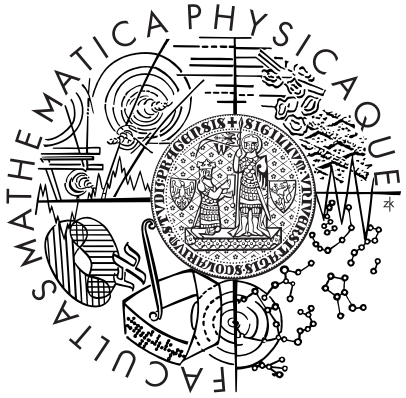
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Ondřej Kunčar

Systémy pro formální matematiku

Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: Mgr. Josef Urban, Ph.D.

Studijní program: Informatika
Studijní obor: Teoretická informatika

PRAHA 2009

Rád bych poděkoval mé rodině za to, že mi umožnila studovat, vedoucímu mé práce Josefu Urbanovi za několik cenných připomínek a Jiřímu Janíčkovi za jazykovou korekturu textu. Tato práce byla napsána v mém rodném domě v Kroměříži, klidnějším to místě než je Praha.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 7. 8. 2009 Ondřej Kunčar

Obsah

Název práce: Systémy pro formální matematiku
Autor: Ondřej Kunčar
Katedra (ústav): Katedra teoretické informatiky a matematické logiky
Vedoucí diplomové práce: Mgr. Josef Urban, Ph.D.
e-mail vedoucího: josef.urban@gmail.com

Abstrakt: Typový systém Mizaru je poměrně sofistikovaný — obsahuje celou řadu vlastností, jako jsou závislé typy, atributy, přetěžování, konstrukci podtypů, struktury a další. Tyto vlastnosti nemalou měrou přispívají k tomu, že formalizace matematiky je v systému Mizar mnohem intuitivnější než v ostatních systémech. Zároveň vyvstává potřeba verifikovat matematické poznatky formalizované v Mizaru i v jiných systémech, aby se zvýšila jistota, že systém Mizar pracuje korektně. Je tedy přirozené pokusit se rekonstruovat tento typový systém v jiných systémech pro formalizaci matematiky.
Tato práce navrhuje takovou rekonstrukci do systému HOL Light. Idea rekonstrukce je reprezentovat typy Mizaru jako predikáty v tomto systému (HOL Light). Součástí této práce je i pokus o co nejpřesnější zachycení relevantní části typového systému Mizaru. V závěru jsou pak uvedeny postřehy, které byly učiněny při návrhu a částečné implementaci navržené rekonstrukce.

Klíčová slova: typový systém, Mizar, HOL Light

Title: Systems for formal mathematics
Author: Ondřej Kunčar
Department: Dep. of Theoretical Computer Science and Mathematical Logic
Supervisor: Mgr. Josef Urban, Ph.D.
Supervisor's e-mail address: josef.urban@gmail.com

Abstract: The Mizar type system is a relatively sophisticated system as it allows for many properties, such as independent types, attributes, overloading, subtyping, structures and many others. All these properties make formalization of mathematics more intuitive in Mizar that in other systems. However, there is a need to verify mathematical results formalized in Mizar in other systems, so that belief in consistency of Mizar system is strengthened. Attempts at reconstruction of this type system in other mathematics formalization systems follow directly from this requisite.
The present work seeks to reconstruct Mizar type system in HOL Light system. The basic idea here is to represent Mizar types as predicates in this system (HOL Light). The present work also aims at precise description of relevant parts of Mizar type system. The thesis concludes by reviewing some of the insights that were arrived at in the course of designing and implementing suggested reconstruction.

Keywords: type system, Mizar, HOL Light

Kapitola 1

Úvod

1.1 Cíl práce

Cílem této práce je navrhnout způsob, jak rekonstruovat typový systém Mizaru v systému HOL Light. Nejdříve bude popsána hlavní část typového systému Mizaru. Následně bude navržen způsob, jak tento typový systém přeložit do systému HOL Light. Některé části tohoto překladu budou přímo implementovány, aby byla ověřena proveditelnost tohoto překladu a aby byla postihnuta i jeho technická stránka.

1.2 Motivace

Motivací pro tuto práci jsou dva základní poznatky:

- Typový systém Mizaru je bohatší než typový systém systému HOL Light — obsahuje struktury, závislé typy, konstrukci podtypů, atributy a umožňuje přetěžování predikátů a funktorů. Celá tato množina vlastností pak umožňuje mnohem lépe zachytit, jak matematik typicky přistupuje k matematickým objektům a jak je kategorizuje [MW02]. Toto není náhodná vlastnost, neboť při tvorbě tohoto typového systému, potažmo celého Mizaru, se vyšlo právě ze studie zabývající se tím, jak matematici zapisují a prezentují své matematické poznatky v odborných článcích. Tímto se systém Mizar odlišil téměř od všech ostatních systémů pro formalizaci matematiky [Sch07]. Proto je velmi přirozené snažit se rekonstruovat tento bohatý a intuitivní typový systém v jiných systémech se snahou vytěžit jeho pozitivní vlastnosti.
- Na druhou stranu nevýhodou Mizaru je, že jeho ověřovač má poměrně komplexní implementaci, přičemž zdrojový kód tohoto ověřovače není veřejně přístupný. Navíc není ani možné vypsat odvození atomických kroků

důkazu[1], které ověřovač našel při verifikaci, tak, aby je bylo možné ověřit dalším nezávislým nástrojem[JU09]. Tyto skutečnosti pak vzbuzují přirozené pochybnosti, zda nemůže být v ověřovači Mizaru chyba, která by způsobila, že by byly ověřeny i důkazy, které nejsou logicky platné.

Proto se nabízí přeložit důkazy zapsané v syntaxi Mizaru do jiného systému a v něm je znovu ověřit. Za předpokladu, že bude tento překlad korektní, můžeme snížit pravděpodobnost, že je v ověřovači Mizaru chyba. Rekonstrukce typového systému Mizaru v systému HOL Light je jedním z předpokladů uskutečnění takového překladu. Navíc, jak bude popsáno v 3.1, systém HOL Light používá LCF metodologii při implementaci ověřovače, a tudíž lze o tomto ověřovači s velkou jistotou tvrdit, že je korektní.

[1] tyto již dále nedělitelné kroky jsou označeny v Mizaru klíčovým slovem **by**

Kapitola 2

Mizar

Současná podoba systému Mizar dosud nebyla v literatuře uspokojivě popsána. Existuje několik rozsáhlejších starších prací [Bon90, Muz93, Try93], které však už neodpovídají současnému stavu systému. Některé novější práce [Wie99, Wie], které se zabývají současnou verzí systému, zase zdaleka nepokrývají celý systém, jsou spíše jakýmsi lehkým uživatelským úvodem.

Proto v této kapitole bude uveden na jednu stranu popis, který nezahrnuje všechny vlastnosti Mizaru[1], na druhou stranu v popisovaných částech byla snaha být co nejpřesnější — což bude mít jistě vzhledem k absenci kvalitní dokumentace svůj nezanedbatelný přínos. Protože cílem je pokusit se rekonstruovat typový systém Mizaru v systému HOL Light, jde autorovi především o principy, nikoliv o co nejúplnější popis všech vlastností Mizaru (často nezdokumentovaných).

2.1 Základ systému

I když nám jde především o typový systém, nelze ve zkratce nezmínit, na jakých základech Mizar stojí.

Systém Mizar používá klasickou predikátovou logiku prvního řádu[2]. Je založen na Tarski-Grothendieckově teorii množin, což je varianta Zermelo-Fraenkelovy teorie množin, ve které je axiom nekonečna nahrazen Tarského axiomem, který zaručuje existenci libovolně velkých silně nedostupných kardinálů [3] [PR01].

2.2 Jazyk

V této části bude stručně popsán jazyk Mizaru. I když nám jde o typový systém, bez stručného popisu jazyka bychom se zřejmě neobešli. Jazyk Mizaru

[1] nepopisujeme například struktury

[2] v některých kontextech jsou dovoleny volné proměnné druhého řádu

[3] což mimo jiné implikuje axiom výběru

je poměrně bohatý, popsány budou jen části, které jsou relevantní pro pochopení dalšího textu.

2.2.1 Termy

Formát termů nevybočuje z obvyklého pojetí termů v predikátové logice [Wie]:

- Každá proměnná je term.
- Pokud je f funktorový[4] symbol, následující zápis je také term:

  $$(t_1, \ldots, t_m)\, f\, (t_{m+1}, \ldots, t_n)$$

  Kde $t_1, \ldots, t_n$ jsou termy. Argumenty funktoru tedy můžou být i před funktorovým symbolem. Pokud se vyskytuje v seznamu jen jedna proměnná, lze závorky vynechat.
- Každý term v závorkách je také term.

2.2.2 Atomické formule

Mizar obsahuje dva typy atomických formulí:

- Jsou to predikáty. Nechť je R predikátový symbol, potom následující zápis je atomická formule [Wie]:

  $$t_1, \ldots, t_m\, R\, t_{m+1}, \ldots, t_n$$

  Kde $t_1, \ldots, t_n$ jsou termy.
- Dalším typem atomických formulí jsou kvalifikující formule. Jejich tvar je následující:

  $$t \textbf{ is } \theta$$
  $$t \textbf{ is } \alpha$$

  Kde t je term, θ je typ a α je atribut. Více lze nalézt v 2.4.7.

[4] označení Mizaru pro funkce

2.2.3 Formule

Formát formulí je krátce shrnut v následující tabulce [Wie], vlevo je zápis v běžném jazyce predikátové logiky, vpravo pak přepis do jazyka systému Mizar:

$\bot$	**contradiction**
$\neg\phi$	**not** ϕ
$\phi \wedge \psi$	ϕ **&** ψ
$\phi \vee \psi$	ϕ **or** ψ
$\phi \Rightarrow \psi$	ϕ **implies** ψ
$\phi \Leftrightarrow \psi$	ϕ **iff** ψ
$\exists x.\psi$	**ex** x **being** θ **st** ψ
$\forall x.\psi$	**for** x **being** θ **holds** ψ
$\forall x.(\phi \Rightarrow \psi)$	**for** x **being** θ **st** ϕ **holds** ψ

Tento přehled je zde uveden především proto, aby bylo vidět, že při uvedení nové proměnné pomocí existenčního či obecného kvantifikátoru je potřeba určit typ nové proměnné.

2.3 Závislé typy

Protože Mizar používá závislé typy, začíná popis typového systému Mizaru krátkým obecným popisem závislých typů.

Závislé typy[5] jsou typy, které můžou záviset na termech. Jde tedy o typy, které jsou jistým způsobem parametrizované. S typy, které jsou parametrizovány, se můžeme setkat ve většině moderních funkcionálních programovacích jazyků. Od závislých typů se pak ale liší tím, že tyto typy závisí na jiném typu. Tyto bývají označovány jako polymorfní typy. Při konstrukci polymorfního typu je pak vyžadováno, aby byl uveden jako parametr jiný typ. Jako příklad může posloužit typ `'a list` z programovacího jazyka OCaml. Představuje seznam prvků typu `'a`. Když dosadíme za typový parametr `'a` konkrétní typ, například `int list`, dostaneme nepolymorfní typ, v našem případě seznam celých čísel.

Závislé typy se liší od polymorfních typů v tom smyslu, že jako parametry nevystupují jiné typy, ale hodnoty (termy) příslušného jazyka. Typickým příkladem může být typ Nat$[n]$, který představuje přirozená čísla menší než n. Příkladem instanciovaného typu je například typ Matrix$[5,6]$, což jsou matice tvaru 5×6. Protože však proměnné jsou také termy, můžeme jako příklad uvést i typ List$[n]$, což jsou všechny seznamy délky n, kde n je někde dříve deklarovaná proměnná. Závislé typy nejsou příliš často implementovány, je tomu tak například v Dependent ML, Coqu a také v Mizaru.

[5] anglicky *dependent types*

2.4 Typový systém

Jak bylo zmíněno v úvodu kapitoly 2, sémantika Mizaru je založena na teorii množin. Přičemž je podstatné, že jde o netypovanou teorii množin. Přesto však Mizar obsahuje typy. Tyto typy ale nejsou vlastnostmi objektů (individuí), ale výrazů (termů) jazyka Mizaru, které odkazují na objekty teorie množin. Jazyk Mizaru tedy není založen na žádné teorii typů [Wie99].

Každá proměnná v Mizaru musí mít svůj typ. Není tedy dovoleno mít proměnné neznámého typu. Na základě deklarace typu každé proměnné lze určit jednoznačný typ každého termu [Ban03]. Typy Mizaru jsou tvořeny z tzv. módů a atributů[6], které budou dále popsány samostatně.

Jelikož každá proměnná musí mít svůj typ, děje se veškeré typování v Mizaru v určitém kontextu, který představuje právě typy volných proměnných. Způsob deklarování kontextu je blíže popsán v 2.5.1. Prozatím se spokojíme s tím, že kontext budeme označovat pomocí Δ, které bude představovat zobrazení z proměnných do jejich typů.

2.4.1 Přetěžování

Typový systém v Mizaru mimo jiné slouží k přetěžování predikátů a funktorů. To znamená, že lze mít například predikáty stejného jména a o stejném počtu argumentů, které se liší v typech těchto argumentů. Je pak na systému, aby vybral „správnou“ variantu podle typů skutečných argumentů. Podobně jsou přetěžovány i módy a atributy. Popis typového systému s tímto přetěžováním by byl zbytečně složitý.[7]

Proto předpokládejme, že vstup je nejdříve předzpracován v tom smyslu, že jsou všechna tato přetížení vyhodnocena a příslušná jména predikátů, funktorů, módů a atributů jsou nahrazena jednoznačnými názvy. Tedy když se budeme odkazovat v předpisu určitého typu na určitý mód, bude název módu v tomto předpisu jednoznačně odkazovat na příslušný mód, který byl původně nalezen pomocí vyhodnocení přetížení — tedy z počtu a typu skutečných argumentů. Předpoklad zjednoznačnění jmen není žádnou újmou na obecnosti a umožní nám zjednodušit následující popis typového systému.

2.4.2 Módy

Módy definují závislé typy (viz 2.3). Závisí tedy na dalších termech, jejichž typ a počet je součástí definice módu [Wie99]. Zde je příklad definice módu:

[6] anglicky *modes and attributes*

[7] Zápis nějakého typu bychom nemohli chápat jako typ samotný, ale spíše jako typový výraz, který bude teprve vyhodnocen na skutečný typ. Přímý přístup, jaký bude dále používán (zápis typu = typ), by nefungoval, protože stejný zápis typu by v různých kontextech představoval jiný typ.

definition
  let a_1, a_2 **be** set;
  mode Function **of** a_1, a_2 **is** ...;
end;

V této definici se definuje nový mód Function, který má dva parametry (a_1 a a_2) typu set. Jedná se tedy o závislý typ, který závisí na dvou termech typu set. Konkrétní definice tohoto módu byla vynechána, ale intuitivně je jasné, že bude definovat funkce z množiny a_1 do množiny a_2.

Instanciací módu termy dostaneme kořenový typ[8]. Jako příklad se nabízí mód Function:

Function **of** NAT, NAT
Function **of** A, B

V prvním příkladě je mód Function aplikován na dva termy NAT, které označují množinu přirozených čísel. Celý kořenový typ pak reprezentuje všechny funkce z přirozených čísel do přirozených čísel. Druhý příklad pak ukazuje, že jako term je možné uvést libovolnou proměnnou, která však musela být někde dříve deklarována s nějakým typem — v našem případě nutně s typem set.

Aby nedošlo ke zmatení, není na škodu explicitně zdůraznit rozdíl mezi dvěma dosud uvedenými koncepty:

- mód – konstruktor – představuje závislý typ, po jeho aplikaci na termy vzniká kořenový typ
- kořenový typ – konstrukce – vznikne instanciací módu

Typ set je vestavěný typ a je to jediný vestavěný typ Mizaru. Každý kořenový typ kromě typu set má svůj nadtyp. Tento nadtyp musí být uveden při definici módu. Protože typ set je jediný vestavěný typ a jako jediný nemusí mít nadtyp, je také nejobecnějším (největším) typem vzhledem k hierarchii typů. Hierarchie kořenových typů má stromovou strukturu, v jejímž kořeni je typ set [Wie99].

2.4.3 Atributy

Jak bylo uvedeno v 2.4.2, hierarchie kořenových typů má stromovou strukturu. Tato vlastnost může někdy vést k nepříjemným problémům. Uvažujme proměnnou A, která má typ TopSpace — představující topologický prostor. Chtěli bychom definovat nový mód Open-Closed-Subset **of** X, v definici pak musíme uvést nadtyp nově vzniknuvšího závislého typu. Představme si, že už jsme někdy dříve definovali módy Open-Subset **of** X a Closed-Subset **of** X. V takovém případě bychom chtěli, aby typy vzniklé z nového módu Open-Closed-Subset **of** X

[8] anglicky *radix type*

měly jako nadtyp jak Open-Subset **of** X, tak i Closed-Subset **of** X. To ovšem v Mizaru nejde, lze uvést jen jeden nadtyp [Try93].

