VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Implementace jazyka TIL-Script

Implementation of the TIL-Script Language

Bc. Filip Peterek

Diplomová práce

Vedoucí práce: prof. RNDr. Marie Duží, CSc.

Ostrava, 2023

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Č.j.VSB/22/078945
vsboee2202603F

Zadání diplomové práce

Student: **Bc. Filip Peterek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Implementace jazyka TIL-Script
Implementation of the TIL-Script Language

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je implementovat funkcionální jazyk TIL-Script. Implementace bude vycházet z gramatiky jazyka, která je v souladu s logickým systémem Transparentní Intensionální Logiky (TIL).

Práce bude obsahovat:
1. Popis systému TIL, tj. jazyk konstrukcí a rozvětvená teorie typů.
2. Definici jazyka TIL-Script.
3. Analýzu, návrh a implementaci jazyka TIL-Script včetně typové kontroly.
4. Dokumentace celého projektu včetně uživatelské příručky.

Implementace bude provedena v jazyce C# nebo Java.

Seznam doporučené odborné literatury:

[1] Duží M., Materna P. (2012): TIL jako procedurální logika (průvodce zvídavého čtenáře Transparentní intensionální logikou). Aleph Bratislava 2012, ISBN 978-80-89491-08-7
[2] Duží M., Jespersen B. and Materna P. (2010): Procedural Semantics for Hyperintensional Logic. Foundations and Applications of Transparent Intensional Logic. First edition. Berlin: Springer, series Logic, Epistemology, and the Unity of Science, vol. 17, ISBN 978-90-481-8811-6.
[3] Ciprich N., Duží, M., Košinár M. (2009): The TIL-Script Language. Frontiers in Artificial Intelligence and Applications, Amsterdam: IOS Press, vol. 190, pp. 166-179.

17. listopadu 2172/15
708 00 Ostrava-Poruba
Česká republika

spojovatelka: +420 597 321 111
epodatelna: epodatelna@vsb.cz
ID datové schránky: d3kj88v

IČ: 61989100
DIČ: CZ61989100

email: studijni.fei@vsb.cz
www.fei.vsb.cz

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **prof. RNDr. Marie Duží, CSc.**

Datum zadání: 01.09.2022

Datum odevzdání: 30.04.2023

Garant studijního oboru: prof. RNDr. Václav Snášel, CSc.

V IS EDISON zadáno: 07.11.2022 11:59:22

Abstrakt

Cílem práce je implementovat programovací jazyk TILScript. Jazyk TILScript slouží jako výpočetní varianta logického kalkulu TIL, jenž umožňuje jednoduchý strojový zápis konstrukcí Transparentní intenzionální logiky, ale také jejich následné provedení. Práce dále řeší praktické problémy s interpretací TILScriptu, a to například definice pojmenovaných funkcí, interakce s databází, apod. Dále se práce snaží navrhnout nadmnožinu jazyka TILSkript, která umožní konstrukce TILu nejen provádět, ale také analyzovat, vytvářet je, a pracovat s nimi.

Klíčová slova

Transparentní intenzionální logika, TILScript, překladač

Abstract

The goal of the thesis is the definition and implementation of the TILScript language. TILScript is a scripting language which serves the purpose of a computational variant of Transparent intensional logic, a logical calculus based on typed lambda calculi. TILScript allows for not just representation, but also execution of TIL constructions. This work also deals with practical problems of TILScript implementation, such as definitions of named functions, interaction with databases, and so on. Furthermore, this thesis attempts to define a superset of the TILScript language, which allows for not just the execution of constructions, but also for their creation and analysis.

Keywords

Transparent intensional logic, TILScript, interpreter

Obsah

Seznam použitých zkratek a symbolů

TIL	–	Transparentní intenzionální logika
JVM	–	Java Virtual Machine
TCO	–	Tail Call Optimization

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Analýza přirozeného jazyka jako disciplína stále rychleji stoupá na oblibě i důležitosti. Jistě málokomu unikly například n-gramové modely založené na predikci následujícího slova z předcházejících n slov, či vektorové modely jako Word2Vec, umožňující reprezentovat význam slov pomocí vektorů. Poslední dobou se velmi často mluví o jazykovém modelu GPT-3.

Výčet přístupů k analýze přirozeného jazyka však nekončí n-gramy a neuronovými sítěmi. K analýze přirozených jazyků lze přistoupit také za pomocí logiky. Mezi průkopníky tohoto přístupu patřil také český logik Pavel Tichý, tvůrce Transparentní intenzionální logiky.

Transparentní intenzionální logika (dále také TIL) je logický systém založený na typovaném lambda kalkulu. TIL umožňuje modelovat a analyzovat výrazy přirozeného jazyka za pomocí logiky. Jako doplněk TILu poté vznikl také funkcionální programovací jazyk TILScript, sloužící k interpretaci konstrukcí Transparentní intenzionální logiky. Syntax i sémantika TILScriptu jsou silně inspirovány TILem, ovšem přirozeně s určitými úpravami tak, aby byl TILScript rozumně zapisovatelný a interpretovatelný na počítači.

A právě interpretací TILScriptu se zabývá tento text. Součástí práce tak je navázání na předcházející vývoj jazyka TILScript, rozšíření gramatiky TILScriptu nutné pro akomodaci nových prvků jazyka (např. direktiva *import* sloužící k importu symbolů definovaných v jiném souboru), nebo také samotná interpretace jazyka TILScript, společně s kontrolou typové koherence. Dále se práce navrhuje způsob, jak volat funkce platformy JVM. Tento přístup otevírá jazyku TILScript celý ekosystém platformy JVM. Zatímco tedy práce nijak neřeší a neimplementuje získávání informací z již existující databáze, soustředí se spíše na zjednodušení přístupu k existujícím knihovnám, aby do budoucna nebyl problém rychle a jednoduše naprogramovat přístup k libovolnému zdroji dat.

Nakonec se práce snaží navrhnout určitou nadmnožinu jazyka TILScript tak, aby TILScript mohl sloužit nejen k interpretaci konstrukcí TIL, ale také k jejich tvorbě a analýze. V současné době pro analýzu a práci s konstrukcemi TILScriptu existují nástroje psané v jazyce Java. Existuje-li však v současné době programovací jazyk vytvořený přímo pro logiky pracující s TILem, není důvod tento jazyk nevyužít a nerozšířit také o nástroje pro analýzu, tvorbu i transformaci TILScript konstrukcí.

