Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra teoretické informatiky

Bakalářská práce

Přepisování číselných řetězců a jeho aplikace v teorii čísel

David Oppl

Vedoucí práce: Ing. Daniel Dombek, PhD.

10. května 2015

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Danielu Dombkovi, PhD. za jeho ochotu, přínosné rady, připomínky a odbornou pomoc při tvorbě této bakalářské práce. Dále děkuji svým přátelům za rady ohledně implementace a pomoc s testováním. V neposlední řadě děkuji své rodině za jejich podporu po celou dobu mého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel licenční smlouvu o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 10. května 2015 .....

Odkaz na tuto práci

Oppl, David. *Přepisování číselných řetězců a jeho aplikace v teorii čísel*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato práce se věnuje studiu aplikací tzv. přepisovacích pravidel, speciálních zobrazení na množině nekonečných slov. Rešeršní část shrnuje problematiku poziční reprezenzace čísel a ukazuje, jak lze přepisovací pravidla použít při normalizaci, oveřování přípustnosti a sčítání reprezentací čísel. Dále se čtenář seznámí s tzv. DUG vlastností algebraických číselných těles a její souvislostí s přepisovacími pravidly. Výsledkem implementační části jsou programy pro přepisování číselných řetězců a paralelní sčítání reprezentací čísel.

Klíčová slova Přepisovací pravidlo, teorie čísel, algebraické číselné těleso, DUG vlastnost, reprezentace čísel, paralelní sčítání, C++, OpenMP

Abstract

This thesis is devoted to the study of applications of the so-called rewriting rules, special maps over the set of infinite words. The survey part contains summary on the topic of positional numeration systems and shows applications of rewriting rules in the problems of normalization, verification of admissibility and addition of the number representations. Furthermore, the reader is apprised with the so-called DUG property of algebraic fields and its connection with the rewriting rules. The goal of the implemental part is to create applications for rewriting digit strings and for parallel addition of number representations.

Keywords Rewriting rule, number theory, algebraic number field, DUG property, number representation, parallel addition, C++, OpenMP

Obsah

Úvod

V této bakalářské práci se budeme věnovat speciálním zobrazením na množině nekonečných slov, tzv. přepisovacím pravidlům, respektive jejich aplikacím.

V první části se seznámíme se základními pojmy z teorie čísel, kombinatoriky na slovech a reprezentací čísel v bázi. Znalost těchto pojmů je nutná pro pochopení dalších částí této práce.

Ve druhé části nejdříve definujeme pojem přepisovací pravidlo a ukážeme si, jak jej lze využít v práci s pozičními reprezentacemi čísel. Prvním uvedeným příkladem bude tzv. normalizace nepřípustných reprezentací a poté se zaměříme na problematiku sčítání reprezentací čísel pomocí přepisovacích pravidel tak, aby výsledný součet byl řetězcem nad stejnou abecedou jako sčítané reprezentace. Následně uvedeme algoritmy, s jejichž využitím lze toto sčítání realizovat paralelně.

V další části se budeme zabývat algebraickými číselnými tělesy, v nichž lze každé celé číslo zapsat jako konečný součet různých algebraických jednotek. Tato tělesa se nazývají DUG (anglicky distinct unit generated). Nejdříve shrneme výsledky na téma klasifikace DUG těles a potom se zaměříme na jedno konkrétní těleso, u kterého byla DUG vlastnost dokázána kombinatoricky s využitím přepisovacích pravidel. Tato část je podkladem pro přepisovací program, jehož implementace je součástí této práce.

V poslední části popíšeme dva programy, jež jsou výstupem této práce. Nejprve shrneme požadavky na program pro přepisování reprezentací čísel následované požadavky na program implementující paralelní sčítání. Dále se budeme zabývat výběrem programovacího jazyka a popisem tříd a funkcí, které jsou využity v implementaci. Následuje měření rychlosti programu pro paralelní sčítání a na závěr shrneme možnosti rozšíření obou implementovaných programů.

Kapitola 1

Základní pojmy

Tato kapitola definuje základní pojmy, týkající se dané problematiky. Většina definic z teorie čísel byla převzata ze skript Teorie čísel [12]. Definice týkající se formálních jazyků byly sepsány na základě skript Jazyky a překlady [13] a knihy Combinatorics on words [10]. Teorie týkající se pozičních reprezentací čísel je citována z [11] a případné další zdroje jsou uvedeny explicitně. Veškerý další obsah obsah této práce se opírá o tyto definice.

1.1 Kombinatorika na slovech

Množinu $\mathcal{A}$ nazveme **abecedou** a její prvky symboly. Pro zjednodušení uvažujme $\mathcal{A} \subseteq \mathbb{Z}$. Posloupnost symbolů abecedy nazveme **řetězcem** nad danou abecedou. **Prázdným řetězcem** rozumíme prázdnou posloupnost symbolů a značíme písmenem ε. Množinu všech konečných řetězců označíme $\mathcal{A}^*$. Množinu všech konečných neprázdných řetězců nad abecedou $\mathcal{A}$ označíme $\mathcal{A}^+$.

Délkou řetězce $w = w_1 w_2 \dots w_n$ rozumíme počet symbolů abecedy $\mathcal{A}$, tedy

$$|w| = |w_1 w_2 \dots w_n| = n, \qquad \text{kde } w_1, w_2, \dots, w_n \neq \varepsilon.$$

Délka prázdného řetězce je 0.

Na množině řetězců $\mathcal{A}^*$ je definována operace **zřetězení** následovně: Nechť $w_1 = a_1 a_2 \dots a_n \in \mathcal{A}^*$ a $w_2 = b_1 b_2 \dots b_m \in \mathcal{A}^*$. Řetězec w nazveme jejich zřetězením, pokud

$$w = w_1 w_2 = a_1 a_2 \dots a_n b_1 b_2 \dots b_m.$$

Prázdný řetězec $\varepsilon \in \mathcal{A}^*$ je neutrálním prvkem k operaci zřetězení, pro každý řetězec $w \in \mathcal{A}^*$ tedy platí

$$\varepsilon w = w\varepsilon = w.$$

Faktor řetězce $w = w_1 w_2 \dots w_n$ je řetězec $\hat{w} = w_i w_{i+1} \dots w_{m+i}$, kde $i \geq 0$ a $m + i \leq n$. Pokud $i = 1$, řetězec $\hat{w}$ nazveme **prefixem** řetězce w. Je-li $m + i = n$, řetězec $\hat{w}$ je **sufixem** řetězce w. Jestliže se prefixy a sufixy nerovnají původnímu řetězci, nazýváme je vlastní.

Množinou $\mathcal{A}^{\mathbb{N}}$ označíme všechny **nekonečné řetězce** nad abecedou $\mathcal{A}$, tedy řetězce ve tvaru $w = w_1 w_2 w_3 \cdots$. U nekonečných řetězců můžeme určit faktor, prefix a sufix obdobně jako u konečných slov, avšak nedefinují se konečný sufix, nekonečný prefix a nekonečný faktor, který není zároveň sufixem.

Pro porovnání dvou nekonečných řetězců nad uspořádanou množinou symbolů A využijeme **lexikografické uspořádání** $\preceq_{lex}$, které definujeme takto: Nechť $w = w_1 w_2 \dots, v = v_1 v_2 \cdots \in \mathcal{A}^{\mathbb{N}}$. Řetězec w je lexikograficky menší nebo roven v, značeno $w \preceq_{lex} v$, právě tehdy, když platí

$$w = v \quad \text{nebo} \quad w_k < v_k \text{ pro } k = min\{i \geq 1 | w_i \neq v_i\}.$$

1.2 Teorie čísel

Množina G s asociativní binární operací $\circ : G \times G \to G$ se nazývá **grupa**, pokud:

- v G existuje neutrální prvek e, pro který platí $x \circ e = x$ pro každé $x \in G$,
- ke každému $x \in G$ existuje inverzní prvek $x \in G$, pro který platí $x \circ y = y \circ x = e$.

Pokud je operace $\circ$ komutativní, nazýváme G komutativní grupou.

Množina R se dvěma asociativními binárními operacemi $\times, + : R \times R \to R$ se nazývá **okruh**, pokud:

- R s operací $+$ je komutativní grupa s neutrálním prvkem 0,
- operace $\times$ je distributivní vůči $+$, tedy platí $x \times (y+z) = (x \times y) + (x \times z)$ a $(y + z) \times x = (y \times x) + (z \times x)$ pro všechna $x, y \in R$.

Okruh R s operacemi $+$ a $\times$ se nazývá **těleso**, pokud $R \setminus \{0\}$ s operací $\times$ je grupa. Je-li operace$\times$ komutativní, nazveme R komutativním tělesem.

Nechť R je okruh. **Polynomem** nad R nazýváme výraz:

$$f(x) = a_0 + a_1 X + \cdots + a_{n-1} X^{n-1} + a_n X^n, \qquad a_0, \dots a_n \in R$$

Polynom f je stupně n, je-li n nejvyšší index, pro který platí $a_n \neq 0$. Pokud $a_i = 0$ pro všechna $i \geq 0$, f je nulový polynom a jeho stupeň je -1. Polynom f stupně n nazveme **monický**, pokud koeficient $a_n = 1$.

Nechť K je komutativní okruh. **Okruh polynomů** nad K označíme $K[X]$ a definujeme jej následovně:

$$K[X] = \left\{ \sum_{i=0}^{m} c_i X^i \quad | \quad m \geq 0, c_i \in K \right\}$$

Nechť L a K jsou komutativní tělesa a $K \subseteq L$. Je-li prvek $x \in L$ kořenem nějakého polynomu $f \in K[X]$, prvek x je **algebraický** nad K. Pro každé takové x existuje monický polynom $f_0 \in K[X]$ nejmenšího stupně m, který nazýváme **minimálním polynomem** x nad K. **Stupeň** x nad K je definován jako stupeň polynomu f_0, tedy m. Jsou-li všechny prvky L algebraické nad K, L je nadtělesem K a K je podtělesem L. Na L lze nahlížet jako na vektorový prostor nad tělesem K. Dimenzi vektorového prostoru L nad K značíme $[L : K]$.

Číslo $\alpha \in \mathbb{C}$ je **algebraické číslo**, pokud existuje monický polynom $f \in \mathbb{Q}[X]$ takový, že $f(\alpha) = 0$.

Algebraické číslo $\alpha \in \mathbb{C}$ je **algebraické celé číslo**, pokud jeho minimální polynom f_0 je prvkem okruhu polynomů $\mathbb{Z}[X]$.

Je-li α konečného stupně n nad $\mathbb{Q}$, lze $\mathbb{Q}(\alpha)$ zapsat následovně:

$$\mathbb{Q}(\alpha) = \{c_0 + c_1\alpha + c_2\alpha^2 + \cdots + c_{n-1}\alpha^{n-1} |\ c_i \in \mathbb{Q}\}$$

Stupeň generujícího prvku α nazýváme **stupněm tělesa** $\mathbb{Q}(\alpha)$.

Nechť K je těleso. Množinu O_K všech algebraických celých čísel obsažebých v K nazýváme **okruh celých čísel** v K.

V každém algebraickém tělese $K = \mathbb{Q}(\alpha)$ stupně n s okruhem algebraických celých čísel O_K existuje uspořádaná množina $\{\beta_1, \ldots, \beta_n\}$, kterou nazýváme **integrální báze**. Její prvky jsou algebraická celá čísla a pro všechna $\beta \in O_K$ platí

$$\beta = \sum_{i=1}^{n} a_i\beta_i \quad a_i \in \mathbb{Z}.$$

Algebraické celé číslo $y \in O_K$ **dělí** $x \in O_K$, pokud existuje $z \in O_K$ takové, že $x = yz$. Pokud x dělí 1, čísla x a $1/x$ jsou **algebraické jednotky**. Množina všech algebraických jednotek tvoří multiplikativní grupu a značíme ji U_K.

Číslo $\zeta \in \mathbb{C}$ nazveme ***n*-tým kořenem jednotky**, pokud platí $\zeta^n = 1$. Pokud n je nejmenší celé číslo ≥ 1, pro které platí $\zeta^n = 1$, nazveme ζ **primitivním** n-tým kořenem jednotky.

Věta 1.1. (Dirichletova) Nechť $K = \mathbb{Q}(\alpha)$ je těleso a minimální polynom algebraického čísla α má s reálných a $2t$ nereálných (v komplexně sdružených párech) kořenů. $r = s + t - 1$ nazveme **unit rank**. Existuje primitivní kořen $\zeta \in K$ a jednotky $\eta_1, \ldots, \eta_r$ takové, že platí

$$U_k = \{\zeta^{j_0}\eta_1^{j_1} \cdots \eta_r^{j_r} \mid j_0, \ldots j_n \in \mathbb{Z}\}.$$

Množinu $\{\eta_1, \ldots, \eta_r\}$ nazýváme **fundamentální systém** jednotek tělesa K.