Řešením je zavést konstrukci, která by připomínala adjektiva, která by nám dovolovala modifikovat už zavedené typy. Neboli dovolovala by nám následující zápis:

open closed Subset **of** X

Na typ Subset **of** X jsme aplikovali atributy open a closed. Nyní tento typ má za nadtyp jakýkoli typ vzniklý vynecháním jednoho či více adjektiv. V našem případě má uvedený typ za nadtypy closed Subset **of** X a open Subset **of** X a samozřejmě Subset **of** X [Try93].

Konstruktory adjektiv jsou nazývány atributy. Aplikováním atributu na nějaký typ vznikne adjektivum. Atributy nelze aplikovat na libovolný typ, naopak pro každý atribut je definována množina typů, na které je atribut aplikovatelný. Při definici atributu je uveden mateřský typ, na jehož základě se určuje množina typů, na které je atribut aplikovatelný.

Definice 1. Nechť atribut α má jako mateřský typ θ. Nechť $\preceq$ je relace podtypu definovaná v definici 8. Definujme množinu typů $\Sigma(\alpha)$ následovně:

$$\Sigma(\alpha) = \{\theta[\chi] \mid \chi \text{ je slučitelné dosazení vzhledem ke kontextu } \alpha\}$$

Kde slučitelnost dosazení je formálně definována v definici 13 a $\theta[\chi]$ je typ θ po substituci za všechny proměnné z kontextu α pomocí dosazovacího zobrazení χ – více viz 2.5.1. Pak definujme množinu typů $\Omega(\alpha)$ následovně:

$$\Omega(\alpha) = \{\tau' \mid \exists \tau \in \Sigma(\alpha) \wedge \tau' \preceq \tau\}$$

Atribut α je pak aplikovatelný na typ τ právě tehdy, když $\tau \in \Omega(\alpha)$.

Výhodou adjektiv je, že mohou být přidávána postupně, tedy poté co už je nějaký hlavní typ definován. Umožňují tak vytvářet speciálnější případy typu, na který jsou aplikována.

Automatické odvozování atributů

Další vlastností mechanismu adjektiv je možnost automatizovat, že určitá množina atributů aplikovaná na nějaký typ odvozuje[9] jinou množinu atributů aplikovaných na stejný typ. Tento fakt budeme značit následovně:

$$A \overset{\theta}{\Longrightarrow} B$$

[9]Též bychom mohli říct *implikuje*, ale budeme opatrnější, neboť popsaný mechanismus je slabší než obecná vlastnost implikování množiny atributů jinou množinou. Hlavní rozdíl spočívá v tom, že vztah automatického odvozování atributů musí být někde explicitně definován, kdežto implikace atributů (pokud chápeme množinu atributů jako jejich konjunkci) je fakt přítomný sám o sobě z logických důvodů.

Uvedený zápis označuje, že množina atributů A aplikovaná[10] na typ θ odvozuje množinu atributů B aplikovaných na stejný typ θ. Pokud je tento fakt explicitně zaveden do systému[11], budeme říkat, že vztah $A \overset{\theta}{\Longrightarrow} B$ je automatizován. Pokud přijmeme konvenci, že slovo **is** budeme používat jako kvalifikaci, že nějaký objekt má určitý typ, lze odvozování množin atributů přepsat formálně následovně:

Definice 2. Nechť je A množina atributů a nechť jsou tyto atributy aplikovatelné na typ θ. Potom, pokud je automatizován následující vztah $A \overset{\theta}{\Longrightarrow} B$, jeho přesná sémantika je, že když x **is** $A'\ \theta$ a $A \subseteq A'$, potom i x **is** $(A' \cup B)\ \theta$.

Pro další potřeby definujme množinu všech atributů, které jsou odvozovány jinou množinou atributů.

Definice 3. Nechť je A množina atributů a nechť jsou tyto atributy aplikovatelné na typ θ. Pak definujme:

- $\Theta_0(A,\theta) = \bigcup\{B \mid A' \overset{\theta}{\Longrightarrow} B \wedge A' \subseteq A\}$
- $\Theta_{i+1}(A,\theta) = \bigcup\{B \mid A' \overset{\theta}{\Longrightarrow} B \wedge A' \subseteq \Theta_i(A,\theta)\}$ pro $i = 0,1,\ldots$
- $\Theta(A,\theta) = \Theta_k$ takové, že $\forall i > k.\Theta_i = \Theta_k$

$\Theta(A,\theta)$ je korektně definováno, neboť je vždy automatizováno jen konečné množství vztahů $A \overset{\theta}{\Longrightarrow} B$, a tedy nutně existuje k takové, že $\forall i > k.\Theta_i = \Theta_k$.

Množinu atributů může také odvozovat nějaký funktor. Tento fakt budeme označovat následovně:

$$\varphi \Longrightarrow A$$

Jeho přesná sémantika je definována v následující definici:

Definice 4. Nechť je φ funktor a A je množina atributů. Potom, pokud je automatizován následující vztah $\varphi \Longrightarrow A$, jeho přesná sémantika je, že když $A'\ \theta$ je návratový typ funktoru φ, potom i $(A \cup A')\ \theta$ je návratový typ funktoru φ.

Stejně jako u odvozování atributů atributy definujeme množinu všech atributů, které jsou odvozovány nějakým funktorem.

Definice 5. Nechť je φ funktor, pak definujme:

$$\Theta(\varphi) = \bigcup\{B \mid \varphi \Longrightarrow B\}$$

[10] Omezení se na množiny aplikovaných atributů není újma na obecnosti, protože na pořadí aplikovaných atributů nezáleží, jakákoli jejich permutace označuje ekvivalentní typy, stejná situace panuje i u vícenásobného zápisu stejného atributu.

[11] Pro postup, jak lze tohoto docílit, viz 2.5.8.

2.4.4 Typy

Jak už bylo neformálně používáno v předešlých oddílech, obecný typ v Mizaru se skládá z kořenových typů a typů dekorovaných posloupností adjektiv.

Definice 6. Nechť jsou $t_1, t_2, \ldots, t_n$ termy jazyka Mizar, nechť je Γ název módu, potom kořenový typ má obecně následující tvar:

$$\Gamma \textbf{ of } t_1, t_2, \ldots, t_n$$

Aby byla definice korektní, musí termy $t_1, t_2, \ldots, t_n$ definovat dosazení slučitelné s kontextem módu Γ – více viz 2.5.2.

Definice 7. Typy Mizaru jsou definovány induktivně následujícími dvěma podmínkami [12]:

1. Každý kořenový typ je typ.
2. Pokud je θ typ a $\alpha_1, \alpha_2, \ldots, \alpha_k$ atributy na něj aplikovatelné, pak je typ i aplikace těchto atributů na θ:

$$\alpha_1\ \alpha_2 \ldots \alpha_k\ \theta$$

Zatím popsaný mechanismus dovoluje velmi lehce vytvořit typ, který neodpovídá žádnému objektu. Například:

empty infinite set

Aby k takovým případům nedocházelo, musí uživatel při definici nového typu zároveň dokázat, že tento typ je neprázdný, tedy že existuje nějaký objekt tohoto typu. Takový důkaz je vyžadován při definici módu a také je potřeba dokázat, které posloupnosti atributů aplikované na konkrétní typ definují neprázdný typ. Pokud nebude uvedeno jinak, budeme dále implicitně předpokládat, že pracujeme s typy, které jsou neprázdné [Sch07].

2.4.5 Relace podtypu

V této části bude formálně definováno, kdy je některý typ Mizaru podtypem jiného.

[12] Uvedená definice je přesnější, než definice uvedená v [Wie, Sch07, Ban03], kde je jako obecný tvar uvedeno

$$\alpha_1\ \alpha_2 \ldots \alpha_k\ \Gamma \textbf{ of } t_1, t_2, \ldots, t_n$$

Definice uvedená zde připouští i aplikaci atributu na obecný typ, nejen na kořenový typ — $\alpha(\beta\ \theta)$.

Definice 8. Nechť jsou τ_1 a τ_2 typy, $\alpha_1, \alpha_2, \ldots, \alpha_k$ atributy aplikovatelné na τ_1, $\beta_1, \beta_2, \ldots, \beta_l$ atributy aplikovatelné na τ_2. Dále mějme typy $\theta_1 = \alpha_1\ \alpha_2 \ldots \alpha_k\ \tau_1$ a $\theta_2 = \beta_1\ \beta_2 \ldots \beta_l\ \tau_2$. Pak relace podtypu $\preceq$ na typech Mizaru je nejmenší relace uspořádání[13] splňující následující podmínky:

1. $\tau_1 \preceq \tau_2$ pokud jsou τ_1 a τ_2 kořenové typy a τ_2 je nadtyp τ_1 nebo jsou τ_1 a τ_2 stejné kořenové typy
2. $\theta_1 \preceq \theta_2$ pokud je $\tau_1 \preceq \tau_2$ a zároveň je $\{\beta_1, \beta_2, \ldots, \beta_l\} \subseteq \Theta(\{\alpha_1, \alpha_2, \ldots, \alpha_k\}, \tau_1)$

2.4.6 Typ termu

V tomto oddíle bude definováno, jak lze odvodit typ termu.

Definice 9. Definujme zobrazení $\Lambda(t, \Delta)$ z termů jazyka Mizar do typů jazyka Mizar, kde Δ je kontext, ve kterém jsou určeny typy všech volných proměnných termu t:

1. Pokud je t proměnná, pak její typ musí být uveden v Δ. Nechť je $\Delta(t) = A\ \theta$, kde A je množina atributů. Potom definujme:
$$\Lambda(t, \Delta) = \Theta(A, \theta) \cup A\ \theta$$
2. Pokud je t funktor, nechť je $A\ \theta$ jeho návratový typ. Potom definujme:
$$\Lambda(t, \Delta) = \Theta(A \cup \Theta(\varphi), \theta) \cup A\ \cup \Theta(\varphi)\ \theta$$

2.4.7 Kvalifikující formule

Mizar obsahuje dvě kvalifikující formule — typovou a atributovou kvalifikující formuli.

Typová kvalifikující formule

Typová kvalifikující formule má obecně tvar:

$$t\ \textsf{is}\ \tau$$

Kde t je term a τ typ. Tuto formuli lze obecně dokazovat pomocí definičních vět, které jsou automaticky generovány například při definici módu — více viz 2.5.3.

Typovou kvalifikující formuli však lze odvodit i pomocí těchto dvou pravidel, která systém Mizar automaticky používá:

$$\frac{\Lambda(t, \Delta) = \tau}{\Delta \vdash t\ \textsf{is}\ \tau}$$

$$\frac{\Delta \vdash t\ \textsf{is}\ \theta \qquad \theta \preceq \theta'}{\Delta \vdash t\ \textsf{is}\ \theta'}$$

[13] reflexivní, antisymetrická a transitivní relace

Atributová kvalifikující formule

Atributová kvalifikující formule má obecně tento tvar:

$$t \textbf{ is } \alpha$$

Kde t je term a α je atribut. Aby tato kvalifikace měla smysl, musí platit [Muz93]:

$$\Lambda(t,\Delta) \in \Omega(\alpha)$$

Důvodem je vyzískání skrytých parametrů atributu — více viz 2.5.7.

Atributové kvalifikující formule lze obecně dokazovat stejně jako u typových kvalifikujících formulí pomocí definičních vět — více viz 2.5.7.

Tyto formule lze ale odvodit i pomocí následujícího pravidla, které systém Mizar automaticky používá, kde A je množina atributů aplikovatelná na typ θ:

$$\frac{\Delta \vdash t \textbf{ is } A\ \theta \qquad \alpha \in A}{\Delta \vdash t \textbf{ is } \alpha}$$

Formule t **is** $\alpha_1 \ldots \alpha_k$ je jen zkratka za

$$t \textbf{ is } \alpha_1 \textbf{ \& } t \textbf{ is } \alpha_2 \textbf{ \& } \ldots \textbf{ \& } t \textbf{ is } \alpha_k$$

Pro tuto formuli tedy musí platit analogická podmínka jako v případě kvalifikace jen jedním atributem [Muz93]:

$$\Lambda(t,\Delta) \in \Omega(\alpha_1) \wedge \Lambda(t,\Delta) \in \Omega(\alpha_2) \wedge \ldots \wedge \Lambda(t,\Delta) \in \Omega(\alpha_k)$$

Což lze přepsat jako:

$$\Lambda(t,\Delta) \in \Omega(\alpha_1) \cap \Omega(\alpha_2) \cap \cdots \cap \Omega(\alpha_k)$$

2.5 Popis formátu definic

Informace jsou čerpány ze zastaralého [Muz93], dále pak z oficiální definice syntaxe Mizaru [miz] a v případě atributů a registrací ještě z [Sch07]. Nemalá část uváděných informací je pak vlastní výzkum vycházející ze samotného používání systému Mizar. Jak už bylo zmíněno v 2, není popisován celý jazyk Mizaru, ale v části, která je popisována, je snaha o co největší přesnost.

Pro potřeby všech následujících definic budeme předpokládat, že σ je permutace na množině $\{1,\ldots,n\}$.

2.5.1 Kontext

Jak bylo uvedeno v 2.4, každá proměnná při svém uvedení musí mít definovaný svůj typ. V následujících popisech formátů definic existuje na začátku definice obecná konstrukce k uvedení nových proměnných — **let** sekce. Protože je společná pro všechny definice, bude obecně popsána na začátku. Tato sekce je důležitá především v tom, že uvádí veškeré volné proměnné, které se v příslušné definici můžou objevit. Jiné volné proměnné jsou zakázány.

Nejdříve zavedeme nějaké značení.

Definice 10. Definujme dva nové zápisy:

1. Zápis $\theta(x_1,\ldots,x_n)$ označuje typ formátu $A\ \Gamma$ **of** $t_1,t_2,\ldots,t_k$, kde A je množina atributů aplikovatelná na příslušný kořenový typ a $t_1,t_2,\ldots,t_k$ jsou termy a kde především platí, že pro každé $i=1,\ldots,k$, když je x proměnná obsažená v t_i, pak $x=x_j$ pro nějaké $j=1,\ldots,n$.

2. Zápis $\theta[x_1,\ldots,x_n]$ vyjadřuje to samé co $\theta(x_1,\ldots,x_n)$ a navíc musí platit, že pro každé $i=1,\ldots,n$ existuje nějaké $j=1,\ldots,k$ takové, že x_i je obsažená v t_j.

Obecný tvar **let** sekce je následující:

$$\textbf{let}\ x_1\ \textbf{be}\ \theta_1, x_2\ \textbf{be}\ \theta_2,\ldots,x_n\ \textbf{be}\ \theta_n;$$

Tato konstrukce uvádí nové proměnné x_1 až x_n a přiřazuje jim příslušné typy θ_1 až θ_n. Na tom by nebylo zase tak moc zajímavého, důležité je však si uvědomit, že při definici příslušného typu θ_i může být použita proměnná jako term, který instanciuje příslušný mód. A protože tato proměnná musí být již někdy dříve uvedena a všechny volné proměnné v definici musejí být uvedeny v její **let** sekci, musí mít tato proměnná typ θ_j pro nějaké $j<i$.

Definice 11. Obecný tvar **let** sekce tedy můžeme upřesnit do následujícího formátu

$$\textbf{let}\ x_1\ \textbf{be}\ \theta_1, x_2\ \textbf{be}\ \theta_2(x_1),\ldots,x_n\ \textbf{be}\ \theta_n(x_1,\ldots,x_{n-1});$$

a **let** sekci v tomto formátu budeme říkat *kontext*.

Pořadí definovaných proměnných v kontextu je tedy významné a obecně nemůže být libovolně zaměněno.

2.5.2 Slučitelnost dosazení s kontextem

Nyní již máme definovány všechny potřebné pojmy, abychom mohli definovat slučitelnost dosazení, kterýžto pojem byl doposud používán spíše neformálně.

Definice 12. Nechť je C kontext tvaru:

$$\textbf{let } x_1 \textbf{ be } \theta_1, x_2 \textbf{ be } \theta_2(x_1), \ldots, x_n \textbf{ be } \theta_n(x_1, \ldots, x_{n-1});$$

Pak χ nazveme dosazení vzhledem ke kontextu C, pokud je χ zobrazení z množiny $\{x_1, \ldots, x_n\}$ do termů jazyka Mizaru.