Pro pokročilejší práci s TILScriptem by tak nebylo třeba učit se jiný programovací jazyk (Java), nebo využívat externí nástroje, ale naopak by byl k dispozici již familiérní nástroj, jehož základy jsou již příznivům Transparentní intenzionální logiky dobře známy. Cílem této nadmnožiny ovšem není předefinovat TIL, či nějak výrazně měnit jádro TILScriptu, jinak by se ostatně také nemohlo jednat o nadmnožinu. Tyto nové prvky jsou od jádra TILScriptu, které slouží převážně jako výpočetní varianta TIL, jednoduše oddělitelné. Při práci s interpreterem, jehož implementace je součástí této práce, není problém se této nadmnožině vyhnout a nevyužívat ji. Také lze, na základě tohoto textu, implementovat interpreter samotného TILScriptu bez jakýchkoliv rozšíření.

Práce se do znatelné míry inspiruje výzkumem amerického profesora Johna McCarthyho a programovacím jazykem Lisp. Jelikož Lisp vychází z lambda kalkulu, pracuje s parciálními funkcemi, a nerozlišuje mezi programem a daty (tedy Lisp programy můžou pracovat s, generovat a transformovat Lisp konstrukce), lze mezi Lispem a TILem nalézt řadu podobností. Primárním odlišovacím znakem Transparentní intenzionální logiky je ovšem rigorózně definovaný typový systém.

Text samotný je rozdělen do čtyř částí. V první části budou představeny základy Transparentní intenzionální logiky, jejichž znalost je nutná k pochopení práce. Druhá část představí programovací jazyk TILScript a vyznačí změny a úpravy, které tato práce navrhuje. Třetí část popíše technické řešení a implementaci interpreteru. V poslední části poté bude zdokumentována standardní i matematická knihovna, které jsou součástí implementace. Dále bude poslední část obsahovat návod na použití, návod na implementaci vlastní knihovny, a nakonec také ukázky programů psaných v jazyce TILScript.

V terminologii programovacích jazyků a překladačů existuje spousta zavedených anglických názvů, jejichž český ekvivalent je méně zavedený, případně neexistuje vůbec. Proto bude práce ve spoustě případů uvádět také anglický ekvivalent k českému výrazu.

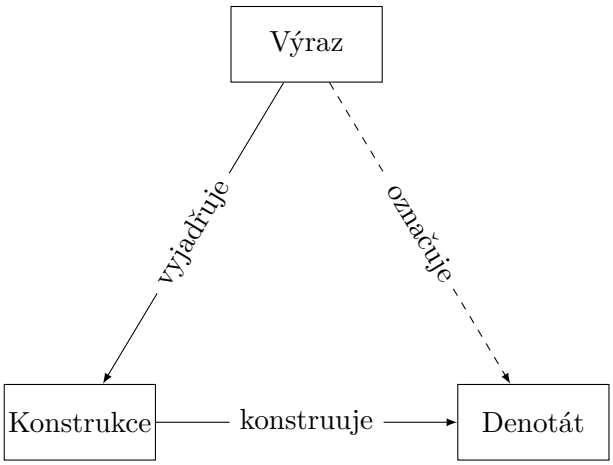
Kapitola 2

Transparentní intenzionální logika

Transparentní intenzionální logika (TIL) je logický systém založený na typovaném lambda kalkulu. TIL je využíván k logické analýze přirozeného jazyka. Oproti tradičnímu lambda kalkulu, jenž se využívá jako komputační model, tedy jako pouhý prostředek k dosažení konkrétní hodnoty – výsledku, v Transparentní intenzionální logice hraje konstrukce kalkulu často důležitější roli, než hodnota, kterou by konstrukce po provedení zkonstruovala.

Jako příklad využití lambda kalkulu jako výpočetní model lze uvést např. funkcionální programovací jazyk Haskell. Interně je Haskell kompilován do lambda kalkulu (přesněji do jeho supersetu obsahujícího např. čísla nebo logické hodnoty, která jinak v lambda kalkulu musíme kódovat pomocí Churchova kódování – K-kombinátorů, apod.). Ultimátně v Haskellu ovšem lambda kalkul slouží pouze k získání konkrétního výsledku. Nadefinujeme vztah mezi vstupem a výstupem, a program napsaný v Haskellu nám vstup transformuje. Pokud zanedbáme efektivitu programu, nezajímá nás, jakým způsobem program spočítal výsledek, dokud jej spočítal správně.

Naopak Transparentní intenzionální logika je hyperintenzionálním kalkulem, který nám umožňuje vytvářet konstrukce vypovídající o jiných konstrukcích. TIL vychází z myšlenky, že výraz přirozeného jazyka sice označuje denotát – konkrétní individuum, významem výrazu ovšem není samotný denotát, který ani nemusí nutně existovat. Význam výrazu je abstraktní a lze jej zachytit konstrukcí. Daná konstrukce poté při provedení zkonstruuje denotát výrazu. Jako příklad lze uvést například výraz "francouzský král."V době psaní této práce Francie krále nemá. Výraz nemá žádný denotát, neuvádí žádné konkrétní individuum. Přesto výrazu "francouzský král"rozumíme, výraz má svůj význam, jen v současné době neuvádí žádnou osobu. A budeme-li chtít o významu výrazu "francouzský král"něco vypovědět, například že francouzský král je monarchou v čele Francie, daný monarcha nemusí existovat. Dále lze uvést například rozdíl mezi výrazy "logaritmus 1024 o základě 2"a "5 + 5". Denotátem obou výrazů je 10. Zadáme-li do interpreteru Haskellu výrazy `logBase 2 1024` a `5 + 5`, získáme v obou případech stejný výsledek. V přirozeném jazyce ovšem chápeme značný rozdíl mezi oběma výrazy, ačkoliv mají stejný denotát. "Logaritmus 1024 o základě 2"vyja-



Obrázek 2.1: Schéma procedurální sémantiky TIL

dřuje číslo, kterým musíme umocnit dvojku, abychom získali 1024. Výraz "5 + 5"očividně vyjadřuje úplně jinou matematickou operaci a jeho výsledek spočítáme jiným postupem.

Denotátem výrazu může být nejen objekt z báze, ale i konstrukce nebo funkce.

Jak již bylo zmíněno, Transparentní intenzionální logika vychází z typovaného lambda kalkulu, proto také každý objekt musí mít svůj typ. Pro správné pochopení TILu, a tedy i této práce, je tak nutné znát typovou hierarchii TIL.

2.1 Báze

Báze je kolekce vzájemně disjunktních neprázdných množin, které dohromady definují s jakými objekty budeme pracovat. Tyto množiny definují atomické objekty. Každá množina dále objektům určuje určitá základní kritéria (např. pokud jako jednu z množin báze zvolíme množinu $\mathbb{N}$, víme, že všechny objekty z této množiny budou čísla). Vždy se ovšem jedná pouze o nejnutnější a nejzákladnější vlastnosti. Báze například nemá žádný vliv na vlastnosti proměnlivé v čase.

