace
st. proměnné: x	$U \neq \emptyset$
predikáty: P arity 1	$\llbracket P \rrbracket \subseteq U$
funkční symboly: z arity 0	$\llbracket z \rrbracket \in U$

Všimněme si, že v této interpretaci píšeme jenom to, co jsme *povinováni* napsat: universum U je nějaká neprázdná množina, $\llbracket P \rrbracket$ je nějaká (pevná) podmnožina U , $\llbracket z \rrbracket$ je nějaký (pevný) prvek universa U . Nic víc o obecné interpretaci nemůžeme říci!

Jak se v této obecné interpretaci přeloží sentence $\forall x.(P(x) \vee \neg P(x))$? Jako tvrzení, že každý prvek U má buď vlastnost P nebo ne. To je ale pravdivé tvrzení.

Ukázali jsme, že sentence $\forall x.(P(x) \vee \neg P(x))$ je pravdivá v jakékoli interpretaci. Tedy $\forall x.(P(x) \vee \neg P(x))$ je tautologie.



3.1.33 Příklad Popište jazyk \mathcal{L} predikátové logiky, ve kterém jsou řetězce

$$\forall x.\exists y.R(x, y) \quad \text{a} \quad \exists y.\forall x.R(x, y)$$

sentence a rozhodněte, zda platí

$$\forall x.\exists y.R(x, y) \models \exists y.\forall x.R(x, y)$$

Popis jazyka \mathcal{L} je jednoduchý: nemáme jinou možnost, než deklarovat x, y jako standardní proměnné a R jako predikátový symbol arity 2.

Z definice 3.1.28 víme, že $\forall x.\exists y.R(x, y) \models \exists y.\forall x.R(x, y)$ platí právě tehdy, když jsou obě sentence pravdivé ve stejných interpretacích.

Popíšeme nyní všechny interpretace $\langle U, \llbracket - \rrbracket \rangle$, ve kterých je pravdivá sentence $\forall x.\exists y.R(x, y)$, a pak všechny interpretace $\langle U, \llbracket - \rrbracket \rangle$, ve kterých je pravdivá sentence $\exists y.\forall x.R(x, y)$.

1. Obecná interpretace, ve které je pravdivá sentence $\forall x.\exists y.R(x, y)$, musí vypadat takto:

jazyk \mathcal{L}	obecná interpretace, ve které je $\forall x.\exists y.R(x, y)$ pravdivá
st. proměnné: x, y	$U \neq \emptyset$
predikáty: R arity 2	$\llbracket R \rrbracket \subseteq U \times U$ tak, že každý prvek U najdeme jako první položku na seznamu uspořádaných dvojic $\llbracket R \rrbracket$

2. Obecná interpretace, ve které je pravdivá sentence $\exists y.\forall x.R(x, y)$, musí vypadat takto:

jazyk \mathcal{L}	obecná interpretace, ve které je $\exists y.\forall x.R(x, y)$ pravdivá
st. proměnné: x, y	$U \neq \emptyset$
predikáty: R arity 2	$\llbracket R \rrbracket \subseteq U \times U$ tak, že existuje prvek $u \in U$, který je druhou položkou na seznamu uspořádaných dvojic $\llbracket R \rrbracket$ a navíc prvek u má k sobě všechny prvky U jako první položky seznamu uspořádaných dvojic $\llbracket R \rrbracket$

Je snadné nalézt interpretaci $\langle U, \llbracket - \rrbracket \rangle$ prvního typu, která není interpretací druhého typu. Opět zvolíme interpretaci co možná nejjednodušší:

jazyk \mathcal{L}	interpretace
st. proměnné: x, y predikáty: R arity 2	$U = \{u, v\}$ $\llbracket R \rrbracket = \{(u, u), (v, v)\}$

V této interpretaci je sentence $\forall x. \exists y. R(x, y)$ pravdivá (protože každý prvek množiny U je první položkou na seznamu dvojic $\llbracket R \rrbracket$) a sentence $\exists y. \forall x. R(x, y)$ nepravdivá (protože neexistuje prvek množiny U , který by byl „universální“ druhou položkou).

Ukázali jsme, že sémantická ekvivalence $\forall x. \exists y. R(x, y) \models \exists y. \forall x. R(x, y)$ *neplatí*.

K příkladu 3.1.33 se vrátíme v příkladu 4.2.23 poté, až zavedeme resoluční algoritmus v predikátové logice.

3.2 Sémantický důsledek v predikátové logice

3.2.1 Definice Ať M je množina sentencí a φ je sentence jazyka \mathcal{L} . Řekneme, že φ je *sémantickým důsledkem* množiny M (značení $M \models \varphi$), pokud pro každou interpretaci $\langle U, \llbracket - \rrbracket \rangle$ platí:

jestliže jsou všechny sentence z množiny M v interpretaci $\langle U, \llbracket - \rrbracket \rangle$ pravdivé, potom je v interpretaci $\langle U, \llbracket - \rrbracket \rangle$ pravdivá i sentence φ .



3.2.2 Příklad Ukažte, že $\exists y. \forall x. R(x, y) \models \forall x. \exists y. R(x, y)$ platí.

Jazyk \mathcal{L} je stejný jako v příkladu 3.1.33. Máme pro každou interpretaci $\langle U, \llbracket - \rrbracket \rangle$ ukázat následující:

jestliže je sentence $\exists y. \forall x. R(x, y)$ v interpretaci $\langle U, \llbracket - \rrbracket \rangle$ pravdivá, potom je v interpretaci $\langle U, \llbracket - \rrbracket \rangle$ pravdivá i sentence $\forall x. \exists y. R(x, y)$.

Musíme tedy vzít obecnou interpretaci $\langle U, \llbracket - \rrbracket \rangle$, ve které je sentence $\exists y. \forall x. R(x, y)$ pravdivá a *ukázat*, že v této interpretaci je pravdivá sentence $\forall x. \exists y. R(x, y)$.

jazyk \mathcal{L}	obecná interpretace, ve které je $\exists y. \forall x. R(x, y)$ pravdivá
st. proměnné: x, y predikáty: R arity 2	$U \neq \emptyset$ $\llbracket R \rrbracket \subseteq U \times U$ tak, že existuje prvek $u \in U$, který je druhou položkou na seznamu uspořádaných dvojic $\llbracket R \rrbracket$ a navíc prvek u má k sobě všechny prvky U jako první položky seznamu uspořádaných dvojic $\llbracket R \rrbracket$

Zvolíme libovolné $v \in U$. Máme ukázat existenci prvku U , který by byl k v druhou položkou na seznamu dvojic $\llbracket R \rrbracket$. Takovou druhou položkou je prvek u .

Ukázali jsme, v interpretaci $\langle U, \llbracket - \rrbracket \rangle$, ve které je sentence $\exists y. \forall x. R(x, y)$ pravdivá, je pravdivá i sentence $\forall x. \exists y. R(x, y)$.

Sémantický důsledek $\exists y. \forall x. R(x, y) \models \forall x. \exists y. R(x, y)$ *platí*.

I v predikátové logice platí věta o sémantickém důkazu sporem. Pochopitelně je formulována pouze pro sentence. Věta 3.2.3 (ve spolupráci s CNF) je základem *resolučních algoritmů*, viz například skriptum

☞ M. Demlová a B. Pondělíček, *Matematická logika*, skriptum FEL ČVUT, Praha, 1997

a kapitola 4 tohoto textu.



3.2.3 Věta (O sémantickém důkazu sporem) Ať M je množina sentencí a ať φ je sentence jazyka \mathcal{L} . Pak jsou následující tvrzení ekvivalentní:

1. Sémantický důsledek $M \models \varphi$ platí.

2. Množina $M \cup \{\neg\varphi\}$ není splnitelná.

DŮKAZ. Důkaz rozdělíme na dvě části:

1. Ať platí $M \models \varphi$. Chceme ukázat, že neexistuje interpretace $\langle U, \llbracket - \rrbracket \rangle$ jazyka \mathcal{L} , ve které by všechny sentence z množiny $M \cup \{\neg\varphi\}$ byly pravdivé současně. Budeme postupovat sporem: ať $\langle U, \llbracket - \rrbracket \rangle$ je interpretace, ve které jsou všechny sentence z množiny $M \cup \{\neg\varphi\}$ pravdivé současně.

To znamená, že v interpretaci $\langle U, \llbracket - \rrbracket \rangle$ jsou pravdivé všechny sentence z množiny M a sentence φ v interpretaci $\langle U, \llbracket - \rrbracket \rangle$ pravdivá není. Pak ale neplatí $M \models \varphi$, a to je spor.

2. Ať množina $M \cup \{\neg\varphi\}$ není splnitelná. Chceme ukázat, že sémantický důsledek $M \models \varphi$ platí.

Zvolme proto jakoukoli interpretaci $\langle U, \llbracket - \rrbracket \rangle$, ve které jsou všechny sentence z množiny M pravdivé. Potom v interpretaci $\langle U, \llbracket - \rrbracket \rangle$ musí být sentence $\neg\varphi$ nepravdivá.

To znamená, že φ je v interpretaci $\langle U, \llbracket - \rrbracket \rangle$ pravdivá a my dokázali, že $M \models \varphi$ platí. ■

3.3 Cvičení

3.3.1 Cvičení Popište přesně rekursivní algoritmus pro tvorbu syntaktického stromu v predikátové logice.

3.3.2 Cvičení Rozhodněte, zda platí:

- $\forall x.(P(x) \wedge Q(y)) \models (\forall x.P(x)) \wedge Q(y)$.
- $\forall x.\exists y.R(x, y) \models \forall x.R(x, f(x))$.

U každého zadání popište jazyk predikátové logiky, ve kterém jde o sentence.

3.3.3 Cvičení V tomto cvičení prozkoumáme *figury syllogismů* dobře známé ze středověké filosofie.

Tradiční čtyři druhy vět⁵ zformalizujeme v jazyce \mathcal{L} predikátové logiky, který má x jako standardní proměnnou a P, Q jako predikáty arity 1.

Věta typu A je sentence $\forall x.(P(x) \Rightarrow Q(x))$.

Takovým sentencím se v klasické filosofii říká *všeobecně kladné*. Příkladem takové věty v češtině je *Všichni lidé jsou smrtelní*.

Věta typu E je sentence $\neg\exists x.(P(x) \wedge Q(x))$.

Takovým sentencím se v klasické filosofii říká *všeobecně záporné*. Příkladem takové věty v češtině je *Žádný člověk není rostlina*.

Věta typu I je sentence $\exists x.(P(x) \wedge Q(x))$.

Takovým sentencím se v klasické filosofii říká *částečně kladné*. Příkladem takové věty v češtině je *Někteří lidé jsou francouzi*.

Věta typu O je sentence $\neg\forall x.(P(x) \Rightarrow Q(x))$.

Takovým sentencím se v klasické filosofii říká *částečně záporné*. Příkladem takové věty v češtině je *Některá zvířata nemají křídla*.

⁵Názvy těchto vět jsou z latinských *affirmo* (tvrdím) a *nego* (popírám). Figury syllogismů popsal Aristoteles ze Stageiry (384–322) v knize *První analytiky*. Objevení této knihy v západní Evropě ve 12. století vedlo k prudkému rozmachu středověké filosofické logiky.

Figury syllogismů ve scholastické filosofii jsou trojice vět, kde první dvě jsou předpoklady úsudku a třetí věta je závěr úsudku.

Například trojice AAA (takzvaná *figura Barbara*) je ve formálním pojetí sématický důsledek tvaru

$$\{\forall x.(P(x) \Rightarrow Q(x)), \forall x.(Q(x) \Rightarrow R(x))\} \models \forall x.(P(x) \Rightarrow R(x))$$

Popište jazyk predikátové logiky, ve kterém jde o sentence a ukažte, že figura Barbara je správným úsudkem.

Zodpovězte následující otázky:

1. Kolik je všech různých figur syllogismů?
2. Napište figuru AII (figuru *Darii*). Je správným úsudkem?
3. Existuje nějaká nesprávná figura syllogismu?

Revize kapitoly

Dozvěděli jsme se:

- ✓ Predikátová logika je *formální* jazyk. Má přesně definovanou *syntaxi* (tj. pravidla jak se věci píšou) a *sémantiku* (tj. pravidla co věci znamenají). Syntaxe přitom *řídí* sémantiku, tj. na pochopení významu syntakticky složitých řetězců musíme nejprve porozumět významu syntakticky jednoduchých řetězců.
- ✓ Sémantické otázky v predikátové logice musíme (zatím) řešit konstrukcí nejrůznějších *interpretací*. V predikátové logice nic jako pravdivostní tabulky *neexistuje*.

Pro přípravu na zkoušku zkuste zodpovědět následující otázky:

- ✓ Zformulujte (neformálně) základní rozdíly mezi výrokovou a predikátovou logikou.
- ✓ Popište syntaxi a sémantiku predikátové logiky.
- ✓ Naučte se formalizovat české věty typu A, E, I, O ze cvičení **3.3.3**.

Doplňující literatura

Doplňující literatura je shodná se seznamem ke kapitole **2**.

Kapitola 4

Resoluční algoritmy

Trpí Politikovou Logikou: Něco se musí udělat. Toto je něco. Takže toto se musí udělat.

Jonathan Lynn a Antony Jay, *Ano, pane ministře*

Resoluční algoritmy, které v této kapitole zavedeme, jsou *syntaktickým* nástrojem, který řeší *sémantickou úlohu* o platnosti $M \models \varphi$. Abychom se co nejdříve přiblížili uplatnění těchto algoritmů v computer science, budeme v této kapitole vždy předpokládat, že množina M je *konečná*.

Uvidíme, že resoluční algoritmy ve výrokové a predikátové logice jsou ve své podstatě *totožné*. Pochopitelně, algoritmus v predikátové logice je poněkud techničtější, což je dáno složitostí syntaxe predikátové logiky.

Oba algoritmy využívají sémantickou větu o důkazu sporem: $M \models \varphi$ platí právě tehdy, když množina $M \cup \{\neg\varphi\}$ není splnitelná (viz věty 2.3.10 a 3.2.3). Oba algoritmy zjišťují splnitelnost množiny $X = M \cup \{\neg\varphi\}$ tak, že vytvářejí generace důsledků dvojic faktů z množiny X . Populace těchto důsledků jednou zastaví svůj růst a pak stačí jen tuto populaci testovat na přítomnost formule **ff**.

Pro vytváření důsledků je vhodné nejprve množinu X upravit na množinu $kt(X)$, které říkáme *klausální tvar* množiny X . Tato úprava bude ryze syntaktická a bude mít následující dvě výhody:

1. Platí, že X je splnitelná právě tehdy, když $kt(X)$ je splnitelná. Bude tedy stačit jen prohledat důsledky množiny $kt(X)$.

Ve výrokové logice bude tvorba množiny $kt(X)$ extrémně jednoduchá, jde o aplikaci tvorby konjunktivních normálních forem.

V predikátové logice bude tvorba množiny $kt(X)$ o něco složitější: velmi často budeme muset pracovat ve *Skolemově rozšíření*¹ původního jazyka.

2. Důsledky dvojic faktů z množiny $kt(X)$ se hledají velmi snadnou syntaktickou metodou, které říkáme *tvorba resolvent*.

Ve výrokové logice bude tvorba resolvent naprosto triviální.

V predikátové logice se bude tvorba resolvent opírat o existenci *maximálního unifikátoru*, což je jistá substituce.

4.1 Resoluční algoritmus ve výrokové logice

4.1.1 Definice Ať X je konečná množina formulí výrokové logiky. Symbolem $kt(X)$ (čteme: *klausální tvar množiny X*) označíme jakoukoli množinu klausulí pro CNF s vlastností: $\alpha \in kt(X)$ právě tehdy, když α je klausulí pro CNF nějaké formule z množiny X .

Klausální tvar dané množiny X není pochopitelně určen jednoznačně. Je to dáno nejednoznačností tvorby konjunktivních normálních forem.

¹Pojmenováno podle norského logika Thoralfa Alberta Skolema (1887–1963).

