Univerzita Karlova v Praze, Filozofická fakulta
Katedra logiky

Petr Révay

FORMALIZACE RELACE ODVODITELNOSTI PRO VÝROKOVÉ FUZZY LOGIKY

FORMALIZATION OF THE DEDUCIBILITY RELATION OF PROPOSITIONAL FUZZY LOGICS

Bakalářská práce

Vedoucí práce: Mgr. Libor Běhounek, Ph.D.

2014

Tímto bych rád poděkoval svému vedoucímu práce doktoru Liboru Běhounkovi za cenné připomínky a rady jak k obsahové, tak i typografické stránce textu; zejména pak za jeho ochotu při dokončování práce.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a že jsem uvedl všechny použité prameny a literaturu.

V Praze 18. srpna 2014

Petr Révay

Abstrakt

Tato bakalářská práce předkládá formalizaci relace odvoditelnosti fuzzy logiky BL v prostředí matematického asistenta Isabelle/HOL a počítačově ověřené důkazy některých teorémů a meta-teorémů této logiky. Zároveň poskytuje popis procesu formalizace a použité prostředky Isabelle/HOL předvádí na jednodušších příkladech. Samostatná kapitola je pak věnována úvodu do logiky BL.

Po nastudování dokumentace a volbě vhodných nástrojů Isabelle/HOL byla implementována formalizace, díky níž lze v programu ověřovat důkazy jak v axiomatizaci BL, tak i důkazy vlastností relace dokazatelnosti. Z těch byl formalizován především důkaz věty o lokální dedukci. Dále byly formalizovány důkazy řady teorémů BL, a to včetně odvození redundantních axiomů *BL2* a *BL3*.

Přínosem této práce je prozkoumání možností verifikátoru Isabelle/HOL z hlediska použití pro fuzzy logiku BL. Vzhledem k uvedeným poznatkům je možné práci použít jako základ širšího projektu formalizace fuzzy logik, které z logiky BL vycházejí.

Klíčová slova: Basic logic, Isabelle, HOL, automatické dokazování, formalizace

Abstract

This bachelor thesis presents the formalization of provability relation of fuzzy logic BL in the environment of a mathematical assistant Isabelle/HOL and also the computer-verified proofs of some theorems and syntactic meta-theorems of this logic. It also provides a description of the process of formalization and describes the tools of Isabelle/HOL used in the code by simple examples. The separate chapter is devoted to the introduction to the logic BL.

The formalization has been implemented after studying the documentation and choosing of the appropriate methods of Isabelle/HOL. The formalization allows the software to verify the proofs in the axiomatization of BL, and moreover, of the properties of the provability relation itself. Especially, the proof of the local deduction theorem has been formalized as well as the proofs of the theorems of BL, including the derivations of the redundant axioms *BL2* and *BL3*.

The major objective of this bachelor thesis is to explore the possibilities of the proof-checker Isabelle/HOL in terms of fuzzy logic BL. With respect to the obtained results the thesis can be used as the starting point of a wider project of formalization of the others fuzzy logics which are based on the logic BL.

Keywords: Basic logic, Isabelle, HOL, automated proving, formalization

Obsah

1 Úvod

1.1 Cíl práce

Cílem práce je formalizovat relaci odvoditelnosti (dále jen dokazatelnosti) fuzzy logiky BL [12] v softwarovém prostředí matematického asistenta Isabelle/HOL [1], demonstrovat využití tohoto nástroje k ověření důkazů některých teorémů a metateorémů logiky BL a nastínit možnost pokračování ve formalizaci dalších objektů a tvrzení. Zároveň by práce měla poskytnou dokumentaci pro potenciální uživatele přiloženého kódu.

1.2 Struktura práce

Práce sestává z popisu prostředí matematického asistenta Isabelle/HOL (včetně ukázek některých postupů na jednodušších datových typech) a dále z teoretické části věnované logice BL, formalizaci její syntaxe a verifikaci důkazů některých tvrzení. Práce jako přílohu obsahuje CD s formalizací BL pro verzi programu Isabelle2013-2.

Poznámka: Program byl provozován na operačním systému GNU/Linux v distribuci Debian 7.0 Wheezy pro architekturu AMD64. Pro provoz Isabelle/HOL na jiných operačních systémech viz sekci *Installation* v [1].

2 Isabelle/HOL

Isabelle/HOL je název instance interaktivního důkazového verifikátoru[1] Isabelle s rozšířením o nástroje pro verifikaci teorémů logiky vyšších řádů (Higher-Order Logic). V Isabelle jsou matematické objekty popisovány pomocí teorie typů, na nichž operují odvozovací pravidla deklarovaná uživatelem [18]. Vzniklý rámec tvoří metalogiku, ve které lze popisovat formalizovanou matematiku či logiku jako takovou. Tento princip strojového ověřování není v Isabelle původní a svými kořeny sahá až do 60. let 20. století.

2.1 Historie

2.1.1 Logika rekurzivních funkcí

Uvedený způsob formalizace vychází z principů použitých při konstrukci důkazového verifikátoru LCF navrženého Robinem Milnerem a jeho týmem ze Standford University v roce 1972 [4]. LCF je zkratkou pro *Logic for Computable Functions*, jak Milner nazývá logický systém vytvořený Danou Scottem pro formální práci s algoritmy skrze typově-teoretický přístup [19].

Scott zavádí základní typy nejnižšího řádu: ι pro typ individuí a o pro typ pravdivostních hodnot. Z nich odvozuje typy vyšších řádů - funkce mezi objekty libovolných typů, tj. $F\colon(\alpha \to \beta)$. Formulemi jeho logiky jsou buď atomické formule tvaru $X \leq Y$ pro nějaké výrazy X a Y,[2] nebo posloupnosti $\varphi_1, \ldots, \varphi_n$, kde φ_i je atomická formule. Tvrzením pak nazývá výraz $\varphi \vdash \psi$, kde φ a ψ jsou posloupnosti atomických formulí považované za konjunkci svých prvků, přičemž symbol $\vdash$ hraje roli implikace mezi těmito konjunkcemi [19, s. 418]. Z dané axiomatiky pak lze generovat platná tvrzení pomocí sady odvozovacích pravidel.

2.1.2 LCF a ML

Důkazový verifikátor LCF byl navržen pro automatické ověřování důkazů Scottovy logiky. Hlavní ideou LCF byla deklarace dokazované formule Scottova systému a její následné dělení na podcíle pomocí implementovaných příkazů.[3] Podcíle byly dokazovány pomocí simplifikátoru,[4] nebo dále děleny na další podcíle, dokud je nebylo možné dokázat.

[1] z angl. *„proof checker“*

[2] Scott vytváří systém, ve kterém lze formalizovat mimo jiné i o netotální funkce. Relaci $\leq$ interpretuje jako *„je méně nebo stejně definován jako“*. Viz str. 419 v [19].

[3] v orig. *„subgoaling commands“*

[4] podrobně později

Standford LCF, jak se tato první verze verifikátoru LFC nazývá, ukládal všechny kroky důkazu a zachycoval tak jeho strukturu, což vedlo k vysoké paměťové náročnosti. Dalším problémem byla nemožnost jednodušše rozšířit sadu řídících příkazů [10].

Po změně působiště proto Robin Milner vyvinul další verzi programu známou jako Edinburgh LCF [11]. V první řadě program začal namísto celého důkazu ukládat do paměti pouze výsledky dílčích kroků, tedy dokázaná tvrzení. Dále Milner zavedl abstraktní datový typ, jehož hodnotami byly instance axiomů, na nichž operují odvozovací pravidla [10].
Pro psaní řídících příkazů Milner a kolektiv vyvinuli funkcionální programovací jazyk ML,[5] v němž jsou příkazy pro generování podcílů reprezentovány funkcemi zvanými *taktiky* (z angl. *„tactics“*) a operace kombinující taktiky a vracející taktiku nově vytvořenou, tzv. *taktické operace*[6] (z angl. *„taticals“*), mohou být reprezentovány funkcemi složenými, tedy funkcemi vyšších řádů.

O výrazné zlepšení paměťových a časových nároků LCF se zasloužil v Cambridge působící Larry Paulson - jeden z autorů Isabelle/HOL. Paulson do LCF přidal další důkazové nástroje a rozšířil logiku LCF tak, aby mohla být použita pro práci s predikátovou logikou (jednalo se zejména o přidání disjunkce a existenčního kvantifikátoru, které Scottův systém neobsahoval). Vznikající další verze verifikátoru byla dokončena okolo roku 1985 a nazvána Cambridge LCF.

2.1.3 HOL

Během vývoje Cambridge LCF pracoval Mike Gordon z University of Cambridge na ověřování digitálního hardware. S využítím některých principů z Milnerova kalkulu CCS (*Calculus for communicating systems*) zavedl pro svoji potřebu vlastní notaci LSM (*Logic of Sequential Machines*). Potřebný verifikátor důkazů pro LSM vytvořil z Cambridge LCF přidáním parseru pro výrazy LSM a implementací jistého axiomu z CCS. Později se ukázalo, že tento axiom lze zachytit inferencemi predikátové logiky a dále že oproti Scottovu systému, tedy i LCF, pro ověřování hardware stačí standardní matematická indukce a typy mohou být reprezentovány jako množiny v běžném slova smyslu.[7] Tyto změny při zachování syntaxe LCF vedly ke vzniku verifikátoru HOL [10, s. 5].

HOL (High Order Logic) přebírá z LCF např. systém typů, ale na rozdíl od LCF upřednostňuje formalizaci teorií pomocí definic spíše než skrze

[5] zkr. *„Meta Language“*
[6] vlastní překlad
[7] srov. *the fixed-point induction* a *Scott domains* v [19]

deklarování axiomatiky. Přestože je zavedení axiomatizace stále možné (např. pro teorii grup), obecně je uživatel před tímto přístupem varován, protože tak lze mnohem snadněji zavést do formalizované teorie spor [17, s. 11, 25, 164]. Zásadním krokem při vývoji HOL byla implementace automatických nástrojů, které z deklarované rekurzivní definice vygenerují schémata indukce pro daný datový typ.

2.1.4 Vznik a vývoj Isabelle/HOL

Zároveň s HOL vznikla řada dalších verifikátorů založených na principech LCF. Jedním z nich je i Isabelle, verifikátor vyvinutý koncem 80. let už zmíněným Larrym Paulsonem z Cambridge. Zatímco v LCF a HOL je metalogika poskytována skripty v ML, v Isabelle je možné popsat ji deklarativně a nechat skripty v ML běžet na pozadí. Metalogická pravidla jsou pak kombinována pomocí meta-odvozování založeném na unifikaci a rezoluci pro typy vyšších řádů [10].

Jednou z objektových logik vyvinutých Paulsonem pro Isabelle je HOL. Jedná se o sadu (polo)automatických nástrojů pro práci s objekty logiky vyšších řádů a nesmí být zaměňována s výše popsaným Gordonovým verifikátorem HOL. Celý systém, jak je dnes distribuován, nese název Isabelle/HOL. Jeho vývoj pokračuje už přes dvacet let na University of Cambridge (Larry Paulson), TU München (Tobias Nipkow) a Université Paris-Sud (Makarius Wenzel) a je prezentován na webové stránce projektu [1].

2.2 Popis pracovního prostředí

2.2.1 Instalace Isabelle/HOL

Aktuální verze verifikátoru Isabelle/HOL je dostupná na domovské stránce projektu v sekci *Downloads*. Po stažení a dekompresi archivu je třeba spustit skript Isabelle2013-2.run, který provede kompilaci součástí programu. První spuštění tedy může trvat delší dobu. Po skončení iniciačních skriptů se otevře okno editoru a program je připraven k práci.

2.2.2 Isabelle/jEdit

Použitým rozhraním pro práci s Isabelle/HOL je textový editor jEdit [3]. Tento editor je napsaný v jazyce Java, tedy jej lze spustit v jakémkoliv operačním systému s podporou Java Virtual Machine (MS Windows, Unix, Mac OS a další). Jeho velkou předností je možnost přidání zásuvných modulů - tím je možné distribuovat Isabelle jako instalaci editoru s implementovaným zásuvným modulem Isabelle (tento celek je označován jako Isabelle/jEdit).

Dále editor podporuje použití maker a programovatelných klávesových zkratek pro snadnější editaci textu.

