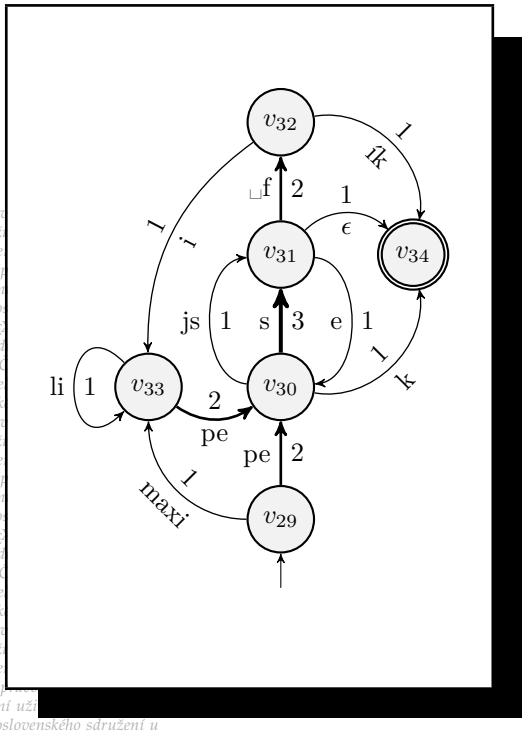
CSTUG



ZPRAVODAJ

Československého sdružení uživatelů TEXu

1–2

2023

ISSN 1211-6661 Tištěná verze

ISSN 1213-8185 Online verze

Ročník 33

OBSAH

Petr Sojka: Úvodník 1

Vít Starý Novotný: Nápadovník jmen pro tvůrčí psaní v LuaTEXu 3

Karel Šebela: Sazba hudebních skladeb 39

Matúš Vančík: Pohľad TEXového nováčika na prezentáciu „Bricks and Jigsaw Pieces“ z TUGu 2022 48

Peter Wilson: Mělo by to fungovat XIII 54

Zpravodaj Československého sdružení uživatelů TEXu je vydáván v tištěné podobě a distribuován zdarma členům sdružení. Vydaná čísla Zpravodaje v elektronické podobě (PDF) jsou bezodkladně veřejně vystavena na webové adrese `https://www.cstug.cz/`.

Své příspěvky do Zpravodaje můžete zasílat v elektronické podobě, nejlépe jako jeden archivní soubor (`.zip, .arj, .tar.gz`), na e-mailovou adresu `bulletin@cstug.cz`. Nezapomeňte přiložit všechny soubory, které dokument načítá (s výjimkou standardních součástí TEX Live).

ISSN 1211-6661 (tištěná verze)
ISSN 1213-8185 (online verze)

Úvodník

Petr Sojka

Milé čtenářky a čtenáři, TEXistky a TEXisté!

Je mi potěšením vám představit články tohoto čísla a seznámit vás s informacemi ze světa našeho oblíbeného sázecího systému.

Stavba babylonské věže ve Světové kronice Rudolfa von Ems, Praha, 3. čtvrtina 14. století.

Autor: Anonymní (Meister 1) – Hochschul- und Landesbibliothek Fulda

Babylónské zmatení jazyků. Tak nám připadá komunikace v dnešním informačním věku. Dávat věcem ta správná jména, odpovídající sémantice, a konzistentně je používat, to je, oč tu běží!

A běží o to v článku Vítka Starého Novotného, jehož nápad na nápadovník a jeho realizace je tématem hlavního, prvního článku čísla. Láká Dášenka, Maxipes Fík a Pes Filipes, častá jména z článku a z titulní stránky tohoto čísla. Pokud se nenecháte odradit asi deseti programovacími a značkovacími jazyky, které v článku Vítek používá, dozvíte se základní principy jazykových modelů a také způsob, jakým je využijete přímo při tvůrčím psaní. Pojďte zjistit, k čemu jsou dobré náhodné procházky, programovací jazyk expl3, Katzův couvající jazykový model, zapomnětlivost, náhodná semínka či multiplikativní lineární kongruenční generátor. Dostanete „na konečky svých prstů“ a do svého dokumentu informace

doi: 10.5300/2023-1-2/1

hlubokých jazykových modelů, což dá vašim textům patřičnou hloubku nebo výšku srovnatelnou s babylónskou věží nebo odbřemení od explicitního přepínání jazyků tím, že „umělá inteligence“ dobře odhadne použitý jazyk textu. Jestli chcete jít s dobou, tak do toho!

Sazba hudebních skladeb je vzhledem ke složitosti notového zápisu výzvou. Možnostem sazby not nejen MusiXTEXem, ale využití specializovaných preprocesorů, které přípravu notového zápisu usnadní, se věnuje přehledový článeček Karla Šebely.

Matúš Vančík se dělí o to, co ho zaujalo v prezentacích hlavní TEXovské konference TUG 2022. Způsob, jak vznikají makrobalíčky, jak je kombinovat nebo jak realizovat puzzle s logem Fakulty informatiky Masarykovy univerzity, jsou jen některé z nových informací, kterými vás článek po přečtení obohatí.

Číslo uzavírá již *třinácté* pokračování komentovaných ukázek TEXových maker z pera Petera Wilsona, které pro čtenáře Zpravodaje již třináct let[1] překládá ze série Glisterings z časopisu TUGboat náš pan šéfredaktor Honza Šustek.

Závěrem chci poděkovat aktivním TEXistům, autorům, členům sdružení uživatelů TEXu. Bez nich, převážně členů CSTUGu a TUGu a jejich „labour of love“ by to nebylo možné.

Odkazy

1. VRABCOVÁ, Tereza. Digitální archivace Zpravodaje CSTUGu. *Zpravodaj CSTUG*. 2022, roč. 32, č. 1, 11–17. Dostupné z DOI: 10.5300/2022-1-4/11.
2. NOVOTNÝ, Vít. *Konverze Zpravodaje CSTUGu pro Českou digitální matematickou knihovnu (DML-CZ) Co, proč a jak?* 2021-03. Dostupné také z: https://www.cstug.cz/informace/zpravy/2021-02-06-valne-shromazdeni-2020/files/novotny-konverze-2021-03-06.pdf.
3. *Zpravodaj CSTUG*. Československé sdružení uživatelů TEXu, 1991–2023. Dostupné také z: https://www.dml.cz/handle/10338.dmlcz/148724.

Summary: Introductory Word

Go forth and participate in CSTUG to make the bright future of TEX & Friends a reality! *You can!*

Masarykova univerzita, Fakulta informatiky, Botanická 68a, 602 00 Brno
sojka@fi.muni.cz

[1] K celému seriálu překladů Glisterings se můžete vrátit na stránkách DML.CZ, kde péčí aktivních členů sdružení [1, 2] vznikl archiv celého Zpravodaje [3].

Nápadovník jmen pro tvůrčí psaní v LuaTEXu

Vít Starý Novotný

Známý výrok informatika Phila Karltona říká, že na informatice jsou obtížné pouze dvě věci: vyprazdňování cache a přidělování jmen. Své o tom vědí i spisovatelé, kteří musí kromě příběhu a světa vymyslet jména všech svých příběhových postav. V tomto článku vyvineme jazykový model, který spisovatelům umožní automaticky generovat jména postav při tvůrčím psaní v LuaTEXu. Kromě pomoci při tvůrčím psaní si představíme i další možná použití jazykových modelů v LuaTEXu, jako je automatické přepínání vzorů dělení slov podle aktuálního jazyka a generování výplňového textu. TEXnicky zaměřeným čtenářům článek poslouží jako prvotní seznámení s programovacími jazyky Lua a expl3 a s LATEXovým balíčkem xparse pro přípravu uživatelských rozhraní.