4.1.2 Příklad Ať $a, b, c \in At$, $X = \{a \Rightarrow (b \vee c), (b \vee c) \wedge (b \vee \neg c)\}$.

Potom CNF jednotlivých formulí mohou mít tvar: $\neg a \vee b \vee c$ a $(b \vee c) \wedge (b \vee \neg c)$ a příslušné $kt(X)$ je množina $\{\neg a \vee b \vee c, b \vee c, b \vee \neg c\}$.

Jinou možností je druhou formuli v množině X upravit na synonymum b a příslušné $kt(X)$ je množina $\{\neg a \vee b \vee c, b\}$.

Předchozí příklad ukázal, jak $kt(X)$ najít obecně: pro každou formuli z množiny X nalezněte její CNF a do $kt(X)$ zapište všechny vzniklé klausule.

4.1.3 Definice Řekneme, že množiny A a B formulí výrokové logiky jsou *ekvisplnitelné*, když A a B jsou buď obě současně splnitelné nebo jsou A a B obě současně nesplnitelné.



4.1.4 Poznámka Pozor! Definice 4.1.3 je nejdůležitějším pojmem tohoto odstavce. Je dobré si opět uvědomit rozdíl mezi splnitelností a splněností.

Množiny formulí $\{a, \neg b\}$ a $\{\neg a, b\}$, kde $a, b \in At$ jsou ekvisplnitelné (jsou totiž obě splnitelné). Neexistuje však ohodnocení u , ve kterém by obě množiny byly současně splněny.

4.1.5 Tvzení Množiny X a $kt(X)$ jsou ekvisplnitelné.

DŮKAZ. Množina X je splnitelná právě tehdy, když existuje ohodnocení u , ve kterém jsou všechny formule z množiny X pravdivé. To nastává právě tehdy, když existuje ohodnocení u , ve kterém jsou pravdivé i všechny příslušné CNF, a tudíž i všechny získané klausule.

Celkově: $kt(X)$ je splnitelná množina formulí právě tehdy, když X je splnitelná množina formulí. ■

4.1.6 Definice Řekneme, že dvě klausule α_1 a α_2 mají *komplementární výskyt* atomické formule a , když buď α_1 obsahuje literál a a α_2 obsahuje literál $\neg a$ nebo naopak.

4.1.7 Definice Ať α_1 a α_2 jsou dvě klausule s komplementárním výskytem atomické formule a . Jako $res_a(\alpha_1, \alpha_2)$ (čteme: *resolventa klausulí α_1, α_2 podle a*) označíme klausuli, která obsahuje všechny literály obsažené v α_1 a α_2 kromě literálů a a $\neg a$.



4.1.8 Poznámka Připomeňme, že mají-li klausule α_1 i α_2 pouze jeden literál, je $res_a(\alpha_1, \alpha_2) = \mathbf{ff}$. To je totiž klausule s nulovým počtem literálů.

Následující tvrzení bude základem resolučního algoritmu: tvorba resolvent nijak nemění splnitelnost. Splnitelnost tak budeme moci ověřovat postupnou tvorbou resolvent.

4.1.9 Tvzení Ať α_1 a α_2 jsou dvě klausule s komplementárním výskytem atomické formule a . Potom jsou množiny $\{\alpha_1, \alpha_2\}$ a $\{res_a(\alpha_1, \alpha_2)\}$ ekvisplnitelné.

DŮKAZ. Bez újmy na obecnosti předpokládejme, že literál a se vyskytuje v α_1 (v opačném případě prohodte α_1 a α_2). Potom můžeme psát $\alpha_1 = \beta_1 \vee a$ a $\alpha_2 = \beta_2 \vee \neg a$ a $res_a(\alpha_1, \alpha_2) = \beta_1 \vee \beta_2$.

Důkaz rozdělíme na dvě části:

1. Ukážeme, že jestliže je množina $\{\alpha_1, \alpha_2\}$ splnitelná, pak je splnitelná i množina $\{res_a(\alpha_1, \alpha_2)\}$.

Předpokládejme, že u je ohodnocení, ve kterém platí $u(\alpha_1) = u(\alpha_2) = 1$. Dokážeme, že $u(\beta_1 \vee \beta_2) = 1$. Mohou nastat dva případy:

- (a) $u(a) = 1$. Potom musí platit $u(\beta_2) = 1$, protože $u(\alpha_2) = 1$. Tudíž platí $u(\beta_1 \vee \beta_2) = 1$.
- (b) $u(a) = 0$. Potom musí platit $u(\beta_1) = 1$, protože $u(\alpha_1) = 1$. Tudíž platí $u(\beta_1 \vee \beta_2) = 1$.

2. Předpokládejme, že množina $\{\text{res}_a(\alpha_1, \alpha_2)\}$ je splnitelná. Chceme dokázat, že je splnitelná i množina $\{\alpha_1, \alpha_2\}$. Budeme postupovat sporem: ať $\{\alpha_1, \alpha_2\}$ splnitelná není.

To znamená, že v každém ohodnocení u platí $u(\alpha_1) = u(\alpha_2) = 0$. Tudíž musí v každém ohodnocení u platit $u(\beta_1 \vee \beta_2) = 0$. To je spor: my předpokládali, že množina $\{\text{res}_a(\alpha_1, \alpha_2)\}$ je splnitelná. ■

4.1.10 Důsledek Ať α_1 a α_2 jsou dvě klausule s komplementárním výskytem atomické formule a . Potom platí $\{\alpha_1, \alpha_2\} \models \text{res}_a(\alpha_1, \alpha_2)$.

DŮKAZ. V první části důkazu tvrzení 4.1.9 jsme vlastně dokázali, že platí $\{\alpha_1, \alpha_2\} \models \text{res}_a(\alpha_1, \alpha_2)$. ■

Resolventa dvojice je tedy sémantickým důsledkem příslušné dvojice. Tvorbu resolvent budeme iterovat. Začneme-li s konečnou množinou klausulí, tato tvorba se jednou musí zastavit.

4.1.11 Definice Ať X' je konečná množina klausulí pro CNF. Posloupnost $\text{Res}_0(X')$, $\text{Res}_1(X')$, $\text{Res}_2(X')$, $\text{Res}_3(X')$, ... definujeme takto:

$$\begin{aligned}\text{Res}_0(X') &= X' \\ \text{Res}_{n+1}(X') &= \text{Res}_n(X') \cup \{\alpha \mid \alpha \text{ je resolventa nějaké dvojice z } \text{Res}_n(X')\}\end{aligned}$$

4.1.12 Věta (Resoluční algoritmus) Ať X' je konečná množina klausulí pro CNF. Potom existuje n_0 tak, že platí $\text{Res}_{n_0+1}(X') = \text{Res}_{n_0}(X')$. Dále platí:

1. Množiny X' a $\text{Res}_{n_0}(X')$ jsou ekvisplnitelné.
2. Množina X' není splnitelná právě tehdy, když platí $\text{ff} \in \text{Res}_{n_0}(X')$.

DŮKAZ. Tvorba posloupnosti $\text{Res}_0(X')$, $\text{Res}_1(X')$, $\text{Res}_2(X')$, ... se musí zastavit, protože množina X' je konečná.

Podmínka 1. plyne okamžitě z tvrzení 4.1.9 a podmínka 2. plyne z toho, že množina X' není splnitelná právě tehdy, když platí $X' \models \text{ff}$. ■



4.1.13 Poznámka Algoritmus 4.1.12 lze v určitém případě zrychlit. Jakmile se totiž formule ff objeví v nějaké množině $\text{Res}_n(X')$, zůstává i v každé množině $\text{Res}_m(X')$, kde $m > n$, protože zjevně platí

$$\text{Res}_0(X') \subseteq \text{Res}_1(X') \subseteq \text{Res}_2(X') \subseteq \text{Res}_3(X') \subseteq \dots$$

A proto bude platit $\text{ff} \in \text{Res}_{n_0}(X')$.

Tudíž tvorbu posloupnosti $\text{Res}_0(X')$, $\text{Res}_1(X')$, $\text{Res}_2(X')$, $\text{Res}_3(X')$, ... můžeme přerušit v okamžiku, kdy vyjde resolventa ff . V tento okamžik se totiž dozvídáme, že množina X' není splnitelná.

4.1.14 Poznámka Ukázali jsme, že splnitelnost je *invariantem* rekursivního algoritmu, který tvoří posloupnost $\text{Res}_0(X')$, $\text{Res}_1(X')$, $\text{Res}_2(X')$, $\text{Res}_2(X')$, ... Hledání invariantu rekursivního algoritmu je důležitou součástí zkoumání *korektnosti* rekursivních algoritmů, viz texty

☞ J. Velebil, *Diskrétní matematika*, <http://math.feld.cvut.cz/velebil/>

☞ J. Velebil, *Logika programů*, <http://math.feld.cvut.cz/velebil/>

nebo knihu

☞ R. C. Backhouse, *Program Construction and Verification*, Prentice-Hall, London, 1986

ve které je podrobně popsána teorie designu korektních rekursivních algoritmů pomocí teorie invariantu.

Nyní již máme vše pro to, abychom vysvětlili, jak resolučním algoritmem postupovat při rozhodování, zda $M \models \varphi$ platí. Základem je pochopitelně věta 2.3.10: úlohu o platnosti $M \models \varphi$ můžeme převést na úlohu o splnitelnosti množiny $M \cup \{\neg\varphi\}$. Splnitelnost množiny umíme ověřovat tvorbou resolvent.

4.1.15 Jak resolučním algoritmem zjistit, zda $M \models \varphi$ platí

Postupujte následovně:

1. Vytvořte množinu $X = M \cup \{\neg\varphi\}$.
2. Spočítejte $kt(X)$ a vzniklou množinu klausulí označte X' .
3. Naleznete n_0 tak, že platí $\text{Res}_{n_0+1}(X') = \text{Res}_{n_0}(X')$.
4. $M \models \varphi$ platí právě tehdy, když platí $\mathbf{ff} \in \text{Res}_{n_0}(X')$.



4.1.16 Poznámka Na přednášce bude zmíněna i mírná varianta klasického resolučního algoritmu 4.1.12, kterou v tomto textu nenajdete. Tuto variantu (*resoluční tabulku*) naleznete ve skriptu

☞ M. Demlová a B. Pondělíček, *Matematická logika*, FEL ČVUT, Praha 1997

Resoluční tabulka však *není pravdivostní tabulka!* Tvorba resoluční tabulky je opět *syntaktický algoritmus*.

4.2 Resoluční algoritmus v predikátové logice

Hlavní rysy resolučního algoritmu v predikátové logice jsou totožné s algoritmem ve výrokové logice. Narazíme však na dvě technické potíže:

1. *Klausální tvar*: ten vyžaduje zavedení CNF v predikátové logice, což jsme ještě neudělali. Zatímco ve výrokové logice jde u CNF o hledání jistého synonyma, v predikátové logice bude situace složitější.
2. *Tvorba resolvent*: komplementarita literálů pro úspěšnou tvorbu resolventy dvou klausulí bude v predikátové logice podmíněna hledáním speciální substituce, které se říká *maximální unifikátor*.

Začneme definicí formule v CNF.

4.2.1 Definice Řekneme, že *sentence* ψ jazyka \mathcal{L} je v CNF, když platí buď

$$\psi = \mathbf{ff}$$

nebo

$$\psi = \psi_1 \wedge \dots \wedge \psi_n$$

pro nějaké přirozené číslo $n \geq 1$, kde každá formule ψ_i ($i \in \{1, \dots, n\}$) je buď rovna formuli **tt**, tj. tautologii, nebo je napsána ve tvaru

$$\forall \vec{x}. (l_1 \vee \dots \vee l_{k_i})$$

pro nějaké přirozené číslo $k_i \geq 1$ a každé l_j ($j \in \{1, \dots, k_i\}$) je buď atomická nebo negace atomické formule, a kde zápisem $\forall \vec{x}$ rozumíme konečný řetězec kvantifikátorů \forall , které váží všechny standardní proměnné ve formulích l_j ($j \in \{1, \dots, k_i\}$).

V tomto kontextu každé formuli ψ_i ($i \in \{1, \dots, n\}$) říkáme *klausule pro CNF*, formuli $l_1 \vee \dots \vee l_{k_i}$ říkáme *tělo klausule* a každému l_j ($j \in \{1, \dots, k_i\}$) říkáme *literál*.

4.2.2 Příklad Pro jazyk \mathcal{L} , kde $x, y, z \in \text{Var}$, $P, Q \in \text{Pred}$ arity 2 a $f \in \text{Func}$ arity 1, $a \in \text{Func}$ arity 0, je sentence

$$\forall x. \forall z. (P(x, f(z)) \vee \neg P(z, z)) \wedge \forall y. (P(y, y) \vee Q(y, a))$$

v CNF.

Tato sentence má dvě klausule: $\forall x. \forall z. (P(x, f(z)) \vee \neg P(z, z))$ a $\forall y. (P(y, y) \vee Q(y, a))$.

V první klausuli jsou formule $P(x, f(z))$ a $\neg P(z, z)$ literály, druhá klausule má také dva literály: $P(y, y)$ a $Q(y, a)$.

Tělo první klausule je formule $P(x, f(z)) \vee \neg P(z, z)$, tělo druhé klausule je formule $P(y, y) \vee Q(y, a)$.



4.2.3 Jak převést sentenci φ jazyka \mathcal{L} na CNF

1. Použijte α -konversi (viz 3.1.12) tak, aby každý kvantifikátor ve φ vázal jinou proměnnou.

Pozor! To může znamenat změnu jazyka (deklaraci nových — fresh — standardních proměnných)!

2. Odstraňte nepohodlné spojky \Rightarrow , \Leftrightarrow pomocí sémantických pravidel tak, aby sentence φ obsahovala pouze spojky \wedge , \vee a \neg .

K tomu použijte pravidla

$$\alpha \Rightarrow \beta \models \neg\alpha \vee \beta \quad \text{a} \quad \alpha \Leftrightarrow \beta \models (\neg\alpha \vee \beta) \wedge (\neg\beta \vee \alpha)$$

3. Spojku \neg přestěhujte po syntaktickém stromu až těsně před atomické formule.

K tomu použijte pravidla

$$\neg(\alpha \wedge \beta) \models \neg\alpha \vee \neg\beta \quad \neg(\alpha \vee \beta) \models \neg\alpha \wedge \neg\beta \quad \neg\forall \vec{x}. \alpha \models \exists \vec{x}. \neg\alpha \quad \neg\exists \vec{x}. \alpha \models \forall \vec{x}. \neg\alpha$$

4. Spojku \vee přestěhujte co nejnižší po syntaktickém stromu.

K tomu použijte pravidla

$$(\alpha \wedge \beta) \vee \gamma \models (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$$

a dvě nová pravidla

$$\exists \vec{x}. \alpha \vee \exists \vec{x}. \beta \models \exists \vec{x}. (\alpha \vee \beta) \quad \forall \vec{x}. \alpha \vee \beta \models \forall \vec{x}. (\alpha \vee \beta)$$

kde v pravidle napravo se ve formuli β nevyskytuje \vec{x} .

5. Spojku \wedge přestěhujte co nejvýše po syntaktickém stromu a odstraňte existenční kvantifikátory *Skolemovým rozšířením* (tomuto postupu se také říká *skolemisace*).

K tomu použijte pravidla

$$\forall \vec{x}. (\alpha \wedge \beta) \models \forall \vec{x}. \alpha \wedge \forall \vec{x}. \beta \quad \exists \vec{x}. (\alpha \wedge \beta) \models \exists \vec{x}. \alpha \wedge \beta$$

kde v pravidle napravo se ve formuli β nevyskytuje \vec{x} .

Pro odstranění existenčních kvantifikátorů použijte pravidla

- (a) Sentenci $\exists y. \alpha$ nahraďte sentencí $\alpha[y := a]$, kde a je *fresh* funkční symbol arity 0.

Znakem $\alpha[y := a]$ tu rozumíme dosazení a za každý výskyt standardní proměnné y ve formuli α .

