Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Přesné výpočty s reálnými čísly

2021

Ondřej Slavík

Vedoucí práce:
doc. RNDr. Michal Krupka, Ph.D.

Studijní obor:
Informatika, prezenční forma

Bibliografické údaje

Autor:	Ondřej Slavík
Název práce:	Přesné výpočty s reálnými čísly
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2021
Studijní obor:	Informatika, prezenční forma
Vedoucí práce:	doc. RNDr. Michal Krupka, Ph.D.
Počet stran:	69
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Ondřej Slavík
Title:	Precise computation of real numbers
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2021
Study field:	Computer Science, full-time form
Supervisor:	doc. RNDr. Michal Krupka, Ph.D.
Page count:	69
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Fenomén vyčíslitelnosti reálných čísel provází každého informatika, který se snaží používat počítač k počítání. Jakmile se totiž musíme spolehnout na výpočty s čísly uloženými jako hodnoty, narážíme na limity přesnosti a rozsahu takto reprezentovaných čísel. Řešením není zpřesňování pomocí vyšší dotace paměťového prostoru (např. binary32 → binary64) a související změna architektury systému, nýbrž fundamentální změna v přístupu k vyčíslení reálných čísel. Tato práce dává návod, jak takovýto přístup přijmout, a přináší knihovnu, která umožňuje základní výpočty a vyčíslení reálných čísel.

Synopsis

Every computer scientist who tries to use a computer to compute encounters the phenomenon of real numbers' computability. Once we have to rely on calculations with numbers stored as values, we come across limits of precision and range of thus represented numbers. The solution is not to refine using a higher memory space allocation (eg binary32 → binary64) and the related change in system architecture, but fundamental change in the approach to computation of real numbers. This work gives direction of how to adopt such an approach, and brings a library that allows the basic calculations and enumerations of real numbers.

Klíčová slova: reálná čísla, funkce, Lisp, líné vyhodnocování, libovolná přesnost, rekurzivní čísla

Keywords: real numbers, functions, Lisp, lazy evaluation, arbitrary precision, recursive numbers

Mockrát děkuji doc. RNDr. Michalu Krupkovi, Ph.D. za vedení práce a rodině za podporu.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam definic

Seznam tabulek

Seznam faktů

Seznam vět a lemmat

Seznam zdrojových kódů

Seznam testů

Seznam zbytku

0 Úvod

Funkcionální programování umožňuje důsledně popsat matematický svět, což bude v této práci ukázáno na příkladu výpočtů reálných čísel. Jako funkcionální jazyk byl zvolen Common Lisp (Lisp) pro jeho syntaktickou jednoduchost, výstižnost a lenost. Každý Lispový kód následuje vždy za matematickým výrazem a ve stejném znění ho převádí. Lisp představuje nástroj, pomocí něhož realizujeme matematické výpočty, hlavním těžištěm poznání je ale vybudovaná matematická teorie, která není specifická pro konkrétní programovací jazyk.

V první části je teoreticky popsáno, co čísla jsou a jak se rozvrstvují podle vlastností na obory. Také je zde ukázáno, jak se s čísly operuje a je představen pojem zobrazení a jeho dva důležité typy – funkce a posloupnost. Je zde zavedena stěžejní představa, co znamená přesná reprezentace čísla pro jeho různé podoby a představeno několik existujících knihoven.

Ve druhé části práce je popsána konstrukce knihovny `tnums` přesně vyčíslující reálná čísla. Je využita existence racionálních čísel v Lispu a přidáno Ludolfovo číslo. Zavedená čísla jsou poté kombinována operacemi a měněna svými funkcemi, v návaznosti na to je zavedeno Eulerovo číslo.

V závěrečné části je uvedeno praktické užití řešení a vymezen pojem uživatelské funkce. Je zde ukázáno, jak lze takové funkce přidávat. Také doplníme poslední konstantu, a to Zlatý řez. V poslední kapitole jsou představeny nápady na urychlení a rozšíření práce knihovny `tnums` a její problémy.

Značení

$\square$	konec důkazu
$\blacksquare$	konec poznámky
modré pozadí	obrázek
nachové pozadí	tabulka
(a, b)	otevřený interval mezi a a b
$[a, b]$	uzavřený interval mezi a a b
$\rightarrow$	logická implikace nebo „do“
$\leftrightarrow$	logická ekvivalence
$\Leftrightarrow$	„právě tehdy, když“
$\cup$	množinové sjednocení
$\cap$	množinový průnik
$\lvert a \rvert$	absolutní hodnota čísla a
$:=$	přiřazení
$\frac{d}{dx}f(x_0)$	derivace funkce f v bodě x_0; je-li jasná proměnná, pak jen $f^{'}(x)$

Část I
Teorie

V teoretické části je představena axiomatická teorie množin a naznačeno, jak se z ní vytváří čísla. Je demonstrováno, že čísla jsou množinami. Poté jsou stručně představeny matematické operace a matematické funkce. Dále vyplyne, že jakékoli číslo lze vyjádřit jako funkci a že funkce je také množina. Závěr teoretické části je zaměřen na problém uložení čísla v paměti počítače, která je fyzicky konečná. Nejprve je rozebrán v čistě teoretické rovině a poté je diskutováno řešení v podobě reálně existujících knihoven.

1 Čísla

Číslo je matematický objekt, který na intuitivní úrovni všichni chápeme. To znamená, že každý dokáže o libovolném objektu říci, zda se jedná o číslo, nebo nikoli, a všichni se na tom shodneme. Je číslem 3? Je číslem 2.999...? Je číslem e?

Pokus o definici čísla by mohl vypadat třeba jako na české wikipedii:

Definice 1 (Číslo – naivní [1]). *Číslo je abstraktní entita užívaná pro vyjádření množství nebo pořadí.*

Uvedená definice přiřazuje číslům dvě funkce – kardinální a ordinální. Jinak o povaze čísel neříká mnoho. Víme ale, že je to abstraktní entita – to znamená, že ji nelze zapsat samu o sobě, ale je třeba použít nějaký symbol. Symbolem reprezentujícím číslo tři je 3, $\frac{6}{2}$ nebo třeba i 2.999.... Symbolický zápis čísla zřejmě není jednoznačný. 3 není to samé jako 2.999..., ale číslo 3 je to samé jako číslo 2.999.... Z toho plyne, že symbol s referencí na nějakou entitu a tato entita samotná se od sebe liší. Alenka takhle zjistila rozdíl mezi tím, jak se říká názvu písně, jaké je její jméno, jak se píseň jmenuje a co píseň opravdu je [2]. Známe-li rozdíl mezi koncepty čísla a symbolu, který ho reprezentuje, je možné využít pro reprezentaci čísel i jiné symboly, než číslice, jako jsou například písmena či složené matematické výrazy – například $\sum_{n\in\mathbb{N}} \frac{1}{n!} = e = \lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n$.

Číslo je tedy charakterizováno nikoli svým zápisem, ale svým obsahem. Tato vlastnost je společná s jinými matematickými objekty – s množinami. Množina je také dána svým obsahem, nikoli svým zápisem (této vlastnosti se říká princip extenzionality). Například $\{0, 1, 2, 3\} = \{n | n \in \mathbb{N} \wedge n \leq 3\}$ je stejná množina zapsaná dvěma různými způsoby. Symbol „=“ je tedy ekvivalence na hodnotách.

1.1 Přirozená čísla

Přirozená čísla jsou čísla, jejichž *kardinalita* určuje počet nedělitelných částí celku. Historicky zřejmě vznikla nejdříve, proto se nazývají přirozená (anglicky *natural* – přírodní). Jedná se o čísla nula, jedna, dva, atd. Běžnými symboly těchto čísel jsou 0, 1, 2, atd. Vlastností přirozených čísel je, že *každé* číslo n má následníka. Toho značíme $s(n)$ nebo $n + 1$ [3].

Hodnota přirozeného čísla je počet entit v nějakém souboru – například počet turů ve stádu. *Následník* takového čísla značí, kolik entit bude v souboru, pokud se nějaká nedělitelná část do souboru přidá – například se do stáda narodí tele.

I přirozená čísla jsou *množiny*. Konkrétně v tomto textu je zavedu v Zermelově-Fraenkelově (ZF) axiomatici teorii množin. Je snadné nahlédnout, že stačí zkonstruovat číslo nula a *zobrazení* s přiřazující každému číslu následníka. Poté budeme mít celou nekonečnou množinu přirozených čísel zkonstruovanou. Protože se jedná o výklad teorie množin, celý zbytek této podkapitoly je pouze velmi stručný výtah z [4].

Definice přirozených čísel je induktivní a stojí na jednoduché myšlence Johna von Neumanna, že „přirozené číslo je množina všech menších přirozených čísel“. Číslo nula je zde prázdná množina značená 0, $\emptyset$ nebo $\{\}$. Následník čísla je sjednocení čísla s množinou obsahující toto číslo, čili $s(n) = n \cup \{n\}$.

Několik nejmenších přirozených čísel vypadá tedy následovně.

Příklad 2 (Nejnižší přirozená čísla coby množiny).

$$0 = \emptyset \tag{1}$$

$$1 = s(0) = 0 \cup \{0\} = \emptyset \cup \{\emptyset\} = \{\emptyset\} \tag{2}$$

$$2 = s(1) = 1 \cup \{1\} = \{\emptyset\} \cup \{\{\emptyset\}\} = \{\emptyset, \{\emptyset\}\} \tag{3}$$

$$3 = s(2) = 2 \cup \{2\} = \{\emptyset, \{\emptyset\}\} \cup \{\{\emptyset, \{\emptyset\}\}\} = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\} \tag{4}$$

$$\begin{aligned} 4 = s(3) = 3 \cup \{3\} = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\} \cup \{\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}\} = \\ = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}\} \end{aligned} \tag{5}$$

Poznámka 3 (Provázání hodnot a podmnožin). Kardinalita v tomto pojetí znamená počet podmnožin, neboli číslo n má n podmnožin. Tím se provázaly dva zdánlivě cizí pojmy a sice *podmnožinovost* a *hodnota* čísla. V tomto kontextu pak není překvapivá podobnost srovnávacích symbolů $\subseteq$ a $\leq$. ■

Zermelova-Fraenkelova teorie stojí na následujících pěti axiomech a jednom axiomovém schématu (také uváděno jako 6 axiomů).

- Axiom extenzionality: $(\forall u)((u \in x \leftrightarrow u \in y) \rightarrow x = y)$
- Axiom fundovanosti: $(\forall a)(a \neq \emptyset \rightarrow (\exists x)(x \in a \wedge x \cap a = \emptyset))$
- Axiom sumy: $(\forall a)(\forall b)(\exists z)(\forall x)(x \in z \leftrightarrow (x = a \vee x = b))$
- Axiom potence: $(\forall a)(\exists z)(\forall x)(x \in z \leftrightarrow x \subset a)$
- Axiom nekonečna: $(\exists z)(\emptyset \in z \wedge (\forall x)(x \in z \rightarrow x \cup \{x\} \in z))$
- Schéma axiomů nahrazení: Je-li $\Psi(u, v)$ formule, která neobsahuje volné proměnné w a z, potom formule

$$\begin{aligned}(\forall u)(\forall v)(\forall w)((\Psi(u,v) \wedge \Psi(u,w)) \rightarrow v = w) \rightarrow \\ \rightarrow (\forall a)(\exists z)(\forall v)(v \in z \leftrightarrow (\exists u)(u \in a \wedge \Psi(u,v)))\end{aligned}$$

  je axiom nahrazení.

Z těchto axiomů je možné odvodit i (slabší) tvrzení, která se někdy nazývají axiomy a sice

- Axiom dvojice: $(\forall a)(\forall b)(\exists z)(\forall x)(x \in z \leftrightarrow (x = a \vee x = b))$
- Schéma axiomů vydělení: Je-li $\varphi(x)$ formule, která neobsahuje volnou proměnnou z, potom formule $(\forall a)(\exists z)(\forall x)(x \in z \leftrightarrow (x \in a \wedge \varphi(x)))$ je axiom vydělení.

Z axiomů ZF je pro naše zkoumání důležitý například axiom nekonečna – existuje alespoň jedna (nekonečná) množina. Za pomoci dalších axiomů poté konstruujeme další prvky, jako třeba prázdnou množinu – tu dostaneme pomocí axiomu vydělení s jakoukoli množinou a a formulí $\varphi(x) = x \neq x$ – prázdná množina je tedy $\emptyset = \{x | x \in a \wedge x \neq x\}$. Dále získáváme, že sjednocení množin je také množina, podobně jejich průnik.

Definice 4 (Induktivní množina). *Množina A je induktivní, pokud platí* $(\emptyset \in A) \wedge (\forall a \in A)((a \cup \{a\}) \in A)$.

Definice 5 (Přirozená čísla).

$$\mathbb{N} = \bigcap \{A | A \text{ je induktivní}\} \tag{6}$$

Množina přirozených čísel je nejmenší induktivní množina a je podmnožinou každé induktivní množiny. $\mathbb{N}$ je nejmenší nekonečný *kardinál* a také jediný *spočetný* kardinál.

Poznámka 6 (Značení podmnožin reálných čísel). Množina z v axiomu nekonečna je stejná množina jako $\mathbb{N}$. V teorii množin se značí ω, při zkoumání kardinality pak $\aleph_0$. Pro označení přirozených čísel bez nuly zaveďme horní index $+$ ($\mathbb{N}^+ = \mathbb{N} - \{0\}$). Podobně $\mathbb{R}^+ = \{x | (x \in \mathbb{R}) \wedge (x > 0)\}$. ■

Poznámka 7 (Nula je přirozená). Z definice přirozených čísel podle ZF přímo vyplývá, že součástí přirozených čísel je i číslo nula. Dle mého i prázdné stádo je stále stádem, byť bez turů. ■

1.2 Vyšší obory čísel

Vyšším oborem čísel jsou čísla celá, značená $\mathbb{Z}$, a jsou to všechna čísla, která mohou vzniknout libovolným odčítáním přirozených čísel. Jejich výčet je diskrétní: $\mathbb{Z} = \{0, 1, -1, 2, -2, \ldots\}$[5]. Zde „$-$“ je znaménko odčítání (snížení hodnoty prvního argumentu o hodnotu druhého).

Další číselný obor jsou čísla racionální, pro která se vžilo označení $\mathbb{Q}$ a racionálním číslem je číslo x, pokud jde zapsat jako y/z, kde $y \in \mathbb{Z}$ a $z \in (\mathbb{Z} \setminus \{0\})$ a „/“ je symbol operace dělení [6]. Množina racionálních čísel je spočetná, $\mathbb{Q} = \{0, 1, 1/2, -1, 1/3, -1/2, 2, 1/4, \ldots\}$.

Po zavedení racionálních čísel stále v budovaném číselném systému nemáme například délku úhlopříčky jednotkového čtverce – takovéto číslo značíme $\sqrt{2}$ – nebo třeba poměr obvodu kružnice k jejímu průměru – toto značíme π. Když do systému doplníme všechna tato čísla, získáváme tzv. číselnou osu (přímku) reprezentující čísla reálná, značená $\mathbb{R}$. Množině přidaných čísel říkáme iracionální (nejsou racionální) a značíme ji $\mathbb{I}$, $\mathbb{I} = \mathbb{R} \setminus \mathbb{Q}$ [7]. Ke každým dvěma reálným číslům r a ε lze najít racionální číslo q tak, aby

$$|r - q| \leq \varepsilon \text{ [8].} \tag{7}$$

V množině reálných čísel matematikové objevili ještě jemnější struktury, než je dělení na obory, které jsem teď představil. První jmenované iracionální číslo $\sqrt{2}$ je tzv. *algebraické*, protože může vzejít jako řešení z nějaké algebraické rovnice, např. $x^2 = 2$ nebo $(-x)^2 = 2$, zatímco druhé jmenované – tzv. *Ludolfovo číslo* (π) – takovouto vlastností nedisponuje, je *nealgebraické* (též *transcendentní*) [4]. Dále v oboru reálných čísel existují tzv. *rekurzivní čísla* (*recursive numbers, computable reals* – vizte kapitolu 2.4.2.5), která se dají vyčíslit v konečném čase. Pro reálné číslo r a dané ε existuje vyčíslitelná funkce (pro konečný vstup někdy skončí), jejímž výsledkem je racionální číslo q tak, že platí nerovnice (7). Ostatní čísla jsou *nerekurzivní* a protože Turingových strojů je spočetně mnoho, je nerekurzivních čísel nespočetně mnoho [9]. Přechod od racionálních čísel k rekurzivním, kterým se zabývá tato práce, je velmi složitý.

Vyššími obory jsou strukturovaná čísla – uvažujeme více číselných os a číslo má 2^n složek. Jedná se o komplexní čísla ($\mathbb{C}, n = 1$), kvaterniony ($\mathbb{H}, n = 2$), oktoniony ($\mathbb{O}, n = 3$) – těmto čtyřem (společně s reálnými čísly) oborům říkáme *normované algebry s dělením* (*normed division algebra*) [10]. Existují ještě vyšší obory (sedeniony – $\mathbb{S}, n = 4$ atd.), tato práce se nicméně dále zabývá pouze jednorozměrnými čísly a v dalším textu je „číslem“ myšleno číslo *reálné.*

Tabulka 1: Symboly operací s čísly

Operace	Značení
sčítání	$x + y$
odčítání	$x - y$
násobení	$x * y$, $x \cdot y$ nebo xy
dělení	x/y nebo $\frac{x}{y}$
umocňování	x^y nebo $x\hat{\ }y$
odmocňování	$\sqrt[y]{x}$

Tabulka zobrazuje zápis binárních operací s čísly x a y.

1.3 Operace s čísly

Již v kapitole 1.2 byly zmíněny 3 operace – *odčítání* (rozdíl), *dělení* (podíl) a *odmocňování* (odmocnina). K nim existují ještě operace opačné, a ty po řadě nazýváme *sčítání* (součet), *násobení* (součin) a *umocňování* (mocnina).

Binární operace se značí znaménkem mezi operandy. Například součet čísel x a y značíme $x+y$ atd. Základní značení operací je uvedeno v Tabulce 1. Některým binárním operacím, které mají jako první operand neutrální prvek (při operaci s číslem je výsledkem toto číslo), budeme říkat unární. Unárními operacemi je číslo opačné (k číslu x je opak číslo $0 - x$ a značíme ho $-x$) a převrácené číslo (k číslu $x \neq 0$ je číslen převráceným číslo $1/x$ a značíme ho též $/x$ nebo x^{-1}).

Operacím $\langle +, - \rangle$ říkáme aditivní, $\langle *, / \rangle$ multiplikativní a $\langle \hat{\ }, \sqrt{\ } \rangle$ mocninné. Operace v uvedených dvojicích jsou příbuzné, příslušné vztahy nazveme „horizontální". Mezi operacemi však existují též vztahy „vertikální". Platí

$$x * n = \underbrace{x + x + \ldots + x}_{n\text{-krát}} \text{ a také } x^n = \underbrace{x * x * \ldots * x}_{n\text{-krát}}. \tag{8}$$

Máme tedy návod, jak teoreticky vytvořit více operací.

Další operace je daná předpisem

$$x\#n = \underbrace{x\hat{\ }x\hat{\ }\ldots\hat{\ }x}_{n\text{-krát}} \tag{9}$$

a nazýváme ji tetrace [11].

Pro operace vyšší arity je pak zvykem používat prefixovou notaci jednoho velkého znaménka – součet $a_0 + a_1 + a_2$ je pak zapsán jako $+(a_0, a_1, a_2)$ nebo též $\sum_{i=0}^{2}(a_i)$. Součin $a_0 * a_1 * a_2$ pak $*(a_0, a_1, a_2)$ nebo $\prod_{i=0}^{2}(a_i)$.

1.4 Funkce čísel

Další manipulace s čísly jsou tzv. funkce, které si představujeme jako nekonečnou tabulku o dvou sloupcích, které představují výstupní a výstupní hodnoty (v Tabulce 2 jsou uvedeny některé hodnoty pro funkci sinus). V informatice též

Tabulka 2: Nekonečná vstupně/výstupní tabulka funkce sinus

Vstup	Výstup
0	0
1	0.8414709848078965 . . .
2	0.9092974268256817 . . .
3	0.1411200080598672 . . .
4	−0.7568024953079282 . . .
⋮	

Tabulka obsahuje v prvním sloupci vstupní hodnoty funkce sinus a ve druhém hodnoty výstupní. Tabulka je nekonečná ve vertikálním směru, v horizontálním jsou pouze dva sloupce – pro argument a vrácenou hodnotu.

mluvíme o argumentu funkce a návratové hodnotě. Příklady funkcí jsou funkce *goniometrické*, funkce *exponenciální* či *logaritmus*.

Funkce jsou velmi užitečným a potřebným nástrojem takřka ve všech odvětvích matematiky a jsou často zdrojem *iracionálních* (*transcendentních*) čísel – například atan(1) dá za výsledek čtvrtinu *Ludolfova čísla* π.

Definice 8 (Uspořádaná n-tice [4]). *Jsou-li dány množiny $a_1, a_2, \dots a_m$, uspořádanou n-tici množin $a_1, a_2, \dots, a_n$ pro $n < m$ definujeme tak, že pro $n = 1$ položíme*

$$\langle a_1 \rangle = a_1 \tag{10}$$

a je-li již definována uspořádaná n-tice $\langle a_1, a_2, \dots, a_n \rangle$, položíme

$$\langle a_1, a_2, \dots a_{n+1} \rangle = \langle \langle a_1, a_2 \dots a_n \rangle, a_{n+1} \rangle. \tag{11}$$

Definice 9 (Kartézský součin [12] [13]). *Nechť $A_0, A_1, \dots, A_n$ jsou množiny. Symbolem $\bigtimes_{i=0}^{n} A_i$ či $A_0 \times A_1 \times \dots \times A_n$ označujeme množinu všech uspořádaných $(n+1)$-tic tvaru $\langle a_0, a_1, \dots a_n \rangle$, přičemž $(a_0 \in A_0) \wedge (a_1 \in A_1) \wedge \dots \wedge (a_n \in A_n)$, neboli*

$$A_0 \times A_1 \times \dots \times A_n = \{ \langle a_0, a_1, \dots, a_n \rangle | (\forall i \in \mathbb{N}, i \leq n)(a_i \in A_i) \} \tag{12}$$

a tuto množinu nazýváme kartézským součinem množin $A_0, A_1, \dots A_n$.