Příklad 1.2. *Nejjednodušším příkladem algebraického číselného tělesa je těleso racionálních čísel $\mathbb{Q}$. Okruh algebraických celých čísel tvoří celá čísla.*

Platí tedy

$$O_{\mathbb{Q}} = \mathbb{Z} \qquad\qquad U_{\mathbb{Q}} = \{1, -1\}.$$

Dalším zajímavým příkladem je rozšíření racionálních čísel o větší z kořenů polynomu $f(X) = X^2 - X - 1$. *Kořeny tohoto polynomu jsou tzv. zlatý řez* $\tau = \frac{1+\sqrt{5}}{2}$ *a* $\tau' = \frac{1-\sqrt{5}}{2} = -\frac{1}{\tau}$. *Těleso* $K = \mathbb{Q}(\tau)$ *obsahuje všechna algebraická čísla ve tvaru* $a + b\frac{1+\sqrt{5}}{2}$, *kde* $a, b \in \mathbb{Q}$. *Okruh algebraických celých čísel* O_K *lze vyjádřit následovně:*

$$O_K = O_{\mathbb{Q}(\tau)} = \left\{ a + b\frac{1+\sqrt{5}}{2} \;\middle|\; a, b \in \mathbb{Z} \right\}$$

Pro grupu jednotek U_K *platí*

$$U_K = U_{\mathbb{Q}(\tau)} = \left\{ \pm \left(\frac{1+\sqrt{5}}{2} \right)^k \middle| k \in \mathbb{Z} \right\}$$

1.3 Reprezentace čísel v bázi

Každé číslo $x \in \mathbb{R}$ může být vyjádřeno v číselné soustavě o celočíselném základu $b > 1$ následovně:

$$x = \pm \left(d_k b^k + \cdots + d_1 b + d_0 + \frac{d_{-1}}{b} + \frac{d_{-2}}{b^2} + \dots \right), \text{ kde } d_i \in \{0, 1, \dots, b-1\}$$

Nekonečnou posloupnost symbolů $d_k d_{k-1} \cdots$ zapíšeme jako nekonečné slovo $d(x) \in \mathcal{A}^{\mathbb{N}}$ takto

$$d(x) = \begin{cases} d_k \cdots d_0 \bullet d_{-1} \cdots & \text{pokud } k \geq 0, \\ 0 \bullet 0^{-k-1} d_k d_{k-1} \cdots & \text{pokud } k < 0, \end{cases}$$

kde zlomková tečka $\bullet$ odděluje koeficienty u nezáporných a záporných mocnin b, tzv. **celou** a **zlomkovou část** reprezentace $d(x)$. Řetězec $d(x)$ nazýváme **reprezentací** čísla x v číselné soustavě o zákadu b. Má-li řetězec $d(x)$ nekonečný sufix 0^ω, nepíšeme jej (celou část reprezentace $d(x)$ však zapíšeme celou) a řekneme, že x má **konečnou reprezentaci** $d(x)$. Mezi všemi reprezentacemi $x \in \mathbb{R} \setminus \{0\}$ v celočíselné bázi $b \geq 2$ existuje vždy jedna reprezentace bez sufixu $(b-1)^\omega$, tzv. **přípustná**.

Poznámka 1.3. *Vztah* $d(x) = d_k \cdots d_0 \bullet d_{-1} \cdots$ *lze zapisovat jako* $x = d_k \cdots d_0 \bullet_b d_{-1} \cdots$, *případně* $d(x) = d_k \cdots d_0\, b$, *pokud je zlomková část* $d(x)$ *nulová.*

Tento způsob zápisu lze rozšířit i pro soustavy s neceločíselným základem $\beta > 1$. Řetězec d se v takovém případě označuje jako β-reprezentace. Pro každé reálné číslo existuje alespoň jedna β-reprezentace. V soustavách se základem $b \in \mathbb{Z}$ nejsou některé nekonečné reprezentace přípustné, ale všechny konečné reprezentace již přípustné jsou. Naproti tomu v soustavách s obecným základem $\beta > 1$ je situace složitější a konečnost reprezentace vůbec nezaručuje její přípustnost nebo jednoznačnost.

Příklad 1.4. *Uvažujme číselnou soustavu o základu* $\beta = \tau = \frac{1+\sqrt{5}}{2}$. *Pro čísla* $x = 11_\tau$ *a* $y = 100_\tau$ *snadno ukážeme, že se rovnají*

$$x = 11_\tau = \frac{1+\sqrt{5}}{2} + 1 = \frac{3+\sqrt{5}}{2}$$
$$y = 100_\tau = \left(\frac{1+\sqrt{5}}{2}\right)^2 = \frac{3+\sqrt{5}}{2}$$

I přes nejednoznačnost zápisu některých čísel v soustavách o neceločíselném základu lze podle [11] pro každé číslo $x \in \mathbb{R}_0^+$ nalézt tzv. hladovým algoritmem jeho unikátní, lexikograficky největší β-reprezentaci $d = d_k d_{k-1} \cdots$. Algoritmus najde největší k splňující podmínku $\beta^k \leq x$ a následně největší d_k splňující $d_k \beta^k \leq x$. V každém dalším kroku j je nalezeno největší d_j, které splňuje $d_j \beta^j + \sum_{i=j+1}^{k} d_i \beta^i \leq x$. Je tedy patrné, že výstupem je nekonečné slovo - lexikograficky největší β-reprezentace čísla x, tzv. **β-rozvoj** čísla x. β-rozvoj je prvkem $\mathcal{A}^{\mathbb{N}}$, kde $\mathcal{A} = \{0, 1, \ldots, \lceil \beta \rceil - 1\}$ je **kanonická abeceda**. β-rozvoj záporného čísla x se definuje jako β-rozvoj $|x|$, jehož nejlevější nenulový symbol je doplněn o znaménko $-$.

Algoritmus 1.1 Nalezení β-rozvoje čísla $x \geq 0$

Vstup: x,
1: základ číselné soustavy β
Výstup: $d = d_k d_{k-1} \cdots d_0 \bullet d_{-1} \cdots$
2: $k \leftarrow \lfloor log_\beta x \rfloor$
3: $d_k \leftarrow \lfloor x/\beta^k \rfloor$
4: $r_k \leftarrow x/\beta^k - d_k$
5: **for** $j = k - 1$ to $-\infty$ **do**
6: $\quad d_j \leftarrow \lfloor \beta r_{j+1} \rfloor$
7: $\quad r_j \leftarrow \beta r_{j+1} - d_j$
8: **end for**
9: **return** d

β-rozvoj reálných čísel lze ekvivalentně definovat pomocí iterací jisté transformace jednotkového intervalu. Tento způsob pochází z práce A. Rényiho [15]:

Věta 1.5. *Nechť je zobrazení* $T_\beta : [0, 1) \to [0, 1)$ *definováno jako*

$$T_\beta : x \to \beta x - \lfloor \beta x \rfloor,$$

kde $\lfloor y \rfloor$ značí dolní celou část y.

Každému $x \in [0,1)$ lze přiřadit řetězec $d_\beta(x) = x_1 x_2 \cdots$ předpisem

$$x_i = \lfloor \beta T^{i-1}(x) \rfloor, \quad \text{pro každé } i \geq 1.$$

Řetězec $d_\beta(x)$ pak splňuje $x = \sum_{i \geq 1} \frac{x_i}{\beta^i}$.

S využitím předchozí věty lze přirozeně rozšířit definici řetězců $d_\beta(x)$ na všechna reálná čísla. Pro každé kladné $x \in \mathbb{R}$ zvolíme nejmenší index k, pro který platí

$$\frac{1}{\beta^k} x \in [0,1).$$

Dle Věty 1.5 existuje řetězec $d_\beta\left(\frac{x}{\beta^k}\right)$. Označíme-li cifry $d_\beta\left(\frac{x}{\beta^k}\right)$ následovně

$$d_\beta\left(\frac{x}{\beta^k}\right) = d_{k-1} d_{k-2} \cdots ,$$

platí $\frac{x}{\beta^k} = \sum_{i \geq 1} d_{k-i} \beta^i$, tedy $x = \sum_{j \leq k-1} d_j \beta^j$ a **β-rozvojem** čísla x nazveme řetězec

$$x = x_\beta = \begin{cases} 0 \bullet d_\beta(x) & \text{pokud } k = 0, \\ d_{k-1} d_{k-2} \cdots d_0 \bullet d_{-1} \cdots & \text{pokud } k > 0, \end{cases}$$

přičemž používáme zkrácený zápis jako v Poznámce 1.3.

V této práci budeme pracovat s podobným, ale obecnějším typem reprezentací, s obecně komplexním základem a různými volbami pro abecedu použitých cifer. Existenci takových reprezentací zaručuje následující věta z práce W. P. Thurstona [17]:

Věta 1.6. *Nechť $\alpha \in \mathbb{C}$ a $|\alpha| > 1$. Pokud daná abeceda $\mathcal{A} \subseteq \mathbb{C}$ a množina $V \subseteq \mathbb{C}$ splňují*

$$\alpha V \subseteq \bigcup_{\alpha \in \mathcal{A}} (V + \alpha), \tag{1.1}$$

potom pro všechna $z \in V$ existuje takový řetězec $a_1 a_2 \cdots \in \mathcal{A}^{\mathbb{N}}$, že

$$z = \frac{a_1}{\alpha} + \frac{a_2}{\alpha^2} + \ldots .$$

Pokud 0 je prvkem vnitřku množiny V, každé $z \in \mathbb{C}$ lze zapsat ve tvaru

$$z = d_k \alpha^k + \cdots + d_1 \alpha + d_0 + \frac{d_{-1}}{\alpha} + \frac{d_{-2}}{\alpha^2} + \ldots$$

kde $k \in \mathbb{Z}$, $d_i \in \mathcal{A}$ a $d_k \neq 0$.

Řetězce $d_k d_{k-1} \cdots \in \mathcal{A}^{\mathbb{N}}$ z druhé části Věty 1.6 nazveme **$\boldsymbol{\alpha}$-reprezentacemi nad abecedou $\boldsymbol{\mathcal{A}}$** a budeme je zapisovat podobně jako výše,

$$x = \begin{cases} d_k \cdots d_0 \bullet d_{-1} \cdots & \text{pokud } k \geq 0, \\ 0 \bullet 0^{-k-1} d_k d_{k-1} \cdots & \text{pokud } k < 0. \end{cases}$$

Poznamenejme, že podmínku 1.1 z Věty 1.6 nebude třeba v konkrétních případech ověřovat. Pro ilustraci lze uvést, že definice Rényiho β-rozvojů spadá pod speciální případ věty 1.6, zvolíme-li bázi $\alpha = \beta - 1$, jednotkový interval $V = [0, 1)$ a kanonickou abecedu $\{0, 1, \ldots, \lceil \beta \rceil - 1\}$.

Na závěr zdůrazněme, že u obecnějších reprezentací nebudeme používat termín rozvoj, neboť tam je problematika jednoznačnosti poněkud komplikovanější.

Kapitola 2

Různé aplikace přepisovacích pravidel

Na úvod této kapitoly si definujeme pojem přepisovací pravidlo. Následně si ukážeme několik příkladů aplikací přepisovacích pravidel při práci s reprezentacemi čísel v bázi.

Poznámka 2.1. *Poznamenejme, že se v této práci odchylujeme od tradičního zápisu konečných i nekonečných slov s indexací od jedničky ($x_1x_2x_3\cdots$), přpadně od nuly. Pro znázornění vztahu slov nad abecedou a reprezentací čísel v bázi bude vhodnější často použít sestupnou indexaci $x_k\cdots x_0\bullet x_{-1}\cdots$. Pokud nebude nutné použít v zápisu zlomkovou tečku $\bullet$, vynecháme ji.*

Definice 2.2. *Nechť $\alpha\in\mathbb{C}, |\alpha|>1$, je kořenem polynomu $p\in\mathbb{Z}[X]$, ne nutně minimálního,*

$$p(x)=a_mX^m+a_{m-1}X^{m-1}+\cdots+a_1X+a_0.$$

*Konečné slovo $a_ma_{m-1}\cdots a_0\in\mathbb{Z}^*$ nazveme **přepisovacím pravidlem** v bázi α.*

*Nechť $x_kx_{k-1}\cdots x_0\bullet x_{-1}\cdots$ je α-reprezentace nějakého $x\in\mathbb{C}$. **Aplikací přepisovacího pravidla** $a_ma_{m-1}\cdots a_0$ na pozici $i\in\mathbb{Z}$ rozumíme přiřazení $x_kx_{k-1}\cdots\to z_lz_{l-1}\cdots$ kde*

$$z_j=\begin{cases}x_j+a_{j-i} & \text{pokud } j\in\{i,i+1,\ldots,i+m\},\\ x_j & \text{jinak.}\end{cases}$$

Aplikaci přepisovacího pravidla budeme znázorňovat takto:

			x_k	x_{k-1}	$\cdots$	x_i	$\cdots$	x_1	x_0	$\bullet$	x_{-1}	$\cdots$
$+$	a_m	a_{m-1}			$\cdots$	a_0						
	z_l	z_{l-1}			$\cdots$	z_i	$\cdots$	z_1	z_0	$\bullet$	z_{-1}	$\cdots$

Z definice přepisovacího pravidla v bázi α je zřejmé, že aplikace přepisovacího pravidla pouze převádí reprezentace čísel v bázi α na jiné, přičemž se může změnit abeceda použitých cifer.