- (b) Sentenci $\forall x_1 \dots \forall x_n. \exists y. \alpha$, kde $n \geq 1$, nahraďte sentencí $\alpha[y := f(x_1, \dots, x_n)]$, kde f je *fresh* funkční symbol arity n .

Znakem $\alpha[y := f(x_1, \dots, x_n)]$ tu rozumíme dosazení termu $f(x_1, \dots, x_n)$ za každý výskyt proměnné y ve formuli α .

Pozor! Body (a) a (b) *mění jazyk!* Symboly a a f musí být pro každé použití *fresh symboly*, tj. dosud nepoužité. Při odstraňování existenčních kvantifikátorů tedy zavedete přesně tolik nových funkčních symbolů, kolik odstraníte existenčních kvantifikátorů.

6. Výslednou formuli označte $\text{cnf}(\varphi)$.

Všechny výše uvedené úpravy, kromě odstraňování existenčních kvantifikátorů, jsou zcela analogické úpravám pro hledání CNF a DNF ve výrokové logice pomocí syntaktických stromů, viz cvičení 2.4.5. Uvedme příklad na odstraňování existenčních kvantifikátorů:

4.2.4 Příklad Ať $x, y, z, t \in \text{Var}$, $P \in \text{Pred}$ arity 3 a $a \in \text{Func}$ arity 0. Odstraňte Skolemovým rozšířením existenční kvantifikátory v sentencích

$$\exists x. P(x, a, a) \quad \exists x. \exists y. P(a, y, x) \quad \forall x. \forall y. \exists z. P(y, z, x) \quad \forall x. \exists t. \forall y. \exists z. (P(y, z, x) \wedge P(t, t, z))$$

Probereme jednotlivé formule:

1. Na formuli $\exists x. P(x, a, a)$ použijeme bezprostředně první pravidlo. Zavedeme *nový* funkční symbol arity 0, nazvěme jej b .

Formule s odstraněným existenčním kvantifikátorem je $P(b, a, a)$.

2. Na formuli $\exists x. \exists y. P(a, y, x)$ použijeme první pravidlo dvakrát.

Nejprve odstraníme $\exists x$. Zavedeme *nový* funkční symbol b arity 0. Výsledkem je $\exists y. P(a, y, b)$.

Poté odstraníme $\exists y$. Zavedeme *nový* funkční symbol c arity 0. Výsledkem je $P(a, c, b)$.

3. Na formuli $\forall x. \forall y. \exists z. P(y, z, x)$ použijeme druhé pravidlo. Zavedeme *nový* funkční symbol f arity 2, nazvěme jej f .

Formule s odstraněným existenčním kvantifikátorem je $\forall x. \forall y. P(y, f(x, y), x)$.

4. Na formuli $\forall x. \exists t. \forall y. \exists z. (P(y, z, x) \wedge P(t, t, z))$ použijeme druhé pravidlo dvakrát.

Nejprve odstraníme $\exists t$ takto: zavedeme *nový* funkční symbol arity 1 a nazvěme jej g .

Získáme formuli $\forall x. \forall y. \exists z. (P(y, z, x) \wedge P(g(x), g(x), z))$. Nyní opět použijeme druhé pravidlo a zavedeme *nový* funkční symbol h arity 2.

Výsledkem je formule $\forall x. \forall y. (P(y, h(x, y), x) \wedge P(g(x), g(x), h(x, y)))$.

Všimněte si, že jména symbolů jsou skutečně nová, při každé aplikaci jakéhokoli pravidla zavádíme další funkční symbol.

4.2.5 Příklad Nalezněte $\text{cnf}(\varphi)$ pro sentenci $\varphi = \forall x. (x = a \Rightarrow (\exists x. (Q(x) \wedge S(a))))$, kde $x \in \text{Var}$, $Q, S \in \text{Pred}$ arity 1, $a \in \text{Func}$ arity 0.

formule	pravidlo	fresh symbols
$\forall x. (x = a \Rightarrow (\exists y. (Q(y) \wedge S(a))))$	1.	$y \in \text{Var}$
$\forall x. (\neg x = a \vee (\exists y. (Q(y) \wedge S(a))))$	2.	
$\forall x. \exists y. (\neg x = a \vee (Q(y) \wedge S(a)))$	4.	
$\forall x. \exists y. ((\neg x = a \vee Q(y)) \wedge (\neg x = a \vee S(a)))$	4.	
$\forall x. ((\neg x = a \vee Q(f(x))) \wedge (\neg x = a \vee S(a)))$	5.	$f \in \text{Func}$ arity 1
$\forall x. (\neg x = a \vee Q(f(x)) \wedge \forall x. (\neg x = a \vee S(a)))$	5.	

Výsledná $\text{cnf}(\varphi)$ je sentence $\forall x. (\neg x = a \vee Q(f(x)) \wedge \forall x. (\neg x = a \vee S(a)))$.

4.2.6 Definice Řekneme, že množiny A a B sentencí (obecně v různých jazycích) jsou *ekvisplnitelné*, když A a B jsou buď obě současně splnitelné nebo jsou A a B obě současně nesplnitelné.

4.2.7 Tvzení Formule $\text{cnf}(\varphi)$, získaná ze sentence φ algoritmem 4.2.3, je sentence a je napsána v CNF. Množiny $\{\varphi\}$ a $\{\text{cnf}(\varphi)\}$ jsou navíc ekvisplnitelné.

DŮKAZ. Formule $\text{cnf}(\varphi)$ je samozřejmě sentencí (obecně v jiném jazyce než φ) a je v CNF.

Pro ekvisplnitelnost si všimněme, že algoritmus 4.2.3 používá sémantickou ekvivalenci, až na dvě pravidla odstraňující existenční kvantifikátory. Musíme tedy ověřit pouze dva následující body:

1. Sentence $\exists y. \alpha$ a $\alpha[y := a]$, kde a je fresh funkční symbol arity 0, jsou ekvivalentní.

Jestliže $\exists y. \alpha$ je pravdivá v interpretaci $\langle U, \llbracket (-) \rrbracket \rangle$ jazyka \mathcal{L} , pak existuje $d \in U$ tak, že α je pravdivá v kontextu $\rho[y := d]$. Interpretaci $\langle U, \llbracket (-) \rrbracket \rangle$ rozšířte o požadavek $\llbracket a \rrbracket = d$. V této nové interpretaci je pravdivá $\alpha[y := a]$.

Jestliže $\alpha[y := a]$ je splněna v interpretaci $\langle U, \llbracket (-) \rrbracket \rangle$ jazyka \mathcal{L} rozšířeného o funkční symbol a , pak pro hodnotu $\llbracket a \rrbracket = d$ je formule α pravdivá v kontextu $\rho[y := d]$. To znamená, že $\exists y. \alpha$ je pravdivá v interpretaci $\langle U, \llbracket (-) \rrbracket \rangle$ původního jazyka \mathcal{L} .

2. Sentence $\forall x_1 \dots \forall x_n. \exists y. \alpha$ a $\alpha[y := f(x_1, \dots, x_n)]$, kde f je fresh funkční symbol arity n , jsou ekvivalentní.

To je zcela analogické předchozímu: máme rozšířit interpretaci $\langle U, \llbracket (-) \rrbracket \rangle$ původního jazyka o funkci $\llbracket f \rrbracket : U^n \rightarrow U$. Pro jakýkoli výběr $(d_1, \dots, d_n) \in U^n$ definujeme $\llbracket f \rrbracket(d_1, \dots, d_n) = d$ tak, aby α byla pravdivá v kontextu $\rho[x_1 := d_1, \dots, x_n := d_n, y := d]$.

Jestliže tedy $\forall x_1 \dots \forall x_n. \exists y. \alpha$ je splněna v interpretaci $\langle U, \llbracket (-) \rrbracket \rangle$ původního jazyka \mathcal{L} , je splněna i formule $\alpha[y := f(x_1, \dots, x_n)]$ v interpretaci jazyka \mathcal{L} rozšířeného o funkční symbol f .

Obráceně: pokud je $\alpha[y := f(x_1, \dots, x_n)]$ splněna v interpretaci rozšířeného jazyka, je v interpretaci původního jazyka \mathcal{L} splněna i formule $\forall x_1 \dots \forall x_n. \exists y. \alpha$.

Důkaz je ukončen. ■



4.2.8 Poznámka Z důkazu tvrzení 4.2.7 je vidět, proč například a musí být *fresh* funkční symbol arity 0. Je to proto, abychom mohli *dodefinovat* interpretaci o požadavek $\llbracket a \rrbracket = d$, a aby takto rozšířená interpretace nekolidovala se starou. Proto se tomuto úkonu říká *rozšíření* — rozšiřujeme tu jazyk „nekonfliktním“ způsobem.

Všimněte si také, že v příkladu 4.2.5 jsme *nepsali* $\varphi \models \text{cnf}(\varphi)$, není to totiž obecně pravda.



4.2.9 Jak najít klausální tvar $\text{kt}(X)$ množiny X sentencí predikátové logiky

1. Pro každou sentenci $\varphi \in X$ nalezněte $\text{cnf}(\varphi)$ algoritmem 4.2.3.
2. Klausule jednotlivých sentencí $\text{cnf}(\varphi)$ zapište do množiny a tuto množinu označte $\text{kt}(X)$.

4.2.10 Tvrzení *At* X je množina sentencí predikátové logiky. Potom množiny X a $\text{kt}(X)$ jsou ekvivalentní.

DŮKAZ. To je zcela analogické situaci ve výrokové logice. Použijeme tvrzení 4.2.7. ■

Zbývá nyní definovat pojem resolventy dvou klausulí podle atomické formule. Zjišťujeme ale, že situace je daleko pestřejší než ve výrokové logice.

4.2.11 Příklad *At* $x, y \in \text{Var}$, $P, Q \in \text{Pred}$ arity 2, $f, g \in \text{Func}$ arity 1.

1. Klausule $\forall x. (P(x, x) \vee Q(x, x))$ a $\forall x. (\neg P(x, x) \vee \neg Q(x, x))$ mají komplementární výskyt atomické formule $P(x, x)$.
Jejich resolventa² je klausule $\forall x. (Q(x, x) \vee \neg Q(x, x))$.
2. Klausule $\forall x. (P(x, x) \vee Q(x, x))$ a $\forall y. (\neg P(y, y) \vee \neg Q(y, y))$ sice komplementární výskyt žádné atomické formule nemají, ale když použijeme α -konverzi na druhou klausuli a přejmenujeme y na x , dostaneme situaci z bodu 1. a resolventu vytvoříme (podle $P(x, x)$).

²Neřekli jsme zatím přesně, co resolventa v predikátové logice je, snažíme se o analogii s výrokovou logikou.

3. Klausule $\forall x. (P(x, x) \vee Q(x, x))$ a $\forall x. \forall y. (\neg P(x, y) \vee \neg Q(f(y), x))$ opět nemají komplementární výskyt žádné atomické formule. Navíc tu α -konverse nepomůže.

Kdybychom ale na druhou klausuli aplikovali *substituci* $y := x$, dostali bychom klausuli $\forall x. (\neg P(x, x) \vee \neg Q(f(x), x))$ a postupovali jako v bodě 1.

Klausule $\forall x. (\neg P(x, x) \vee \neg Q(f(x), x))$ je evidentně *důsledkem* klausule $\forall x. \forall y. (\neg P(x, y) \vee \neg Q(f(y), x))$.

Protože při tvorbě resolvent nám jde o důsledky, nemusel by tento postup vadit.

4. Klausule $\forall x. (P(x, f(x)) \vee Q(x, x))$ a $\forall x. \forall y. (\neg P(x, g(y)) \vee \neg Q(f(y), x))$ opět nemají komplementární výskyt žádné atomické formule.

Zkusíme opět použít substituci $y := x$ na druhou klausuli. Dostáváme $\forall x. (\neg P(x, g(x)) \vee \neg Q(f(x), x))$. Ještě stále nemáme komplementární výskyt atomických formulí, a proto se pokoušíme o další substituci. Potřebovali bychom nějak *ztotožnit* termy $f(x)$ a $g(x)$. To ale, alespoň na první pohled, nejspíš zařídit nepůjde.

Zjišťujeme tedy, že v predikátové logice budeme muset hledat jistou substituci, která nám dovolí *získat* komplementární výskyty atomických formulí. Této substituci budeme říkat *maximální unifikátor* a budeme ji hledat *unifikačním algoritmem*.

Myšlenka unifikačního algoritmu je jednoduchá: čtete první znaky dvou řetězců. Pokud jsou stejné, umažte je, pokud ne, snažte se rozšířit substituci o požadavek ztotožnění těchto znaků. Znak v jistých případech ztotožnit nepůjdou, pak maximální unifikátor neexistuje, pokud ztotožnit půjdou, ztotožněte je, umažte je a pokračujte dále.



4.2.12 Jak najít maximální unifikátor atomických formulí α a β .

1. Inicializace: maximální unifikátor ϑ je prázdný.
 2. Přečtete první znaky α a β (tj. příslušné predikátové symboly nebo symbol rovnosti).
Nejsou-li predikátové symboly stejné, zastavte algoritmus: *maximální unifikátor α a β neexistuje*.
Jsou-li predikátové symboly stejné, umažte je (spolu s příslušnými závorkami) a pokračuje dále.
 3. Dokud jsou oba řetězce neprázdné, dělejte následující:
Nejsou-li první znaky stejné, rozlište tyto situace:
 - (a) Jeden ze znaků je proměnná, řekněme x , druhý znak je proměnná, řekněme y .
Udělejte update $\vartheta := \vartheta \cup \{x := y\}$, tuto novou substituci proveďte na celé dva řetězce a vraťte se na začátek bodu 3.
 - (b) Jeden ze znaků je proměnná, řekněme x , druhý znak funkční symbol, řekněme f , arity n . Tento funkční symbol musí být v řetězci následován svými argumenty, řekněme, že je tam napsáno $f(t_1, \dots, t_n)$.
Zjistěte, zda se v řetězci t_1, \dots, t_n vyskytuje znak x .
Pokud ano, zastavte algoritmus: *maximální unifikátor α a β neexistuje*.
Pokud ne, udělejte update $\vartheta := \vartheta \cup \{x := f(t_1, \dots, t_n)\}$, tuto novou substituci proveďte na celé dva řetězce a vraťte se na začátek bodu 3.
 - (c) Oba znaky jsou (nutně různé) funkční symboly. Zastavte algoritmus: *maximální unifikátor α a β neexistuje*.
- Jsou-li první znaky stejné, odmažte je a vraťte se na začátek bodu 3.
4. Maximální unifikátor je ϑ .

4.2.13 Příklad Ať $x, y \in \text{Var}$, $P, Q \in \text{Pred}$ arity 2, $f \in \text{Func}$ arity 1 a $a \in \text{Func}$ arity 0.

1. Unifikujte $P(x, x)$ a $Q(x, x)$.

levý řetězec	pravý řetězec	test	maximální unifikátor
$P(x, x)$	$Q(x, x)$	$P = Q?$	neexistuje

2. Unifikujte $P(x, x)$ a $P(y, a)$.

levý řetězec	pravý řetězec	test	maximální unifikátor
$P(x, x)$	$P(y, a)$	$P = P?$	\emptyset
x, x	y, a	$x = y?$	$\{x := y\}$
y, y	y, a	$y = y?$	$\{x := y\}$
y	a	$y = a?$	$\{x := y, y := a\}$
a	a	$a = a?$	$\{x := y, y := a\}$
prázdný řetězec	prázdný řetězec		$\{x := y, y := a\}$

Po unifikaci $P(x, x)[x := y, y := a] = P(a, a)$, $P(y, a)[x := y, y := a] = P(a, a)$.