Poznámka 10 (Množinovost kartézského součinu). Že je kartézský součin (KS) množina jsem zavedl definitoricky, jako to udělaly autorky v [12], nemusí to být úplně jasné, je to ovšem dokazatelné. V [4] se kartézský součin zavádí jako třída (soubor množin, který množina být nemusí – například třída všech množin množinou není) a poté se pomocí axiomu potence jeho množinovost dokazuje. ■

Definice 11 (Relace [12]). *Relace mezi množinami A a B je libovolná podmnožina $\mathcal{R}$ kartézského součinu $A \times B$. Je-li $A = B$, mluvíme o relaci na A.*

Náleží-li dvojice $\langle a, b\rangle$ relaci $\mathcal{R}$, t.j. $\langle a, b\rangle \in \mathcal{R}$, říkáme, že a a b jsou v relaci $\mathcal{R}$, a zapisujeme též $a\mathcal{R}b$.

Relaci $f \subseteq A \times B$ nazveme zobrazením množiny A do množiny B, jestliže platí, že ke každému prvku $x \in A$ existuje právě jeden prvek $y \in B$ takový, že $\langle x, y\rangle \in f$.

Je-li relace $f \subseteq A \times B$ zobrazení, pak skutečnost, že $\langle x, y\rangle \in f$ zapisujemme ve tvaru $y = f(x)$. Rovněž používáme zápis $f : A \to B$, což znamená, že f je zobrazení A do B, x nazýváme *nezávisle proměnnou* a y *závisle proměnnou* [14].

Definice 12 (Reálná posloupnost [12]). *Zobrazení množiny $\mathbb{N}$ do množiny $\mathbb{R}$ nazýváme reálná posloupnost.*

Místo obecného značení $a : \mathbb{N} \to \mathbb{R}$ pro zobrazení se vžilo pro posloupnost značení $\{a_n\}_{n\in\mathbb{N}}$ nebo $\{a_n\}_{n=0}^{\infty}$. Obraz bodu n se značí a_n a říkáme mu také n-tý člen posloupnosti a [12].

Definice 13 (Nekonečná číselná řada, divergence a konvergence [15]). *Nechť $\{a\}_{n\in\mathbb{N}}$ je reálná posloupnost. Symbol $\sum_{n\in\mathbb{N}} a_n$ nebo $a_0 + a_1 + a_2 + \ldots$ nazýváme nekonečnou číselnou řadou.*

Posloupnost $\{s_n^a\}_{n\in\mathbb{N}}$, kde $s_m^a = \sum_{i=0}^{m} a_i$ nazýváme posloupnost částečných součtů řady $\sum_{n\in\mathbb{N}} a_n$.

Existuje-li vlastní limita $\lim_{n\to\infty} s_n^a = s$, řekneme, že řada $\sum_{n\in\mathbb{N}} a_n$ konverguje a má součet s.

Neexistuje-li vlastní limita $\lim_{n\to\infty} s_n^a$, řekneme, že řada $\sum_{n\in\mathbb{N}}$ diverguje. Pokud limita $\lim_{n\to\infty} s_n^a$ neexistuje, říkáme, že řada osciluje.

Pokud je $\lim_{n\to\infty} s_n^a = \infty$, pak říkáme, že řada diverguje k ∞. Pokud je $\lim_{n\to\infty} s_n^a = -\infty$, pak říkáme, že řada diverguje k $-\infty$.

Nechť $\sum_{n\in\mathbb{N}} a_n$ je konvergentní řada. Její součet s lze psát ve tvaru $s = s_n^a + R_n^a$, kde $s_n^a = \sum_{i=0}^{n} a_i$ je n-tý částečný součet řady $\sum_{n\in\mathbb{N}} a_n$ a $R_n^a = \sum_{i=n+1}^{\infty} a_i$ se nazývá zbytek po n-tém členu řady $\sum_{n\in\mathbb{N}} a_n$.

Definice 14 (Reálná funkce reálné proměnné [14]). *Buď $M \subseteq R$. Zobrazení $f : M \to \mathbb{R}$ nazýváme reálnou funkcí reálné proměnné nebo stručně funkcí jedné proměnné.*

Množina M se nazývá definiční obor funkce f a značí se $D(f)$, množina $H(f) = \{f(x)|x \in M\}$ se nazývá obor hodnot funkce f.

Příklad 15 (Sinus jako zobrazení). Funkce $y = sin(x)$ je definována pro všechna $x \in \mathbb{R}$ a její obor hodnot je interval $[-1, 1]$, jedná se tedy o reálnou funkci reálné proměnné. Pokud budeme hledět jen na obrazy $sin(x), x \in \mathbb{N}$ (jako v Tabulce 2), jedná se o reálnou posloupnost.

Zobrazení $f : A \to B$ je definováno tak, že A i B jsou množiny. V poznámce 10 je řečeno, že je množinou i kartézský součin množin. Tedy lze definovat též zobrazení $\mathbb{R} \times \mathbb{I} \to \mathbb{Q} \times \mathbb{Z} \times \mathbb{N}$ atp., aniž bychom museli definici zobrazení měnit.

Mezi operacemi a funkcemi existují určité podobnosti. Sčítání lze vyjádřit jako funkci: $+(a, b) = a + b$. Matematickou operaci pak chápeme jako speciální případ funkce, n-ární reálná operace je funkce $\bigtimes_{i=1}^{n} \mathbb{R} \to \mathbb{R}$.

Funkce $\{\emptyset\} \to \mathbb{R}$ je konstanta, neboť podle definice zobrazení, pokud jsou v relaci $\langle \emptyset, x\rangle$ a $\langle \emptyset, y\rangle$, pak $x = y$. Funkce se vždy zobrazuje na stejné číslo, a proto se jedná o konstantu. Z toho plyne, že funkcemi můžeme modelovat jakákoli čísla i představenou manipulaci s nimi. Víme též, že funkce je množina.

Definice 16 (Mocninná řada [15]). *Buď $\{a_n\}_{n\in\mathbb{N}}$ posloupnost reálných čísel, x_0 libovolné reálné číslo. Mocninnou řadou se středem v bodě x_0 a koeficienty a_n rozumíme řadu ve tvaru*

$$a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + \ldots + a_n(x - x_0)^n + \ldots = \sum_{n\in\mathbb{N}} a_n(x - x_0)^n. \quad (13)$$

Definice 17 (Taylorova řada [15]). *Nechť funkce f má v bodě x_0 derivace všech řádů. Mocninnou řadu ve tvaru $\sum_{n\in\mathbb{N}} \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n$ nazýváme Taylorovou řadou funkce f v bodě x_0.*

Je-li $x_0 = 0$, mluvíme o Maclaurinově řadě funkce f a je ve tvaru $\sum_{n\in\mathbb{N}} \frac{f^{(n)}(0)}{n!}x^n$.

Zbytku po n-tém členu Taylorovy řady říkáme n-tý Taylorův zbytek a značíme ho $R_n^{f,a}(x)$.

Definice 18 (Geometrická řada [15]). *Řadu nazýváme geometrickou, pokud je ve tvaru*

$$a + a * q + a * q^2 + a * q^3 + \ldots = \sum_{i\in\mathbb{N}} aq^i. \quad (14)$$

Fakt 19 (Částečný součet a součet geometrické řady [15]). *Pro geometrickou řadu ve tvaru $\sum_{i\in\mathbb{N}} b_i, b_i = aq^i, |q| < 1$ platí*

$$s_n^b = \sum_{i=0}^{n} aq^i = a\frac{1 - q^{n+1}}{1 - q}. \quad (15)$$

Pro geometrickou řadu ve tvaru $\sum_{i\in\mathbb{N}} aq^i, |q| < 1$ platí

$$\sum_{i\in\mathbb{N}} aq^i = \frac{a}{1 - q}. \quad (16)$$

Fakt 20 (Zbytek geometrické řady). *Pro n-tý zbytek geometrické řady $\sum_{i\in\mathbb{N}} b_i$, $b_i = aq^i$ platí*

$$R_n^b = \frac{aq^{n+1}}{1 - q}. \quad (17)$$

2 Čísla v počítači

Standardně jsou čísla v počítači uložena jako sled bitů v nějaké reprezentaci vyjadřující hodnotu. Tímto způsobem lze reprezentovat číslo jen s danou přeností a rozsahem, což je však v řadě případů dostačující. Čísla lze reprezentovat i naprosto přesně – pomocí struktur je reprezentujícími a s nimi manipulujícími. I tyto samozřejmě musí být uloženy v paměti, v tomto případě se však nejedná o uložení hodnoty čísla, nýbrž se ukládá abstrakce, která číslo na požádání umí vygenerovat. Tento přístup nazýváme líným. Nejprve bude popsáno ukládání čísel jako hodnot a zřetel bude brán na přesnost takto uložených čísel. Poté navrhneme struktury pro přesná čísla a nakonec budou ukázány existující knihovny.

2.1 Čísla uložená jako hodnoty

Paměť počítače většinou pracuje s tzv. *bity*, paměťovými buňkami nabývajícími dvou rozlišitelných hodnot [16]. Ke kódování čísel se tedy používá výhradně dvojková soustava. Celá následující podkapitola pojednávající o uložení čísel jejich hodnotami je převzata z [17].

2.1.1 Vážený poziční kód

Číslo v soustavě o základu b lze dešifrovat následovně:

$$(\dots a_2 a_1 a_0 . a_{-1} a_{-2} \dots)_b = \dots + a_2 b^2 + a_1 b + a_0 + \frac{a_{-1}}{b} + \frac{a_{-2}}{b^2} + \dots \tag{18}$$

kde čísla a_n nazýváme číslice a symbol „.“ řádovou tečkou – speciálně pak tečkou *desetinnou* ($b = 10$) nebo *dvojkovou* ($b = 2$).

Příklad 21 (Převod z váženého pozičního kódu).

$$(101010.101)_2 = (42.625)_{10} \tag{19}$$

2.1.2 Záporná čísla

V případě zápisu čísla váženým pozičním kódem lze strukturu pokrývající i záporná čísla přímočaře vytvořit přidáním příznaku (rozšířit jeho reprezentaci o jeden bit), zda se jedná o číslo kladné či nikoliv. V této reprezentaci lze vyjádřit zápornou i kladnou nulu. Tomutu kódování se říká kód *velikostí a znaménkem (signed-magnitude)*.

Jinou možností je záporná čísla vnímat jako opak čísel kladných i na úrovni reprezentace, čili záporné číslo opačné ke kladnému v binární reprezentaci vypadá jako negace bitů daného kladného čísla. Opět zde vyvstává problém se zápornou nulou. Tomutu kódování se říká *jedničkový doplněk (ones' complement)*.

Když od všech záporných čísel v jedničkovém doplňku odečteme číslo jedna, rozprostřeme čísla efektivně, tj. odpadne problém s dvojí reprezentací nuly. Tomutu kódování se říká *dvojkový doplněk (two's complement)*.

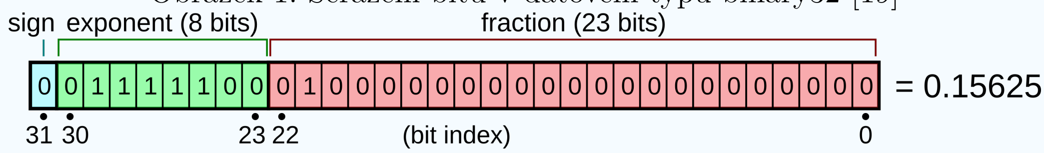
2.1.3 Plovoucí řádová tečka

U váženého pozičního kódu známe přesně pozici řádové tečky – je mezi číslicemi a_0 a a_{-1}. Alternativním přístupem je kódování s *plovoucí* řádovou tečkou.

Plovoucí číslo je dáno uspořádanou dvojicí $\langle e, f \rangle = f * b^{e-q}$, kde e nazýváme *exponent* a f *zlomkovou částí (fraction)*. Čísla b a q jsou konstanty dané použitým typem, b se nazývá *základ* a q *přesah.*

Příklad 22 (Plovoucí číslo – jednoduchá přesnost). Číslo s jednoduchou přesností (*single* dle IEEE 754-1985, *binary32* dle IEEE 754-2008) je uloženo v paměti jako 32b struktura. První bit je příznak znaménka, dalších 8 bitů je pro exponent a dalších 23 bitů pro fraction (vizte Obrázek 1). Konstanty jsou ohodnoceny následovně: $q = 127$, $b = 2$. V *C*, *C++*, *C#* nebo *Javě* se tento číselný datový typ nazývá *float*, v *Haskellu Float* a v *Lispu* pak *single-float* [18].

Obrázek 1: Seřazení bitů v datovém typu binary32 [19]



32-bitový datový typ pro čísla s plovoucí řádovou tečkou je dle standardu uložen v konfiguraci $\langle s, e, f \rangle = \langle 1, 8, 23 \rangle$.

2.2 Přesná reprezentace čísel jako hodnot

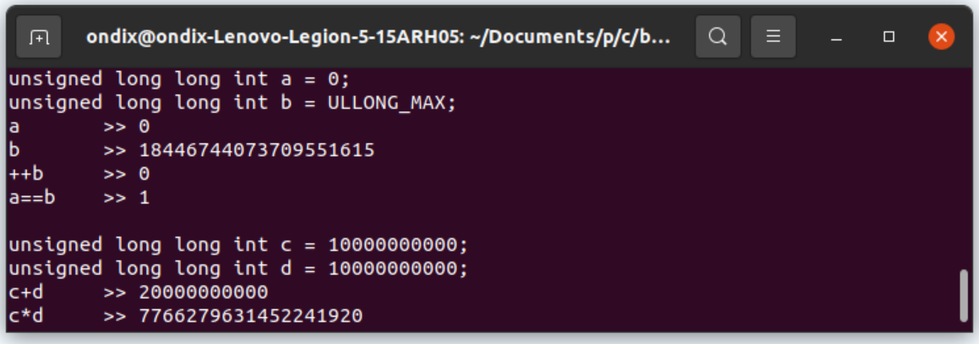
Nyní se podívejme, jaká čísla uložená jako hodnoty považujeme za přesná. Projdeme opět všechny obory jako v první kapitole a poprvé propojíme matematickou teorii s informatickou realitou. Jako modelový jazyk nám poslouží C.

Všechny ukázky v této kapitole představují reálné chování představených datových typů na konkrétní AMD64 architektuře. Nejde teď tedy o žádnou teorii a jsou jen prezentovány reálné limity. Na 128-bitové architektuře mohou být tyto limity jiné a typy použitelnější. Nicméně tato práce směřuje k přesným výpočtům rekurzivních čísel bez ohledu na architekturu.

2.2.1 Přirozená čísla

Přirozená čísla jsou uzavřena na operace sčítání a násobení. V počítači se jako hodnoty ukládají pomocí váženého pozičního kódu a tento má horní limit, jak velké číslo lze na dané architektuře uložit. Takto uložená přirozená čísla ovšem nejsou na operace uzavřena, protože může dojít k přetečení – číslo ukládané se liší od čísla uloženého. Příklad tohoto chování vydíme na Obrázku 2.

Obrázek 2: Přirozená čísla v jazyce C



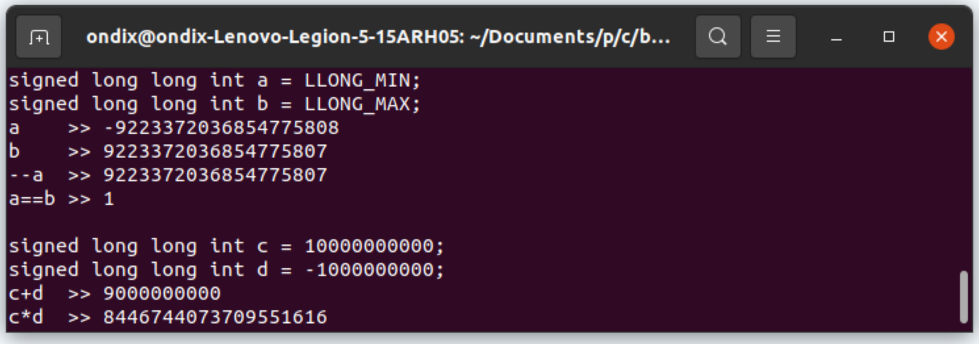
Největším datovým typem, který pracuje s přirozenými čísly je v jazyce C typ `unsigned long long int` a na 64-bitové architektuře umí uchovat čísla v intervalu $[0, 2^{64}-1]$. Na příkladu vidíme přetečení u inkrementace i násobení.

Pokud ale ukládáme přirozené číslo v intervalu, ve kterém jej zvládne uložit datový typ jako hodnotu, můžeme ho považovat za přesné.

2.2.2 Celá čísla

Celá čísla jsou uzavřena na sčítání, odčítání a násobení. V paměti počítače se pak musí používat kódování záporných čísel jako hodnot, často jde o dvojkový komplement. Opět zde vyvstává problém s limity jakéhokoli hodnotového datového typu, totiž že vyjadřitelná čísla jsou ohraničená a hrozí přetečení (a i podtečení). Implementace celého čísla jako hodnoty tedy není na operace uzavřená (Obr. 3).

Obrázek 3: Celá čísla v jazyce C



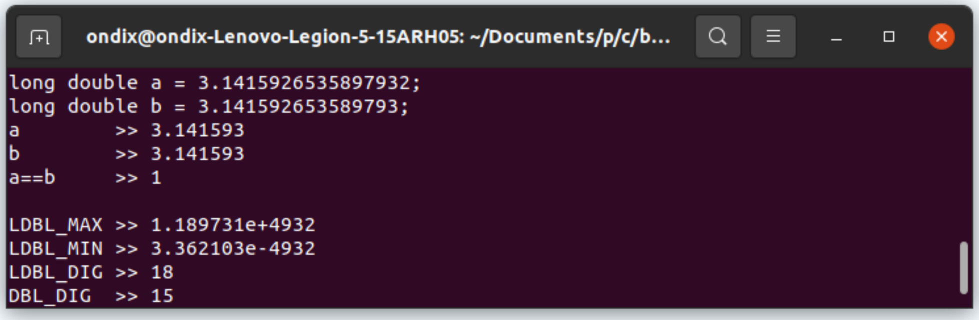
Nejširší datový typ jazyka C, který umí uložit celá čísla je `signed long long int`. Ten na 64-bitové architektuře zvládne uložit čísla z intervalu $[-2^{63}, 2^{63}-1]$. Na příkladu vidíme, že není uzavřen vůči operacím a že dochází k podtékání.

Pakliže ukládáme celé číslo jako hodnotu, která se vejde do datového typu bez přetečení nebo podtečení, lze takto vyjádřené číslo považovat za přesné.

2.2.3 Racionální čísla

Racionální číslo se v jazyce C ukládá jako číslo s plovoucí řádovou tečkou. S největší přesností se ukládá datový typ `long double`. Není specifikováno, jak má být přesný, pouze že má být minimálně tak přesný jako `double` [20]. Čísla, která jsou různá, ale kvůli zaokrouhlení se vyjádří jako stejná hodnota plovoucího čísla, jsou poté nerozeznatelná. Každý plovoucí typ pak má garantovanou přesnost, na kterou by se různá čísla neměla reprezentovat stejně.

Obrázek 4: Racionální čísla v jazyce C



Nejrozsáhlejším datovým typem čísla s plovoucí řádovou tečkou je v jazyce C `long double`. Na příkladu vidíme, že jeho přesnost by měla být 18 desetinných míst, ale už na patnácti místech jsou dvě různá zapisovaná čísla zapsána stejně. Někde se tedy stala chyba a přesnost `long double` je stejná jako `double`, ale knihovna `float.h` to nereflektuje.

Racionální čísla vyjádřená jako plovoucí můžeme považovat za přesná, pokud se nedostaneme mimo interval a přesnost stanovenou typem. Na Obrázku 4 vidíme, že rozsah datového typu `long double` je téměř 10^4 řádů a přesnost je maximálně 15 desetinných míst. Čísla s plovoucí řádovou tečkou jsou nejlepší přiblížení k racionálním číslům, které lze vyjádřit jako hodnoty.

2.3 Přesná reprezentace čísel jako struktur

Již víme, že některá čísla lze přesně uložit jejich hodnotou. Nicméně standardní datové typy mají své limity, například přetékání nebo nedostatečná přesnost. Číslo lze však namísto hodnotového datového typu reprezentovat typem referenčním. V kapitole 1 bylo nastíněno, že stejné číslo lze vyjádřit několika symboly, i když jeho hodnota je pouze jedna. Takto reprezentovaná čísla umíme velmi dobře v počítači vyjadřovat. Struktury v této podkapitole jsou pouze teoretickými koncepty bez konkrétní implementace.

2.3.1 Přirozená čísla

Jak bylo ukázáno v minulé podkapitole, přirozená čísla jsou uzavřena na sčítání a násobení, `unsigned int` ovšem nikoli. Cílem je tedy vymyslet strukturu, která dokáže reprezentovat jakékoli přirozené číslo. Jako hodnoty umíme na n bitech uložit čísla 0 až $2^n - 1$, proto by pro přirozená čísla bylo dobré adekvátně upravovat toto n. Velikost čísla by pak byla omezena jen velikostí paměti.