Vzhledem k celkovému zaměření Isabelle na čitelný zdrojový kód ve smyslu syntaxe založené na běžných matematických symbolech, je pro tento účel jEdit vhodným prostředím a práce v něm je velmi přirozená a připomíná psaní v prostředí LaTeX s tím rozdílem, že v jEdit je výsledná podoba textu generována okamžitě. Například pro zápis symbolů řecké abecedy stačí napsat řetězec \phi, v našeptávacím okně vybrat správný symbol a potvrdit klávesou TAB.[8]

Stejně tak je možné editovat vlastní zkratky pro psaní symbolů. Pro formalizaci logiky BL (viz odd. 4.2) je použita notace klasických spojek s dolním indexem BL. Pro rychlý zápis $\longrightarrow_{BL}$ lze pak v Utilities / Global Options / jEdit / Abbreviations nastavit například zkratku BLimp - napsáním tohoto řetězce přímo do kódu a stiskem Ctrl + ; pak dojde k jeho přepsání na požadovanou skupinu symbolů, což velmi urychluje práci.

Pracovní prostředí Isabelle/jEdit se skládá z několika částí:

- Text area pro zadávání kódu,
- Output window pro sledování výstupu automatických skriptů,
- Documentation panel pro rychlý přístup k dokumentaci v pdf.

Output window lze dále přepnout na funkci vyhledávání v textu, správu dokazovacího nástroje Sledgehammer (viz s. 36) a seznam předdefinovaných symbolů. Rovněž Documentation panel navíc nabízí přehled editovaných souborů a jejich vnitřních struktur. Rozložení oken i panelů lze nastavit podle potřeby, a to včetně jejich přepnutí do plovoucího režimu například pro práci na více monitorech.

Isabelle/HOL je interaktivní matematický asistent, tedy poskytuje zpětnou vazbu k zadanému kódu v reálném čase. To může díky výpočetní náročnosti zpomalovat editaci, proto je často vhodné v panelu dokumentace po přepnutí na režim Theories zrušit výběr zaškrtávacího políčka Continuous checking.

2.3 Dokumentace

Dokumentace k Isabelle/HOL je dostupná v podobě sady tutoriálů a manuálů zaměřených na dílčí části systému, popřípadě obecně na práci s programem. Aktuální verze dokumentace je dodávána v rámci distribuce programu a lze

[8] platí ve verzi Isabelle2013-2

k ní přistupovat přímo v editoru jEdit skrz panel Documentation, případně je uvedena na domovské stránce projektu [1].

Pro potřeby logiky BL byly použity především dokumenty *tutorial.pdf* [17] - obsáhlý uživatelský tutoriál pro práci v HOL; dále *prog-prove.pdf* [15] - stručnější tutoriál na straně uživatele předpokládající základní zkušenosti s funkcionálním programováním a logickou a množinově-teoretickou notací. Dále také *isar-ref.pdf* [21] s podrobným popisem syntaxe použitého jazyka.

2.4 Syntax

Verifikátor Isabelle je naprogramovaný v již zmíněném jazyce ML, což se místy promítá do podoby syntaxe použitého jazyka. V Isabelle/HOL je použit vlastní jazyk *Isabelle/Isar* (zkráceně *Isar*), který umožňuje volání implementovaných skriptů jazyka ML jednodušší a čitelnější cestou. Jednou z hlavních předností HOL je zaměření projektu právě na čitelnost výsledného kódu tak, aby byl nejen přehledný, ale dokonce i přímo publikovatelný jako matematický text. Isar proto podporuje standarní matematické symboly, vkládání textu a členění dokumentu pro export do formátu LaTeX.[9]

Poznámka Jazyk Isar obsahuje značné množství příkazů. Zde uvedené příklady se omezují na konstrukce použité při formalizaci BL, tedy popis jazyka Isar zdaleka není vyčerpán.[10]

2.4.1 Teorie

Zadávání vstupu pro Isabelle/HOL spočívá ve tvorbě zdrojových souborů s příponou .thy, tzv. *teorií*.[11] Struktura teorie je daná klíčovými slovy **theory**, **imports**, **begin** a **end**, jak je vidět na následující ukázce kódu Isabelle/HOL.

Formalizace 2.4.1. *Struktura zdrojového souboru – teorie*

```
theory  NewTheory
imports Main

begin
  deklarace, definice, tvrzení, ...
end
```

[9] více v [17, odd. 4.2]

[10] Pro úplný seznam klíčových slov a gramatiku Isar viz [21]

[11] v orig. *the theory; the theory file*

Mezi **begin** a **end** se nachází samotný kód vstupu, tedy všechny deklarace, definice, důkazy apod. Klíčové slovo **imports** slouží k připojení teorie z jiného souboru - k její identifikaci slouží jméno teorie uvedené za klíčovým slovem **theory** (v našem případě *NewTheory*), které se musí shodovat s názvem souboru před příponou .thy.

Možnost importovat teorie je velmi podstatná. Jednak je možné vlastní projekt strukturovat do více částí, a jednak existuje velké množství předdefinovaných teorií pro práci se základními objekty.

Jednou z významných teorií je *Main* obsahující předdefinované teorie pro práci s přirozenými čísly, aritmetikou, funkcemi, seznamy, množinami a více než 50 dalšími teoriemi z knihovny HOL.[12]. Autoři Isabelle v tutoriálu doporučují importovat *Main* vždy, pokud neexistuje závažný důvod to nedělat.[13]

2.4.2 Datové typy

Abstraktní vrstva HOL je logikou typů, jimž odpovídají datové typy funkcionálních jazyků jako jsou ML, Haskell [17] [sec.1.3] nebo právě Isar. Jedná se o:

- **základní typy**, zejména pravdivostní hodnoty `bool`, přirozená čísla `nat` a celá čísla `int`,
- **typové konstruktory**, zejména seznamy `list` a množiny `set`,
- **funkcionální typy** značené $\Rightarrow$,
- **typové proměnné** značené `'a`, `'b` atd.

Typové konstruktory jsou zapisovány v postfixové notaci, tedy např. zápis `(nat)set` odpovídá množině objektů typu `nat`, tj. přirozených čísel, apod.

Pomocích typových proměnných a konstruktorů lze definovat objekty, jejichž typ je dán až konkrétní instancí objektu. Například konstruktor `('a)list` je seznamem objektů nějakého typu `'a`. Pokud bychom uvážili konstruktor `pair` se dvěma argumenty definovaný jako `('a,'a)pair`, vyžadoval by argumenty stejného typu.

V jazyce Isar je datový typ definován pomocí klíčového slova **datatype** uvedením jeho názvu a konstruktorů. Formalizace datového typu přirozených čísel by pak mohla vypadat takto:

[12] pro úplný výčet prvků v *Main* viz [16]
[13] [17, odd. 1.2]

Formalizace 2.4.2. *Datový typ přirozených čísel*

```
datatype Natural = Zero | Suc Natural
```

Formalizaci lze číst „Objektem datového typu `Natural` je objekt jménem `Zero` nebo objekt složený ze jména `Suc` a objektu typu `Natural`“. Symbol `|` odděluje jednotlivé případy konstruktorů a má význam „nebo“. Objekty definovaného typu `Natural` jsou tedy například `Zero`, `Suc(Suc(Suc(Zero)))` atd.

Implementace jazyka Isar disponuje typovou inferencí, tedy automatickým odvozováním nedeklarovaných typů objektů. Pokud není možné typ objektu odvodit, je nutné jej uvést explicitně způsobem jako např. ve výrazu $x \leq$ `(y::nat)`. Pokud je $\leq$ definována jako typ `nat` $\Rightarrow$ `nat` $\Rightarrow$ `bool`, program sám odvodí, že `x::nat` [17].

2.4.3 Matematická notace

Prostředí Isabelle/jEdit podporuje používání symbolické notace ve smyslu zkratek za jména konstruktorů objektů. Například v deklaraci datového typu `Natural` lze definovat zápis konstruktoru `Suc` pomocí symbolu $\mathcal{S}$, a to jeho uvedením ve tvaru `("`$\mathcal{S}$`")` za deklaraci konstruktoru. Jméno objektu $\mathcal{S}$`(`$\mathcal{S}$`(Zero))` má pak pro Isabelle stejný význam jako `Suc(Suc(Zero))`.

Pro konstruktory s více než jedním argumentem je možné definovat zápis v infixové notaci s uvedením priority a smyslem závorkování, jak bude vidět v oddílu 2.4.4 při formalizaci operace sčítání na typu `Natural`.[14]

2.4.4 Funkce

V Isabelle lze funkce definovat více způsoby. V textu se omezíme na totální rekurzivní funkce definované pomocí klíčového slova **fun**, u nichž Isabelle automaticky dokazuje omezenost rekurze – terminaci. Provádí tak ověřením, že při každém rekurzivním volání dochází ke zmenšení některého z argumentů [13]. Obecnějším způsobem deklarace je pak příkaz **function** vhodný pro ty funkce, na nichž automatické nástroje Isabelle selhávají a jejichž terminaci je tak třeba dokázat ručně.[15]

[14] Některé symboly v Isabelle jsou přetížené, a to zejména po importu natolik obsáhlé knihovny jako `Main`. Pro seznam přetížených symbolů viz Appendix A v [17], pro seznam předdefinovaných symbolů Isabelle/HOL s jejich ASCII zápisem viz Appendix B v [21].

[15] Selhání automatického dokázaní terminace v případě **fun** může být způsobeno chybnou deklarací definice, nebo omezenými schopnostmi automatického nástroje v Isabelle. Pro více podrobností viz [13, odd. 3].

Funkce je definována jako objekt vyššího řádu klíčovým slovem **fun**, určením jména objektu a deklarací jeho funkcionálního typu, jak je vidět na následující formalizaci:

Formalizace 2.4.3. *Sčítání na typu* `Natural`

```
fun Add :: "Natural ⇒ Natural ⇒ Natural"
(infixr "⨁" 50)
where
  "(m ⨁ Zero) = m" |
  "(m ⨁ 𝒮 n) = 𝒮 (m ⨁ n)"
```

Za deklarací typu je možné definovat symbolickou notaci. Výraz **infixr** určuje infixovou notaci automaticky závorkovanou zprava[16] s uvedenou preferencí `50`[17] a symbol ⨁ je novým synonymem pro jméno `Add`, které může být použito již v konstruktorech funkce. Následuje klíčové slovo **where** a samotné konstruktory funkce oddělené symbolem `|` stejně jako v případě formalizace 2.4.2.

Funkce z formalizace 2.4.3 je rekurzivní definicí sčítání na typu `Natural` a její konstruktory vycházejí z axiomů Q4 a Q5 Peanovy aritmetiky. Konstruktory funkce vyčerpávají konstruktory datových typů argumentů, tedy `Zero` a $\mathcal{S}$ a výpočet návratové hodnoty vychází z principu *pattern matching*, který je v Isabelle/HOL rovněž implementována [13, odd. 2.1].

Při volání funkce se tak program snaží „dosadit“ dané argumenty do předpisu na levé straně rovnítka v konstruktoru, a pokud argumenty odpovídají předpisu, je vypočtena návratová hodnota funkce. Protože znak `m` nebyl doposavad vytížen, je považován za proměnnou typu `Natural`, respektive za pozici v předpisu, na kterou lze v rámci pattern matching dosadit libovolný objekt tohoto typu.

2.4.5 Množiny

Množiny jsou v Isabelle/HOL reprezentovány datovým typem `set`, jehož definice je součástí importované teorie `Main`. Množina může obsahovat objekty jakéhokoliv datového typu.

Pro formalizaci logiky BL je důležitá možnost definovat induktivní množiny. Příkladem může být definice množiny sudých čísel jako podmnožiny

[16] zleva pomocí **infixl**

[17] povolený rozsah preference závorkování je 0–1000, viz odd. 4.1.1 v [17]

všech čísel typu `Natural`. Definice je čitelná a pojmenování konstruktorů je zavedeno kvůli pozdějším odkazům. Více viz [17, kap. 7]).

Formalizace 2.4.4. *Definice induktivní množiny*

```
inductive_set EvenNat :: "Natural set"
where
Base: "Zero ∈ EvenNat" |
Step: "n ∈ EvenNat ⟹ 𝒮(𝒮 n) ∈ EvenNat"
```

2.4.6 Evaluace

K ohodnocení termů v Isabelle/HOL slouží příkaz **value**. Termy lze jednak ohodnotit objekty typu, ze kterého je term odvozen, tedy například:

```
value "𝒮(𝒮(Zero)) ⨁ 𝒮(Zero)"
```

V okně výstupu je pak zobrazena funkční hodnota uvedeného výrazu. Obecně lze hodnotu formalizované n-ární funkce f s argumenty $x_1, \ldots, x_n$ lze zjistit příkazem **value** `"f x`$_1$ `... x`$_n$`"`.