3. Unifikujte $P(x, f(a))$ a $P(y, a)$.

levý řetězec	pravý řetězec	test	maximální unifikátor
$P(x, f(a))$	$P(y, a)$	$P = P?$	\emptyset
$x, f(a)$	y, a	$x = y?$	$\{x := y\}$
$y, f(a)$	y, a	$y = y?$	$\{x := y\}$
$f(a)$	a	$f = a?$	neexistuje

4. Unifikujte $P(x, x)$ a $P(y, f(x))$.

levý řetězec	pravý řetězec	test	maximální unifikátor
$P(x, x)$	$P(y, f(x))$	$P = P?$	\emptyset
x, x	$y, f(x)$	$x = y?$	$\{x := y\}$
y, y	$y, f(y)$	$y = y?$	$\{x := y\}$
y	$f(y)$	$y = f?$ a y mezi argumenty $f?$	neexistuje

4.2.14 Definice Ať α a β jsou dvě klausule v jazyce predikátové logiky. Řekneme, že α a β mají komplementární výskyt predikátového symbolu, řekneme P , když je v α literál začínající P a β literál začínající $\neg P$, nebo naopak.

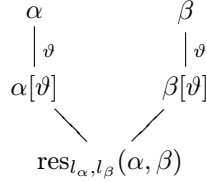
Pokud tedy klausule α a β mají komplementární výskyt nějakého predikátu, je to signál, že se můžeme pokusit příslušné atomické formule unifikovat. Pokud je unifikace úspěšná, vytvoříme resolventu (podle příslušné atomické formule), pokud maximální unifikátor neexistuje, neexistuje ani resolventa (podle příslušné atomické formule).



4.2.15 Tvorba resolventy klausulí α a β s komplementárním výskytem predikátového symbolu P

1. Označte jako l_α a l_β ty literály v α a β , které mají komplementární výskyt P .
2. Unifikujte atomické formule z literálů l_α a l_β
Pokud maximální unifikátor neexistuje, *neexistuje resolventa α a β podle l_1 a l_2* .
Pokud maximální unifikátor existuje, označte jej ϑ a pokračujte dalším krokem.
3. Provedte substituci ϑ na obě formule α, β . Výsledné formule označte $\alpha[\vartheta], \beta[\vartheta]$.
Obě formule $\alpha[\vartheta]$ a $\beta[\vartheta]$ jsou opět klausule a atomické formule, které jsou v literálech $l_\alpha[\vartheta]$ a $l_\beta[\vartheta]$, jsou v nich *stejné*.
4. Vezměte všechny literály formulí $\alpha[\vartheta]$ a $\beta[\vartheta]$ *kromě* těch dvou unifikovaných, vytvořte tělo nové klausule a doplňte toto tělo kvantifikátory \forall na klausuli.
5. Výslednou klausuli označte $\text{res}_{l_\alpha, l_\beta}(\alpha, \beta)$.

4.2.16 Poznámka Postup z algoritmu 4.2.15 se často graficky vyjadřuje takto:



Formule, které jsou na tomto obrázku na nižší hladině, jsou důsledky formulí na vyšší hladině, pokud jsou spolu spojeny hranou.

4.2.17 Příklad Ať $x, y, z \in \text{Var}$, $R, Q \in \text{Pred}$ arity 2, $f \in \text{Func}$ arity 1 a $a \in \text{Func}$ arity 0. Utvořte (pokud existuje) resolventu klausulí

$$\alpha = \forall x. \forall y. (R(x, a) \vee \neg Q(x, y)) \quad \beta = \forall z. (\neg R(a, a) \vee Q(a, f(z)))$$

Zřejmě se můžeme pokusit vytvořit resolventy *dvě*: v klausulích jsou komplementární výskyty R a Q .

1. Pro komplementární výskyt R je $l_{\alpha} = R(x, a)$, $l_{\beta} = \neg R(a, a)$.

Hledáme maximální unifikátor $R(x, a)$ a $R(a, a)$ algoritmem 4.2.12.

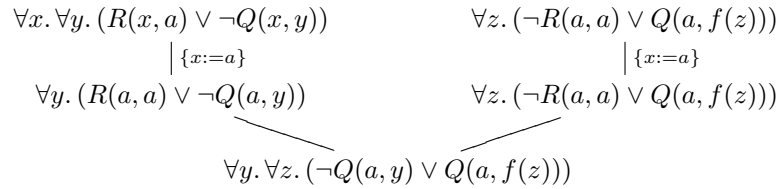
Takový unifikátor existuje: $\vartheta = \{x := a\}$.

Platí $\alpha[\vartheta] = \forall y. (R(a, a) \vee \neg Q(a, y))$ a $\beta[\vartheta] = \forall z. (\neg R(a, a) \vee Q(a, f(z)))$.

Tělo nové resolventy je $\neg Q(a, y) \vee Q(a, f(z))$.

Celá resolventa je $\text{res}_{l_{\alpha}, l_{\beta}}(\alpha, \beta) = \forall y. \forall z. (\neg Q(a, y) \vee Q(a, f(z)))$.

Graficky:



2. Pro komplementární výskyt Q je $l_{\alpha} = \neg Q(x, y)$, $l_{\beta} = Q(a, f(z))$.

Hledáme maximální unifikátor $Q(x, y)$ a $Q(a, f(z))$ algoritmem 4.2.12.

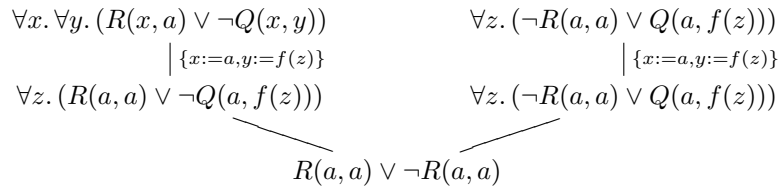
Takový unifikátor existuje: $\vartheta = \{x := a, y := f(z)\}$.

Platí $\alpha[\vartheta] = \forall z. (R(a, a) \vee \neg Q(a, f(z)))$ a $\beta[\vartheta] = \forall z. (\neg R(a, a) \vee Q(a, f(z)))$.

Tělo nové resolventy je $R(a, a) \vee \neg R(a, a)$.

Celá resolventa je $\text{res}_{l_{\alpha}, l_{\beta}}(\alpha, \beta) = R(a, a) \vee \neg R(a, a)$.

Graficky:



4.2.18 Definice Ať X' je konečná množina klausulí v predikátové logice. Posloupnost $\text{Res}_0(X')$, $\text{Res}_1(X')$, $\text{Res}_2(X')$, $\text{Res}_3(X')$, ... definujeme takto:

$$\begin{aligned}
 \text{Res}_0(X') &= X' \\
 \text{Res}_{n+1}(X') &= \text{Res}_n(X') \cup \{\alpha \mid \alpha \text{ je resolventa nějaké dvojice z } \text{Res}_n(X')\}
 \end{aligned}$$

4.2.19 Věta (Resoluční algoritmus) *At X' je konečná množina klausulí v predikátové logice. Potom existuje n_0 tak, že platí $\text{Res}_{n_0+1}(X') = \text{Res}_{n_0}(X')$. Dále platí:*

1. Množiny X' a $\text{Res}_{n_0}(X')$ jsou ekvivalentní.
2. Množina X' není splnitelná právě tehdy, když platí $\mathbf{ff} \in \text{Res}_{n_0}(X')$.

DŮKAZ. Důkaz je podobný jako důkaz věty 4.1.12. Je však poměrně technický, nebudeme jej uvádět. ■



4.2.20 Poznámka Algoritmus 4.2.19 lze v určitém případě zrychlit. Jakmile se totiž formule \mathbf{ff} objeví v nějaké množině $\text{Res}_n(X')$, zůstává i v každé množině $\text{Res}_m(X')$, kde $m > n$, protože zjevně platí

$$\text{Res}_0(X') \subseteq \text{Res}_1(X') \subseteq \text{Res}_2(X') \subseteq \text{Res}_3(X') \subseteq \dots$$

A proto bude platit $\mathbf{ff} \in \text{Res}_{n_0}(X')$.

Tudíž tvorbu posloupnosti $\text{Res}_0(X')$, $\text{Res}_1(X')$, $\text{Res}_2(X')$, $\text{Res}_3(X')$, ... můžeme přerušit v okamžiku, kdy vyjde resolventa \mathbf{ff} . V tento okamžik se totiž dozvídáme, že množina X' není splnitelná.

4.2.21 Jak resolučním algoritmem zjistit, zda $M \models \varphi$ platí (v predikátové logice)

Postupujte následovně:

1. Vytvořte množinu $X = M \cup \{\neg\varphi\}$.
2. Spočítejte $\text{kt}(X)$ algoritmem 4.2.9. Vzniklou množinu klausulí označte X' .
3. Nalezněte n_0 tak, že platí $\text{Res}_{n_0+1}(X') = \text{Res}_{n_0}(X')$.
4. $M \models \varphi$ platí právě tehdy, když platí $\mathbf{ff} \in \text{Res}_{n_0}(X')$.



4.2.22 Poznámka Na přednášce bude zmíněna i heuristika klasického resolučního algoritmu 4.2.19 (*zamítací strom*), kterou v tomto textu nenajdete. Naleznete ji ve skriptu

☞ M. Demlová a B. Pondělíček, *Matematická logika*, FEL ČVUT, Praha 1997

Poznamenejme je, že tvorba zamítacího stromu je opravdu heuristika: pokud se nám nepovede zamítací strom vytvořit napoprvé, *neznamená* to ještě, že zamítací strom neexistuje. Speciální resoluční algoritmy (například *lineární resoluce*) konstruuji zamítací stromy algoritmicky. Vysvětlení chodu těchto speciálních algoritmů je značně mimo rozsah těchto přednášek. Odkazujeme na doplňující literaturu na konci této kapitoly.

Nyní vyřešíme příklad 3.1.33 resoluční metodou.

4.2.23 Příklad Popište jazyk \mathcal{L} predikátové logiky, ve kterém jsou řetězce

$$\forall x.\exists y.R(x, y) \quad \text{a} \quad \exists y.\forall x.R(x, y)$$

sentence a resoluční metodou rozhodněte, zda platí

$$\forall x.\exists y.R(x, y) \models \exists y.\forall x.R(x, y)$$

Popis jazyka \mathcal{L} je jednoduchý: nemáme jinou možnost, než deklarovat x, y jako standardní proměnné a R jako predikátový symbol arity 2.

Protože pro jakékoli sentence α a β platí $\alpha \models \beta$ právě tehdy, když platí $\alpha \models \beta$ a $\beta \models \alpha$ současně, rozdělíme úlohu do dvou částí:

1. Resolucí rozhodněte, zda $\forall x.\exists y.R(x, y) \models \exists y.\forall x.R(x, y)$ platí.

Utvoříme množinu X : $X = \{\forall x.\exists y.R(x, y), \neg\exists y.\forall x.R(x, y)\}$.

Klausální tvar: $X' = \{\forall x.R(x, f(x)), \forall y.\neg R(g(y), y)\}$, kde f a g jsou fresh funkční symboly arity 1.

Maximální unifikátor $R(x, f(x))$ a $R(g(y), y)$ neexistuje.

Proto je $X' = \text{Res}_0(X') = \text{Res}_1(X')$.

Protože $\text{ff} \notin \text{Res}_0(X')$, důkaz sporem nejde uskutečnit: sémantický důsledek $\forall x.\exists y.R(x, y) \models \exists y.\forall x.R(x, y)$ *neplatí*.

2. Resolucí rozhodněte, zda $\exists y.\forall x.R(x, y) \models \forall x.\exists y.R(x, y)$ platí.

Utvoříme množinu X : $X = \{\exists y.\forall x.R(x, y), \neg\forall x.\exists y.R(x, y)\}$.

Klausální tvar: $X' = \{\forall x.R(x, a), \forall y.\neg R(b, y)\}$, kde a a b jsou *fresh* funkční symboly arity 0.

Maximální unifikátor $R(x, a)$ a $R(b, y)$ je $\vartheta = \{x := b, y := a\}$.

Protože platí

$$\begin{array}{ccc} \forall x.R(x, a) & & \forall y.\neg R(b, y) \\ \downarrow \{x:=b, y:=a\} & & \downarrow \{x:=b, y:=a\} \\ R(b, a) & & \neg R(b, a) \\ & \text{ff} & \end{array}$$

platí i $\text{ff} \in \text{Res}_1(X')$.

Tudíž podle poznámky 4.2.20 množina X' není splnitelná. Proto není splnitelná ani množina X . Dokázali jsme, že sémantický důsledek $\exists y.\forall x.R(x, y) \models \forall x.\exists y.R(x, y)$ platí.

Celkově jsme tedy dokázali, že sémantická ekvivalence $\forall x.\exists y.R(x, y) \models \exists y.\forall x.R(x, y)$ *neplatí*.

4.3 Cvičení

4.3.1 Cvičení Ať $a, b, c, d, e, f \in At$. Resolučním algoritmem rozhodněte, zda množina formulí výrokové logiky $\{a \vee c \vee f, a \vee \neg b, b \vee \neg d \vee e \vee \neg f, b \vee c \vee \neg e, e \vee f\}$ je splnitelná.

4.3.2 Cvičení Ať $a, b, c \in At$. Resolučním algoritmem rozhodněte, zda platí $\{a \Rightarrow b, b \Rightarrow c\} \models a \Rightarrow c$.

4.3.3 Cvičení Popište jazyk \mathcal{L} predikátové logiky, ve kterém je řetězec $\forall x.(P(x, a) \Leftrightarrow \exists x.C(x))$ sentencí predikátové logiky. Nalezněte její CNF.

4.3.4 Cvičení Popište nějaký jazyk \mathcal{L} predikátové logiky, ve kterém jsou zadané řetězce atomickými formulemi a poté hledejte jejich maximální unifikátor.

1. $P(x, y, z(y))$ a $P(y, z(a), x)$.
2. $Q(u, v, f(u, v))$ a $Q(v, v, w)$.

4.3.5 Cvičení Resolučním algoritmem rozhodněte, zda platí

$$\{\forall x.R(x, x), \forall x.\forall y.(R(x, y) \Rightarrow R(y, x))\} \models \forall x.\exists y.R(x, y)$$

kde všechny tři řetězce jsou sentencemi jazyka predikátové logiky. Tento jazyk popište.

Revize kapitoly

Dozvěděli jsme se:

- ✓ Sémantickou úlohu o platnosti $M \models \varphi$ lze řešit *syntakticky*, jak ve výrokové, tak v predikátové logice.

Pro přípravu na zkoušku zkuste zodpovědět následující otázky:

- ✓ Navrhněte algoritmus, který rozhoduje o platnosti $\alpha \models \beta$ jak ve výrokové, tak v predikátové logice. Dokažte korektnost tohoto algoritmu.

Doplňující literatura

Další informace o resolučních algoritmech najdete ve standardním odkazu

☞ M. Demlová a B. Pondělíček, *Matematická logika*, FEL ČVUT, Praha 1997

Resoluční algoritmy jsou ideálním nástrojem pro jazyky z oblasti *logického programování*, jakým je například PROLOG. Pro podrobnosti o jazyce PROLOG lze doporučit knihu

☞ W. F. Clocksin a C. S. Mellish, *Programming in Prolog*, Springer, Berlin, 1987 (3. vydání)

dobrým úvodem do logického programování je kniha

☞ C. J. Hogger, *Essentials of Logic Programming*, Clarendon Press, Oxford, 1990

Kapitola 5

Formalizace českých vět

Léta politického tréninku a zkušeností naučila Hackera použít dvacet slov tam, kde by stačilo jedno, diktovat milióny slov tam, kde by stačily pouhé tisíce, a používat jazyk k zamlžení a překrucování faktů a událostí tak, aby se ostatním staly naprosto nepochopitelnými.

Nepochopitelnost může být pro některé politiky rájem, protože v ní spočívá dočasné bezpečí.

Jonathan Lynn a Antony Jay, *Ano, pane ministře*

V této kapitole předvedeme, jak řešit otázky *formalizace* českých vět ve výrokové nebo predikátové logice. Otázka adekvátní formalizace je obecně poměrně těžká: formální logika pochopitelně nemůže vystihnout přesně všechny jemné odstíny přirozeného jazyka, viz poznámku 5.2.2.