Struktura představující přirozené číslo musí zajišťovat, aby jeho hodnota při aplikaci matematických operací nepřetekla. Pokud struktura obdrží požadavek na násobení nebo sčítání, musí mít alokovaný další prostor a tam posunout bity, které by normálně přetekly. To je možné zajistit tak, že ve svém pomyslném paměťovém prostoru bude vždy po provedení operace procházet svých levých 2^{n-1} bitů, a pokud zde najde alespoň jednu jedničku, požádá o alokaci dalšího prostoru o velikost 2^n, tedy zvedne n o jedna.

Pak už bude na operačním systému, kolik paměti bude moci struktuře přiřadit. Tato paměť je vždy prakticky omezená, teoreticky je však neomezená. Strukturu si tedy lze představit jako pouhý chytrý řadič bloků paměti za sebe. Pak lze *naprosto přesně* zapsat jakékoli přirozené číslo. Paměťově toto není příliš efektivní, jedná se však o funkční princip. Optimalizace je ponechána až na konkrétní implementace.

2.3.2 Celá čísla

Situace v případě celých čísel je velmi podobná jako u přirozených čísel. Kromě přetečení navíc hrozí také podtečení, protože celá čísla musí být uzavřena i na odčítání.

Vytvořme strukturu pro celé číslo jako dvojici přirozeného čísla a příznaku kladnosti. Metody struktury pak budou jen nastavovat příznak kladnosti – u násobení jako exkluzivní disjunkci, odčítání odpovídá sčítání s negací příznaku a u sčítání se příznak nastaví podle příznaku u většího čísla. Takto vytvořené struktury pak *naprosto přesně* reprezentují jakékoli celé číslo.

2.3.3 Racionální čísla

Racionální čísla jsme definovali jako podíl celého a nenulového celého čísla. Racionální čísla (bez nuly) jsou uzavřená i na dělení. Zaveďme teď racionální číslo jako dvojici celých čísel (čitatele a jmenovatele) s invariantem, že druhé číslo nesmí být nikdy nula a že obě čísla jsou nesoudělná.

Násobení bude fungovat jako násobení na složkách, dělení pak bude jen prohození obou složek druhého argumentu a následné násobení. Odčítání je sčítání s opačným číslem a sčítání musí najít společný jmenovatel a poté specificky přenásobovat a sčítat jednotlivé operandy.

Metody musí stále kontrolovat, zda nedochází k dělení nulou. Po konci výpočtu musí dojít ke zkrácení obou složek čísla. Predikát rovnosti dvou racionálních čísel kvůli podmínce nesoudělnosti je pak jednoduchý a je to rovnost na složkách.

Tato struktura je *naprosto přesnou* reprezentací racionálních čísel. Jazyk Lisp implementuje racionální čísla v této úrovni přesnosti a nabízí datový typ `ratio`.

2.4 Reálná čísla

Předchozí číselné obory lze s dostatkem paměti naprosto přesně reprezentovat. Zbývají už jen iracionální čísla a máme všechna reálná čísla v přesné reprezentaci. Iracionálních čísel je nespočetně mnoho (stejně jako transcendentních a nerekurzivních čísel), a tak je není možné vytvořit z přirozených čísel nabalováním struktur jako předchozí obory. Nic jiného ale v paměti, která pracuje s diskrétními hodnotami, neumíme vytvořit.

Dosud jsme mluvili o *naprosto přesných* číslech, struktura vyjadřující přesné číslo však může být i taková, která počítá jen přibližná čísla, nastavitelně vzdálená od (neznámé) přesné hodnoty. Takových čísel je spočetně mnoho a říkáme jim čísla rekurzivní. Ve skutečnosti je právě hranice mezi racionálními čísly a rekurzivními čísly velká bariéra, která odděluje svět, kde vše funguje relativně jednoduše (jak jsme viděli) a ten, kde je vše o řád těžší.

2.4.1 Představa

Než struktury definovat imperativně pomocí definic, jak by co měla implementovat, se spíše pokusím o definici struktury deklarativně, čili pomocí vlastností, které by měla splňovat.

Abstraktní struktura reprezentující reálné číslo by měla umožnit

- jeho vyčíslení – když struktura existuje, pak po zavolání vhodného nástroje je výsledkem hodnota, kterou tato struktura představuje;
- jeho přesnost – i když má číslo nekonečný rozvoj nebo je velmi velké, abstrakce umožňuje jeho vyčíslení na danou přesnost v konečném čase;
- podporovat matematické operace – ve smyslu kapitoly 1.3;
- podporovat matematické funkce – ve smyslu kapitoly 1.4;
- být vracena jako výsledek funkce – mít jasně popsanou strukturu, aby se dala rozšiřovat funkčnost;
- být použita jako argument nějaké funkce – typicky funkce pro vyčíslení.

První dvě podmínky jsou přímo esenciální – zajišťují, že abstrakce mohou reprezentovat reálná čísla na libovolnou (kladnou) přenost. Kdykoli budu potřebovat číslo dané konkrétní abstrakcí, zavolám příslušnou funkci s parametrem ε reprezentujícím přesnost, na kterou toto číslo potřebuji. Obecně totiž nemusí být výsledkem vyčíslovací funkce přesná hodnota hledaného čísla, avšak díky těmto podmínkám budu výsledku vzdálen maximálně o zadanou hodnotu odchylky. *Přesné* číslo tedy představuje číslo, které je od výsledku vzdálené o jasně definovanou hodnotu vyčíslitelné konečným výpočtem. Uvažujme strukturu s_r přesně vyjadřující nějaké číslo $r \in \mathbb{R}$, zavolejme vyčíslovací funkci *enum* a jako argumenty volme tuto strukturu a libovolné kladné ε, pak by výsledkem mělo být číslo $enum(s_r, \varepsilon)$ splňující nerovnost

$$|enum(s_r, \varepsilon) - r| \leq \varepsilon. \tag{20}$$

Druhé dvě podmínky vštěpují dané struktuře reprezentující reálná čísla vlastnosti reálných čísel, a sice že s nimi jde sčítat, násobit atp. a že mohou být argumentem nějaké matematické funkce. Jistě nejde o to tyto struktury vkládat do stejných operací jako si představujeme s normálními čísly, nýbrž chceme existenci ekvivalentní operace pro tyto struktury. To nutně neznamená, že musí být skutečně někde implementována, ale potřebujeme docílit stavu, kdy existence této není dokazatelně vyloučena.

Třetí a poslední dvě podmínky jsou spíše návod pro praktické použití těchto hypotetických struktur, aby se s nimi dalo smysluplně pracovat. To vlastně znamená, že musí být elementy prvního řádu (*first-class citizen*), čili implementovány jako niterná součást použitého jazyka a ne jako svébytná konstrukce, která sice funguje, ale je uživatelsky nerozšiřitelná.

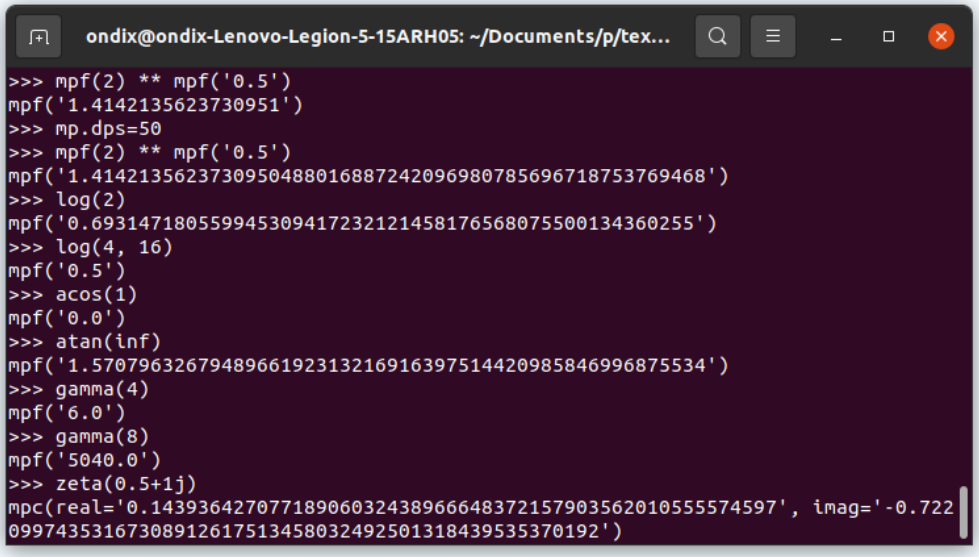
2.4.2 Existující nástroje

Nyní se podíváme, co na poli vyčíslování reálných čísel existuje v současné době.

2.4.2.1 Mpmath [21] `mpmath` je velmi rozsáhlá knihovna pro Python. Je publikována pod licensí BSD a je dosažitelná i pomocí `pipu`. Kromě základní funkčnosti pro výpočet funkcí a operací s čísly jsou implementovány i funkce pro výpočet funkcí intervalů, určitých integrálů, podpora tvorby grafů a mnoho dalšího. Dokumentace je velice hezky zpracovaná s množstvím příkladů. Přesnost se nastavuje proměnnou `mp.dps`. Jde o počet vypisovaných míst. Knihovna oplývá tolika možnostmi, že dokonce existuje stránka pro výpočet Ludolfova čísla sty možnými one-linery. Knihovna používá tři vlastní datové typy a sice `mpf` pro reálná čísla, `mpc` pro komplexní čísla a `matrix` pro matice. Příklad použití je na Obrázku 5.

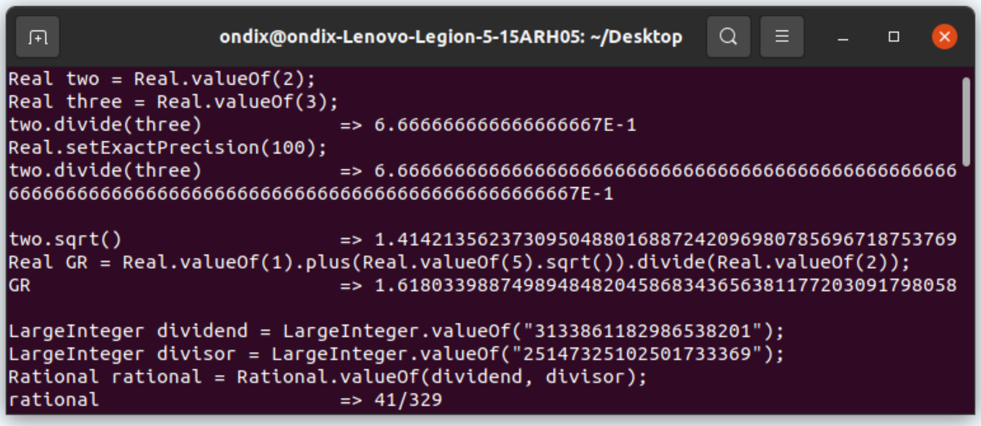
2.4.2.2 JScience [22] `JScience` je knihovna pro jazyk Java. Její část pro práci s jednotkami se dostala do knihovny `javax`. Knihovna je široce rozkročena. Přináší podporu pro porovnávání a počítání jednotek z fyziky, geografie nebo ekonomie. Z matematiky je zde podpora pro jednoduchou symbolickou analýzu a strukturální algebru. Nás nejvíce zajímá `org.jscience.mathematics.number`. Zde jsou 3 zajímavé datové typy a to `Real` umožnující základní výpočty s nastavitelnou přesností, `LargeInteger` ukládající velká celá čísla a `Rational` ukládající dvojice `LargeIntegerů` a umožňující jejich implicitní usměrňování a základní matematické operace. Příklad použití je na Obrázku 6.

Obrázek 5: Používání knihovny `mpmath`



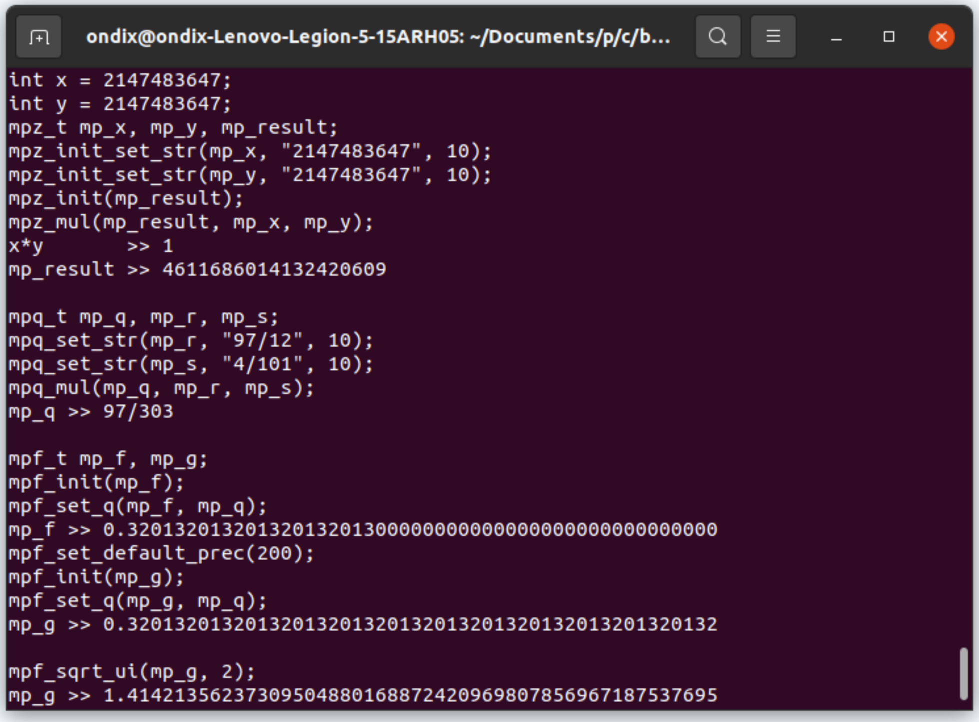
Struktura `mpf` podporuje matematické operace, na příkladu je vyobrazena odmocnina ze dvou. Ta je zde ve dvou provedeních, s defaultní přesností a s nastavenou na 50. Ukázány jsou i další matematické funkce, jmenovitě logaritmus, cyklometrické, gamma a Riemannova zeta funkce, na jejímž vstupu i výstupu vidíme komplexní číslo.

Obrázek 6: Používání knihovny `JScience`



Na obrázku je ukázáno používání třídy `Real`. Operace jsou použitelné jako metody, nikoli operátorem. Konkrétně je představeno nastavování přesnosti, výpočet druhé odmocniny, zlatého řezu a zlomkové struktury s implicitním krácením.

Obrázek 7: Používání knihovny GMP

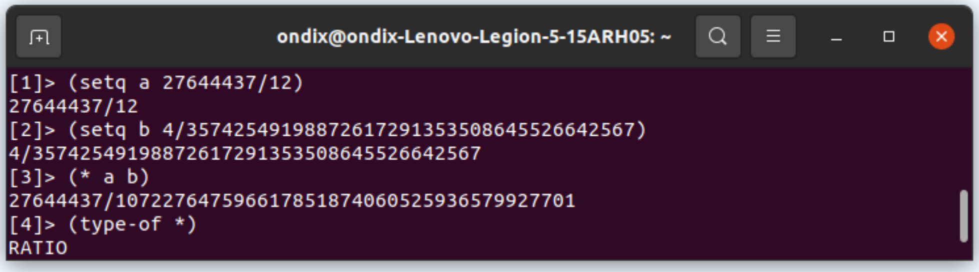


Na příkladu vidíme nejprve porovnání násobení klasických `intů` versus struktur typu `mpz_t`. U `intů` dojde k přetečení, u mpz nikoli. Dále vidíme práci se zlomky, jejich inicializaci ze `stringu` a násobení. Nakonec vidíme základní operace s typem `mpf_t`, protějškem čísel s plovoucí řádovou tečkou.

2.4.2.3 GNU Multiple Precision Arithmetic Library [23] (GMP)

`GMP` je knihovna pro jazyk C. Datový typ `mpz_t` představuje celé číslo, u kterého nehrozí přetečení nebo podtečení. Zlomky velkých čísel představuje `mpq_t` a operace podporují usměrňování. Přesný ekvivalent čísla s plovoucí řádovou tečkou představuje `mpf_t`. Minimální počet bytů, ve kterém je uložen v paměti, se nastavuje funkcí `mpf_set_default_prec`. Všechny typy implementují základní matematické operace. Protože je GMP součástí projektu GNU a protože je to knihovna pro C, je velmi mnoho nadstavbových knihoven, které její funkcionalitu využívají a rozšiřují. Z C-čkových jmenujme například `GNU MPFR`, která je na GMP přímo založena [24] a přináší matematické funkce floatů, nebo `MPIR` – paralelní projekt odtrhnuvší se od vývoje GMP a jdoucí svojí cestou, přesto snažící se implementovat rozhraní GMP, aby byly zastupitelné [25]. Dále existují wrappery pro kompatibilitu s jinými jazyky a tudíž je GMP velmi rozšířená, ač se to nemusí uživateli zdát. Například pro platformu .NET je to knihovna `Math.GMP.Native`, pro Python `gmpy`, pro R `gmp`. Příklad použití je na Obrázku 7.

Obrázek 8: Implicitní používání knihovny `CLN`



Knihovnu CLN používá CLisp, interpret jazyka Lisp. Využívá implementace čísel. Příklad ukazuje, že se zlomky automaticky zkracují. Že je CLisp založen na CLN usuzuji podle osoby Bruna Heible-a, který figuruje jako autor jak u CLN, tak u CLisp-u.

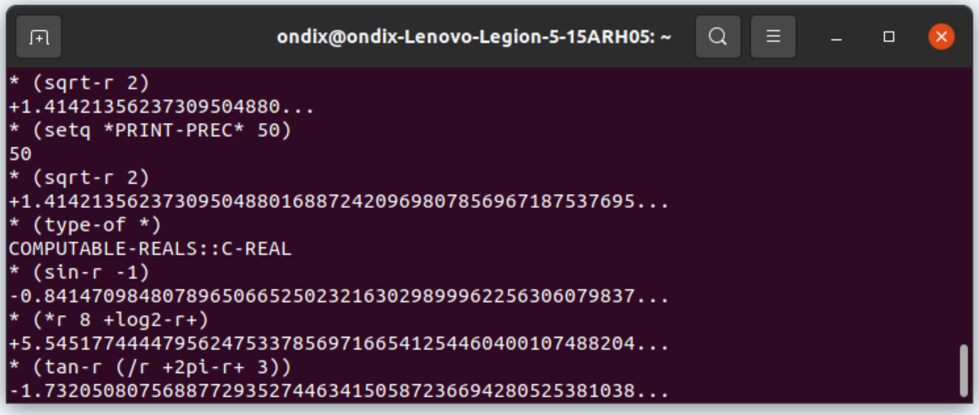
2.4.2.4 Class Library for Numbers [26] (CLN) `CLN` je knihovna pro jazyk C++ a je zdarma šířená pod licensí GPL. Implementuje čísla s plovoucí řádovou tečkou, racionální čísla jako zlomky, navíc komplexní čísla. Za přesná se považují racionální čísla a komplexní čísla s přesnou imaginární i reálnou částí. Plovoucí čísla pak typově přesně kopírují Lispovské a lze je tedy tedy použít k Lispovským implementacím. Zkratku CLN je tedy možné chápat i jako „Common Lisp Numbers“. Nepřímé použití je na Obrázku 8.

2.4.2.5 Computable-reals [27] `computable-reals` je Lispovská knihovna. Je volně ke stažení a dokonce k dostání pomocí `quicklispu`. Podporuje základní funkcionalitu. Její funkce poznáme tak, že končí koncovkou `-r`. Vracené výsledky nejsou čísla, ale vlastního typu `C-REAL`. Defaultně se tisknou výsledky na 20 míst, ale nastavením proměnné `*print-prec*` se tento počet dá měnit [28]. Kromě odmocniny jsou zde třeba základní násobky čísla π, logaritmy, mocniny, základní goniometrické funkce, arcus tangens. Knihovna se používá jako kalkulačka s nastavitelnou přesností. Příklad použití je na Obrázku 9.

2.4.2.6 Cíl práce Tolik tedy k strukturám již nyní implementujícím čísla a jejich operace, případně funkce. V některých implementacích se jedná o třídy, v jiných jde o strukturované datové typy. V následujícím textu se pokusíme navázat tam, kde končí naprostá přesnost nad racionálními čísly jazyka Lisp a naprogramujeme knihovnu přinášející některá iracionální čísla.

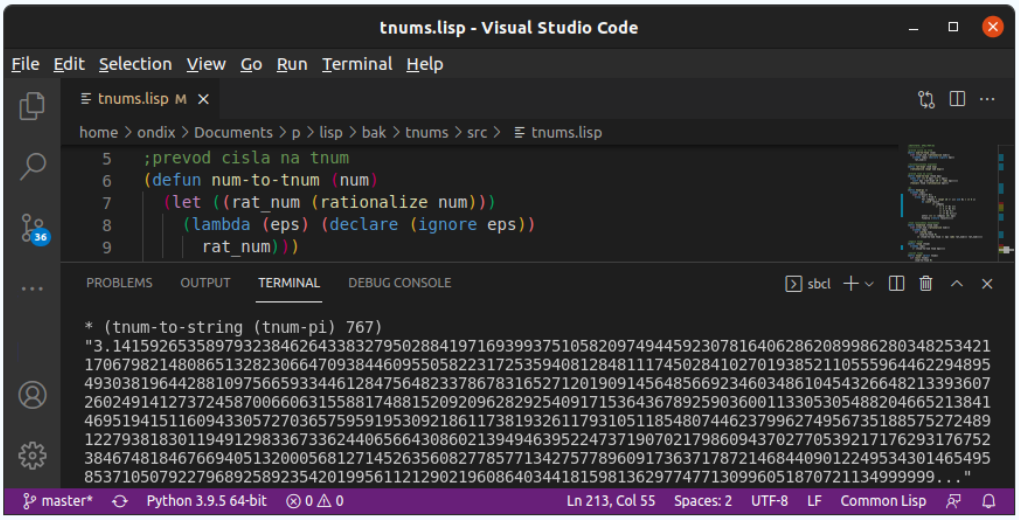
V této práci nám jde o přesnost a nikoli o rychlost, a proto jako nativní typy budu používat právě zlomky, ačkoli pro rychlé operace s desetinnými čísly se používají plovoucí čísla, které mají často i hardwarovou podporu v jednotce FPU. Ostatně proto jsou v Lispu i plovoucí typy [29].