V následujících podkapitolách se budeme věnovat využití přepisovacích pravidel k získávání reprezentací čísel na reprezentace s požadovanou vlastností. Nejdříve se zaměříme na přepis nepřípustných rozvojů na přípustné, potom se zmíníme o přepisování reprezentací nad velkou abecedou na menší abecedu a kapitolu zakončíme paralelizací uvedených algoritmů.

2.1 Úprava nepřípustných rozvojů

Jak bylo ukázáno v Příkladu 1.4, β-reprezentace některých čísel nemusí být jednoznačné. Typickým příkladem jsou nepřípustné reprezentace čísel v bázi $\beta \in \mathbb{Z}$ končící periodickým sufixem $(\beta-1)^\omega$. Pro všechna $0 \leq n \leq \beta-2$ mají řetězce $n(\beta-1)^\omega$ a $(n+1)0^\omega$ stejnou hodnotu. Jelikož každé $\beta \in \mathbb{Z}$ je kořenem polynomu $X-\beta$, vhodným přepisovacím pravidlem v bázi β je řetězec $1(-\beta)$, zkráceně zapsáno $1\bar{\beta}$. Je zřejmé, že β-reprezentace ve tvaru $n\bullet(\beta-1)^\omega$ lze převést na přípustnou reprezentaci stejného čísla (tzv. β-rozvoj) spočetně mnoha aplikacemi pravidla $1\bar{\beta}$, a to na všech pozicích $i \leq -1$.

Libovolné nekonečné slovo nazveme ***β*-přípustné** (**přípustné**) právě tehdy, když hraje roli β-rozvoje nějakého reálného čísla. Zajímavější podmínky pro přípustnost nalezneme v soustavách s neceločíselnou bází $\beta > 1$, kde lze pro ověření přípustnosti reprezentací použít kritérium z práce W. Parryho [14]. Nejdříve potřebujeme definici tzv. **Rényiho rozvoje jedničky** $d_\beta(1)$. Ten je definován jako

$$d_\beta(1) = t_1 t_2 t_3 \cdots, \quad \text{kde} \quad t_1 = \lfloor \beta \rfloor, \quad t_2 t_3 \cdots = d_\beta(\beta - \lfloor \beta \rfloor)$$

a $d_\beta()$ označuje řetězec podle Věty 1.5.

Jako **nekonečný Rényiho rozvoj jedničky** $d^*_\beta(1) = t'_1 t'_2 t'_3 \cdots$ se označuje řetězec

$$d^*_\beta(1) = \begin{cases} (t_1 \cdots t_{k-1}(t_k - 1))^\omega & \text{pokud} \quad d_\beta(1) = t_1 \cdots t_k 0^\omega \quad \text{pro} \quad t_k \neq 0, \\ d_\beta(1) & \text{jinak.} \end{cases}$$

Věta 2.3. *[14] β-reprezentace $x = x_k x_{k-1} \cdots x_0 \bullet x_{-1} \cdots \in \mathcal{A} = \{0, 1, \ldots, \lceil \beta \rceil - 1\}$ je **přípustná** právě tehdy, když*

$$0^\omega \preceq_{lex} x_l x_{l-1} \cdots \prec_{lex} d^*_\beta(1) \quad \textit{pro všechna } l \leq k.$$

Pro jednoduchost navážeme na Příklad 1.4, kde $\beta = \tau = \frac{1+\sqrt{5}}{2}$.

Příklad 2.4. *Nechť $\beta = \tau = \frac{1+\sqrt{5}}{2}$. Protože $d^*_\beta(1) = (10)^\omega$, z Věty 2.3 vyplývá, že přípustné β-reprezentace neobsahují faktor 11 a nemají sufix $(10)^\omega$.*

Uvažujme číslo $x = 11\frac{1+\sqrt{5}}{2} + 7$ *a jednu z jeho* β*-reprezentací,* $111011\bullet$*. Aplikací přepisovacího pravidla* $1\bar{1}\bar{1}$*, které je odvozeno od minimálního polynomu* $X^2 - X - 1$*, můžeme z této nepřípustné reprezentace získat přípustnou.*

$$
\begin{array}{cccccccccc}
 & & & 1 & 1 & 1 & 0 & 1 & 1 & \bullet \\
+ & & & & & & 1 & \bar{1} & \bar{1} & \\
\hline
 & & & 1 & 1 & 1 & 1 & 0 & 0 & \bullet \\
+ & 1 & \bar{1} & \bar{1} & & & & & & \\
\hline
 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \bullet \\
+ & & & 1 & \bar{1} & \bar{1} & & & & \\
\hline
 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & \bullet
\end{array}
$$

Snadno lze ověřit, že $1010000\bullet$ *je přípustná* β*-reprezentace (a tedy* β*-rozvoj)* x*.*

2.2 Normalizace po sčítání

Následující algoritmy algoritmy pracují s reprezentacemi čísel v celočíselné bázi $\beta \geq 2$ nad symetrickou abecedou $\mathcal{A} = \{-a, -a+1, \dots a\}$. Nad abecedou $\mathcal{A}$ lze realizovat odčítání stejně jako sčítání, což je výhoda oproti běžnější abecedě $\{0, 1, \dots, a\}$. Pro zjednodušení uvažujme pouze konečné β-reprezentace čísel. Obsah této podkapitoly byl přejat z publikace [7], případně z prací v ní citovaných.

Je-li $a < \frac{\beta-1}{2}$, existují čísla, jejichž β-reprezentace nelze vyjádřit symboly z abecedy $\mathcal{A}$. Takovouto abecedu nazýváme nekompletní. Pokud je $a > \frac{\beta-1}{2}$, existují čísla, která mají více než jednu β-reprezentaci na $\mathcal{A}$. V tomto případě nazveme $\mathcal{A}$ redundantní abecedou. Jestliže $a = \frac{\beta-1}{2}$, $\mathcal{A}$ je úplná, ale nikoli redundantní, abeceda. Vyskytují-li se v β-reprezentaci symboly $\{-a, \dots, -1\}$, zapisujeme je jako $\bar{a}, \dots, \bar{1}$.

Nechť $x, y \in \mathbb{R}$ mají β-reprezentace nad abecedou $\mathcal{A} = \{-a, -a+1, \dots, a\}$. Sečteme-li reprezentace x a y po složkách, dostaneme řetězec $z = z_n z_{n-1} \cdots z_0$, kde $z_i = x_i + y_i$. Je zřejmé, že z nemusí být řetězcem nad $\mathcal{A}$, ale obecně nad abecedou $\mathcal{A} + \mathcal{A} = \{-2a, -2a+1, \dots, 2a\}$. Chceme-li z vyjádřit jako β-reprezentaci nad abecedou $\mathcal{A}$, musíme provést tzv. **normalizaci**, tj. přepsat reprezentaci $z \in \mathcal{A} + \mathcal{A}$ na reprezentaci $v \in \mathcal{A}$, aby z a v byly reprezentacemi stejného čísla.

V následujících odstavcích uvedeme dvě podmínky. Budeme vyžadovat, aby používané reprezentace umožňovaly na základě první nenulové cifry určit znaménko daného čísla a také aby abeceda $\{-a, -a+1, \dots, a\}$ byla redundantní, čehož později využijeme u paralelních algoritmů.

Aby bylo možné určit, zda-li je libovolné číslo z kladné nebo záporné pouze podle nejlevějšího nenulového symbolu β-reprezentace, musí $a \leq \beta - 1$. Tuto podmínku si ilustrujeme na následujícím příkladu.

Poznámka 2.5. *Uvažujme konečnou reprezentaci $x = 1 \bullet x_1 x_2 \cdots x_n$ nad abecedou $\mathcal{A} \in \{-a, \cdots, a\}$. Platí*

$$1 \bullet x_1 x_2 \cdots x_n = 1 + \sum_{i=1}^{n} \frac{x_i}{\beta^i}$$

Pokud chceme, aby cifra 1 na nejlevější pozici zaručila kladnost čísla x, musí být $x > 0$, nezávisle na zbytku reprezentace. Pro odhad spodní meze reprezentace $x = 1 \bullet x_1 x_2 \cdots$ platí

$$\begin{aligned} 1 \bullet x_1 x_2 \cdots x_n &\geq 1 + \sum_{i=1}^{n} \frac{-a}{\beta} \\ &> 1 - a \sum_{i=1}^{\infty} \left(\frac{1}{\beta}\right)^i \\ &> 1 - a \frac{1}{\beta\left(1 - \frac{1}{\beta}\right)} = 1 - \frac{a}{\beta - 1} \end{aligned}$$

Jestliže má být x kladné, musí platit

$$\frac{a}{\beta - 1} \leq 1 \Leftrightarrow a \leq \beta - 1.$$

Dále, pokud je abeceda $\{-a, -a+1, \ldots, a\}$ redundantní, potom $\beta \leq 2a$. Nejdříve pro zjednodušení uvažujme, že $\beta/2 < a \leq \beta - 1$. Pro normalizaci z použijeme následující Algoritmus 2.1 publikovaný A. Avizienisem [1].

Algoritmus 2.1 [1] Přepis z abecedy $\{-2a, -2a+1, \ldots, 2a\}$ do abecedy $\mathcal{A} = \{-a, -a+1, \ldots, a\}$, je-li $\beta/2 < a \leq \beta - 1$

Vstup: $z = z_n z_{n-1} \cdots z_0 \in \{-2a, -2a+1, \ldots, 2a\}^+$
Výstup: $v = v_{n+1} v_n \cdots v_0 \in \mathcal{A}^+$

```
for i = 0 to n do
    if a ≤ z_i ≤ 2a then
        r_{i+1} ← 1 and s_i ← z_i − β
    end if
    if −2a ≤ z_i ≤ −a then
        r_{i+1} ← −1 and s_i ← z_i + β
    end if
    if −a + 1 ≤ z_i ≤ a − 1 then
        r_{i+1} ← 0 and s_i ← z_i
    end if
end for
```

12: **for** $i = 0$ to n **do**
13: $v_i \leftarrow s_i + r_i$
14: **end for**
15: $v_{n+1} \leftarrow r_{n+1}$
16: **return** v

Uvážíme-li $\beta = 2a$, výše uvedený algoritmus nemusí fungovat. Pokud operace na řádcích 3 nebo 6 nastaví v kroku i přenos r_{i+1} na 1 (resp. -1) a je-li $z_{i+1} = -a$ (resp. $z_{i+1} = a$), operace v kroku $i+1$ nastaví $s_{i+1} = \mp a \pm \beta = \pm a$. Výsledná hodnota v_{i+1} bude nastavena na řádku 13 na hodnotu $a + 1$ (resp. $-a - 1$), tedy na symbol, který není prvkem $\mathcal{A}$. V případě, že $\beta = 2a$, je tedy třeba zohlednit znaménko prvku napravo od právě upravovaného. Toto zobecnění provedla dvojice C. Y. Chow a J.E. Robertson [4]. Výsledkem je Algoritmus 2.2.