Klíčová slova: tvůrčí psaní, trie, jazykové modely, LuaTEX, Lua, expl3, xparse

Úvod

Se vzestupem aplikací postavených na velkých jazykových modelech od firmy OpenAI stoupá zájem laické veřejnosti o jazykové modelování. Nástroj ChatGPT na požádání napíše báseň, esej i dopis, připraví kód v programovacím jazyce nebo zkontroluje gramatické a ortografické chyby v článku. Vyhledávač Bing provede automatické srovnání cen produktů na základě výsledků vyhledávání, zodpoví faktické dotazy včetně citace zdrojů a naplánuje dovolenou. Velké jazykové modely lze ale snadno zneužít pro hromadné generování dezinformací a generování kódu pro napadení informačních systémů, což přitahuje pozornost regulátorů. Stejně tak jsou velké jazykové modely výpočetně náročné a dostupné pouze přes placené webové rozhraní, což omezuje jejich využitelnost.

V tomto článku se svezeme na vlně zájmu o jazykové modely a vyvineme v jazyce Lua malý jazykový model, který můžeme snadno využít v dokumentech sázených pomocí LuaTEXu. Náš jazykový model bude sloužit spisovatelům pro generování kreativních jmen postav při tvůrčím psaní, ale uvedeme i další potenciální využití jazykových modelů v TEXu, jako je automatické přepínání vzorů dělení slov podle aktuálního jazyka a generování výplňového textu pro přípravu maket.

Nejprve v sekci 1 představíme náš jazykový model pomocí obrázků, příkladů a matematických definic. V sekci 2 na straně 8 popsaný model implementujeme v programovacím jazyce Lua a v sekci 3 na straně 16 mu v programovacím jazyce expl3 a pomocí LATEXového balíčku xparse připravíme uživatelské rozhraní pro snadné využití v dokumentech. TEXnicky zaměřeným čtenářům tato sekce poslouží

doi: 10.5300/2023-1-2/3

jako prvotní seznámení s programovacími jazyky Lua a expl3 a s LaTeXovým balíčkem xparse. Následně v sekci 4 na straně 26 natrénujeme vyvinutý model na několika příkladových databázích jmen a ukážeme, jak bychom vygenerovali jména postav a zahrnuli je do textu povídky. Čtenáři, kteří se nezajímají o detaily jazykového modelu, mohou začít touto sekcí. Nakonec si v sekci 5 na straně 31 uvedeme možná vylepšení jazykového modelu a další aplikace jazykových modelů v TeXu, jako je automatické přepínání vzorů dělení slov podle aktuálního jazyka a generování výplňového textu. V sekci 6 na straně 36 článek uzavřeme shrnutím výsledků článku a jejich praktického přínosu pro čtenáře.

1. Jazykový model pro generování jmen postav

1.1. Trie (prefixové stromy)

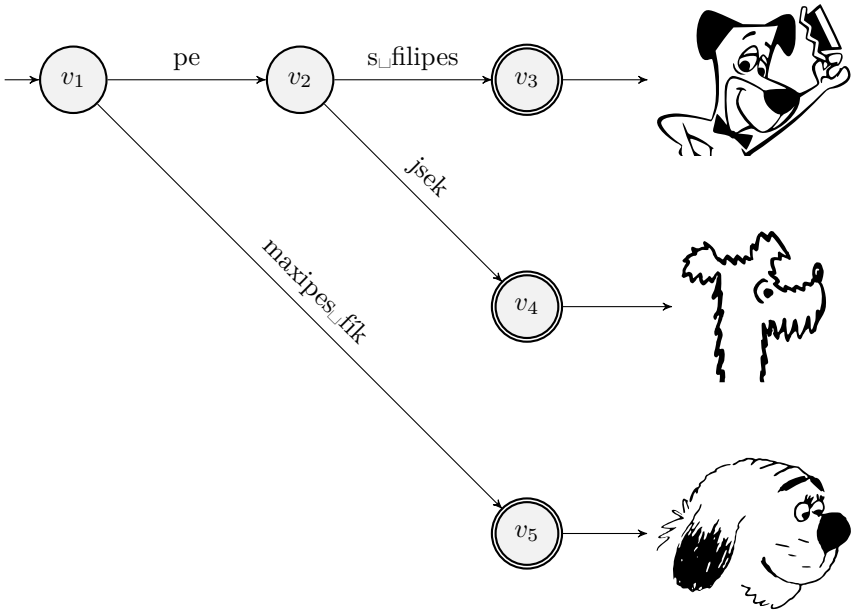
Trie [1] je stromová datová struktura, pomocí které můžeme implementovat asociativní pole s textovými klíči. Na rozdíl od hašové tabulky nám trie umožňuje mj. efektivně vyhledávat všechny klíče buď s danou předponou, nebo příponou,[1] vizte např. Obrázek 1 s triemi, které asociují texty „pes filipes“, „pejsek“ a „maxipes fík“ s obrázky pohádkových psů. Pokud chceme získat jména a obrázky všech pohádkových psů s předponou pe-, sestoupíme nejprve do vrcholu v_2 a následně projdeme celý podstrom. Takto získáme jména „pes filipes“ a „pejsek“ spolu s odpovídajícími obrázky. Pro jména a obrázky s příponou -k sestoupíme nejprve do vrcholu v_9 a opět projdeme celý podstrom.

1.2. Základní jazykový model

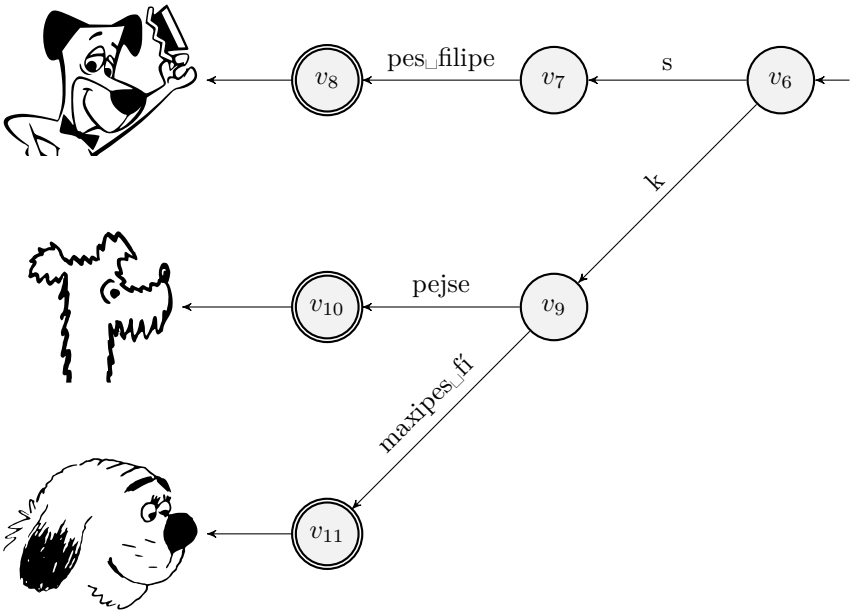
Kromě asociativních polí můžeme trie využít pro implementaci textových množin s efektivním hledáním buď podle předpon, nebo přípon. Příkladem jsou klávesnice v mobilních telefonech, které uživateli našeptávají slova ze slovníku podle rozepsaného slova. Pokud si k hranám trie navíc poznačíme váhu c, která nám udává, kolikrát jsme hrany během sestavování trie navštívili, poslouží nám trie i jako jazykový model. Na Obrázku 2 vidíme trii, pomocí které můžeme spočítat pravděpodobnost výskytu slova S jako součin podmíněných pravděpodobností výskytů jednotlivých znaků slova a konce slova:

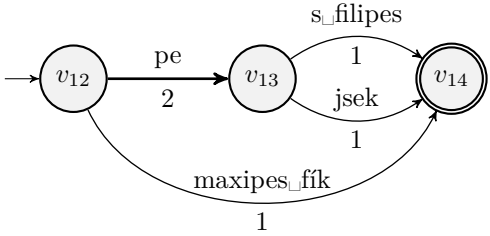
$$P(S) = \Big(\prod_{i=1,\ldots,|S|} P(S[i] \mid S[1,\ldots,i-1]) \Big) \cdot P(\epsilon \mid S), \text{ kde } \epsilon \text{ je prázdné slovo.}$$

[1] Předponami a příponami zde máme na mysli libovolné posloupnosti znaků na začátku a na konci textového klíče. Nejedná se tedy o předpony a přípony ve smyslu nauky o stavbě slov. Kromě vyhledávání textových klíčů buď podle předpon, nebo podle přípon lze v triích vyhledávat i podle kombinace předpony a přípony nebo podle libovolného podřetězce, a to pomocí tzv. *permutermínů*, vizte sekci 5.2.1 na straně 32.