Dále je možné term ohodnotit proměnnými, případně kombinací konstant a proměnných. To je užitečné pro odstraňování chyb ve formalizaci, popřípadě pro průběžnou kontrolu kódu.

2.4.7 Definice

Definice zavádějí nová jména pro již existující objekty, a to na úrovni typů a termů. Jména typů jsou definována pomocí klíčového slova **type_synonym** a jsou vhodná pro zkrácení kódu či autodokumentační účely.

Uvažme funkci `Add` z formalizace 2.4.3, která je objektem HOL typu `Natural ⇒ Natural ⇒ Natural`. Pro tento typ lze pomocí typové definice zavést intuitivnější pojmenování:

Formalizace 2.4.5. *Přejmenování typů*

```
type_synonym binary_operation_N = "Natural ⇒ Natural ⇒
Natural"
```

V Isabelle jsou takto definované zkratky interně překládány do původního významu a ve výstupu programu se už neobjevují (viz [17, odd. 2.7.1]).

Jména termů jsou zaváděna klíčovým slovem **definition** a slouží k přejmenovávání dříve definovaných objektů HOL. Definice jsou vždy nerekurzivní[18] a dají se použít k zavedení symbolické notace.

Jako příklad definujme zkratku `Double` pro aplikaci operace $\bigoplus$ na dva stejné argumenty a její notaci `2 *`, která ji bude v kódu zastupovat. Formalizace vypadá následovně:

Formalizace 2.4.6. *Definice Double(x)*

```
definition Double :: "Natural ⇒ Natural"
("2 *")
where
  "Double x ≡ x ⊕ x"
```

Definice nejsou aplikovány automaticky a je třeba na ně odkazovat jejich jménem s připojeným řetězcem `_def`. Odkazem na definici `Double` je tedy `Double_def`.

2.4.8 Proměnné

Isabelle automaticky odlišuje volné a vázané proměnné a vázané proměnné vnitřně přejmenovává, aby nedocházelo k jejich záměně s proměnnými volnými. Dále používá vlastní typ proměnných, tzv. *schématické proměnné* začínající znakem `?`. Isabelle jimi ve formuli vnitřně nahrazuje volné proměnné, které musí být během důkazu instanciovány nějakým termem.

2.4.9 Termy a formule HOL

Termy jazyka Isar jsou tvořeny aplikováním funkcí na argumenty. Pokud f je term typu $\tau_1 \Rightarrow \tau_2$ a t je term typu τ_1, pak $f\ t$ je typu τ_2 [17]. Zápis $t :: \tau$ pak znamená, že term t je typu τ.

Formule jsou termy typu `bool`. Použity jsou konstanty `True` a `False` a běžné výrokové spojky (sestupně podle priority závorkování): $\neg$, $\wedge$, $\vee$ a $\rightarrow$. Všechny binární spojky jsou automaticky závorkovány zprava.

[18] [17, odd. 2.7]

Rovnost je funkcí `=` typu `'a ⇒ 'a ⇒ bool`. Při porovnávání výrazů typu `bool` zastupuje roli spojky *právě tehdy, když*. Výraz $t_1 \neq t_2$ je zkratkou pro $\neg$`(`t_1 `=` t_2`)`. Rovnost má ve formulích, na rozdíl od logické ekvivalence, nejvyšší prioritu, proto je nutné výrazy pečlivě závorkovat.

Kvantifikátory jsou zapisovány jako $\forall$`x. P x` a $\exists$`x. P x`. Použití více stejných kvantifikátorů za sebou lze zkrátit do tvaru $\forall$`x y z. R x y z`.

2.5 Dokazování v Isabelle/HOL

Po definování požadovaných objektů je teorie připravena k formulaci tvzení a následné verifikaci jejich důkazů, případně k jejich asistovanému dokazování. Tím, že verifikátor s použitím důkazových metod rozkládá tvrzení na dílčí podcíle zobrazované v reálném čase, je možné jej použít jako inspiraci pro hledání důkazu.

2.5.1 Přirozená dedukce

K dokazování formulí HOL používá Isabelle/HOL kalkulus přirozené dedukce. Pro spojky a kvantifikátory HOL jsou tak implementována pravidla jejich zavedení a eliminace. V přiložené formalizaci se explicitní usuzování na úrovni formulí HOL takřka nevyskytuje, proto pro více detailů k syntaxi a použití kalkulu viz [17, kap. 5].

2.5.2 Formulace tvrzení

Tvrzení jsou vyjádřena formulemi HOL z oddílu 2.4.9. Jedná se tedy o termy typu `bool`. Jako příklad uvažme následující lemma z teorie `NaturalNumbers` o přičítání nuly zleva předložené již ve své formalizované podobě:

Formalizace 2.5.1. *Lemma o neutralitě nuly vůči přičítání zleva*

```
lemma NeutralityOfZeroFromLeft [simp]:
"∀x. (Zero ⊕ x) = x"
```

Tvrzení je uvedeno klíčovými slovem **lemma**, případně **theorem** či **corollary**. Tato slova jsou v kódu volně zaměnitelná a jejich užití záleží na potřebě uživatele odlišit významnost formalizovaných tvrzení.

Dále následuje nepovinné, avšak doporučené pojmenování, kterým lze později k tvrzení odkazovat. Atribut `simp` v hranatých závorkách předává

programu informaci, že tvrzení lze v dalších důkazech používat jako *simplifikační pravidlo*, tedy pro přepis výrazu `Zero` $\oplus$ `x` na výraz `x`. Tento atribut je – stejně jako jméno tvrzení – nepovinný. V rámci budování teorie jej však lze obecně doporučit.

Po uvedení hlavičky následuje tvrzení samotné. Jeho uvedením se verifikátor přepne do *důkazového režimu* a vygeneruje cíl, který má být dokázán. Důkazový režim je ukončen úspěšným dokončením důkazu, bez nějž nelze pokračovat ve formalizaci.

Důkaz tvrzení může být veden dvěma způsoby, a to buď postupným rozkládáním aktuálního cíle na jednodušší automaticky dokazatelné podcíle pomocí *emulace taktických skriptů*, nebo naopak odvozením cíle z již dokázaných tvrzení vedením *strukturovaného důkazu*. Oba způsoby se odlišují především čitelností a délkou výsledného kódu a jsou popsány níže.

2.5.3 Emulace taktických skriptů

Každý krok důkazu je verifikován aplikací některé z *důkazových metod m*, a to příkazem **apply** *m*. Metoda *m* je automaticky aplikována na první podcíl, který buď dokáže, nebo transformuje na podcíl nový, a to v závislosti na použité metodě.

Podcíle jsou automaticky číslovány a jejich pořadí lze měnit příkazy **prefer** ***n*** a **defer** ***n***, kde ***n*** je číslo podcíle. Důkaz končí dokázáním všech podcílů, kdy je na výstupu verifikátoru zobrazena zpráva „`No subgoals!`“, a následným uvedením příkazu **done**.

Aplikací důkazové metody dochází k volání příslušného taktického skriptu v jazyce ML, který vrací výsledek na výstup verifikátoru. Důkazové metody v následujícím přehledu byly použity při formalizaci logiky BL. Jejich stručný popis vychází z dokumentace v [21], popřípadě [17].

simp volá implementovaný nástroj Simplifier, který zkouší opakovaně transformovat cíl podle *simplifikačních* (či *přepisovacích*) *pravidel* dostupných v importovaných teoriích, mezi dříve dokázanými tvrzeními s atributem `[simp]` nebo mezi předchozími kroky důkazu. Transformace spočívá v přepisování termů podle rovnosti v simplifikačním pravidle, a to zleva doprava. Metoda selhává, pokud po přepsání zůstane term nezměněn.

blast je metoda z implementovaného balíku nástrojů pro dokazování v klasické logice. Je vhodná například pro dokazování formulí predikátové logiky či teorie množin.

auto kombinuje klasické dokazování a simplifikaci. Metoda je vhodná v situacích, kdy je třeba dokázat velké množství triviálních podcílů. Ty, na kterých skript selže, zůstávají mezi cíli k dokázání.

metis odkazuje na implementovaný automatický dokazovací nástroj pro prvořádovou logiku s rovností (viz [14]). V kombinaci s nástrojem *Sledgehammer*, generujícím argumenty pro metis, se jedná o jedinou metodu použitou pro verifikaci objektových důkazů v kalkulu BL.

rule ***a*** aplikuje pravidlo ***a*** na aktuální cíl. Varianta **erule** aplikuje pravidlo v opačném směru.

subgoal_tac ***A*** přidá k aktuálnímu podcíli předpoklad ***A*** a vygeneruje tuto skutečnost jako nový podcíl.

induct_tac ***d*** generuje podcíle podle induktivní definice objektu ***d***.

Pro větší přehlednost a stylistickou úpravu kódu podporuje Isabelle/HOL zkratky pro některé sekvence příkazů. Jejich kompletní výčet je uveden v [21, odd. A.1.2]. Například zakončení důkazu pomocí **apply** ***m*** **done** může být nahrazeno příkazem **by** ***m***, jak je vidět na následujícím příkladu důkazu lemmatu z formalizace 2.5.1 emulací taktických skriptů:

Formalizace 2.5.2. *Důkaz emulací taktických skriptů*

```
lemma "∀x. (Zero ⊕ x) = x"
apply auto
 apply (induct_tac x)
by simp+
```

Metoda `auto` přeloží kvantifikátor ∀ na jeho interní variantu ⋀ sloužící pro práci s libovolnou hodnotou [17, odd. 5.9]. Metoda `induct_tac x` vygeneruje podcíle podle konstruktorů `x` – to je díky typové inferenci rozpoznáno jako objekt typu `Natural`, a podcíle jsou tedy rozložením tvrzení na případy `x = Zero` a `x = Suc n`. Ty jsou pak dokázány metodou `simp`, jejíž opakované použití je zkráceno znamínkem `+`.

2.5.4 Strukturovaný důkaz

Druhou možností verifikace tvrzení je sestavení strukturovaného důkazu, který je vymezen příkazy **proof** a **qed** uvozujícími tělo důkazu. To sestává z příkazů **fix** `x` pro fixování proměnné, **assume** `A` pro deklaraci předpokladu `A`; dále konstrukce **from** `B` **have** `C` řetězí dílčí tvrzení, přičemž `C` musí být dokázáno, a to buď emulací taktických skriptů, nebo opět strukturovaným důkazem. Příkaz **this** zastupuje poslední dokázaný krok. Důkaz je pak dokončen příkazem **from** `D` **show** `E`, kdy verifikátor z řetězce dílčích tvrzení vygeneruje pravidlo, jímž dokáže aktuální cíl důkazu.

Jednotlivé kroky důkazu lze pojmenovávat tak, aby k nim bylo možné referovat později. Strukturovaný důkaz je možné číst jako matematický text, čemuž napomáhá i pojmenování jednotlivých příkazů. Navíc jsou zavedena synonyma některých konstrukcí. Za všechny uveďme, že příkaz **from** `A` je zkratkou za **note** `A` **then** a příkaz **then** pak zastupuje **from this**. Příkaz **have** `A` **using** `B` je ekvivalentní zápisu **from** `B` **have** `A`.