Definice 13. Nechť je C kontext tvaru:

$$\textbf{let } x_1 \textbf{ be } \theta_1, x_2 \textbf{ be } \theta_2(x_1), \ldots, x_n \textbf{ be } \theta_n(x_1, \ldots, x_{n-1});$$

Řekneme, že dosazení χ je slučitelné s C, pokud je χ dosazení vzhledem k C a pokud platí:

$$\begin{aligned}
\Lambda(\chi(x_1), \Delta) &\preceq \theta_1 \\
\Lambda(\chi(x_2), \Delta) &\preceq \theta_2(x_1)[\chi(x_1)/x_1] \\
&\vdots \\
\Lambda(\chi(x_n), \Delta) &\preceq \theta_n(x_1, \ldots, x_{n-1})[\chi(x_1)/x_1, \ldots, \chi(x_{n-1})/x_{n-1}]
\end{aligned}$$

Pokud tedy máme definici módu:

definition
  let x_1 **be** θ_1, x_2 **be** $\theta_2(x_1), \ldots, x_n$ **be** $\theta_n(x_1, \ldots, x_{n-1})$;
  mode Γ **of** $x_{\sigma(1)}, \ldots, x_{\sigma(n)}$ -> ...
  ⋮
end;

Pak zápis

$$\Gamma \textbf{ of } t_1, t_2, \ldots, t_n$$

definuje dosazení χ vzhledem ke kontextu módu Γ následujícím předpisem:

$$\chi(x_j) := t_i$$

kde $\sigma(i) = j$.

2.5.3 Definice módu

Nový mód lze definovat dvěma způsoby, pomocí konstrukce **means** a pomocí konstrukce **is**.

Definice pomocí means

Definice módu pomocí konstrukce **means** má následující tvar:

definition
  let x_1 **be** θ_1, x_2 **be** $\theta_2(x_1)$, ..., x_n **be** $\theta_n(x_1,\ldots,x_{n-1})$;
  mode Γ **of** $x_{\sigma(1)},\ldots,x_{\sigma(n)}$ -> $\theta(x_1,\ldots,x_n)$ **means**
  : L : $\delta(x_1,\ldots,x_n,$ **it**$)$;
  existence ...;
end;

Kde Γ je název nového módu a $\theta(x_1,\ldots,x_n)$ je nadtyp nově definovaného módu. Důležité je povšimnout si, že parametry módu $x_{\sigma(1)},\ldots,x_{\sigma(n)}$ odpovídají právě jedna k jedné deklarovaným proměnným v kontextu, byť můžou být uvedeny v jiném pořadí než v jakém jsou uvedeny v kontextu.

Vlastní mód je potom definován pomocí formule δ, která obsahuje speciální proměnnou **it**. Formule δ pak představuje podmínku, zda **it** patří do nově definovaného módu, nebo ne. Ještě lépe to může být pochopitelné na definiční větě, která je automaticky vytvořena systémem a na niž se lze později odkazovat pomocí identifikátoru L. Její obecný tvar je následující:

for x_1 **being** $\theta_1,\ldots,x_n$ **being** θ_n, y **being** θ **holds**
y **is** Γ **of** $x_{\sigma(1)},\ldots,x_{\sigma(n)}$ **iff** $\delta(x_1,\ldots,x_n,y)$;

V sekci **existence** pak musí být uveden důkaz následujícího tvrzení:

ex x **being** $\theta(x_1,\ldots,x_n)$ **st** $\delta(x_1,\ldots,x_n,x)$;

Tedy tvrzení, že existuje objekt, který splňuje podmínku módu, a tedy nově definovaný mód definuje neprázdný typ. Nutnost dokázat tento fakt byla diskutována v 2.4.4.

Definice pomocí is

Definice módu pomocí konstrukce **is** má následující tvar:

definition
  let x_1 **be** θ_1, x_2 **be** $\theta_2(x_1)$, ..., x_n **be** $\theta_n(x_1,\ldots,x_{n-1})$;
  mode Γ **of** $x_{\sigma(1)},\ldots,x_{\sigma(n)}$ **is** $\theta(x_1,\ldots,x_n)$;
end;

Tato konstrukce módu vlastně představuje možnost, jak lehce vytvářet jakési zkratky pro už existující typy. Například:

definition
  let H **be** Group;
  mode Endomorphism **of** H **is** Homomorphism **of** H,H;
end;

Protože jde o zkratku typu, který už existuje, není systémem Mizar vyžadována žádná **existence** sekce. Mizar přistupuje k této konstrukci jako k syntaktickému cukru, při zpracování článku je každý výskyt módu, který je definován pomocí **is** konstrukce, nahrazen typem uvedeným v jeho definici, tedy typem, který zkracuje [Muz93].

2.5.4 Definice predikátu

Definice predikátu má následující tvar:

definition
  let x_1 **be** θ_1, x_2 **be** $\theta_2(x_1), \ldots, x_n$ **be** $\theta_n(x_1, \ldots, x_{n-1})$;
  pred $x_{\sigma(1)}, \ldots, x_{\sigma(k)} \Pi x_{\sigma(k+1)}, \ldots, x_{\sigma(n)}$ **means**
  : L : $\delta(x_1, \ldots, x_n)$;
end;

Kde Π je název nového predikátu. Žádný existenční důkaz není vyžadován a obecný formát definiční věty je následující:

$$\textbf{for } x_1 \textbf{ being } \theta_1, \ldots, x_n \textbf{ being } \theta_n \textbf{ holds}$$
$$x_{\sigma(1)}, \ldots, x_{\sigma(k)} \Pi x_{\sigma(k+1)}, \ldots, x_{\sigma(n)} \textbf{ iff } \delta(x_1, \ldots, x_n);$$

Jako příklad nám může posloužit definice predikátu vyjadřujícího dělitelnost dvou celých čísel:

definition
  let i_1 **be** Integer, i_2 **be** Integer;
  pred i_1 divides i_2 **means**
  : Def9 :
  ex i_3 **being** Integer **st** $i_2 = i_1 * i_3$;
end;

2.5.5 Definice funktoru

Nový funktor[14] lze definovat dvěma způsoby, pomocí konstrukce **means** a pomocí konstrukce **equals**.

Definice pomocí means

Definice funktoru pomocí konstrukce **means** má následující tvar:

definition
  let x_1 **be** θ_1, x_2 **be** $\theta_2(x_1), \ldots, x_n$ **be** $\theta_n(x_1, \ldots, x_{n-1})$;
  func $(x_{\sigma(1)}, \ldots, x_{\sigma(k)}) \varphi (x_{\sigma(k+1)}, \ldots, x_{\sigma(n)})$ -> $\theta(x_1, \ldots, x_n)$ **means**
  : L : $\delta(x_1, \ldots, x_n, \textbf{it})$;
  existence ...;
  uniqueness ...;
end;

[14] Budeme používat terminologii Mizaru, která používá termín funktor pro funkci. Snaží se tak odlišit funkci v jazyce logiky a funkci v teorii množin (množina dvojic) [Wie]. Termín funktor tedy zde nemá nic společného s funktory v teorii kategorií.

kde φ je název nového funktoru a $\theta(x_1,\ldots,x_n)$ je typ hodnot, které funktor vrací, neboli typ oboru hodnot funktoru φ. Parametry funktoru

$$x_{\sigma(1)},\ldots,x_{\sigma(k)},x_{\sigma(k+1)},\ldots,x_{\sigma(n)}$$

opět odpovídají právě deklarovaným proměnným v kontextu.

Hodnoty funktoru jsou definovány pomocí formule δ, která představuje podmínku na návratovou hodnotu **it**, pokud známe n-tici argumentů funktoru. Aby takto definovaný funktor byl korektním funktorem, je potřeba dokázat o uvedené definici δ dvě vlastnosti:

a) První z nich se dokazuje v **existence** sekci. Dokazuje se následující tvrzení:

$$\textbf{ex } x \textbf{ being } \theta(x_1,\ldots,x_n) \textbf{ st } \delta(x_1,\ldots,x_n,x);$$

tedy, že podmínka δ definuje pro každou n-tici argumentů aspoň jednu návratovou hodnotu.

b) Druhá vlastnost se dokazuje v **uniqueness** sekci. Dokazuje se pro změnu toto tvrzení:

$$\textbf{for } y,z \textbf{ being } \theta(x_1,\ldots,x_n) \textbf{ st } \delta(x_1,\ldots,x_n,y) \textbf{ \& } \delta(x_1,\ldots,x_n,z) \textbf{ holds } y = z;$$

tedy, že návratová hodnota – pokud existuje – je určena podmínkou δ jednoznačně.

Zbývá ještě uvést formát definiční věty:

$$\textbf{for } x_1 \textbf{ being } \theta_1,\ldots,x_n \textbf{ being } \theta_n, y \textbf{ being } \theta \textbf{ holds } y = (x_{\sigma(1)},\ldots,x_{\sigma(k)})\varphi(x_{\sigma(k+1)},\ldots,x_{\sigma(n)}) \textbf{ iff } \delta(x_1,\ldots,x_n,y);$$

Definice pomocí equals

Definice funktoru pomocí konstrukce **equals** má následující tvar:

```
definition
  let x_1 be θ_1, x_2 be θ_2(x_1), ..., x_n be θ_n(x_1, ..., x_{n-1});
  func (x_σ(1), ..., x_σ(k))φ(x_σ(k+1), ..., x_σ(n)) -> θ(x_1, ..., x_n) equals
  : L : δ(x_1, ..., x_n);
  coherence ...;
end;
```

Tento formát definice mnohem více odpovídá tomu, jak jsou funkce definovány, než je tomu v případě definice pomocí konstrukce **means**. A sice pomocí výrazu, který říká, jak hodnotu funktoru „vypočítat“. Proto není potřeba dokazovat

podmínku existence a jednoznačnosti jako v případě definice pomocí konstrukce **means**. Zato je potřeba dokázat, že výraz definující hodnotu funktoru má skutečně typ $\theta(x_1,\ldots,x_n)$. Je potřeba dokázat následující tvrzení v sekci **coherence**:

$$\delta(x_1,\ldots,x_n) \textbf{ is } \theta(x_1,\ldots,x_n);$$

Formát definiční věty je pak velmi přímočarý:

$$\textbf{for } x_1 \textbf{ being } \theta_1,\ldots,x_n \textbf{ being } \theta_n \textbf{ holds}$$
$$(x_{\sigma(1)},\ldots,x_{\sigma(k)})\varphi(x_{\sigma(k+1)},\ldots,x_{\sigma(n)}) = \delta(x_1,\ldots,x_n);$$

2.5.6 Redefinice

Redefinice jsou poměrně důležitým prvkem Mizaru z pohledu typového systému. Umožňují změnit – redefinovat— původní definici módu nebo funktoru. Nás bude zajímat změna definice nadtypu, respektive typu oboru hodnot [15]. Tento typ nelze změnit libovolně, nýbrž jen na typ, který je speciálnější – menší vzhledem k relaci podtypu $\preceq$ – než typ uvedený v původní definici. Jinak řečeno, typ, který byl uveden v původní definici, je změněn na speciálnější. Možnost takovéto redefinice je důležitá, protože praxe psaní článků v Mizaru odpovídá zásadě, aby uživatel začal s co možná nejobecnější definicí a postupně ji specializoval.

Abychom mohli redefinici provést, musí v následujících definicích platit tyto dvě podmínky:

1. Nechť je kontext původní definice následující:

   $$\textbf{let } x_1 \textbf{ be } \theta_1', x_2 \textbf{ be } \theta_2'(x_1),\ldots,x_n \textbf{ be } \theta_n'(x_1,\ldots,x_{n-1});$$

   Potom musí platit, že pro každé $i = 1,\ldots,n$ je $\theta_i \preceq \theta_i'$.

2. Nechť je původní nadtyp, respektive typ oboru hodnot θ', pak musí platit, že $\theta \preceq \theta'$.

Redefinice módu

Redefinice módu má následující tvar:

definition
  let x_1 **be** θ_1, x_2 **be** $\theta_2(x_1),\ldots,x_n$ **be** $\theta_n(x_1,\ldots,x_{n-1})$;
  redefine mode Γ **of** $x_{\sigma(1)},\ldots,x_{\sigma(n)}$ -> $\theta(x_1,\ldots,x_n)$;
  coherence ...;
end;

kde $\theta(x_1,\ldots,x_n)$ je nový nadtyp. Aby byla redefinice korektní, musíme v sekci **coherence** dokázat následující tvrzení:

$$\textbf{for } x \textbf{ being mode } \Gamma \textbf{ of } x_{\sigma(1)},\ldots,x_{\sigma(n)} \textbf{ holds } x \textbf{ is } \theta(x_1,\ldots,x_n);$$

[15] Lze změnit i vlastní formuli, která definuje mód nebo funktor. Z hlediska typového systému je ale tento typ redefinice nepodstatný, proto se mu nebudeme věnovat.

Redefinice funktoru

Redefinice funktoru má následující tvar:

definition
  let x_1 **be** θ_1, x_2 **be** $\theta_2(x_1), \ldots, x_n$ **be** $\theta_n(x_1, \ldots, x_{n-1})$;
  redefine func $(x_{\sigma(1)}, \ldots, x_{\sigma(k)})\varphi(x_{\sigma(k+1)}, \ldots, x_{\sigma(n)})$ -> $\theta(x_1, \ldots, x_n)$;
  coherence ...;
end;

kde $\theta(x_1, \ldots, x_n)$ je nový typ oboru hodnot. Aby byla redefinice korektní, musíme opět v sekci **coherence** dokázat následující tvrzení:

$$(x_{\sigma(1)}, \ldots, x_{\sigma(k)})\varphi(x_{\sigma(k+1)}, \ldots, x_{\sigma(n)}) \textbf{ is } \theta(x_1, \ldots, x_n);$$

2.5.7 Definice atributu

Definice atributu má následující tvar:

definition
  let x_1 **be** θ_1, x_2 **be** $\theta_2(x_1), \ldots, x_n$ **be** $\theta_n(x_1, \ldots, x_{n-1}), y$ **be** $\theta[x_1, \ldots, x_n]$;
  attr y **is** α **means**
  : L : $\delta(x_1, \ldots, x_n, y)$;
end;

Kde α je název nového atributu a $\theta(x_1, \ldots, x_n)$ je mateřský typ nově definovaného atributu. Zde se poprvé setkáváme se značením $\theta[x_1, \ldots, x_n]$ — mateřský typ tedy musí povinně použít všechny definované nové proměnné v kontextu. I atribut má tedy něco, čemu bychom mohli říkat parametry — $x_1, \ldots, x_n$, jsou to ale skryté parametry. Při aplikaci atributu na typ se nikde explicitně neuvádějí. Jsou ale získány z typu, na který je atribut aplikován. Proto musí být v mateřském typu všechny doposud definované proměnné kontextu použité, aby se při aplikaci atributu na typ mohly získat všechny hodnoty jeho skrytých parametrů. Dosazením za tyto skryté parametry pak vzniká adjektivum. Vidíme zde tedy podobný vztah jako byl mezi módem a kořenovým typem, který byl uveden v 2.4.3. I zde je atribut jakýsi konstruktor a adjektivum je konstrukce. Adjektivum zde však nevzniká explicitní instanciací parametrů jako v případě módu, ale při aplikaci atributu na nějaký typ, kdy jsou hodnoty parametrů vyzískány z typu, na nějž je atribut aplikován.

Vlastní atribut je, jak už mohlo být několikrát viděno, definován pomocí formule δ, která obsahuje speciální proměnnou **it**. Formule δ opět představuje podmínku, která určuje, zda objekt **it** patří do nově definovaného atributu, nebo ne. Pod identifikátorem L pak bude v systému vystupovat definiční věta následujícího tvaru:

$$\textbf{for } x_1 \textbf{ being } \theta_1, \ldots, x_n \textbf{ being } \theta_n, y \textbf{ being } \theta \textbf{ holds}$$
$$y \textbf{ is } \alpha \textbf{ iff } \delta(x_1, \ldots, x_n, y);$$

Jistou odlišností definice atributu od ostatních zde uváděných definic je vlastnost, že výše uvedený tvar definice definuje nejen nový atribut α, ale také atribut s názvem non α – například atribut non empty. Tento atribut představuje negaci původního atributu. Není pro něj vygenerována žádná definiční věta, nýbrž jsou v samotném Mizaru zabudována následující odvozovací pravidla:

$$\frac{\Delta \vdash t \textbf{ is non } \alpha}{\Delta \vdash \textbf{not } (t \textbf{ is } \alpha)}$$

$$\frac{\Delta \vdash \textbf{not } (t \textbf{ is } \alpha)}{\Delta \vdash t \textbf{ is non } \alpha}$$

kde t je term a α atribut. Tato pravidla už umožňují dokázat, že nějaký term je non α, pomocí definiční věty pro samotné α.

Jak bylo popsáno v 2.4.4, o každém typu, pokud jej chceme používat, musí být dokázáno, že je neprázdný. V případě kořenových typů se tak děje pomocí **existence** sekce v definici módu, ze kterého kořenový typ vzniknul. Aplikací atributů na kořenové typy vznikají nové typy, tedy i zde musí přijít ke slovu mechanismus, pomocí kterého dokážeme neprázdnost tohoto nově vzniklého typu. Děje se tak pomocí existenciálních registrací.