Obrázek 1: Trie, které mapují jména pohádkových psů na jejich obrázky [2, 3, 4]. Trie nahoře umožňuje hledání podle předpon a trie dole hledání podle přípon.





Slovo S	$P(S)$
pes filipes	$33,\overline{3}$ %
pejsek	$33,\overline{3}$ %
maxipes fík	$33,\overline{3}$ %

Obrázek 2: Jazykový model, který nám umožnuje generovat jména pohádkových psů náhodnými procházkami v trii. Tabulka vpravo uvádí všechna jména, která můžeme vygenerovat, a jejich pravděpodobnost.

Pro výpočet podmíněné pravděpodobnosti znaku sestoupíme nejprve z počátečního vrcholu po znacích $S[1,\ldots,i-1]$ do vrcholu v. Následně zvolíme hranu h z vrcholu v se znakem $S[i]$. Pravděpodobnost spočítáme jako podíl váhy c hrany h vůči součtu vah všech hran H vycházejících z vrcholu v:

$$P(S[i] \mid S[1,\ldots,i-1]) = \frac{c(e)}{\sum_{e' \in E} c(e')}.$$

Např. pravděpodobnost slova „pejsek“ spočítáme následovně:

$$P(\text{pejsek}) = P(\text{pe}|\epsilon) \cdot P(\text{jsek}|\text{pe}) \cdot P(\epsilon|\text{pejsek}) = \frac{2}{3} \cdot \frac{1}{2} \cdot 1 = \frac{1}{3} = 33,\overline{3}\ \%.$$

Slova můžeme také vybírat náhodnými procházkami v trii. Takto vypadá náhodná procházka po vrcholech v_{12}, v_{13} a v_{14}, která vybere jméno „pes filipes“:

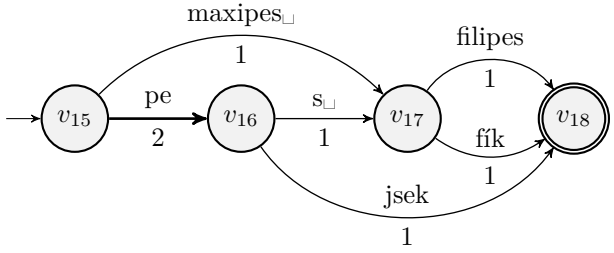
1. Začneme v počátečním vrcholu v_{12}.
2. S $66,\overline{6}\%$ pravděpodobností přejdeme po hraně „pe“ do vrcholu v_{13}.
3. S 50% pravděpodobností přejdeme po hraně „s␣filipes“ do vrcholu v_{14}.
4. Ukončíme náhodnou procházku, protože z vrcholu v_{14} nevede žádná hrana.

1.3. Zapomnětlivý jazykový model

Náš jazykový model sice dokáže vygenerovat slova, pomocí kterých jsme ho sestavili, ale nedovede vymyslet nová slova. Abychom zvýšili kreativitu modelu, upravíme jeho pravděpodobnostní funkci tak, aby místo všech předchozích znaků brala v potaz pouze *kontext* posledních n znaků a na další znaky „zapomněla“:

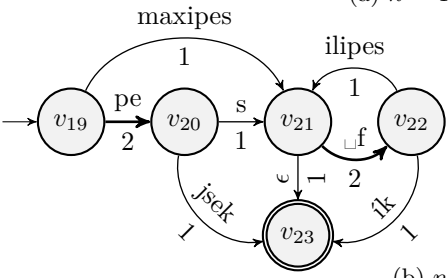
$$P'(S) = \prod_{i=1,\ldots,|S|} P\big(S[i] \mid S[i-n,\ldots,i-1]\big) \cdot P\big(\epsilon \mid S\big[|S|-n,\ldots,|S|\big]\big).$$

Pro délky kontextu $n \geq 6$ obdržíme stejný jazykový model jako na Obrázku 2. Jazykové modely pro délky kontextu $0 < n \leq 5$ najdeme na Obrázku 3. Při $n = 4$



Slovo S	$P(S)$
pejsek	$33{,}\overline{3}$ %
pes fík	$16{,}\overline{6}$ %
pes filipes	$16{,}\overline{6}$ %
maxipes fík	$16{,}\overline{6}$ %
maxipes filipes	$16{,}\overline{6}$ %

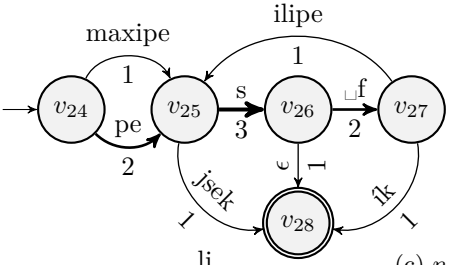
(a) $n = 4$ a $n = 5$



Slovo S	$P(S)$
pejsek	$33{,}\overline{3}$ %
$(\text{maxi})^i\text{pes}(␣\text{filipes})^j(␣\text{fík})^k$	$0{,}\overline{3}^j \cdot 11{,}\overline{1}$ %

$i \in \{0, 1\}, j \geq 0, k \in \{0, 1\}$

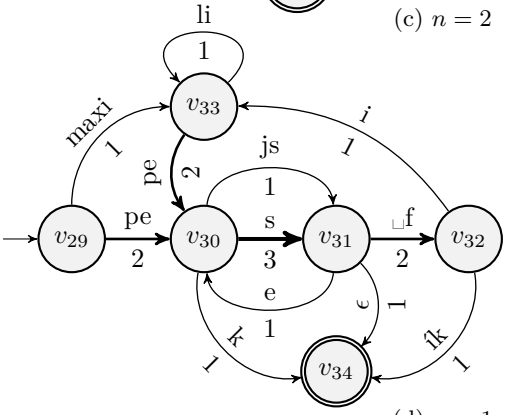
(b) $n = 3$



Slovo S	$P(S)$
$(\text{maxi})^i\text{pe}(\text{s}␣\text{filipe})^j\text{jsek}$	$0{,}5^i \cdot 0{,}25^j \cdot 16{,}\overline{6}$ %
$(\text{maxi})^i\text{pes}(␣\text{filipes})^j(␣\text{fík})^k$	$0{,}25^j \cdot 8{,}\overline{3}$ %

$i \in \{0, 1\}, j \geq 0, k \in \{0, 1\}$

(c) $n = 2$



Slovo S	$P(S)$
$\Big(\text{maxi}(\text{li})^i\Big)^j\text{p}\Big((\text{ej}^k\text{s})^l␣\text{fi}(\text{li})^u\text{p}\Big)^v(\text{ej}^w\text{s})^x(␣\text{fík})^y$	$0{,}\overline{3}^{ij+klv+uv+wx} \cdot 0{,}6^{lv+x} \cdot 0{,}1\overline{6}^v \cdot 16{,}\overline{6}$ %
$\Big(\text{maxi}(\text{li})^i\Big)^j\text{p}\Big((\text{ej}^k\text{s})^l␣\text{fi}(\text{li})^u\text{p}\Big)^v(\text{ej}^w\text{s})^x\text{ek}$	$0{,}\overline{3}^{ij+klv+uv+wx} \cdot 0{,}6^{lv+x} \cdot 0{,}1\overline{6}^v \cdot 3{,}\overline{3}$ %

$i \geq 0, j \in \{0, 1\}, k \in \{0, 1\}, l \geq 0, u \geq 0,$
$v \geq 0, w \in \{0, 1\}, x \geq 1, y \in \{0, 1\}$

(d) $n = 1$

Obrázek 3: Zapomnětlivé jazykové modely pro různé velikosti paměti n, které nám umožňují generovat nová slova a slovní spojení podobná jménům pohádkových psů. Tabulka vpravo uvádí všechny možné výstupy modelu a jejich pravděpodobnost.

a $n = 5$ dokáže jazykový model vymyslet nová slovní spojení „pes fík“ a „maxipes filipes“. Při $n = 3$ dostáváme nové slovo „pes“ a v grafu nám vzniká cyklus mezi vrcholy v_{21} a v_{22}, který s nízkou pravděpodobností generuje slovní spojení libovolné délky jako „maxipes filipes filipes fík“ s pravděpodobností přibližně 1,23 %. Při $n = 2$ dokážeme vygenerovat jména jako „maxipes filipejsek“, která obsahují nová odvozená slova. Při $n = 1$ vznikají v modelu krátké cykly nad vrcholem v_{33} a mezi vrcholy v_{30} a v_{31}, díky kterým dokážeme vygenerovat nová jména s opakovanými slabikami jako „maxilipes filipes fík“ a „peses filipejsejsek“.