Jako příklad uveďme opět důkaz lemmatu z formalizace 2.5.1, tentokrát však v jeho strukturované podobě:

Formalizace 2.5.3. *Strukturovaný důkaz neutrality nuly vůči přičítání zleva*

```
lemma NeutralityOfZeroFromLeft [simp]:
  "∀x. (Zero ⨁ x) = x"
proof (auto, induct_tac)
  show ZeroCase: "(Zero ⨁ Zero) = Zero"
    by simp
  fix k
  assume IH: "(Zero ⨁ k) = k"
  have 2: "(Zero ⨁ (Suc k)) = Suc (Zero ⨁ k)"
    by simp
  from 2 have 3: "Suc (Zero ⨁ k) = Suc k"
    using IH by simp
  then show "(Zero ⨁ (Suc k)) = Suc k"
    using 2 and 3 by simp
qed
```

Metody v parametru příkazu **proof** jsou aplikovány při iniciaci důkazu – `auto` přeloží kvantifikátor a `induct_tac` rozdělí hlavní cíl na dva podcíle podle induktivní definice `x`. První z nich je dokázán simplifikací podle definice $\bigoplus$. Zbývá tedy dokázat indukční krok. Ten je na výstupu zobrazen jako podcíl:

```
⋀x Natural.
  (Zero ⨁ Natural) = Natural ⟹
  (Zero ⨁ 𝒮 Natural) = 𝒮 Natural.
```

V této formuli je jménem `Natural` automaticky pojmenována proměnná typu `Natural`. Důkaz pak pokračuje fixováním libovolného `k` a předpokládáním indukčního kroku, který odpovídá levé straně metaimplikace v podcíli. Předpoklad je pro pozdější použití pojmenován jako `IH`.

Další kroky lze číst bez podrobnějšího popisu. Tvrzení za příkazem **show** pak odpovídá závěru implikace v hlavním cíli. Po aplikaci tohoto příkazu je tedy vytvořeno pravidlo spojující předpoklad `IH` se závěrem pomocí $\Longrightarrow$, jímž lze dokázat podcíl, a protože už žádný další nezbývá, důkaz je uzavřen příkazem **qed**, verifikátor ukončí důkazový mód a je opět připraven pro deklaraci definic či formulaci dalšího tvrzení.

Popsané kostrukce a nástroje Isabelle/HOL dostačují pro formalizaci logiky BL. Případy, které v této kapitole nebyly uvedy, jsou rozebrány operativně v popisu formalizace. Cílem kapitoly bylo ukázat, že objekty logiky BL lze v Isabelle/HOL zachytit podobně jako přirozená čísla v ukázkové teorii `NaturalNumbers`.

V následující kapitole je předložen stručný popis logiky BL, a to s důrazem na její syntax, která je předmětem formalizace.

3 Logika BL

Logika BL definovaná profesorem Petrem Hájkem v [12], je případem výrokové matematické fuzzy logiky. V této kapitole o ní budeme vykládat podle [6, kap. 1].

3.1 Matematická fuzzy logika

Matematická fuzzy logika je rozšířením klasické dvouhodnotové logiky na systém L pravdivostních hodnot, nejčastěji z reálného intervalu $[0, 1]$. Fuzzy logiky jsou typické kladením jistých podmínek na vícehodnotovou sémantiku výrokových spojek, které jsou navrhovány za účelem sestavení výrokového a predikátového kalkulu.

Každá n-ární výroková spojka c je sémanticky interpretována pravdivostní funkcí $F_c\colon L^n \to L$ – pravdivostní hodnota formule $c(\varphi_1, \ldots, \varphi_n)$ je definována funkcí $F_c(x_1, \ldots, x_n)$, kde x_i je pravdivostní hodnota podformule φ_i pro $i \in \{1, \ldots, n\}$. [6, odd. 1.1].

Krajní pravdivostní hodnoty 0 a 1 reprezentují *úplnou nepravdivost* a *úplnou pravdivost* odpovídající klasické dvouhodnotové logice, vnitřní hodnoty z intervalu $(0, 1)$ pak *„částečnou pravdivost“*,[19] která je předmětem zkoumání fuzzy logik. Na intervalu $[0, 1]$ je zavedena relace $\leq$ z reálných čísel reprezentující pravdivostní sílu tvrzení – to je tím pravdivější, čím je vyšší jeho pravdivostní hodnota.

3.2 Výroková fuzzy logika

3.2.1 Interpretace spojek pomocí funkce spojité t-normy

Výroková fuzzy logika obvykle klade určité podmínky na pravdivostní funkce spojek. Mezi nejběžnější patří následující podmínky na spojku konjunkce:

- *Komutativita:* $x * y = y * x$
- *Asociativita:* $(x * y) * z = x * (y * z)$
- *Monotonie:* když $x \leq x'$ a $y \leq y'$, pak $x * y \leq x' * y'$
- *Jednotka:* $x * 1 = x$
- *Spojitost:* Funkce $*$ je spojitá na $[0, 1]^2$.

[19] *Částečná pravdivost* je zde (stejně jako v [6]) použita ryze jako technický termín označující stupeň jisté kvality přiřazované tvrzením.

Podmínky odpovídají požadavkům na očekávané vlastnosti konjunkce a jsou otázkou *designu* [12, s. 27]. Zde uvedený výčet tedy klade požadavky na komutativitu a asociativitu kojunkce, dále na monotonní chování vůči změně jednoho z argumentů. Dále výraz s pravdivostí 1 nemá mít vliv na pravdivost tvrzení, s nímž je v konjunkci. Požadavek na spojitost $*$ pak plyne z intuice, že nekonečeně malá změna pravdivostní hodnoty argumentu nemá způsobit výraznější změnu hodnoty celého výrazu. Další vlastnosti $*$, jako je neutralita hodnoty 0, plynou již z vlastností výše uvedených, popřípadě by byly příliš omezující.[20]

Funkce $*$ splňující první čtyři podmínky se nazývá *triangulární norma*, zkráceně *t-norma*.

Definice 3.2.1. [6, odd. 1.1] *Binární funkce* $*\colon [0,1]^2 \to [0,1]$ *se nazývá* t-norma *(nebo* triangulární norma*), pokud je komutativní, asociativní, monotonní a* 1 *je jejím jednotkovým prvkem.*

T-norma $*$ interpretuje takzvanou *silnou konjunkci* (&). Nyní uvedeme interpretace ostatních spojek výrokové fuzzy logiky založené na spojitých t-normách.

Věta 3.2.1. [6, odd. 1.1] *Pro každou spojitou t-normu* $*$ *existuje jednoznačně určená binární operace* $\Rightarrow_*$ *na* $[0,1]$ *(zvaná* reziduum funkce $*$*) taková, že pro všechna* $x, y, z \in [0,1]$ *platí:*

$$z * x \leq y \quad \textit{iff} \quad z \leq x \Rightarrow_* y.$$

Spojka implikace ($\to$) je interpretována právě operací $\Rightarrow_*$, jejíž některé vlastnosti uvedeme v následující větě:

Věta 3.2.2. [6, věta 1.1.9] *Pro spojitou t-normu* $*$*, její reziduum* $\Rightarrow_*$ *a každé* $x, y \in [0,1]$ *platí:*

1. $x \Rightarrow_* y = 1 \quad \textit{iff} \quad x \leq y$
2. $0 \Rightarrow_* y = 1$
3. $1 \Rightarrow_* y = y$
4. $\min\{x, y\} = x * (x \Rightarrow_* y)$
5. $\max\{x, y\} = \min\{(x \Rightarrow_* y) \Rightarrow_* y, (y \Rightarrow_* x) \Rightarrow_* x\}$

[20] viz [6, s. 4]

Ve výrokové fuzzy logice operace minima z bodu *4* věty 3.2.2 interpretuje další spojku, takzvanou *slabou konjunkci* ($\wedge$), operace maxima z bodu *6* pak interpretuje *slabou disjunkci* ($\vee$).

Negace je interpretována funkcí $\neg_* x = x \Rightarrow_* 0$. Poslední spojka, *ekvivalence* ($\leftrightarrow$), je pak interpretována operací *bireziduum* definovanou následovně:

Definice 3.2.2. [6, s. 8] *Pro spojitou t-normu* $*$ *a všechna* $x, y \in [0,1]$ *je definována operace* bireziduum $\Leftrightarrow_*$ *takto:*

$$x \Leftrightarrow_* y = \min\{x \Rightarrow_* y, y \Rightarrow_* x\}$$

Jak je patrné, pro definici spojek logiky spojitých t-norem je podoba & stěžejní. Volba operace $*$ tak určuje podobu celé logiky.

3.2.2 Prominentní t-normy

V následující definici jsou uvedeny příklady tří prominentních spojitých t-norem a jejich reziduí:

Definice 3.2.3. [6, odd. 1.1] *Příkladem spojitých t-norem a jejich reziduí jsou:*

pojmenování	t-norma	reziduum
Gödelova	$x *_G y = \min\{x, y\}$	$x \Rightarrow_G y = y$
produktová	$x *_\Pi y = x \cdot y$	$x \Rightarrow_\Pi y = y/x$
Łukasiewiczova	$x *_L y = \max\{x + y - 1, 0\}$	$x \Rightarrow_L y = 1 - x + y$

Každou t-normu lze vyjádřit pomocí těchto tří t-norem [6, věta 1.1.7(5)]. *Reziduální negace* prominentních t-norem vychází následovně:

Definice 3.2.4. [6, s. 8]

$$\neg_L x = 1 - x \qquad \neg_G x = \neg_\Pi x = \begin{cases} 1 \text{ pro } x = 0 \\ 0 \text{ pro } x > 0 \end{cases}$$

Definice 3.2.5. [6, def. 1.1.12] *Pro spojitou t-normu* $*$ *definujeme* t-algebru *této t-normy jako algebru*

$$[0,1]_* = \langle [0,1], *, \Rightarrow_*, \min, \max, 0, 1 \rangle,$$

kde min *a* max *jsou minimum a maximum v běžném smyslu na reálných číslech. Operace* $\Leftrightarrow_*$ *a* $\neg_*$ *jsou pak odvozeny podle výše uvedených definic 3.2.2 a 3.2.4.*

Pro každou množinu K spojitých t-norem značíme odpovídající algebru $\mathbb{K}$ *a naopak.*

3.2.3 Syntax a sémantika

Syntax a sémantika logiky spojitých t-norem jsou definovány podle definice 1.1.13 v [6] následovně:

Definice 3.2.6. [6] Jazyk $\mathcal{L}$ výrokové fuzzy logiky L_K *množiny* K *spojitých t-norem sestává z:*

- *výrokových proměnných p, q, r... z množiny Var,*

binárních spojek

- $\&$ *(silná konjunkce),*
- $\rightarrow$ *(implikace),*
- $\wedge$ *(slabá konjunkce),*
- $\vee$ *(slabá disjunkce),*
- $\leftrightarrow$ *(ekvlivalence)*

unární spojky

- $\neg$ *(negace)*

a pravdivostních konstant

- $\bar{0}$ *a* $\bar{1}$.

Formule jazyka $\mathcal{L}$ *je pak utvořena způsobem jako v klasické výrokové logice.*[21] *Dále pro všechna* $n \in \mathbb{N}$ *a formuli* φ *zavedeme zkrácený zápis:*

$$\varphi^0 \equiv_{df} \bar{1}$$
$$\varphi^{n+1} \equiv_{df} \varphi^n \,\&\, \varphi.$$

Ohodnocení výrokových proměnných je zobrazením $e\colon \mathrm{Var} \rightarrow [0,1]$; to je jednoznačně rozšiřitelné na zobrazení e_* z množiny formulí $\mathcal{L}$ do $[0,1]$, a to rekurzivní definicí ohodnocení v [6, def. 1.1.3]. Podle této definice, definic $\Rightarrow_*$ a $\neg_*$ a věty 3.2.2 lze považovat spojky $\&$, $\rightarrow$ a $\bar{0}$ za základní – ostatní jsou z těchto odvozeny.