Jejich obecný formát je následující:

registration
  let x_1 **be** θ_1, x_2 **be** $\theta_2(x_1), \ldots, x_n$ **be** $\theta_n(x_1, \ldots, x_{n-1})$;
  cluster $\alpha_1 \ldots \alpha_k\ \theta[x_1, \ldots, x_n]$;
  existence ...;
end;

Nový typ vystupuje za klíčovým slovem **cluster** :

$$\alpha_1 \ldots \alpha_k\ \theta[x_1, \ldots, x_n]$$

Samotná neprázdnost nového typu se dokazuje v sekci **existence** dokázáním následující formule:

$$\textbf{ex } x \textbf{ being } \theta[x_1, \ldots, x_n] \textbf{ st } x \textbf{ is } \alpha_1 \ldots \alpha_k$$

Velmi příjemnou vlastností je princip dědičnosti existenciálních registrací. Nechť jsou A a B množiny atributů a platí, že $A \subseteq B$. Potom registrace atributů B s nějakým typem θ způsobí, že registrace atributů A se stejným typem θ není potřeba. Řečeno jinými slovy, registrace atributů B s typem θ způsobí registraci každé podmnožiny atributů B s typem θ.

2.5.8 Zaokrouhlování

Mizar umí tzv. zaokrouhlování[16]. Umožňuje automatické doplnění atributů podle určitých pravidel. S prvním z nich jsme se už setkali v 2.4.3. Šlo o odvozování jedné množiny atributů druhou. Toto odvozování je definováno pomocí podmínkové registrace. Další možnost, jak definovat zaokrouhlování, je pomocí funktorové registrace.

Podmínková registrace

Obecný formát podmínkové registrace je následující:

registration
  let x_1 **be** θ_1, x_2 **be** $\theta_2(x_1), \ldots, x_n$ **be** $\theta_n(x_1, \ldots, x_{n-1})$;
  cluster $\alpha_1 \ldots \alpha_k$ -> $\beta_1 \ldots \beta_l$ $\theta[x_1, \ldots, x_n]$;
  coherence ...;
end;

Tato definice definuje následující automatické odvození adjektiv:

$$\{\alpha_1, \ldots, \alpha_k\} \overset{\theta}{\Longrightarrow} \{\beta_1, \ldots, \beta_l\}$$

Formát definice umožňuje, aby nebyl žádný předpoklad odvození atributů, tedy aby se v definici $k = 0$. Avšak $l \geq 1$.

Funktorová registrace

Obecný formát funktorové registrace je následující:[17]

registration
  let x_1 **be** θ_1, x_2 **be** $\theta_2(x_1), \ldots, x_n$ **be** $\theta_n(x_1, \ldots, x_{n-1})$;
  cluster $(x_{\sigma(1)}, \ldots, x_{\sigma(k)})\varphi(x_{\sigma(k+1)}, \ldots, x_{\sigma(n)})$ -> $\alpha_1 \ldots \alpha_k$;
  coherence ...;
end;

[16] anglicky *rounding up*

[17] V Mizaru je použit silnější mechanismus. Je možné obecně registrovat termy a ne jen funktory — tedy některé proměnné x_i v uváděné definici můžou být instanciovány.

Kapitola 3

Překlad typového systému

3.1 HOL Light

Systém HOL Light používá klasickou logiku vyššího řádu s axiomy nekonečna, extensionality a výběru. Je založen na jednoduše typovaném λ-kalkulu s polymorfními typovými proměnnými (Hindley-Milner), tedy každý term je buď proměnná, konstanta, aplikace nebo abstrakce [JH06, Gor08].

Systém je implementován podle LCF metodologie — obsahuje tzv. „dokazovací jádro“[1], které definuje deset základních odvozovacích pravidel. Všechny dokázané věty (teorémy) v systému mají typ `thm` a odvozovací pravidla z dokazovacího jádra nejsou pak nic jiného než funkce, které mají jako návratový typ právě typ `thm`. Například odvozovací pravidlo `EQ_MP`, modus ponens pro rovnost, je funkce, která má typ $\texttt{thm} \times \texttt{thm} \to \texttt{thm}$. Pokud jako argumenty dostane věty $\Gamma \vdash p = q$ a $\Delta \vdash p$, vrátí novou větu $\Gamma \cup \Delta \vdash q$. Definuje tedy odvozovací pravidlo [Har07]:

$$\frac{\Gamma \vdash p = q \qquad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}$$

Používání pouze deseti základních odvozovacích pravidel pro dokazování by bylo příliš pracné. Uživatel však může naprogramovat libovolně sofistikované další dokazovací procedury. Může tedy vytvořit libovolné další vrstvy, které mu umožní dokazování na mnohem vyšší úrovni, než jakou poskytují základní odvozovací pravidla. Klíčovou vlastností LCF metodologie je, že typ `thm` je abstraktní a hodnoty tohoto typu můžou být vytvářeny jen předem definovanými funkcemi — a to je právě těch deset základních odvozovacích pravidel dokazovacího jádra. Pokud tedy uživatel naprogramuje složitější odvozovací pravidla, musí nutně při jejich implementaci použít jen odvozovací pravidla z dokazovacího jádra. Otázka korektnosti odvozování v systému HOL Light se tak díky LCF metodologii redukuje na otázku korektnosti dokazovacího jádra. A toto jádro, protože obsahuje

[1] anglicky *proof kernel*

jen deset základních odvozovacích pravidel, představuje jen velmi malé množství kódu[2], které musí být ověřeno, že je korektní. Proto je systém HOL Light velmi lehce rozšiřitelný a zároveň velmi spolehlivý [Har07, JH06].

3.2 Neformální popis

Vlastní systém HOL Light nabízí mechanismus pro vytváření nových typů. Pokud máme nějakou neprázdnou množinu hodnot typu θ definovanou její charakteristickou funkcí `P` typu $\theta \to$ `bool`, lze pomocí vhodného volání funkce `new_type_definition` definovat nový typ, který bude reprezentovat právě hodnoty dané charakteristickou funkcí `P`. Stačí pouze dokázat, že `P` definuje neprázdnou množinu. Funkce `new_type_definition` pak vrací dvojici bijekcí, které zobrazují nový typ do typu θ a zpět. To je takřka stejný mechanismus, jaký je používán v Mizaru. Tam jsou také nové typy tvořeny jako podmnožiny nějakého už existujícího typu pomocí podmínky na prvky nového typu. Problémem však je, že Mizar používá závislé typy. A tento typ může třeba obsahovat proměnnou, přes kterou se kvantifikuje ve formuli, kde je tento typ přítomen, například:

$$\textbf{ex } x \textbf{ being } \theta \textbf{ st for } y \textbf{ being } \theta'[x] \textbf{ holds } \psi(y);$$

Pokud bychom dosadili za x v existenční kvantifikaci svědka, změnil by se i typ proměnné y, protože její typ závisí na proměnné x. A to představuje problém, protože pokud by $\theta'[x]$ byl typ systému HOL Light, byl by typ proměnné y pevně dán a nemohl by se změnit tím, že bychom za x dosadili nějakého svědka.

Proto se zdá, že přímo typový systém systému HOL Light nepůjde využít. Budeme muset typový systém Mizaru nějak simulovat přímo logickými prostředky systému HOL Light. Jako inspirace může sloužit práce [BJ93], ve které autoři navrhli převod závislých typů — formulovaných Martin-Löfem v jeho intuicionistické teorii typů — do systému HOL. Základní myšlenka je prostá, reprezentovat typy jako predikáty. Tuto základní myšlenku použijeme a modifikujeme ji pro potřeby překladu typového systému Mizaru.

Jako predikát je v systému HOL Light chápána jakákoliv funkce následujícího tvaru:

$$\texttt{P} : \sigma \to \texttt{bool}$$

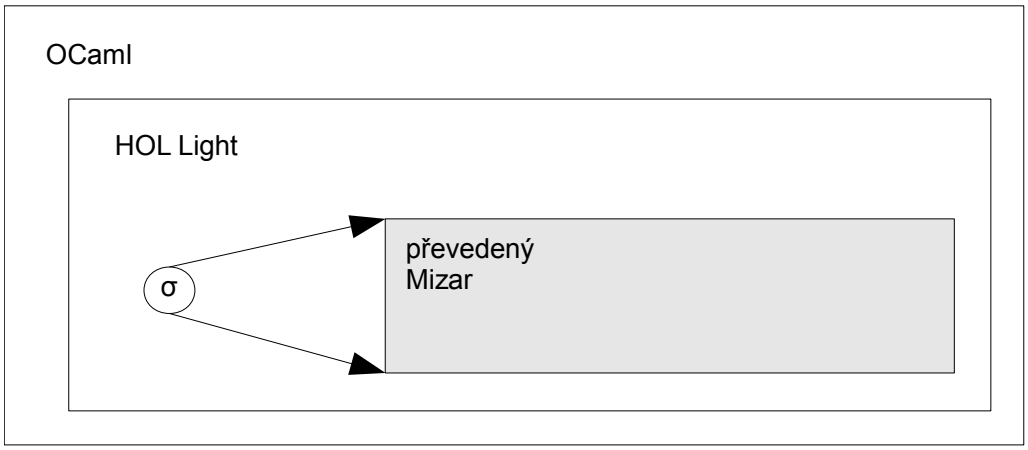
Ta tvrdí o argumentu typu σ zda má, či nemá vlastnost `P`. My jako predikáty pro potřeby našeho překladu budeme chápat funkce poněkud speciálnějšího tvaru:

$$\texttt{P} : \sigma_1 \to \cdots \to \sigma_n \to \sigma \to \texttt{bool}$$

Tedy budeme uvažovat, že predikát může mít nějaké parametry. V uvedeném tvaru je těchto parametrů n a mají typy σ_i. Ve skutečnosti nebudeme potřebovat

[2] jen okolo 500 řádků kódu v OCamlu včetně pomocných funkcí

takovou typovou obecnost parametrů predikátu, nýbrž si pro celý překlad zvolíme jeden pevný atomický typ σ[34], který bude reprezentovat objekty systému HOL Light, a těmto objektům budeme přiřazovat přeložené typy Mizaru. Situace se může stát poněkud nepřehlednou, neboť tu máme několik pojmů typu, které vystupují na různé metaúrovni.



Obrázek 3.1: Metaúrovně typových systémů

Schematicky je situace zachycena na obrázku 3.1. OCaml je programovací jazyk, ve kterém je systém HOL Light implementován. Ocaml zde hraje roli matajazyka, pomocí kterého uvažujeme o logice systému HOL Light.[5] Pomocí funkcí Ocamlu lze přidávat nové definice do systému HOL Light nebo třeba manipulovat s větami systému HOL Light. Ocaml, jako celá řada jiných funkcionálních jazyků, obsahuje typy. Systém HOL Light také obsahuje typy, jsou to typy termů, které vystupují v jeho logice. Jsou to tedy typy výrazů z jazyka systému HOL Light. To je rozdíl mezi OCamlem sloužícím jako metajazyk a systémem HOL Light jako jazykem.

Situace se ještě dále zkomplikuje, pokud začneme uvažovat typy přeložené z Mizaru. Ty představují další úroveň a třetí „druh“ typů. Tyto typy budou simulovány pomocí predikátů systému HOL Light a budou přiřazovány objektům logiky systému HOL Light, které mají pevně zvolený typ (ve smyslu typ systému HOL Light) σ. Snad bude po tomto vysvětlení dále z kontextu patrné, k jaké úrovni v dané chvíli zmíněný typ patří.

Uveďme příklad, který základní myšlenku překladu ještě více objasní. Mějme následující typovou kvalifikaci v Mizaru:

$$y \textbf{ is } \mathsf{empty\ Element} \textbf{ of } X$$

[3]Tedy σ není ani funkční typ, ani typová proměnná.

[4]V implementaci je zvolen typ `set`, který je definován pomocí funkce `new_type`.

[5]Ocaml lze považovat za dialekt jazyka ML, jehož název je zkratkou termínu *metalanguage* — česky *metajazyk*.

Tato kvalifikace bude přeložena do systému HOL Light následovně:

$$\texttt{I}_{\texttt{empty}}\ \texttt{y}\ /\backslash\ \texttt{I}_{\texttt{Element}}\ \texttt{X}\ \texttt{y}$$

Symboly $\texttt{I}_{\texttt{empty}}$ a $\texttt{I}_{\texttt{Element}}$ jsou predikáty, které odpovídají atributu empty, respektive kořenovému typu Element. Jejich typy jsou následující:

$$\texttt{I}_{\texttt{empty}} : \sigma \to \texttt{bool}$$
$$\texttt{I}_{\texttt{Element}} : \sigma \to \sigma \to \texttt{bool}$$

$\texttt{I}_{\texttt{empty}}$ je tedy predikát, který přiřazuje objektům typu σ vlastnost býti prázdným a predikát $\texttt{I}_{\texttt{Element}}$ přiřazuje zase objektům typu σ vlastnost býti podmnožinou X, což je jeho první parametr.

Pokud si tedy pevně zvolíme nějaký typ σ, budou mít predikáty s n parametry následující tvar:

$$\texttt{P} : \underbrace{\sigma \to \cdots \to \sigma}_{\text{n-krát}} \to \sigma \to \texttt{bool}$$

V následující tabulce je neformálně shrnuto, jak je rekonstrukce provedena. Detaily budou vysvětleny v dalším textu.

Mizar	rekonstrukce v systému HOL Light
kořenový typ	predikát
adjektivum	predikát
typ	konjunkce adjektiv a kořenového typu
deklarace přiřazující proměnné x typ θ	předpoklad, že x splňuje formuli odpovídající typu θ
kontext	seznam předpokladů ve speciálním tvaru
predikát	predikát sytému HOL Light (libovolná funkce do typu bool)
funktor	funkce

3.3 Definiční rozšíření systému HOL Light

Ze zatím popsané idey překladu plyne, že bude třeba rozšířit systém HOL Light o nové konstanty, které budou odpovídat námi nově definovaným funkcím reprezentujícím typy, funktory a predikáty. Jednou z možností je použít funkci new_constant a poté vlastními prostředky dokázat definiční věty, které by tyto nové konstanty charakterizovaly. Systém HOL Light ale poskytuje několik funkcí pro definiční rozšíření:

- new_definition – funkce má typ term → thm. Jako argument vystupuje term v předem určeném tvaru. Tento tvar může být obecnější než tvar, který zde bude uveden. Nám však postačí jeho speciálnější podoba. Tedy podoba, kterou budeme využívat, je následující:

$$\texttt{c}\ \texttt{x}_1\ \texttt{x}_2\ ...\ \texttt{x}_\texttt{n} = \texttt{t}$$

kde c je název nové konstanty, $\texttt{x}_1\ \texttt{x}_2\ ...\ \texttt{x}_\texttt{n}$ jsou její argumenty a t je hodnota nové konstanty, ve které všechny volné proměnné jsou obsaženy mezi $\texttt{x}_1\ \texttt{x}_2\ ...\ \texttt{x}_\texttt{n}$. Funkce vrací definiční větu charakterizující nově uvedenou konstantu. Tato věta má následující tvar:

$$\vdash\ !\texttt{x}_1\ \texttt{x}_2\ ...\ \texttt{x}_\texttt{n}.\texttt{c}\ \texttt{x}_1\ \texttt{x}_2\ ...\ \texttt{x}_\texttt{n} = \texttt{t}$$

- `new_specification` – funkce má typ $\texttt{string list} \rightarrow \texttt{thm} \rightarrow \texttt{thm}$. Funkce se liší od funkce `new_definition` tím, že nedefinuje novou konstantu pomocí termu, který by udával její hodnotu (tedy pomocí rovnosti), nýbrž pomocí tvrzení, které charakterizuje její hodnotu. Tvar volání, který zde bude uveden, opět odpovídá poněkud speciálnějšímu tvaru, ale plně bude stačit našim potřebám:

  $$\texttt{new\_specification}\ [\text{"}\texttt{c}\text{"}]\ \vdash\ ?\texttt{x}.\texttt{t}$$

  V prvním argumentu se uvede název nové konstanty. Term t v druhém argumentu představuje ono zmiňované tvrzení, které charakterizuje konstantu c tím, že na ni klade podmínku skrze jedinou volnou proměnnou x v termu t. Druhý argument je pak tvrzení, které tvrdí, že existuje taková konstanta, která danou podmínku splňuje. Návratovou hodnotou funkce `new_specification` je následně tvrzení, které novou konstantu charakterizuje. Vzhledem k povaze druhého argumentu je jeho tvar velmi přímočarý:

  $$\vdash\ \texttt{t}[\texttt{c}/\texttt{x}]$$

3.4 Překlad jazyka

V této části definujeme formálně zobrazení T, které nám bude překládat konstrukce jazyka Mizaru do systému HOL Light. Nejdříve vyřešme problém s identifikátory pro módy, atributy, predikáty a funktory. Předpokládejme, že pro každý takový identifikátor A lze nalézt identifikátor $\texttt{I}_\texttt{A}$, který bude odpovídat lexikálním konvencím systému HOL Light a bude jednoznačný. Dále definujeme pomocná zobrazení T_{atr}, T_{typ} a T_{con}, která nám budou překládat atributy, typy a kontexty Mizaru.