Algoritmus 2.2 [4] Přepis z abecedy $\{-2a, -2a + 1, \ldots, 2a\}$ do abecedy $\mathcal{A} = \{-a, -a + 1, \ldots, a\}$, je-li $\beta/2 \leq \alpha \leq \beta - 1$

Vstup: $z = z_n z_{n-1} \cdots z_0 \in \{-2a, -2a + 1, \ldots, 2a\}^+$
Výstup: $v = v_n v_{n-1} \cdots v_0 \in \mathcal{A}^+$
1: **for** $i = 0$ to $n - 1$ **do**
2: **if** $a + 1 \leq z_i \leq 2a$ **then**
3: $r_{i+1} \leftarrow 1$ **and** $s_i \leftarrow z_i - \beta$
4: **end if**
5: **if** $-2a \leq z_i \leq -a - 1$ **then**
6: $r_{i+1} \leftarrow -1$ **and** $s_i \leftarrow z_i + \beta$
7: **end if**
8: **if** $-a + 1 \leq z_i \leq a - 1$ **then**
9: $r_{i+1} \leftarrow 0$ **and** $s_i \leftarrow z_i$
10: **end if**
11: **if** $z_i = a$ **then**
12: **if** $z_{i-1} \leq 0$ **then**
13: $r_{i+1} \leftarrow 0$ **and** $s_i \leftarrow z_i$
14: **end if**
15: **if** $z_{i-1} > 0$ **then**
16: $r_{i+1} \leftarrow 1$ **and** $s_i \leftarrow z_i - \beta$
17: **end if**
18: **end if**
19: **if** $z_i = -a$ **then**
20: **if** $z_{i-1} \geq 0$ **then**
21: $r_{i+1} \leftarrow 0$ **and** $s_i \leftarrow z_i$
22: **end if**

```
23:       if  z_{i-1} < 0 then
24:         r_{i+1} ← −1 and s_i ← z_i + β
25:       end if
26:     end if
27: end for
28: for  i = 0 to n − 1 do
29:     v_i ← s_i + r_i
30: end for
31: v_n ← s_n
32: return v
```

Na základě Algoritmu 2.1 a 2.2 si lze povšimnout, že normalizace po sčítání je realizovatelná (podobně jako úprava nepřípustných reprezentací na přípustné) opakovanou aplikací přepisovacího pravidla. V celočíselné bázi je to pravidlo ve tvaru $\pm(1\bar{\beta})$.

2.3 Paralelní algoritmy

Z uvedených Algoritmů 2.1 a 2.2 je patrné, že normalizaci po sčítání dvou čísel s reprezentacemi délky n pomocí těchto algoritmů lze provést v čase $T_{sekv.} = O(n)$. Oba algoritmy jsou však napsány tak, že každá iterace v libovolném cyklu je nezávislá na předchozích iteracích stejného cyklu. Z tohoto důvodu je možné jednotlivé iterace daných cyklů provádět paralelně. Přesto oba algoritmy obsahují datové závislosti mezi jednotlivými cykly. V pořadí druhý cyklus vyžaduje výsledky prvniho cyklu, proto při paralelním výpočtu musí na konci každého cyklu dojít k synchronizaci výpočetních prostředků a rozeslání vypočtených hodnot ostatním. Paralelizací těchto algoritmů můžeme snížit jejich časovou složitost při využití n výpočetních vláken na hodnotu $T_{par.} = O(1)$.

V této podkapitole popíšeme paralelní sčítací algoritmy rovněž publikované v práci [7]. Uvažujme bázi β, algebraické číslo splňující $|\beta| > 1$. O β řekneme, že má tzv. **silnou reprezentaci nuly**, pokud pro nějaké nezáporné celočíselné indexy k a h existují taková celá čísla $b_k, b_{k-1}, \ldots, b_0, \ldots, b_{-h}$, že β je kořenem výrazu

$$S(X) = b_k X^k + b_{k-1} X^{k-1} + \cdots + b_1 X + b_0 + \frac{b_{-1}}{X} + \cdots + \frac{b_{-h}}{X^h}$$

a zároveň platí

$$b_0 > 2M := 2 \sum_{i \in \{-h,\ldots,k\} \setminus \{0\}} |b_i|.$$

S se nazývá **silným polynomem** β. Algoritmus 2.3 reprezentuje sčítání β-reprezentací nad abecedou

$$\mathcal{A} = \{-a, \ldots, 0, \ldots, a\}, \text{kde } a = \left\lceil \frac{b_0 - 1}{2} \right\rceil + \left\lceil \frac{b_0 - 1}{2(b_0 - 2M)} \right\rceil M.$$

Vlastní Algoritmus 2.3 využívá tzv. vnitřní abecedu $\mathcal{A}' = \{-a', \ldots, 0, \ldots, a'\} \subset \mathcal{A}$, kde

$$a' = \left\lceil \frac{b_0 - 1}{2} \right\rceil$$

a dále je pro přehlednost označeno

$$c = \left\lceil \frac{b_0 - 1}{2(b_0 - 2M)} \right\rceil.$$

Poznámka 2.6. *Algoritmus 2.3 je uveden s mírnou modifikací, která snižuje režii při synchronizaci výpočetních prostředků. V původním algoritmu uvedeném v [7] se mezi řádky 2 a 3 předpokládá rozeslání hodnot z_i ostatním výpočetním prostředkům. To však není nutné, neboť operace na řádku 3 využívá v j-té iteraci pouze hodnotu z_j, která byla spočtena ve stejné iteraci daného cyklu (a tedy stejným vláknem či procesem).*

Algoritmus 2.3 [7] Sčítání v bázi β, která má silnou reprezentaci nuly

Vstup: $x = x_n x_{n-1} \cdots x_0 \in \mathcal{A}^+$
$y = y_n y_{n-1} \cdots y_0 \in \mathcal{A}^+$
Je-li to nutné, doplníme tyto reprezentace symboly 0.
Výstup: $z = z_{n+k} z_{n+k-1} \cdots z_{-h} \in \mathcal{A}^+$ **do in parallel**
1: **for** $i = -h$ to $n + k$ **do**
2: $z_i \leftarrow x_i + y_i$
3: najdi $q_i \in \{-c, \ldots, 0, \ldots, c\}$ takové, že $z_i - q_i b_0 \in \mathcal{A}'$
4: **end for**
5: **for** $i = -h$ to $n + k$ **do in parallel do**
6: $z_i \leftarrow z_i - \sum_{j=-h}^{k} q_{i-j} b_j$
7: **end for**
8: **return** z

Formální důkaz správnosti tohoto algoritmu lze najít v [7].

Lze si povšimnout, že tento algoritmus používá k normalizaci reprezentací přepisovací pravidlo $\pm(b_k b_{k-1} \cdots b_0 \cdots b_{-h})$.

Příklad 2.7. *Nechť je dána báze $\beta = \frac{1+\sqrt{5}}{2}$. β je kořenem silného polynomu*

$$S(X) = -X^4 + 7 - \frac{1}{X^4}$$

Z výše uvedených definic plyne

$$b_0 = 7 \qquad M = 2 \qquad c = 1 \qquad a' = 3 \qquad a = 5$$

Uvažujme čísla $x = x_8 \cdots x_0 = 5\bar{4}\bar{3}\bar{2}\bar{1}0123$ *a* $y = y_8 \cdots y_0 = 420\bar{2}\bar{3}\bar{3}3\bar{1}1$. *Nechť* $z = x + y$.

$x =$		5	$\bar{4}$	$\bar{3}$	$\bar{2}$	$\bar{1}$	0	1	2	3	$\bullet$
$y =$		4	2	0	$\bar{2}$	$\bar{3}$	$\bar{3}$	3	$\bar{1}$	1	$\bullet$
$z =$		9	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{4}$	$\bar{3}$	4	1	4	$\bullet$

Všechna z_i, *která nejsou součástí abecedy* $\{-3, \ldots, 3\}$, *musíme upravit. Úpravu provedeme tak, že aplikujeme přepisovací pravidlo* $\pm(\bar{1}00070001)$ *na pozici* i *tak, aby symbol* $z_i + 7$ *(resp.* $z_i - 7$*) náležel abecedě* $\{-3, \ldots, 3\}$.

	$z =$					9	$\bar{2}$	$\bar{3}$	$\bar{4}$	$\bar{4}$	$\bar{3}$	4	1	4	$\bullet$			
$+$	$0 =$	1	0	0	0	$\bar{7}$	0	0	0	1								
$+$	$0 =$				$\bar{1}$	0	0	0	7	0	0	0	$\bar{1}$					
$+$	$0 =$					$\bar{1}$	0	0	0	7	0	0	0	$\bar{1}$	$\bullet$			
$+$	$0 =$							1	0	0	0	$\bar{7}$	0	0	$\bullet$	0	1	
	$0 =$									1	0	0	0	$\bar{7}$	$\bullet$	0	0	0	1
	$z =$	1	0	0	$\bar{1}$	1	$\bar{2}$	$\bar{2}$	3	5	$\bar{3}$	$\bar{3}$	0	$\bar{4}$	$\bullet$	0	1	0	1

Výsledkem sčítání je $z = 100\bar{1}1\bar{2}\bar{2}35\bar{3}\bar{3}0\bar{4} \bullet 0101 \in \{-5, \ldots, 5\}^*$.

Jak lze vidět na předchozím příkladu, Algoritmus 2.3 pracuje s reprezentacemi nad poměrně velkou abecedou $\{-5, \ldots, 5\}$. Ve stejné bázi $\beta = \frac{1+\sqrt{5}}{2}$ však lze zapsat všechna nezáporná reálná čísla pomocí kanonické abecedy $\{0, 1\}$, v případě symetrické abecedy pomocí $\{-1, 0, 1\}$.

Pro každý řetězec $x = x_n ... x_m$ definujeme **váhu** řetězce $W(x)$ následovně:

$$W(x) = \sum_{i=m}^{n} |x_i|$$

Uvažujme bázi $\beta = \frac{1+\sqrt{5}}{2}$ a řetězec $x = x_n x_{n-1} \cdots x_0 \in \{-5, \ldots, 0, \ldots, 5\}$. Platí

$$-\beta^2 + 3 - \frac{1}{\beta^2} = 0$$

$\bar{1}030\bar{1}$ v bázi β tedy reprezentuje číslo 0. V našem kontextu se jedná o přepisovací pravidlo. Pokud pravidlo $\bar{1}030\bar{1}$ aplikujeme na řetězec x tak, abychom snížili absolutní hodnotu x_i o 3, vzroste absolutní hodnota na pozicích $i - 2$ a $i + 2$ maximálně o 1. Celková váha řetězce x se tedy sníží minimálně o 1. Tímto způsobem můžeme snadno zapsat x pomocí menší abecedy než při využití silného polynomu β.

Nechť je báze $|\beta| > 1$. β má tzv. **slabou reprezentaci nuly**, pokud pro nějaké nezáporné celočíselné indexy k a h existují taková celá čísla $b_k, b_{k-1}, \ldots, b_{-h}$, že β je kořenem výrazu

$$W(X) = b_k X^k + b_{k-1} X^{k-1} + \cdots + b_1 X + b_0 + \frac{b_{-1}}{X} + \cdots + \frac{b_{-h}}{X^h}$$

a zároveň platí

$$b_0 > M := \sum_{i \in \{-h,\ldots,k\} \setminus \{0\}} |b_i|$$

W se nazývá **slabý polynom** β.

Algoritmus 2.4 realizuje sčítání β-reprezentací nad abecedou

$$\mathcal{A} = \{-a, \ldots, 0, \ldots, a\}, \text{kde } a = \left\lceil \frac{b_0 - 1}{2} \right\rceil + M.$$

Pro vnitřní reprezentaci je použita, podobně jako u Algoritmu 2.3, abeceda $\mathcal{A}' = \{-a', \ldots, 0, \ldots, a'\}$, kde $a' = \left\lceil \frac{b_0 - 1}{2} \right\rceil$. Nechť jsou dány β-reprezentace $x = x_n x_{n-1} \cdots x_0 \in \mathcal{A}^*$ a $y = y_n y_{n-1} \cdots y_0 \in \mathcal{A}^*$ a jejich součet $z = x + y = z_n z_{n-1} \cdots z_0$. Symboly z_i jsou prvky abecedy {-2a, …, 0 , … 2a}. V prvním kroku lze řetězec z přepsat tak, aby neobsahoval symboly $-2a$ a $2a$. Takto lze zmenšovat abecedu, až po $\left\lceil \frac{a}{b_0 - M} \right\rceil$ krocích budou všechny symboly z_i součástí abecedy $\mathcal{A}$. Tento počet kroků označíme s, tj. $s = \left\lceil \frac{a}{b_0 - M} \right\rceil$.

Poznámka 2.8. *Obdobně jako Algoritmus 2.3, je i Algoritmus 2.4 mírně upraven tak, aby lépe zobrazoval datové závislosti a potřebu synchronizace výpočetních prostředků.*

Algoritmus 2.4 [7] Sčítání v bázi β, která má slabou reprezentaci nuly

Vstup: $x = x_n x_{n-1} \cdots x_0 \in \mathcal{A}^+$
$y = y_n y_{n-1} \cdots y_0 \in \mathcal{A}^+$
Je-li to nutné, doplníme tyto reprezentace symboly 0.

Výstup: $z = z_{n+ks} z_{n+ks-1} \cdots z_{-hs} \in \mathcal{A}^+$

```
1:  for i = −hs to n + ks do in parallel do
2:      z_i ← x_i + y_i
3:  end for
4:  for j = 1 to s do
5:      for i = −hs to n + ks do in parallel do
6:          if z_i ∈ A' then
7:              q_i ← 0
8:          else
9:              q_i ← sgn(z_i)
10:         end if
11:     end for
```

12: **for** $i = -hs$ to $n + ks$ **do in parallel do**
13: $z_i \leftarrow z_i - \sum_{l=-h}^{k} q_{i-l} b_l$
14: **end for**
15: **end for**
16: **return** z

Důkaz správnosti tohoto algoritmu je uveden v [7].