Obrázek 9: Používání knihovny `computable-reals`



Na příkladu vidíme výpis odmocniny ze dvou na různé přesnosti, operace násobení a několik matematických funkcí.

Obrázek 10: Editor `code`



Na obrázku vidíme snímek editoru. Patrné je podbarvování syntaxe a duhové závorky. Dále terminál přímo v prostředí editoru a v něm spuštěný kompilátor. Zároveň vypsání π až do Feynmanova bodu pomocí knihovny `tnums`, která vznikne na dalších stránkách.

K programování jsem jako textový editor použil *Visual Studio Code* s rozšířeními *2gua.rainbow-brackets* a *qingpeng.common-lisp*, jako kompilátor *SBCL*. Jak přibližně vypadalo kódování a testování je zobrazeno na Obrázku 10. Odsazování jsem nakonec poupravil v LispWorks.

Část II
Implementace

V implementační části aplikujeme teorii a dáme vzniknout knihovně `tnums`. Všechna čísla i manipulaci s nimi lze vyjádřit jako funkce (množiny), přesně toto naprogramujeme. Situace nás vede k užití funkcionálního programovacího jazyka, zvolen byl Lisp. Budou ukázány převody, konstanty, operace a funkce.

3 Tnumy

Struktury jsou dány jako funkce jedné proměnné – přesnosti. Je potřeba je matematicky nadefinovat a poté v Lispu vymodelovat.

3.1 Vztah čísel a tnumů

Funkce, kterými budu modelovat rekurzivní čísla a které budu poté v Lispu implementovat nazývám *True Numbers*, zkráceně *tnums*. Jsou totiž opravdovější než čísla, která jsou uložena jako hodnoty, čímž ztrácí na přesnosti. Vzniknuvší knihovna se pak jmenuje `tnums`.

Definice 23 (Tnum). *Funkce* $\mathsf{t}^x : (0,1) \to \mathbb{Q}$*, která pro všechna* $\varepsilon \in (0,1)$ *vrací hodnotu* $\mathsf{t}^x(\varepsilon)$ *splňující nerovnost*

$$|\mathsf{t}^x(\varepsilon) - x| \leq \varepsilon \tag{21}$$

se nazývá tnum [ti:nʌm] *čísla* x*.*

Množinu všech tnumů čísla x značíme $\mathcal{T}^x$, $(\forall x \in \mathbb{R})(\mathcal{T}^x = \{\mathsf{t}^x | (\forall \varepsilon \in (0,1)) : |\mathsf{t}^x(\varepsilon) - x| \leq \varepsilon\})$. Množinu všech tnumů pak značíme symbolem $\mathfrak{T}$.

Tnum t^x je struktura představující rekurzivní číslo x. Při výpočtu jeho hodnoty nejprve tento tnum vytvoříme (výpočetně rychlé), a pak ho necháme vyčíslit, tj. zavolat s přesností (výpočetně pomalé). Vyčíslení tedy odkládáme na nejpozdější možnou dobu, mluvíme o líném vyhodnocování.

Tnumy přesných čísel lze vyčíslit s dokonalou přesností. Lisp přesně reprezentuje všechna racionální čísla. To je ve shodě s principem vyčíslení rekurzivního čísla. Pomocí $\mathsf{t}^r(\varepsilon)$ tedy získáváme q z nerovnice (7) a též $enum(s_r, \varepsilon)$ z nerovnice (20). Číslo, jak ho chápe Lisp dále označuji *num*.

Lemma 24 (O numu jako tnumu). *Pro všechna* $x \in \mathbb{R}$ *a všechna* $\varepsilon \in (0,1)$ *platí: číslo* $\mathsf{t}^x(\varepsilon)$ *lze nahradit číslem* x*.*

Důkaz. Z nerovnosti (21) získáváme $|\mathsf{t}^x(\varepsilon) - x| \leq \varepsilon$. Po dosazení $\mathsf{t}^x(\varepsilon) := x$ pak $|x - x| = 0 \leq \varepsilon$, což platí pro všechna $x \in \mathbb{R}$ i $\varepsilon \in (0,1)$. □

Nejpřesnější reprezentace čísla je toto číslo samotné. Lisp pracuje i se zlomky (datový typ `ratio`), proto mohou být přesná všechna racionální čísla.

```
(defun num-to-tnum (num)
  (let ((rat_num (rationalize num)))
    (lambda (eps) (declare (ignore eps))
      rat_num)))
```

Lispový kód 1 (`num-to-tnum`)**:** *Funkce převádějící num na tnum*

Převod opačným směrem je přímočarý. Chceme-li číslo $x \in \mathbb{R}$ s přesností $\varepsilon \in (0,1)$, stačí zavolat $\mathfrak{t}^x(\varepsilon)$. Mimo přípustný interval ε chápeme jako $10^{-|\varepsilon|}$.

Lemma 25 (O převodu tnumu na num). *Pokud existuje funkce $\mathfrak{t}^x \in \mathcal{T}^x$, pak po zavolání s argumentem ε vrací hodnotu $\mathfrak{t}^x(\varepsilon)$ splňující $(|\mathfrak{t}^x(\varepsilon) - x| \leq \varepsilon)$.*

Důkaz. Plyne přímo z definice 23. □

```
(defun rat-expt (num exp)
  (rationalize (expt num exp)))
```

Lispový kód 2 (`rat-expt`)**:** *Funkce pro racionální umocňování*

```
(defun tnum-to-num (tnum eps)
  (when (or (>= 0 eps) (<= 1 eps))
    (setf eps (rat-expt 10 (- (abs eps)))))
  (funcall tnum (rationalize eps)))
```

Lispový kód 3 (`tnum-to-num`)**:** *Funkce převádějící tnum na num*

Z čísla na tnum je převod jednoduchý, opačným směrem lze převádět jen, pokud tnum existuje. V knihovně teď jsou jen tnumy racionálních čísel a aparát pro převody mezi $\mathfrak{T}$ a $\mathbb{Q}$. Teď půjde o tvorbu co nejvíce tnumů iracionálních čísel.

3.2 Ludolfovo číslo

První iracionální konstantou, která do knihovny přibude je Ludolfovo číslo.

Definice 26 (Ludolfovo číslo [30]). *Ludolfovo číslo π je dáno jako poměr obvodu kružnice k jejímu průměru.*

Ludolfovo číslo je nejslavnější transcendentní konstanta, pro jejíž vyčíslení existuje přemnoho vzorců, například vzorec Leibnizův: $\pi = 4\sum_{n\in\mathbb{N}} \frac{(-1)^n}{2n+1}$ [31].

Rychleji konverguje řada v BBP (tvůrci Bailey, Borwein, Plouffe) vzorci.

Fakt 27 (Kohoutkový BBP vzorec [32]).

$$\pi = \sum_{i\in\mathbb{N}} \frac{1}{16^i}\left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6}\right). \tag{22}$$

Máme řadu čísla, jež chceme přidat do `tnums`. Výraz v závorce je pro $i > 0$ menší než jedna. Proto lze každý i-tý člen, kde $i > 0$, zhora omezit 16^{-i}. Omezující členy tvoří geometrickou řadu, jejíž zbytek je dle faktu 20 roven $\frac{1}{16^i * 15}$. Platí

$$\left|\pi - \sum_{i=0}^{n} \frac{1}{16^i}\left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6}\right)\right| \leq \frac{1}{16^n * 15}. \tag{23}$$

Důsledek 28 (Tnum Ludolfova čísla). *Nechť* $\mathfrak{t}()$ *je funkce s předpisem*

$$\mathfrak{t}()(\varepsilon) = \left[\begin{array}{l} 1.\ \textit{Najdi nejmenší } n \in \mathbb{N}^+ \textit{ tak, aby } /(16^n 15) \leq \varepsilon; \\ 2.\ \textit{Vrať } \sum_{i=0}^{n} \frac{1}{16^i}\left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6}\right), \end{array}\right. \tag{24}$$

pak $\mathfrak{t} \in \mathcal{T}^{\pi}$.

```lisp
(defun tnum-pi ()
  (lambda (eps)
    (let ((result 0))
      (loop for i from 0
            for /16powi = (expt 16 (- i)) and 8i = (* 8 i)
            do (incf result
                     (* /16powi
                        (- (/ 4 (+ 8i 1))
                           (/ 2 (+ 8i 4))
                           (/ (+ 8i 5))
                           (/ (+ 8i 6)))))
            until (<= (/ /16powi 15) eps)
            finally (return result)))))
```

Lispový kód 4 (`tnum-pi`)**:** *Funkce pro tnum Ludolfova čísla*

3.3 Přenásobování numem

Racionální násobky tnumů umožní vyčíslení například čísla $2\pi/3$.

Věta 29 (O přenásobení tnumu racionální konstantou). *Nechť* $c \in \mathbb{Q}$, $\mathfrak{t}^x \in \mathcal{T}^x, x \in \mathbb{R}$ *a funkce* $\mathfrak{t}(\mathfrak{t}^x, c)$ *má předpis*

$$\mathfrak{t}(\mathfrak{t}^x, c)(\varepsilon) = \begin{cases} c * \mathfrak{t}^x\left(\frac{\varepsilon}{|c|}\right) & \textit{pro } c \neq 0, \\ 0 & \textit{pro } c = 0, \end{cases} \tag{25}$$

pak $\mathfrak{t}(\mathfrak{t}^x, c) \in \mathcal{T}^{x*c}$.

Důkaz. Pokud přenásobíme jakékoli číslo nulou, je výsledkem nula (agresivní prvek vůči násobení). Znění věty pro nenulovou konstantu dokážeme tak, že z předpokladu $|\mathfrak{t}^x(\varepsilon) - x| \leq \varepsilon$ odvodíme $|c * \mathfrak{t}^x(\frac{\varepsilon}{|c|}) - c * x| \leq \varepsilon$. Protože pracujeme s nerovnicemi, budeme postupovat dvěmi větvemi – pro c kladné a záporné.

Z definice tnumu předpokládáme

$$|\mathfrak{t}^x(\varepsilon) - x| \leq \varepsilon, \tag{26}$$

po přenásobení kladným $c > 0$ dostáváme

$$c * |\mathfrak{t}^x(\varepsilon) - x| \leq c * \varepsilon, \tag{27}$$

protože je ale c kladné, lze jím absolutní hodnotu roznásobit

$$|c * \mathfrak{t}^x(\varepsilon) - c * x| \leq c * \varepsilon, \tag{28}$$

a nyní na pravé straně potřebujeme dostat přesnost ε. Protože dle definice platí $|\mathfrak{t}^y(\varepsilon) - y| \leq \varepsilon$, platí jistě i $|\mathfrak{t}^y(\frac{\varepsilon}{c}) - y| \leq \frac{\varepsilon}{c}$, pak ale musí platit i

$$\left| c * \mathfrak{t}^x\left(\frac{\varepsilon}{c}\right) - c * x \right| \leq \varepsilon. \tag{29}$$

Pro zápornou konstantu je důkaz podobný, a protože jako argument tnumů bereme kladné číslo, přibývá v děliteli v argumentu tnumu ještě absolutní hodnota. Dohromady pak získáváme

$$\left| c * \mathfrak{t}^x\left(\frac{\varepsilon}{|c|}\right) - c * x \right| \leq \varepsilon, \tag{30}$$

což jsme chtěli odvodit. □

```
(defun tnum*num (tnum num)
  (let ((rat_num (rationalize num)))
    (lambda (eps)
      (if (zerop num)
          (num-to-tnum 0)
        (* (tnum-to-num tnum (/ eps (abs rat_num))) rat_num)))))
```

Lispový kód 5 (`tnum*num`)**:** *Funkce přenásobující tnum racionální konstantou*

Důsledek 30 (Opačný tnum). *Nechť* $\mathfrak{t}^x \in \mathcal{T}^x, x \in \mathbb{R}$ *a funkce* $\mathfrak{t}(\mathfrak{t}^x)$ *má předpis*

$$\mathfrak{t}(\mathfrak{t}^x)(\varepsilon) = -\mathfrak{t}^x(\varepsilon), \tag{31}$$

pak $\mathfrak{t}(\mathfrak{t}^x) \in \mathcal{T}^{-x}$.

Důkaz. Protože $-x = (-1)x$ a $|-1| = 1$, pak z přechozí věty dostáváme $-\mathfrak{t}^x(\varepsilon) = (-1)\mathfrak{t}^x(\varepsilon) = (-1)\mathfrak{t}^x(\frac{\varepsilon}{1}) = (-1)\mathfrak{t}^x(\frac{\varepsilon}{|-1|}) = \mathfrak{t}^{(-1)x}(\varepsilon) = \mathfrak{t}^{-x}(\varepsilon) \in \mathcal{T}^{-x}$. □

```
(defun -tnum (tnum)
  (lambda (eps)
    (- (tnum-to-num tnum eps))))
```

Lispový kód 6 (`-tnum`)**:** *Funkce pro opačný tnum*

4 Operace tnumů

Implementace matematických operací je různě složitá. Zatímco aditivní operace jsou celkem jednoduché, multiplikativní jsou řádově složitější, tak mocninné dokončíme až na konci následující kapitoly.

4.1 Aditivní operace

Operace `+` bere přirozený počet argumentů. Pro žádný vrací nulu (neutrální prvek aditivní grupy [13]), pro jeden vrací tento a pro více pak jejich součet (je třeba doprogramovat součet). Operace `-` potom vyžaduje alespoň jeden argument, v případě zadání pouze tohoto se vrací tnum k němu opačný, v případě více pak jejich rozdíl (je třeba doprogramovat rozdíl).

Věta 31 (O součtu tnumů). *Nechť* $\mathfrak{t}^{x_0} \in \mathcal{T}^{x_0}, \mathfrak{t}^{x_1} \in \mathcal{T}^{x_1}, \ldots, \mathfrak{t}^{x_n} \in \mathcal{T}^{x_n}$ *a funkce* $\mathfrak{t}(\mathfrak{t}^{x_0}, \mathfrak{t}^{x_1}, \ldots, \mathfrak{t}^{x_n})$ *má předpis*

$$\mathfrak{t}(\mathfrak{t}^{x_0}, \mathfrak{t}^{x_1}, \ldots, \mathfrak{t}^{x_n})(\varepsilon) = \sum_{i=0}^{n} \mathfrak{t}^{x_i}\left(\frac{\varepsilon}{n+1}\right), \tag{32}$$

pak $\mathfrak{t}(\mathfrak{t}^{x_0}, \mathfrak{t}^{x_1}, \ldots, \mathfrak{t}^{x_n}) \in \mathcal{T}^{x_0+x_1+\ldots+x_n}$.

Důkaz. Předpokládejme $\mathfrak{t}^{x_i}(\varepsilon) + \varepsilon \geq x_i$ a $\mathfrak{t}^{x_i}(\varepsilon) - \varepsilon \leq x_i$ pro $i \in \{0, 1, \ldots, n\}$ a ukažme $\sum_{i=0}^{n} \mathfrak{t}^{x_i}\left(\frac{\varepsilon}{n+1}\right) + \varepsilon \geq \sum_{i=0}^{n} x_i$ a $\sum_{i=0}^{n} \mathfrak{t}^{x_i}\left(\frac{\varepsilon}{n+1}\right) - \varepsilon \leq \sum_{i=0}^{n} x_i$.

Z definice tnumu předpokládáme

$$\begin{gathered}
\mathfrak{t}^{x_0}\left(\frac{\varepsilon}{n+1}\right) + \frac{\varepsilon}{n+1} \geq x_0 \wedge \mathfrak{t}^{x_0}\left(\frac{\varepsilon}{n+1}\right) - \frac{\varepsilon}{n+1} \leq x_0, \\
\ldots \\
\mathfrak{t}^{x_n}\left(\frac{\varepsilon}{n+1}\right) + \frac{\varepsilon}{n+1} \geq x_n \wedge \mathfrak{t}^{x_n}\left(\frac{\varepsilon}{n+1}\right) - \frac{\varepsilon}{n+1} \leq x_n.
\end{gathered} \tag{33}$$

Sečtěme teď všechny výrazy a získáváme

$$\sum_{i=0}^{n} \mathfrak{t}^{x_i}\left(\frac{\varepsilon}{n+1}\right) + \sum_{i=0}^{n} \frac{\varepsilon}{n+1} \geq \sum_{i=0}^{n} x_i \wedge \sum_{i=0}^{n} \mathfrak{t}^{x_i}\left(\frac{\varepsilon}{n+1}\right) - \sum_{i=0}^{n} \frac{\varepsilon}{n+1} \leq \sum_{i=0}^{n} x_i, \tag{34}$$

dále $\sum_{i=0}^{n} \frac{\varepsilon}{n+1} = \varepsilon$, takže

$$\sum_{i=0}^{n} \mathfrak{t}^{x_i}\left(\frac{\varepsilon}{n+1}\right) + \varepsilon \geq \sum_{i=0}^{n} x_i \wedge \sum_{i=0}^{n} \mathfrak{t}^{x_i}\left(\frac{\varepsilon}{n+1}\right) - \varepsilon \leq \sum_{i=0}^{n} x_i, \tag{35}$$

což jsme chtěli ukázat. $\square$

```
(defun tnum+ (&rest tnums)
  (if (null tnums)
      (num-to-tnum 0)
    (lambda (eps)
      (let ((new-eps (/ eps (list-length tnums))))
        (apply '+
               (mapcar (lambda (tnum)
                         (tnum-to-num tnum new-eps))
                       tnums))))))
```

Lispový kód 7 (`tnum+`): *Funkce na součet tnumů*

Fakt 32 (Rozdíl tnumů). *Nechť* $\mathfrak{t}^{x_i} \in \mathcal{T}^{x_i}, x_i \in \mathbb{R}$ *pro* $i \in \{-1, 0, \ldots, n\}$ *a funkce* $\mathfrak{t}(\mathfrak{t}^{x_{-1}}, \mathfrak{t}^{x_0}, \ldots, \mathfrak{t}^{x_n})$ *má předpis*

$$\mathfrak{t}(\mathfrak{t}^{x_{-1}}, \mathfrak{t}^{x_0}, \ldots, \mathfrak{t}^{x_n})(\varepsilon) = \mathfrak{t}^{x_{-1} - \sum_{i=0}^{n} x_n}(\varepsilon) = \mathfrak{t}^{x_{-1} + \left(-\sum_{i=0}^{n} x_n\right)}(\varepsilon), \tag{36}$$

pak $\mathfrak{t}(\mathfrak{t}^{x_{-1}}, \mathfrak{t}^{x_0}, \ldots, \mathfrak{t}^{x_n}) \in \mathcal{T}^{x_{-1} - x_0 - \ldots - x_n}$.

```
(defun tnum- (tnum1 &rest tnums)
  (if (null tnums)
      (-tnum tnum1)
    (tnum+ tnum1 (-tnum (apply 'tnum+ tnums)))))
```

Lispový kód 8 (`tnum-`): *Funkce pro opačný tnum a rozdíl tnumů*

4.2 Multiplikativní operace

Jako první multiplikativní operaci představím převrácení hodnoty tnumu. Je to unární operace jako Opačný tnum, jen jde o jinou inverzi. Protože převrácení pracuje jen s nenulovými čísly, přidáme ještě aparát pro tnumy nenulových čísel.

Definice 33 (Nenulový tnum). *Tnum* $\mathfrak{t}$*, který nikdy nenabývá nulové hodnoty, neboli* $(\forall \varepsilon \in (0,1))(\mathfrak{t}(\varepsilon) \neq 0)$ *budeme nazývat nenulový tnum a budeme jej značit* $\mathfrak{t}_{\emptyset}$.

Nenulová čísla tedy budeme moci reprezentovat nenulovými tnumy.

Definice 34 (Bezpečné epsilon). *K libovolnému* $\varepsilon \in (0,1)$ *a tnumu* $\mathfrak{t}^x \in \mathcal{T}^x, x \neq 0$ *uvažujeme číslo* $\varepsilon_{\emptyset}(\mathfrak{t}^x, \varepsilon)$ *tak, že*

1. $0 < \varepsilon_{\emptyset}(\mathfrak{t}^x, \varepsilon) \leq \varepsilon$,
2. $\mathfrak{t}^x(\varepsilon_{\emptyset}(\mathfrak{t}^x, \varepsilon)) \neq 0$ *a*

3. $|\mathsf{t}^x(\varepsilon_\emptyset(\mathsf{t}^x, \varepsilon))| > \varepsilon_\emptyset(\mathsf{t}^x, \varepsilon)$

a nazýváme jej bezpečným epsilonem.

Lemma 35 (O nenulovém tnumu nenulového čísla). *Tnum nenulového čísla lze vyčíslit nenulově, čili* $(\forall x \in (\mathbb{R} \setminus \{0\}))((\exists \mathsf{t}^x) \rightarrow (\exists \mathsf{t}^x_\emptyset))$.