Bázi volíme dle potřeb konkrétní aplikace a univerza diskurzu. Například používáme-li TIL k logické analýze matematických vět, jako bázi lze zvolit například množinu celých čísel, množinu reálných čísel, a množinu pravdivostních hodnot. Musíme však vzít v potaz, že tato báze neobsahuje čísla komplexní.

Patří-li objekt x do množiny α z báze, říkáme, že se jedná o objekt typu α. K explicitnímu uvedení typu objektu x využíváme zápis x/α. Množinám tvořícím bázi lze přirozeně říkat typy.

Pro analýzu přirozeného jazyka se většinou volí objektová báze skládající se z typů o, ι, τ, ω. Tyto typy jsou podrobněji popsány v tabulce 2.1.

Tabulka 2.1: Výchozí báze pro analýzu přirozeného jazyka

Typ	Popis typu
o	Množina pravdivostních hodnot
ι	Množina individuí (univerzum diskurzu)
τ	Množina časových okamžiků/reálných čísel
ω	Množina logicky možných světů

2.2 Funkce

V matematice se jako základní molekulární typ využívají relace. Funkce je poté speciální typ relace, který je zprava jednoznačný. V TIL je však základním molekulárním typem funkce. Chceme-li v TIL vyjádřit n-ární relaci nad množinou $\alpha_1 \times ... \times \alpha_n$, lze tak samozřejmě udělat definicí n-ární funkce z $\alpha_1 \times ... \times \alpha_n$ do o, která každému prvku z $\alpha_1 \times ... \times \alpha_n$ přiřadí pravdivostní hodnotu na základě toho, zda prvek do relace patří, nebo ne.

Narozdíl od tradičního lambda kalkulu, kde jsou funkce pouze nulární nebo unární, v Transparentní intenzionální logice není arita funkce omezena. Dále můžeme v TIL pracovat s funkcemi parciálními.

2.2.1 Intenze a extenze

V TIL dále rozlišujeme funkce na tzv. *intenze* a *extenze*. Intenze jsou funkce z možných světů. Extenze jsou funkce, jejichž doménou množina možných světů není, a tudíž jejich funkční hodnota nezávisí na stavu světa.

Intenze jsou obecně funkce typu $(\alpha\omega)$ pro libovolný typ α. Nejčastěji se však jedná o funkce typu $((\alpha\tau)\omega)$, tedy funkce zobrazující možné světy do chronologií objektů typu α.

2.3 Konstrukce TIL

Konstrukce v Transparentní intenzionální logice jsou abstraktní procedury. Tyto procedury jsou strukturované – nejedná se o množiny, mají pevně danou strukturu, a na uspořádání případných podprocedur záleží. Tyto konstrukce lze podle definovaných pravidel provést. Provedením konstrukce, získáme výstup, případně nezískáme nic. Konstrukce, které nekonstruují žádný výstup, nazýváme *nevlastní* (anglicky *improper*). V TIL pracujeme s šesti druhy konstrukcí.

Jak již bylo zmíněno, konstrukce můžou v TIL operovat nejen nad objekty, které nejsou konstrukcemi, tedy nad objekty z báze, ale také nad jinými konstrukce. Konstrukce však může operovat pouze nad konstrukcemi nižšího řádu, než je konstrukce samotná, viz 2.5. Každou podkonstrukci, kterou musíme provést při provedení konstrukce, nazýváme *konstituentem*. V TIL existuje šest různých druhů konstrukcí. Dvě atomické – mají pouze jeden konstituent, a to sebe samotné, a čtyři

molekulární. Atomickými konstrukcemi jsou *trivializace* a *proměnné*. Mezi molekulární konstrukce poté řadíme *kompozice*, *uzávěry*, *provedení* a *dvojí provedení*.

Proměnné jsou konstrukce, které na základě valuace v v-konstruují objekty. Skutečnost, že proměnná x v-konstruuje hodnotu typu α značíme $x \rightarrow_v \alpha$.

Trivializace pro libovolný objekt X konstruuje samotný objekt X. Konstrukce je v Transparentní intenzionální logice potřebná, neboť výchozím módem pro konstrukce je provedení. Samotná konstrukce X by tak byla automaticky provedena, a místo konstrukce X bychom dostali pouze její denotát. Pokud bychom chtěli zkonstruovat konstrukci X, musíme ji trivializovat. Tím se provede pouze konstrukce trivializace. A protože trivializace nemá jiný konstituent, než sebe samotnou, konstrukce X se tak neprovede. V literatuře se trivializace X tradičně značí ${}^0X$. Obzvlášť se používá také zápis $'X$. Tento zápis je poté využit i v jazyce TILScript. Trivializaci lze považovat za ekvivalent funkce `QUOTE` z jazyka Lisp. Trivializace taktéž bývá využívána ke konstruování hodnot, které nelze provést (objekty z báze, funkce) a tudíž je nelze zmínit netrivializované.

Kompozice je využívána k aplikaci funkcí. Kompozice $[XY_1...Y_m]$ značí aplikaci funkce konstruované konstrukcí X na argumenty zkonstruované konstrukcemi $Y_1, ..., Y_m$. Pokud konstrukce X konstruuje funkci f, všechny podkonstrukce $Y_1, ..., Y_m$ konstruují hodnotu, a je-li funkce f na daných argumentech definovaná, kompozice v-konstruuje funkční hodnotu na těchto argumentech. V opačném případě je kompozice nevlastní.

Uzávěr $\lambda x_1...x_m Y$ je konstrukce v-konstruující funkci. $x_1, ...x_m$ musí být navzájem různé proměnné, Y musí být konstrukcí. Konstruce uzávěru je velmi podobná abstrakci v lambda kalkulu. Narozdíl od lambda kalkulu však v TILu může uzávěr konstruovat funkce s aritou vyšší než jedna. Uzávěr nemůže být nikdy nevlastní, může však konstruovat tzv. *degenerovanou funkci*, tedy funkci, která je nedefinovaná na celém definičním oboru.

Provedení ${}^1X$ je konstrukce v-konstruující objekt konstruovaný konstrukcí X. Pokud je konstrukce X v-nevlastní, je provedení ${}^1X$ také v-nevlastní. Jelikož je však provedení výchozím módem pro objekty, většinou se neuvádí. Provést lze pouze konstrukce. Objekty z báze (tedy čísla, individua, apod...) či funkce nelze provést, jejich provedení nekonstruuje nic. Proto je potřeba tyto objekty vždy trivializovat.