1.4. Katzův couvající jazykový model

Při generování nových jmen můžeme náhodné procházky uvozovat buď požadovanou předponou, nebo příponou. Při dlouhých kontextech ale bude mít většina předpon a přípon nulovou pravděpodobnost, zatímco při krátkých kontextech budou generovaná jména vypadat nahodile. Pokud chceme např. vygenerovat jméno pohádkového psa s předponou i-, selžeme pro všechny hodnoty $n \geq 2$ a při $n = 1$ získáme nepravděpodobná jména jako „ipek“ a „ililipejejs“.

Možným řešením popsaného problému je Katzův jazykový model [5], který během náhodné procházky „couvá“ s délkou kontextu n na nejvyšší takovou hodnotu, při které má aktuální předpona nebo přípona nenulovou pravděpodobnost. Náhodná procházka s délkou kontextu $n = 4$ a předponou i- v Katzově modelu neselže a vygeneruje pouze střízlivá jména jako „ilipes“, „ipes fík“ a „ipes filipes“.

2. Implementace jazykového modelu

V této sekci rozpracujeme soubor `randomnames.lua`, který bude obsahovat implementaci našeho jazykového modelu v jazyce Lua. Nejprve se v sekci 2.1 zaměříme na sestavení modelu. Následně v sekci 2.2 na straně 12 uděláme malou odbočku a naprogramujeme vlastní generátor pseudonáhodných čísel, abychom mohli nakonec v sekci 2.3 na straně 14 implementovat (pseudo)náhodné procházky v našem modelu. Hotový soubor `randomnames.lua` můžeme stáhnout online [6].

Příklady z této sekce uvozené dosavadním obsahem souboru `randomnames.lua` si můžeme vždy uložit do textového souboru `priklad.lua` a přeložit příkazem `texlua priklad.lua`.

2.1. Sestavení jazykového modelu

Pro sestavení zapomnětlivého jazykového modelu potřebujeme znát pouze délku kontextu n a množinu trénovacích slov.

```
1 local ForgetfulModel = {}
2
```

```
3  function ForgetfulModel.new(context_size)
4    local model = {}                    -- Vytvoř prázdný objekt.
5    model.context_size = context_size   -- Ulož si délku kontextu
6    model.forward_graph = {}            -- a prázdný dopředný
7    model.reverse_graph = {}            -- a zpětný jazykový model.
8    local mt = {                        -- Zděď metody[2]
9      __index = ForgetfulModel }        -- třídy ForgetfulModel.
10   setmetatable(model, mt)
11   return model
12 end
13
14 local function trim_context(context,           -- Oříznі kontext
15                              max_context_size)  -- na danou délku.[3]
16   local context_size = utf8.len(context)
17   if context_size > max_context_size then
18     local character_offset = context_size - max_context_size + 1
19     local byte_offset = utf8.offset(context, character_offset)
20     context = context:sub(byte_offset, #context)
21   end
22   return context
23 end
24
25 local function add_name(graph, name, max_context_size)
26   name = name                           -- Konec jména kóduj
27        .. "\n"                          -- jako konec řádku.
28   local context = ""
29   for _, code in utf8.codes(name) do    -- Procházej znaky jména.
30     if graph[context] == nil then       -- Pokud je třeba,
31       graph[context] = {}               -- přidej nový vrchol
32     end                                 -- pro současný kontext.
33     local vertex = graph[context]
34     local character = utf8.char(code)
35     if vertex[character] == nil then    -- Pokud je třeba,
36       vertex[character] = 0             -- přidej novou hranu
37       table.insert(vertex, character)   -- pro současný znak.
```

[2]Jazyk Lua používá tzv. *prototypální dědičnost* pomocí *metatabulek* [7, sekce 20.4], která je obecnější než třídní dědičnost, kterou známe z objektově orientovaných jazyků, jako jsou C++ a Java. Zde jsme vytvořili prázdnou hašovou tabulku `model` s metatabulkou `{ __index = ForgetfulModel }`, která říká, že při přístupu k `model`u se mají neznámé klíče hledat v hašové tabulce `ForgetfulModel`. To nám umožňuje volat metody jako `model.add_name(model, name)` nebo krátce `model:add_name(name)`.

[3]Unární operátor mřížky (`#`) v jazyce Lua vrací délku pole nebo textového řetězce.

```
38      end
39      vertex[character] =                    -- Navyš hodnotu hrany
40        vertex[character] + 1                -- pro současný znak.
41      context = trim_context(                -- Aktualizuj kontext
42        context .. character,                -- a seřízni ho na
43        max_context_size)                    -- požadovanou délku.
44    end
45  end
46
47  local function reverse(name)               -- Ulož znaky jména
48    local index = utf8.len(name)             -- v obráceném pořadí.
49    local result = {}
50    for _, code in utf8.codes(name) do
51      local character = utf8.char(code)
52      result[index] = character
53      index = index - 1
54    end
55    local reversed_name = table.concat(result)
56    return reversed_name
57  end
58
59  function ForgetfulModel.add_name(model, name)
60    add_name(model.forward_graph,            -- Zaznamenej jméno
61             name,                           -- zepředu i pozpátku
62             model.context_size)             -- pro náhodné procházky
63    add_name(model.reverse_graph,            -- uvozené buď
64             reverse(name),                  -- požadovanou předponou,
65             model.context_size)             -- nebo příponou.
66  end
67
68  function ForgetfulModel.input_names(model, filename)
69    local file = io.open(filename, "r")      -- Procházej řádky
70    assert(file)                             -- zadaného textového
71    for name in file:lines() do              -- souboru se jmény.
72      if #name:match("^%s*(.-)%s*$") == 0    -- Přeskoč prázdné řádky.
73      then
74        goto continue
75      end
76      model:add_name(name)                   -- Zanes jméno do modelu.
77      ::continue::
78    end
79  end
```

Katzův couvající jazykový model sestavíme ze zapomnětlivých modelů pro délky kontextu $n = i, i+1, \ldots, j-1, j$, kde hraniční hodnoty i, j zadává uživatel.

```
80  local BackoffModel = {}
81
82  function BackoffModel.new(min_context_size, max_context_size)
83    local model = {}                  -- Vytvoř prázdný objekt.
84    model.min_context_size =          -- Ulož si hraniční hodnoty
85      min_context_size                -- délky kontextu.
86    model.max_context_size =
87      max_context_size
88    model.submodels = {}              -- Vytvoř zapomnětlivé modely.
89    for context_size = min_context_size, max_context_size do
90      model.submodels[context_size]
91        = ForgetfulModel.new(context_size)
92    end
93    local mt = {                      -- Zděď metody
94      __index = BackoffModel }        -- třídy BackoffModel.
95    setmetatable(model, mt)
96    return model
97  end
98
99  function BackoffModel.for_each_submodel(model, func,
100                                          min_context_size,
101                                          max_context_size)
102   assert(min_context_size >= model.min_context_size)
103   assert(max_context_size <= model.max_context_size)
104   local i, j = min_context_size, max_context_size
105   for context_size = j, i, -1 do    -- V pořadí klesající
106     local submodel                  -- délky kontextu uplatni
107       = model.submodels[            -- funkci nad všemi
108         context_size]               -- zapomnětlivými modely.
109     local result = func(submodel)   -- Pokud funkce vrátí hodnotu
110     if result == false then         -- false, přeruš zpracování
111       break                         -- předčasně.
112     end
113   end
114 end
115
116 function BackoffModel.add_name(model, name)
117   model:for_each_submodel(
118     function(submodel)
```

```
      submodel:add_name(name)        -- Zanes jméno do všech
    end,                             -- zapomnětlivých modelů.
    model.min_context_size,
    model.max_context_size)
end

function BackoffModel.input_names(model, filename)
  model:for_each_submodel(
    function(submodel)
      submodel:input_names(          -- Zanes jména do všech
        filename)                    -- zapomnětlivých modelů.
    end,
    model.min_context_size,
    model.max_context_size)
end
```