[21] jako rozšíření definice 1.1.1 v [20] o spojku $\&$

Definice 3.2.7. [6, s. 14] Odvozené spojky *logiky spojitých t-norem lze definovat pomocí spojek* $\&$, $\rightarrow$ *a* $\bar{0}$ *takto:*

$$\begin{aligned} \varphi \wedge \psi &\equiv_{df} \varphi \,\&\, (\varphi \rightarrow \psi)) \\ \varphi \vee \psi &\equiv_{df} ((\varphi \rightarrow \psi) \rightarrow \psi) \wedge ((\psi \rightarrow \varphi \rightarrow \varphi)) \\ \varphi \leftrightarrow \psi &\equiv_{df} (x \rightarrow y) \,\&\, (y \rightarrow x) \\ \neg\varphi &\equiv_{df} \varphi \rightarrow \bar{0} \\ \bar{1} &\equiv_{df} \neg\bar{0} \end{aligned}$$

Definice 3.2.8. [6, def. 1.1.4] *Nechť* K *je množinou spojitých t-norem. Pak formule* φ *je* tautologií logiky L_K*, pokud* $e_*(\varphi) = 1$ *pro každou t-normu* $* \in K$ *a každé ohodnocení* e_**.*

Nyní přistupme k vymezení logiky BL:

Definice 3.2.9. [6, def. 1.1.9] *Logiky určené prominentními t-normami z definice 3.2.3 jsou po řadě nazývány Gödelova, produktová a Łukasiewiczova logika (značeno* G, Π *a* Ł*). Logika všech spojitých t-norem se nazývá* BL *(z angl. Basic Logic).*

3.3 Axiomatika logiky spojitých t-norem

Definice 3.3.1. [6, odd. 1.2] Hilbertovský kalkulus logiky BL *je dán axiomy*

$(BL1)$	$(\varphi \rightarrow \psi) \rightarrow ((\psi \rightarrow \chi) \rightarrow (\varphi \rightarrow \chi))$
$(BL4)$	$(\varphi \,\&\, (\varphi \rightarrow \chi)) \rightarrow (\chi \,\&\, (\chi \rightarrow \varphi))$
$(BL5a)$	$((\varphi \,\&\, \psi) \rightarrow \chi) \rightarrow (\varphi \rightarrow (\psi \rightarrow \chi))$
$(BL5b)$	$((\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \,\&\, \psi) \rightarrow \chi))$
$(BL6)$	$((\varphi \rightarrow \psi) \rightarrow \chi) \rightarrow (((\psi \rightarrow \varphi) \rightarrow \chi) \rightarrow \chi)$
$(BL7)$	$\bar{0} \rightarrow \varphi$

a odvozovacím pravidlem modus ponens

$$\frac{\varphi,\ \varphi \rightarrow \psi}{\psi}.$$

Tento systém je korektní a úplný vůči výše popsané sémantice logiky BL. Axiom $(BL1)$, *sufixace*, uvádí implikaci $\rightarrow$ do vztahu s uspořádáním pravdivostních hodnot $\leq$ a zavádí tak do syntaxe její tranzitivitu. Axiom $(BL4)$, zvaný též *axiom divizibility*, pak zachycuje komutativitu slabé konjunkce $\wedge$. Axiomy $(BL5a)$ a $(BL5b)$ odpovídají podmínce *reziduace* a $(BL6)$ podmínce

prelinearity.[22] Axiom $(BL7)$ vyjadřuje princip *ex falso quodlibet* z klasické logiky [6, s. 15].
Číslování axiomů odpovídá původnímu značení v [12], kde byly v rámci axiomatiky navíc uvedeny následující dva axiomy:

$(BL2)$ $(\varphi \,\&\, \psi) \to \varphi$

$(BL3)$ $(\varphi \,\&\, \psi) \to (\psi \,\&\, \varphi)$

Axiomy $(BL2)$ a $(BL3)$ jsou klasickými axiomy konjunkce. Jejich závislost na axiomatice z definice 3.3.1 byla dokázána později v [7] a [8].[23] Oba axiomy jsou nyní v [6] zařazeny mezi teorémy logiky BL.

Logika BL je základní ve smyslu možnosti jejího axiomatického rozšíření na logiky prominentních t-norem, tedy logiky G, Π a Ł, a to těmito axiomy:

(G) $\varphi \to (\varphi \,\&\, \varphi)$

$(Ł)$ $\neg\neg\varphi \to \varphi$

(Π) $\neg\varphi \vee ((\varphi \to \varphi \,\&\, \psi) \to \psi)$

Výsledné logiky jsou tedy definovány jako $G = \mathrm{BL} + (G)$, $Ł = \mathrm{BL} + (Ł)$ a $\Pi = \mathrm{BL} + (\Pi)$. Rozšířením axiomatiky BL o *zákon vyloučeného třetího*, $\varphi \vee \neg\varphi$, získáme systém klasické výrokové logiky.[24]

Pojem důkazu v předložených kalkulech je pak definován klasicky:

Definice 3.3.2. [6, odd. 1.2] Důkazem formule φ z množiny formulí Γ *je posloupnost formulí* $\varphi_1, \ldots, \varphi_n$ *taková, že* $\varphi_n = \varphi$ *a pro každou formuli* φ_i *z posloupnosti platí, že (i) je instancí axiomu BL, nebo (ii) je prvkem* Γ*, nebo (iii)* $\exists j, k \leq i$ *taková, že* φ_i *vznikla aplikací pravidla modus ponens na formule* φ_j *a* φ_k*.*

Pokud existuje důkaz φ *z množiny* Γ*, říkáme, že* φ je dokazatelná z Γ *a píšeme* $\Gamma \vdash \varphi$*.*

3.4 Vlastnosti BL

Následující tvrzení platí o logice BL a jejich verifikace je součástí formalizace – důkazy jsou tak předloženy až v kapitole 4 a vycházejí ze svých formalizovaných verzí. Výběr vlastností BL je podřízen především záměru formalizovat větu o lokální dedukci.

V první řadě uveďme jednu ze základních vlastností relace dokazatelnosti, a to *monotonii*:

[22] Divisibilita, reziduace a prelinearita jsou vlastnostmi algebraické struktury zachycující sémantiku logiky BL. Viz [6, def. 1.3.1].

[23] Oba důkazy jsou formalizovány v příloze `BL_theorems.thy`

[24] pro podrobnější popis rozšíření BL viz [6, odd. 1.2]

Lemma 3.4.1. *Nechť φ je formule BL a Γ, Δ množiny formulí BL. Pak platí, že*

$$\Gamma \vdash_{BL} \varphi \implies \Gamma \cup \Delta \vdash_{BL} \varphi.$$

Instancí tohoto lemmatu je pak jednoduché tvrzení o dokazatelnosti z prázdné množiny:

Lemma 3.4.2. *Nechť φ je formulí BL a Γ množinou formulí BL. Pak platí následující:*

$$\vdash_{BL} \varphi \implies \Gamma \vdash_{BL} \varphi.$$

Další použitou vlastností je teorém kalkulu BL, který slouží k přepisování formulí obsahujících výrazy tvaru φ^n:

Lemma 3.4.3. *Nechť φ je formulí BL a m, n jsou přirozenými čísly. Pak platí následující:*

$$\vdash_{BL} (\varphi^n \,\&\, \varphi^m) \leftrightarrow \varphi^{n+m}.$$

Logika BL je úplná [6]. Jednoduchým důsledkem této vlastnosti je následující lemma o odvoditelnosti násobné konjunkce, které je však ve formalizaci dokázáno pouze syntaktickými prostředky:

Lemma 3.4.4. *Nechť φ je formulí BL, Γ množinou formulí a n přirozeným číslem. Pak platí, že*

$$\Gamma \cup \{\varphi\} \vdash_{BL} \varphi^n.$$

A konečně můžeme uvést větu o lokální dedukci z [6]:

Věta 3.4.1. *(O lokální dedukci) Nechť φ a ψ jsou formulemi BL, Γ množinou formulí. Pak platí, že:*

$$\Gamma \cup \{\varphi\} \vdash_{BL} \psi \iff \exists n \in \mathbb{N}.\ \Gamma \vdash_{BL} \varphi^n \to \psi.$$

4 Formalizace BL

4.1 Struktura teorií

Formalizace logiky BL je rozdělena do několika souborů. Formule BL, násobná konjunkce φ^n a relace dokazatelnosti jsou formalizovány s pomocí teorií *Main* a *String* v teorii *BasicLogic*. Ta je následně importována do teorie *BL_theorems*, obsahující důkazy v kalkulu logiky. Obě teorie jsou pak importovány do teorie *BL_LocalDeduction* s důkazem věty o lokální dedukci.

4.2 Formalizace syntaxe

4.2.1 Formule

Formule logiky BL je reprezentována datovým typem *formulaBL* definovaným v jazyce BL podle [6]:

Formalizace 4.2.1. *Formalizace formule BL*

```
datatype formulaBL =
  Var char (".")
| Const_0
| Conjuction_BL_strong formulaBL formulaBL (infixr "&BL" 50)
| Implication_BL formulaBL formulaBL (infix "⟶BL" 35)
```

Argumentem konstruktoru atomických formulí *Var* je objekt typu *char* (písmeno) z teorie *String*. Písmena jsou v jazyce Isar vyhrazena pro jména proměnných HOL, od kterých je potřeba atomické formule odlišit. Zvolena byla prefixová notace pomocí tečky - atomy jsou tak vyjádřeny ve tvaru *.p*, *.q*, *.r* atd.

Další konstruktor *Const_0* odpovídá konstantě $\bar{0}$. Konstruktory základních spojek *Conjuction_BL_strong* a *Implication_BL* mají dva argumenty typu *formulaBL* a je pro ně zavedena infixová notace s uvedenou prioritou závorkování.

Obdobně jako v případě písmen jsou spojky klasické logiky v Isar vyhrazeny pro formule HOL. Pro odlišení jsou proto všechny formalizované spojky BL indexovány názvem logiky.

Další spojky BL jsou definovány pomocí **definition** jako ekvivalentní zápisy některých objektů typu *formulaBL*. Konstantu $\bar{1}$ pak zachycuje definice[25] objektu *Const_1* typu *formulaBL*:

[25]Použitá definice konstanty $\bar{1}$ z [12] na rozdíl od definice v [6] používá pouze základní spojky.

Formalizace 4.2.2. *Formalizace konstanty* $\bar{1}$

```
definition Const_1 :: "formulaBL"
where
  "Const_1 ≡ (Const_0 ⟶BL Const_0)"
```

Odvozené spojky BL jsou typu *formulaBL* ⇒ *formulaBL* ⇒ *formulaBL* pro spojky binární, případně *formulaBL* ⇒ *formulaBL* pro spojky unární.

Pro typové konstruktory odvozených spojek jsou definovány názvy *binary_conn* a *unary_conn* pomocí příkazu **type_synonym**, jak je vidět na následující formalizaci:

Formalizace 4.2.3. *Přejmenování typů*

```
type_synonym unary_conn = "formulaBL ⇒ formulaBL"

type_synonym binary_conn = "formulaBL ⇒ formulaBL ⇒
formulaBL"
```

S využitím synonym jsou odvozené spojky BL formalizovány takto:

Formalizace 4.2.4. *Odvozené spojky BL*

```
definition Conjuction_BL_weak :: binary_conn (infixr "∧BL" 50)
where
  "Conjuction_BL_weak φ ψ ≡ (φ &BL (φ ⟶BL ψ))"

definition Disjunction_BL_weak :: binary_conn (infixr "∨BL" 50)
where
  "Disjunction_BL_weak φ ψ ≡
    ((φ ⟶BL ψ) ⟶BL φ) ∧BL ((ψ ⟶BL φ) ⟶BL φ)"

definition Equivalence_BL :: binary_conn (infixr "⟷BL" 30)
where
  "Equivalence_BL φ ψ ≡ ((φ ⟶BL ψ) &BL (ψ ⟶BL φ))"

definition Negation_BL :: unary_conn ("¬BL")
where
  "Negation_BL φ ≡ φ ⟶BL Const_0"
```

Zvláště je definována násobná konjunkce – φ^n. Z typového hlediska se jedná o objekt kombinující typ formulí a typ přirozených čísel tak, že výsledný objekt je typu formulí. Vzhledem k návratové hodnotě je formalizován jako binární rekurzivní funkce HOL podle definice 1.1.13 v [6]:

Formalizace 4.2.5. *Násobná konjunkce formulí BL*

```
fun MultiConj_BL :: "formulaBL ⇒ nat ⇒ formulaBL" (infixr
"^BL" 60)
where
  "Phi_n φ 0 = Const_1"
| "Phi_n φ n = (φ &BL (Phi_n φ (n - 1)))"
```

4.2.2 Relace dokazatelnosti

Relace dokazatelnosti logiky BL je binární relací na množinách formulí a formulích a je dána axiomatikou BL a pravidlem modus ponens (viz odd. 3.3). V literatuře uváděné (viz [12], [6]) definice relace dokazatelnosti skrz pojem *důkazu* jakožto posloupnosti se pro formalizaci v Isabelle jeví jako nevhodná. Důvodem je její obtížné zachycení v teorii typů, resp. pro Isabelle přílišná složitost výsledného kódu.
V Isabelle by se jevilo přirozené zachytit důkaz podle definice 3.3.2 jako objekt typu `list`, jehož prvky by splňovaly uvedené vlastnosti, o čemž by bylo rozhodovala implementovaná funkce do **2**, tedy metapredikát na doméně množin formulí a formulí.