Dvojí provedení ${}^2X$ je poslední z výčtu konstrukcí. Je-li X konstrukcí v-konstruující konstrukci Y, a v-konstruuje-li konstrukce Y objekt Z, pak ${}^2X$ v-konstruuje Z. V opačném případě je dvojí provedení ${}^2X$ v-nevlastní.

Jiné konstrukce v Transparentní intenzionální logice neexistují.

2.3.1 Princip kompozicionality

Princip kompozicionality je důležitým rysem Transparentní intenzionální logiky. Princip kompozicionality uvádí, že je-li libovolný konstituent konstrukce X v-nevlastní a v dané valuaci nekonstruuje žádnou hodnotu, pak je v-nevlastní i konstrukce X.

2.4 Typy 1. řádu

Definice je skoro slovo od slova převzata z knihy *TIL jako procedurální logika – Průvodce zvídavého čtenáře Transparentní intensionální logikou*. Tato sekce slouží jako krátké vysvětlení základů Transparentní intenzionální logiky se čtenář může obrátit například na tuto knihu.

Nechť B je báze. Pak:

i) Každá množina z báze B je atomický typ řádu 1 nad B.

ii) Nechť $\alpha, \beta_1, ..., \beta_m (m > 0)$ jsou typy řádu 1 nad B. Pak soubor všech m-árních parciálních funkcí $(\alpha\beta_1...\beta_m)$, tedy zobrazení z $\beta_1 \times ... \times \beta_m$ do α, je molekulární typ řádu 1 nad B.

iii) Nic jiného není typem řádu 1 nad bází B.

2.5 Rozvětvěná hierarchie typů

Definice je opět skoro slovo od slova převzata z.

Nechť B je báze. Pak:

2.5.1 T_1 (typy řádu 1)

Viz sekce 2.4.

2.5.2 C_n (konstrukce řádu n)

i) Nechť x je proměnná v-konstruující objekt typu řádu n. Pak x je *kontrukce řádu n nad b*.

ii) Nechť X je prvek typu řádu n. Pak trivializace ${}^0X$, provedení ${}^1X$ a dvojí provedení ${}^2X$ jsou *konstrukcemi řádu n nad b*.

iii) Nechť $X, Y_1, ...Y_m$ jsou konstrukce řádu n nad B. Pak kompozice $[XY_1...Y_m]$ je *konstrukce řádu n nad b*.

iv) Nechť $x_1, ..., x_m$ jsou vzájemně různé proměnné a X je konstrukce řádu n nad B. Pak uzávěr $[\lambda x_1...x_m X]$ je *konstrukce řádu n nad b*.

v) Nic jiného není konstrukcí řádu n nad bází B.

2.5.3 T_{n+1} (typy řádu n+1)

Nechť $*_n$ je kolekce všech konstrukcí řádu n nad B.

i) $*_n$ a každý typ řádu n jsou *typy řádu n+1 nad B*.

ii) Jsou-li $\alpha, \beta_1, ..., \beta_m$ typy řádu $n+1$ nad B, pak $(\alpha\beta_1...\beta_m)$, tedy kolekce parciálních funkcí, je *typy řádu n+1 nad B*.

iii) Nic jiného není typ řádu $n+1$ nad B.

2.6 Analytické a empirické výrazy

V TIL lze výrazy dělit na dva různé typy výrazu, a to empirické a analytické typy.

Analytické výrazy jsou výrazy takové, které nezávisí na současném stavu světa. Jedná se například o matematické věty nebo rekvizity vlastností (např. věta "Všechny velryby jsou savci"je analytická a nutně pravdivá nezávisle na stavu světa, neboť existuje-li individuum, které je velrybou, pak bude vždy také savcem.)

Empirické výrazy naopak na stavu světa závisí. Abychom vyhodnotili takový výraz, musí prozkoumat konkrétní stav světa ve zkoumaném momentě.

Kapitola 3

TILScript

Nyní konečně přišel čas představit TILScript. TILScript je interpretovaný funkcionální programovací jazyk, do znatelné míry inspirovaný jazyky jako Haskell nebo Lisp. Syntax TILScriptu by měla co nejvíce připomínat syntaktické prvky Transparentní intenzionální logiky, aby pouhá znalost TILu stačila k okamžitému pochopení TILScriptu. Sémantika by poté měla být stejná.

Tato kapitola je rozdělena do tří sekcí. V první sekci jsou popsány důležité základní rysy TILScriptu. Druhá sekce popisuje již existující prvky TILScriptu, a případně dokumentuje změny oproti předchozím verzím TILScriptu. Poslední sekce se věnuje navrhovaným rozšířením jazyka TILScript.

3.1 Charakteristické rysy TILScriptu

Tato sekce popisuje charakteristické rysy TILScriptu v takové podobě, jakou nabývá v této práci. Pokud se v některém bodě TILScript neshoduje s TILem či předchozími verzemi TILScriptu, je rozdíl náležitě popsán a vysvětlen.

3.1.1 Lambda kalkul parciálních funkcí

3.1.1.1 Shora neomezená arita funkcí

Narozdíl od lambda kalkulu ve své tradiční podobě, nebo například jazyka Haskell, v Transparentní intezionální logice není arita funkce shora omezená. TILScript musí tento fakt reflektovat. Proto tento jazyk umožňuje definici i aplikaci funkcí libovolné (samozřejmě nezáporné) arity. Také zde neexistuje rozvíjení funkcí (anglicky *currying*). Zatímco např. v Haskellu jsou funkce arity dvě nebo vyšší automaticky rozvinuty na sérií několika unárních funkcí, jejichž oborem hodnot jsou unární nebo nulární funkce, a jedné nulární funkce která vrací žádaný výsledek, v TILScriptu není arita nijak omezená.

3.1.1.2 Parciální funkce a respektování principu kompozicionality

Jelikož v TIL můžou být funkce parciální, musí i TILScript počítat s parcialitou funkcí. Dále musí TILScript respektovat princip kompozicionality, základní rys Transparentní intenzionální logiky. Jedním z důsledků principu kompozicionality je, že konstrukce, jejíž přinejmenším jeden konstituent je nevlastní, bude také nutně nevlastní. Reprezentaci stavu, kdy parciální funkce je aplikována na argumenty, na kterých není definována, se věnuje podsekce *Hodnota Nil* 3.3.3 této kapitoly.

Jedinou výjimkou je funkce `IsNil`, jež vrací pravdivostní hodnotu `True`, pokud je její jediný argument `Nil`, v opačném případě je jejím výsledkem `False`. Tato speciální sémantika funkce `IsNil` , ačkoliv porušuje princip kompozicionality a vyžaduje aplikaci unární funkce na "nic,"je zvolena jako doplněk k funkci `Improper/(o*_n)` definované v Průvodci čtenáře, a jako kompromis mezi dodržením principů TIL a umožněním zpracování chyb.