Důkaz. Vezměme za $\mathsf{t}^x_\emptyset$ funkci $\mathsf{t}(\mathsf{t}^x)$, s předpisem $\mathsf{t}(\mathsf{t}^x)(\varepsilon) = \mathsf{t}^x(\varepsilon_\emptyset(\mathsf{t}^x, \varepsilon))$. Pak díky podmínce 1 v definici 34 vyčíslení proběhne v pořádku a díky bodu 2 ve stejné definici bude vyčíslení nenulové, díky čemuž se jedná o nenulový tnum. □

Bezpečných epsilonů je nekonečně mnoho, stačí nám najít jediné. Funkce pro jeho výpočet potřebuje tnum a epsilon. To je ve shodě se zavedeným symbolem $\varepsilon_\emptyset(\mathsf{t}^x, \varepsilon)$. Dále musí kvůli kontrole nenulovosti vypočítat i num zadaného tnumu a musí také vracet nové epsilon. Aby se tnum nevyčísloval vícekrát, když už jeho hodnotu známe, vrací funkce i tento num.

```
(defun get-nonzero-num+eps (tnum eps)
  (let ((num (tnum-to-num tnum eps)))
    (if (or (zerop num) (<= (abs num) eps))
        (get-nonzero-num+eps tnum (/ eps 10))
      (values num eps))))
```

Lispový kód 9 (`get-nonzero-num+eps`)**:** *Funkce pro nalezení bezpečného epsilonu a numu nenulového tnumu*

Věta 36 (O převráceném tnumu). *Nechť* $\mathsf{t}^x \in \mathcal{T}^x, x \neq 0$ *a funkce* $\mathsf{t}(\mathsf{t}^x)$ *má předpis*

$$\mathsf{t}(\mathsf{t}^x)(\varepsilon) = /\,[\mathsf{t}^x(\varepsilon_\emptyset(\mathsf{t}^x, (\varepsilon * |\mathsf{t}^x(\varepsilon_\emptyset(\mathsf{t}^x, \varepsilon))| * (|\mathsf{t}^x(\varepsilon_\emptyset(\mathsf{t}^x, \varepsilon))| - \varepsilon_\emptyset(\mathsf{t}^x, \varepsilon)))))]\,, \tag{37}$$

pak $\mathsf{t}(\mathsf{t}^x) \in \mathcal{T}^{/x}$.

Důkaz. Podle rovnice (21) platí

$$|\mathsf{t}^x(\varepsilon) - x| \leq \varepsilon, \tag{38}$$

což lze díky lemmatu 35 a předpokladu nenulovosti x přepsat na

$$|\mathsf{t}^x(\varepsilon_\emptyset(\mathsf{t}^x, \varepsilon)) - x| \leq \varepsilon, \tag{39}$$

díky absolutní hodnotě pak platí

$$|x - \mathsf{t}^x(\varepsilon_\emptyset(\mathsf{t}^x, \varepsilon))| \leq \varepsilon. \tag{40}$$

Nerovnici vydělíme kladným číslem $|\mathsf{t}^x(\varepsilon_\emptyset(\mathsf{t}^x, \varepsilon)) * x|$

$$\frac{|x - \mathsf{t}^x(\varepsilon_\emptyset(\mathsf{t}^x, \varepsilon))|}{|\mathsf{t}^x(\varepsilon_\emptyset(\mathsf{t}^x, \varepsilon)) * x|} \leq \frac{\varepsilon}{|\mathsf{t}^x(\varepsilon_\emptyset(\mathsf{t}^x, \varepsilon)) * x|} \tag{41}$$

a protože $|a| * |b| = |a * b|$, po dvojí aplikaci platí

$$\left|\frac{x - \mathfrak{t}^x(\varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon))}{\mathfrak{t}^x(\varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon)) * x}\right| \leq \frac{\varepsilon}{|\mathfrak{t}^x(\varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon))| * |x|} \tag{42}$$

a po roztržení levého výrazu na rozdílné jmenovatele dostáváme

$$\left|\frac{1}{\mathfrak{t}^x(\varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon))} - \frac{1}{x}\right| \leq \frac{\varepsilon}{|\mathfrak{t}^x(\varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon))| * |x|}. \tag{43}$$

Dále díky předpokladu 3 z definice 34 $|x| \geq |\mathfrak{t}^x(\varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon))| - \varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon)$ a proto

$$\left|\frac{1}{\mathfrak{t}^x(\varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon))} - \frac{1}{x}\right| \leq \frac{\varepsilon}{|\mathfrak{t}^x(\varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon))| * (|\mathfrak{t}^x(\varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon))| - \varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon))}, \tag{44}$$

takže po úpravě přesnosti dostáváme

$$\left|\frac{1}{\mathfrak{t}^x(\varepsilon_\emptyset(\mathfrak{t}^x, (\varepsilon * |\mathfrak{t}^x(\varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon))| * (|\mathfrak{t}^x(\varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon))| - \varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon)))))} - \frac{1}{x}\right| \leq \varepsilon. \tag{45}$$

□

```
(defun /tnum (tnum)
  (lambda (eps)
    (multiple-value-bind (num eps0)
        (get-nonzero-num+eps tnum eps)
      (let* ((absnum (abs num))
             (neweps (* eps absnum (- absnum eps0))))
        (/ (if (>= neweps eps) num
             (get-nonzero-num+eps tnum neweps)))))))
```

Lispový kód 10 (`/tnum`)**:** *Funkce pro převrácenou hodnotu tnumu*

Operace `*` bere přirozený počet argumentů. Pro žádný vrací jedničku (neutrální prvek multiplikativní grupy [13]), pro jeden vrací tento a pro více pak jejich součin (doprogramovat zbývá součin).

Věta 37 (O součinu dvou tnumů). *Nechť* $\mathfrak{t}^x \in \mathcal{T}^x, x \in \mathbb{R}$ *a* $\mathfrak{t}^y \in \mathcal{T}^y, y \in \mathbb{R}$ *a funkce* $\mathfrak{t}(\mathfrak{t}^x, \mathfrak{t}^y)$ *má předpis*

$$\mathfrak{t}(\mathfrak{t}^x, \mathfrak{t}^y)(\varepsilon) = \mathfrak{t}^x\left(\frac{\varepsilon}{2 * \max\{(|\mathfrak{t}^y(\varepsilon)| + \varepsilon); 1\}}\right) * \mathfrak{t}^y\left(\frac{\varepsilon}{2 * \max\{(|\mathfrak{t}^x(\varepsilon)| + \varepsilon); 1\}}\right), \tag{46}$$

pak $\mathfrak{t}(\mathfrak{t}^x, \mathfrak{t}^y) \in \mathcal{T}^{x*y}$.

Důkaz. • Nejprve si dokažme nerovnici

$$|x| \leq |\mathsf{t}^x(\varepsilon)| + \varepsilon. \tag{47}$$

Odečtením $|\mathsf{t}^x(\varepsilon)|$ získáváme

$$|x| - |\mathsf{t}^x(\varepsilon)| \leq \varepsilon, \tag{48}$$

což bude platit, pokud dokážeme silnější tvrzení

$$||x| - |\mathsf{t}^x(\varepsilon)|| \leq \varepsilon. \tag{49}$$

To díky trojúhelníkové nerovnosti $||a| - |b|| \leq |a - b|$ lze přepsat na

$$|x - \mathsf{t}^x(\varepsilon)| \leq \varepsilon, \tag{50}$$

v absolutní hodnotě můžeme prohodit sčítance beze změny její hodnoty. Získaný vztah

$$|\mathsf{t}^x(\varepsilon) - x| \leq \varepsilon \tag{51}$$

platí přímo z definice tnumu.

- Nerovnice

$$\frac{|y|}{\max\{(|\mathsf{t}^y(\varepsilon)| + \varepsilon); 1\}} \leq 1 \tag{52}$$

vyplývá z nerovnice (47), ekvivalentně je ji totiž možné zapsat $\frac{|x|}{|\mathsf{t}^x(\varepsilon)|+\varepsilon} \leq 1$ a maximum ve jmenovateli jen tvrzení posiluje.

- Nerovnice

$$\frac{\left|\mathsf{t}^x\left(\frac{\varepsilon}{2*\max\{(|\mathsf{t}^y(\varepsilon)|+\varepsilon);1\}}\right)\right|}{\max\{(|\mathsf{t}^x(\varepsilon)| + \varepsilon); 1\}} \leq 1 \tag{53}$$

vyplývá ze skutečnosti, že $0 \leq \frac{\varepsilon}{2*\max\{(|\mathsf{t}^y(\varepsilon)+\varepsilon);1\}} \leq \varepsilon$, a proto

$$\frac{\left|\mathsf{t}^x\left(\frac{\varepsilon}{2*\max\{(|\mathsf{t}^y(\varepsilon)|+\varepsilon);1\}}\right)\right|}{\max\{(|\mathsf{t}^x(\varepsilon)| + \varepsilon); 1\}} \leq \frac{|\mathsf{t}^x(\varepsilon)| + \varepsilon}{\max\{(|\mathsf{t}^x(\varepsilon)| + \varepsilon); 1\}} \leq 1, \tag{54}$$

což platí.

- Nyní přejděme k důkazu věty. Aby věta platila, musí být splněna nerovnost

$$\left|\mathsf{t}^x\left(\frac{\varepsilon}{2*\max\{(|\mathsf{t}^y(\varepsilon)| + \varepsilon); 1\}}\right) * \mathsf{t}^y\left(\frac{\varepsilon}{2*\max\{(|\mathsf{t}^x(\varepsilon)| + \varepsilon); 1\}}\right) - xy\right| \leq \varepsilon. \tag{55}$$

V dalším kvůli přehlednosti zápisu budu označovat $\varepsilon_x := \frac{\varepsilon}{2*\max\{(|\mathsf{t}^y(\varepsilon)|+\varepsilon);1\}}$ a $\varepsilon_y := \frac{\varepsilon}{2*\max\{(|\mathsf{t}^x(\varepsilon)|+\varepsilon);1\}}$.

Rozepíšeme levou stranu, odečtením a přičtením členu $\mathfrak{t}^x(\varepsilon_x) * y$ dostáváme

$$|\mathfrak{t}^x(\varepsilon_x) * \mathfrak{t}^y(\varepsilon_y) - \mathfrak{t}^x(\varepsilon_x) * y + \mathfrak{t}^x(\varepsilon_x) * y - x * y| \leq \tag{56}$$

a z trojúhelníkové nerovnosti

$$\leq |\mathfrak{t}^x(\varepsilon_x) * \mathfrak{t}^y(\varepsilon_y) - \mathfrak{t}^x(\varepsilon_x) y| + |\mathfrak{t}^x(\varepsilon_x) * y - x * y| \leq \tag{57}$$

a po vytknutí $|\mathfrak{t}^x(\varepsilon_x)|$ z prvních dvou členů a $|y|$ z druhých dvou máme

$$\leq |\mathfrak{t}^x(\varepsilon_x)| * |\mathfrak{t}^y(\varepsilon_y) - y| + |y| * |\mathfrak{t}^x(\varepsilon_x) - x| \leq \tag{58}$$

a po dvojím použití vztahu $|\mathfrak{t}^x(\varepsilon) - x| \leq \varepsilon$ získáváme

$$\begin{aligned} \leq \frac{\varepsilon}{2} * |\mathfrak{t}^x(\varepsilon_x)| * \left| \frac{1}{\max\{(|\mathfrak{t}^x(\varepsilon)| + \varepsilon); 1\}} \right| + \\ + \frac{\varepsilon}{2} * |y| * \left| \frac{1}{\max\{(|\mathfrak{t}^y(\varepsilon)| + \varepsilon); 1\}} \right| = \end{aligned} \tag{59}$$

a zjednodušíme-li zápis pomocí zlomku, dostáváme

$$= \frac{\varepsilon}{2} * \frac{|\mathfrak{t}^x(\varepsilon_x)|}{\max\{(|\mathfrak{t}^x(\varepsilon)| + \varepsilon); 1\}} + \frac{\varepsilon}{2} * \frac{|y|}{\max\{(|\mathfrak{t}^y(\varepsilon)| + \varepsilon); 1\}} \leq \tag{60}$$

a díky dokázaným nerovnostem (52) a (53) pak platí

$$\leq \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon. \tag{61}$$

□

Právě dokázaná věta mluví o součinu dvou tnumů. Zobecnění na konečný počet tnumů by v tomto bodě šlo naprogramovat akumulací. To je ale neefektivní řešení, a proto by bylo dobré najít obecnou funkci. Následující vztah není dokázaný, vychází však z tvaru násobení pro dva tnumy.

Hypotéza 38 (Součin tnumů). *Nechť* $\mathfrak{t}^{x_i} \in \mathcal{T}^{x_i}, x_i \in \mathbb{R}$ *pro* $i = 0, 1, \ldots, n$ *a funkce* $\mathfrak{t}(\mathfrak{t}^{x_0}, \mathfrak{t}^{x_1}, \ldots, \mathfrak{t}^{x_n})$ *má předpis*

$$\mathfrak{t}(\mathfrak{t}^{x_0}, \mathfrak{t}^{x_1}, \ldots, \mathfrak{t}^{x_n})(\varepsilon) = \prod_{i=0}^{n} \mathfrak{t}^{x_i} \left(\frac{\varepsilon}{(n+1) * \max\left\{\prod_{j=0, i \neq j}^{n} (|\mathfrak{t}^{x_j}(\varepsilon)| + \varepsilon); 1\right\}} \right), \tag{62}$$

pak $\mathfrak{t}(\mathfrak{t}^{x_0}, \mathfrak{t}^{x_1}, \ldots, \mathfrak{t}^{x_n}) \in \mathcal{T}^{\prod_{i=0}^{n} x_i}$.

Hypotéza mluví o nenulových číslech. Nesnižujeme ale obecnost, protože nula je agresivní prvek a výsledkem násobení čehokoli s nulou je nula, takže se ostatní numy ani nemusejí počítat a výsledek se může vrátit.

Samotná implementace pak využívá pomocnou mapovací funkci.

```lisp
(defun create-list-for-multiplication (tnums eps)
  (let* ((result nil) (len (list-length tnums)) (el (/ eps len))
         (nums (mapcar (lambda (tnum)
                         (tnum-to-num tnum el)) tnums)))
    (dotimes (i len result)
      (let ((neps 1))
        (dotimes (j len)
          (unless (= i j)
            (setf neps (/ neps (+ (abs (nth j nums)) eps)))))
        (setf result (cons
                      (if (<= neps 1) (nth i nums)
                        (tnum-to-num (nth i tnums)
                                     (/ el (max neps 1))))
                      result))))))
```

Lispový kód 11 (`create-list-for-multiplication`): *Pomocná funkce pro násobení*

```lisp
(defun tnum* (&rest tnums)
  (if (null tnums)
      (num-to-tnum 1)
    (lambda (eps)
      (apply '* (create-list-for-multiplication tnums eps)))))
```

Lispový kód 12 (`tnum*`): *Funkce pro násobení tnumů*

Operace `/` vyžaduje alespoň jeden argument, v případě zadání pouze tohoto vrací převrácený tnum, v případě více pak jejich podíl (zbývá doprogramovat podíl).

Fakt 39 (Podíl tnumů). *Nechť* $\mathfrak{t}^{x_i} \in \mathcal{T}^{x_i}, x_i \in \mathbb{R} \setminus \{0\}$ *pro* $i \in \{0, 1, \ldots, n\}$ *a* $x_i \in \mathbb{R}$ *pro* $i = -1$ *a funkce* $\mathfrak{t}(\mathfrak{t}^{x_{-1}}, \mathfrak{t}^{x_0}, \ldots, \mathfrak{t}^{x_n})$ *má předpis*

$$\mathfrak{t}(\mathfrak{t}^{x_{-1}}, \mathfrak{t}^{x_0}, \ldots, \mathfrak{t}^{x_n}) = \mathfrak{t}^{x_{-1}/\prod_{i=0}^{n} x_i} = \mathfrak{t}^{x_{-1}*\left(/\prod_{i=0}^{n} x_i\right)}, \tag{63}$$

pak $\mathfrak{t}(\mathfrak{t}^{x_{-1}}, \mathfrak{t}^{x_0}, \ldots, \mathfrak{t}^{x_n}) \in \mathcal{T}^{x_{-1}/x_0/\ldots/x_n}$.

```lisp
(defun tnum/ (tnum1 &rest tnums)
  (if (null tnums)
      (/tnum tnum1)
    (tnum* tnum1 (/tnum (apply 'tnum* tnums)))))
```

Lispový kód 13 (`tnum/`): *Funkce pro dělení tnumů*

Pro tnumy $a, b \in \mathfrak{T}$ existuje tnum $a + b$ a $a * b$. Množina $\mathfrak{T}$ je tedy uzavřena na operace `+` a `*`. Tnumy pak i z pohledu strukturální algebry dobře reprezentují rekurzivní čísla, ta jsou totiž číselným tělesem [33].

4.3 Mocninné operace

K implementaci mocninných operací potřebujeme funkce, které přidáme až v další kapitole. Uveďme zde alespoň vztahy, podle kterých lze mocninné operace naprogramovat.

Obecná mocnina využívá přirozený logaritmus a exponenciálu

$$a^b = e^{\ln(a)^b} = e^{(b*\ln(a))}, a > 0. \quad (64)$$

Odmocninu lze vyjádřit pomocí převrácení hodnoty mocniny

$$\sqrt[b]{a} = a^{(/b)}, b \neq 0. \quad (65)$$

5 Funkce tnumů

Knihovna `tnums` nyní implementuje racionální čísla, Ludolfovo číslo, aditivní a multiplikativní operace. Dobrým zdrojem dalších iracionálních čísel, jak jsem napsal již v podkapitole 1.4, jsou matematické funkce. V této kapitole zavedeme exponenciálu, logaritmus, goniometrické funkce a mocninné operace.

5.1 Aproximace funkcí

Stejně jako matematické funkce berou reálné číslo a vrací reálné číslo, musejí tnumovské funkce přijímat tnumy a vracet tnumy. Nástroj, který poskytuje vyčíslování funkcí i odhad chyby vyčíslení se nazývá Taylorův polynom (TP). Ten z bodu a, kterému říkáme počátek, odhaduje průběh funkce pomocí polynomů určitého stupně. Pro funkci f budu TP stupně n se středem v a značit $T_n^{f,a}$. K implementaci TP využijeme funkci faktoriálu přirozeného čísla.

```
(defun factorial (n)
  (let ((result 1))
    (loop for i from n downto 1
          do (setf result (* result i)))
    result))
```

Lispový kód 14 (`factorial`)**:** *Funkce pro výpočet faktoriálu v iterativní verzi, kvůli přetékání zásobníku při velkých vstupech není použita rekurze*

TP počítá aproximaci funkce f v bodě x následovně: nejjednodušší je vzít funkci $T_0^{f,a}(x) = f(a)$. Přesněji funkci v daném bodě reprezentuje TP prvního stupně – přímka, která se dotýká grafu funkce f v bodě a: $T_1^{f,a}(x) = f(a) + f'(a)(x - a)$. Tato aproximace nebere v úvahu jen hodnotu funkce f v bodě a, ale i její první derivaci – směr, kam se z počátku pohybuje. Ještě přesnější je TP druhého stupně – parabola přimknutá k grafu funkce f v bodě a: $T_2^{f,a}(x) =$

$f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2$. Teď už TP zohledňuje funkční hodnotu, směr křivky i konvexnost. Ještě lepší je použít $T_3^{f,a}(x) = f(a)+f'(a)(x-a)+\frac{f''(a)}{2}(x-a)^2 + \frac{f'''(a)}{6}(x-a)^3$ a tak dále [34].

Připomenutí 40 (Taylorova a Maclaurinova řada). Kdybychom takto postupovali donekonečna – $\lim_{n\to\infty} T_n^{f,a}(x)$ – dostali bychom Taylorovu řadu z definice 17. Pro $a = 0$ pak Taylorovu řadu nazýváme řadou Maclaurinovou.

Pokud Taylorovy zbytky konvergují k nule, lze Taylorovou řadou $T_\infty^{f,a}$ nahradit funkci f [15]. My pro výpočty s libovolnou přesností navíc potřebujeme velikosti zbytků odhadovat.

Fakt 41 (Taylorova věta [35]). *Nechť f má spojité derivace až do řádu $n+1$ na libovolném intervalu obsahujícím a. Pak pro každé x z tohoto intervalu máme Taylorův vzorec*

$$f(x) = T_n^{f,a}(x) + R_n^{f,a}(x), \text{ kde} \tag{66}$$

$$T_n^{f,a}(x) = \sum_{i=0}^{n} \frac{f^{(i)}(a)}{i!}(x-a)^i \text{ a také} \tag{67}$$

$$R_n^{f,a}(x) = \int_a^x \frac{(x-t)^n}{n!} f^{(n+1)}(t)dt. \tag{68}$$

Navíc existuje číslo ξ, z intervalu s krajními body x a a takové, že

$$R_n^{f,a}(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-a)^{n+1}. \tag{69}$$

Pro důkaz vizte kapitolu 7.5 v [35].

Součet v rovnici (67) nazýváme Taylorův polynom funkce f stupně n v bodě a, $R_n^{f,a}(x)$ nazýváme n-tým Taylorovým zbytkem. Vyjádření (68) pak říkáme *integrální tvar* zbytku a (69) je Lagrangeův tvar zbytku [34].

Takto vyjádřené zbytky můžeme zhora omezovat, podobně jako tomu bylo u geometrické řady.

5.2 Exponenciála

Exponenciála je funkce s předpisem $\exp(x) = e^x$ kde e je tzv. *Eulerovo číslo* definované jako $e = \lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n$ [12]. Eulerovo číslo je transcendentní konstanta, je též základem přirozeného logaritmu. Exponenciála je z tohoto důvodu nazývána též přirozená mocnina. Platí: $\frac{d}{dx}\exp(x) = \exp(x), D(\exp) = \mathbb{R}, H(\exp) = \mathbb{R}^+$.