Definice 14. Definujme zobrazení T_{atr} z atributů Mizaru do jazyka systému HOL Light. Nechť je atribut α aplikovatelný na typ τ a je definován následující definicí:

definition
  let x_1 **be** θ_1, x_2 **be** $\theta_2(x_1), \ldots, x_n$ **be** $\theta_n(x_1, \ldots, x_{n-1}), y$ **be** $\theta[x_1, \ldots, x_n]$;
  attr y **is** α **means**
  ⋮
end;

A nechť je τ' kořenový typ vyskytující se v typu τ. Pak nutně existuje[6] dosazení slučitelné s kontextem α takové, že $\theta[\chi] = \tau'$. Potom definujme

$$\mathrm{T}_{\mathrm{atr}}(\alpha, \tau, t) = \mathtt{I}_{\alpha}(\mathrm{T}(\chi(x_1)))\ldots(\mathrm{T}(\chi(x_n)))(\mathrm{T}(t))$$

Definice 15. Definujme zobrazení $\mathrm{T}_{\mathrm{typ}}$ z typů Mizaru do jazyka systému HOL Light. Definice sleduje induktivní definici typů uvedenou v 2.4.4.

- Pokud je θ kořenový typ tvaru:

  $$\Gamma \textbf{ of } t_1, \ldots, t_n$$

  pak

  $$\mathrm{T}_{\mathrm{typ}}(\theta, t) = \mathtt{I}_{\Gamma}(\mathrm{T}(t_1))\ldots(\mathrm{T}(t_n))(\mathrm{T}(t))$$

- Pokud je θ typ tvaru:

  $$\alpha_1 \ldots \alpha_k \ \tau$$

  pak

  $$\mathrm{T}_{\mathrm{typ}}(\theta, t) = \mathrm{T}_{\mathrm{atr}}(\alpha_1, \tau, t) \ /\backslash \ \ldots \ /\backslash \ \mathrm{T}_{\mathrm{atr}}(\alpha_n, \tau, t) \ /\backslash \ \mathrm{T}_{\mathrm{typ}}(\tau, t)$$

Definice 16. Nechť je C kontext tvaru:

$$\textbf{let } x_1 \textbf{ be } \theta_1, x_2 \textbf{ be } \theta_2(x_1), \ldots, x_n \textbf{ be } \theta_n(x_1, \ldots, x_{n-1});$$

Potom definujme zobrazení $\mathrm{T}_{\mathrm{con}}$ z kontextů jazyka Mizar do seznamů předpokladů systému HOL Light následovně:

$$\mathrm{T}_{\mathrm{con}}(C) = \mathrm{T}_{\mathrm{typ}}(\theta_1, x_1), \mathrm{T}_{\mathrm{typ}}(\theta_2(x_1), x_2), \ldots, \mathrm{T}_{\mathrm{typ}}(\theta_n(x_1, \ldots, x_{n-1}), x_n)$$

Definice 17. Definujme zobrazení T z formulí jazyka Mizaru do jazyka systému HOL Light následujícími rovnostmi:

- Překlad termů a predikátů sleduje induktivní definici těchto konstrukcí:

  $$\mathrm{T}(x) = \mathtt{x}$$

  $$\mathrm{T}((t_1, \ldots, t_m)\, f\, (t_{m+1}, \ldots, t_n)) = \mathtt{I}_{\mathtt{f}}(\mathrm{T}(t_1))\ldots(\mathrm{T}(t_m))(\mathrm{T}(t_{m+1}))\ldots(\mathrm{T}(t_n))$$

  $$\mathrm{T}((t_1, \ldots, t_m)\, R\, (t_{m+1}, \ldots, t_n)) = \mathtt{I}_{\mathtt{R}}(\mathrm{T}(t_1))\ldots(\mathrm{T}(t_m))(\mathrm{T}(t_{m+1}))\ldots(\mathrm{T}(t_n))$$

[6] plyne z definice 1

- Překlad kvalifikujících formulí:

$$\mathrm{T}(t \textbf{ is } \theta) = \mathrm{T}_{\mathrm{typ}}(\theta, t)$$

$$\mathrm{T}(t \textbf{ is } \alpha) = \mathrm{T}_{\mathrm{atr}}(\alpha, \Lambda(t), t)$$

- Překlad formulí s výrokovými spojkami je přímočarý a je zde uveden spíše pro úplnost:

$$\begin{aligned}
\mathrm{T}(\textbf{contradiction}) &= \mathtt{F} \\
\mathrm{T}(\textbf{not } \phi) &= \sim \mathrm{T}(\phi) \\
\mathrm{T}(\phi \textbf{ \& } \psi) &= \mathrm{T}(\phi) \ /\backslash\ \mathrm{T}(\psi) \\
\mathrm{T}(\phi \textbf{ or } \psi) &= \mathrm{T}(\phi) \ \backslash/\ \mathrm{T}(\psi) \\
\mathrm{T}(\phi \textbf{ implies } \psi) &= \mathrm{T}(\phi) \ ==>\ \mathrm{T}(\psi) \\
\mathrm{T}(\phi \textbf{ iff } \psi) &= \mathrm{T}(\phi) \ <=>\ \mathrm{T}(\psi)
\end{aligned}$$

- Složitější je překlad formulí s kvantifikátory:

$$\begin{aligned}
\mathrm{T}(\textbf{ex } x \textbf{ being } \theta \textbf{ st } \psi) &= ?\mathtt{x}.\, \mathrm{T}_{\mathrm{typ}}(\theta, \mathtt{x}) \ /\backslash\ \mathrm{T}(\psi) \\
\mathrm{T}(\textbf{for } x \textbf{ being } \theta \textbf{ holds } \psi) &= !\mathtt{x}.\, \mathrm{T}_{\mathrm{typ}}(\theta, \mathtt{x}) \ ==>\ \mathrm{T}(\psi) \\
\mathrm{T}(\textbf{for } x \textbf{ being } \theta \textbf{ st } \phi \textbf{ holds } \psi) &= !\mathtt{x}.\, \mathrm{T}_{\mathrm{typ}}(\theta, \mathtt{x}) \ ==>\ \mathrm{T}(\phi) \ ==>\ \mathrm{T}(\psi)
\end{aligned}$$

V překladu formulí s kvantifikátory jsme mohli přímo specifikovat, jaký typ systému HOL Light mají vázané proměnné. Například takto pro obecný kvantifikátor:

$$?\mathtt{x} : \sigma.\, \mathrm{T}_{\mathrm{typ}}(\theta, x) \ /\backslash\ \mathrm{T}(\psi)$$

Pak by bylo velmi dobře vidět, jak používáme několik úrovní typů. Vázané proměnné x je v systému HOL Light přiřazen typ σ a zároveň pomocí konjunkce predikátů, která vznikne jako výsledek překladu $\mathrm{T}_{\mathrm{typ}}(\theta, \mathtt{x})$, typ Mizaru θ. Explicitní přiřazení typu σ však není třeba, neboť inferenční mechanismus typového systému systému HOL Light ho odvodí sám. Důvodem je fakt, že vázané proměnné je následně přiřazen typ Mizaru pomocí nějakého predikátu, který má za typ svých parametrů σ. Inferenční algoritmus Hindley–Milner pak velmi snadno může odvodit, že i x má typ σ.

Pokud tedy máme nějaké tvrzení ϕ Mizaru v kontextu C, přeložilo by se do systému HOL Light následovně[7]:

$$\mathrm{T}_{\mathrm{con}}(C) | - \mathrm{T}(\phi)$$

[7] Takové tvrzení se musí samozřejmě v systému HOL Light dokázat, zde nám jde pouze o syntaktickou stránku překladu.

3.5 Převod definic

V této části budou používány odvozovací funkce systému HOL Light – například funkce `DISCH_ALL` nebo `CONJUNCT1`. Dokumentaci k těmto funkcím lze nálezt v [Har09].

3.5.1 Převod definice módu

Definice nového symbolu

Mějme definici módu v Mizaru:

definition
  let x_1 **be** θ_1, x_2 **be** $\theta_2(x_1)$, ..., x_n **be** $\theta_n(x_1,\ldots,x_{n-1})$;
  mode Γ **of** $x_{\sigma(1)},\ldots,x_{\sigma(n)}$ -> $\theta(x_1,\ldots,x_n)$ **means**
  : L : $\delta(x_1,\ldots,x_n,\textbf{it})$;
  existence ...;
end;

Využijeme funkce `new_definition` představené v 3.3. Tuto funkci zavoláme s následujícím argumentem:

$$\mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} = \mathrm{T}(\theta,\mathtt{y})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]$$

Toto by mohlo svádět k tomu, uvést jako definici $\mathtt{I}_\Gamma$ pouze překlad formule δ. Ale je nutné si uvědomit, že sémantika nového typu v Mizaru je určena podmínkou δ, které nám vymezuje nový typ jako podmnožinu hodnot typu θ. Z tohoto důvodu je potřeba dodat podmínku na y, které nemůže být libovolné. Ostatně lze to vidět i z tvaru existenční věty, která je do systému HOL Light přeložena následovně[8]:

$$\mathrm{T}_{\mathrm{con}}(C)\ \ |-\ ?\mathtt{x}.\,\mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{x})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{x}/\mathtt{it}]$$

kde C je kontext výše uvedené definice módu. Říká, že daný mód je neprázdný tím, že tvrdí, že existuje objekt, který splňuje definici módu. A po přeložení do systému HOL Light je více než patrné, že součástí této definice je i požadavek na typ onoho objektu y.

Symbol $\mathtt{I}_\Gamma$ bude definován v systému HOL Light s následujícím typem:

$$\mathtt{I}_\Pi : \underbrace{\sigma \to \cdots \to \sigma}_{\text{n-krát}} \to \sigma \to \mathtt{bool}$$

Voláním funkce `new_definition` s výše uvedeným parametrem pak dostaneme následující tvrzení[9]:

$$|-\ !\mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y}.\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} <=>\ \mathrm{T}(\theta,\mathtt{y})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]$$

Z tohoto tvrzení odvodíme nadtypovou větu a definiční větu.

[8] Protože víme, jak C vypadá, mohli bychom místo $\mathrm{T}_{\mathrm{con}}(C)$ psát $\mathrm{T}_{\mathrm{typ}}(\theta_1,\mathtt{x}_1)$, ..., $\mathrm{T}_{\mathrm{typ}}(\theta_\mathtt{n},\mathtt{x}_\mathtt{n})$. Z důvodu úspory místa však budeme používat většinou první formu zápisu.

[9] Rovnost mezi výrazy typu `bool` je vypisována jako ekvivalence (<=>), i když interně jde pořád o rovnost. Budeme tuto notaci dodržovat.

Odvození nadtypové věty

Nadtypová věta má následující tvar

$$\mathrm{T}_{\mathrm{con}}(C),\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y}\ \mid\!-\ \mathrm{T}(\theta,\mathtt{y})$$

a přímočaře zachycuje sémantiku nadtypů. Neboli při kontextu C a pokud je y typu $\mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}$, je y také typu θ. Bude nám sloužit k dokazování, že nějaký typ je podtypem jiného. Tuto větu získáme následujícím odvozením:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mid\!-\ !\mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y}.\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} <\!=\!>\ \mathrm{T}(\theta,\mathtt{y})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}{\mid\!-\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} <\!=\!>\ \mathrm{T}(\theta,\mathtt{y})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}\ \text{SPEC\_ALL}}{\mid\!-\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} =\!=\!>\ \mathrm{T}(\theta,\mathtt{y})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}\ \text{fst o EQ\_IMP\_RULE}}{\mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y}\ \mid\!-\ \mathrm{T}(\theta,\mathtt{y})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}\ \text{UNDISCH}}{\mid\!-\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y}\ \mid\!-\ \mathrm{T}(\theta,\mathtt{y})}\ \text{CONJUNCT1}}{\mid\!-\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} =\!=\!>\ \mathrm{T}(\theta,\mathtt{y})}\ \text{DISCH\_ALL}}{\mathrm{T}_{\mathrm{con}}(C)\ \mid\!-\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} =\!=\!>\ \mathrm{T}(\theta,\mathtt{y})}\ \text{ADD\_CONTEXT } C}{\mathrm{T}_{\mathrm{con}}(C),\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y}\ \mid\!-\ \mathrm{T}(\theta,\mathtt{y})}\ \text{UNDISCH}$$

Odvození definiční věty

Definiční věta vznikne přeložením definiční věty pro módy v Mizaru, jak byla definována v 2.5.3. Nejdříve odvodíme dvě implikace ϕ_1 a ϕ_2:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mid\!-\ !\mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y}.\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} <\!=\!>\ \mathrm{T}(\theta,\mathtt{y})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}{\mid\!-\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} <\!=\!>\ \mathrm{T}(\theta,\mathtt{y})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}\ \text{SPEC\_ALL}}{\mid\!-\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} =\!=\!>\ \mathrm{T}(\theta,\mathtt{y})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}\ \text{fst o EQ\_IMP\_RULE}}{\mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y}\ \mid\!-\ \mathrm{T}(\theta,\mathtt{y})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}\ \text{UNDISCH}}{\mid\!-\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y}\ \mid\!-\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}\ \text{CONJUNCT2}}{(\phi_1)\quad \mid\!-\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} =\!=\!>\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}\ \text{DISCH\_ALL}$$

$$\vdots$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mid\!-\ !\mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y}.\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} <\!=\!>\ \mathrm{T}(\theta,\mathtt{y})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}{\mid\!-\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} <\!=\!>\ \mathrm{T}(\theta,\mathtt{y})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}\ \text{SPEC\_ALL}}{\mid\!-\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} <\!=\!>\ \mathrm{T}(\theta,\mathtt{y})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}\ \text{snd o EQ\_IMP\_RULE}}{\mid\!-\ \mathrm{T}(\theta,\mathtt{y})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}] =\!=\!>\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y}}\ \text{rewrite IMP\_IMP}}{\mid\!-\ \mathrm{T}(\theta,\mathtt{y}) =\!=\!>\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}] =\!=\!>\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y}}\ \text{UNDISCH}}{(\phi_2)\quad \mathrm{T}(\theta,\mathtt{y})\ \mid\!-\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}] =\!=\!>\ \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y}}$$

$$\vdots$$

Složením dokázaných implikací a několika úpravami dostaneme požadovaný tvar definiční věty:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots \quad \vdots}{\phi_1 \quad \phi_2}}{\mathrm{T}(\theta,\mathtt{y}) \mid\!- \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} <\!=\!> \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}\ \text{IMP\_ANTISYM\_RULE}}{\mid\!- \mathrm{T}(\theta,\mathtt{y}) ==\!> \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} <\!=\!> \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}\ \text{DISCH\_ALL}}{\mathrm{T}_{\mathrm{con}}(C) \mid\!- \mathrm{T}(\theta,\mathtt{y}) ==\!> \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} <\!=\!> \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}\ \text{ADD\_CONTEXT } C}{\mathrm{T}_{\mathrm{con}}(C),\ \mathrm{T}(\theta,\mathtt{y}) \mid\!- \mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} <\!=\!> \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}\ \text{UNDISCH}}{\mid\!- \mathtt{!x_1}.\mathrm{T}_{\mathrm{typ}}(\theta_1,x_1) ==\!> \ldots\ \mathtt{!x_n}.\mathrm{T}_{\mathrm{typ}}(\theta_n,x_n) ==\!>\ \mathtt{!y}.\mathrm{T}(\theta,\mathtt{y}) ==\!> (\mathtt{I}_\Gamma\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}\ \mathtt{y} <\!=\!> \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}])}\ \text{DISCH\_CONTEXT}$$

3.5.2 Převod definice predikátu

Definice nového symbolu

Mějme definici predikátu v Mizaru:

definition
  let x_1 **be** θ_1, x_2 **be** $\theta_2(x_1)$, …, x_n **be** $\theta_n(x_1,\ldots,x_{n-1})$;
  pred $x_{\sigma(1)},\ldots,x_{\sigma(k)}\Pi x_{\sigma(k+1)},\ldots,x_{\sigma(n)}$ **means**
  : L : $\delta(x_1,\ldots,x_n)$;
end;

Pomocí funkce `new_definition` s následujícím argumentem definujeme nový symbol:

$$\mathtt{I}_\Pi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})} = \mathrm{T}(\delta)$$

Symbol $\mathtt{I}_\Pi$ bude mít v systému HOL Light tento typ:

$$\mathtt{I}_\Pi : \underbrace{\sigma \to \cdots \to \sigma}_{\text{n-krát}} \to \mathtt{bool}$$

Funkce `new_definition` pak vrátí následující tvrzení:

$$\mid\!- \mathtt{!x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}.\ \mathtt{I}_\Pi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})} <\!=\!> \mathrm{T}(\delta)$$

Odvození definiční věty

Definiční věta vznikne stejně jako u definice módu přeložením příslušné definiční věty z Mizaru. Ačkoliv by se mohlo zdát, že jsme jako výsledek volání funkce `new_definition` dostali definiční větu, není tomu tak. Získané tvrzení nemá přesně její tvar. Proměnné zavedené obecnými kvantifikátory nemají přiřazeny mizarovské typy. Proto provedeme následující odvození. Nechť je C kontext výše uvedené definice predikátu.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mid\!- \ !\mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}.\ \mathtt{I}_{\Pi}\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})} <=> \mathrm{T}(\delta)}{\mid\!- \mathtt{I}_{\Pi}\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})} <=> \mathrm{T}(\delta)}\ \text{SPEC\_ALL}}{\mathrm{T}_{\mathrm{con}}(C)\ \mid\!- \mathtt{I}_{\Pi}\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})} <=> \mathrm{T}(\delta)}\ \text{ADD\_CONTEXT } C}{\mid\!- \ !\mathtt{x}_1.\mathrm{T}_{\mathrm{typ}}(\theta_1,x_1) ==> \ldots \ !\mathtt{x}_{\mathtt{n}}.\mathrm{T}_{\mathrm{typ}}(\theta_n,x_n) ==> \mathtt{I}_{\Pi}\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})} <=> \mathrm{T}(\delta)}\ \text{DISCH\_CONTEXT}$$

Pomocí tohoto odvození jsme dosáhli požadovaného tvaru definiční věty – vázané proměnné mají přiřazeny své mizarovské typy.