3.1.2 Neměnnost proměnných a symbolů (*Immutability*)

Jelikož je TILScript funkcionální jazyk, jsou hodnoty všech proměnných konstantní – tedy jakmile je proměnné jednou přiřazena valuace, nelze její hodnotu změnit. Dále nelze proměnnou zastínit (angl. *to shadow, shadowing*) v rámci oblasti platnosti (angl. *scope*), ve které byla definována. Proměnnou lze zastínit vytvořením nové oblasti platnosti (tedy například na novém rámci zásobníku, angl. *stack frame*).

Obdobně nelze redefinovat funkce nebo změnit typ symbolické hodnoty (viz 3.2.14).

3.1.3 Definice a deklarace symbolů

TILScript nově rozlišuje mezi deklaracemi a definicemi proměnných a funkcí. Deklarace pouze uvědomí překladač o existenci proměnné nebo funkce, nijak ale nedefinuje valuaci proměnné nebo sémantiku funkce. Deklarace umožňuje funkci či proměnnou zmínit (např. v trivializaci, v uzávěru), neumožňuje nám však proměnnou provést nebo funkci aplikovat – jak také, když neznáme hodnotu proměnné, případně sémantiku dané funkce. Provedení deklarované, avšak nedefinované proměnné je chybou, při které interpreter ohlásí chybu a běh programu je ukončen. Deklarovat jeden symbol lze vícekrát, deklarace však nesmí být konfliktní a lišit se typy.

```
$ java -jar interpreter/build/libs/tilscript.jar examples/undef-var.tils
** Error **
(4, 1): myVar.
    ~~~ ^ ~~~
        Variable 'myVar' is declared but undefined
$ java -jar interpreter/build/libs/tilscript.jar examples/undef-fn.tils
** Error **
(2, 1): MyFn/(Int Int Int).
```

```
~~~ ^ ~~~
    Function MyFn is declared but undefined, application is impossible
```

Listing 3.1: Hlášení chyby při chybějící definici

Definice přiřadí proměnné valuaci, funkci sémantiku. Proměnné s řádnou definicí lze provést a můžou tak být konstituentem prováděné konstrukce. Funkce s řádnou definicí lze aplikovat. Funkce i proměnné lze definovat pouze jednou. Opakovaná definice je chybou a vyústí v předčasné ukončení programu.

Symbolické hodnoty, viz 3.2.14, lze pouze deklarovat.

Deklarace jsou automaticky odvozeny z definic. Proto, pokud je známa definice, není třeba dodávat také deklaraci. K interpretaci deklarací automaticky dochází před interpretací definic, aby byla umožněna např. definice vzájemně rekurzivních funkcí. Definice jsou interpretovány v takovém pořadí, v jakém jsou uvedené ve zdrojovém kódu.

Deklarace bez řádných definic na první pohled můžou působit zbytečně. K čemu může sloužit funkce, kterou nelze aplikovat? Nesmíme však zapomenout, že konstrukce TIL samy vyjadřují význam, a nemusí nutně sloužit k provedení. Provedením konstrukce sice dostaneme její denotát, ten nás ale ne vždy zajímá. Představme si tedy případ, kdy provádíme analýzu výrazu přirozeného jazyka. Výraz analyzujeme pomocí Transparentní intenzionální logiky a získáme konstrukci. S danou konstrukcí chceme dále pracovat a chceme ji strojově zpracovat. Její denotát nás ovšem nezajímá, zajímá nás pouze význam konstrukce. Současně daná konstrukce obsahuje funkci, jejíž definici neznáme, známe, ale nejsme schopni ji strojově vyjádřit, nebo nás pouze nezajímá. Jelikož víme, že konstrukci nebudeme provádět, a tedy nebudeme ani aplikovat funkci v ní zmíněnou, nepotřebujeme znát její přesnou definici. Stačí nám znát pouze její název a typ.

Jako příklad nevyjádřitelné funkce lze uvést například všeobecný kvantifikátor. Ačkoliv všeobecný kvantifikátor existuje jako funkce v Transparentní intenzionální logice, nelze jej korektně definovat tak, aby byl vždy strojově vyhodnotitelný.

Názvosloví *deklarace*, *definice* je převzáno z programovacího jazyka C, kde deklarace pouze uvědomí překladač o existenci symbolu, definice poté přiřadí symbolu konkrétní hodnotu. Počet deklarací je shora neomezený, zato definice může existovat nanejvýš jedna. Deklarace nedefinovaného symbolu není chybou, ovšem snaha nedefinovaný symbol využít (např. volání funkce, přístup k proměnné) vyústí v chybu při procesu linkování.

3.2 TILScript jako výpočetní varianta TILu

Tato sekce popisuje základní výrazy a konstrukce TILScriptu, které existovaly již v předchozích verzích jazyka. Pokud práce tyto výrazy nějakým způsobem upravuje, je úprava náležitě popsána a zdůvodněna.

3.2.1 Věty TILScriptu

V TILScriptu za věty (*sentence*) považuje výrazy na nejvyšší úrovni v programu. Větou je tedy například konstrukce taková, že není podkonstrukcí jiného výrazu než sebe samotné, ale také definice funkce, proměnné, typu, apod. Každá věta musí být ukončena terminátorem. Roli terminátoru v TILScriptu zastává znak `.` (ASCII tečka).

3.2.2 Atomické datové typy

Atomické datové typy v TILScriptu vycházejí z výchozí báze využívané v Transparentní intenzionální logice k analýze přirozeného jazyka, tedy množinám o, ι, τ, ω. TILScript ovšem rozlišuje mezi časy a reálnými čísly, a pro tyto hodnoty definuje dva nekompatibilní typy, mezi kterými neexistuje implicitní konverze. Dále TILScript využívá datový typ ν představující celá čísla. Nakonec TILScript pro názvy typů nevyužívá řecká písmena, která nelze prakticky a jednoduše zapisovat na spoustě rozložení klávesnic, ale anglická slova nebo zkratky. Názvy typů vždy začínají velkým písmenem.

Typ o představující pravdivostní hodnoty TILScript pojmenovává `Bool` a může nabývat hodnot `True` a `False`.

Typ ι, v TILScriptu `Indiv`, označujeme jako množinu individuí. Individua v Transparentní intenzionální logice považujeme za "holá"– žádnou netriviální vlastnost nemají nutně. Všechny netriviální vlastnosti individuí jsou určeny stavem světa. Individuum samotné nemá žádnou inherentní valuaci. Slouží pouze jako unikátní indentifikátor. Obdobně hodnoty `Indiv` v jazyce TILScript nemají žádnou konkrétní reprezentaci. Typ `Indiv` je využíván v konjunkci se symbolickými hodnotami, viz 3.2.14. Tímto TILScript umožňuje uživateli referovat na konkrétní individuum pouze pomocí symbolického identifikátoru, aniž by individuím musely být přiřazeny arbitrárně zvolené konkrétní hodnoty.

Reálná čísla TILScript reprezentuje typem `Real`. V implementaci překladače vytvořeném v rámci této práce jsou reálná čísla interně reprezentována typem `double`. TILScript samotný žádné omezení na reprezentaci reálných čísel nestanovuje, prakticky však reálná čísla v současné implementaci reprezentujeme pomocí 64bitové reprezentace dle IEEE 754.

Celá čísla TILScript reprezentuje typem `Int`. Obdobně jako u typu `Real` neexistuje omezení pro reprezentaci celých čísel. Interně je využíván datový typ `long`, jedná se tedy o 64bitové znaménkové číslo reprezentované dvojkovým doplňkem.

Množinu možných časů modelujeme typem `Time`. Pro interní reprezentaci časových okamžiků byl v této práci zvolen datový typ `long`. Uživatel se sám může rozhodnout, jak bude tyto hodnoty interpretovat. Ve standardní knihovně lze však nalézt např. funkci `Now`, jejíž aplikací získáme počet milisekund uplynulých od 1. ledna 1970.

Dále byl TILScript rozšířen o atomický typ sloužící k reprezentaci textu. Tento typ je podrobněji popsán v sekci o rozšířeních TILScriptu, viz 3.3.2.

3.2.3 Generický typ *Any*

V Transparentní intenzionální logice není neobvyklé definovat typově polymorfní funkce. Zvykem je označovat předem neznámé typy řeckým písmenem α. Obdobně TILScript umožňuje definici typově polymorfních funkcí. Generické typy v TILScriptu značíme slovem `Any`, okamžitě následovaným indexem polymorfního typu. Index je libovolné číslo z rozsahu $\langle 0; 2^{32} - 1 \rangle$. Nenulové indexy nesmí začínat číslem 0.

Generické typy lze použít pouze v typech funkcí. Má-li více argumentů funkce stejný generický typ, tedy typ `Any` se stejným indexem, interpreter při procesu typové kontroly zajistí, že argumenty, na něž je funkce za běhu aplikována, jsou stejného typu.

3.2.4 Funkce

V matematice známe funkce jako jednoznačné zobrazení z množiny možných argumentů (definiční obor) do množiny možných obrazů (obor hodnot). V programovacích jazycích konstrukce nazývané funkcemi často nemusí být nejen jednoznačné (výstup nemusí odpovídat pouze argumentům), ale dokonce nemusí být ani zobrazením (nevrací žádnou hodnotu, v takovém případě většinou modifikují stav světa, jako příklad lze uvést funkce s návratovou hodnotou `void` v jazyce C).

V jazyce TILScript, obdobně jako v TIL, jsou funkce vždy zobrazením do určitého oboru hodnot. Díky parcialitě může být funkce degenerovaná, a v takovém případě nebude vracet hodnotu pro žádnou kombinaci argumentů. V takovém případě se však jedná o určitou formu chybového stavu, spíše než záměr, jak by tomu bylo v případě např. jazyka C. V čem se ovšem TILScript od Transparentní intenzionální logiky liší, je možnost modifikovat stav světa. TILScript je funkcionální jazyk, proto je na nejlepším uvážení uživatele, aby takovéto funkce nedefinoval, a jejich využívání omezil na nutné minimum. Příkladem funkcí s vedlejšími efekty (tedy modifikujících stav světa) můžou být například funkce `Print`, `Println` ze standardní knihovny, funkce pro zápis do databáze, apod. Arita funkce musí být vždy alespoň jedna.

Pro zápis typu funkce využíváme podobnou notaci jako v Transparentní intenzionální logice. Typy funkcí denotujeme kulatými závorkami. Uvnitř závorek uvedeme nejprve obor hodnot funkce, poté uvádíme postupně typy argumentů. Jediný rozdíl oproti TIL spočívá v nutnosti zapsat mezeru mezi názvy jednotlivých typů. Tedy ekvivalentem k typu $(o\nu\tau)$ by v TILScriptu byl typ `(Bool Int Real)`.

Funkce lze deklarovat uvedením názvu funkce následovaným lomítkem a jejím typem. Deklarujeme-li více funkcí stejného typu, můžeme uvést více názvů oddělených čárkami. Pro definici funkce byla přidána nová syntax popsaná v sekci 3.3.1.