Příklad 2.9. *Nechť je dána báze* $\beta = \frac{1+\sqrt{5}}{2}$. *$\beta$ je kořenem slabého polynomu*

$$W(X) = -X^2 + 3 - \frac{1}{X^2}$$

Z definic uvedených výše vyplývá

$$b_0 = 3 \qquad M = 2 \qquad s = 3 \qquad a' = 1 \qquad a = 3$$

Uvažujme čísla $x = x_4 \cdots x_0 = \bar{3}2\bar{3}0\bar{3}$ *a* $y = y_4 \cdots y_0 = \bar{3}0\bar{3}1\bar{2}$. *Nechť* $z = x + y$.

		$x =$					$\bar{3}$	2	$\bar{3}$	0	$\bar{3}$	•			
	+	$y =$					$\bar{3}$	0	$\bar{3}$	1	$\bar{2}$	•			
		$z =$					$\bar{6}$	2	$\bar{6}$	1	$\bar{5}$	•			
krok 1	+	$0 =$			$\bar{1}$	0	3	0	$\bar{1}$						
	+	$0 =$				1	0	$\bar{3}$	0	1					
	+	$0 =$					$\bar{1}$	0	3	0	$\bar{1}$				
	+	$0 =$							$\bar{1}$	0	3	•	0	$\bar{1}$	
		$z =$			$\bar{1}$	1	$\bar{4}$	$\bar{1}$	$\bar{5}$	2	$\bar{3}$	•	0	$\bar{1}$	
krok 2	+	$0 =$			$\bar{1}$	0	3	0	$\bar{1}$						
	+	$0 =$					$\bar{1}$	0	3	0	$\bar{1}$				
	+	$0 =$						1	0	$\bar{3}$	0	•	1		
	+	$0 =$							$\bar{1}$	0	3	•	0	$\bar{1}$	
		$z =$			$\bar{2}$	1	$\bar{2}$	0	$\bar{4}$	$\bar{1}$	$\bar{1}$	•	1	$\bar{2}$	
krok 3	+	$0 =$	$\bar{1}$	0	3	0	$\bar{1}$								
	+	$0 =$			$\bar{1}$	0	3	0	$\bar{1}$						
	+	$0 =$					$\bar{1}$	0	3	0	$\bar{1}$				
	+	$0 =$									$\bar{1}$	•	0	3	0	$\bar{1}$
		$z =$	$\bar{1}$	0	0	1	$\bar{1}$	0	$\bar{2}$	$\bar{1}$	$\bar{3}$	•	1	1	0	$\bar{1}$

$z = \bar{1}001\bar{1}0\bar{2}\bar{1}\bar{3}\bullet 110\bar{1}$ *je součtem* x *a* y *a řetězcem nad abecedou* $\{-3, \ldots, 3\}$.

Kapitola 3

DUG vlastnost algebraických těles

Některá tělesa mají vlastnost, že v nich lze každé algebraické celé číslo zapsat jako konečný součet různých jednotek. Tato tělesa nazýváme **DUG** (anglicky **distinct unit generated**). Této vlastnosti si všiml B. Jacobson [9] a dokázal ji u těles $\mathbb{Q}(\sqrt{2})$ a $\mathbb{Q}(\sqrt{5})$. Domníval se, že žádná další kvadratická tělesa tuto vlastnost nesplňují. Jeho hypotézu pak potvrdil J.Śliwa [16] a dokázal, že žádné čistě kubické těleso, tedy těleso $\mathbb{Q}(\sqrt[3]{n})$, pro $n \in \mathbb{Z}$ a $\sqrt[3]{n} \notin \mathbb{Z}$, není DUG. Další výsledky rozšiřující klasifikaci DUG vlastností u těles nižších stupňů lze nalézt v [2] [3] [18].

Následující definice pochází z práce J. Thuswaldnera a V. Zieglera [18] a určuje, jak vzdálené je dané těleso DUG vlastnosti.

Definice 3.1. *Nechť O_K je okruh celých čísel v tělese K a $\alpha \in O_K$. Předpokládejme, že α lze zapsat jako lineární kombinaci jednotek $\epsilon_1, \ldots, \epsilon_l \in U_K$*

$$\alpha = a_1\epsilon_1 + \cdots + a_l\epsilon_l,$$

*kde $a_1 \geq \cdots \geq a_l \geq 1$ jsou celá čísla. Zvolíme-li reprezentaci s nejmenším možným a_1, potom $\omega(\alpha) = a_1$ je tzv. **jednotková výška** (anglicky **unit sum height**) α. Jestliže $\alpha = 0$, pak $\omega(\alpha) = \omega(0) = 0$. Pokud α nelze zapsat jako lineární kombinaci jednotek, $\omega(\alpha) = \infty$. Jednotkovou výšku tělesa K definujeme předpisem*

$$\omega(K) = max\,\{\omega(\alpha) \,|\, \alpha \in O_K\}$$

Je-li $\omega(K) = 1$, těleso K je DUG.

Nejaktuálnější výsledek na téma klasifikace DUG těles pochází z prací [8] [6]. Ten je ve formě seznamu obsahujícího všechna DUG tělesa čtvrtého stupně, která jsou současně tzv. **plně komplexní**, anglicky **totally complex** (tedy tělesa $\mathbb{Q}(\gamma)$, kde minimální polynom čísla γ nemá žádné reálné kořeny). Doposud však nebylo dokázáno, že všechna tělesa z daného seznamu DUG vlastnost opravdu splňují.

Věta 3.2. *[8] Každé plně komplexní těleso čtvrtého stupně s DUG vlastností je nutně jedno z těles v následujícím seznamu (ζ_n značí primitivní n-tý kořen z jednotky).*

- $\mathbb{Q}(\zeta_\mu)$, *kde* $\mu = 5, 8, 12$,
- $\mathbb{Q}(\gamma)$, *kde* γ *je kořenem jednoho z polynomů* $X^4 - X + 1$, $X^4 + 2X^2 - 2X + 1^{\triangle}$, $X^4 + X^2 - X + 1$, $X^4 - X^3 + X + 1^{\triangledown}$, $X^4 - X^3 + X^2 + X + 1^{\triangledown}$, $X^4 - X^3 + 2X^2 - X + 2^{\triangle}$,
- $\mathbb{Q}(\sqrt{a + ib})$ *pro* $(a, b) = (1, 1), (1, 2), (1, 4), (7, 4)^{\triangle}$,
- $\mathbb{Q}(\sqrt{a + \zeta_3 b})$ *pro* $(a, b) = (2, 1)$, $(4, 1)$, $(8, 1)$, $(3, 2)$, $(4, 3)$, $(7, 3)$, $(11, 3)$, $(5, 4)$, $(9, 4)$, $(13, 4)$, $(12, 5)$, $(11, 7)$, $(9, 8)$, $(15, 11)$, $(19, 11)^{\triangle}$, $(17, 12)^{\triangle}$, $(17, 16)^{\triangle}$,
- $\mathbb{Q}(\zeta_3, \sqrt{d})$ *pro* $d = 5, 6, 21$,
- $\mathbb{Q}(\zeta_4, \sqrt{5})$,
- $\mathbb{Q}\left(\sqrt{-1 - \sqrt{2}}\right)$ *nebo* $\mathbb{Q}\left(\sqrt{-\frac{1+\sqrt{5}}{2}}\right)$.

Věta 3.3. *[6] Pokud je K plně komplexní těleso čtvrtého stupně ze seznamu ve Větě 3.2, váha tohoto tělesa splňuje $\omega(K) \leq 3$. Tělesa označená symbolem $\triangle$ navíc splňují $\omega(K) \leq 2$. Jednotková výška těles označených $\triangledown$ splňuje podmínku $\omega(K) \leq 3$. Všechna ostatní tělesa jsou DUG, tedy $\omega(K) = 1$.*

Součástí této práce je program pro přepisování reprezentací čísel v bázi γ, která je kořenem polynomu $X^4 - X + 1$. Proto si nyní přiblížíme těleso $\mathbb{Q}(\gamma)$, u kterého byla DUG vlastnost dokázána (narozdíl od zbylých DUG těles) kombinatoricky a tento výsledek publikovaný v práci [5] si na následujících stranách shrneme.

Nechť $K = \mathbb{Q}(\gamma)$, kde γ je libovolným kořenem polynomu $X^4 - X + 1$. Kořeny tohoto polynomu jsou čtyři nereálná čísla ve dvou komplexně sdružených párech. Označíme je $\gamma_1, \bar{\gamma}_1$ a $\gamma_2, \bar{\gamma}_2$, přičemž platí $|\gamma_1| > 1 > |\gamma_2|$. Zvolíme-li $\gamma \in \{\gamma_1, \bar{\gamma}_1\}$, má smysl uvažovat obecné reprezentace čísel v komplexní bázi γ ve smyslu Věty 3.4. Pokud zvolíme $\gamma \in \{\gamma_2, \bar{\gamma}_2\}$ v absolutní hodnotě menší než jedna, můžeme využít faktu uvedeného v [12], že pro každé algebraické číslo $\gamma \neq 0$ platí $\mathbb{Q}(\gamma) = \mathbb{Q}\left(\frac{1}{\gamma}\right)$, a zvolit $\gamma \in \left\{\frac{1}{\gamma_2}, \frac{1}{\bar{\gamma}_2}\right\}$ v absolutní hodnotě větší než jedna. Bez újmy na obecnosti tedy můžeme dále předpokládat, že $|\gamma| > 1$.

Množina $\{1, \gamma, \gamma^2, \gamma^3\}$ tvoří integrální bázi okruhu celých čísel

$$O_K = \left\{ \sum_{n=-\infty}^{\infty} u_n \gamma^n \right\},$$

pro $u_n \in \mathbb{Z}$ a $u_n \neq 0$ pro maximálně konečně mnoho indexů n. Ve smyslu Věty 1.6 můžeme zapsat všechna celá čísla $\alpha \in O_K$ jako γ-reprezentace ve tvaru

$$\alpha = \cdots u_1 u_0 \bullet u_{-1} u_{-2} \cdots ,$$

nad abecedou $\mathbb{Z}$. Bez újmy na obecnosti však můžeme uvažovat pouze konečné γ-reprezentace, tedy reprezentace ve tvaru $v_k \cdots v_0 \bullet v_{-1} \cdots v_l$, kde $l, k \in \mathbb{Z}$.

Jelikož γ je fundamentální jednotkou tělesa K(viz [6]) a zároveň K neobsahuje žádný nereálný kořen z jednotky(pouze ± 1), grupu jednotek U_K lze zapsat takto

$$U_K = \left\{ \pm \gamma^k | k \in \mathbb{Z} \right\}$$

Má-li tedy být každé celé číslo $\alpha \in O_K$ vyjádřeno součtem různých jednotek, musí pro každé α existovat γ-reprezentace

$$v_k \cdots v_0 \bullet v_{-1} \cdots v_l, \quad \text{kde } v_i \in \{\bar{1}, 0, 1\} \text{ a } l, k \in \mathbb{Z}$$

Existenci takového přepisu pro každý prvek okruhu O_K ukážeme v následujících dvou podkapitolách. Budeme následovat důkaz uvedený v [6] a [5], jehož základem je aplikace přepisovacího pravidla $100\bar{1}1$, které odpovídá minimálnímu polynomu $X^4 - X + 1$.

3.1 Přepisování řetězců do $\{\bar{2}, \bar{1}, 0, 1, 2\}^*$

Nejdříve ukážeme, že všechna $\alpha \in O_K$ mají γ-reprezentaci $v \in \{\bar{2}, \bar{1}, 0, 1, 2\}$, která je tzv. řídká, tj. neobsahuje nenulové cifry na sousedních pozicích.

Věta 3.4. *[5] Všechny γ-reprezentace $u \in \mathbb{Z}^*$ lze aplikací přepisovacího pravidla $100\bar{1}1$ přepsat na γ-reprezentace $v = v_k \cdots v_0 \bullet v_{-1} \cdots v_l$, pro které platí tyto podmínky*

- $v_i \in \{\bar{2}, \bar{1}, 0, 1, 2\}$ *pro všechna* $i \in \{k, \ldots, l\}$
- $v_{i+m} \cdot v_i > 0 \Rightarrow m \notin \{1, 2, 4\}$
- $v_{i+m} \cdot v_i < 0 \Rightarrow m \notin \{1, 3\}$
- $v_{i+2} \cdot v_i < 0 \Rightarrow v_{i+4} = v_{i+5} = 0$
- $v_{i+3} \cdot v_i > 0 \Rightarrow v_{i+6} = 0$

Zřetězení symbolů v_i a v_j značíme $v_i v_j$. Součin dvou celých čísel je označen jako $v_i \cdot v_j$.