3.5.3 Převod definice funktoru

Definice nového symbolu

Mějme definici funktoru v Mizaru:

definition
let x_1 **be** θ_1, x_2 **be** $\theta_2(x_1)$, ..., x_n **be** $\theta_n(x_1,\ldots,x_{n-1})$;
func $(x_{\sigma(1)},\ldots,x_{\sigma(k)})\varphi(x_{\sigma(k+1)},\ldots,x_{\sigma(n)})$ -> $\theta(x_1,\ldots,x_n)$ **means**
: L : $\delta(x_1,\ldots,x_n,\mathbf{it})$;
existence ...;
uniqueness ...;
end;

K definici nového symbolu $\mathtt{I}_{\varphi}$ použijeme funkci `new_specification` popsanou v 3.3. Jako druhý argument se musí uvést existenční věta tvrdící, že existuje konstanta splňující podmínku, která má tuto konstantu definovat. Při definici funktoru je v Mizaru podobné existenční tvrzení uvedeno. To má po přeložení do systému HOL Light následující tvar:

$$\mathrm{T}_{\mathrm{con}}(C)\ \mid\!- \ ?\mathtt{x}.\mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{x})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{x}/\mathtt{it}]$$

Kde C je kontext výše uvedené definice funktoru. Toto tvrzení převedeme do tvaru, který bude odpovídat existenčnímu tvrzení vyžadovanému funkcí `new_specification`:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathrm{T}_{\mathrm{con}}(C)\ \mid\!- \ ?\mathtt{x}.\mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{x})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{x}/\mathtt{it}]}{\mid\!- \ \mathrm{T}_{\mathrm{typ}}(\theta_1,\mathtt{x}_1) ==> \ldots ==> \mathrm{T}_{\mathrm{typ}}(\theta_{\mathtt{n}},\mathtt{x}_{\mathtt{n}}) ==>}\ \text{DISCH\_ALL}}{?\mathtt{x}.\mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{x})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{x}/\mathtt{it}]}{\mid\!- \ ?\mathtt{x}.\mathrm{T}_{\mathrm{typ}}(\theta_1,\mathtt{x}_1) ==> \ldots ==> \mathrm{T}_{\mathrm{typ}}(\theta_{\mathtt{n}},\mathtt{x}_{\mathtt{n}}) ==>}\ \text{prenex}}{\mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{x})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{x}/\mathtt{it}]}{\mid\!- \ !\mathtt{x}_{\sigma(1)} \ldots !\mathtt{x}_{\sigma(\mathtt{n})}\ ?\mathtt{x}.\mathrm{T}_{\mathrm{typ}}(\theta_1,\mathtt{x}_1) ==> \ldots ==>}\ \text{generalizace}}{\mathrm{T}_{\mathrm{typ}}(\theta_{\mathtt{n}},\mathtt{x}_{\mathtt{n}}) ==> \mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{x})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{x}/\mathtt{it}]}{\mid\!- \ ?\mathtt{f}.\ !\mathtt{x}_{\sigma(1)} \ldots !\mathtt{x}_{\sigma(\mathtt{n})}.\mathrm{T}_{\mathrm{typ}}(\theta_1,\mathtt{x}_1) ==> \ldots ==>}\ \text{skolemizace}}$$
$$\mathrm{T}_{\mathrm{typ}}(\theta_{\mathtt{n}},\mathtt{x}_{\mathtt{n}}) ==> \mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{f}\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})})\ /\backslash\ \mathrm{T}(\delta)[\mathtt{f}\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}/\mathtt{it}]$$

Poslední tvrzení odpovídá tvaru, který vyžaduje funkce `new_specification`; označme si toto tvrzení jako ϕ. Nyní lze zavolat funkci `new_specification` následujícím způsobem:

$$\texttt{new\_specification ["I}_\varphi\texttt{"]}\ \phi$$

Nový symbol $\mathtt{I}_\varphi$ bude mít v systému HOL Light tento typ:

$$\mathtt{I}_\varphi : \underbrace{\sigma \to \cdots \to \sigma}_{\text{n-krát}} \to \sigma$$

a volání vrátí následující větu:

$$\begin{aligned} &| - \ !\mathtt{x}_{\sigma(1)} \ldots !\mathtt{x}_{\sigma(\mathtt{n})}.\, \mathrm{T}_{\mathrm{typ}}(\theta_1, \mathtt{x}_1) ==> \ldots ==> \mathrm{T}_{\mathrm{typ}}(\theta_\mathtt{n}, \mathtt{x}_\mathtt{n}) \\ &==> \mathrm{T}_{\mathrm{typ}}(\theta, \mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}) \ /\backslash\ \mathrm{T}(\delta)[\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}/\mathtt{it}] \end{aligned}$$

Z této věty odvodíme větu návratového typu a definiční větu.

Odvození věty návratového typu

Věta návratového typu má následující tvar

$$\mathrm{T}_{\mathrm{con}}(C)\ |-\ \mathrm{T}(\theta, \mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})})$$

a dokazuje, jaký je návratový typ funkce φ. Bude sloužit k výpočtu typu termu. Větu návratového typu získáme následujícím odvozením:

$$\dfrac{\dfrac{\dfrac{| - \ !\mathtt{x}_{\sigma(1)} \ldots !\mathtt{x}_{\sigma(\mathtt{n})}.\, \mathrm{T}_{\mathrm{typ}}(\theta_1, \mathtt{x}_1) ==> \ldots ==> \mathrm{T}_{\mathrm{typ}}(\theta_\mathtt{n}, \mathtt{x}_\mathtt{n}) \\ ==> \mathrm{T}_{\mathrm{typ}}(\theta, \mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}) \ /\backslash\ \mathrm{T}(\delta)[\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}/\mathtt{it}]}{| - \ \mathrm{T}_{\mathrm{typ}}(\theta_1, \mathtt{x}_1) ==> \ldots ==> \mathrm{T}_{\mathrm{typ}}(\theta_\mathtt{n}, \mathtt{x}_\mathtt{n}) ==> \\ \mathrm{T}_{\mathrm{typ}}(\theta, \mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}) \ /\backslash\ \mathrm{T}(\delta)[\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}/\mathtt{it}]}\ \text{SPEC\_ALL}}{\mathrm{T}_{\mathrm{con}}(C)}\ \text{UNDISCH\_ALL}}{| - \ \mathrm{T}_{\mathrm{typ}}(\theta, \mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}) \ /\backslash\ \mathrm{T}(\delta)[\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}/\mathtt{it}]}\ \text{CONJUCT1}$$

Odvození definiční věty

Definiční věta vznikne jako v předešlých případech přeložením definiční věty pro funktory v Mizaru. Protože je definiční věta ekvivalence, dokážeme nejdříve dvě implikace, ϕ_1 a ϕ_2:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\begin{array}{c}\mid\!- \ !\mathtt{x}_{\sigma(1)} \ldots !\mathtt{x}_{\sigma(\mathtt{n})}.\,\mathrm{T}_{\mathrm{typ}}(\theta_1,\mathtt{x}_1) ==> \ldots ==> \mathrm{T}_{\mathrm{typ}}(\theta_{\mathtt{n}},\mathtt{x}_{\mathtt{n}}) \\ ==> \mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}) \ /\backslash\ \mathrm{T}(\delta)[\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}/\mathtt{it}]\end{array}}\ \text{SPEC\_ALL}}{\begin{array}{c}\mid\!- \ \mathrm{T}_{\mathrm{typ}}(\theta_1,\mathtt{x}_1) ==> \ldots ==> \mathrm{T}_{\mathrm{typ}}(\theta_{\mathtt{n}},\mathtt{x}_{\mathtt{n}}) ==> \\ \mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}) \ /\backslash\ \mathrm{T}(\delta)[\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}/\mathtt{it}]\end{array}}\ \text{UNDISCH\_ALL}}{\mathrm{T}_{\mathrm{con}}(C)}\ \text{CONJUCT2}}{\begin{array}{c}\mid\!- \ \mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}) \ /\backslash\ \mathrm{T}(\delta)[\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}/\mathtt{it}] \\ \mathrm{T}_{\mathrm{con}}(C) \ \mid\!- \ \mathrm{T}(\delta)[\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}/\mathtt{it}]\end{array}}\ [\mathtt{y}/\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}]}{\begin{array}{c}\mathrm{T}_{\mathrm{con}}(C),\ \mathtt{y} = \mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})} \ \mid\!- \ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]\end{array}}\ \text{DISCH}}{\begin{array}{cc}(\phi_1) & \mathrm{T}_{\mathrm{con}}(C) \ \mid\!- \ \mathtt{y} = \mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})} ==> \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]\end{array}}$$

$$\vdots$$

Přeložením tvrzení v **uniqueness** sekci v Mizaru (viz 2.5.5) do systému HOL Light dostaneme následující tvrzení:

$$\begin{array}{c}\mathrm{T}_{\mathrm{con}}(C) \ \mid\!- \ !\mathtt{y}.\,\mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{y}) ==> !\mathtt{z}.\,\mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{z}) ==> \\ (\mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}] \ /\backslash\ \mathrm{T}(\delta)[\mathtt{z}/\mathtt{it}]) ==> \mathtt{y} = \mathtt{z}\end{array}$$

Z tohoto tvrzení odvodíme opačnou implikaci ϕ_2:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\begin{array}{c}\mathrm{T}_{\mathrm{con}}(\mathrm{C}) \ \mid\!- \ !\mathtt{y}.\,\mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{y}) ==> !\mathtt{z}.\,\mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{z}) \\ ==> (\mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}] \ /\backslash\ \mathrm{T}(\delta)[\mathtt{z}/\mathtt{it}]) ==> \mathtt{y} = \mathtt{z}\end{array}}\ \text{UNDISCH\_VARS}}{\begin{array}{c}\mid\!- (\mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}] \ /\backslash\ \mathrm{T}(\delta)[\mathtt{z}/\mathtt{it}]) ==> \mathtt{y} = \mathtt{z} \\ \mathrm{T}_{\mathrm{con}}(\mathrm{C}),\ \mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{y}),\ \mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{z})\end{array}}\ \text{rewrite IMP\_IMP}}{\begin{array}{c}\mid\!- \ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}] ==> \mathrm{T}(\delta)[\mathtt{z}/\mathtt{it}] ==> \mathtt{y} = \mathtt{z} \\ \mathrm{T}_{\mathrm{con}}(\mathrm{C}),\ \mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{y}),\ \mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{z}),\end{array}}\ \text{UNDISCH}}{\begin{array}{c}\mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}] \ \mid\!- \ \mathrm{T}(\delta)[\mathtt{z}/\mathtt{it}] ==> \mathtt{y} = \mathtt{z} \\ \mathrm{T}_{\mathrm{con}}(\mathrm{C}),\ \mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}),\ \mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{z}),\end{array}}\ [\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}/\mathtt{y}]}{\begin{array}{c}\mathrm{T}(\delta)[\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}/\mathtt{it}] \ \mid\!- \ \mathrm{T}(\delta)[\mathtt{z}/\mathtt{it}] ==> \mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})} = \mathtt{z} \\ \mathrm{T}_{\mathrm{con}}(\mathrm{C}),\ \mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}),\ \mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{y}),\ \mathrm{T}(\delta)[\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}/\mathtt{it}]\end{array}}\ [\mathrm{y/z}]$$

$$\mid\!- \ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}] ==> \mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})} = \mathtt{y}$$

Poslední tvrzení již má skoro tvar druhé implikace, ale obsahuje dva předpoklady, které nám porušují obecný tvar kontextu:

$$\begin{array}{c}\mathrm{T}_{\mathrm{typ}}(\theta,\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}) \\ \mathrm{T}(\delta)[\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)} \ldots \mathtt{x}_{\sigma(\mathtt{n})}/\mathtt{it}]\end{array}$$

První říká, jaký má funktor φ návratový typ, a druhý říká, že splňuje podmínku δ. Oba dva fakty nám ale dokazuje tvrzení (označme si jej ψ), které jsme dostali

jako výsledek při volání funkce `new_specification`[10]. Proto můžeme použít pravidlo řezu a těchto dvou předpokladů se zbavit.

$$\dfrac{\dfrac{\dfrac{\mathrm{T_{con}(C),\ T_{typ}}(\theta,\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}),\ \mathrm{T_{typ}}(\theta,\mathtt{y}),\ \mathrm{T}(\delta)[\mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}/\mathtt{it}] \mid\!-\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}] ==\!> \mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})} = \mathtt{y}}{\mathrm{T_{con}(C),\ T_{typ}}(\theta,\mathtt{y})\ \mid\!-\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}] ==\!> \mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})} = \mathtt{y}}\ \text{cut}\ \psi}{(\phi_2)\quad \mathrm{T_{con}(C),\ T_{typ}}(\theta,\mathtt{y})\ \mid\!-\ \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}] ==\!> \mathtt{y} = \mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})}}\ \text{sym} =$$

$$\vdots$$

Složením dokázaných implikací ϕ_1 a ϕ_2 dostaneme požadovaný tvar definiční věty:

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots & \vdots\\ \phi_1 & \phi_2\end{matrix}}{\mathrm{T_{con}(C),\ T_{typ}}(\theta,\mathtt{y})}\ \text{IMP\_ANTISYM\_RULE}}{\mid\!-\ \mathtt{y} = \mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})} <\!=\!> \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]}\ \text{DISCH\_CONTEXT}}{\begin{matrix}\mid\!-\ !\mathtt{x}_1.\mathrm{T_{typ}}(\theta_1,x_1) ==\!> \ldots\ !\mathtt{x}_\mathtt{n}.\mathrm{T_{typ}}(\theta_n,x_n) ==\!>\\ !\mathtt{y}.\mathrm{T}(\theta,\mathtt{y}) ==\!> \mathtt{y} = \mathtt{I}_\varphi\ \mathtt{x}_{\sigma(1)}\ \ldots\ \mathtt{x}_{\sigma(\mathtt{n})} <\!=\!> \mathrm{T}(\delta)[\mathtt{y}/\mathtt{it}]\end{matrix}}$$

[10] Museli bychom ještě provést několik úprav formule ψ, abychom mohli použít přímo pravidlo řezu implementované v systému HOL Light, pro zkrácení si ale dovolíme použít přímo ψ.

Kapitola 4

Implementace

V této kapitole je prezentována implementace převodu typového systému (popsaného v předešlé kapitole) v systému HOL Light. Současná implementace plně nepokrývá celý popsaný převod. K úplné implementaci je potřeba dokončit mechanismy spojené s atributy. Autor se nicméně domnívá, že zde diskutovaná implementace je dostatečně rozsáhlá k zachycení technických otázek spojených s převodem a že chybějící část bude již jen inkrementálně přidána k realizované implementaci.

4.1 Identifikátory

Jak bylo diskutováno v 3.4, předpokládali jsme, že pro každý identifikátor v Mizaru představující mód, atribut, predikát nebo funktor, lze nalézt jeho jednoznačnou reprezentaci v systému HOL Light. Jmenné prostory identifikátorů v Mizaru pro různé syntaktické kategorie (módy, atributy, . . .) jsou oddělené. Protože však tyto kategorie reprezentujeme v systému HOL Light funkcemi, máme k dispozici v systému HOL Light jen jeden jmenný prostor. Proto budeme muset tento prostor pro jednotlivé kategorie nějak uměle rozdělit. S výhodou se dá využít způsob, jakým jsou tyto kategorie označovány v Mizaru. Každé kategorii odpovídá jedno velké písmeno, jak je shrnuto v následující tabulce [miz]:

M	mód
V	atribut
R	predikát
O	funktor

Ačkoliv jsou tato písmena v Mizaru velká, my budeme používat jejich malé varianty. To bude první složka přeloženého jména. Druhou složkou bude samotné jméno, protože identifikátory v Mizaru můžou obsahovat takřka libovolný znak znakové sady, vynecháme při překladu takové znaky, které neodpovídají lexikálním konvencím systému HOL Light. Třetí složkou bude arita symbolu, velmi jednoduše ji můžeme definovat jako počet deklarovaných proměnných v kon-

textu definice příslušného symbolu[1]. Ani tato doposud popsaná konvence nám nemusí zajistit jednoznačnost.