Poznámka 42 (Značení exponenciály). Ačkoli to působí nezvykle, značím exponenciálu jako exp, protože potřebuji vyjadřovat i název funkce, nikoli jen její hodnotu a pro to se nehodí obvyklejší zápis pomocí e^x. Navíc zápis $\exp(x)$ pro závisle proměnnou lépe vyjadřuje, že je exponenciála funkcí. ■

5.2.1 Exponenciála čísla

Fakt 43 (Exponenciála jako Maclaurinova řada [15]). *Funkci* exp *lze vyjádřit jako Maclaurinovu řadu ve tvaru*

$$\exp(x) = \sum_{i\in\mathbb{N}} \frac{x^i}{i!} = \frac{1}{1} + \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \tag{70}$$

Zbytek této řady vyjádřený v Lagrangeově tvaru je pro neznámé ξ v intervalu s krajními body 0 a x roven

$$R_n^{exp,0}(x) = \frac{e^\xi}{(n+1)!} x^{n+1}. \tag{71}$$

Exponenciála je funkce rostoucí a $e < 2.72$, a platí tedy

$$(\forall \xi \in (0,x))(\exp(\xi) < \exp(x) < 2.72^x). \tag{72}$$

Získáváme odhad chyby aproximace exponenciály pomocí TP.

Fakt 44 (Omezení Taylorova zbytku exponenciály). *Pro* $x \in \mathbb{R}$ *platí*

$$|R_n^{exp,0}(x)| = \left|\exp(x) - \sum_{i=0}^{n} \frac{x^i}{i!}\right| \leq \left|\frac{2.72^x}{(n+1)!} x^{n+1}\right|. \tag{73}$$

Důsledek 45 (Exponenciála numu). *Nechť* $x \in \mathbb{Q}$ *a funkce* $\mathtt{t}(x)$ *má předpis*

$$\mathtt{t}(x)(\varepsilon) = \left[\begin{array}{l} 1.\ \textit{Najdi nejmenší } n \in \mathbb{N}^+ \textit{ tak, aby } \left|\frac{2.72^x}{(n+1)!}x^{n+1}\right| \leq \varepsilon; \\ 2.\ \textit{Vrať } \sum_{i=0}^{n} \frac{x^i}{i!}, \end{array}\right. \tag{74}$$

pak $\mathtt{t}(x) \in \mathcal{T}^{\exp(x)}, x \in \mathbb{Q}$.

Důkaz. Existence čísla n je zřejmá z definice limity posloupnosti a z toho, že limita podílu polynomu a faktoriálu je rovna nule.

Dále protože $\exp(x) = T_n^{exp,0}(x) + R_n^{exp,0}(x)$, lze psát

$$T_n^{exp,0}(x) \in [\exp(x) - |R_n^{exp,0}(x)|, \exp(x) + |R_n^{exp,0}(x)|], \tag{75}$$

přičemž dle předchozího faktu platí

$$T_n^{exp,0}(x) \in \left[\exp(x) - \left|\frac{2.72^x}{(n+1)!}x^{n+1}\right|, \exp(x) + \left|\frac{2.72^x}{(n+1)!}x^{n+1}\right|\right] \tag{76}$$

a z kroku 1. pak

$$T_n^{exp,0}(x) \in [\exp(x) - \varepsilon, \exp(x) + \varepsilon]. \tag{77}$$

□

Při implementaci stačí jen iterovat přes n a po cestě sčítat řadu, dokud nebude zbytek menší než ε. Stejný přístup jsme viděli již u Ludolfova čísla.

```
(defun num-exp (num eps)
  (let ((above (rat-expt 272/100 num)) (result 0))
    (loop for n from 0
          for nfact = (factorial n)
          for xpown = (expt num n)
          do (incf result (/ xpown nfact))
          until (<= (/ (* above xpown) nfact) eps)
          finally (return result))))
```

Lispový kód 15 (`num-exp`): *Funkce pro exponenciálu numu*

Poznámka 46 (Eulerovo číslo jako exponenciála čísla jedna). Protože $e = e^1$, lze přidat i tnum Eulerova čísla.

```
(defun tnum-e ()
  (lambda (eps)
    (num-exp 1 eps)))
```

Lispový kód 16 (`tnum-e`): *Funkce pro tnum Eulerova čísla* ■

Do knihovny přibyl tnum exponenciály čísla, lze tedy napsat

$$(\texttt{num-exp}\ \ q) \in \mathcal{T}^{\exp(q)}, q \in \mathbb{Q}. \tag{78}$$

5.2.2 Exponenciála tnumu

I při implementaci exponenciály tnumu vyvstává rozdíl mezi přístupem k racionálním číslům a číslům rekurzivním.

Víme, že $(\forall \mathbf{t}^x \in \mathcal{T}^x)(\mathbf{t}^x(\varepsilon) \in [x-\varepsilon, x+\varepsilon])$. Na Obrázku 11 je znázorněno, že $[\exp(x)-\varepsilon, \exp(x)+\varepsilon] \neq [\exp(x-\varepsilon), \exp(x+\varepsilon)]$.

Dále na Obrázku 12 vidíme, že abychom dodrželi přesnost závisle proměnné, musíme manipulovat s přesností nezávisle proměnné. Závislost můžeme pospat rovnicí

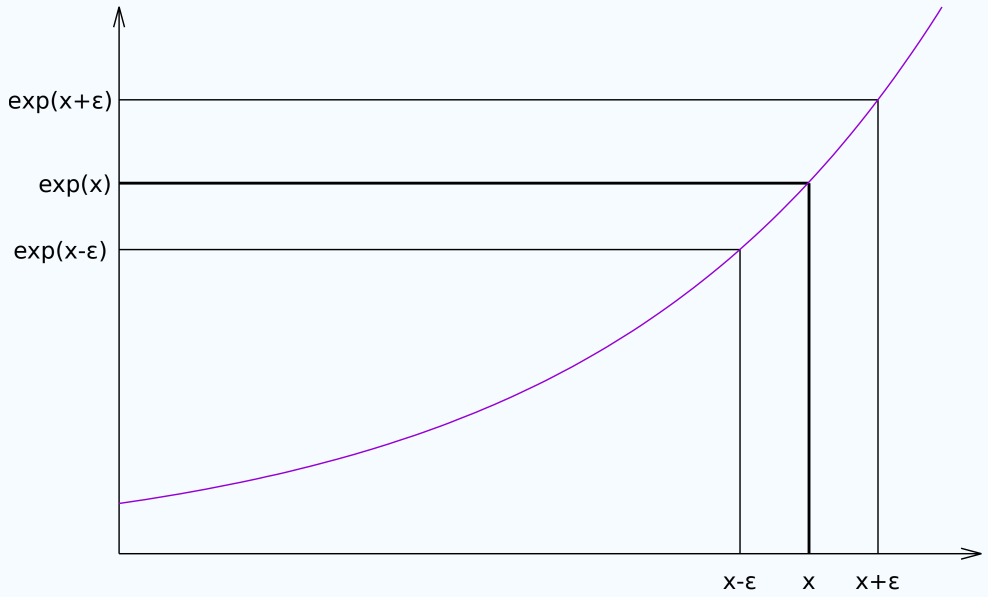
$$\exp(x) + \varepsilon = \exp(x+w), \tag{79}$$

kde hledaná neznámá je w. Ilustraci tohoto vztahu najdeme na Obrázku 13.

Je třeba najít vztah mezi w a ε. Pro úhel α při vrcholu W platí $\operatorname{ctan}(\alpha) = \frac{\varepsilon}{w}$. Tedy

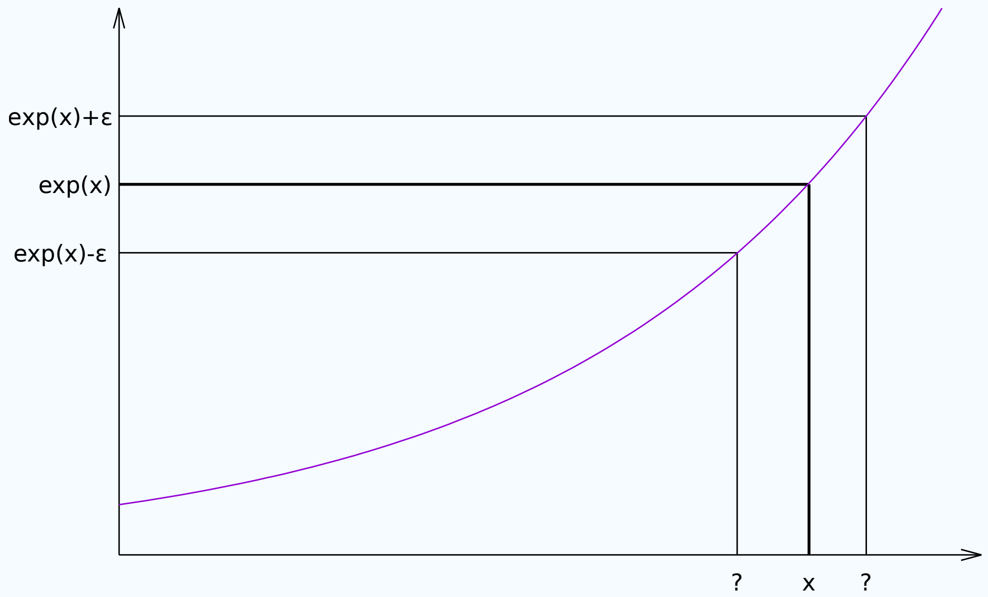
$$w = \frac{\varepsilon}{\operatorname{ctan}(\alpha)} = \frac{\varepsilon}{\tan(\frac{\pi}{2}-\alpha)} = \frac{\varepsilon}{exp(x+w)}. \tag{80}$$

Poslední úprava je přepis skutečnosti, že tangens je v tomto bodě roven derivaci a derivace exponenciály je exponenciála. Vychází rekurzivní vztah pro w, Obrázek 11: Obraz přesnosti po průchodu exponenciálou



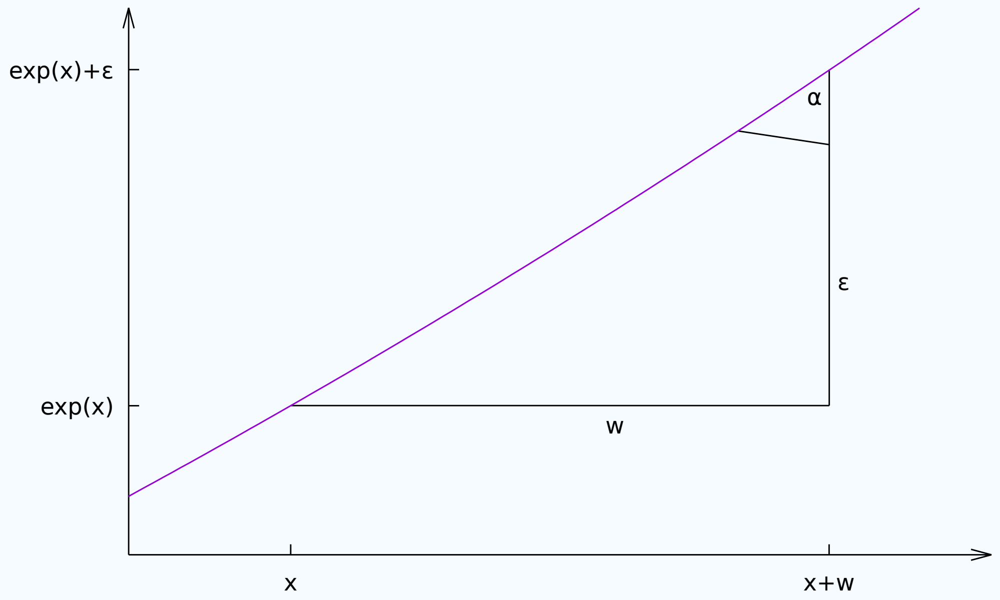
Interval $[x-\varepsilon, x+\varepsilon]$ se nezobrazí na $[e^x-\varepsilon, e^x+\varepsilon]$. Tím pádem $\mathfrak{t}^{\exp(\mathfrak{t}^x(\varepsilon))} \notin \mathcal{T}^{\exp(x)}, x \in \mathbb{R}$.

Obrázek 12: Vzor přesnosti před průchodem exponenciálou



Hledáme, jaké okolí bodu x se zobrazí na ε-okolí bodu e^x.

Obrázek 13: Zobrazení neznámé w



Neznámou w lze zobrazit jako jednu z odvěsen, druhá je ε. Fialová čára není přepona, ale exponenciála. Protože se ale ε může neomezeně zmenšovat, exponenciála se k přeponě blíží.

po jehož dosazení do vztahu (79) dostáváme

$$\exp(x) + \varepsilon = \exp\left(x + \frac{\varepsilon}{\exp\left(x + \frac{\varepsilon}{\exp(x+\ldots)}\right)}\right). \tag{81}$$

Podobným způsobem lze odvodit i vztah pro opačný kraj okolí a to

$$\exp(x) - \varepsilon = \exp\left(x - \frac{\varepsilon}{\exp\left(x - \frac{\varepsilon}{\exp(x-\ldots)}\right)}\right). \tag{82}$$

Poznámka 47 (Obecnější přesnost vzoru). Právě odvozený vztah není specifický jen pro exponenciálu, uveďme si obecnější znění.

Fakt 48 (Přesnost závisle proměnné). *Nechť f je neklesající funkce spojitá na celém definičním oboru, pak*

$$\begin{gathered}[f(x) - \varepsilon, f(x) + \varepsilon] = [f(x - v), f(x + w)],\\ \textit{kde } v = f\left(x - \frac{\varepsilon}{f'(x - v)}\right), w = f\left(x + \frac{\varepsilon}{f'(x + w)}\right).\end{gathered} \tag{83}$$ ■

Z výše uvedených vztahů je jasné, že při implementaci budeme hledat pevný bod. Proto si zavedeme tzv. *precizní iterátor.*

Definice 49 (Precizní iterátor exponenciály). *Nechť* $\mathfrak{t}^{\exp(q)} \in \mathcal{T}^{\exp(q)}, q \in \mathbb{Q}$. *Pak definujme následující posloupnost:*

$$[\mathfrak{t}^x]_0^{\exp,\varepsilon} = \mathfrak{t}^{\exp(\mathfrak{t}^x(\varepsilon))}(\varepsilon), \tag{84}$$

$$[\mathfrak{t}^x]_{n+1}^{\exp,\varepsilon} = \mathfrak{t}^{\exp\left(\mathfrak{t}^x\left(\frac{\varepsilon}{|[\mathfrak{t}^x]_n^{\exp,\varepsilon}|+\varepsilon}\right)\right)}(\varepsilon) \tag{85}$$

a pokud existuje m tak, že

$$[\mathfrak{t}^x]_m^{\exp,\varepsilon} = [\mathfrak{t}^x]_{m+1}^{\exp,\varepsilon}, \tag{86}$$

pak klademe

$$[\mathfrak{t}^x]_\infty^{\exp,\varepsilon} := [\mathfrak{t}^x]_m^{\exp,\varepsilon}. \tag{87}$$

Důsledek 50 (Exponenciála tnumu). *Nechť* $\mathfrak{t}^x \in \mathcal{T}^x, x \in \mathbb{R}$ *a funkce* $\mathfrak{t}(\mathfrak{t}^x)$ *má předpis*

$$\mathfrak{t}(\mathfrak{t}^x)(\varepsilon) = [\mathfrak{t}^x]_\infty^{exp,\varepsilon}, \tag{88}$$

pak $\mathfrak{t}(\mathfrak{t}^x) \in \mathcal{T}^{\exp(x)}, x \in \mathbb{R}$.

Důkaz. Vychází přímo ze vztahů (81) a (82). □

```lisp
(defun tnum-exp (tnum)
  (lambda (eps)
    (loop for i from 0
          for num = (if (zerop i) (tnum-to-num tnum eps) new)
          for new = (if (zerop i) 1
                        (if (> expnum 1)
                            (tnum-to-num tnum (/ eps
                                                 (+ expnum eps)))
                          num))
          for expnum = (num-exp num eps)
          until (= num new)
          finally (return expnum))))
```

Lispový kód 17 (`tnum-exp`): *Funkce pro exponenciálu tnumu*

5.3 Goniometrické

Goniometrické funkce jsou reálné funkce reálné proměnné. Platí $\frac{d}{dx}\sin(x) = \cos(x)$, $\frac{d}{dx}\cos(x) = -\sin(x)$ a $H(\sin) = [-1, 1] = H(\cos)$.

5.3.1 Sinus

Fakt 51 (Sinus jako Maclaurinova řada [15]). *Funkci* sin *lze vyjádřit jako Maclaurinovu řadu ve tvaru*

$$\sin(x) = \sum_{i \in \mathbb{N}} (-1)^i \frac{x^{2i+1}}{(2i+1)!} = \frac{x}{1} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots \tag{89}$$

Dále protože jsou funkční hodnoty všech možných derivací v intervalu $[-1, 1]$, lze Lagrangeův tvar zbytku vyjádřit bez znaménka, a pak díky Taylorově větě platí

$$\left|\mathcal{R}_n^{sin,0}(x)\right| \leq \left|\frac{x^{2n+1+1}}{(2n+1+1)!}\right| = \left|\frac{x^{2n+2}}{(2n+2)!}\right|. \tag{90}$$

Důsledek 52 (Sinus numu). *Nechť* $x \in \mathbb{Q}$ *a funkce* $\mathfrak{t}(x)$ *má předpis*

$$\mathfrak{t}(x)(\varepsilon) = \left[\begin{array}{l} 1.\ \textit{Najdi nejmenší } n \in \mathbb{N}^+ \textit{ tak, aby } \left|\frac{x^{2n+2}}{(2n+2)!}\right| \leq \varepsilon; \\ 2.\ \textit{Vrať } \sum_{i=0}^{n} (-1)^i \frac{x^{2i+1}}{(2i+1)!}, \end{array}\right. \tag{91}$$

pak $\mathfrak{t}(x) \in \mathcal{T}^{\sin(x)}, x \in \mathbb{Q}$.

Důkaz. Jde opět o polynom nad faktoriálem, proto je vidět limita i existence n. Z omezení $R_n^{\sin,0}$ je zřejmé, že pokud platí $T_n^{\sin,0} \in [\sin(x) - |R_n^{\sin,0}|, \sin(x) + |R_n^{\sin,0}|]$, pak musí platit i $T_n^{\sin,0} \in [\sin(x) - |\frac{x^{2n+2}}{(2n+2)!}|, \sin(x) + |\frac{x^{2n+2}}{(2n+2)!}|]$, a tudíž $T_n^{\sin,0} \in [\sin(x) - \varepsilon, \sin(x) + \varepsilon]$. □

```lisp
(defun num-sin (x eps)
  (let ((result 0))
    (loop for n from 0
          for 2n+1 = (1+ (* 2 n))
          do (incf result
                   (/ (rat-expt x 2n+1)
                      (factorial 2n+1) (expt -1 n)))
          until (< (abs (/ (rat-expt x (1+ 2n+1))
                           (factorial (1+ 2n+1))))
                   eps)
          finally (return result))))
```

Lispový kód 18 (num-sin)**:** *Funkce pro sinus čísla*

Protože derivace sinu je kosinus, který nabývá hodnot mezi -1 a 1, nebude nutné provádět korekce přesnosti podle funkční hodnoty derivace. Tato skutečnost vede na jednoduchý vztah.

Lemma 53 (O sinu tnumu). *Nechť* $\mathfrak{t}^x \in \mathcal{T}^x, x \in \mathbb{R}$ *a* $\mathfrak{t}^{\sin(q)} \in \mathcal{T}^{\sin(q)}, q \in \mathbb{Q}$ *a funkce* $\mathfrak{t}(\mathfrak{t}^x)$ *má předpis*

$$\mathfrak{t}(\mathfrak{t}^x)(\varepsilon) = \mathfrak{t}^{sin(\mathfrak{t}^x(\varepsilon))}(\varepsilon), \tag{92}$$

pak $\mathfrak{t}(\mathfrak{t}^x) \in \mathcal{T}^{\sin(x)}, x \in \mathbb{R}$.

Důkaz. Plyne nepřímo z faktu 48 a z omezení absolutních funkčních hodnot kosinu jedničkou. Fakt nelze použít doslovně, protože kosinus není neklesající funkce, ale z argumentu o omezení funkčních hodnot plyne, že přesný tvar vzoru přesnosti a tudíž ani bod derivace hledat nemusíme. □

```
(defun tnum-sin (tnum)
  (lambda (eps)
    (num-sin (tnum-to-num tnum eps) eps)))
```

Lispový kód 19 (`tnum-sin`)**:** *Funkce pro sinus tnumu*

5.3.2 Kosinus

Fakt 54 (Kosinus jako Maclaurinova řada [15]). *Funkci* $\cos(x)$ *lze vyjádřit jako Maclaurinovu řadu ve tvaru*

$$\cos(x) = \sum_{i \in \mathbb{N}} (-1)^i \frac{x^{2i}}{(2i)!} = \frac{1}{1} - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots \tag{93}$$

Z Taylorovy věty získáváme omezení Taylorova zbytku

$$|R_n^{cos}(x)| \leq \left| \frac{x^{2n+1}}{(2n+1)!} \right|, \tag{94}$$

a proto opět hledáme takové n, že když pro jakékoli ε je $|\frac{x^{2n+1}}{(2n+1)!}| \leq \varepsilon$, pak vrátíme n-tý částečný součet řady z rovnice (93).