Snadno lze nahlédnout, že přepisovací pravidlo $w_1 = 100\bar{1}1$ změní pouze γ-reprezentaci čísla, ale nikoli jeho hodnotu. Několikanásobnou aplikací pravidla w_1 můžeme získat pravidla $w_2 = 10000003000001$, $w_3 = 10001001\bar{1}1$ a $w_4 = 11100001$. Pro ilustraci si ukážeme vznik přepisovacího pravidla w_2.

$$\begin{array}{cccccccccccccc}
 & 1 & 0 & 0 & \bar{1} & 1 & & & & & & & & \\
+ & & & & 1 & 0 & 0 & \bar{1} & 1 & & & & & \\
+ & & & & & \bar{1} & 0 & 0 & 1 & \bar{1} & & & & \\
+ & & & & & & & 1 & 0 & 0 & \bar{1} & 1 & & \\
+ & & & & & & & & 1 & 0 & 0 & \bar{1} & 1 & \\
+ & & & & & & & & & 1 & 0 & 0 & \bar{1} & 1 \\
+ & & & & & & & & & & 1 & 0 & 0 & \bar{1} \\
\hline
 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}$$

Pravidla w_3 a w_4 vznikla analogicky.

Nesplňuje-li γ-reprezentace $u \in \mathbb{Z}^*$ podmínky Věty 3.4, lze ji aplikací pravidel w_1, w_2, w_3, w_4 přepsat následovně

(a) Pokud $|v_i| \geq 3$

	v_{i+7}	v_{i+6}	$\cdots$	v_i	$\cdots$	v_{i-5}	v_{i-6}
$\pm$	1	0	$\cdots$	3	$\cdots$	0	1
	$v_{i+7} \pm 1$	v_{i+6}	$\cdots$	$\underline{v_i \pm 3}$	$\cdots$	v_{i-5}	$v_{i-6} \pm 1$

(b) Pokud $v_{i+1}.v_i < 0$

	v_{i+4}	v_{i+3}	v_{i+2}	v_{i+1}	v_i
$\pm$	1	0	0	$\bar{1}$	1
	$v_{i+4} \pm 1$	v_{i+3}	v_{i+2}	$\underline{v_{i+1} \mp 1}$	$\underline{v_i \pm 1}$

(c) Pokud $v_{i+3}.v_i < 0$

	v_{i+3}	v_{i+2}	v_{i+1}	v_i	v_{i-1}
$\pm$	1	0	0	$\bar{1}$	1
	$\underline{v_{i+3} \pm 1}$	v_{i+2}	v_{i+1}	$\underline{v_i \mp 1}$	$v_{i-1} \pm 1$

(d) Pokud $v_{i+4}.v_i > 0$

	v_{i+4}	v_{i+3}	v_{i+2}	v_{i+1}	v_i
$\pm$	1	0	0	$\bar{1}$	1
	$\underline{v_{i+4} \pm 1}$	v_{i+3}	v_{i+2}	$\underline{v_{i+1} \mp 1}$	$v_i \pm 1$

(e) Pokud $v_{i+2}.v_i < 0$ a $v_{i+5}.v_i < 0$

	v_{i+9}	$\cdots$	v_{i+5}	v_{i+4}	v_{i+3}	v_{i+2}	v_{i+1}	v_i	v_{i-1}
$\pm$	1	$\cdots$	1	0	0	1	0	$\bar{1}$	1
	$v_{i+9} \pm 1$	$\cdots$	$\underline{v_{i+5} \pm 1}$	v_{i+4}	v_{i+3}	$\underline{v_{i+2} \pm 1}$	v_{i+1}	$\underline{v_i \mp 1}$	$v_{i-1} \pm 1$

(f) Pokud $v_{i+3}.v_i > 0$ a $v_{i+6}.v_i > 0$

	v_{i+10}	$\cdots$	v_{i+6}	v_{i+5}	v_{i+4}	v_{i+3}	v_{i+2}	v_{i+1}	v_i
$\pm$	1	$\cdots$	1	0	0	1	0	$\bar{1}$	1
	$v_{i+10} \pm 1$	$\cdots$	$\underline{v_{i+6} \pm 1}$	v_{i+5}	v_{i+4}	$\underline{v_{i+3} \pm 1}$	v_{i+2}	$\underline{v_{i+1} \mp 1}$	$v_i \pm 1$

(g) Pokud $v_{i+1}.v_i > 0$

	v_{i+1}	v_i	v_{i-1}	v_{i-2}	$\cdots$	v_{i-6}
$\pm$	1	1	1	0	$\cdots$	1
	$\underline{v_{i+1} \pm 1}$	$\underline{v_i \pm 1}$	$v_{i-1} \pm 1$	v_{i-2}	$\cdots$	$v_{i-6} \pm 1$

(h) Pokud $v_{i+2}.v_i > 0$

	v_{i+2}	v_{i+1}	v_i	v_{i-1}	$\cdots$	v_{i-5}
$\pm$	1	1	1	0	$\cdots$	1
	$\underline{v_{i+2} \pm 1}$	$v_{i+1} \pm 1$	$\underline{v_i \pm 1}$	v_{i-1}	$\cdots$	$v_{i-5} \pm 1$

Pravidla jsou aplikována tak, aby se absolutní hodnota číslic na podtržených pozicích snížila. Použití pravidel (a)-(f) vždy sníží váhu řetězce $W(v)$ definovanou v Sekci 2.3. Pravidla (g) a (h) váhu řetězce snížit nemusí, ale tyto pravidla je možné použít nejvýše konečně mnohokrát před dalším použitím jednoho z pravidel (a)-(f). Proto po konečně mnoha aplikacích pravidel w_1, w_2, w_3, w_4 získáme řetězec splňující podmínky Věty 3.4.

Vyskytují-li se v řetězci, který splňuje podmínky Věty 3.4, symboly ± 2, vzdálenost těchto symbolů je minimálně 6 pozic nebo faktor řetězce obsahující oba tyto symboly je prvkem množiny

$$F = \{\pm(20\bar{2}), \pm(2002), \pm(2000\bar{2}), \pm(200002), \pm(20000\bar{2}), \pm(20\bar{1}00\bar{2})\}.$$

Toho využijeme v druhé podkapitole přepisování řetězců.

3.2 Přepisování řetězců do $\{\bar{1}, 0, 1\}^*$

Zatímco v předchozí Podkapitole 3.1 nezáleželo na konkrétním pořadí, v jakém byla přepisovací pravidla aplikována (záleželo pouze na snížení váhy řetězce), zde bude aplikace přepisovacího pravidla přesně určena. Ukážeme, že lze danou reprezentaci projít zleva doprava a při každém výskytu symbolu ± 2 aplikovat přepisovací pravidlo $w_1 = 100\bar{1}1$.

Obsahuje-li řetězec jeden symbol ± 2 nebo více těchto symbolů, jejichž vzdálenost v řetězci je alespoň 6 pozic, lze je přepsat aplikací pravidla w_1 podle Tabulky 3.1

Tabulka 3.1: přepis symbolů ± 2 vzdálených alespoň 6 pozic

vstup	výstup
$00\bar{1}020$	$\bar{1}0\bar{1}110$
010020	$\bar{1}10110$
000020	$\bar{1}00110$
$\bar{1}00020$	$\bar{1}10011$

Aplikace pravidla $100\bar{1}1$ podle Tabulky 3.1 přepisuje cifry vzdálené nejvýše čtyři pozice nalevo a nejvýše jednu pozici napravo od symbolu ± 2. V případě více výskytů symbolů ± 2 vzdálených alespoň 6 pozic se tedy aplikace pravidel $100\bar{1}1$ navzájem neovlivňují.

Jsou-li dva po sobě jdoucí symboly ± 2 vzdálené pět nebo méně pozic, může nastat, že aplikace pravidla $100\bar{1}1$ na jednom výskytu cifry ± 2 může změnit cifry na již dříve přepsaných pozicích. Aby se zabránilo případnému vzniku nových symbolů ± 2, je třeba zvážit všechny případy, které mohou nastat. Všechny možné případy pro vzdálenost 5 znázorňuje Tabulka 3.2.

Tabulka 3.2: přepis symbolů ± 2 vzdálených 5 pozic

vstup	přepsání levého symbolu ± 2	výstup
2000020	1000020	$1\bar{1}00110$
	1100020	1000110
$\bar{2}000020$	$\bar{1}000020$	$\bar{1}\bar{1}00110$
	$\bar{1}\bar{1}00020$	$\bar{1}\bar{1}10011$
$(0)\bar{2}010020$	$\bar{1}010020$	$\bar{1}\bar{1}10110$
	$(0)\bar{1}\bar{1}10020$	$(1)\bar{1}\bar{1}11011$

V případě, že symboly ± 2 jsou vzdálené 4 pozice, lze je přepsat (obdobně jako u předchozího případu) přepsáním levého a následně pravého symbolu ± 2, jak znázorňuje Tabulka 3.3.

Tabulka 3.3: přepis symbolů ± 2 vzdálených 4 pozice

vstup	přepsání levého symbolu ± 2	výstup
$\bar{2}00020$	$\bar{1}00020$	$\bar{1}10011$
	$1\bar{1}0020$	$\bar{1}00011$

Přepsání symbolů ± 2 vzdálených 3 pozice znázorňuje Tabulka 3.4.

Tabulka 3.4: přepis symbolů ± 2 vzdálených 3 pozice

vstup	přepsání levého symbolu ± 2	výstup
020020	011020	$\bar{1}11110$
	110020	010110
	010020	$\bar{1}10110$

Jestliže jsou symboly ± 2 vzdálené 2 pozice, analogicky k předchozím případům lze přepsat nejdříve levý a následně pravý symbol ± 2 podle Tabulky 3.5.

Tabulka 3.5: přepis symbolů ± 2 vzdálených 2 pozice

vstup	přepsání levého symbolu ± 2	výstup
$00\bar{2}020$	$00\bar{1}\bar{1}20$	$\bar{1}0\bar{1}010$
	$0\bar{1}\bar{1}020$	$\bar{1}\bar{1}\bar{1}110$

Každý řetězec splňující podmínky Věty 3.4 lze tedy přepsat na řetězec nad abecedou $\{-1, 0, 1\}$ a tím je dokázána DUG vlastnost u tělesa $\mathbb{Q}(\gamma)$, kde γ je kořenem polynomu $X^4 - X + 1$. Podrobnější vysvětlení platnosti výše uvedeného přepisu lze najít v [5].

Kapitola 4

Implementace

Tato kapitola se zabývá implementací dvou programů, jež jsou součástí této práce.

4.1 Zadání

V této podkapitole jsou uvedeny funkční a nefunkční požadavky na požadované programy.

4.1.1 Přepisování řetězců

Úkolem této bakalářské práce je vytvořit program přepisující reprezentace čísel v bázi γ, která je kořenem polynomu $X^4 - X + 1$, na reprezentace nad abecedou $\{-1, 0, 1\}$. Jedná se tedy o program, který každé algebraické celé číslo $x \in O_K$ v tělese $K = \mathbb{Q}(\gamma)$ vyjádří jako součet různých jednotek.

4.1.1.1 Funkční požadavky

- Program načte posloupnost čísel reprezentující γ-reprezentaci čísla a vypíše γ-reprezentaci stejného čísla nad abecedou $\{-1, 0, 1\}$ včetně zlomkové tečky.
- Načtení probíhá ze standardního vstupu nebo ze zadaného souboru.
- Výstup programu je možné zobrazit na standardní výstup nebo uložit do souboru.

4.1.1.2 Nefunkční požadavky

- Program běží na systému Windows (7 a vyšší).
- Program běží na lokálním systému, není vyžadována komunikace či ovládání přes internet.

- Navrhovaný program je přehledný a jednoduchý na ovládání, grafické rozhraní není vyžadováno.

4.1.2 Paralelní sčítání

Druhým úkolem této práce je vytvořit program realizující paralelní sčítání dvou reprezentací čísel s využitím silného nebo slabého polynomu podle Algoritmů 2.3 a 2.4.

4.1.2.1 Funkční požadavky

- Program načte posloupnost čísel (polynom) a zkontroluje, zda-li se jedná o silný či slabý polynom.
- Program načte dvě posloupnosti čísel (reprezentace čísel), sečte je a sníží cifry součtu na původní abecedu danou silným či slabým polynomem.
- Načtení probíhá ze standardního vstupu nebo ze zadaného souboru.
- Je možné měnit počet výpočetních vláken.

4.1.2.2 Nefunkční požadavky

- Program běží na systému Windows (7 a vyšší).
- Program běží na lokálním systému, není vyžadována komunikace či ovládání přes internet.
- Navrhovaný program je přehledný a jednoduchý na ovládání, grafické rozhraní není vyžadováno.