Katzův couvající jazykový model z Obrázku 3 pro délky kontextu $n = 1, 2, \ldots, 5$ sestavíme v LuaTEXu následně:

```
model = BackoffModel.new(1, 6)         -- Vytvoř prázdný model.
model:add_name("pes filipes")          -- Zanes do modelu jména
model:add_name("pejsek")               -- pohádkových psů.
model:add_name("maxipes fík")
```

2.2. Generátor pseudonáhodných čísel

Pro náhodné procházky v jazykovém modelu potřebujeme hrací kostku, která nám řekne, po které hraně grafu se máme vydat. Jazyk Lua poskytuje vestavěný generátor pseudonáhodných čísel, avšak ten má několik slabin: Pro generování čísel slouží funkce `math.random()`, která generuje různá čísla na různých platformách. My ale chceme, aby náš model generoval stejná náhodná jména na všech aktuálních instalacích TEXu. Pro resetování vestavěného generátoru pomocí náhodného semínka slouží funkce `math.randomseed()`. Naším záměrem však je současně používat několik jazykových modelů se vzájemně nezávislými generátory. Uvedené problémy vyřešíme implementací vlastního generátoru pseudonáhodných čísel.

Jazyk Lua 5.3 používá alespoň 32bitová celá čísla se znaménkem a neurčitou endianitou [8, sekce 2.1]. Pro nezávislost na platformě tedy musíme zvolit algoritmus, který nepoužívá bitové operace ani jinak nezávisí na endianitě a který pracuje pouze s čísly v rozmezí od 0 do $2^{31} - 1$. Takovým algoritmem je například multiplikativní lineární kongruenční generátor [9, sekce 7.1]:

$$x_n = a \cdot x_{n-1} \pmod{m},$$

kde $x_1 \neq 0 \pmod{m}$ je náhodné semínko, x_n je n-té pseudonáhodné číslo a a, m

jsou tabulkové hodnoty. Pro dosažení nezávislosti na platformě požadujeme, aby mezivýsledky nepřesáhly hodnotu $2^{31} - 1$. Toho dosáhneme vhodným výběrem hodnot $a = 771\,645\,345$, $m = 2^{30} - 35 = 1\,073\,741\,789$ [10, tabulka 2], omezením semínka na hodnoty od 1 do $m - 1$ a použitím algoritmu pro modulární násobení.

```
local Random = { A = 771645345, M = 1073741789 }

function Random.new(seed)
  assert(seed >= 0 and              -- Povol semínko z rozsahu
         seed <= 2147483647)        -- od 0 do 2^31 - 1.
  seed = seed                       -- Převeď semínko do rozsahu
       % (Random.M - 1) + 1         -- od 1 do m - 1.
  local random = { x = seed }       -- Vytvoř objekt se semínkem.
  local mt = { __index = Random }   -- Zděď metody třídy Random.
  setmetatable(random, mt)
  return random
end

local function multiply_modulo(a, x, m)
  local result = 0                  -- Algoritmus pro modulární
  a = a % m                         -- násobení zajistí, že
  while x > 0 do                    -- mezivýsledky nepřesáhnou
    if x % 2 == 1 then              -- hodnotu 2 * m.
      result = (result + a) % m
    end
    a = (a * 2) % m
    x = x // 2
  end
  return result
end

function Random.get_next_number(random, from, to)
  random.x = multiply_modulo(Random.A, random.x, Random.M)
  local result = from               -- Navrať celé číslo
               + (random.x          -- v zadaném rozsahu.
                 % (to - from + 1))
  return result
end
```

Generátor můžeme v LuaTEXu použít následně:

```
random = Random.new(42)         -- Vytvoř generátor se semínkem 42.
for i = 1, 30 do                -- Vypiš 30 hodů 6stěnnou kostkou.
```

```
  tex.sprint(random:get_next_number(1, 6))
  if i < 30 then tex.sprint(", ") end
end
```

4, 2, 1, 2, 3, 1, 1, 4, 4, 2, 4, 5, 3, 4, 6, 3, 1, 5, 4, 2, 1, 3, 3, 4, 6, 5, 5, 2, 6, 6

2.3. Náhodné procházky

Náhodné procházky Katzova couvajícího modelu sestávají z elementárních náhodných kroků jednotlivých zapomnětlivých modelů.

```
177 local function take_random_step(graph, context, random)
178   if graph[context] == nil then  -- Pokud neexistuje vrchol pro
179     return nil                   -- současný kontext, selži.
180   end
181   local vertex = graph[context]  -- Vyber vrchol pro současný
182   assert(#graph[context] > 0)    -- kontext.
183   local total_weight = 0         -- Sečti hodnoty odchozích hran.
184   local weight
185   for _, character in ipairs(vertex) do
186     weight = vertex[character]
187     total_weight = total_weight + weight
188   end
189   local random_number =          -- Vyber náhodnou hranu.
190     random:get_next_number(0, total_weight)
191   local weight_accumulator = 0
192   for _, character in ipairs(vertex) do
193     weight = vertex[character]
194     weight_accumulator = weight_accumulator + weight
195     if weight_accumulator >= random_number then
196       return character           -- Navrať znak na hraně.
197     end
198   end
199   assert(false)                  -- Sem bychom se neměli dostat.
200 end
201
202 function ForgetfulModel.take_random_step(model, graph_type,
203                                          context, random)
204   local graph = model[           -- Umožni náhodné procházky
205     graph_type .. "_graph"]      -- zepředu i pozpátku.
206   assert(graph ~= nil)
207   context = trim_context(        -- Seřízni kontext na
208     context,                     -- požadovanou délku.
209     model.context_size)
```

```
210   return take_random_step(graph, context, random)
211 end
212
213 function BackoffModel.take_random_walk(model, graph_type,
214                                        affix, random,
215                                        min_context_size,
216                                        max_context_size)
217   local name = affix              -- Umožni zadat buď předponu,
218   if graph_type == "reverse"      -- nebo příponu jména.
219     then name = reverse(name)
220   end
221   while true do
222     local character
223     model:for_each_submodel(
224       function(submodel)
225         character = submodel:take_random_step(
226           graph_type, name, random)
227         if character ~= nil then  -- Najdi zapomnětlivý
228           return false            -- model s nejvyšší
229         end                       -- délkou kontextu, ve
230       end,                        -- kterém existuje vrchol
231       min_context_size,           -- pro současný kontext.
232       max_context_size)
233     if character == nil then      -- Pokud takový model
234       return nil                  -- neexistuje, selži.
235     end
236     if character == "\n" then     -- Pokud jsme vygenerovali
237       break                       -- znak konce řádku,
238     end                           -- ukončíme procházku.
239     name = name .. character
240   end
241   if graph_type == "reverse"      -- Při náhodné procházce
242     then name = reverse(name)     -- pozpátku otoč
243   end                             -- vygenerované jméno.
244   return name                     -- Navraťvygenerované jméno.
245 end
```