Důsledek 55 (Kosinus numu). *Nechť* $x \in \mathbb{Q}$ *a funkce* $\mathfrak{t}(x)$ *má předpis*

$$\mathfrak{t}(x)(\varepsilon) = \left[\begin{array}{l} 1.\ \textit{Najdi nejmenší } n \textit{ tak, aby } \left|\frac{x^{2n+1}}{(2n+1)!}\right| \leq \varepsilon; \\ 2.\ \textit{Vrať } \sum_{i=0}^{n} (-1)^i \frac{x^{2i}}{(2i)!}, \end{array} \right. , \tag{95}$$

pak $\mathfrak{t}(x) \in \mathcal{T}^{\cos(x)}, x \in \mathbb{Q}$.

```lisp
(defun num-cos (x eps)
  (let ((result 0))
    (loop for n from 0
          for 2n = (* 2 n)
          do (incf result
                   (/ (rat-expt x 2n)
                      (factorial 2n)
                      (expt -1 n)))
          until (< (abs (/ (rat-expt x (1+ 2n))
                           (factorial (1+ 2n))))
                   eps)
          finally (return result))))
```

Lispový kód 20 (`num-cos`): *Funkce pro výpočet kosinu čísla*

A nakonec právě naprogramovanou funkci využijeme ke kosinování jakéhokoli tnumu. Postup je stejný jako u sinu, a proto už nepíši příslušné lemma.

```lisp
(defun tnum-cos (tnum)
  (lambda (eps)
    (num-cos (tnum-to-num tnum eps) eps)))
```

Lispový kód 21 (`tnum-cos`): *Funkce pro výpočet kosinu tnumu*

Zbylé goniometrické funkce už naprogramujeme uživatelsky.

5.3.3 Další goniometrické funkce

Další goniometrickou funkcí je tangens. Lze ho vyjádřit pomocí sinu a kosinu, díky čemuž ho nemusíme vyjadřovat jako řadu. Pro všechny goniometrické funkce řady existují, nejsou ale konvergentní na celém definičním oboru, takže se pro naši knihovnu nehodí.

Fakt 56 (Tangens jako poměr sinu a kosinu [7]).

$$\operatorname{tg}(x) = \frac{\sin(x)}{\cos(x)} \tag{96}$$

Úmluva 57 (O vypuštění některých důsledků). Nyní by měl následovat důsledek že $\mathfrak{t}(\mathfrak{t}^x) = \mathfrak{t}^{\mathfrak{t}^{\sin(x)}/\mathfrak{t}^{\cos(x)}} \in \mathcal{T}^{\tan(x)}, x \in \mathbb{R} \setminus \{\frac{\pi}{2} + k\pi | k \in \mathbb{Z}\}$, což je zřejmé, a proto zde ani ve zbytku kapitoly nejsou tyto důsledky uvedeny. ■

```
(defun tnum-tan (tnum)
  (tnum/ (tnum-sin tnum) (tnum-cos tnum)))
```

Lispový kód 22 (`tnum-tan`)**:** *Funkce pro výpočet tangentu tnumu*

Zbylé funkce jsou obrácenou hodnotou již napsaných.

Fakt 58 (Kosekans jako obrácená hodnota sinu [7]).

$$\csc(x) = \sin^{-1}(x) \tag{97}$$

```
(defun tnum-csc (tnum)
  (/tnum (tnum-sin tnum)))
```

Lispový kód 23 (`tnum-csc`)**:** *Funkce pro výpočet kosekantu tnumu*

Fakt 59 (Sekans jako obrácená hodnota kosinu [7]).

$$\sec(x) = \cos^{-1}(x) \tag{98}$$

```
(defun tnum-sec (tnum)
  (/tnum (tnum-cos tnum)))
```

Lispový kód 24 (`tnum-sec`)**:** *Funkce pro výpočet sekantu tnumu*

Fakt 60 (Kotangens jako obrácená hodnota tangentu [7]).

$$\operatorname{cotg}(x) = tan^{-1}(x) = \frac{\cos(x)}{\sin(x)} \tag{99}$$

```
(defun tnum-ctan (tnum)
  (tnum/ (tnum-cos tnum) (tnum-sin tnum)))
```

Lispový kód 25 (`tnum-ctan`)**:** *Funkce pro výpočet kotangentu tnumu*

5.4 Logaritmus

Logaritmus je inverzní funkce k exponenciále a je opět vyjadřitelná řadou.

Fakt 61 (Logaritmus jako řada [36]). *Pro $x \in \mathbb{R}^+$ platí*

$$\ln(x) = 2\sum_{i\in\mathbb{N}} \frac{1}{2i+1}\left(\frac{x-1}{x+1}\right)^{2i+1}. \tag{100}$$

Člen $\frac{1}{2i+1}(\frac{x-1}{x+1})^{2i+1}$ je menší roven $(\frac{x-1}{x+1})^{2i+1}$ a tento je menší než $(\frac{x-1}{x+1})^{2i}$. Toto je geometrická posloupnost, jejíž n-tý zbytek je roven $\frac{(\frac{x-1}{x+1})^{2n+2}}{1-\frac{x-1}{x+1}}$ podle faktu 20.

Fakt 62 (Logaritmus numu). *Nechť $x \in \mathbb{R}^+$ a nechť funkce $\mathfrak{t}(x)$ má předpis*

$$\mathfrak{t}(x)(\varepsilon) = \left[\begin{array}{l} 1.\ \textit{Najdi nejmenší } n \in \mathbb{N}^+ \textit{ tak, aby } \left| \frac{(\frac{x-1}{x+1})^{2n+2}}{1-\frac{x-1}{x+1}} \right| \leq \varepsilon; \\ 2.\ \textit{Vrať } \sum_{i=0}^{n} \frac{1}{2i+1} \left(\frac{x-1}{x+1}\right)^{2i+1}, \end{array} \right. \tag{101}$$

pak $\mathfrak{t}(x) \in \mathcal{T}^{\ln(x)}$.

```
(defun num-ln (x eps)
  (setf eps (/ eps 2))
  (let ((n 0) (result 0) (q (/ (1- x) (1+ x))))
    (loop
     do (progn
          (incf result
                (/ (expt q (1+ (* 2 n))) (1+ (* 2 n))))
          (incf n))
     until (<= (abs (/ (expt q (* 2 (1+ n))) (- 1 q)))
               eps)
     finally (return (* 2 result)))))
```

Lispový kód 26 (`num-ln`): *Funkce pro logaritmus čísla*

Tím bychom měli přirozený logaritmus pro čísla. Podívejme se nyní, jak vypadá omezení nezávislé proměnné pro logaritmus. Po dosazení do vztahu (83) získáváme

$$\ln(x) + \varepsilon = \ln\left(x + \frac{\varepsilon}{\ln'(x+w)}\right) \tag{102}$$

a protože $\ln'(x+w) = (x+w)^{-1}$, pak

$$\ln(x) + \varepsilon = \ln(x + \varepsilon(x+w)), \tag{103}$$

tedy násobíme přesnost hodnotou proměnné a protože přesnost nemůže být nulová, použijeme k vyčíslení $\mathfrak{t}^x$ nenulový tnum, a tím je problém vyřešen. Vezmeme totiž pesimistický odhad $\mathfrak{t}^x(\varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon)) - \varepsilon_\emptyset(\mathfrak{t}^x, \varepsilon)$ a přenásobíme jím epsilon. Lze to udělat díky třetí podmínce v definici 34.

Fakt 63 (Logaritmus tnumu). *Nechť $\mathfrak{t}^x \in \mathcal{T}^x, x \in \mathbb{R}$ a $\mathfrak{t}^{\ln(q)} \in \mathcal{T}^{\ln(q)}, q \in \mathbb{Q}$ a funkce $\mathfrak{t}(\mathfrak{t}^x)$ má předpis*

$$\mathfrak{t}(\mathfrak{t}^x)(\varepsilon) = \mathfrak{t}^{\ln(\mathfrak{t}^x(\varepsilon(\mathfrak{t}^x(\varepsilon_\emptyset(\mathfrak{t}^x,\varepsilon)) - \varepsilon_\emptyset(\mathfrak{t}^x,\varepsilon))))}(\varepsilon), \tag{104}$$

pak $\mathfrak{t}(\mathfrak{t}^x) \in \mathcal{T}^{\ln(x)}, x \in \mathbb{R}$.

```
(defun tnum-ln (tnum)
  (lambda (eps)
    (multiple-value-bind (num eps0)
        (get-nonzero-num+eps tnum eps)
      (num-ln (tnum-to-num tnum eps) (* eps (- num eps0))))))
```

Lispový kód 27 (`tnum-ln`)**:** *Funkce pro logaritmus tnumu*

Poznámka 64 (Dokončení systému). Pomocí logaritmu v kombinaci s exponenciálou lze přinést i mocninné operace, čímž dojde k ucelení základní funkcionality knihovny `tnums`. Při konstrukci mocniny vyjdeme z rovnice (64).

```
(defun tnum-expt (tnum1 tnum2)
  (tnum-exp (tnum* tnum2 (tnum-ln tnum1))))
```

Lispový kód 28 (`tnum-expt`)**:** *Funkce pro umonování tnumů*

Při implementaci odmocniny pak vyjdeme z rovnice (65). Oproti mocnině je zde pořadí argumentů opačné, vycházíme totiž z pořadí při slovní reprezentaci odmocniny, například „třetí odmocnina z devíti“.

```
(defun tnum-root (tnum1 tnum2)
  (tnum-expt tnum2 (/tnum tnum1)))
```

Lispový kód 29 (`tnum-root`)**:** *Funkce pro odmocňování tnumu* ■

Část III
Rozhraní

V poslední části pojednáme o uživatelském pohledu na knihovnu `tnums` – používání (převody, konstanty, operace, funkce), rychlost a uživatelskou rozšiřitelnost.

6 Uživatelské funkce

Uživatelská funkce je funkce, která nezná vnitřní implementaci tnumů (jako funkcí přesnosti) a volá jen funkce rozhraní zobrazené v Tabulce 4. Během programování knihovny již některé mimoděk vznikly (například `tnum-`), v této kapitole další přidáme. Ukážeme, že síla knihovny spočívá v jednoduchém vytváření nových tnumů a ve velmi silně oddělené vnitřní implementaci od vnějšího chování.

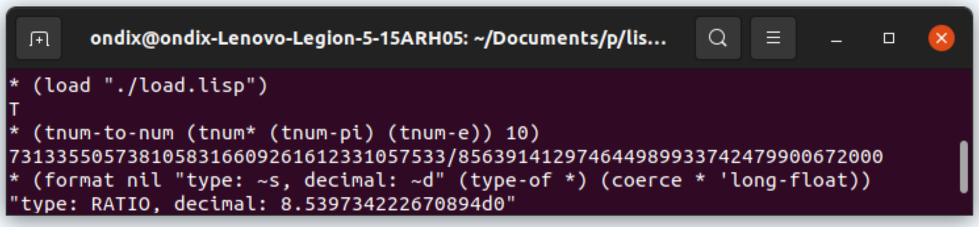
6.1 Instalace

Knihovna `tnums` je k dostání na githubu na odkaze ⟨`https://github.com/slavon00/tnums`⟩ nebo pro čtenáře tištěné verze na přiloženém fyzickém disku. Je to Lispová knihovna, od uživatele se předpokládá znalost základů práce s Lispem.

Vše, co jsme doposud naprogramovali najdeme v souboru `src/tnums.lisp` a všechny funkce, které přidáme v této kapitole, pak v `src/user-functions.lisp`. Testy, které budu ukazovat jsou v souboru `src/tests.lisp`. Vše je možné jednoduše načíst evaluací souboru `load.lisp`, jak ukazuje Obrázek 14. Při ručním překladu je nutné nejdřív přeložit soubor `src/tnums.lisp` a až poté `src/user-functions.lisp`. Pro kompletní obsah adresáře vizte přílohu A.

Knihovna se v budoucnosti může měnit, takže tyto informace mohou zastarat. Knihovna na githubu bude ale vždy obsahovat soubor `README.md` nebo ekvivalentní, s popisem postupu instalace.

Obrázek 14: Načtení knihovny `tnums` do `SBCL`



Aby se nemusely ručně načítat všechny soubory, lze knihovnu `tnums` též načíst jen evaluací souboru `load.lisp`.

6.2 Převody a konstanty

Téměř všechny naprogramované funkce berou jako vstup tnumy. To jsou abstraktní struktury (vyjadřující rekurzivní čísla), které interně reprezentujeme jako funkce malých čísel. K vytvoření tnumu z čísla slouží funkce `num-to-tnum`.

```
* (num-to-tnum 42.123)
#<FUNCTION (LAMBDA (EPS) :IN NUM-TO-TNUM) 10020A056B>
```

Lispový test 1 (`num-to-tnum`): *Představení funkce pro převod* $\mathbb{R} \to \mathfrak{T}$

Pro převod opačným směrem slouží inverzní funkce `tnum-to-num`, která bere tnum a druhý argument představující přesnost, se kterou chceme tnum vyčíslit.

```
* (tnum-to-num * 0.1)
34246/813
```

Lispový test 2 (`tnum-to-num`): *Představení funkce pro převod* $\mathfrak{T} \to \mathbb{Q}$

Výsledkem vyčíslení je racionální číslo.

```
* (type-of *)
RATIO
* (float **)
42.123
```

Lispový test 3 (Typ výstupu je číslo): *Ověření typu vraceného numu*

Výsledkem vyhodnocení je číslo, které můžeme použít jako vstup do dalších funkcí. Naprogramujme nyní funkci, která bude tnumy převádět na textové řetězce. Takto získáme dlouhé rozvoje v čitelné podobě namísto velkých zlomků.

```
(defun tnum-to-string (tnum count)
  (let ((num (tnum-to-num tnum (1+ count))) (output ""))
    (flet ((get-digit (&optional (end ""))
             (multiple-value-bind (digit rem) (floor num)
               (setf output (concatenate 'string output
                                          (write-to-string digit)
                                          end)
                     num (* 10 rem)))))
      (when (< num 0) (setf output "-" num (- num)))
      (get-digit ".")
      (dotimes (i count (concatenate 'string output "..."))
        (get-digit)))))
```

Lispový kód 30 (`tnum-to-string`): *Funkce na převod tnumu na textový řetězec*

Funkce bere jako vstup tnum a přirozené číslo značící počet desetinných míst, která má řetězec obsahovat. Otestujeme ji na výpisu Ludolfova čísla.

```
* (tnum-to-string (tnum-pi) 50)
"3.14159265358979323846264338327950288419716939937510..."
```

Lispový test 4 (`tnum-string` a `tnum-pi`)**:** *Vyčíslení Ludolfova čísla na přesnost 50 desetinných míst*

Vzhledem k rozšíření množiny přípustných hodnot druhého parametru funkce `tnum-to-num` mimo $(0, 1)$ je možné pohodlně přepínat mezi návratovou hodnotou jako číslem a textovým řetězcem. Ukážeme si to na vyčíslení Eulerova čísla.

```
* (tnum-to-num (tnum-e) 20)
611070150698522592097/224800145555521536000
* (tnum-to-string (tnum-e) 20)
"2.71828182845904523536..."
```

Lispový test 5 (`tnum-string` a `tnum-e`)**:** *Vyčíslení Eulerova čísla na přesnost 20 desetinných míst a jeho vrácení jako čísla a jako stringu*

6.3 Operace

Operace tnumu je funkce $\bigtimes_{i=0}^{n-1} \mathfrak{T} \to \mathfrak{T}$, kde $n \in \mathbb{N}$ nazýváme aritou. Operace `tnum+` a `tnum*` mohou mít libovolný počet argumentů, operace `-tnum` a `/tnum` jsou striktně unární, operace `tnum-` a `tnum/` potřebují alespoň jeden argument a operace `tnum-expt` a `tnum-root` jsou binární.

V lispu jsou dvě šikovné funkce na inkrementaci a dekrementaci čísla. To samé nyní přidáme pro tnumy.

```
(defun tnum-1+ (tnum)
  (tnum+ (num-to-tnum 1) tnum))
```

Lispový kód 31 (`tnum-1+`)**:** *Funkce pro inkrementaci tnumu o jedničku*

Funkce pro dekrementaci se také dá napsat pohodlně uživatelsky.

```
(defun tnum-1- (tnum)
  (tnum- tnum (num-to-tnum 1)))
```

Lispový kód 32 (`tnum-1-`)**:** *Funkce pro dekrementaci tnumu o jedničku*

Také by šla napsat nejpoužívanější odmocnina a sice druhá.

```
(defun tnum-sqrt (tnum)
  (tnum-root (num-to-tnum 2) tnum))
```

Lispový kód 33 (`tnum-sqrt`)**:** *Funkce pro druhou odmocninu tnumu*

Výše naprogramované použijeme k zavedení další konstanty, zlatého řezu.

Definice 65 (Zlatý řez [37]). *Zlatý řez představuje kladné řešení rovnice* $x^2 - x - 1 = 0$*, je tedy roven hodnotě*

$$\varphi = \frac{1+\sqrt{5}}{2}. \tag{105}$$

Jedná se o další iracionální konstantu, tentokrát algebraickou, protože je řešením algebraické rovnice a její tnum je jen syntaktický přepis uvedené definice.

```
(defun tnum-phi ()
  (tnum/ (tnum-1+ (tnum-sqrt (num-to-tnum 5))) (num-to-tnum 2)))
```

Lispový kód 34 (`tnum-phi`)**:** *Funkce pro tnum Zlatého řezu*

A ještě test.

```
* (tnum-to-string (tnum-phi) 50)
"1.61803398874989484820458683436563811772030917980576..."
```

Lispový test 6 (`tnum-phi`)**:** *Představení funkce pro zlatý řez*

6.4 Funkce

Funkce tnumů jsou všechny unární. Jedná se o přirozený logaritmus, goniometrické funkce a exponenciálu. Omezení definičního oboru jsou stejná jako jsme zvyklí, naprogramované funkce tedy nejsou o nic „slabší“.

Exponenciálu jsme využili už při psaní obecné mocniny. V podobném duchu nyní zavedeme obecný logaritmus. Vyjdeme z faktu, že lze převádět mezi různými základy.

Fakt 66 (Obecný logaritmus jako podíl přirozených [7]). *Pro* $a > 1$ *a* $x \in \mathbb{R}^+$ *platí*

$$\log_a(x) = \frac{ln(x)}{ln(a)} \tag{106}$$

Nová funkce bude brát dva argumenty, a proto nejde tak úplně o funkci tnumu jak ji v této práci chápeme, ale spíše o matematickou operaci. Nicméně na eleganci zápisu to nic neubírá.

```
(defun tnum-log (tnum1 tnum2)
  (tnum/ (tnum-ln tnum2) (tnum-ln tnum1)))
```

Lispový kód 35 (`tnum-log`)**:** *Funkce pro výpočet obecného logaritmu*

Uživatelská funkce potom podobně jako u odmocniny prohazuje argumenty, protože mluvíme vždy o nějakém logaritmu něčeho, například „devítkový logaritmus dvou“. Ten je i předmětem následujícího testu.

```
* (tnum-to-string (tnum-log (num-to-tnum 9) (num-to-tnum 2)) 50)
"0.31546487678572871854976355717138042714979282006594..."
```

Lispový test 7 (`tnum-log`)**:** *Vyčíslení devítkového logaritmu dvou*

Goniometrické funkce jsou z velké části napsány též uživatelsky, takže by mělo být jasné, jak se s nimi z tohoto pohledu pracuje. Ukážu tedy jen vyčíslení, aby bylo vidět, že funkce opravdu fungují. Následuje výpočet sinu jedničky.

```
* (coerce (tnum-to-num (tnum-sin (num-to-tnum 1)) -20)
    'long-float)
0.8414709848078965d0
```

Lispový test 8 (`tnum-sin`)**:** *Představení funkce na výpočet sinu tnumu*

6.5 Rychlost

Tabulka 3 zobrazuje, jak dlouho běžně trvá vyhodnocení výrazů. Konkrétní hodnoty budou vždy závislé na konkrétním stroji a jeho momentálním zatížení, obecnou představu o časové náročnosti výpočtů však poskytují.

Tabulka 3: Doba výpočtů daných výrazů

Výraz	Doba vyhodnocení (s)
`(let ((tn (tnum/ (tnum-pi) (tnum-e) (tnum-phi))))`	-
`(tnum-to-string tn 50)`	0.427424
`(tnum-to-string (tnum-sin tn) 50)`	31.412191
`(tnum-to-string (tnum-csc tn) 50)`	4690.939552
`(tnum-to-string (tnum-ctan tn) 50))`	26756.683374

Tabulka v prvním sloupci zobrazuje výrazy, které byly vyhodnocovány, a ve druhém čas, který toto vyhodnocení zabralo. Doba byla měřena makrem `time` a hodnoty jsou z řádku „(...) seconds of total run time“.

6.6 Vnější volání

Rozhraní knihovny `tnums` – funkce určené k volání uživatelem – zobrazuje Tabulka 4.

Tabulka 4: Funkce nabízené knihovnou `tnums`

Název	Argumenty	Význam
`tnum-to-num`	`tnum:tnum`, `eps:num`	převod `tnumu` na číslo s přesností `eps`
`tnum-to-string`	`tnum:tnum`, `count:num`	převod `tnumu` na textový řetězec o `count` desetinných místech
`num-to-tnum`	`num:num`	převod `numu` na tnum
`tnum-pi`	$\emptyset$	tnum Ludolfova čísla
`tnum-e`	$\emptyset$	tnum Eulerova čísla
`tnum-phi`	$\emptyset$	tnum Zlatého řezu
`-tnum`	`tnum:tnum`	$-$`tnum`
`tnum+`	0+ `tnumů`	tnumovský protějšek `+`
`tnum-`	1+ `tnumů`	tnumovský protějšek `-`
`/tnum`	`tnum:tnum`	`tnum`
`tnum*`	0+ `tnumů`	tnumovský protějšek `*`
`tnum/`	1+ `tnumů`	tnumovský protějšek `/`
`tnum-expt`	`arg1:tnum`, `arg2:tnum`	$\mathtt{arg1}^{\mathtt{arg2}}$
`tnum-sqrt`	`arg1:tnum`, `arg2:tnum`	$\sqrt[\mathtt{arg1}]{\mathtt{arg2}}$
`tnum-log`	`arg1:tnum`, `arg2:tnum`	$\log_{\mathtt{arg1}}(\mathtt{arg2})$
`tnum-1+`	`tnum:tnum`	$\mathtt{tnum} + 1$
`tnum-1-`	`tnum:tnum`	$\mathtt{tnum} - 1$
`tnum-exp`	`tnum:tnum`	přirozená mocnina `tnumu`
`tnum-ln`	`tnum:tnum`	přirozený logaritmus `tnumu`
`tnum-sqrt`	`tnum:tnum`	druhá odmocnina `tnumu`
`tnum-sin`	`tnum:tnum`	sinus `tnumu`
`tnum-cos`	`tnum:tnum`	kosinus `tnumu`
`tnum-tan`	`tnum:tnum`	tangens `tnumu`
`tnum-csc`	`tnum:tnum`	kotangens `tnumu`
`tnum-sec`	`tnum:tnum`	sekans `tnumu`
`tnum-ctan`	`tnum:tnum`	kosekans `tnumu`

Tabulka v prvním sloupci zobrazuje funkční symbol, v posledním význam funkce aplikované na argumenty z prostředního sloupce. Čtyři části rozdělené horizontálními čarami jsou po řadě funkce pro převody, konstanty, operace a matematické funkce. Součástí jsou i uživatelské funkce.