Při formalizaci se však ukázalo, že implementace rekurzivní definice není tak přímočará jako její poloformální zápis v textu. Zásadním problémem je skutečnost, že posloupnost formulí takřka nezachycuje strukturu důkazu a při jednotlivých iteracích rekurzivního průchodu posloupností musí algoritmus zpětně prohledávat už prošlé formule a hledat mezi nimi premisy pravidla modus ponens.

Pro důkazy v kalkulu bylo navíc záměrem docílit jednoduchého zápisu důkazů pro snadnou obsluhu programu potenciálními uživateli, čímž lze vyloučit zápisu důkazu jako stromu apod.[26]

Dále měla formalizace dokazatelnosti zachytit tuto relaci nejen pro ověřování důkazů v daném kalkulu, ale zejména pro verifikaci metadůkazů tvrzení *o této relaci*, a to mimo jiné i těch induktivních podle složitosti

[26]Představu o implementaci důkazu jako stromu či posloupnosti a množství použitých závorek při jeho zápisu lze ponechat na čtenáři.

odvození. Deklarace komplexní rekurzivní funkce neumožňuje automaticky generovat pravidla indukce pro tento objekt.

Vzhledem k výše uvedeným skutečnostem se ukázalo být vhodným řešením definování dokazatelnosti skrze *důkazový uzávěr množiny formulí* popsaný v [9, odd. 1.1].

Definice 4.2.1. *Důkazovým uzávěrem množiny* Γ *je nejmenší množina CnBL* Γ *taková, že jejími prvky jsou (i) instance axiomů BL1–BL7, (ii) formule z množiny* Γ*, (iii) pokud jsou formule* φ *a* $\varphi \rightarrow_{BL} \psi$ *prvky CnBL* Γ*, pak je i* ψ *prvkem CnBL* Γ*.*

Formalizován je pak pomocí **inductive_set** takto:

Formalizace 4.2.6. *Důkazový uzávěr CnBL*

```
inductive_set CnBL :: "formula set ⇒ formula set" for Γ
where
  BL1: "((φ ⟶BL ψ) ⟶BL ((ψ ⟶BL χ) ⟶BL (φ ⟶BL χ)))
        ∈ CnBL Γ"
| BL4: "((φ &BL (φ ⟶BL ψ)) ⟶BL (ψ &BL (ψ ⟶BL φ)))
        ∈ CnBL Γ"
| BL5a: "((φ &BL ψ ⟶BL χ) ⟶BL (φ ⟶BL (ψ ⟶BL χ)))
        ∈ CnBL Γ"
| BL5b: "(((φ ⟶BL (ψ ⟶BL χ)) ⟶BL
          ((φ &BL ψ) ⟶BL χ))) ∈ CnBL Γ"
| BL6: "(((φ ⟶BL ψ) ⟶BL χ) ⟶BL
          (((ψ ⟶BL φ) ⟶BL χ) ⟶BL χ)) ∈ CnBL Γ"
| BL7: "(Const_0 ⟶BL φ) ∈ CnBL Γ"
| Assumption: "φ ∈ Γ ⟹ φ ∈ CnBL Γ"
| MP: "ψ ∈ CnBL Γ ∧ (ψ ⟶BL φ) ∈ CnBL Γ ⟹ φ ∈ CnBL Γ"
```

Symboly φ, ψ a χ v konstruktorech `BL1-BL7` pojmenovávají proměnné, za něž může být dosazena libovolný objekt typu `formulaBL`, respektive kdykoliv je formule ξ dosazena za proměnnou φ do schématu $BL(\varphi)$, Isabelle vyhodnotí výraz BL(ξ) $\in$ CnBL Γ jako pravdivý.
Stěžejní pro definici uzávěru `CnBL` jsou pak podmíněné konstruktory `Assumption` a `MP`. Jejich čtení je z hlediska běžného matematického zápisu intuitivní. Samotná relace dokazatelnosti je pak definována skrze relaci náležení:

Definice 4.2.2. *Formule* φ *je v logice BL* dokazatelná z množiny formulí Γ *(píšeme* $\Gamma \vdash_{BL} \varphi$*) právě tehdy, když* φ *náleží do důkazového uzávěru CnBL* Γ*. Relace* $\vdash_{BL}$ *je* relací dokazatelnosti *logiky BL.*

Formalizace relace je pak implementována jako definice HOL následovně:

Formalizace 4.2.7. *Relace dokazatelnosti BL*

```
definition ProvRelBL :: "formula set ⇒ formula ⇒ bool"
(infixr "⊢BL" 10)
where
  "ProvRelBL Γ φ ≡ φ ∈ CnBL Γ"
```

V tuto chvíli již je předložená teorie dostatečně obsáhlá pro formalizaci teorémů BL jakožto tvrzení metalogiky HOL.

4.3 Důkazy v kalkulu BL

Ve srovnání s definicí relace dokazatelnosti v kalkulech hilbertovského typu, která se objevuje v literatuře ([6], [12]), není podle definice 4.2.2 tento vztah mezi množinou formulí a formulí prokazován instanciací existenčního kvantifikátoru objektem s požadovanou strukturou (důkazem – posloupností), nýbrž zkoumáním důkazového uzávěru skrze relaci náležení $\in$, jejíž definice již je implementována v importované knihovně `Main`.

Uvažme tedy například tvrzení, že konstanta $\bar{1}$ je teorémem BL, tedy že je dokazatelná z prázdné množiny předpokladů:

Věta 4.3.1. $\vdash_{BL} \bar{1}$

Důkaz Z definice 4.2.2 a definice odvozených spojek dostaneme postupně ekvivalentní tvrzení $\bar{1} \in \mathrm{CnBL}\,\{\}$ a $\bar{0} \to \bar{0} \in \mathrm{CnBL}\,\{\}$, přičemž druhé z nich je instancí konstruktoru $\mathrm{CnBL}\,\Gamma$, tedy je prvkem uzávěru.

■

Verifikace důkazu v kalkulu může být provedena více způsoby. Při každém z nich však v některém z kroků dojde k aplikaci definice 4.2.2. Protože jsou podle této definice zápisy $\Gamma \vdash_{BL} \varphi$ a $\varphi \in \mathrm{CnBL}\,\Gamma$ ekvivalentí, můžeme definovanou relaci $\vdash_{BL}$ považovat za redundantní, v kódu ji ponechat pro případné budoucí použití a u dokazovaných formulí rovnou ověřovat jejich náležení do důkazového uzávěru.

Formalizovaný důkaz formule BL může být veden jako zpětný skrze emulaci taktických skriptů, což je vhodné pro instance již dokázaných tvrzení a kratší důkazy z definic, jak je vidět na formalizaci důkazu tvrzení 4.3.1:

Formalizace 4.3.1. *Lemma 2.2.14/(19) v [12]*

```
theorem Hajek_MFL_2_2_14_19 [simp]:
  "Const_1 ∈ CnBL {}"
apply (unfold Const_1_def)
by simp
```

Obecně je však vhodnější konstruovat strukturovaný důkaz tvaru **proof** ... **qed**, na němž jsou vidět jednotlivé kroky zachycující strukturu dokazování mimo program, což přispívá k čitelnosti důkazu, a který v případě komplexního odvození umožňuje rozdělení důkazu na méně složité podcíle dokazatelné pomocí automatických nástrojů.

4.3.1 Dokazování teorémů BL

V případě teorémů BL je třeba zdůraznit, že se jedná – podobně jako u axiomů – o schémata formulí a z hlediska HOL o termy ohodnocené proměnnými. To hraje roli při formulaci tvrzení o dokazatelnosti teorému – je nutné alespoň jednu z jeho proměnných vázat metakvantifikátorem a v důkazu ji fixovat. Verifikátor pak snáze exportuje pravidlo pro řešení hlavního cíle.

Použitým vzorcem při formalizaci důkazů v kalkulu je následující schéma:

```
theorem pojmenování teorému [simp]:
  "∀φ. dokazovaný teorém ∈ CnBL {}"

proof
  fix φ
  from předpoklad_i and ... and předpoklad_j have
  1: "odvozená formule ∈ CnBL {}"
    by důkazová metoda
    ⋮
  from předpoklad_k and ... and předpoklad_l show
  n: "dokazovaný teorém ∈ CnBL {}"
    by důkazová metoda
qed
```

Předpoklady mezi klíčovými slovy **from** a **have** (případně **show**) tvoří množinu faktů, s nimiž pracuje *důkazová metoda*. Předpokladem může být jakýkoliv dříve dokázaný teorém či předchozí krok důkazu, na něž je odka-

zováno pojmenováním teorému či jeho označením v rozsahu *1* až *n*.[27]

Pro hledání automatického skriptu lze na základě praxe doporučit nástroj *sledgehammer* hledající mezi dostupnými fakty parametry pro důkazovou metodu *metis*. Zvýšit šanci na úspěch tohoto heuristického prohledávání lze právě explicitním uvedením předpokladů.[28]
Přestože je *sledgehammer* velmi užitečným nástrojem pro práci s velkým množstvím dokázaných tvrzení, na mnoha důkazech v kalkulu selhává. Jako jeden z těchto případů lze uvést odvození dvojí aplikací pravidla modus ponens. V této situaci je třeba rozdělit tvrzení na dva kroky a aplikovat *sledgehammer* na každý zvlášť.

4.3.2 Deklarace odvozovacího pravidla

Z libovolného teorému obsahujícího implikaci může být pomocí pravidla *modus ponens* deklarováno nové odvozovací pravidlo kalkulu. V kontextu logiky vyššího řádu lze takové pravidlo obecně popsat následující definicí:

Definice 4.3.1. *Odvozovací pravidlo logiky BL je formulí následujícího tvaru:*

(a) $\psi_1 \in CnBL\ \Gamma\ \wedge\ \ldots\ \wedge\ \psi_n \in CnBL\ \Gamma\ \Longrightarrow\ \varphi \in CnBL\ \Gamma$

Jejím formalizovaným ekvivalentem je pak následující formule HOL:

(b) $[\![\psi_1 \in \texttt{CnBL}\ \Gamma\texttt{;}\ \ldots\ \texttt{;}\psi_n \in \texttt{CnBL}\ \Gamma]\!] \Longrightarrow \varphi \in \texttt{CnBL}\ \Gamma$

Potřeba deklarace nového pravidla záleží na četnosti opakování některých kroků v důkazech. Právě kvůli výše popsané nutnosti rozepisovat mezikroky při opakované aplikaci pravidla *modus ponens* bylo do formalizace zařazeno nové pravidlo `BL_transitivity_of_implication` stavějící odvození z axiomu *BL1* na úroveň metalogického pravidla.

Zavedení pravidla spočívá v dokázání tvrzení podle definice 4.3.1(b) s parametrem `[rule_format, simp]`:

Formalizace 4.3.2. *Deklarace odvozovacího pravidla z axiomu BL1*

```
lemma BL_transitivity_of_implication [rule_format, simp]:
"⟦(φ ⟶BL ψ) ∈ CnBL Γ; (ψ ⟶BL χ) ∈ CnBL Γ⟧ ⟹
 (φ ⟶BL χ) ∈ CnBL Γ"
```

[27] Dokazované mezikroky mohou být značeny libovolným řetězcem. Pro udržení přehlednosti je však zvoleno běžné číslování kroků.

[28] *Sledgehammer* rovněž disponuje možností manuálního nastavení mimo jiné časových i prostorových omezení heuristiky, a to pomocí příkazu **sledgehammer_params** uvedeným mimo prostředí důkazu. Pro více podrobností k práci s nástrojem *sledgehammer* viz [5].

Symbol $\Longrightarrow$ je metalogickým operátorem HOL určeným právě pro definování odvozovacích pravidel [17, odd. 5.9]. Formule na levé straně operátoru uvedené v tučných závorkách a oddělené středníkem jsou předpoklady definovaného pravidla. Na pravé straně operátoru je pak odvozovaný závěr.

Stejně jako pravidlo tranzitivity je zavedena i $\wedge$-*adjunkce* z [6] odvozená z teorémů *TBL10* a *TBL11*, a sice jako pravidlo zavedení slabé kojunkce $\wedge_{BL}$:

Formalizace 4.3.3. *Deklarace odvozovacího pravidla pro zavedení* $\wedge_{BL}$

```
lemma BL_ConjWeak_adjunction [rule_format, simp]:
"⟦φ ∈ CnBL Γ; ψ ∈ CnBL Γ⟧ ⟹ (φ ∧BL ψ) ∈ CnBL Γ"
```

Důkazy obou lemmat jsou součástí přílohy `BL_theorems.thy`.