Náhodné procházky provedeme v LuaTEXu následně:

```
for i = 1, 5 do  -- Vypiš výsledky 5 procházek s předponou pe- se
  tex.sprint(    -- zapomnětlivým modelem s délkou kontextu 6.
    model:take_random_walk("forward", "pe", random, 6, 6))
  if i < 5 then tex.sprint(", ") end
```

```
end
```
pejsek, pes filipes, pes filipes, pejsek, pes filipes
```
for i = 1, 20 do -- Vypiš výsledky 20 procházek s příponou -k se
  tex.sprint(    -- zapomnětlivým modelem s délkou kontextu 6.
    model:take_random_walk("reverse", "k", random, 6, 6))
  if i < 20 then tex.sprint(", ") end
end
```
pejsek, pejsek, pejsek, pejsek, pejsek, pejsek, pejsek, pejsek, pejsek, pejsek, pejsek, pejsek, maxipes fík, pejsek, pejsek, pejsek, pejsek, pejsek, maxipes fík, pejsek
```
for i = 1, 5 do  -- Vypiš výsledky 5 procházek bez afixu se
  tex.sprint(    -- zapomnětlivým modelem s délkou kontextu 3.
    model:take_random_walk("forward", "", random, 3, 3))
  if i < 5 then tex.sprint(", ") end
end
```
pes filipes, pes filipes, maxipes fík, pes filipes filipes filipes, pes
```
for i = 1, 10 do -- Vypiš výsledky 10 procházek bez afixu se
  tex.sprint(    -- zapomnětlivým modelem s délkou kontextu 2.
    model:take_random_walk("forward", "", random, 2, 2))
  if i < 10 then tex.sprint(", ") end
end
```
pes filipes filipes, pes filipes, maxipejsek, maxipes filipes filipes fík, maxipes fík, maxipes filipes filipejsek, pes, pes fík, pes filipejsek, pes
```
for i = 1, 10 do -- Vypiš výsledky 10 procházek bez afixu se
  tex.sprint(    -- zapomnětlivým modelem s délkou kontextu 1.
    model:take_random_walk("forward", "", random, 1, 1))
  if i < 10 then tex.sprint(", ") end
end
```
pes, pes fipes fililipes, maxipejs fík, pes fipes fipes fipek, pek, pes fipejs, pes fililipes filipek, maxilipes filipesejs fililililipejs fipek, maxipes fipejseses, maxipes
```
for i = 1, 5 do  -- Vypiš výsledky 5 procházek s předponou i- nad
  tex.sprint(    -- Katzovým modelem s délkami kontextu 1 až 4.
    model:take_random_walk("forward", "i", random, 1, 4))
  if i < 5 then tex.sprint(", ") end
end
```
ipes filipes filipes filipes, ipes filipes fík, ilipes filipes, ilipes, ilipes

3. Uživatelské rozhraní pro vkládání jmen do dokumentů

V této sekci ukončíme soubor `randomnames.lua` a vytvoříme dva nové soubory `randomnames.tex` a `randomnames.sty`, které budou obsahovat uživatelské roz-

hraní pro snadné využití našeho jazykového modelu v LaTeXových dokumentech. Hotové soubory `randomnames.lua`, `randomnames.tex` a `randomnames.sty` můžeme stáhnout online [6].

Nejprve v sekci 3.1 doplníme do souboru `randomnames.lua` fasádu, která poslouží jako programátorské rozhraní pro naši implementaci v jazyce Lua. Následně v sekci 3.2 na straně 19 vytvoříme soubor `randomnames.tex` s přemostěním mezi Luou a TeXem pomocí programovacího jazyka expl3. Nakonec v sekci 3.3 na straně 25 vytvoříme soubor `randomnames.sty` s uživatelským rozhraním pro snadné využití našeho jazykového modelu v LaTeXových dokumentech.

3.1. Fasáda pro vytváření jazykových modelů v jazyce Lua

Dosud jsme náš jazykový model a generátor pseudonáhodných čísel ukládali do proměnných `model` a `random`. Při práci s mnoha modely a generátory je ale použití proměnných nepřehledné. V této sekci vytvoříme fasádu, která nám umožní vytvářet pojmenované modely s přidruženými generátory a snadno k nim přistupovat bez použití proměnných.

```
276 local randomnames = {}       -- Veřejné rozhraní našeho Lua modulu.
277
278 local models = {}            -- Vytvoř prázdnou databázi modelů
279 local randoms = {}           -- a generátorů pseudonáhodných čísel.
280
281 local function hash(name)    -- Vypočti ze jména modelu počáteční
282   local result = 0            -- semínko s hašovací funkcí sdbm.
283   local modulus = Random.M - 1
284   for _, code in utf8.codes(name) do
285     result = multiply_modulo(result, 65599, modulus)
286     result = (result + code) % modulus
287   end
288   return result
289 end
290
291 function randomnames.new_model(name, min_context_size,
292                                max_context_size, seed)
293   assert(models[name] == nil,
294          [[Model named "]] .. name .. [[" already exists]])
295   min_context_size =          -- Pokud uživatel nedodal minimální
296     min_context_size or 1     -- nebo maximální délku kontextu,
297   max_context_size =          -- použij výchozí hodnoty 1 a 3.
298     max_context_size or 3
299   local model =               -- Vytvoř nový jazykový model.
300     BackoffModel.new(min_context_size, max_context_size)
```

```
301   if seed == nil then        -- Pokud uživatel nedodal náhodné
302     seed = hash(name)        -- semínko, spočítej ho jako haš
303   end                        -- jména jazykového modelu.
304   local random =             -- Vytvoř nový generátor
305     Random.new(seed)         -- pseudonáhodných čísel.
306   models[name] = model       -- Ulož model a generátor do
307   randoms[name] = random     -- databáze pod zadaným jménem.
308 end
309
310 local function get_model(name)
311   local model
312   model = models[name]       -- Zkus načíst z databáze model.
313   assert(model ~= nil,       -- Pokud neexistuje, vypiš chybu.
314          [[Model named "]] .. name .. [[" does not exist]])
315   return model
316 end
317
318 function randomnames.add_name(model_name, name)
319   local model =              -- Načti z databáze model
320     get_model(model_name)    -- a zanes do něj jméno.
321   model:add_name(name)
322 end
323
324 function randomnames.input_names(model_name, filename)
325   local model =              -- Načti z databáze model
326     get_model(model_name)    -- a zanes do něj jména.
327   model:input_names(filename)
328 end
329
330 function randomnames.take_random_walk(name, graph_type,
331                                       affix, seed,
332                                       min_context_size,
333                                       max_context_size)
334   local model =              -- Načti z databáze model.
335     get_model(name)
336   graph_type = graph_type    -- Pokud uživatel nedodal směr
337             or "forward"     -- procházky, jdi zepředu.
338   affix = affix              -- Pokud uživatel nedodal předponu
339         or ""                -- ani příponu, použij prázdnou.
340   local random               -- Pokud uživatel nedodal náhodné
341   if seed == nil then        -- semínko, načti z databáze
342     random = randoms[name]   -- generátor náhodných čísel
```

```
  else                        -- přidružený k modelu.
    random =                  -- Jinak vytvoř nový generátor
      Random.new(seed)        -- pomocí náhodného semínka.
  end
  if min_context_size         -- Pokud uživatel nedodal
       == nil then            -- minimální nebo maximální
     min_context_size =       -- délku kontextu, použij
       model.                 -- hodnoty z modelu.
         min_context_size
  end
  if max_context_size == nil then
     max_context_size = model.max_context_size
  end
  local result =              -- Proveď náhodnou procházku.
    model:take_random_walk(graph_type, affix, random,
                           min_context_size,
                           max_context_size)
  return result
end
```

Pomocí naší fasády vytvoříme Katzův couvající jazykový model a provedeme v něm náhodnou procházku v LuaTEXu následně:

```
randomnames.new_model("pes", 1, 3, 22)          -- Vytvoř model.
randomnames.add_name("pes", "pes filipes")      -- Zanes do modelu
randomnames.add_name("pes", "pejsek")           -- jména
randomnames.add_name("pes", "maxipes fík")      -- pohádkových psů.
tex.print(randomnames.take_random_walk("pes"))  -- Vygeneruj jméno.
```

maxipes filipes fík

Na konci souboru `randomnames.lua` navrátíme naši fasádu:

```
return randomnames
```