4.2 Výběr programovacího jazyka

Při výběru programovacího jazyku jsem se rozhodoval na základě rychlosti, paměťové náročnosti, složitosti syntaxe a svých zkušeností s danými jazyky. Jako hlavní kandidáty jsem vybral C++, Javu a Python.

Všechny výše uvedené jazyky umožňují používat objektově orientované paradigma, které se k řešení tohoto problému nabízí. Mezi výhody objektového programování patří především přehlednost a znovupoužitelnost napsaného kódu a zapouzdření dat.

Hlavní výhodou C++ je, že se jedná o kompilovaný jazyk, což ve většině případů znamená, že je rychlejší než interpretované jazyky. Práce s pamětí narozdíl od Javy a Pythonu není v C++ automatická a je tedy nutné alokovanou paměť uvolňovat. Z pohledu programátora se proto může jevit jednodušší programovat v Javě, ale za cenu vyššího využití paměti a menší kontroly nad ní.

Syntaxe C++ je velmi podobná Javě, přesto je složitější. Gramatika popisující kód C++ je daleko více závislá na kontextu.

Java je kompromisem mezi interpretovaným a kompilovaným jazykem. Kód napsaný v Javě se kompiluje do bytecódu, který se následně spouští na Java virtual machine. Z tohoto důvodu je výsledný program poměrně rychlý a přenositelný. Kód napsaný v Javě je sice přehledný, ale oproti Pythonu velmi dlouhý.

Výhodou Pythonu je především jeho jednoduchost a přehlednost. I když se syntaxí liší od C++ a Javy, lze v něm psát programy poměrně rychle, a to i díky velkému množství knihoven. Referenční implementace Pythonu je napsána v jazyce C. Co se týče paměťové náročnosti, Python sice využívá méně paměti než Java, ale v některých případech může být velmi pomalý.

Na základě výše uvedených faktů jsem pro implementaci zvolil programovací jazyk C++.

Paralelizaci je možné provést několika způsoby. Hlavním rozdílem je, zda paralelní výpočet probíhá nad distribuovanou nebo sdílenou pamětí. Jelikož má dle požadavků program běžet na lokálním systému, lze využít sdílené paměti. Mezi hlavní výhody programování nad sdílenou pamětí patří rychlé vytváření vláken, rychlé přepínání kontextu mezi vlákny a jednodušší komunikace mezi nimi. Hlavní nevýhodou je správa distribuované paměti. Je nutné zajistit, aby v případě zápisu do paměti jedním vláknem další vlákna neměla k tomuto paměťovému prostoru přístup. To lze provést pomocí různých synchronizačních prostředků, z nichž hlavní jsou tzv. mutexy, které zabrání, aby byl určitý kus zdrojového kódu vykonáván více vlákny současně. Další nevýhodou programování nad sdílenou pamětí je obtížné debugování programu. V případě špatné synchronizace vláken může dojít k časově závislým chybám, které vznikají v souvislosti s přepínáním kontextu jednotlivých procesů. Právě proto, že přidělování procesorového času mezi vlákna není vždy stejné, může být obtížné tyto chyby reprodukovat a tedy i odhalit příčinu jejich vzniku. V případě C++ řeší tyto nevýhody OpenMP - soubor direktiv pro překladač a knihovních procedur. OpenMP vytvoří určitý počet vláken a těmto vláknům pak přiděluje práci. Programátor tak pouze uvede, jaké části kódu mají být provedeny paralelně, nastaví vlastnosti proměnných a o samotnou synchronizaci se postará OpenMP. Jelikož je použití OpenMP snadné a zároveň dosahuje velmi dobrých výsledků, není pro daný program důvod pro použití jiného způsobů paralelizace.

4.3 Návrh tříd a funkcí

Pro požadovaný program je postačující jedna třída.

4.3.1 Třída NumString

NumString je třída reprezentující číselný řetězec. Všechny symboly (cifry), jsou ukládány v dynamicky alokovaném poli *numbers* délky *max_length*. Počet symbolů v poli udává proměnná *length*. Pozici zlomkové tečky značí proměnná *frac_point*. Třída obsahuje následující metody

- ***int readString(istream & input)*** je metoda, která načte ze vstupního streamu *input* čísla a uloží je do pole *numbers*.
- ***void debug(ostream & output)*** je metoda, která slouží pro výpis číselného řetězce do výstupního streamu *output*.
- ***int getMaxHeight(void);*** je metoda, která vrací jednotkovou výšku daného řetězce
- ***bool isMaxHeight(int number)*** je metoda, která vrací *true*, pokud je jednotková výška číselného řetězce nejvýše *number*. Je-li jednotková výška řetězce vyšší, metoda vrátí *false*.
- ***int getWeight(void)*** je metoda, která vrací váhu řetězce.
- ***int getLength(void)*** je metoda, která vrací počet platných číslic v textovém řetězci.
- ***int* getNumbers(void)*** je metoda, která vrací ukazatel na pole cifer *numbers*.
- ***void add(int number)*** je metoda, která přidá do pole *numbers* číslo *number*. V případě potřeby dojde z rozšíření pole *numbers*.
- ***void addVector(int offset, string vector, int op)*** je metoda, která za předpokladu *op=ADD* (1) přičte k číselnému řetezci řetězec *vector* na pozici *offset*. Je-li *op=SUBSTRACT* (2), dojde k odečtení řetězce *vector* na pozici *offset*.
- ***void resize(int direction)*** je metoda, která rozšíří pole *numbers* doleva(*direction=LEFT*(1)) nebo doprava(*direction=RIGHT*(2))
- ***int getFracPoint()*** je metoda, která vrátí pozici zlomkové tečky.
- ***void setFracPoint(int n)*** je metoda, která nastaví pozici zlomkové tečky *frac_point* na hodnotu *n*.
- ***int getLength()*** je metoda, která vrátí velikost pole *numbers*.
- ***int *getNumbers()*** je metoda, která vrátí pole *numbers*.

4.3.2 Funkce pro přepisování řetězců

Program přepisující reprezentace čísel v bázi γ, která je kořenem polynomu $X^4 - X + 1$, využívá kromě třídy *NumString* také 4 funkce, které nesouvisí obecně s číselnými řetězci, ale pouze s konkrétním tělesem $\mathbb{Q}(\gamma)$.

První funkce *bool isValid2(NumString)* kontroluje, zda-li textový řetězec splňuje podmínky Věty 3.4.

Druhá funkce *void rewrite2(NumString)* přepíše libovolný řetězec na řídký řetězec nad abecedou $\{-2, -1, 0, 1, 2\}$. Tato funkce využívá aplikací přepisovacích pravidel podle Podkapitoly 3.1. Tuto funkci lze popsat pseudokódem 4.1.

Algoritmus 4.1 Přepis řetězců z $\mathbb{Z}^+$ na $\{-2, -1, 0, 1, 2\}^+$

Vstup: $x = x_m x_{m-1} \cdots x_n \in \mathbb{Z}^+$
Je-li to nutné, doplníme tuto reprezentaci symboly 0.
Výstup: $x = x_k x_{k-1} \cdots x_l \in \{-2, -1, 0, 1, 2\}^+$

```
1:  while not isValid2(x) do
2:    for all x_i do
3:      if x_i * x_{i-1} < 0 then
4:        applyRule(100$\bar{1}$1) at position i − 4
5:      end if
6:    end for
7:    for all x_i do
8:      if x_i * x_{i-3} < 0 then
9:        applyRule(100$\bar{1}$1) at position i − 3
10:     end if
11:   end for
12:   for all x_i do
13:     if x_i * x_{i-4} > 0 then
14:       applyRule(100$\bar{1}$1) at position i − 4
15:     end if
16:   end for
17:   for all x_i do
18:     if x_i * x_{i-2} < 0 and x_i * x_{i-5} < 0 then
19:       applyRule(10001001010$\bar{1}$1) at position i − 9
20:     end if
21:   end for
22:   for all x_i do
23:     if x_i * x_{i-3} > 0 and x_i * x_{i-6} > 0 then
24:       applyRule(10001001010$\bar{1}$1) at position i − 10
25:     end if
26:   end for
```

```
27:    for all x_i do
28:      if x_i * x_{i-1} > 0 then
29:        applyRule(11100001) at position i − 1
30:      end if
31:    end for
32:    for all x_i do
33:      if x_i * x_{i-2} > 0 then
34:        applyRule(11100001) at position i − 2
35:      end if
36:    end for
37:  end while
38:  return
```

Další funkcí je funkce *void rewrite1(NumString)*, která přepíše řetězec splňující podmínky Věty 3.4 na řetězec nad abecedou $\{-1, 0, 1\}$. Přepisování probíhá podle Podkapitoly 3.2 a může být popsáno následujícím pseudokódem 4.2

Algoritmus 4.2 Přepis řetězců z $\{-2, -1, 0, 1, 2\}^+$ na $\{-1, 0, 1\}^+$

Vstup: $x = x_m x_{m-1} \cdots x_n \in \{-2, \cdots, 2\}^+$ splňující podmínky z Věty 3.4
Je-li to nutné, doplníme tuto reprezentaci symboly 0.

Výstup: $x = x_k x_{k-1} \cdots x_l \in \{-1, 0, 1\}$

```
 1:  for all x_i do
 2:    if x_i = ±2 then
 3:      if x_{i-4} = ∓1 then
 4:        applyRule(100$\bar{1}$1) at position i − 3
 5:        if x_{i-3} = ±2 then
 6:          applyRule(100$\bar{1}$1) at position i − 6
 7:        end if
 8:      else
 9:        applyRule(100$\bar{1}$1) at position i − 4
10:      end if
11:    end if
12:  end for
13:  return
```

Poslední funkcí je funkce *bool processFile(const char * input , const char * output)*, která načítá textové řetězce ze souboru *input*, provádí přepis podle Algoritmů 4.1 a 4.2 a výsledné řetězce zapisuje do souboru *output*.

4.3.3 Funkce pro paralelní sčítání

Implementovaný program pro paralelní sčítání dvou reprezentací čísel využívá třídu *NumString* a další tři funkce.

První funkcí je funkce *int readPol(istream & input)*, která načte ze vstupního streamu *input* polynom a zkontroluje, zda-li se jedná o silný či slabý polynom, nastaví globální proměnné pro vnitří abecedu, vstupní abecedu a počet kroků. V případě silného polynomu funkce vrací *STRONG* (1), v případě slabého *WEAK* (-1) a v ostatních případech *NONE* (0).

Druhou funkcí je funkce *NumString * sum(NumString & a, NumString & b)*, která realizuje sčítání reprezentací a a b podle Algoritmů 4.1 a 4.2.

Poslední funkcí je, obdobně jako u přepisování řetězců, funkce *bool processFile(const char * input , const char * output)*, která načte ze souboru *input* silný či slabý polynom a dva číselné řetězce. Tyto číselné řetězce sečte a výsledek zapíše do souboru *output*.

4.4 Měření rychlosti

Z povahy přepisovacích Algoritmů 4.1 a 4.2 nemá příliš velký význam měřit rychlost jejich implementace. Uvedené algoritmy mají poměrně velkou asymptotickou složitost a minimální prostor pro optimalizaci.

Naproti tomu u programu pro paralelní sčítání bylo provedeno měření doby výpočtu. Měření probíhalo ve čtyřech testech, z nichž každý obsahoval 10 instancí daného problému. Každý test byl postupně spuštěn na 1, 2, 4 a 8 výpočetních vláknech. Prvním testem s názvem `big_long` bylo sčítání náhodně vygenerovaných reprezentací čísel dlouhých 3000 pozic s využitím náhodně vygenerovaného slabého polynomu délky 401. Jednotlivé koeficienty polynomu nabývaly hodnot -400 až 400. Koeficient b_0 vznikl sečtením 400 pozic daného polynomu a zvýšením o 1. Tím byl zajištěn maximální počet kroků s podle Algoritmu 2.4. Druhý test `big_short` probíhal nad stejně generovanými polynomy jako test `big_long`, pouze sčítané reprezentace byly dlouhé 500 pozic. Třetí test `small_long` zahrnoval sčítání s využitím slabého polynomu délky 201, jehož koeficienty (vyjma b_0) byly z rozsahu -100 až 100. Reprezentace čísel v tomto testu byly dlouhé 3000 pozic. Posledním testem s názvem `small_short` bylo sčítání reprezentací délky 500 s využitím slabého polynomu vygenerovaného stejně jako u testu `small_long`.

Všechny testy byly spuštěny na počítači se základní deskou Gigabyte GA-Z87X-UD5H osazenou procesorem Inte Core i7-4770 taktovaném na frekvenci 3,4GHz a 8GB operační paměti. Poznamenejme, že se jedná o čtyřjádrový procesor s funkcí hyper-threading. Každé jádro procesoru tedy obsahuje dvě řídící jednotky a proto je schopné zpracovávat dvě výpočetní vlákna současně. Jejich zpracování však není tak rychlé jako by bylo u dvou fyzických jader procesoru bez funkce hyper-threading. Na počítači byl nainstalován operační systém Windows 7 Professional Build 7601.