4.3.3 Formalizované důkazy

Formalizace teorémů a pravidel je obsažena v teorii `BL_theorems`. Teorémy jsou vybrány a značeny podle oddílu 1.2 v [6], doplnění několika dalších pochází z [12]. U teorémů, které se v použité literatuře nevyskytují, je zavedeno značení vlastní.

Teorémy jsou formalizovány v pořadí, v jakém byly dokázány. Volba, který teorém v daném důkazu použít, zůstává na uživateli programu, a jelikož je v kódu možné odkazovat pouze na fakta dříve uvedená, lze s pořadím důkazů experimentovat bez nebezpečí zanesení důkazu kruhem.[29]

Teorie `BL_theorems` je uvedena deklarací odvozovacího pravidla pro tranzitivitu implikace. Následuje formalizace odvození redundantního axiomu *BL2* (odpovídajícího teorému *TBL6*), a to podle Karla Chvalovského v [7]. Celý důkaz je rozdělen do více lemmat, bylo možné používat instance průběžné dokázaných formulí. Dále je formalizován důkaz redundance axiomu *BL3* (teorém *TBL7*) z článku Petra Cintuly [8].

Některé z předložených důkazů jsou inspirovány kapitolou 2.2 v [12], kde jsou uvedeny jejich náčrty. Teorémy obsahující ekvivalenci $\longleftrightarrow_{BL}$ jsou kvůli zjednodušení důkazu rozděleny na dvě implikace a označeny `TBL..a` a `TBL..b`.

4.4 Věta o lokální dedukci

Završením formalizace syntaxe BL je verifikace důkazu věty 3.4.1 o lokální dedukci. Důkaz je obsažen v teorii `BL_LocalDeduction`, která obsahuje několik

[29] Za všechny lze uvést důkaz teorému *TBL15* z *TBL24* (respetive z jeho varianty *TBL24b'*)

dalších tvrzení potřebných k důkazu věty a důkaz samotný je pak rozdělen do dvou částí podle směru implikace. Přestože by všechna lemmata i oba směry implikace mohly být zahrnuty do jednoho důkazu, pro větší přehlednost jsou uvedeny jako samostatná tvzení. Ta jsou předložena v pořadí, v jakém byla dokázána.

Stejně jako v případě formalizace důkazů v kalkulu BL, v důkazech tvrzení *o dokazatelnosti* je vynechána definovaná relace $\vdash_{BL}$ a místo ní je použita ekvivalentní notace skrze náležení do důkazového uzávěru.

Připomeňme lemma 3.4.2 o dokazatelnosti z prázdné množiny. Jeho formalizace vypadá následovně:[30]

Formalizace 4.4.1. *Lemma o dokazatelnosti z prázdné množiny*

```
lemma emptyAsm [simp]: "φ ∈ CnBL {} ⟹ φ ∈ CnBL T"
apply (erule CnBL.induct)
by blast+
```

Dále dokažme lemma 3.4.3 pro úpravu výrazů obsahujících formule tvaru φ^n.

4.4.1 Lemma o násobné konjunkci

Vzhledem k délce formalizovaného důkazu jsou některé kroky vynechány a kód je proložen komentáři. Vynechané kroky jsou nahrazeny řetězcem `(...)`. Plné znění důkazu je uvedeno v příloze `BL_LocalDeduction.thy`.

Formalizace 4.4.2. *Lemma o násobné konjunkci*

```
lemma MultiConjDistribution [simp]:
  "((φ ^BL n &BL φ ^BL m) ⟷BL φ ^BL n + m) ∈ CnBL {}"
proof (unfold Equivalence_BL_def, induct n, simp)
(...)
```

Při iniciaci důkazu je aplikována definice ekvivalence BL a vygenerovány podcíle pro důkaz indukcí podle `n`. V podcíli pro případ `n = 0` pak metoda `simp` přepíše výskyty výrazů $\varphi\hat{}^{BL}$`n` podle definice násobné konjunkce na `Const_1`.

Aplikace jednotlivých teorémů a pravidla modus ponens jsou patrné z plného znění formalizace v příloze práce. Z axiomů a teorémů BL dokážeme bázi indukce:

[30] Lemma je formalizováno v teorii `BL_theorems` pro potřeby zavedení odvozovacích pravidel.

```
(...)
  from 6 and 8 show
  9: "((Const_1 &_BL φ ^BL m ⟶_BL φ ^BL m) &_BL
       (φ ^BL m ⟶_BL Const_1 &_BL φ ^BL m)) ∈ CnBL {}"
    by (metis CnBL.MP)
(...)
```

Nyní je třeba dokázat indukční krok pro `Suc n`. Příkazem **next** jsou odděleny bloky důkazu a číslování kroků tedy začíná znovu od 1.

Předpokládejme levou stranu implikace z vygenerovaného podcíle – indukční hypotézu:

```
(...)
next
  fix n
  assume
  1: "((φ ^BL n &_BL φ ^BL m ⟶_BL φ ^BL n + m) &_BL
       (φ ^BL n + m ⟶_BL φ ^BL n &_BL φ ^BL m)) ∈ CnBL {}"
(...)
```

Z prvního členu předpokládané konjunkce dostaneme postupnými úpravami formuli, která se blíží první implikaci v dokazované ekvivalenci:

```
(...)
  from 1 and 2 have
  3: "(φ ^BL n &_BL φ ^BL m ⟶_BL φ ^BL n + m) ∈ CnBL {}"
    by (metis (full_types) CnBL.MP)
(...)
  then have
  7: "(((φ &_BL φ ^BL n) &_BL φ ^BL m) ⟶_BL
       (φ &_BL φ ^BL n + m)) ∈ CnBL {}"
    by (metis (full_types) BL_transitivity_of_implication TBL9a)
  have
(...)
```

Formule z kroku `7` je dále upravena podle tvrzení `8` a `9` plynoucích z formalizované definice φ^n, čímž je první implikace dokázána:

```
(...)
  have
  8: "(φ &BL φ ^BL n) = (φ ^BL Suc n)"
    by simp
  have
  9: "(φ &BL φ ^BL n + m) = (φ ^BL Suc n + m)"
    by simp
  from 7 and 8 and 9 have
  10: "((φ ^BL Suc n &BL φ ^BL m) ⟶BL (φ ^BL Suc n + m))
        ∈ CnBL {}"
    by metis
(...)
```

Obrácená implikace je dokázána obdobně postupnými úpravami druhé části konjunkce v indukčním předpokladu a následným aplikováním rovností z kroků *8* a *9*:

```
(...)
  from 1 and 11 have
  12: "(φ ^BL n + m ⟶BL φ ^BL n &BL φ ^BL m) ∈ CnBL {}"
    by (metis CnBL.MP)
(...)
  from 16 and 8 and 9 have
  17: "(φ ^BL Suc n + m ⟶BL φ ^BL Suc n &BL φ ^BL m)
        ∈ CnBL {}"
    by metis
(...)
```

Z kroků *10* a *17* je postupným aplikováním pravidla modus ponens na *TBL10* vytvořena konjunkce, čím je důkaz uzavřen:

```
(...)
  from 17 and 19 show
  20: "(((φ ^BL Suc n &BL φ ^BL m) ⟶BL (φ ^BL Suc n + m))
&BL
    (φ ^BL Suc n + m ⟶BL φ ^BL Suc n &BL φ ^BL m)) ∈ CnBL
{}"
    by (metis CnBL.MP)
qed
```

4.4.2 První část věty o lokální dedukci

Důkaz první z implikací ve větě 3.4.1 je veden jako v [12] induktivně z definice důkazového uzávěru *CnBL*, podle jehož konstruktorů pravidlo *CnBL.induct*, aplikované při iniciaci důkazu, vytvoří potřebné podcíle. Následuje fixování proměnných.

Formalizace 4.4.3. *První část věty o lokální dedukci*

```
theorem LocalDeductionBL1 [simp]:
"ψ ∈ CnBL (Γ ∪ {φ}) ⟹ (∃n. ((φ ^BL n) →BL ψ) ∈ CnBL
Γ)"
proof (rule CnBL.induct)
  fix φ' ψ χ
(...)
```

Nejprve dokažme případy konstruktorů zachycující axiomatiku logiky BL. Jedná se o triviální krok, protože axiomy jsou z definice v uzávěru pro každé Γ, takže závěr implikace vždy platí. Vhodné je tedy instanciovat existenční kvantifikátor číslem *0* a z teorému *TBL2* s pomocí nástroje *sladgehammer* odvodit důkaz požadované formule:

```
(...)
  have
  1: "(φ ^BL 0 →BL ((φ' →BL ψ) →BL ((ψ →BL χ)
→BL
       (φ' →BL χ)))) ∈ CnBL Γ"
    by (metis (full_types) CnBL.BL1 CnBL.MP TBL2 emptyAsm)
  then show
  2: "∃n. (φ ^BL n →BL ((φ' →BL ψ) →BL ((ψ →BL χ)
       →BL (φ' →BL χ)))) ∈ CnBL Γ"
    by metis
(...)
```

Důkazy ostatních axiomů jsou vedeny obdobně jako v uvedeném příkladu pro axiom BL1.

Dalším případem je náležení formule ψ do množiny $\Gamma \cup \{\varphi\}$ a je dokázán po případech. Důkaz tohoto podcíle můžeme uvést v plném znění a ponechat jej bez podrobnějšího komentáře.

```
(...)
  fix φ'
  assume "φ' ∈ (Γ ∪ {φ})"
  then show
  9: "∃n. (φ ^BL n ⟶BL φ') ∈ CnBL Γ"
    proof
      assume "φ' ∈ Γ"
      then show "∃n. (φ ^BL n ⟶BL φ') ∈ CnBL Γ"
      by (metis (full_types) CnBL.Assumption CnBL.MP TBL2
emptyAsm)
    next
      assume "φ' ∈ {φ}"
      then have "φ' = φ"
        by simp
      hence "(φ ^BL 1 ⟶BL φ') ∈ CnBL Γ"
        by simp
      then show "∃n. (φ ^BL n ⟶BL φ') ∈ CnBL Γ"
        by (rule exI)
    qed
(...)
```

Zbývá tedy dokázat indukční krok. Isabelle tento podcíl vygeneruje jako konjunkci tvrzení, z nichž má plynout hlavní cíl důkazu. Tuto konjunkci je třeba explicitně předpokládat a získat z ní potřebné předpoklady a proměnné `n` a `m`:

```
(...)
  fix φ' ψ
  assume
  1: "(ψ ∈ CnBL (Γ ∪ {φ}) ∧ (∃n. (φ ^BL n ⟶BL ψ) ∈ CnBL
Γ))
        ∧ (ψ ⟶BL φ') ∈ CnBL (Γ ∪ {φ}) ∧
        (∃n. (φ ^BL n ⟶BL (ψ ⟶BL φ')) ∈ CnBL Γ)"
  from 1 obtain n where
  2: "(φ ^BL n ⟶BL ψ) ∈ CnBL Γ"
    by metis
  from 1 obtain m where
  3: "(φ ^BL m ⟶BL (ψ ⟶BL φ')) ∈ CnBL Γ"
    by metis
(...)
```

Z uvedených předpokladů a teorémů *Hajek_4*, *Hajek_7* a *TBL10* je v kalkulu odvozeno tvrzení *11*, z *MultiConjDistribution* a *BL1* pak *16*. Z *11* a *16* je dokazatelné tvrzení *17*, ve kterém je předložen svědek pro instanciování existenčního kvantifikátoru. Krok *18* pak už jen uzavírá důkaz:

```
(...)
  from 8 and 10 have
  11: "((φ ^BL n &BL φ ^BL m) ⟶BL φ') ∈ CnBL Γ"
    by (metis CnBL.MP)
  have
(...)
  from 14 and 15 have
  16: "(((φ ^BL n &BL φ ^BL m) ⟶BL φ') ⟶BL
        ((φ ^BL n + m) ⟶BL φ')) ∈ CnBL Γ"
    by (metis (full_types) CnBL.MP)
  from 11 and 16 have
  17: "((φ ^BL n + m) ⟶BL φ') ∈ CnBL Γ"
    by (metis CnBL.MP)
  from 17 show
  18: "∃n. ((φ ^BL n) ⟶BL φ') ∈ CnBL Γ"
    by metis
qed
```

4.4.3 Odvoditelnost násobné konjunkce

Před dokončením důkazu věty o lokální dedukci je zvláště dokázáno lemma 3.4.4. Důkaz je veden *strukturovanou indukcí* podle `n`. Je tedy uvozen s argumentem *induct n* a případy jsou v něm rozebrány příkazem **case** a odděleny pomocí **next**. Výraz *?case* pak vrací aktuáně rozebíraný případ jako tvrzení.