Poté můžeme v LuaTEXu načíst fasádu odkudkoliv příkazem `require()`:

```
local randomnames =  -- Načti fasádu ze souboru randomnames.lua.
   require("randomnames")
```

3.2. Přemostění mezi jazyky Lua a TEX pomocí jazyka expl3

Abychom mohli jazykové modely používat přímo z TEXu, budeme nyní chtít napojit naši fasádu na TEXová makra. Na rozdíl od jazyka Lua, ve kterém můžeme použít speciální prázdnou hodnotu `nil` pro nepovinné parametry, nemáme v jazyce TEX přímočarý způsob, jak část parametrů funkce vynechat. Využijeme proto datový typ key–value programovacího jazyka expl3 [11, kapitola 26], která nám při volání funkcí umožní uvést pouze část nepovinných parametrů.

```
\ifx \ExplSyntaxOn \undefined          % Načti programovací jazyk
  \input expl3-generic                 % expl3 do plain TeXu.
\fi
\ExplSyntaxOn

\tl_new:N                              % Vytvoř lokální proměnné
  \l_randomnames_min_context_size_tl   % pro nepovinné parametry
\tl_new:N                              % funkce randomnames.
  \l_randomnames_max_context_size_tl   % new_model().
\tl_new:N \l_randomnames_seed_tl
\tl_gset:Nn \l_randomnames_min_context_size_tl { nil }
\tl_gset:Nn \l_randomnames_max_context_size_tl { nil }
\tl_gset:Nn \l_randomnames_seed_tl { nil }

\keys_define:nn                        % Vytvoř sběrnici,[4] která
  { randomnames / new_model }          % naplní lokální proměnné
  {                                    % zadanými hodnotami
    min_context_size .code:n =         % nepovinných parametrů
      {                                % ve formátu požadovaném
        \tl_set:Nn                     % jazykem Lua.
          \l_randomnames_min_context_size_tl
          { tonumber(" \lua_escape:e { #1 } ") }
      },
    max_context_size .code:n =
      {
```

[4] V programovacím jazyku expl3 můžeme s datovým typem key–value nakládat dvěma způsoby. Mějme např. key–value `{ min_context_size = 3, seed = 42 }`. Tento key–value můžeme uložit do hašové tabulky [11, kapitola 24] a následně přistupovat k hodnotám jednotlivých klíčů:

```
\prop_set_from_keyval:Nn               % Vytvoř v pomocné proměnné
  \l_tmpa_prop                         % \l_tmpa_prop hašovou tabulku
  { min_context_size = 3, seed = 42 }  % a naplň ji naším key-value.
Náhodné~semínko~je:~
\prop_item:Nn                          % Získej hodnotu klíče seed
  \l_tmpa_prop                         % z naší hašové tabulky.
  { seed } .
```

Alternativně můžeme vytvořit datovou sběrnici [11, kapitola 26], u které nejprve nastavíme způsob zpracování hodnot jednotlivých klíčů a následně do ní náš key–value vpustíme:

```
\keys_define:nn                        % Vytvoř datovou sběrnici, která
  { randomnames / process_key_value }  % při zpracování klíče seed
  { seed .code:n = { Náhodné~semínko~je:~#1. } }  % vypíše jeho hodnotu.
\keys_set:nn                           % Vpusť do datové sběrnice náš
  { randomnames / process_key_value }  % key-value.
  { min_context_size = 3, seed = 42 }
```

Oba příklady vypíšou text „Náhodné semínko je 42.“. V našem přemostění používáme druhý popsaný způsob s datovou sběrnicí.

```
        \tl_set:Nn
          \l_randomnames_max_context_size_tl
          { tonumber(" \lua_escape:e { #1 } ") }
      },
    context_size .meta:n =              % Umožni snadno vytvářet
      {                                 % zapomnětlivé modely
        min_context_size = { #1 },      % s pevnou délkou kontextu,
        max_context_size = { #1 },      % které nemohou couvat.
      },
    seed .code:n =
      {
        \tl_set:Nn
          \l_randomnames_seed_tl
          { tonumber(" \lua_escape:e { #1 } ") }
      },
  }

\cs_new:Nn
  \randomnames_new_model:nn
  {
    \group_begin:
    \keys_set:nn                        % Naplň lokální proměnné
      { randomnames / new_model }       % zadanými hodnotami
      { #1 }                            % nepovinných parametrů.
    \lua_now:e                          % Zavolej funkci
      {                                 % randomnames.new_model()
        local~randomnames =             % v jazyce Lua.[5]
          require("randomnames")
        randomnames.new_model(
          " \lua_escape:e { #2 } ",
          \l_randomnames_min_context_size_tl,
          \l_randomnames_max_context_size_tl,
          \l_randomnames_seed_tl
        )
      }
    \group_end:                         % Resetuj hodnoty lokálních
  }                                     % proměnných zpět na nil.
```

[5] Znak vlnovky (~) v jazyce expl3 vkládá mezeru, která nám v kódu v jazyce Lua odděluje klíčové slovo `local` od názvu proměnné `randomnames`. Pokud bychom vlnovku neuvedli, definovali bychom místo lokální proměnné `randomnames` globální proměnnou `localrandomnames`.

```
\cs_new:Nn
  \randomnames_add_name:nn
  {
    \lua_now:e                          % Zavolej funkci
      {                                 % randomnames.add_name()
        local~randomnames =             % v jazyce Lua.
          require("randomnames")
        randomnames.add_name(
          " \lua_escape:e { #1 } ",
          " \lua_escape:e { #2 } "
        )
      }
  }

\cs_new:Nn
  \randomnames_input_names:nn
  {
    \lua_now:e                          % Zavolej funkci
      {                                 % randomnames.
        local~randomnames =             % input_names() v jazyce
          require("randomnames")        % Lua.
        randomnames.input_names(
          " \lua_escape:e { #1 } ",
          " \lua_escape:e { #2 } "
        )
      }
  }

\tl_new:N                               % Vytvoř lokální proměnné
  \l_randomnames_graph_type_tl          % pro nepovinné parametry
\tl_new:N                               % funkce randomnames.
  \l_randomnames_affix_tl               % take_random_walk().
\tl_gset:Nn \l_randomnames_graph_type_tl { nil }
\tl_gset:Nn \l_randomnames_affix_tl { nil }

\bool_new:N                             % Vytvoř proměnné, pomocí
  \l_randomnames_save_bool              % kterých bude uživatel moci
\bool_gset_false:N                      % ukládat jména, jež využívá
  \l_randomnames_save_bool              % opakovaně.
\tl_new:N \l_randomnames_save_tl

\keys_define:nn                         % Vytvoř sběrnici, která
```

```
    { randomnames / take_random_walk }  % naplní lokální proměnné
    {                                   % zadanými hodnotami
      graph_type .code:n =              % nepovinných parametrů
        {                               % ve formátu požadovaném
          \tl_set:Nn                    % jazykem Lua.
            \l_randomnames_graph_type_tl
            { " \lua_escape:e { #1 } " }
        },
      affix .code:n =
        {
          \tl_set:Nn
            \l_randomnames_affix_tl
            { " \lua_escape:e { #1 } " }
        },
      save .code:n =                    % Umožni ukládat
        {                               % vygenerovaná jména
          \bool_set_true:N              % pro opakované použití.
            \l_randomnames_save_bool
          \tl_set:Nn \l_randomnames_save_tl { #1 }
        },
      prefix .meta:n =                  % Umožni zadávat předpony
        {                               % a přípony náhodných
          graph_type = { forward },     % procházek tak, aby se
          affix = { #1 },               % zároveň automaticky
        },                              % nastavil správný směr
      suffix .meta:n =                  % náhodné procházky.
        {
          graph_type = { reverse },
          affix = { #1 },
        },
    }
  \keys_define:nn                       % Poděď zpracování
    { randomnames }                     % nepovinných parametrů
    {                                   % funkce randomnames.
      take_random_walk .inherit:n =     % new_model(), abychom
        { randomnames / new_model },    % se vyhnuli duplikaci
    }                                   % kódu.

  \cs_new:Nn
    \randomnames_take_random_walk:nn
    {
      \group_begin:
```

```
    \keys_set:nn                          % Naplň lokální proměnné
      {                                   % zadanými hodnotami
        randomnames /                     % nepovinných parametrů.
        take_random_walk
      }
      { #1 }
    \lua_now:e                          % Zavolej funkci randomnames.
      {                                 % take_random_walk()
        local~randomnames =             % v jazyce Lua.
          require("randomnames")
        local~result = randomnames.take_random_walk(
          " \lua_escape:e { #2 } ",
          \l_randomnames_graph_type_tl,
          \l_randomnames_affix_tl,
          \l_randomnames_seed_tl,
          \l_randomnames_min_context_size_tl,
          \l_randomnames_max_context_size_tl
        )
        token.set_macro(                % Ulož vygenerované jméno
          "l_tmpa_tl", result)          % do proměnné \l_tmpa_tl
      }                                 % jazyka expl3.
    \bool_if:NT                         % Pokud uživatel uvedl
      \l_randomnames_save_bool          % parametr save, ulož
      {                                 % vygenerované jméno
        \cs_gset:cpx                    % do zadaného TeXového
          \l_randomnames_save_tl        % příkazu.
          { \l_tmpa_tl }
      }
    \tl_use:N \l_tmpa_tl                % Vlož jméno na výstup.
    \group_end:                         % Resetuj hodnoty lokálních
  }                                     % proměnných zpět na nil.

\ExplSyntaxOff
```