```
Add, Sub, Mult, Div/(Int Int Int).
```

Listing 3.2: Deklarace funkcí

3.2.5 Literály

Názvosloví *literál* je přejato z jiných programovacích jazyků. V TILScriptu literály myslíme nefunkce, tedy členy množin tvořících bázi. Literály lze uvádět celá i reálná čísla, pravdivostní hodnoty, a také text (viz oddíl 3.3.2 věnující se typu `Text`). Individua pomocí symbolických hodnot, viz Symbolické hodnoty 3.2.14. Literály ovšem nesmíme zapomenout trivializovat.

3.2.6 Trivializace

Trivializace v TILScriptu slouží ke stejnému účelu jako v Transparentní intezionální logice. Narozdíl od TIL ovšem trivializaci denotujeme jednoduchým apostrofem namísto nuly zapsané jako levý horní index.

```
'1         -- Trivializace konstanty typu Int
'3.14159   -- Trivializace konstanty typu Real
'['+ '1 '2] -- Trivializace kompozice
```

Listing 3.3: Příklad trivializace.

3.2.7 Proměnné

Proměnné v TILScriptu opět slouží stejném účelu jako proměnné v TIL, tedy v-konstruují hodnotu v závislosti na valuaci v. Každé proměnné lze přiřadit hodnotu pouze jednou – pro volné proměnné při definici, pro λ-vázané proměnné při aplikaci funkce.

Volné proměnné lze deklarovat bez přiřazení valuace, ale také definovat a přiřadit jim konkrétní hodnotu. Proměnné deklarujeme, pokud nás nezajímá konkrétní valuace. Přiřazení hodnoty využijeme například v případě, kdy si chceme uložit výsledek drahé operace (např. čtení z databáze), nebo si třeba jen chceme zkrátit zápis dlouhé konstrukce, a chceme pracovat s konkrétní valuací v.