Cykly ve výsledném programu bylo možné paralelizovat se statickým nebo dynamickým plánováním vláken. Při dynamickém plánování každé nevyužité vlákno odešle dotaz runtimové knihovně OpenMP a ta vláknu vrátí číslo ite-

race, kterou má zpracovat. Naproti tomu při statickém plánování jsou každému vláknu přiděleny iterace ještě před vstupem do cyklu.

Výsledné časy jednotlivých testů se statickým plánováním vláken jsou uvedeny v Tabulce 4.1.

Tabulka 4.1: Doba běhu jednotlivých testů se statickým plánováním vláken

	1 vlákno	2 vlákna	4 vlákna	8 vláken
test `big_long`	383,63s	195,029s	106,294s	99,826s
test `big_short`	384,844s	196,15s	105,707s	100,793s
test `small_long`	47,991s	24,672s	13,484s	12,633s
test `small_short`	46,4s	23,824s	13,399s	12,315s

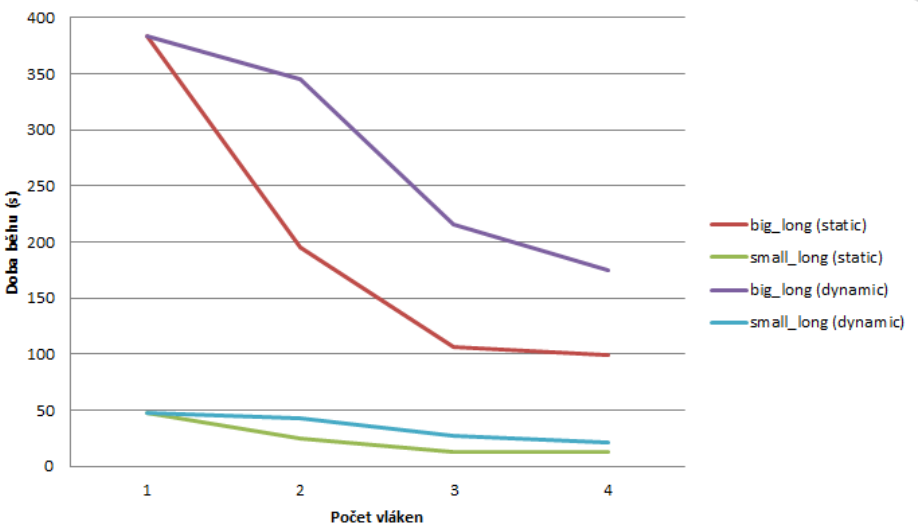
Doba výpočtu v jednotlivých testech s dynamickým plánováním je uvedena v Tabulce 4.2.

Tabulka 4.2: Doba běhu jednotlivých testů s dynamickým plánováním vláken

	1 vlákno	2 vlákna	4 vlákna	8 vláken
test `big_long`	383,63s	344,655s	215,15s	174,559s
test `big_short`	384,844s	346,201s	216,526s	175,912s
test `small_long`	47,991s	43,022s	26,913s	21,752s
test `small_short`	46,4s	42,967s	26,365s	21,427s

Uvedené výsledky naznačují, že dobu výpočtu ovlivňuje především zvolený polynom. Délka sčítaných reprezentací má na dobu běhu minimální vliv, proto jsou pro přehlednost v následujícím grafu zobrazeny pouze testy `big_long` a `small_long`. Závislost doby výpočtu na počtu vláken zobrazuje Obrázek 4.1.

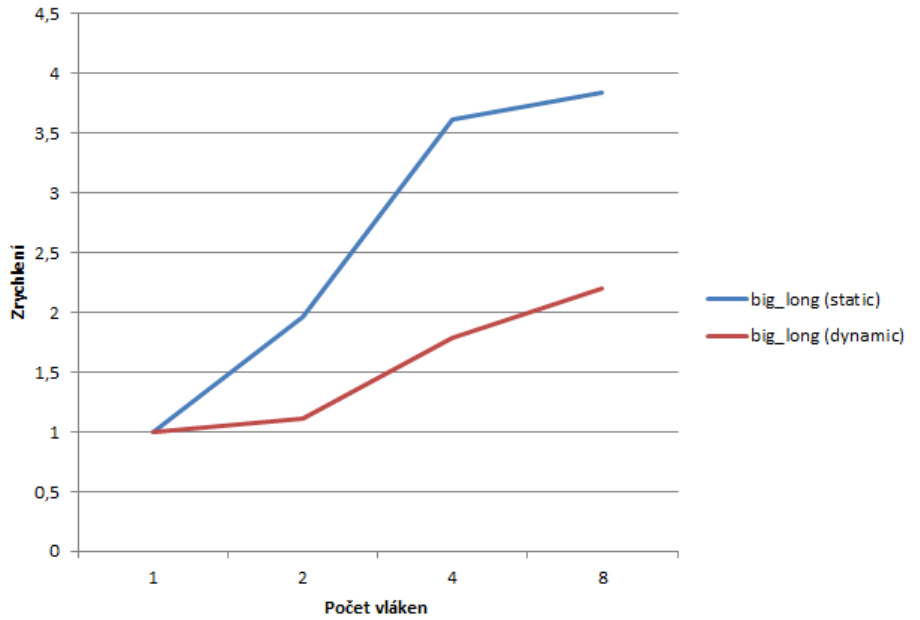
Obrázek 4.1: Graf závislosti doby výpočtu na počtu vláken



Statické plánování vláken bylo ve všech testech lepší než dynamické. To je pravděpodobně způsobeno velkou režií při dynamickém plánování.

Všechny testy při statickém plánování mají přibližně stejné zrychlení. Stejně je tomu i při dynamickém plánování. V grafu 4.2 porovnávajícím zrychlení při statickém a dynamickém plánování je tedy uveden pouze test `big_long`.

Obrázek 4.2: Graf zrychlení programu v závislosti na počtu vláken



Lze si povšimnout, že zrychlení s využitím 8 vláken je jen o málo větší než při využití 4 vláken. Tato skutečnost je způsobena funkcí hyper-threading a nikoli špatnou škálovatelností zvoleného algoritmu.

4.5 Možnosti rozšíření

Tato podkapitola reší možné rozšíření implementovaných programů. Jedná se o rozšíření, která jsou nad rámec této práce, avšak mohou být náplní případné další práce.

Z obou programů má větší potenciál na rozšíření pravděpodobně program pro přepisování řetězců. Tento program využívá konkrétní přepisovací pravidla a přepisuje reprezentace čísel z konkrétního tělesa. Program by bylo možné zobecnit pro všechna DUG tělesa tak, že by bylo možné měnit přepisovací pravidla (např. vstupem ze souboru nebo načtením ze standardního vstupu). To by však vyžadovalo funkcionální programování pro uchování daných pravidel v paměti programu. Vybízí se využití tzv. lambda kalkulu, metaprogramování, funktorů nebo stromových struktur. Pravděpodobně nejjednodušší bude načítání pravidel z textové podoby pomocí lexikálního a syntaktického analyzátoru a uložení pravidla do stromu. Lexikální analyzátor je možné naprogramovat pomocí jednoduchého automatu, který si bude pamatovat všechny načtené lexikální symboly. Ty pak zpracuje LL(1) syntaktický analyzátor, který je možné realizovat pomocí rekurzivního sestupu. Při zadávání pravidel by měl být program schopen rozlišit a uložit přepisovací pravidlo a podmínku, za které se pravidlo využije. Dále je nutné uložit u každého pravidla pozici, na které se aplikuje a pozici symbolu, který se má v absolutní hodnotě snížit použitím daného pravidla.

Program pro paralelní sčítání příliš možností k rozšíření nenabízí. Za zvážení by stály kompilátorové optimalizace a případná vektorizace. V případě nasazení programu na výpočetních svazcích by bylo nutné upravit program tak, aby výpočty probíhaly nad sdílenou pamětí, např. s využitím OpenMPI.

Závěr

Cílem této práce bylo seznámit se s pojmy potřebnými k vypracování rešerše na téma využití přepisovacích pravidel a tuto rešerši vypracovat. Dalším cílem bylo zkoumané přepisovací algoritmy implementovat.

Klíčovou částí bylo provedení poměrně obsáhlé rešerše napříč různými matematickými disciplínami od kombinatoriky na slovech, přes aritmetiku na pozičních reprezentacích, po teorii čísel. Vybrané problémy byly dány do souvislosti s pojmem přepisovací pravidlo.

Výstupem práce jsou dva programy implementující algoritmy z rešeršní části. Program pro přepisování číselných řetězců vychází z Kapitoly 3 a využívá třídu navrženou pro reprezentaci číselných řetězců. Základem programu realizujícího paralelní sčítání reprezentací čísel byly Algoritmy 2.3 a 2.4, které byly v rámci implementace optimalizovány. Program pro paralelní sčítání využívá k paralelizaci OpenMP a dosáhl během testování paralelního zrychlení poměrně zajímavých výsledků (např. při využití 4 výpočetních vláken byl výpočet 3,6krát rychlejší oproti sekvenční verzi).

Implementovaný program pro přepisování číselných řetězců byl navržen tak, aby jej v budoucnu bylo snadné rozšířit. Již během vývoje došlo k návrhu rozšíření programu pro všechna algebraická číselná tělesa. Jedná se o úpravu, která je však nad rámec této práce a její realizace může být cílem například diplomové práce.

Literatura

[1] Avizienis, A.: Signed-Digit Number Representations for Fast Parallel Arithmetic. *Electronic Computers, IRE Transactions on*, 1961: s. 389–400.

[2] Belcher, P.: Integers Expressible as Sums of Distinct Units. *Bulletin of the London Mathematical Society*, 1974: s. 66–68.

[3] Belcher, P.: A test for integers being sums of distinct units applied to cubic fields. *Journal of the London Mathematical Society*, 1976: s. 141–148.

[4] Chow, C.-Y.; Robertson, J. E.: Logical design of a redundant binary adder. *Proc. 4th IEEE Symposium on Computer Arithmetic*, 1978: s. 109–115.

[5] Dombek, D.: *Non-standard representations of numbers*. Disertační práce, České vysoké učení technické v Praze, 2014.

[6] Dombek, D.; Masáková, Z.; Ziegler, V.: On distinct unit generated fields that are totally complex. *ArXiv e-prints*, 2014, 1403.0775. Dostupné z: http://arxiv.org/abs/1403.0775

[7] Frougny, C.; Pelantová, E.; Svobodová, M.: Parallel addition in non-standard numeration systems. *ArXiv e-prints*, 2011, 1102.5683. Dostupné z: http://arxiv.org/abs/1102.5683

[8] Hajdu, L.; Ziegler, V.: On linear combinations of units with bounded coefficients. *Mathematics of Computation*, 2014: s. 1495–1512.

[9] Jacobson, B.: Sums of Distinct Divisors and Sums of Distinct Units. *Proceedings of the American Mathematical Society*, 1964: s. 179–183.

[10] Lothaire, M.: *Combinatorics on words*, díl 17. Addison-Wesley Publishing Co., 1983.

[11] Lothaire, M.: *Algebraic combinatorics on words*, díl 90. Addison-Wesley Publishing Co., 2002.

[12] Masáková, Z.; Pelantová, E.: *Teorie čísel*. Česká technika - nakladatelství ČVUT, 2010.

[13] Melichar, B.: *Jazyky a překlady*. České vysoké učení technické v Praze, 2003.

[14] Parry, W.: On the β-expansions of real numbers. *Acta Mathematica Academiae Scientiarum Hungarica*, 1960: s. 401–416. Dostupné z: `http://dx.doi.org/10.1007/BF02020954`

[15] Rényi, A.: Representations for real numbers and their ergodic properties. *Acta Mathematica Academiae Scientiarum Hungarica*, díl 8, č. 3-4, 1957: s. 477–493.

[16] Śliwa, J.: Sums of distinct units. *Bulletin de L'Academie Polonaise des Sciences*, 1974: s. 11–13.

[17] Thurston, W. P.: Groups, tilings and finite state automata. 1989.

[18] Thuswaldner, J.; Ziegler, V.: On linear combinations of units with bounded coefficients. *Mathematika*, 2011: s. 247–262.

Příloha **A**

Obsah přiloženého CD

- `readme.txt`
- `exe`
  - `parallel adder` .... adresář se spustitelným programem pro paralelní sčítání
  - `string rewriter` .adresář se spustitelným programem pro přepisování řetězců
- `src`
  - `parallel adder` ..... zdrojové kódy programu pro paralelní sčítání
  - `string rewriter` ....zdrojové kódy programu pro přepisování řetězců
- `text`
  - `src` ..... adresář s textem této práce ve formátu LaTeX
  - `thesis.pdf` ..... text práce ve formátu PDF