7 Diskuze

V bakalářské práci, která se zabývá přesnou reprezentací reálných čísel jsme pochopili, že vše je funkce a funkcionálním paradigmatem implementovali rekurzivní čísla. Vzniklou knihovnu `tnums` je možné rozvíjet ve smyslu rozšiřování funkcionality, zrychlováním výpočtů a opravou chyb.

7.1 Souvislosti

Rekurzivní číslo reprezentujeme jako funkci přesnosti a říkáme mu *tnum*. Současná verze knihovny (`tnums.3`) je třetí implementací tnumů. Oproti implementaci pomocí řad (`tnums.1`) je tato reprezentace rychlejší. Oproti implementaci pomocí posloupností (`tnums.2`) je kód přehlednější.

Implementovali jsme 3 matematické konstanty, obousměrné převody mezi racionálními čísly a tnumy, matematické operace nad tnumy a tnumovské funkce. Operace i funkce kopírují lispovskou syntaxi matematických výrazů.

Implementace je relativně jednoduchá v tom smyslu, že velmi přesně kopíruje matematický jazyk, a tak není moc prostoru pro chyby. Také kód vypadá – až na výjimku v podobě funkce `create-list-for-multiplication` – velmi spořádaně a v základním rozsahu (po začátek části III) ho tvoří jen 169 řádků.

Narozdíl od knihoven představených v kapitole 2.4.2 vrací `tnums` čísla a nikoli vlastní datové typy. Sice se vytváří abstraktní datové struktury reprezentující čísla, ale výsledky vyčíslení jsou nativního typu `ratio`. Knihovna tedy nedegraduje na kalkulačku, ale může být přirozeně nasazena v části systému kritické na přesnost. Uživatel je odstíněn od implementace tnumů.

Knihovna toho neumí tolik jako třeba `mpmath`, naopak umí více než `computable reals`.

7.2 Výhled

Funkcionalita knihovny je velmi základní. Pro její rozšíření by bylo možné přidat cyklometrické funkce, další konstanty, případně funkce více proměnných či tetraci.

Dále je možné všechnu funkcionalitu určenou tnumům rozšířit pomocí funkce `num-to-tnum` i na numy a vytvořit tak přesnou kalkulačku.

Rychlost knihovny odpovídá stručnosti kódu. V kapitole 7.4 jsou navrženy dva směry, kudy by se mohla ubírat optimalizace. V existující reprezentaci by se rychlost mohla zvýšit dílčím zapamatováváním proměnných, řádová změna v rychlosti výpočtů bez fundamentálního zásahu do zdrojového kódu ale možná není. Knihovna je relativně pomalá, ale její poslání je přesnost a nikoli rychlost.

7.3 Úskalí

Při psaní knihovny `tnums` byl dbán důraz na korektnost. I tak jsou v knihovně lokalizovány dvě chyby, první příliš omezující není, druhá je závažná.

7.3.0.1 Dotaz na n-tou číslici

Pokud řetězec končí řadou číslic 0 nebo 9, nemusí být tyto a jedna předcházející číslice správně. Číslo sice správně je (2.999... = 3), ale číslice bez kontextu správně nejsou.

```
* (tnum-to-string (tnum-sqrt (num-to-tnum 9)) 50)
"2.99999999999999999999999999999999999999999999999999..."
```

Lispový test 9 (Rozvoj $\sqrt{9}$)**:** *Číslo 3 lze zapsat jako číslo „2.999 . . .“*

7.3.0.2 Odmocnina čísla ≤ 1

Knihovna vrací správně odmocniny jen čísel větších než jedna.

```
* (tnum-to-string (tnum-sqrt (num-to-tnum 1)) 50)
"2.71828182845904523536028747135266249775724709369995..."
```

Lispový test 10 ($\sqrt{1}$)**:** *Při volání odmocniny čísla jedna se místo jedničky vrací Eulerovo číslo*

7.4 Optimalizace

Jak jsme viděli v Tabulce 3, vyčíslení tnumu nemusí být dílem okamžiku, ale může trvat i velmi dlouho. Existují nejméně dva směry, kudy by se mohla ubírat optimalizace pro zrychlení výpočtů čísel s libovolnou přesností – paralelizace a databáze.

7.4.1 Paralelizace

Všechen napsaný kód je sekvenční. Když vyčíslujeme `tnum-pi`, po vypočtení prvního členu se jde na další, ten se přičte a takto se to opakuje až po ukončení cyklu. Rychlejší by bylo, pokud by jeden proces byl zodpovědný pouze za sčítání řady a jednotlivé členy sčítané posloupnosti delegoval na výpočet jiným procesům. Celý výpočet by pak nemusel běžet jen v jednom procesu, na jednom jádře. To stejné platí u funkcí.

Dalším vhodným místem pro použití paralelizace je funkce `tnum+`. Ta sčítá namapovaný seznam tnumů a mapování by mohlo probíhat paralelně, jednotlivé výsledky mohou přijít v různých časech a hlavní vlákno by se staralo jen o vytvoření výsledného seznamu. Výpočet jednotlivých členů by mohl běžet pro každý člen zvlášť, ve vlastním procesu.

Další vhodné místo je funkce `create-list-for-multiplication`, která také vrací seznam. Jde o seznam čísel, která byla nezávisle na ostatních vypočtena funkcí `tnum-to-num`. Tyto výpočty mohou opět vykonávat paralelní procesy, je to tedy vhodné místo pro nasazení paralelizace.

7.4.2 Databáze

Druhým podstatným vylepšením by bylo vytvoření nástroje pro přístup k již vypočteným výsledkům. Při pokusu o vyčíslení tnumu dojde k dotazu na tnum a přesnost a při schodě se výsledek vrátí. Jinak se tnum vypočte, uloží a vrátí.

Otázek s tímto zlepšením je několik:

- Jak ukládat tnumy?
- Jak zabránit, aby nám někdo nahrával chybné výsledky?
- Je morální užívat výsledků, za které zaplatil strojovým časem někdo jiný?

Lepší by bylo, kdyby se na výpočet dalo navazovat. Pak by se nemusel každý výsledek bez záznamu počítat odznova, ale jen od nejbližší horší přesnosti.

Další problém je, jak ukládat tnumy do databáze. O žádné funkcionální databázi nevím. Mohly by se v klasické relační implementaci ukládat textové řetězce.

I zde je prostor pro urychlení některých výpočtů. Například pokud bude dotaz mířit na číslo opačné k číslu, které v databázi záznam má, může se použít tento záznam a jen změnit znaménko.

Největší výzvu představuje odhalování ekvivalencí tnumů. Pokud je dotaz mířen na tnum, jehož ekvivalent již v systému máme, lze vrátit tento. Triviální je sčítání s nulou, násobení a mocnění jedničkou. Další ekvivalence tnumů už může být mnohem skrytější. Číslo x^3 lze napsat jako $x * x * x$. Číslo $x * 3$ lze napsat jako $x+x+x$. To už nejsou tak triviální vztahy a přitom jejich souvislost může vést k mnohem rychlejším výpočtům. A že platí rovnost $4 * \sum_{i\in\mathbb{N}} \frac{(-1)^i}{2i+1} = \pi = \sum_{i\in\mathbb{N}} \frac{1}{16^i}(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6})$? To už je velmi složité.

7.5 Adekvátnost

V kapitole 2.4.1 bylo zadáno 6 podmínek, které musí abstraktní struktury splňovat. Tnumy všechny tyto podmínky dodržují:

- vyčíslování tnumů zajišťuje funkce `tnum-to-num`,
- přesnost zajišťuje samotná podstata tnumů jako funkcí přesnosti,
- matematické operace tnumy podporují,
- matematické funkce tnumy podporují,
- tnumy lze vracet jako výsledky funkcí,
- tnumy lze použít jako argumenty funkcí.

Tnumy tak splňují podmínky na abstraktní datové struktury. Tnumy jsou adekvátní pro realizaci přesných výpočtů s reálnými čísly.

Závěr

Popsal jsem, jak jsou vytvořena přirozená čísla pomocí teorie množin, také jak na tomto základě vznikají další číselné obory. Dále jsem popsal, jak se s **čísly** pracuje v paměti v počítače.

Také bylo zmíněno, že číselná osa je tvořena **reálnými čísly** a pokud použijeme více os, dostáváme strukturovaná čísla. Zamysleli jsme se, jestli jsou všechna reálná čísla rekurzivní a bohužel jsme dostali negativní odpověď.

Představil jsem, jak vypadají **výpočty s reálnými čísly**. Kromě matematických operací to byly matematické funkce. Zjistili jsme, že všechny tyto výpočty, včetně samotných reálných konstant lze reprezentovat jako funkce.

V textu jsem se věnoval i produktu celého tohoto snažení a sice programování Lispovské knihovny `tnums` implementující **přesné výpočty s reálnými čísly**. Také jsem přinesl několik příkladů, jak uživatelsky funkcionalitu rozšiřovat.

Conclusions

I have described how natural **numbers** are created using set theory, as well as how other number systems are created on this basis. I have also described how it works with numbers in a computer memory.

It was also mentioned that the number line is made up of **real numbers** and if we use more axes, we get structured numbers. We wondered if all of the real numbers are recursive and unfortunately we got a negative answer.

I have presented what **computation of real numbers** looks like. In addition to mathematical operations, there were mathematical functions. We have found out that all these calculations, including the real constants themselves, can be represented as functions.

In the text, I also focused on the product of all this effort, namely programming Lisp `tnums` library implementing **precise computation of real numbers**. I have also come up with some examples of how user can extend the functionality.

Seznam literatury

1. PŘISPĚVATELÉ WIKIPEDIE. *Číslo* [online]. Wikipedie: Otevřená encyklopedie, 2021 [cit. 2021-02-21]. Dostupné z: ⟨`https://cs.wikipedia.org/w/index.php?title=%5C%C4%5C%8C%5C%C3%5C%ADslo&oldid=19341576`⟩.
2. CARROLL, Lewis. *Alenka v kraji divů a za zrcadlem.* 1. vyd. Praha: Městská knihovna v Praze, 2018. ISBN 978-80-7602-231-7.
3. ROJAS, Raul. *A Tutorial Introduction to the Lambda Calculus* [online]. Berlin: Freie Universität, 2015 [cit. 2021-04-27]. Dostupné z: ⟨`http://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/tutorials/lambda.pdf`⟩.
4. BALCAR, Bohuslav; ŠTĚPÁNEK, Petr. *Teorie množin.* 2., opravené a rozšířené. Praha: Academia, nakladatelství AV ČR, 2001. ISBN 80-200-0470-X.
5. SLOANE, Neil James Alexander. *A001057: Canonical enumeration of integers: interleaved positive and negative integers with zero prepended* [online] [cit. 2021-04-27]. Dostupné z: ⟨`https://oeis.org/A001057`⟩.
6. SPIVAK, Michael. 3. vyd. Houston: Publish or Perish, 2006. ISBN 9780521867443.
7. MIKULČÁK, Jiří; CHARVÁT, Jura; MACHÁČEK, Martin; ZEMÁNEK, František. *Matematické, fyzikální a chemické tabulky a vzorce pro střední školy.* 1. vyd. Havlíčkův brod: Prometheus, 2012. ISBN 9788071962649.
8. HALAŠ, Radomír. *Teorie čísel.* 2., upravené. Olomouc: Univerzita Palackého, 2014. ISBN 978-80-244-4068-2.
9. PŘISPĚVATELÉ WIKIPEDIE. *Computable number* [online]. Wikipedie: Otevřená encyklopedie, 2020 [cit. 2021-03-03]. Dostupné z: ⟨`https://en.wikipedia.org/w/index.php?title=Computable_number&oldid=997421913`⟩.
10. BAEZ, John C. *Division Algebras and Quantum Theory* [online]. 2011 [cit. 2021-04-27]. Dostupné z: ⟨`https://arxiv.org/pdf/1101.5690.pdf`⟩.
11. RUBSTOV, Constantin A.; ROMERIO, Giovanni F. *Ackerman's Function and New Arithmetical Operations* [online]. 2004 [cit. 2021-04-27]. Dostupné z: ⟨`http://www.rotarysaluzzo.it/Z_Vecchio_Sito/filePDF/Iperoperazioni%5C%20(1).pdf`⟩.
12. PELANTOVÁ, Edita; VONDRÁČKOVÁ, Jana. *Matematická analýza I.* Praha: České vysoké učení technické, 2004. Dostupné také z: ⟨`http://km.fjfi.cvut.cz/ma/data/uploads/skripta-matematicka-analyza-1.pdf`⟩.
13. HORT, Daniel; RACHŮNEK, Jiří. *Algebra 1.* Olomouc: Univerzita Palackého, 2003.
14. DOŠLÁ, Zuzana; KUBEN, Jaromír. *Diferenciální počet funkcí jedné proměnné.* 1. vyd. Brno: Masarykova univerzita, 2004. ISBN 80-210-3121-2.

15. DOŠLÁ, Zuzana; NOVÁK, Vítězslav. *Nekonečné řady.* 1. vyd. Brno: Masarykova univerzita, 2002. ISBN 80-210-1949-2.

16. KEPRT, Aleš. *Operační systémy.* Olomouc: Univerzita Palackého, 2007. Dostupné také z: ⟨https://phoenix.inf.upol.cz/esf/ucebni/OpSys.pdf⟩.

17. KNUTH, Donald Ervin. *Umění programování. 2. díl: Seminumerické algoritmy.* 1. vyd. Brno: Computer Press, 2010. ISBN 978-80-251-2898-5.

18. PŘISPĚVATELÉ WIKIPEDIE. *Single-precision floating-point format* [online]. Wikipedie: Otevřená encyklopedie, 2021 [cit. 2021-02-20]. Dostupné z: ⟨https://en.wikipedia.org/w/index.php?title=Single-precision_floating-point_format&oldid=1005180810⟩.

19. STANNERED. *Binary representation of a 32-bit floating-point number* [online]. Wikimedia Commons, 2008 [cit. 2021-03-29]. Dostupné z: ⟨https://upload.wikimedia.org/wikipedia/commons/d/d2/Float_example.svg⟩. Dostupný pod licencí CC BY-SA 3.0.

20. PŘISPĚVATELÉ WIKIPEDIE. *Long double* [online]. Wikipedie: Otevřená encyklopedie, 2021 [cit. 2021-03-18]. Dostupné z: ⟨https://en.wikipedia.org/w/index.php?title=Long_double&oldid=1003717804⟩.

21. JOHANSSON, Fredrik a kol. *mpmath: a Python library for arbitrary-precision floating-point arithmetic* [online]. 2013 [cit. 2021-03-29]. Dostupné z: ⟨http://mpmath.org⟩.

22. DAUTELLE, Jean-Marie. *jscience: Tools & Libraries for the Advancement of Sciences* [online]. GitHub.io, 2017 [cit. 2021-03-29]. Dostupné z: ⟨https://github.com/javolution/jscience⟩.

23. GRANLUND, Torbjörn a kol. *GNU MP: The GNU Multiple Precision Arithmetic Library* [online]. 2020 [cit. 2021-03-29]. Dostupné z: ⟨https://gmplib.org/gmp-man-6.2.1.pdf⟩.

24. *GNU MPFR: The Multiple Precision Floating-Point Reliable Library* [online]. 2020 [cit. 2021-03-29]. Dostupné z: ⟨https://www.mpfr.org/mpfr-current/mpfr.pdf⟩.

25. GRANLUND, Torbjörn; HART, William; GLADMAN, Brian a kol. *MPIR: The Multiple Precision Integers and Rationals Library* [online]. 2017 [cit. 2021-03-29]. Dostupné z: ⟨http://mpir.org/mpir-3.0.0.pdf⟩.

26. HAIBLE, Bruno; KRECKEL, Richard B. *CLN: a Class Library for Numbers* [online]. 2019 [cit. 2021-03-29]. Dostupné z: ⟨https://www.ginac.de/CLN/cln.pdf⟩.

27. STOLL, Michael. *computable-reals: Arbitrary precision, automatic re-computing real numbers in Common Lisp* [online]. GitHub.io, 2021 [cit. 2021-03-29]. Dostupné z: ⟨https://github.com/stylewarning/computable-reals⟩.

28. THE COMMON LISP COOKBOOK PROJECT. *The Common Lisp Cookbook: Numbers* [online]. GotHub.io, 2020 [cit. 2021-03-29]. Dostupné z: ⟨https://lispcookbook.github.io/cl-cookbook/numbers.html⟩.
29. SEIBEL, Peter. *Practical Common Lisp*. 1. vyd. New York: Apress, 2005. ISBN 1590592395.
30. RICHESON, David. *Circular Reasoning: Who First Proved That C/d Is a Constant?* [Online]. Dickinson College, 2010 [cit. 2021-04-27]. Dostupné z: ⟨https://arxiv.org/pdf/1303.0904.pdf⟩.
31. JANSSON, Madeleine. *Approximation of* π [online]. Lund University, 2019 [cit. 2021-04-27]. Dostupné z: ⟨https://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8983341&fileOId=8983342⟩.
32. BAILEY, David; BORWEIN, Peter; PLOUFFE, Simon. *On the Rapid Computation of Various Polylogarithmic Constants* [online]. 1997 [cit. 2021-04-27]. Dostupné z: ⟨https://www.ams.org/journals/mcom/1997-66-218/S0025-5718-97-00856-9/S0025-5718-97-00856-9.pdf⟩.
33. RICE, Henry Gordon. Recursive Real Numbers. *Proceedings of the American Mathematical Society* [online]. 1954, roč. 5, č. 5, s. 784–791 [cit. 2021-07-22]. Dostupné z: ⟨https://www.ams.org/journals/proc/1954-005-05/S0002-9939-1954-0063328-5/home.html⟩.
34. HABALA, Petr. *Taylorův polynom* [online]. Praha: České vysoké učení technické [cit. 2021-03-29]. Dostupné z: ⟨https://math.fel.cvut.cz/mt/txtc/4/txc3ca4e.htm⟩.
35. APOSTOL, Tom M. *Calculus: Volume I*. 2. vyd. USA: John Wiley & Sons, 1967. ISBN 0-471-00005-1.
36. ABRAMOWITZ, Milton; STEGUN, Irene A. *Handbook of Mathematical Functions: With Formulas, Graphs and Mathematical Tables* [online]. 10, with corrections. Washington: U.S. Government Printing Office, 1964 [cit. 2021-04-28]. Dostupné z: ⟨http://www.math.ubc.ca/~cbm/aands/abramowitz_and_stegun.pdf⟩.
37. SPIRA, Michel. *On the Golden Ratio* [online]. Universidade Federal de Minas Gerais [cit. 2021-04-27]. Dostupné z: ⟨https://www.mathunion.org/fileadmin/ICMI/Conferences/ICME/ICME12/www.icme12.org/upload/submission/1948_F.pdf⟩.

A Obsah přiloženého CD/DVD

Zde je obsah adresáře na přiloženém fyzickém disku. Bez adresáře `install/` je stejná struktura též na GitHubu.

`doc/`
: Adresář se soubory

- `OndrejSlavikBP.pdf` – tento text ve formátu PDF a
- `bak/` – adresář se všemi soubory pro vysázení tohoto textu, stačí dvakrát přeložit `PDFLaTex`em.

`load.lisp`
: Přeložením tohoto souboru ve vašem oblíbeném interpretu/kompilátoru Lispu získáte plnou funkcionalitu knihovny `tnums`, kterou jsme právě doprogramovali. Načítá soubory `src/tnums.lisp` a `src/user-functions.lisp`.

`src/`
: Adresář se soubory

- `tnums.lisp` – základ knihovny z části 2 této práce,
- `user-function.lisp` – rozšíření knihovny o vědomě uživatelské funkce ze šesté kapitoly a
- `tests.lisp` – zakomentované výrazy, které zde byly popsány jako Lispový test i s výsledky, na které se vyhodnotí.

`README.md`
: Soubor představující knihovnu `tnums` a obsahující i několik příkladů výpočtů, které podporuje. Součástí je i kapitola o načtení knihovny evaluací souboru `load.lisp`, nejedná se tedy o instalaci v pravém slova smyslu.

`install/`
: Adresář se soubory

- `sbcl-2.1.6-source.tar.bz2` – intalátor SBCL, konzolového kompilátoru ANSI Common Lispu,
- `code_1.58.2-1626302803_amd64.deb` – instalátor VS code, rozšiřitelného textového editoru,
- `2gua.rainbow-brackets-0.0.6.vsix` – instalátor rozšíření Rainbow Brackets pro přehledné obarvování závorek a
- `qingpeng.common-lisp-0.0.2.vsix` – instalátor rozšíření Common Lisp pro podbarvování kódu.

`LISENCE`
: Licenší soubor – knihovna je publikována pod GNU GPLv3.