```
x -> Int.             -- Deklarace proměnné v-konstruující hodnotu typu Int
y, z -> Int.          -- Deklarace více proměnných najednou
pi -> Real := '3.1415. -- Definice proměnné pi aproximující hodnotu $\pi$
['* pi '2].           -- Využití proměnné pi jako konstituent konstrukce
long_varName123 -> Int.
```

Listing 3.4: Příklad využití proměnných

Název proměnné musí začínat malým písmenem. Je-li konstrukce, jejíž hodnotu přiřazujeme proměnné, nevlastní, a tedy nekonstruuje žádnou hodnotu, program skončí chybou. Vždy je třeba uvést typ hodnoty, kterou proměnná konstruuje.

3.2.8 Provedení

Provedení se sémantikou opět nijak neliší od svého ekvivalentu v Transparentní intenzionální logice. Syntax ovšem musela být kvůli praktičnosti upravena, a proto bylo upuštěno od pravých horních indexů. Provedení denotujeme `^1`. Pro dvojí provedení poté využíváme `^2`. Dřívejší verze TILScriptu definovaly i trojí až deváté provedení. Protože však trojí a vícenásobné provedení není v praxi skoro vůbec potřeba (dle Průvodce TIL taková potřeba nenastala), a protože limit devátého provedení byl poněkud arbitrární (proč ne například desetinásobné provedení), je tato práce konzervativní a drží se definice provedení z Průvodce.

Jelikož jsou všechny konstrukce standardně v módu provedení, není třeba provedení využívat příliš často. Hlavně tedy budeme používat dvojí provedení.

```
^1 x.
^2['GetComposition argument1 argument2].
```

Listing 3.5: Příklad využití provedení

3.2.9 Kompozice

Kompozice umožňuje aplikovat funkce na argumenty. Kompozice využívají stejnou syntax jako v TIL. Protože arita funkce musí být alespoň jedna, musí i kompozice obsahovat alespoň dvě podkonstrukce – funkci samotnou a alespoň jeden její argument. Počet argumentů, na něž funkci aplikujeme, musí odpovídat počtu argumentů funkce. Dále nesmí být žádný argument nevlastní (s výjimkou funkcí `If` a `IsNil`). V opačném případě k aplikaci funkce vůbec nedojde, neboť nemáme argumenty, na které bychom funkci aplikovali, a kompozice je tak nevlastní.

```
['* '2 '6].
```

Listing 3.6: Příklad využití kompozice

3.2.10 Uzávěry

Sémantika uzávěrů je opět stejná jako v Transparentní intenzionální logice, syntax ovšem byla upravena. Zatímco v TIL často vypouštíme hranaté závorky, v TILScriptu musíme závorky zapsat vždy. Řecké písmeno lambda nahrazuje v TILScriptu znak zpětného lomítka.

Zpětné lomítko následuje seznam argumentů funkce konstruované uzávěrem. V dřívějších verzí TILScriptu bylo možné typ argumentu v některých případech vynechat – existovala-li volná proměnná se stejným názvem jako λ-vázaná proměnná v uzávěru, a nebyl-li typ λ-vázané proměnné uveden explicitně, byl typ λ-vázané proměnné automaticky dedukován podle typu stejnojmenné volné proměnné. Tato práce však od této automatické dedukce upouští. Jelikož byly do TILScriptu přidány výrazy `import` umožňující importovat volné proměnné z jiného souboru, uživatel TILScriptu

by se mohl dostat do situace, kdy musí procházet několik importovaných souborů, jen aby zjistil typ λ-vázané proměnné. V případě, kdy máme více λ-vázaných proměnných, použijeme čárku pro jejich oddělení.

Za seznamem argumentů může nově následovat explicitní specifikace oboru hodnot konstruované funkce. Specifikaci oboru hodnot denotujeme znaky `->` které následuje název existujícího typu. Explicitní specifikace je nepovinná, může však sloužit k zdůraznění úmyslu uživatele, aby čtenář zdrojového kódu na první pohled znal typ funkce. Dále explicitní specifikace typu může pomoct při typové kontrole.

Nakonec je třeba uvést konstrukci, nad kterou provádíme abstrakci.

```
[\x: Int -> Int ['+ x '2]].
[\x: Int, y: Int -> Int ['+ x y]].
[\x: Int, y: Int ['+ x y]].

[[\x: Int, y: Int -> Int ['+ x y]] '2 '3].
```

Listing 3.7: Příklad využití uzávěrů

3.2.11 Zkrácený zápis typu intenzí a extenzionalizace

V TIL často využíváme zkráceného zápisu jak pro typy intenzí, tak pro jejich extenzionalizaci. Pro hodnoty typu α závislé na světamihu zkracujeme zápis $((\alpha\tau)\omega)$ na $(\alpha_{\tau\omega})$. Obdobně pro extenzionalizaci intenze a využíváme zkráceného zápisu a_{wt} ekvivalentnímu konstrukci $[[aw]t]$, kde a je konstrukce konstruující funkci (většinou se jedná o trivializaci), a $w \rightarrow_v \omega$, $t \rightarrow_v \tau$. Jedná se však pouze o notační zkratku – o dohodu, ne součást TIL.

Ekvivalentní zkratky můžeme využívat také v TILScriptu. Chceme-li specifikovat typ intenze, můžeme využít notační zkratku `@tw`. Zkratku `@wt` využijeme k extenzionalizaci intenze. Zkratka `@wt` vždy extenzionalizuje za využití proměnných `w` a `t`. Proměnná `w -> World` je součástí standardní knihovny, proměnnou `t -> Time` musí uživatel definovat sám. Zkrácenou notaci nelze využít s proměnnými jiného názvu.