5.1 Formalizace jednoduchých vět

5.1.1 Příklad Dokažte (formalizací ve výrokové logice), že následující dvě české věty

Levné jídlo není dobré. Dobré jídlo není levné.

znamenaají totéž.

Máme přikázáno, že máme obě věty formalizovat ve výrokové logice. Udělejme si plán:

1. Vyjadřovací sílu výrokové logiky volíme volbou množiny A atomických formulí. Musíme tedy v obou větách vyhledat „dále nedělitelné“ výroky, které zformalizujeme jako atomické formule.
2. Volba množiny atomických formulí okamžitě rozjede syntaxi příslušné výrokové logiky. Obě české věty musí být zformalizovány jako formule (nazvěme je α a β) výrokové logiky. V obou českých větách tedy musíme nalézt způsob, jakým jsou obě věty z „dále nedělitelných“ částí pospojovány.
3. Úloha zněla, zda obě české věty znamenají totéž: ve výrokové logice jde o otázku, zda platí sémantická ekvivalence $\alpha \models \beta$. Takový problém umíme vyřešit prohlížením pravdivostních tabulek (viz příklad 2.1.9).
4. Opět se vrátíme do světa českých vět a na zadanou otázku odpovíme podle toho, co nám vyšlo ve formálním světě výrokové logiky.

Jednotlivé úkoly nyní vyřešíme:

1. Dále nedělitelné jsou dva výroky: *Jídlo je levné.* a *Jídlo je dobré.* Zvolíme tedy následující volbu atomických formulí:

výroková logika	české věty
at. formule: L	Jídlo je levné.
at. formule: D	Jídlo je dobré.

2. První větu můžeme pomocí „dále nedělitelných“ výroků přepsat jako *Jestliže je jídlo levné, pak není pravda, že je jídlo dobré*. Podobně přepíšeme druhou větu: *Jestliže je jídlo dobré, pak není pravda, že je jídlo levné*.

Formalizace se tedy rozrostla na:

výroková logika	české věty
at. formule: L	Jídlo je levné.
at. formule: D	Jídlo je dobré.
formule: $L \Rightarrow \neg D$	Levné jídlo není dobré.
formule: $D \Rightarrow \neg L$	Dobré jídlo není levné.

3. Přidejme na obě strany otázku, zda obě věty znamenají totéž:

výroková logika	české věty
at. formule: L	Jídlo je levné.
at. formule: D	Jídlo je dobré.
formule: $L \Rightarrow \neg D$	Levné jídlo není dobré.
formule: $D \Rightarrow \neg L$	Dobré jídlo není levné.
platí $L \Rightarrow \neg D \models D \Rightarrow \neg L$?	Znamenají obě věty totéž?

a vyřešíme problém sémantické ekvivalence pravdivostní tabulkou:

L	D	$L \Rightarrow \neg D$	$D \Rightarrow \neg L$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Inspekci tabulky zjišťujeme, že $L \Rightarrow \neg D \models D \Rightarrow \neg L$ platí.

4. Vrátime se nyní do světa českých vět a tvrdíme, že (při uvedené formalizaci ve výrokové logice) věty *Levné jídlo není dobré*. a *Dobré jídlo není levné*. znamenají totéž.

Samozřejmě: na otázku, zda obě české věty *skutečně* znamenají totéž, nemůže sama formální logika odpovědět. Obě věty mají své *psychologické konotace*: první věta má konotaci spíše negativní, druhá jednoznačně pozitivní.

5.1.2 Příklad Zformalizujte v predikátové logice větu

Ne každý, kdo hraje na housle, je Sherlock Holmes.

Opět si uděláme plán. Máme formalizovat v predikátové logice, musíme tedy postupovat následovně:

1. Zjistíme, o jakých objektech se ve větě mluví. Tím popíšeme universum.
2. Ve větě nalezneme všechny „atomické vlastnosti“. Tyto vlastnosti budou predikáty. Je-li ve větě použito sloveso *být* ve smyslu identifikace (jeden objekt *je* druhý objekt), použijeme na jeho formalizaci rovnost.
3. Ve větě nalezneme všechny objekty, které jsou nějakým způsobem vytvořené z jiných objektů. Každý takový způsob vytváření bude jeden funkční symbol našeho jazyka.
4. Poté, co deklarujeme jazyk \mathcal{L} výše uvedeným způsobem, sepíšeme v jazyce \mathcal{L} sentenci, která se pak automaticky přeloží jako zadaná věta.

Jednotlivé úkoly nyní vyřešíme:

1. Ve větě se mluví o literárních postavách. Universum interpretace tedy bude neprázdná množina všech literárních postav.¹

Zatím tedy máme:

jazyk \mathcal{L} predikátové logiky	české věty
	neprázdná množina všech U literárních postav

2. Atomické vlastnosti rozeznáváme dvě: *hrát na housle* a *být Sherlock Holmes*. První větu budeme formalizovat predikátem H arity 1. Sloveso *být* je ve vlastnosti *být Sherlock Holmes* použito ve smyslu identifikačním: říkáme, že literární postava *je totožná* s literární postavou Sherlocka Holmese.

Formalizace se rozrůstá na:

jazyk \mathcal{L} predikátové logiky	české věty
	neprázdná množina všech U literárních postav
predikáty: H arity 1	množina literárních postav, které hrají na housle

3. Ve větě nalézáme jen jeden objekt, který je vytvořen z (nulového počtu) jiných objektů, jde o literární postavu *Sherlock Holmes*. Tento objekt budeme formalizovat funkčním symbolem h arity 0.

Další snímek formalizace je:

jazyk \mathcal{L} predikátové logiky	české věty
	neprázdná množina U všech literárních postav
predikáty: H arity 1	množina literárních postav, které hrají na housle
funkční symboly: h arity 0	literární postava Sherlock Holmes

4. Nyní sestavíme sentenci v jazyce \mathcal{L} . Hledaná sentence je $\neg\forall x.(H(x) \Rightarrow x = h)$. Abychom tuto sentenci mohli napsat, musíme deklarovat x jako standardní proměnnou.

Celkově máme:

jazyk \mathcal{L} predikátové logiky	české věty
	neprázdná množina U všech literárních postav
st. proměnné: x	
predikáty: H arity 1	množina literárních postav, které hrají na housle
funkční symboly: h arity 0	literární postava Sherlock Holmes
sentence: $\neg\forall x.(H(x) \Rightarrow x = h)$	Ne každý, kdo hraje na housle, je Sherlock Holmes.

5.1.3 Příklad Zformalizujte v predikátové logice úsudek²

1. *Všichni vrahové jsou šílení.*
2. *Pan Hyde je vrah.*
3. *Doktor Jekyll je pan Hyde.*

Tedy:

4. *Doktor Jekyll je šílený.*

¹Zde se naplno projevují další obtíže při formalizaci: mluví se zde skutečně *jen* o literárních postavách? Nebo se mluví o literárních postavách a žijících lidech? Ostatně: co znamená *množina všech lidí*? Zahrnuje všechny lidi žijící v tomto okamžiku? Nebo všechny lidi, kteří kdy na zemi žijí a žili? Viz poznámku 3.1.31. Podobnou potíž ovšem máme i s množinou *všech* literárních postav.

²Podle známé knihy z roku 1886 *Podivný případ dr. Jekylla a pana Hyda* skotského spisovatele Roberta Louise Stevensona (1850–1894). Stevenson se pravděpodobně inspiroval dvojím životem přes den ctihodného radního města Edinburgh, Williama Deacon Brodieho (1741–1788), který po nocích loupil, aby mohl platit své dluhy z hazardních her.

Postupujeme podobně jako v příkladu 5.1.2, musíme ovšem zformalizovat v jednom jazyce *všechny čtyři* české věty. Hledaná formalizace je (všimněte si dvojího použití slovesa *být*: v první, druhé a čtvrté větě jako predikát, ve třetí větě jako identifikace):

jazyk \mathcal{L} predikátové logiky	české věty
	neprázdná množina U všech literárních postav
st. proměnné: x	
predikáty: V arity 1	množina literárních postav, které jsou vrahy
predikáty: S arity 1	množina literárních postav, které jsou šílené
funkční symboly: h arity 0	literární postava pan Hyde
funkční symboly: j arity 0	literární postava doktor Jekyll
sentence: $\forall x.(V(x) \Rightarrow S(x))$	Všichni vražedci jsou šílení.
sentence: $V(h)$	Pan Hyde je vrah.
sentence: $j = h$	Doktor Jekyll je pan Hyde.
sentence: $S(j)$	Doktor Jekyll je šílený.

Úloha zněla zformalizovat úsudek. Víme, že to se děje pomocí sémantického důsledku. V našem případě jde o sémantický důsledek

$$\{\forall x.(V(x) \Rightarrow S(x)), V(h), j = h\} \models S(j)$$

v námi deklarovaném jazyce predikátové logiky.

5.2 Obtížnost formalizací



5.2.1 Příklad Zformalizujte v predikátové logice českou větu:

*Ulrich je muž bez vlastností.*³

Pokud budeme postupovat podobně jako v minulých příkladech, nabízí se následující formalizace (všimněte si použití slovesa *být*):

jazyk \mathcal{L} predikátové logiky	české věty
	neprázdná množina U všech literárních postav
predikáty: V arity 1	množina literárních postav, které nemají vlastnosti
funkční symboly: u arity 0	literární postava Ulrich
sentence: $V(u)$	Ulrich je muž bez vlastností.

Zdánlivě je vše v pořádku, vzniká však problém: *nemít žádnou vlastnost je přeci vlastnost literárních postav!* Ulrich tedy nějakou vlastnost přeci jen má. Co to znamená? Narážíme tu na *další úskalí* formalizace v predikátové logice — v přirozeném používáme slovo *vlastnost* volněji, než v jazyce predikátové logiky. Zadanou větu tedy v jazyce predikátové logiky formalizovat *nemůžeme*.



5.2.2 Poznámka Shrňme všechny problémy formalizace českých vět, na které jsme v této kapitole narazili:

1. V přirozeném jazyce má většina vět *psychologické konotace*, které naše formalizace nemusí dobře vystihnout.

Navíc v přirozeném jazyce ne vždycky používáme *standardní čtení* logických spojek a kvantifikátorů

\neg	Není pravda, že ...	\forall	Pro všechna ...
\wedge	... a současně ...	\exists	Existuje ...
\vee	... nebo ...		
\Rightarrow	Jestliže ..., potom ...		
\Leftrightarrow	... právě tehdy, když ...		

³Ulrich je hlavní postavou románu *Muž bez vlastností* rakouského spisovatele Roberta Musila (1880–1942).

Proto je vhodné ve větách přirozeného jazyka nejprve všechny případné výskyty logických spojek a kvantifikátorů přepsat pomocí těchto standardních čtení. To ovšem vyžaduje obrovský jazykový cit: porovnejme například stavbu vět *Nikdo není dokonalý.* a *Nobody is perfect.*

Formalizace vět přirozeného jazyka je tak typickým příkladem „cesty tam a zase zpátky“. Při obou cestách můžeme něco ztratit a/nebo něco získat a nemusíme se nutně vrátit stejní, jací jsme vyšli.

Pokud formalizací řešíme nějaký problém přirozeného jazyka, měli bychom při odpovědi i přesně uvést, jak tato formalizace vypadá.

2. Při formalizaci je nutné se rozhodnout, *jakou* z možných logik pro formalizaci použijeme. V tomto textu jsme uvedli příklady dvou logik: výrokové a predikátové. I když predikátová logika podstatně zobecňuje výrokovou logiku, nemusí ani predikátová logika pro adekvátní formalizaci stačit, viz příklad 5.2.1.

Predikátovou logikou spektrum formálních logik samozřejmě nekončí. Zvláště v computer science je používání dalších logik velmi obvyklé. Zmíníme alespoň některé možné další „neklasické“ logiky:

- (a) *Intuicionistická logika.* Velmi zjednodušeně řečeno jde o logiku, ve které je porušena klasická sémantika. V intuicionistické logice *neplatí* zákon vyloučeného třetího, viz větu 2.1.12. Neplatí tak automaticky, že formule tvaru $\alpha \vee \neg\alpha$ je pravdivá. Příkladem je věta:

V decimálním rozvoji čísla π je 197 cifer 7 za sebou nebo ne.

V klasické logice je tato věta pravdivá, v intuicionistické logice musíme vědět, zda je v decimálním rozvoji π skutečně 197 sedmiček za sebou nebo ne.

- (b) *Kvantová logika.* Jde o logiku, používanou k popisu situací na kvantové úrovni. V kvantovém světě obecně neplatí některé zákonitosti klasické logiky, například sémantický distributivní zákon spojek \wedge a \vee (viz větu 2.1.12). Pravdivostní hodnoty formulí kvantové logiky musí tvořit matematickou strukturu zvanou *ortomodulární svaz*.
- (c) *Modální logika.* Klasická modální logika se věnuje následujícím dvěma modalitám vět

Je nutné, že ... Je možné, že ...

Syntaxe standardní modální logiky je obohacena o dva symboly \square a \diamond . Tato změna syntaxe vyžaduje drastickou změnu sémantiky: jedná se o takzvanou *Kripkeho sémantiku možných světů*.

V computer science se běžně využívá rozšíření standardní modální logiky o další modalities, které popisují chování (nedeterministických) systémů ve smyslu výpočetních kroků. Jde o modalities:

Po každém provedení výpočetního kroku platí ...
Po nějakém provedení výpočetního kroku platí ...

Sémantika takové modální logiky je opět sémantikou možných světů.

- (d) *Deontická logika.* Tato logika je logikou *normativních* tvrzení: formalizuje větné konstrukce jako

Je správné, že ... Je morální, že ...

a podobně. Sémantika deontických logik je opět sémantikou možných světů, je však obecně složitější, než u standardní modální logiky.

- (e) *Temporální a dynamická logika.* V této logice se studuje pravdivost či nepravdivost formulí v čase. Opět se jedná o variantu modální logiky, studují se zde konstrukce tvaru

Jednou bude platit ... Platí ..., dokud platí, že ...

a podobně. Je vidět, že dynamická a temporální logika je ideálním nástrojem pro analýzu tvrzení o běhu algoritmů.

- (f) *Logika vyššího řádu*. V klasické predikátové logice jsme povolili pouze kvantifikování objektů, nikoli predikátů. Logikám, kde se povoluje kvantifikace predikátů (a vlastností predikátů, vlastností těchto vlastností apod.) se říká logika vyšších řádů. V computer science se tyto logiky využívají například při analýze *polymorfismu* v programování.

Poznamenejme ještě, že predikátové logice, zavedené v tomto textu, se také říká *logika prvního řádu*.

Úvodem do neklasických logik a jejich použití v computer science se zabývá například text

- ☞ J. Velebil, *Neklasické logiky*, <http://math.feld.cvut.cz/velebil/>, text bude dostupný počátkem roku 2008

5.3 Cvičení

5.3.1 Cvičení Zformalizujte (každou buď ve výrokové nebo v predikátové logice) následující české věty:

1. Kdo jinému jámu kopá, sám do ní padá.
2. Nebude-li přšet, nezmoknem.
3. Kdykoli někdo hraje na piáno, tak sousedův pes teskně vyje.
4. Nikdo není dokonalý.
5. Adéla ještě nevečeřela.
6. Rimmer se velmi opatrně vydal přes trávník směrem ke stroji času, následován Kennedym, Van Goghem, Einsteinem a Césarem. Elvis si nacpal steak do pusy, druhý si nacpal do kapsy, popadl čtyři rohlíky a následoval je.⁴
7. Součin druhých mocnin libovolných přirozených čísel je druhá mocnina nějakého přirozeného čísla.
8. O čem nelze mluvit, o tom se musí mlčet.⁵

U každé formalizace uveďte důvod, proč považujete použitou logiku za adekvátní.

Revize kapitoly

Dozvěděli jsme se:

- ✓ Některé české věty můžeme ve výrokové nebo v predikátové logice *formalizovat*. Formalizace *nemusí* přesně odrazet všechny psychologické aspekty přirozeného jazyka.