Formalizace 4.4.4. *Odvoditelnost násobní konjunkce*

```
lemma DerivabilityOfStrongConj [simp]:
  "(φ ^BL n) ∈ CnBL (Γ ∪ {φ})"
proof (induct n)
  case 0
  show ?case by simp
next
  have
    "∀n. (φ &BL φ ^BL n) = (φ ^BL Suc n)"
      by simp
```

```
  then have
  1: "∀n. ((φ &_BL φ ^^_BL n) ⟶_BL (φ ^^_BL Suc n))
       ∈ CnBL (Γ ∪ {φ})"
    by simp
  from 1 have
  2: "∀n. (φ ⟶_BL (φ ^^_BL n ⟶_BL (φ ^^_BL Suc n)))
       ∈ CnBL (Γ ∪ {φ})"
    by simp
  from 2 have
  3: "∀n. (φ ^^_BL n ⟶_BL (φ ^^_BL Suc n)) ∈ CnBL (Γ ∪ {φ})"
    by blast
  case (Suc n)
  then show ?case using 3
    by blast
qed
```

4.4.4 Monotonie dokazatelnosti v BL

Dalším tvrzením aplikovaným v dokončení důkazu věty o lokální dedukci je lemma 3.4.1 o monotonii dokazování v BL.

Aplikací indukčních pravidel `CnBL.induct` je hlavní cíl rozložen na podcíle, které jsou dokázané opakovaným použitím metody `blast`:

Formalizace 4.4.5. *Monotonie dokazatelnosti v BL*

```
lemma CnBLMonotonicity [simp]:
  "φ ∈ CnBL T ⟹ φ ∈ CnBL (T ∪ Δ)"
apply (rule CnBL.induct)
by blast+
```

Tímto máme dostatek přostředků k dokončení důkazu věty o lokální dedukci.

4.4.5 Dokončení důkazu věty o lokální dedukci

Důkaz druhé implikace z věty 3.4.1 je veden emulací taktických skriptů. Oproti strukturovanému důkazu je méně čitelný, zato je výrazně kratší. Uveďme formalizovaný důkaz v plném znění a následně jej rozeberme:

Formalizace 4.4.6. *Dokončení důkazu věty o lokální dedukci*

```
theorem LocalDeductionBL2 [simp]:
  "(∃n. (φ^^BL n ⟶BL ψ) ∈ CnBL T) ⟹ ψ ∈ CnBL (T ∪ {φ})"
apply (erule exE)
apply(subgoal_tac "(φ^^BL n ⟶BL ψ) ∈ CnBL (T ∪ {φ})")
prefer2
apply(rule CnBLMonotonicity)
apply metis
apply(subgoal_tac "(φ ^^BL n) ∈ CnBL (T ∪ {φ})")
prefer 2
apply (rule DerivabilityOfStrongConj)
by blast
```

Důkaz je veden zpětně. Nejdříve je aplikováno obrácené pravidlo eliminace `erule exE`, které převede ∃ na interní obecný kvantifikátor ⋀.

Přidaní předpokladu `(φ^^BL n ⟶BL ψ) ∈ CnBL (T ∪ {φ})` pomocí metody `subgoal_tac` vede k reformulaci cílů – předpoklad je vložen do dokazované metaimplikace a dále je vygenerován nový podcíl, a sice že tento předpoklad plyne z antecedentu hlavního cíle. To je po posunutí podcíle na první místo dokázáno aplikací lemmatu `CnBLMonotonicity` a metodou `metis`.

Stejně tak je přidán předpoklad `(φ ^^BL n) ∈ CnBL (T ∪ {φ})`, tedy lemma 3.4.4 o odvoditelnosti silné konjunkce. Nový podcíl je dokázán jednoduše aplikací formalizované verze lemmatu.

Metoda `blast` v závěru důkazu pak pravidlem modus ponens odvodí z obou přidaných předpokladů dokazovaný závěr. Tím je platnost hlavní metaimplikace verifikována a důkaz je uzavřen.

V této kapitole byly formalizovány důkazy obou implikací ve větě o lokální dedukci z přílohy `BL_LocalDeduction.thy`. Podoba důkazů vychází ze záměru demonstrovat možnosti verifikace v Isabelle/HOL, spíše než předložit krátký kód. Zejména struturovaný důkaz první implikace by bylo možné zkrátit pomocí nástroje *sledgehammer* postupným odebíráním kroků a jejich dohledáváním v celé teorii pomocí heuristiky. V případě pokračování formalizace logiky BL by toto byla jednou z prvních úprav.

5 Závěr

Práce předkládá formalizaci syntaktických objektů logiky BL v prostředí matematického asistenta Isabelle/HOL. Text přibližuje pracovní prostředí Isabelle/HOL potenciálním uživatelům tak, aby byli schopni pracovat s kódem v příloze této práce.

V oddílu 2.1 začíná popis shrnutím vývoje důkazových verifikátorů vycházejících z potřeby ověřovat důkazy logického systému Dany Scotta a vedoucího až k implementaci matematického asistenta Isabelle a jeho nástavby HOL. Z uvedených faktů je patrné, že asistované dokazování patří už několik dekád k živým oborům teoretické informatiky, čehož může být dokladem i pravidelné vydávání nových verzí Isabelle.[31]

V oddílu 2.4 jsou pak popsány konstrukce jazyka Isar použité ve formalizaci logiky BL, a to na příkladě jednoduché teorie přirozených čísel.

Stručný úvod do logiky BL v kapitole 3 uvádí matematické pozadí celé práce a přehled objektů určených k formalizaci.

V kapitole 4 byly formule logiky BL formalizovány jako základní datový typ teorie. Funkcí do množiny formulí pak byla zachycena zkratka pro násobnou konjunkci φ^n. Důkazový uzávěr logiky byl formalizován jako induktivní množina `CnBL` a relace dokazatelnosti pak definována jako ekvivalentní zápis náležení do této množiny.

Formule a důkazový uzávěr tvoří základ pro práci ve formalizované metalogice BL. Popsán byl způsob verifikace důkazů v kalkulu, jejichž vyjádření v jazyce Isar formou strukturovaného důkazu **proof-qed** je snadno čitelné i pro čtenáře neznalého syntaxe jazyka verifikátoru.

V teorii `BL_theorems` jsou ověřeny důkazy rendundantních axiomů *BL2* a *BL3* a většiny teorémů uvedených v [6, odd. 1.2] a [12, odd. 2.2], které byly potřeba k dalším důkazům. Jako jedna ze základních vlastností syntaxe BL byl předložen a popsán důkaz věty o lokální dedukci formalizovaný v teorii `BL_LocalDeduction`.

Matematický asistent Isabelle/HOL nabízí množství nástrojů pro verifikaci důkazů jak výrokové fuzzy logiky, tak její metalogiky. Předložená formalizace může sloužit jako základ širšího projektu formalizace fuzzy logik.

Definici důkazového úzávěru `CnBL` lze snadno rozšířit pro logiky prominentních t-norem, a to přidáním konstruktoru axiomu rozšiřujícího axiomatiku BL. Na způsob dokazování v kalkulu by takovýto zásah neměl žádný dopad.

V případě pokračování projektu by dalším krokem byla formalizace věty o kongruenci [6, věta 1.2.5], která říká, pokud z formule χ získána formule

[31] viz [2]

χ' nahrazením nějaké podformule φ ekvivalentní formulí ψ, pak platí, že

$$\varphi \leftrightarrow \psi \ \vdash_{BL} \ \chi \leftrightarrow \chi'.$$

Tato věta hraje důležitou roli v důkazu úplnosti logiky BL [6, s. 19] a mohla by též sloužit ke zkrácení důkazů v kalkulu. Pro zachování přehlednosti kódu by však její použití nejspíš vyžadovalo definování vlastních taktických skriptů běžících v pozadí, což přesahuje rámec této práce.

Reference

[1] Isabelle Homepage. Verze k 18.8.2014.
URL http://isabelle.in.tum.de/

[2] Isabelle NEWS – history user-relevant changes. Verze k 18.8.2014.
URL http://isabelle.in.tum.de/dist/Isabelle2013-2/NEWS

[3] jEdit Homepage. Verze k 18.8.2014.
URL http://www.jedit.org/

[4] *Logic for Computable Functions: description of a machine implementation.*

[5] Blanchette, J. C.; Paulson, L. C.: *Hammering Away: A User's Guide to Sledgehammer for Isabelle/HOL*. 2013, verze k 18.8.2014.
URL http://isabelle.in.tum.de/documentation.html

[6] Běhounek, L.; Cintula, P.; Hájek, P.: Introduction to mathematical fuzzy logic. In *Handbook of Mathematical Fuzzy Logic*, editace P. Cintula; P. Hájek; C. Noguera, College Publications, 2011, s. 1–101.

[7] Chvalovský, K.: On the independence of axioms in BL and MTL. *Fuzzy Sets and Systems*, ročník 197, 2012: s. 123–129.

[8] Cintula, P.: Short note: on the redundancy of axiom (A3) in BL and MTL. *Soft Comput.*, ročník 9, č. 12, 2005: s. 942–942.

[9] Font, J.; Jansana, R.; Pigozzi, D.: A Survey of Abstract Algebraic Logic. *Studia Logica*, ročník 74, č. 1-2, 2003: s. 13–97, ISSN 0039-3215.
URL http://dx.doi.org/10.1023/A%3A1024621922509

[10] Gordon, M.: From LFC to HOL: a short history. In *Proof, Language, and Interaction: Essays in Honour of Robin Milner*, editace G. Plotkin; C. P. Stirling; M. Tofte, MIT Press, 2000.

[11] Gordon, M.; Milner, R.; Wadsworth, C.: *Edinburgh LCF: A Mechanised Logic of Computation*. Lecture notes in computer science, Springer-Verlag, 1979, ISBN 9783540097242.

[12] Hájek, P.: *Metamathematics of Fuzzy Logic*. Kluwer Academic Publishers, 1988.

[13] Krauss, A.: *Defining Recursive Functions in Isabelle/HOL*. Verze k 18.8.2014.
URL http://isabelle.in.tum.de/documentation.html

[14] Leslie-Hurd, J.: Metis Homepage. Verze k 18.8.2014.
URL http://www.gilith.com/software/metis/index.html

[15] Nipkow, T.: *Programming and Proving in Isabelle/HOL*. Verze k 18.8.2014.
URL http://isabelle.in.tum.de/documentation.html

[16] Nipkow, T.: *What's in Main*. 2013, verze k 18.8.2014.
URL http://isabelle.in.tum.de/documentation.html

[17] Nipkow, T.; Paulson, L. C.; Wenzel, M.: *Isabelle/HOL: A Proof Assistant For Higher-Order Logic*. 2013, verze k 18.8.2014.
URL http://isabelle.in.tum.de/documentation.html

[18] Paulson, L. C.: The Foundantion of Generic Theorem Prover. *Journal of Automated Reasoning*, ročník 5, 1989: s. 363–397.

[19] Scott, D. S.: A Type-theoretical Alternative to ISWIM, CUCH, OWHY. *Theor. Comput. Sci.*, ročník 121, č. 1-2, Prosinec 1993: s. 411–440, ISSN 0304-3975, doi:10.1016/0304-3975(93)90095-B.
URL http://dx.doi.org/10.1016/0304-3975(93)90095-B

[20] Švejdar, V.: *Logika: neúplnost, složitost a nutnost*. Academia - nakladatelství Akademie věd ČR, 2002, ISBN 9788020010056.
URL http://books.google.cz/books?id=8HveAAAACAAJ

[21] Wenzel, M.: *The Isabelle/Isar Reference Manual*. 2013, verze k 18.8.2014.
URL http://isabelle.in.tum.de/documentation.html

Přílohy

- CD nosič se soubory formalizace pro program Isabelle2013-2