```
(Bool Indiv)@tw -- intenze typu (((Bool Indiv) Time) World)
['Rektor@wt 'VSB] -- extenzionalizace funkce Rektor/(((Indiv Indiv) Time) World)
                  -- a nasledna aplikace na argument VSB.
```

Listing 3.8: Příklad využití zkrácené notace

3.2.12 Seznamy

Seznamy se v Transparentní intenzionální logice příliš neobjevují. Při strojové analýze a zpracování je však vhodné mít k dispozici způsob k vyjádření kolekce dat, proto TILScript seznamy obsahuje.

Seznam představuje homogenní seřazenou kolekci potenciálně neomezené délky. Seznamy můžou obsahovat duplicity.

V této práci jsou všechny seznamy neměnné. Seznamy jsou definovány induktivně. Seznam (list) je buď prázdný list, nebo *cons cell* skládající se z hlavičky (známé jako head nebo CAR listu z jiných jazyků) – prvního prvku v seznamu, a z podseznamu reprezentujícího zbytek listu (tail, CDR). Z definice tedy vyplývá, že vytvoření nového seznamu vložením prvku na začátek lze provést v konstantním čase. Vytvoření nového seznamu přidáním prvku na konec jiného listu naopak bude lineární vzhledem k velikosti původního kolekce. Tato implementace listu je volena zejména proto, že umožňuje snadnou iteraci pomocí rekurzivních funkcí. Stejnou implementaci listů využíval již jazyk LISP.

Typ seznamu denotujeme slovem `List` následovaným kulatými závorkami, v nichž je uveden typ prvku ukládaného v seznamu.

Standardní knihovna obsahuje řadu funkcí pro práci s listy. Tyto funkce jsou v této kapitole pouze nastíněny při ilustraci využití seznamů, podrobněji jsou však zdokumentovány v kapitole popisující standardní knihovnu.

Dále interpreter implementuje syntaktický cukr pro jednodušší vytváření listů. Pro vytvoření seznamu stačí v kompozici aplikovat funkci `ListOf` na nenulový, avšak shora neomezený počet argumentů. Během parsování bude tato kompozice korektně přepsána na sérii kompozic využívajících funkci `Cons` k vytvoření `cons cells`. K žádné aplikaci funkce na libovolný počet argumentů tedy nedochází. Při vytváření seznamu pomocí `ListOf` musíme dodat alespoň jeden prvek listu, aby bylo možné správně dedukovat typ prvků ukládaných v seznamu. `ListOf` je pouze syntaktický cukr překládaný při procesu parsování. Pokud bychom chtěli generovat konstrukce konstruující listy dynamicky za běhu programu, museli bychom zřetězit aplikace `Cons`.

```
['ListOf '1 '2 '3].                -- vytvoreni listu
['Cons '1 ['Cons '2 ['ListOfOne '3]]]. -- ekvivalent predchoziho radku a priklad
   prepisu ListOf
['Head ['ListOf '1 '2 '3]]         -- 1
['Tail ['ListOf '1 '2 '3]]         -- ['ListOf '2 '3]
```

Listing 3.9: Příklad využití seznamů

3.2.13 N-tice

N-tice doznály oproti předchozím verzím TILScriptu změny. Dříve byly n-tice homogenní kolekce nespecifikované, avšak konečné délky. Roli homogenní kolekce však již zastává List, a v TILScriptu neexistoval způsob, jak seskupit více hodnot jiného typu dohromady.

V matematice jsou n-tice například prvky kartézského součinu. N-tice lze definovat více způsoby, vždy jsou však uspořádané, konečné a můžou být heterogenní (například provádíme-li kartézský

součin nad neidentickými množinami). Obdobnou roli plní n-tice i v TILScriptu. Délka n-tice je pevně daná, stejně jako typy hodnot v n-tici. Celý typ n-tice je tedy dán uspořádaným výčtem všech typů všech hodnot obsažených v této n-tici. Syntaktický zápis typu n-tice je podobný typu seznamu. Typ n-tice denotujeme slovem `Tuple`, následovaným kulatými závorkami. V závorkách ovšem uvádíme výčet typů. Jednotlivé typy oddělujeme čárkami. Délka n-tice je určena počtem uvedených typů.

Práce s n-ticemi je do jisté míry podobná práci s listy. N-tici vytvoříme aplikací variadické funkce `MkTuple`. Stejně jako u funkce `ListOf` se však jedná pouze o syntaktický cukr. Funkce `MkTuple` skutečně existuje, je to ale funkce binární, nikoliv variadická, a vytváří dvojice z dodaných argumentů. Pokud ovšem funkci `MkTuple` dodáme více než dva argumenty, parser aplikaci `MkTuple` rozepíše na jednu aplikaci `MkTuple` a následné řetězení funkce `PrependToTuple`. Funkce pro práci s n-ticemi jsou podrobněji popsány v uživatelské dokumentaci.

```
['MkTuple '1 '2.0 'True].                -- vytvoreni n-tice
['PrependToTuple '1 ['MkTuple '2.0 'True]]. -- ekvivalent predchoziho radku

x -> Tuple(Int, Real, Int) := ['MkTuple '1 '3.14159 '10].
```

Listing 3.10: Příklad využití n-tic

3.2.14 Symbolické hodnoty

Název *symbolické hodnoty* je v kontextu TILScriptu nový, jedná se však pouze o praktickou implementaci entit, které již v TILScriptu existovaly. Entitu lze specifikovat uvedením jejího názvu následovaného lomítkem a typem entity. Tímto byl názvu přiřazen typ, ale ne konkrétní hodnota. Název *symbolické hodnoty* je opět inspirován jazykem Lisp, kde podobný koncept symbolů, tedy jmen bez hodnoty, již dlouhou dobu existuje.

Symbolické hodnoty využíváme, když potřebujeme o daném objektu referovat jménem, a ne hodnotou. Jako příklad lze uvést například individua. Individua nemají žádnou inherentní hodnotu. Objekt typu ι je pouze identifikátorem, unikátním jménem. Vlastnosti jsou individuím přiřazeny intenzemi a závisí na čase a stavu světa, případně se jedná o triviální vlastnosti bez vypovídací hodnoty. Dále můžeme zmínit například číslo π. Číslo π sice bezpochyby hodnotu má, jedná se o reálné číslo. Jelikož je však π číslo iracionální, tedy číslo s nekonečným desetinným rozvojem, bohužel jej nemůžeme reprezentovat v počítači. V praktické implementaci se tak musíme spokojit pouze s aproximací čísla π, nebo právě se symbolickou reprezentací.

Symbolické hodnoty nelze provést. Chceme-li je zmínit, musíme využít trivializaci. Dále nad nimi nelze provádět všechny operace, které jdou provádět s nesymbolickými konkrétními hodnotami. Například jak bychom mohli příčíst k jinému číslu číslo π, když neznáme přesnou hodnotu čísla π? Kde to však dává smysl, můžeme naprogramovat dodatečnou podporu pro symbolické hod-

noty. Funkce `Sin`, `Cos` v matematické knihovně mají zabudovanou kontrolu, zda zda obdržely jako svůj argument symbol `Pi`/`Real`. Pokud ano, funkce vrátí korektní hodnotu. Obdobně by například výsledkem přirozeného logaritmu mohlo být číslo 1 při aplikaci na symbol `E`/`Real`.

```
Pi/Real.
['Sin 'Pi]. -- 0
['Cos 'Pi]. -- -1
['+ '1 'Pi]. -- Nil - vysledek existuje, na pocitaci jej vsak nelze spocitat
```

Listing 3.11: Příklad využití symbolických hodnot

3.3 Rozšíření TILScriptu

3.3.1 Definice funkcí

3.3.2 Typ *Text*

3.3.3 Hodnota *Nil*

3.3.4 Struktury