Pro přípravu na zkoušku zkuste zodpovědět následující otázky:

- ✓ Promyslete příklad české věty, kterou nelze adekvátně zformalizovat ve výrokové logice. Vysvětlete proč.
- ✓ Naučte se formalizovat české věty typu A, E, I, O ze cvičení **3.3.3**.

Doplňující literatura

Do větší hloubky je formalizace ve výrokové a predikátové logice probrána ve skriptu

- ☞ M. Demlová a B. Pondělíček, *Matematická logika*, FEL ČVUT, Praha 1997

kteří jsme několikrát zmínili. Toto skriptum doporučujeme i jako zdroj dalších příkladů.

Některé problémy logiky, filosofie a přirozeného jazyka jsou řešeny v knize

- ☞ P. Kolář, *Argumenty filosofické logiky*, Filosofia, Praha 1999

⁴Z knihy *Better Than Life* Roba Granta a Douga Naylora.

⁵Závěrečná věta knihy *Tractatus logico-philosophicus* rakouského filosofa Leopolda Wittgensteina (1889–1951). Připomínáme, že větu máte zformalizovat buď ve výrokové nebo v predikátové logice.

Příloha A

Naivní teorie množin

Georg Cantor stvořil ráj, z něhož matematiky nikdo nevyžene.

David Hilbert

V této kapitole se seznámíme s *naivní teorií množin*. Tuto teorii pravděpodobně všichni používáme při jednoduché matematické práci. Naivní teorie množin má však svá omezení, proto je nutné při studiu množin používat nějakou axiomatickou teorii množin. O omezeních naivní teorie promluvíme v odstavci [A.2](#), do axiomatických teorií se v tomto textu pouštět nebudeme, odkazujeme na seznam doplňující literatury na konci kapitoly.

A.1 Pojem množiny

A.1.1 Definice *Množinou* A rozumíme souhrn určitých a rozlišitelných objektů x existujících v naší mysli. Těmto objektům říkáme *prvky množiny* A .

Definici [A.1.1](#) jsme přejali doslovně z prvního odstavce díla *Příspěvky k základům teorie transfinitních čísel* německého matematika Georga Cantora.¹ Po svém zavedení slavila naivní teorie množin relativní úspěchy, často však narážela na nepochopení a odpor některých světových matematiků. Teorie množin totiž otevřela matematikům možnost studovat objekty, jejichž existence se často vymyká intuitivnímu chápání světa.

V tomto odstavci připomeneme jen některá základní značení a pojmy naivní teorie.



A.1.2 Poznámka Připomeňme dva nejrozšířenější zápisy množiny:

1. Výčtem prvků: to můžeme udělat pouze v případech, kdy buď vypíšeme celou množinu nebo kdy jsou prvky množiny „zřejmé“.

Příklad: množina $\{a, x, v\}$ má prvky a, x, v .

Jiný příklad: množina přirozených čísel $\{0, 1, 2, 3, \dots\}$ Předpokládáme tu „nezákeřnost“, tj. tři tečky jsou konvencí tvrdící „dál je to jasné, žádný podraz typu $\sqrt{3}$ mezi prvky množiny nečekejte“.

Připomeňme, že definice [A.1.1](#) nezakazuje, aby prvky množiny byly opět množiny, takže $\{0, \mathbb{N}, a\}$ je množina mající za prvky přirozené číslo 0, množinu přirozených čísel \mathbb{N} a symbol a . Stejně tak můžeme například napsat množinu $\{9, \{89\}\}$, jejímiž prvky jsou přirozené číslo 9 a množina $\{89\}$.

2. Charakteristickou vlastností: typický zápis je $\{x \mid x \text{ má vlastnost } V\}$, což čteme takto: jde o množinu těch prvků x , které mají vlastnost V .

Příklad: $\{x \mid x \text{ je celé číslo dělitelné dvěma}\}$.

¹Georg Ferdinand Ludwig Philipp Cantor se narodil v Petrohradě v roce 1845. Poté, co se jeho rodina přestěhovala zpět do Německa, studoval Cantor nejprve v Zurichu a poté v Berlíně u Karla Weierstrasse. Své největší objevy — teorii množin a teorii mohutností množin — Cantor uskutečnil na popud matematika Heinricha Heineho na universitě v Halle, kde Cantor působil až do konce své vědecké kariéry. Cantor zemřel 6. ledna 1918, sužován těžkou duševní chorobou.

Dále připomeňme, že značením $x \in A$ vyjadřujeme fakt, že x je prvkem množiny A . Fakt, že x není prvkem množiny A , zapisujeme $x \notin A$.

Podle definice A.1.1 je množina svými prvky určena jednoznačně. Z toho okamžitě plyne důležitý *princip extensionality*:

Pro množiny A, B platí rovnost $A = B$ právě tehdy, když současně platí:

1. Každý prvek množiny A je prvkem množiny B .
2. Každý prvek množiny B je prvkem množiny A .

Zavedeme-li značení $A \subseteq B$ pro *inklusi* množiny A v množině B , tj. pro tvrzení: každý prvek množiny A je prvkem množiny B , můžeme princip extensionality přepsat do nám dobře známého tvaru:

Pro množiny A, B platí rovnost $A = B$ právě tehdy, když platí $A \subseteq B$ a současně $B \subseteq A$.



A.1.3 Poznámka Základní způsoby, jakými lze z množin tvořit další množiny, jsou tyto:

1. *Prázdná množina* je množina \emptyset , kde $\emptyset = \{x \mid x \neq x\}$.
Tato definice vyžaduje porozumění pojmu rovnosti. Pro žádné x neplatí $x \neq x$. Proto charakteristickou vlastností prázdné množiny žádné x nemá. To znamená, že *prázdná množina nemá žádné prvky*.
2. *Průnik množin A a B* je množina $A \cap B$, kde $A \cap B = \{x \mid x \in A \text{ a současně } x \in B\}$.
3. *Sjednocení množin A a B* je množina $A \cup B$, kde $A \cup B = \{x \mid x \in A \text{ nebo } x \in B\}$.
4. *Rozdíl množin A a B* je množina $A \setminus B$, kde $A \setminus B = \{x \mid x \in A \text{ a současně } x \notin B\}$.
5. *Kartézský součin množin A a B* je množina $A \times B$, kde $A \times B = \{(x, y) \mid x \in A \text{ a současně } y \in B\}$.
Symbolem (x, y) tu rozumíme *uspořádanou dvojici*, tj. rozlišujeme pořadí první a druhé položky.
6. *Potenční množina množiny A* je množina $\exp A$, kde $\exp A = \{M \mid M \subseteq A\}$.

S touto základní výbavou lze již o (naivních) množinách dokázat poměrně hodně důležitých faktů. Dělat to nebudeme, odkazujeme na skriptum

☞ M. Demlová a B. Pondělíček, *Matematická logika*, skriptum FEL ČVUT, Praha, 1997

A.2 Paradox pana Bertranda Russella

Naivní představa množiny jakožto pytle, obsahujícího navzájem rozlišitelné prvky, však zavedla matematiku na okraj propasti a tam do ní strčila: naivní teorie množin totiž není konsistentní!

Má-li být jakákoli teorie základem matematiky, pak jistě neočekáváme, že by v ní byl spor. Takový spor však v naivní teorii množin je: dáme-li všechny množiny do jednoho pytle, pak jde (podle naší naivní definice) opět o množinu. A to není možné: tento výsledek nese název *Russellův paradox*.²

A.2.1 Věta (Russell) *Množina všech množin neexistuje.*

DŮKAZ. Budeme postupovat sporem: předpokládejme, že máme množinu všech množin. Označíme ji U . Protože (v naivní teorii) je U souhrn nějakých rozlišitelných věcí, můžeme z množiny U vzít jakoukoli část a bude to opět množina. Takže

$$R = \{M \in U \mid M \notin M\}$$

je množina. Musí nastat právě jedna ze dvou následujících možností:

²Bertrand Russell (1872–1970) se věnoval nejrůznějším oborům: od matematiky přes ekonomii až po etiku. Svým třísvazkovým dílem *Principia Mathematica*, které napsal spolu s Alfredem Northem Whiteheadem, dovršil základy tvorby formální logiky.

1. $R \in R$. V tomto případě musí R mít charakteristickou vlastnost prvků množiny R a musí tedy platit $R \notin R$. To je spor.
2. $R \notin R$. V tomto případě má R charakteristickou vlastnost prvků množiny R a musí tedy platit $R \in R$. To je spor.

Množina R tedy nemůže existovat. Takže nemůže existovat ani množina U . ■

Tento výsledek ukázal, že pro skutečnou práci s množinami je definice A.1.1 neudržitelná. Existence Russellova paradoxu vyústila v nejrůznější *axiomatické* teorie množin, viz seznam doplňující literatury.

A.3 Cvičení

A.3.1 Cvičení Dokažte, že $\emptyset \subseteq A$ platí pro každou množinu A .

A.3.2 Cvičení Dokažte, že rovnost $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ platí pro libovolné množiny A , B a C .

A.3.3 Cvičení Vypište všechny prvky množiny $\exp\{a, b, c\}$. Mělo by jich být 8.

Revize kapitoly

Dozvěděli jsme se:

- ✓ Naivní pojem množiny dostačuje pro základní práci s množinami. Pro studium teorie množin je však naivní pojem množiny naprosto špatný.

Pro přípravu na zkoušku zkuste zodpovědět následující otázky:

- ✓ Vysvětlíte, proč Russellův paradox brání systematicky prohledávat možné interpretace jazyka predikátové logiky.

Doplňující literatura

Naivní teorie množin je do větší hloubky popsána ve skriptu

☞ M. Demlová a B. Pondělíček, *Matematická logika*, skriptum FEL ČVUT, Praha, 1997

Cantorův text o množinách, doplněný komentářem Stephena Hawkinga, naleznete ve sbírce

☞ S. Hawking, *God Created the Integers (The Mathematical Breakthroughs that Changed History)*, Penguin Books, Londýn, 2006

kde je komentováno a přeloženo celkem 17 matematických prací, které změnily dějiny vědy: od Eukleida až po Alana Mathisona Turinga.

Axiomatiku teorie množin, takzvanou *Zermelovu-Fraenkelovu*, studují například knihy

☞ B. Balcar a P. Štěpánek, *Teorie množin*, Academia, Praha, 2001 (2. vydání)

☞ T. Jech, *Set Theory*, Springer, New York, 2006 (4. vydání)

Teorie množin, která má velký význam při studiu nekonečných rekursivních procesů v computer science, takzvaná *non-wellfounded set theory*, je vyložena v knize

☞ J. Barwise a L. Moss, *Vicious Circles*, CSLI Publications, Stanford University, 1996, dostupné online na <http://standish.stanford.edu>

Příloha B

Mohutnosti

Tři minuty přemýšlení stačí k tomu, aby na to člověk přišel. Ale myšlení je nudné a tři minuty jsou dlouhá doba.

A. E. Houseman

V tomto dodatku zavedeme důležité pojmy běžně využívané v celé moderní matematice. Zhruba řečeno, jde o pojmy týkající se počtu prvků množin. Přesto, a to dělá celou teorii velmi užitečnou, se budeme snažit po celou dobu této kapitoly zapomenout, že umíme počítat. Základními pojmy totiž nebudou čísla (jakožto počet prvků), ale porovnávání: tady je více prvků než jinde, tady je stejně prvků jako jinde, apod.

Nestudujeme teorii množin, všechny množiny v této kapitole jsou chápány naivně ve smyslu kapitoly A.

B.1 Funkce jako nálepkovací schéma

B.1.1 Definice Řekneme, že f je funkce z množiny A do množiny B , pokud platí $f \subseteq A \times B$ a současně pro každé $x \in A$ existuje právě jedno $y \in B$ tak, že $(x, y) \in f$. Toto jednoznačně určené y značíme $f(x)$. Fakt, že f je funkce z A do B , značíme $f : A \rightarrow B$.



B.1.2 Poznámka Definice funkce je poměrně komplikovaná (a možná vypadá nezvykle), proto ji podrobně rozebereme:

Fakt, že $f \subseteq A \times B$ znamená, že funkce je množina (seznam) uspořádaných dvojic: první položka každé dvojice musí být prvek v A , druhá položka každé dvojice musí být prvek v B .

Navíc musí tento seznam f uspořádaných dvojic splňovat tuto dodatečnou podmínku: každý prvek A je první položkou, a je první položkou pro jediný prvek z B (sice pro příslušnou funkční hodnotu).

Proč jsme nepoužili způsob, jakým se obvykle funkce definují v analýze? To jest: proč *nemluvíme o funkčním předpisu*? Především: zadat takový předpis není obvykle možné. Nepracujeme tu totiž s reálnými funkcemi, ale s funkcemi mezi abstraktními množinami. V analýze se udržela terminologie 19. století, kdy se představa o tom, co je a co není funkce, začínala formovat. Vybavte si pojmy jako *průběh funkce*, *funkce nabývá maxima*, a podobné. Používání těchto pojmů reflektovalo fakt, že na funkci se z počátku nahlíželo *dynamicky*, funkce měla svůj vývoj (vždyť počítáme funkční hodnoty). Pak ovšem přišel tvůrce teorie množin Georg Cantor a vše bylo jinak. Pochopil, že vše, co je pro pojem funkce podstatné, je právě vyjádření faktu, že jde o speciální množinu uspořádaných dvojic. Funkce od dob Cantorových je tedy chápána přesně tak, jak jsme to vyjádřili v definici B.1.1.

Cantorova definice měla obrovský vliv i na klasickou analýzu: matematici tak mohli začít studovat i funkce podivných vlastností, například tuto funkci $f : \mathbb{R} \rightarrow \mathbb{R}$:

$$f(x) = \begin{cases} \frac{1}{b}, & \text{jestliže } x = \frac{a}{b} \text{ je racionální číslo a } a \text{ a } b \text{ jsou nesoudělná celá čísla} \\ 0, & \text{jestliže } x \text{ je iracionální} \end{cases}$$

která hraje důležitou roli v teorii integrálu.

Ještě poznamenejme, že slova *funkce* a *zobrazení* jsou synonyma.



B.1.3 Poznámka Připomeňme nejrůznější zápisy funkcí:

1. Funkčním předpisem: to je zřejmě nejznámější způsob.

Příklad: $f(x) = \sin x$, $x \in \mathbb{R}$. Také se používá zápis $f : x \mapsto \sin x$, $x \in \mathbb{R}$.

Pozor! Poněkud laxní způsob psaní funkce je zápis $\sin x$. To může vést k nedorozumění, nepoužívejte jej.

2. Výčtem funkčních hodnot: to můžeme udělat, pouze v případech, kdy buď vypíšeme celou funkci nebo kdy jsou funkční hodnoty „zřejmé“.

Příklad: $f(a) = 2$, $f(x) = 4$, $f(v) = 2$ je zápis funkce $f : \{a, x, v\} \rightarrow \{2, 4\}$.

Jiný příklad: $h(0) = 1$, $h(1) = 2$, $h(2) = 3$, ... Zde se zřejmě zapisuje funkce $h : \mathbb{N} \rightarrow \mathbb{N}$, $h(n) = n + 1$, $n \in \mathbb{N}$. Předpokládáme tu „nezákeřnost“, tj. tři tečky jsou konvencí tvrdící „dál je to jasné, žádný podraz typu $h(12\,845) = 17$ nečekejte“.

3. Větvením: při tomto zápisu funkce $f : A \rightarrow B$ nesmíme žádnou část množiny A vynechat.

Příklad: funkce $f : \mathbb{R} \rightarrow \mathbb{R}$ je definována takto:

$$f(x) = \begin{cases} 0, & \text{jestliže } x \in (-\infty, 5) \\ 1, & \text{jestliže } x \in (5, 18) \\ 42, & \text{jestliže } x \in (18, +\infty) \end{cases}$$

Existuje řada jiných způsobů zápisu funkcí (například λ -notace, známá z λ -kalkulu nebo funkcionálního programování), my je v tomto textu nebudeme potřebovat.