Symboly stejné kategorie, stejného jména a arity by se nám mapovaly na stejný identifikátor v systému HOL Light. Takové případy budou nastávat, protože systém Mizar nezakazuje definovat další symbol stejné kategorie, jména a arity. Zjednoznačnění dosáhneme velmi jednoduše. Symboly, které by se zobrazovaly na stejnou třídu určenou kategorií, jménem a aritou, oindexujeme číslem odpovídajícím pořadí výskytu definice tohoto symbolu v rámci nejednoznačných symbolů. Pokud tedy definujeme nějaký symbol, který je i-tým definovaným symbolem stejné kategorie, jména a arity, bude mít v poslední složce číslo $i-1$.

Složky budeme oddělovat pomocí znaku _. Přeložený identifikátor bude mít tedy obecně následující tvar:

`kategorie_jméno_arita_index`

Ukažme si překlad na dvou příkladech:

Element **of** X	`--→`	`m_Element_1_0`
empty	`--→`	`v_empty_0_0`

Takto přeloženému identifikátoru budeme říkat *interní identifikátor* nebo též *interní jméno*. Skutečným jménem budeme chápat jméno, pod kterým daný mód (funktor, predikát) vystupuje v Mizaru, což je z podstaty věci jméno nejednoznačné.

4.2 Datové struktury

4.2.1 Datová struktura pro módy

Datová struktura pro módy je variant – obsahuje tři varianty formátu módu:

- `Set` – speciální varianta módu. Tato varianta byla vyčleněna především proto, že mód `set` jako jediný neobsahuje nadtyp. Žádné další informace nejsou uchovávány.
- `Mizmode` – mód definovaný pomocí **means**. Jedná se o record, který obsahuje další položky.
- `Mizabbr` – mód definovaný pomocí **is**. Jedná se o record, který obsahuje další položky.

Nyní podrobněji rozebereme, jaké položky se uchovávají v případě definice módu pomocí **means**, tedy jaká je struktura recordu `Mizmode`:

[1] U atributů nebudeme do kontextu započítávat proměnnou, která v definici zastupuje objekt, který by měl mít právě definovaný atribut.

- interní jméno módu, odpovídá názvu predikátu, který vznikne přeložením z Mizaru tak, jak bylo definováno v 4.1.
- jméno módu, které bylo zadáno v Mizaru. Bude sloužit pro pretty-printing.
- index módu. Pokud je provedena redefinice nějakého módu, je tento redefinovaný mód přidán jako nový plnohodnotný mód. Aby ale bylo poznat, který mód tento nový mód redefinuje, sdílejí všechny redefinice stejný index jako má původní mód, který redefinují. Tedy pokud definujeme nový mód, hodnota idexu je různá od všech indexů ostatních módů, pokud ale redefinujeme nějaký mód, rovná se indexu módu, který redefinujeme.
- kontext, ve kterém je nový mód definován.
- seznam parametrů módu. Jedná se o permutaci proměnných uvedených v kontextu.
- nadtyp a nadtypová věta – viz 3.5.1.
- existenční tvrzení – tvrdí, že definovaný mód je neprázdný, více viz 2.5.3.
- definiční věta – charakterizuje nově definovaný mód, více viz 2.5.3.

Pro úplnost uveďme, jaké položky obsahuje `Mizabbr`:

- skutečné jméno módu pro pretty-printing.
- kontext, ve kterém byla definice módu pomocí **is** definována.
- seznam parametrů.
- typ, který nový mód zkracuje.

4.2.2 Datové struktury pro predikáty a pro funktory

Datové struktury pro predikáty a pro funktory jsou téměř stejné, proto budou popsány společně. Jedná se o record, který obsahuje následující položky:

- interní jméno, stejně jako u módu jde o jméno přeložené z Mizaru mechanismem popsaným v 4.1.
- jméno určené pro pretty-printing.
- kontext, ve kterém byl predikát, respektive funktor definován.
- seznam parametrů.
- návratový typ a věta návratového typu – viz 3.5.3. Tuto položku obsahuje jen record pro funktory. U predikátů jí není ze zřejmých důvodů třeba.

4.2.3 Tabulky

Datové struktury pro módy, predikáty a funktory jsou uloženy ve speciálních tabulkách, aby se k těmto datům dalo rychle přistupovat. Pro každou syntaktickou kategorii existuje samostatná tabulka. Tato tabulka se zvnějšku jeví jako jedna tabulka, vnitřně je však implementována pomocí dvou tabulek[2]:

- První tabulka je hašovací tabulka, která jako typ klíče používá dvojici jméno módu a jeho aritu. Datovou částí je pak seznam všech módů, které mají stejný název a aritu, seřazený podle času, kdy byl daný mód definován. Na začátku seznamu jsou nejmladší módy. Tato tabulka bude sloužit především pro parser, který na základě skutečného jména a arity módu dostane seznam všech módů, které těmto údajům odpovídají. Z tohoto seznamu na základě typů parametrů vybere nejvhodnější mód[3]. Po tomto výběru již se lze odkazovat na vybraný mód pomocí interního jména, které je již jednoznačné.

- Druhá tabulka je také hašovací tabulka, která má jako typ klíče interní jméno módu. Datovou částí je samotný mód odpovídající internímu jménu uvedenému v klíči. Tato tabulka slouží k vyhledání informací o módu, pokud již známe jeho interní jméno.

Další tabulkou přítomnou v implementaci je tabulka pojmenovaných vět. Je to jednoduchá hašovací tabulka, která jako klíč má název věty a jako data větu, která tomuto klíči odpovídá. Do této tabulky jsou ukládány všechny definiční věty pod názvem : L :, který byl uveden při definici.

4.3 Reprezentace vět

Jak bylo popsáno v 3.1, každá věta v systému HOL Light má typ `thm`. I naše věty obsahující přeložené typy Mizaru budou reprezentovány pomocí typu `thm`. Nebude se tak ovšem dít přímo, ale typ `thm` zabalíme do námi definovaného typu `miz_thm`:

```
type miz_thm = Mizthm of thm; ;
```

Důvodem je větší typová bezpečnost, aby pomocné funkce, které budou mít smysl jen pro věty obsahující typy Mizaru, nešlo aplikovat i na běžné věty systému HOL Light. A samozřejmě i opačně, abychom měli kontrolu nad tím, které funkce běžně používané pro věty systému HOL Light lze aplikovat na věty s typy Mizaru.

[2]Popis učiníme pro případ, kdy do tabulky ukládáme módy. Nicméně popis by byl úplně stejný i pro predikáty a funktory, tabulka je navržena polymorfně a je parametrizována typem dat, typ klíčů zůstává pro všechny tři případy typů dat stejný.

[3]V případě nejednoznačností se vybere nejmladší z vhodných.

Manipulace s větami typu `miz_thm` je omezena buď na přímo napsané funkce pro tento typ, nebo lze použít funkce pro typ `thm` díky pomocným interním funkcím `m` nebo `m2`. Jejich typy jsou následující a snad i napoví jejich použití:

$$\texttt{m} : (\texttt{thm} \to \texttt{thm}) \to \texttt{miz\_thm} \to \texttt{miz\_thm}$$
$$\texttt{m2} : (\texttt{thm} \to \texttt{thm} \to \texttt{thm}) \to \texttt{miz\_thm} \to \texttt{miz\_thm} \to \texttt{miz\_thm}$$

Tyto funkce rozbalí argument typu `miz_thm` a na vnitřní data typu `thm` aplikují běžnou funkci pracující s větami typu `thm`. Tímto způsobem sice lze aplikovat téměř libovolnou funkci pro práci s původními větami na věty s typy Mizaru, ale musí se tak dít explicitně pomocí funkcí `m` nebo `m2`. Tím je riziko nechtěné manipulace s nesprávným druhem vět minimalizováno.

Jistou pozornost je také potřeba věnovat reprezentaci kontextu. Jak bylo definováno v definici 16, kontext do systému HOL Light překládáme jako seznam předpokladů. Jak už bylo zmíněno v samotné definici kontextu v Mizaru v části 2.5.1, záleží na pořadí definovaných proměnných. S tím může být trochu potíž, neboť jak se píše v [Har09], není v reprezentaci věty typu `thm` obecně zaručeno pořadí předpokladů. Proto odvozovací pravidlo `DISCH_ALL` může převést předpoklady věty na předpoklady implikace v libovolném pořadí.

Proto bylo definováno odvozovací pravidlo `DISCH_CONTEXT`[4], které tento nedostatek zohledňuje a převádí předpoklady věty na předpoklady implikace ve správném pořadí.

4.4 Implementace odvození typových vět

4.4.1 Typ termu

Implementace odvození typu termu používá věty návratového typu dokázané pro každý nově definovaný funktor – jejich odvození viz 3.5.3. Tyto věty jsou jen vhodně instanciovány podle skutečných argumentů funktoru. Pokud je ale term, jehož typ chceme odvodit, pouze proměnná, je použit pouze její typ z kontextu.

Implementace odvození věty typu termu je obsažena ve funkci s názvem `get_term_type_mth_c`. Její typ je následující:

$$\texttt{var\_decl list} \to \texttt{term} \to \texttt{miz\_thm}$$

Typ `var_decl list` je abstraktní typ představující kontext, typ `term` reprezentuje term a `miz_thm` je typ definovaný v 4.3, který reprezentuje naše věty.

Jestliže funkci `get_term_type_mth_c` zavoláme s následujícími parametry

```
get_term_type_mth_c c t,
```

[4] Pravidlo ještě navíc oproti `DISCH_ALL` kvantifikuje definované proměnné v kontextu obecným kvantifikátorem.

dostaneme jako výsledek větu typu termu, která nám bude říkat, jaký typ má term `t` v kontextu `c`. Její tvar[5] bude následující:

```
c |- T t
```

kde `T` je nalezený typ termu `t`.

4.4.2 Relace podtypu

Implementace relace podtypu používá nadtypové věty dokázané pro každý nově definovaný mód – jejich odvození viz 3.5.1. Při procházení typové hierarchie jsou tyto nadtypové věty transitivně skládány, až je odvozena podtypová věta tvrdící, že nějaký typ je podtypem jiného.

Při implementaci byla použita následující optimalizace: při procházení typové hierarchie za účelem odvození podtypové věty tvrdící, že typ τ_1 je podtypem τ_2, se uplatňuje lazy strategie. To znamená, že postup odvození podtypové věty, který je během procházení postupně vytvářen, je vyhodnocen až v okamžiku, kdy je skutečně zjištěno, že typ τ_1 je podtypem τ_2.

Není tedy zbytečně dokazována podtypová věta v případech, kdy τ_1 není podtypem τ_2. Zároveň není použit přístup, ve kterém by se v pozitivním případě procházela typová hierarchie dvakrát. Poprvé k zjištění samotného faktu, že $\tau_1 \preceq \tau_2$, a podruhé k odvození podtypové věty. Jedinou nevýhodou použitého přístupu je, že v případě dlouhé cesty mezi typy τ_1 a τ_2 může zaznamenání odvození podtypové věty vést k větší paměťové náročnosti. Požadavek na větší rychlost má však vyšší prioritu než paměťová náročnost.

Implementace odvození podtypové věty je obsažena v celé řadě pomocných funkcí, které jsou zastřešeny funkcí `get_type_widening_mth_c`. Její typ je následující:

$$\texttt{var\_decl list} \rightarrow \texttt{miz\_type} \rightarrow \texttt{miz\_type} \rightarrow \texttt{miz\_thm}$$

Typ `var_decl list` je abstraktní typ představující kontext, `miz_type` je abstraktní typ zachycující přeložený typ z Mizaru a `miz_thm` reprezentuje naše věty.

Pokud tedy funkci `get_type_widening_mth_c` zavoláme následujícím způsobem

```
get_type_widening_mth_c c t1 t2,
```

funkce uspěje, pokud je `t1` podtypem `t2` v kontextu `c`. V opačném případě je vyvolána výjimka `Failure "string"`, kde místo `"string"` bude podrobnější popis důvodu neúspěchu.

V případě úspěchu bude vrácena věta následujícího tvaru:

```
c, t1 x |- t2 x
```

[5]Formát tohoto tvaru je poměrně schematický a příliš neodpovídá předešlým formálním popisům, přesto podává poměrně přesný popis situace.

4.5 Demonstrace

V této části budou demonstrovány hlavní výsledky současné implementace na malém příkladu.

4.5.1 Definice

Úplný kód této demonstrace lze nalézt na přiloženém CD. Náš malý příklad vychází z těchto definic v Mizaru:[6]

definition
  let x_1 **be** set, x_2 **be** set;
  func F1(x_1, x_2) -> set **means**
  : F1def : **it** $= x_1$;
  existence ...;
  uniqueness ...;
end;
definition
  let x **be** set, y **be** set, z **be** set;
  mode M1 **of** x, y, z -> set **means**
  : M1def : **it** $= x$;
  existence ...;
end;
definition
  let x_1 **be** set, x_2 **be** M1 **of** x_1, x_1, x_1;
  func F2(x_1, x_2) -> M1 **of** x_1, x_2, x_2 **means**
  : F2def : **it** $= x_1$;
  existence ...;
  uniqueness ...;
end;
definition
  let x **be** set, y **be** set;
  mode M2 **of** x, y -> M1 **of** x, y, y **means**
  : M2def : **it** $= x$;
  existence ...;
end;
definition
  let x **be** set, y **be** set, z **be** M2 **of** x, y;
  mode M3 **of** x, y, z -> M2 **of** F1$(x, y), z$ **means**
  : M3def : **it** $= x$;
  existence ...;

[6] Vlastní definiční formule za klíčovým slovem **means** nemají žádný hlubší význam, v našem příkladě nám půjde o demonstraci, která na těchto formulích nezávisí.

end;
definition
  let x_1 **be** set, x_2 **be** set;
  pred x_1 P1 x_2 **means**
  : P1def : $x_1 = x_2$;
end;

Tyto definice jsou do systému zaneseny pomocí funkcí `add_miz_mode`, `add_miz_predicate` a `add_miz_functor`. Tyto funkce mají nízkoúrovňový charakter a v další práci budou obaleny parserem. Po přidání nových definic si můžeme nechat vypsat definiční věty, které byly do systému zaneseny, pomocí tohoto volání:

```
print_defs_table miz_thms;;
```

Dostaneme následující výstup:[7]

$$\texttt{M1def}: \ |- \ !\texttt{x}.\ \texttt{m\_set\_0\_0\ x} ==> (!\texttt{y}.\ \texttt{m\_set\_0\_0\ y} ==> (!\texttt{z}.\ \texttt{m\_set\_0\_0\ z} ==> (!\texttt{y}'.\ \texttt{m\_set\_0\_0\ y}' ==> (\texttt{m\_M1\_3\_0\ x\ y\ z\ y}' <=> \texttt{y}' = \texttt{x}))))$$

$$\texttt{F2def}: \ |- \ !\texttt{x1}.\ \texttt{m\_set\_0\_0\ x1} ==> (!\texttt{x2}.\ \texttt{m\_M1\_3\_0\ x1\ x1\ x1\ x2} ==> (!\texttt{y}.\ \texttt{m\_M1\_3\_0\ x1\ x2\ x2\ y} ==> (\texttt{o\_F2\_2\_0\ x1\ x2} = \texttt{y} <=> \texttt{y} = \texttt{x1})))$$

$$\texttt{M3def}: \ |- \ !\texttt{x}.\ \texttt{m\_set\_0\_0\ x} ==> (!\texttt{y}.\ \texttt{m\_set\_0\_0\ y} ==> (!\texttt{z}.\ \texttt{m\_M2\_2\_0\ x\ y\ z} ==> (!\texttt{y}'.\ \texttt{m\_M2\_2\_0}\ (\texttt{o\_F1\_2\_0\ x\ y})\ \texttt{z\ y}' ==> (\texttt{m\_M3\_3\_0\ x\ y\ z\ y}' <=> \texttt{y}' = \texttt{x}))))$$

$$\texttt{F1def}: \ |- \ !\texttt{x1}.\ \texttt{m\_set\_0\_0\ x1} ==> (!\texttt{x2}.\ \texttt{m\_set\_0\_0\ x2} ==> (!\texttt{y}.\ \texttt{m\_set\_0\_0\ y} ==> (\texttt{o\_F1\_2\_0\ x1\ x2} = \texttt{y} <=> \texttt{y} = \texttt{x1})))$$

$$\texttt{M2def}: \ |- \ !\texttt{x}.\ \texttt{m\_set\_0\_0\ x} ==> (!\texttt{y}.\ \texttt{m\_set\_0\_0\ y} ==> (!\texttt{y}'.\ \texttt{m\_M1\_3\_0\ x\ y\ y\ y}' ==> (\texttt{m\_M2\_2\_0\ x\ y\ y}' <=> \texttt{y}' = \texttt{x}))))$$

$$\texttt{P1def}: \ |- \ !\texttt{x1}.\ \texttt{m\_set\_0\_0\ x1} ==> (!\texttt{x2}.\ \texttt{m\_set\_0\_0\ x2} ==> (\texttt{r\_P1\_2\_0\ x1\ x2} <=> \texttt{x1} = \texttt{x2}))$$

Hlavním výsledkem implementace je možnost odvozování vět návratového typu a podtypových vět.