B.1.4 Poznámka Budeme funkce používat pro porovnávání „počtu prvků“. Pro tento úkol je vhodná následující představa.

Ať $f : A \rightarrow B$ je funkce. Množinu A si představme jako „množinu nálepek“. Těmito nálepkami budeme „polepovat“ prvky množiny B . Nálepku $x \in A$ nalepíme na prvek $f(x) \in B$. Fakt, že máme funkci jen říká, že každou nálepku někam nalepíme. Nalepování (tj. funkce f) ovšem může mít speciální vlastnosti, například (srovnejte s definicí B.1.5):

1. Na žádný prvek z množiny B nenalepíme dvě různé nálepky. Jinými slovy: jestliže $f(x) = f(x')$, potom $x = x'$, a to pro všechna x, x' z množiny A .

To odpovídá této situaci: představme si, že A je množina šatnových lístků opatřených čísly od 1 do 50 a B je množina kabátů v divadelní šatně. Šatnář(ka) špendlí jednotlivé lístky na kabáty. Když toto špendlení nazveme f (tj. lístek s číslem x je přišpendlen na kabát $f(x)$), mělo by při korektním špendlení platit, že z rovnosti $f(x) = f(x')$ plyne $x = x'$, a to pro všechna x, x' z množiny A . Na žádném kabátu nejsou přišpendleny dva různé lístky.

Co existence takového f vypovídá o počtu kabátů v šatně? Jsou-li spotřebovány všechny šatnové lístky, můžeme říci, že *v šatně je alespoň tolik kabátů, kolik je šatnových lístků*. Nemůžeme totiž vyloučit, že někdo umístil v šatně kabát bez zaplacení.

Později budeme říkat (viz definice B.1.16), že mohutnost množiny A je nanejvýš rovna mohutnosti množiny B .

2. Abychom měli jistotu, že v šatně je stejně kabátů jako rozdaných lístků, musí funkce f splňovat navíc následující podmínku: každý prvek z množiny B má nějakou nálepku. To znamená, že pro každé $y \in B$ existuje $x \in A$ tak, že $y = f(x)$.

Existence f splňující obě podmínky pak zaručí, že množiny A a B mají stejnou mohutnost, viz definice B.1.6.

Existence zobrazení $f : A \rightarrow B$ splňující určité vlastnosti tedy může vypovídat o porovnání počtu prvků množin A a B , *aniž bychom museli umět tyto prvky spočítat*. Všimněme si totiž, že ve výše uvedeném případě není nutné umět počítat od 1 do 50.

B.1.5 Definice Řekneme, že funkce $f : A \rightarrow B$ je:

1. *injektivní* (také se říká *injekce* nebo *prostá funkce*), když platí: jestliže $f(x) = f(x')$, potom $x = x'$, a to pro všechna x, x' z množiny A .
2. *surjektivní* (také se říká *surjekce* nebo *funkce na*), když platí: pro každé y z množiny B existuje x z množiny A tak, že platí $y = f(x)$.
3. *bijektivní* (také se říká *bijekce*), když f je surjektivní a injektivní současně.

B.1.6 Definice Ať A a B jsou dvě množiny. Řekneme, že *množiny A a B mají stejnou mohutnost* (také se říká *stejnou kardinalitu*), když existuje bijekce $f : A \rightarrow B$. Tento fakt značíme $\text{card } A = \text{card } B$.

B.1.7 Příklad Množiny $\{a, b, c\}$ a $\{1, 2, 3\}$ mají stejnou mohutnost. Svědkem je například funkce $f : \{a, b, c\} \rightarrow \{1, 2, 3\}$, kde $f(a) = 1$, $f(b) = 2$ a $f(c) = 3$. Tato funkce je bijekce.

B.1.8 Příklad Označme jako \mathbb{S} množinu všech sudých přirozených čísel. Ať $f : \mathbb{N} \rightarrow \mathbb{S}$ je funkce zadaná přepisem $f(n) = 2n$, $n \in \mathbb{N}$. Ukážeme, že f je bijektivní.

1. f je injektivní: ať $f(n) = f(n')$. Potom $2n = 2n'$, a tedy $n = n'$, pro libovolná n a n' .
2. f je surjektivní: ať s je sudé přirozené číslo. Potom existuje přirozené číslo n tak, že $s = 2n$. To ale znamená, že $s = f(n)$.

Ukázali jsme, že množiny \mathbb{S} a \mathbb{N} mají stejnou mohutnost.¹

B.1.9 Příklad Ukážeme, že libovolné dva neprázdné otevřené intervaly (a, b) a (c, d) mají stejnou mohutnost. Stačí sestavit bijekci $f : (a, b) \rightarrow (c, d)$. Takovou je například (část) lineární funkce

$$f(x) = \frac{d-c}{b-a} \cdot (x-a) + c, \quad x \in (a, b)$$

Protože f je rostoucí spojitá funkce a $\lim_{x \rightarrow a^+} f(x) = c$ a $\lim_{x \rightarrow b^-} f(x) = d$, je $f : (a, b) \rightarrow (c, d)$ bijekce. To jsme chtěli ukázat.

B.1.10 Příklad Funkce $\tan : (-\frac{\pi}{2}, \frac{\pi}{2}) \rightarrow \mathbb{R}$ je bijekce, proto mají množiny $(-\frac{\pi}{2}, \frac{\pi}{2})$ a \mathbb{R} stejnou mohutnost.

B.1.11 Tvrzení *Identická funkce na jakékoli množině je bijekce. Složení bijekcí je bijekce.*

DŮKAZ. Ať A je jakákoli množina. Identická funkce $\text{id}_A : A \rightarrow A$ je definována takto: $\text{id}_A(x) = x$, $x \in A$. Je to triviálně bijekce.

Ať A, B a C jsou libovolné množiny a ať $f : A \rightarrow B$ a $g : B \rightarrow C$ jsou bijekce. Máme ukázat, že $g \circ f : A \rightarrow C$ je bijekce.

To je zřejmé: jestliže $g(f(x)) = g(f(x'))$, potom $f(x) = f(x')$ (protože g je injekce), a tedy $x = x'$ (protože f je injekce), a to pro libovolná x a x' z množiny A . Ukázali jsme, že $g \circ f$ je injekce.

Dále: pro libovolné $y \in C$ existuje $z \in B$ tak, že $y = g(z)$ (protože g je surjekce) a pro toto z existuje $x \in A$ tak, že $f(x) = z$ (protože f je surjekce). Celkově $y = g(f(x))$. Dokázali jsme, že $g \circ f$ je surjekce. ■

B.1.12 Důsledek *Pro jakoukoli množinu A platí $\text{card } A = \text{card } A$. Pro jakékoli množiny A, B a C platí: jestliže $\text{card } A = \text{card } B$ a současně $\text{card } B = \text{card } C$, potom $\text{card } A = \text{card } C$.*

¹Ukázali jsme právě, že sudých čísel „je stejně“ jako všech přirozených čísel. Podivné, ale pravdivé.

B.1.13 Příklad Pro libovolný neprázdný otevřený interval (a, b) platí $\text{card}(a, b) = \text{card } \mathbb{R}$.

Stačí použít fakt, že $\text{card}(a, b) = \text{card}(-\frac{\pi}{2}, \frac{\pi}{2})$ (viz příklad B.1.9) a $\text{card}(-\frac{\pi}{2}, \frac{\pi}{2}) = \text{card } \mathbb{R}$ (viz příklad B.1.10). Pak použijeme důsledek B.1.12.

B.1.14 Tvrzení Pro funkci $f : A \rightarrow B$ jsou následující dvě podmínky ekvivalentní:

1. f je bijekce.
2. Existuje jediná funkce $g : B \rightarrow A$ tak, že platí $g \circ f = \text{id}_A$ a $f \circ g = \text{id}_B$ současně.

Navíc funkce g z bodu 2. je bijekce.

DŮKAZ. Z 1. plyne 2.: Definujme $g = \{(y, x) \in B \times A \mid (x, y) \in f\}$.

Právě definované g je skutečně funkce: pro libovolné $y \in B$ existuje právě jedno $x \in A$ tak, že $x = g(y)$. To plyne z faktu, že f je bijekce.

Definované g je tedy funkce a rovnost $y = f(x)$ platí právě tehdy, když $g(y) = x$.

Z těchto pozorování plynou okamžitě rovnosti $g \circ f = \text{id}_A$ a $f \circ g = \text{id}_B$.

Ze 2. plyne 1.: chceme ukázat, že f je bijektivní.

1. f je injektivní: ať $f(x) = f(x')$. Potom $x = g(f(x)) = g(f(x')) = x'$.
2. f je surjektivní: ať $y \in B$. Definujte $x = g(y)$. Pak platí $f(x) = f(g(y)) = y$.

Dokázali jsme, že f je bijektivní.

Fakt, že g je bijekce plyne okamžitě z rovností $g \circ f = \text{id}_A$ a $f \circ g = \text{id}_B$ a z toho, že f je bijekce. ■

Výše uvedené tvrzení nám dovoluje dokázat:

B.1.15 Důsledek Ať A a B jsou jakékoli množiny. Jestliže platí $\text{card } A = \text{card } B$, potom platí $\text{card } B = \text{card } A$.

B.1.16 Definice Ať A a B jsou dvě množiny. Řekneme, že množina A má nanejvýš stejnou mohutnost jako množina B , když existuje injekce $f : A \rightarrow B$. Tento fakt značíme $\text{card } A \leq \text{card } B$.

B.1.17 Tvrzení Pro jakoukoli množinu A platí $\text{card } A \leq \text{card } A$. Pro jakékoli množiny A , B a C platí: jestliže $\text{card } A \leq \text{card } B$ a současně $\text{card } B \leq \text{card } C$, potom $\text{card } A \leq \text{card } C$.

DŮKAZ. Přečtěte si důkaz tvrzení B.1.11 a použijte jen tu část, kde se mluví o injkcích. ■

B.1.18 Příklad Ukážeme, že $\text{card } \mathbb{N} \leq \text{card } \mathbb{R}$. K tomu stačí sestrojít injekci $f : \mathbb{N} \rightarrow \mathbb{R}$. Definujme $f(n) = n$, $n \in \mathbb{N}$. Toto zobrazení je zřejmě injekce.

Následující tvrzení je „intuitivně zřejmé“. Důkaz je však poměrně komplikovaný, proto jej neuvádíme.

B.1.19 Věta (Cantor-Schröder-Bernstein) Ať A a B jsou množiny. Jestliže platí $\text{card } A \leq \text{card } B$ a současně $\text{card } B \leq \text{card } A$, pak platí $\text{card } A = \text{card } B$.

B.1.20 Definice Ať A a B jsou dvě množiny. Řekneme, že množina A má menší mohutnost než množina B , když platí $\text{card } A \leq \text{card } B$ a neplatí $\text{card } A = \text{card } B$. Tento fakt značíme $\text{card } A < \text{card } B$.

Následující tvrzení nám dovolí vytvářet množiny stále větších mohutností: množina všech podmnožin množiny A má vždy větší mohutnost než množina A .

B.1.21 Věta (Cantor) Ať A je jakákoli množina. Označme $B = \exp A$. Potom platí $\text{card } A < \text{card } B$.

DŮKAZ. Nejprve ukážeme, že platí $\text{card } A \leq \text{card } B$. K tomu stačí sestrojít injekci $f : A \rightarrow B$. Takovou injekcí je $f : x \mapsto \{x\}$. Jde totiž jistě o funkci a z rovnosti $f(x) = f(x')$ plyne rovnost $x = x'$.

Ukážeme, že neplatí $\text{card } A = \text{card } B$. K tomu stačí ukázat, že neexistuje bijekce $g : A \rightarrow B$. Budeme postupovat sporem, tj. ať *existuje* bijekce $g : A \rightarrow B$. Protože g musí být surjektivní, musí pro prvek $y = \{a \in A \mid a \notin g(a)\} \in B$ existovat prvek $x \in A$ tak, že $g(x) = y$. Tento prvek x smíme dále používat. Mohou nastat dva případy:²

1. $x \in g(x)$. Potom x musí mít charakteristickou vlastnost prvků množiny $g(x)$, tj. musí platit $x \notin g(x)$. To je spor.
2. $x \notin g(x)$. Potom x má charakteristickou vlastnost prvků množiny $g(x)$, tj. musí platit $x \in g(x)$. To je spor.

Zobrazení g nemůže být bijekce. Tudíž rovnost $\text{card } A = \text{card } B$ neplatí. ■

B.2 Konečnost a nekonečnost

V tomto odstavci zavedeme pojmy konečnosti a nekonečnosti množin. Tyto dva pojmy dávají základní rozdělení množin: každá množina je buď konečná nebo nekonečná.

B.2.1 Definice Množina A je *konečná*, když je buď prázdná nebo existuje kladné přirozené číslo n tak, že $\text{card } A = \text{card}\{1, \dots, n\}$.

B.2.2 Poznámka Konečnost množiny A tedy znamená jednu z těchto dvou věcí: buď je množina A prázdná, nebo existuje kladné přirozené číslo n a příslušná bijekce $f : \{1, \dots, n\} \rightarrow A$. V druhém případě můžeme označit $a_1 = f(1), \dots, a_n = f(n)$, takže platí $A = \{a_1, a_2, \dots, a_n\}$. Tudíž konečná množina je buď prázdná nebo lze její prvky očíslovat čísly od 1 do n , pro nějaké kladné přirozené číslo n .

B.2.3 Příklad Množina $\{a, b, c\}$ je konečná, viz příklad B.1.7.

B.2.4 Definice Množina A je *nekonečná*, když není konečná.

B.2.5 Příklad Množina \mathbb{N} je nekonečná. Musíme ukázat, že \mathbb{N} není konečná množina.

Protože platí $\mathbb{N} \neq \emptyset$, stačí (viz poznámku B.2.2) ukázat, že neexistuje kladné přirozené číslo n tak, že $\mathbb{N} = \{a_1, \dots, a_n\}$. Budeme postupovat sporem.

Ať n je kladné přirozené číslo takové, že platí $\mathbb{N} = \{a_1, \dots, a_n\}$. Definujme $y = \max(a_1, \dots, a_n) + 1$. Toto y je jistě přirozené číslo, ale platí $y \notin \{a_1, \dots, a_n\}$. To je spor.

B.3 Spočetnost a nespočetnost

Další dělení nekonečných množin je na množiny spočetné a nespočetné. Každá nekonečná množina je buď spočetná nebo nespočetná.

B.3.1 Definice Řekneme, že množina A je *spočetná*, když platí $\text{card } A = \text{card } \mathbb{N}$.

B.3.2 Poznámka Spočetnost množiny A tedy znamená přesně to, že prvky množiny A můžeme srovnat do prosté nekonečné posloupnosti: vezměte bijekci $f : \mathbb{N} \rightarrow A$ a označte $a_n = f(n)$. Potom $A = \{a_0, a_1, a_2, \dots\}$.

B.3.3 Příklad Množina \mathbb{S} všech sudých přirozených čísel je spočetná množina, viz příklad B.1.8.

²Porovnejte s důkazem Russellova paradoxu v A.2.1.

B.3.4 Příklad Množina \mathbb{Z} všech celých čísel je spočetná množina.

Stačí sestrojít bijekci $f : \mathbb{N} \rightarrow \mathbb{Z}$. Definujeme:

$$f(n) = \begin{cases} \frac{n}{2}, & \text{jestliže } n \text{ je sudé} \\ -\frac{n+1}{2}, & \text{jestliže } n \text{ je liché} \end{cases}$$

Především jsme skutečně definovali funkci. Ukážeme, že f je bijekce.

1. f je injektivní. Ať platí $f(n) = f(n')$. Mohou nastat dva případy:

- (a) $f(n) \geq 0$. Pak rovnost $f(n) = f(n')$ znamená rovnost $\frac{n}{2} = \frac{n'}{2}$. To ovšem znamená, že platí $n = n'$.
- (b) $f(n) < 0$. Pak rovnost $f(n) = f(n')$ znamená rovnost $-\frac{n+1}{2} = -\frac{n'+1}{2}$. To ovšem znamená, že platí $n = n'$.