[7] Definice jsou vypsány v pořadí, v jakém jsou uloženy v hašovací tabulce. Toto pořadí nemusí odpovídat pořadí, v jakém byly definovány.

4.5.2 Demonstrace odvození věty typu termu

Věta typu termu nám dokazuje, jakého je term typu. Zachycuje tedy výpočet typu termu popsaný v 2.4.6. Příkladem věty typu termu může být věta, která nám bude dokazovat typ termu, který je tvořen funktorem F2, do nějž jsme za první argument dosadili funktor F1. V systému Mizar by tato věta měla následující znění:

theorem
  let p **be** set, q **be** set, r **be** M1 **of** F1(p,q), F1(p,q), F1(p,q);
  F2(F1(p,q), r) **is** M1 **of** F1(p,q), r, r;
end;

K odvození věty typu termu použijeme funkci `get_term_type_mth_c` popsanou v 4.4.1. Budeme ji volat s následujícími argumenty:

```
get_term_type_mth_c
  [ Vardecl (v"p",mk_type_from_mode Set (v"p"));
    Vardecl (v"q",mk_type_from_mode Set (v"q"));
    Vardecl (v"r",mk_type_from_mode2
      (find_mode_by_name "m_M1_3_0")
      [`o_F1_2_0 p q`;`o_F1_2_0 p q`;`o_F1_2_0 p q`]
      (v"r"))]
  `o_F2_2_0 (o_F1_2_0 p q) r`
;;
```

Jako výsledek funkce vydá očekávanou větu typu termu:

```
m_set_0_0 p, m_set_0_0 q, m_M1_3_0(o_F1_2_0 p q) (o_F1_2_0 p q) ...
(o_F1_2_0 p q) r |- m_M1_3_0 (o_F1_2_0 p q) r r (o_F2_2_0 (o_F1_2_0 p q) r)
```

4.5.3 Demonstrace odvození podtypové věty

Podtypová věta tvrdí, že nějaký typ je podtypem jiného. Nejde o nic jiného než zachycení relace podtypu. Vrátíme-li se k našemu příkladu, zjistíme, že obsahuje hierarchie módů M1, M2 a M3. Příkladem tvrzení, které bychom chtěli o této hierarchii dokázat, může být tvrzení, že typ vzniklý z módu M3 je podtypem jistého typu vzniklého z módu M1.

V systému Mizar bychom příklad tohoto tvrzení vyjádřili takto:

theorem
  let p **be** set, q **be** set, r **be** M2 **of** p, q, x **be** M3 **of** p, q, r;
  x **is** M1 **of** F1(p,q), r, r;
end;

Tedy pokud je x typu M3 **of** p, q, r, pak je také typu M1 **of** F1 (p,q), r, r. Instanciace parametrů módu M1 nemůže být libovolná, nýbrž musí odpovídat definicím nadtypů u módů M3 a M2. Tak se do výsledného typu dostal například funktor F1, který je uveden u definice nadtypu módu M3.

K odvození podtypové věty použijme funkci `get_type_widening_mth_c` popsanou v 4.4.2. Zavoláme ji s následujícími argumenty:

```
get_type_widening_mth_c
  [ Vardecl (v"p",mk_type_from_mode Set (v"p"));
    Vardecl (v"q",mk_type_from_mode Set (v"q"));
    Vardecl (v"r",mk_type_from_mode2
      (find_mode_by_name "m_M2_2_0")
      [v"p";v"q"] (v"r"))
  ]
  (mk_type_from_mode2 (find_mode_by_name "m_M3_3_0" )
      [v"p";v"q";v"r"] (v"x"))
  (mk_type_from_mode2 (find_mode_by_name "m_M1_3_0" )
      [o_F1_2_0 p q;v"r";v"r"] (v"x"))
;;
```

Jako výsledek dostaneme odvozenou kýženou podtypovou větu:

$$\texttt{m\_set\_0\_0 p}, \texttt{m\_set\_0\_0 q}, \texttt{m\_M2\_2\_0 p q r}, \texttt{m\_M3\_3\_0 p q r x} \mid\!- \texttt{m\_M1\_3\_0 (o\_F1\_2\_0 p q) r r x}$$

4.6 Integrace

Doposud popsaná implementace umožňuje odvozovat věty o typovém systému Mizaru přeloženém do systému HOL Light. K tomu, abychom naplnili motivace uvedené v úvodu této práce, je potřeba naimplementovaný typový systém integrovat do mechanismů, které by umožňovaly plné dokazování vět Mizaru v systému HOL Light. Taková integrace již přesahuje rámec této práce a bude předmětem dalšího výzkumu. Přesto zde bude v krátkosti nastíněna její možná realizace.

Systém HOL Light byl vybrán pro překlad typového systému mimo jiné také proto, že ze všech systémů pro formalizaci matematiky obsahuje nejvíce doimplementovaných nástrojů, které umožňují překlad Mizaru do jiného systému (tedy v tomto případě do systému HOL Light). Jedná se o dvě oblasti, které je potřeba uvážit.

4.6.1 Důkazový jazyk

Formát zápisu důkazu v Mizaru lze charakterizovat jako variantu přirozené dedukce. Pokusy o implementaci jazyka důkazů systému Mizar byly doposud

učiněny dva. Starší z nich vytvořil John Harrison [Har96]. V řešení Johna Harrisona byl vytvořen vlastní parser jazyka důkazů Mizaru, který tento jazyk překládá do funkcí systému HOL Light manipulujících s důkazy. Mladší pokus implementoval Freek Wiedijk [Wie01]. V jeho přístupu není použit žádný speciální parser, nýbrž je přímo využito prostředí systému HOL Light.

Ať už bude vybrán kterýkoli z těchto pokusů, integrace typového systému se bude muset zaměřit na kroky v důkazu Mizaru **take**, **consider** a **let**, odpovídající krokům v přirozené dedukci: zavedení existenčního kvantifikátoru, eliminace existenčního kvantifikátoru a zavedení obecného kvantifikátoru. Tedy obecně u kroků, kde se pracuje s proměnnými (které mají nějaký typ) a s dosazováním termů (které také mají nějaký typ).

4.6.2 Dokazovač atomických kroků

Dokazovač atomických kroků je v důkazech v systému Mizar reprezentován klíčovým slovem **by**. Cílem autorů tohoto dokazovače bylo, aby jeho síla odpovídala intuitivnímu pojmu zřejmých kroků v matematickém důkazu. Tedy aby tento dokazovač byl schopen dokázat kroky, které jsou v běžném matematickém důkazu zřejmé a není třeba je dále rozvádět. Na druhou stranu by neměl být zase moc silný, aby jeho síla nesváděla uživatele nerozepisovat v důkazu některé kroky, které zase tak zřejmé nejsou. Takový přístup by vedl k méně pochopitelným důkazům.

Idea zřejmých kroků je založena na omezení dokazovače, který může univerzálně kvantifikované formule instanciovat jen jednou. Pokud chceme, aby se tak stalo víckrát, musíme tuto formuli uvést jako předpoklad kroku **by** vícekrát. Takový přístup není zase tak umělý, jak by se mohlo zdát, neboť to odpovídá běžné matematické praxi, kdy se například při dokazování říká: „a nyní použijeme dvakrát transitivitu . . . “ [Wie01].

Dokazovač Mizaru byl s některými omezeními implementován v systému HOL Light Johnem Harrisonem na základě popisu jeho implementace v Mizaru v [FW00]. Harrisonova implementace je zastřešena funkcí `MIZAR_BY` a oproti originální verzi přítomné v Mizaru neobsahuje typový systém a speciální procedury pro dokazování v aritmetice. Integrace typového systému do této implementace byla už autorem této práce zkoumána a dotkla by se dvou míst.

Dokazovač funguje tak, že závěr, který se má dokázat z předpokladů, je znegován, a dokazovač se snaží odvodit spor. V části, která se nezabývá rovností, se zhruba řečeno formule převedou do disjunktivní normální formy a opakovanou instanciací obecně kvantifikovaných proměnných se hledá unifikace s ostatními formulemi tak, aby se odvodil spor [FW00]. Při hledání unifikací se dosazuje za proměnné a zde je první důležité místo pro integraci typového systému. Mohli bychom totiž na základě znalosti typů termů a proměnných předem vyloučit takové instanciace, které by typově neseděly. Lze očekávat, že toto prořezání typově nesouhlasících instanciací povede ke zrychlení dokazovače.

V části dokazovače, která pracuje s rovností, se hledají třídy termů, které si jsou rovny. Tedy uzávěr vzhledem k rovnosti [FW00]. Typy opět můžou pomoci odvodit spor v případě, že se nám podaří odvodit rovnost dvou termů, které mají neslučitelné typy, a tedy se nemohou rovnat.

Kapitola 5

Závěr

5.1 Splnění cíle

V rámci této diplomové práce byla popsána sémantika významné části typového systému Mizaru. Při tomto popisu byl kladen důraz na co nejpřesnější popis. Vzhledem k současnému neutěšenému stavu dokumentace systému Mizar a vzhledem ke komplexnosti typového systému Mizaru je výsledek této snahy významným přínosem a otevírá další širší možnosti pro experimentování s Mizarem.

Ná základě tohoto formálního popisu byl navržen překlad typového systému Mizaru do systému HOL Light. Tento překlad používá ideu reprezentovat závislé typy a atributy jako predikáty a výsledné typy reprezentovat jako konjunkci těchto predikátů. Součástí překladu je popis přeložení definic Mizaru pomocí částečného využití definičních prostředků systému HOL Light.

Součástí této diplomové práce je také implementace navrženého překladu. Kromě mechanismů spojených s atributy se podařilo realizovat celý překlad. Jedním z přínosů implementace je, že umožnila zachytit technické otázky spojené s převodem typového systému Mizaru. Jejím hlavním výsledkem je možnost odvozovat věty o typovém systému a sice podtypovou větu, která zachycuje relaci podtypu, a větu typu termu, která zachycuje výpočet typu termu.

Po integraci implementovaného typového systému do již realizovaných částí Mizaru v systému HOL Light lze dosáhnout faktické reimplementace Mizaru v jiném systému pro formalizaci matematiky. Pak bude možné zapsané důkazy v Mizaru verifikovat i v jiném systému, nebo vytvářet nové formalizace v systému HOL Light v prostředí, které využívá výhod Mizaru. Pokud by těchto dvou výsledků bylo dosaženo, byl by to nepřehlédnutelný pokrok na poli formalizace matematiky.

5.2 Možnosti dalšího vývoje

V této části budou nastíněny možnosti dalšího pokračování této práce. Většina z nich se už objevila někde dříve v tomto textu. Půjde tedy o stručné shrnutí:

- Doplnění popisu a překladu struktur, které nebyly v této práci uvažovány.
- Doplnění funktorových registrací obecně pro termy.
- Dokončení implementace i pro atributy.
- Integrace převodu typového systému do implementace dokazovacího jazyka a dokazovače atomických kroků Mizaru.
- Přidání implementace procedur pro dokazování v aritmetice do `MIZAR_BY`.
- Převod typového systému Mizaru do jiného sytému než je HOL Light. Nabízí se Isabelle/ZF, protože tento systém byl vybrán jako základ pro pokus o matematickou wiki vdash [vda].

Literatura

[Ban03] Grzegorz Bancerek. On the structure of mizar types. *Electronic Notes in Theoretical Computer Science*, 85(7):69 – 85, 2003. Mathematics, Logic and Computation (Satellite Event of ICALP 2003). Dostupné z: http://www.sciencedirect.com/science/article/B75H1-4DDWKN2-N0/2/b934352a640ded0897498ad796d1b1a8, doi:DOI:10.1016/S1571-0661(04)80758-8.

[BJ93] Tom Melham Bart Jacobs. Translating dependent type theory into higher order logic. In *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 209–229. Springer Berlin / Heidelberg, 1993. Dostupné z: http://www.springerlink.com/content/qx62712504838651/,doi:10.1007/BFb0037108.

[Bon90] Ewa Bonarska. *An Introduction to PC Mizar*. Brussels, 1990.

[FW00] Andrzej Trybulec Freek Wiedijk. *CHECKER - Notes on the Basic Inference Step in Mizar*, 2000. Dostupné z: http://www.cs.ru.nl/~freek/mizar/by.ps.gz.

[Gor08] Mike Gordon. Twenty years of theorem proving for HOLs past, present and future. In *Theorem Proving in Higher Order Logics*, volume 5170 of *Lecture Notes in Computer Science*, pages 1–5. Springer Berlin / Heidelberg, 2008. Dostupné z: http://www.springerlink.com/content/v09876746405t946/,doi:10.1007/978-3-540-71067-7_1.

[Har96] John Harrison. A mizar mode for HOL. In *Theorem Proving in Higher Order Logics*, volume 1125 of *Lecture Notes in Computer Science*, pages 203–220. Springer Berlin / Heidelberg, 1996. Dostupné z: http://www.springerlink.com/content/r826284r11657937/, doi:10.1007/BFb0105406.

[Har07] John Harrison. *HOL Light Tutorial*, 2007. Dostupné z: http://www.cl.cam.ac.uk/~jrh13/hol-light/tutorial_220.pdf.

[Har09] John Harrison. *The HOL Light System*, 2009. Dostupné z: http://www.cl.cam.ac.uk/~jrh13/hol-light/reference_220.pdf.

[JH06] Rob Arthan John Harrison, Konrad Slind. HOL. In *The Seventeen Provers of the World*, volume 3600 of *Lecture Notes in Computer Science*, pages 11–19. Springer Berlin / Heidelberg, 2006. Dostupné z: http://www.springerlink.com/content/r777730148r2w33h/,doi:10.1007/11542384_3.

[JU09] Geoff Sutcliffe Josef Urban. ATP-based Cross-Verification of Mizar Proofs: Method, Systems, and First Experiments. *Mathematics in Computer Science*, 2:231–251, 2009.

[miz] Mizar syntax, verze 7.11.02 [online]. Dostupné z: http://mizar.uwb.edu.pl/language/mizar-grammar.xml [citováno 23. 1. 2009].

[Muz93] Michal Muzalewski. *An Outline of PC Mizar*. Brussels, 1993.

[MW02] Freek Wiedijk Markus Wenzel. A comparison of Mizar and Isar. *J. Automated Reasoning*, 29:389–411, 2002.

[PR01] Andrzej Trybulec Piotr Rudnicki, Christoph Schwarzweller. Commutative algebra in the mizar system. *Journal of Symbolic Computation*, 32(1-2):143 – 169, 2001. Dostupné z: http://www.sciencedirect.com/science/article/B6WM7-457V5FY-B/2/e5bf2e99e5df22932748496babffc20e, doi:DOI:10.1006/jsco.2001.0456.

[Sch07] Christoph Schwarzweller. Mizar Attributes: A Technique to Encode Mathematical Knowledge into Type Systems. *Studies in Logic, Grammar and Rhetoric*, 10(23):387–400, 2007.

[Try93] Andrzej Trybulec. *Some Features of the Mizar Language*, 1993.

[vda] vdash: a formal math wiki [online]. Dostupné z: http://www.vdash.org/ [citováno 7. 8. 2009].

[Wie] Freek Wiedijk. *Writing a Mizar article in nine easy steps*. Dostupné z: http://www.cs.ru.nl/~freek/mizar/mizman.pdf.

[Wie99] Freek Wiedijk. *Mizar: An Impression*, 1999. Dostupné z: http://www.cs.ru.nl/~freek/mizar/mizarintro.pdf.

[Wie01] Freek Wiedijk. Mizar light for HOL light. In *Theorem Proving in Higher Order Logics*, volume 2152 of *Lecture Notes in Computer Science*, pages 378–393. Springer Berlin / Heidelberg, 2001. Dostupné z: http://www.springerlink.com/content/wegk7kcy57rxwuqc/, doi:10.1007/3-540-44755-5_26.

Příloha A

Obsah přiloženého CD

K této diplomové práci je přiloženo CD, které obsahuje tuto práci v elektronické podobě, implementaci překladu v systému HOL Light a kód příkladu použitého v 4.5. Obsah tohoto CD je následující:

- `thesis.pdf` – tato diplomová práce v elektronické podobě
- `code/` – adresář s kódem implementace překladu v systému HOL Light
- `code/mizar_types.ml` – vlastní implementace překladu
- `code/example.ml` – kód příkladu použitý pro demonstraci v 4.5.

Pro spuštění implementace je potřeba mít nainstalován systém HOL Light. Pro zde prezentovanou implementaci byl použit systém HOL Light ve vývojové verzi 2.20+ z 25. 5. 2009. Tento systém byl spouštěn v prostředí OCamlu verze 3.11.0 s Camlp5 verze 5.12. Pro spouštění přiložené implementace je doporučeno použít stejné nebo novější verze uvedených programů.

Po spuštění systému HOL Light lze nahrát implementaci z CD následujícím příkazem v systému HOL Light:

```
# loads "<cesta k CD>/code/mizar_types.ml";;
```

Po jejím načtení lze spustit kód demonstrace analogickým příkazem:

```
# loads "<cesta k CD>/code/example.ml";;
```