Ukázali jsme, že f je injekce.

2. f je surjektivní. Ať z je celé číslo. Mohou nastat dva případy:

- (a) $z \geq 0$. Definujme $n = 2z$. Potom n je sudé přirozené číslo a platí $f(n) = z$.
- (b) $z < 0$. Definujme $n = -2z - 1$. Potom n je liché přirozené číslo a platí $f(n) = z$.

Ukázali jsme, že f je surjekce.

Sestrojít příklad nespočetné množiny je snadné.

B.3.5 Příklad $\exp \mathbb{N}$ je nespočetná množina. K tomu stačí použít větu B.1.21.

Následující výsledek má velké použití v computer science.

B.3.6 Tvrzení Ať Σ je neprázdňá konečná množina (chápaná jako abeceda). Označte jako Σ^* množinu všech konečných řetězců znaků ze Σ (takovému řetězci říkáme slovo nad Σ). Potom platí:

1. Σ^* je spočetná množina.
2. $\exp \Sigma^*$ je nespočetná množina.

DŮKAZ. Stačí dokázat první tvrzení, druhé z něj potom plyne pomocí Cantorovy věty B.1.21.

Označme $\Sigma = \{x_1, \dots, x_n\}$, kde $n \geq 1$. Pro lepší vyjadřování budeme říkat, že seřazení x_1, \dots, x_n je seřazení písmen ze Σ podle abecedy.

Označme dále, pro každé $k \in \mathbb{N}$, jako W_k množinu všech slov nad Σ délky přesně k . Potom každá množina W_k je konečná a má přesně n^k prvků. To je proto, že v každém slově záleží na pořadí písmen, a proto každé slovo délky k je uspořádaná k -tice vytvořená z n různých znaků.

Slova z množiny W_k uspořádáme lexikograficky, tj. $w_1^k = x_1x_1 \dots x_1x_1$, $w_2^k = x_1x_1 \dots x_1x_2$, \dots , $w_{n^k-1}^k = x_nx_n \dots x_nx_{n-1}$, $w_{n^k}^k = x_nx_n \dots x_nx_n$.

Abychom ukázali, že množina Σ^* je spočetná, stačí srovnat prvky Σ^* do prosté nekonečné posloupnosti. To uděláme takto:

$$w_1^0, w_1^1, \dots, w_n^1, w_1^2, \dots, w_{n^2}^2, w_1^3, \dots, w_{n^3}^3, w_1^4, \dots, w_{n^4}^4, \dots, w_1^k, \dots, w_{n^k}^k, \dots$$

To znamená: postupně za sebe budeme psát slova délky 0, délky 1, délky 2, a tak dále, a jednotlivé bloky slov pevné délky budou seřazeny lexikograficky. ■



B.3.7 Poznámka Proč je výsledek tvrzení B.3.6 důležitý? V computer science se každé podmnožině L množiny Σ^* říká formální jazyk nad Σ . Každá taková množina L je totiž nějaká množina slov nad abecedou Σ .

Příklad: pro každou konečnou množinu At atomických formulí je množina formulí výrokové logiky nad At formálním jazykem nad abecedou $\Sigma = \{\mathbf{tt}, \wedge, \vee, \Rightarrow, \Leftrightarrow, \neg, (,)\} \cup At$.

Tvrzení B.3.6 říká, že formálních jazyků nad konečnou abecedou je „hodně“, totiž nespočetně mnoho.

Následující příklad bude základem důležité konstrukce nespočetné množiny.

B.3.8 Příklad Představme si, že máme matici M rozměru $n \times n$ obsahující znaky 0 a 1. Naším úkolem je napsat vektor znaků 0, 1 délky n , který je *různý* od všech řádků matice M .

Tuto úlohu lze samozřejmě vyřešit řadou způsobů, například hrubou silou. My ale popíšeme elegantní a především *uniformní* algoritmus, který bude pracovat pro jakoukoli hodnotu n a jakoukoli matici M stejně.

Označíme jako $r_1 = (r_{11}, \dots, r_{1n}), \dots, r_n = (r_{n1}, \dots, r_{nn})$ řádky matice M . Hledaný vektor je

$$v = (1 - r_{11}, \dots, 1 - r_{nn}).$$

Vektory v a r_1 jsou jistě různé, protože se liší v první souřadnici. Z podobných příčin platí, že vektory v a r_2 jsou různé (liši se v druhé souřadnici), a tak dále.

Příklad:

$$M = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad v = (0001)$$

Všimněme si co jsme udělali: vzali jsme diagonálu matice M jako zárodek vektoru v a vektor v jsme pak z tohoto zárodku získali prohozením 0 a 1.

Stejný algoritmus (nyní v jeho nekonečné variantě) použijeme při důkazu následující věty.

B.3.9 Věta (Cantorův diagonální trik) Označme jako \mathbb{P} množinu všech nekonečných posloupností znaků 0 a 1. Potom množina \mathbb{P} je nespočetná.

DŮKAZ. Důkaz rozdělíme do dvou částí:

1. Ukážeme, že množina \mathbb{P} je nekonečná. Budeme postupovat sporem: ať je množina \mathbb{P} konečná.

Protože množina \mathbb{P} je jistě neprázdná, musí existovat kladné přirozené číslo n tak, že $\mathbb{P} = \{p_1, \dots, p_n\}$.

Napišme posloupnosti p_1, \dots, p_n do řádků pod sebe a zaměřme se na matici rozměru $n \times n$, která vznikne uvažováním jen o prvních n členech každé posloupnosti. Algoritmem z příkladu B.3.8 pak vytvoříme vektor délky n . Doplňme tento vektor samými znaky 0 na nekonečnou posloupnost a označme ji p . Potom platí $p \neq p_1, \dots, p \neq p_n$. Přesto $p \in \mathbb{P}$.

To je spor: množina \mathbb{P} je nekonečná.

2. Ukážeme, že množina \mathbb{P} je nespočetná. Budeme postupovat sporem: ať je množina \mathbb{P} spočetná (konečná být nemůže, to už jsme dokázali).

Podle poznámky B.3.2 existuje způsob, jak srovnat prvky množiny \mathbb{P} do prosté nekonečné posloupnosti, $\mathbb{P} = \{p_0, p_1, p_2, \dots\}$. Napišme prvky p_0, p_1, p_2, \dots do řádků pod sebe.

Vznikla „nekonečná matice“. Na tuto matici nyní použijeme „nekonečnou variantu“ algoritmu z příkladu B.3.8. Přesněji: definujeme posloupnost v znaků 0 a 1 takto: $v_i = 1 - p_{ii}$ pro každé přirozené číslo i .

Potom platí $v \neq p_n$ pro každé přirozené číslo n . Protože zjevně $v \in \mathbb{P}$, dostáváme spor.

Množina \mathbb{P} je nespočetná. ■

Důkaz je ukončen.

B.3.10 Poznámka Povšimněme si, že množina \mathbb{P} z věty B.3.9 je jakousi nekonečnou analogií množiny slov Σ^* pro abecedu $\Sigma = \{0, 1\}$. Prvky množiny \mathbb{P} ovšem *nejsou slova* nad abecedou Σ , protože každé slovo musí být konečný řetězec písmen ze Σ . V computer science se nekonečné posloupnosti prvků abecedy Σ říká *stream* nad Σ a pro množinu všech streams nad Σ se používá značení Σ^ω . Při tomto značení je \mathbb{P} z věty B.3.9 přesně množina $\{0, 1\}^\omega$.

B.3.11 Poznámka Cantorův diagonální trik patří do pokladnice krásných matematických úvah. Nalezl uplatnění v řadě konstrukcí v matematice a computer science (například *Halting Problem* pro Turingovy stroje).



B.3.12 Poznámka Znovu zdůrazňujeme, že v tomto textu u nekonečných množin *nemluvíme o počtu prvků*. Například množiny \mathbb{N} a $\exp \mathbb{N}$ jsou obě nekonečné, ale podle věty **B.1.21** víme, že platí $\text{card } \mathbb{N} < \text{card } \exp \mathbb{N}$. Tudíž množina $\exp \mathbb{N}$ má „více prvků“ než \mathbb{N} . Museli bychom tedy zavést speciální čísla pro různé „velikosti nekonečnosti“.

V axiomatické teorii množin se taková čísla opravdu zavádějí a říká se jim *kardinální čísla*. Například se tak píše $\text{card } \mathbb{N} = \aleph_0$ a $\text{card } \exp \mathbb{N} = 2^{\aleph_0}$. Symbol \aleph je první písmeno *álef* hebrejské abecedy. Toto značení a terminologii zavedl Georg Cantor. Pro literaturu o kardinálních číslech odkazujeme na seznam doplňující literatury v kapitole **A**.

B.4 Cvičení

B.4.1 Cvičení Ať $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ je funkce definovaná předpisem $f(i, j) = 2^i \cdot 3^j$, $i, j \in \mathbb{N}$. Dokažte, že f je injekce a že f není surjekce.

B.4.2 Cvičení Existuje konečná množina A , pro kterou platí $\text{card } A = \text{card}(A \times A)$?

B.4.3 Cvičení Ať $f : A \times A \rightarrow A$ je bijekce. Dokažte, že funkce $g : A \times A \times A \rightarrow A$, definovaná předpisem $g(x, y, z) = f(f(x, y), z)$, je opět bijekce.

Zformulujte a dokažte na základě výše dokázaného nějakou větu o mohutnostech množin A , $A \times A$ a $A \times A \times A$.

B.4.4 Cvičení Dokažte, že platí $\text{card}(\mathbb{N} \times \mathbb{N}) = \text{card } \mathbb{N}$.

Návod: vymyslete nejprve algoritmus, který očíslování všechny prvky čtvercové matice rozměrů $n \times n$ číslu od 1 do n^2 , a to uniformně pro n . Poté použijte „nekonečnou variantu“ tohoto algoritmu na „nekonečnou matici“ $\mathbb{N} \times \mathbb{N}$.

B.4.5 Cvičení Zobecněte větu **B.3.9** takto (značení Σ^ω je z poznámky **B.3.10**):

Ať Σ je neprázdná konečná abeceda obsahující alespoň dvě písmena. Potom množina Σ^ω je nespočetná.

Návod: jde o to vymyslet analogii proházování 0 a 1 na diagonále příslušné „nekonečné matice“.

Revize kapitoly

Dozvěděli jsme se:

- ✓ Funkci $f : A \rightarrow B$ lze použít jako svědka porovnávání mohutností množin A a B .
- ✓ Množiny dělíme na konečné a nekonečné. Nekonečné množiny dále dělíme na spočetné a nespočetné.

Pro přípravu na zkoušku zkuste zodpovědět následující otázky:

- ✓ Musí být každá podmnožina konečné množiny opět konečná množina?
- ✓ Může být podmnožina spočetné množiny množina nespočetná?

Doplňující literatura

V této kapitole jsme se nepouštěli do dokazování nejrůznějších faktů o konečných, nekonečných, spočetných a nespočetných množinách. Jako standardní referenci odkazujeme na

☞ M. Demlová a B. Pondělíček, *Matematická logika*, skriptum FEL ČVUT, Praha, 1997

Rejstřík

- α -konverse, 43
- adekvátní množina spojek, 34
- álef, 82
- anuloid, 26
- Aristoteles ze Stageiry, 51
- Backus, John Warner, 12
- Backusova normální forma, 12
- Backusova-Naurova forma, 12
- bázická matice
 - jedniček, 26
 - nul, 26
- Cantor, Georg, 72, 75
- Church, Alonzo, 4
- CNF
 - ireducibilní (ve výrokové logice), 30
 - úplná (ve výrokové logice), 20
 - v predikátové logice, 56
 - ve výrokové logice, 17
- deontická logika, 70
- disjunktivní normální forma, 17
 - ireducibilní (ve výrokové logice), 30
 - úplná (ve výrokové logice), 20
- DNF, 17
 - ireducibilní (ve výrokové logice), 30
 - úplná (ve výrokové logice), 20
- Doyle, sir Arthur Conan, 39
- důkaz sporem, 34
- dynamická logika, 70
- ekvisplnitelnost
 - v predikátové logice, 58
 - ve výrokové logice, 54
- Epimenidův paradox, 4
- Eukleides, 74
- formální jazyk nad Σ , 80
- funkce, 75
- Gödel, Kurt, 4, 38
- Gödelova věta o neúplnosti, 4, 38
- Hennesyho-Milnerova modální logika, 38
- inkluse, 73
- intuicionistická logika, 70
- invariant rekursivního algoritmu, 55
- jazyk
 - ALGOL 60, 12
 - HASKELL, 4
 - LISP, 4
 - MIRANDA, 4
 - PROLOG, 65
- kardinální číslo, 82
- Karnaugh, Maurice, 17
- Karnaughova mapa, 24
- kartézský součin množin, 73
- klausule pro CNF
 - v predikátové logice, 56
 - ve výrokové logice, 17
- klausule pro DNF
 - ve výrokové logice, 17
- komplementární výskyt
 - atomické formule, 54
 - predikátového symbolu, 61
- konjunktivní normální forma, 17
 - ireducibilní (ve výrokové logice), 30
 - úplná (ve výrokové logice), 20
- kontext standardních proměnných, 44
- kontradikce, 16, 47
- kvantová logika, 70
- legální přejmenování proměnných, 43
- literál
 - v predikátové logice, 56
 - ve výrokové logice, 17
- logika
 - Hoarových trojic, 38
 - prvního řádu, 71
 - vyššího řádu, 71
- maximální unifikátor, 60
- maxterm, 17
- minterm, 17
- modální logika, 70
- Naur, Peter, 12
- non-wellfounded set theory, 74
- Occamova břitva, 8

- ohodnocení ve výrokové logice, 13
- Pāṇini, 12
- potenční množina, 73
- pravdivostní tabulka, 15
- prázdná množina, 73
- průnik množin, 73
- princip extensionality, 73
- přirozená dedukce, 37
- Recorde, Robert, 39
- relaxace syntaxe
 - v predikátové logice, 41
 - ve výrokové logice, 13, 17
- resoluční algoritmus
 - v predikátové logice, 63
 - ve výrokové logice, 55
- resoluční tabulka, 56
- resolventa klausulí
 - v predikátové logice, 61
 - ve výrokové logice, 54
- Riemann, Georg Friedrich Bernhard, 7
- rozdíl množin, 73
- Russell, Bertrand, 73
- Russelův paradox, 73
- sanskrt, 12
- sémantická ekvivalence, 15, 47
- sémantický důkaz sporem
 - v predikátové logice, 50
 - ve výrokové logice, 34
- sémantický důsledek
 - v predikátové logice, 50
 - ve výrokové logice, 32
- sémantika
 - formulí predikátové logiky, 45
 - termů predikátové logiky, 44
- sentence, 43
- sentence v CNF, 56
- sjednocení množin, 73
- Skolem, Thoralf, 53
- skolemisace, 57
- Skolemovo rozšíření, 57
- slovo nad Σ , 80
- splnitelná
 - formule, 33
 - množina formulí, 33
 - množina sentencí, 46
 - sentence, 46
- Stevenson, Robert Louis, 68
- stream nad Σ , 81
- sylogismus, 51
- syntaktický důsledek, 37
- syntaktický strom, 12
- syntaxe formule
 - predikátové logiky, 40
 - výrokové logiky, 12
- syntaxe termu, 40
- tautologie, 16, 47
- temporální logika, 70
- tělo klausule, 56
- torus, 26
- Turing, Alan Mathison, 4, 74
- unifikační algoritmus, 60
- universum interpretace, 44
- update kontextu standardních proměnných, 44
- úspěšné tablo, 37
- výskyt proměnné
 - vázaný, 42
 - volný, 42
- Whitehead, Alfred North, 73
- Wiles, Andrew, 9
- zamítací strom, 63
- Zermelova-Fraenkelova teorie množin, 74
- zobrazení, 75