Pomocí našeho přemostění vytvoříme Katzův couvající jazykový model a provedeme v něm náhodnou procházku v plain LuaTEXu takto:

```
\input randomnames                                  % Načti naše
\ExplSyntaxOn                                       % přemostění.
\randomnames_new_model:nn                           % Vytvoř model.
  {
    min_context_size = 1,
    max_context_size = 3,
```

```
    seed = 22,
  }
  { pes }
\randomnames_add_name:nn { pes } { pes~filipes } % Zanes do modelu
\randomnames_add_name:nn { pes } { pejsek }      % jména
\randomnames_add_name:nn { pes } { maxipes~fík } % pohádkových psů.
\randomnames_take_random_walk:nn { } { pes }     % Vygeneruj jméno.
\ExplSyntaxOff \bye
```

maxipes filipes fík

3.3. Uživatelské rozhraní pomocí LaTeXového balíčku xparse

Programovací jazyk expl3 má pro uživatele TeXu poměrně neobvyklou syntax. Pomocí LaTeXového balíčku xparse proto vytvoříme uživatelské rozhraní, které bude pro uživatele přirozenější.

```
\input randomnames\relax                % Načti naše přemostění.

\ProvidesExplPackage
  {randomnames}%
  {2023-06-25}%
  {1.0.2}%
  {Character name generators for creative writing in LuaLaTeX}

\RequirePackage{xparse}                 % Načti balíček xparse.

\NewDocumentCommand                     % Vytvoř LaTeXový příkaz
  { \newmodel }                         % \newmodel s jedním
  { O{} m }                             % nepovinným a jedním
  {                                     % povinným argumentem,
    \randomnames_new_model:nn           % který zavolá funkci
      { #1 }                            % randomnames.new_model()
      { #2 }                            % v jazyce Lua.
  }

\NewDocumentCommand                     % Vytvoř LaTeXový příkaz
  { \addname }                          % \addname se dvěma
  { m m }                               % povinnými argumenty,
  {                                     % který zavolá funkci
    \randomnames_add_name:nn            % randomnames.add_name()
      { #1 }                            % v jazyce Lua.
      { #2 }
  }
```

```
\NewDocumentCommand                     % Vytvoř LaTeXový příkaz
  { \inputnames }                       % \inputnames se dvěma
  { m m }                               % povinnými argumenty,
  {                                     % který zavolá funkci
    \randomnames_input_names:nn         % randomnames.input_names()
      { #1 }                            % v jazyce Lua.
      { #2 }
  }

\NewDocumentCommand                     % Vytvoř LaTeXový příkaz
  { \randomname }                       % \randomname s jedním
  { O{} m }                             % nepovinným a jedním
  {                                     % povinným argumentem,
    \randomnames_take_random_walk:nn    % který zavolá funkci
      { #1 }                            % randomnames.
      { #2 }                            % take_random_walk()
  }                                     % v jazyce Lua.
```

Pomocí našeho uživatelského rozhraní příkazů vytvoříme Katzův couvající jazykový model a provedeme v něm náhodnou procházku v LuaLaTeXu následně:

```
\documentclass{article}
\usepackage[czech]{babel}
\usepackage{randomnames}
\newmodel[min_context_size=1,                   % Vytvoř model.
          max_context_size=3,
          seed=22]{pes}
\addname{pes}{pes filipes}                      % Zanes do modelu
\addname{pes}{pejsek}                           % jména
\addname{pes}{maxipes fík}                      % pohádkových psů.
\begin{document}
\randomname{pes}                                % Vygeneruj jméno.
\end{document}
```

maxipes filipes fík

4. Příklady užití

V této sekci stáhneme několik příkladových databází jmen, natrénujeme na nich náš model a ukážeme si, jak bychom vygenerovali jména postav a zahrnuli je do textu povídky.

Při stahování příkladových databází používáme terminál UNIXového systému s příkazovým procesorem `/bin/bash` a s nainstalovaným programem `xmllint` z knihovny libxml2. Příkladové dokumenty si můžeme uložit do textového souboru

`priklad.tex` a přeložit příkazem `lualatex priklad.tex`. Pro úspěšný překlad stáhneme do pracovního adresáře soubory `randomnames.lua`, `randomnames.tex` a `randomnames.sty` [6], které jsme vytvořili v sekcích 2 a 3.

4.1. Soudobá křestní jména

V soudobé fikci se nám může hodit generovat křestní jména postav. Stáhneme si proto z webu rodina.cz seznam 629 mužských a 605 ženských křestních jmen:

```
$ wget -O- https://www.rodina.cz/scripts/jmena/default.asp?muz=0 |
  xmllint -html -xpath '//a[contains(@href,"jmeno.asp")]/text()' \
  - > krestni-jmena-zeny.txt
$ wget -O- https://www.rodina.cz/scripts/jmena/default.asp?muz=1 |
  xmllint -html -xpath '//a[contains(@href,"jmeno.asp")]/text()' \
  - > krestni-jmena-muzi.txt
```

Následně sestavíme dva modely, jeden pro mužská křestní jména a druhý pro ženská křestní jména, a použijeme je v textu povídky:

```
\documentclass{article}
\usepackage[czech]{babel}
\usepackage{randomnames}
\newmodel{zena}
\newmodel{muz}
\inputnames{zena}{krestni-jmena-zeny.txt}
\inputnames{muz}{krestni-jmena-muzi.txt}
\begin{document}
Když se setkali na výstavě psů, \randomname[prefix=Ro]{muz}
a \randomname[suffix=lie]{zena} si okamžitě uvědomili, že se už
nikdy nebudou chtít rozloučit.
\end{document}
```

Když se setkali na výstavě psů, Rodan a Aurélie si okamžitě uvědomili, že se už nikdy nebudou chtít rozloučit.

V našem příkladu jsme použili výchozí maximální délku kontextu 3, abychom modely pobídli k vyšší kreativitě. Už s délkou kontextu 6 bychom ale dokázali vygenerovat 23 nových přesvědčivých jmen, která nebyla součástí množiny trénovacích jmen z webu rodina.cz: Adalbertina, Adalbertýna, Budimíra, Budislava, Erharda, Fridolína, Gvendolín, Jarolína, Jonathanael, Karolím, Karolíma, Klementýn, Mojmíra, Mstislava, Něhoslava, Stanimíra, Stojmíra, Svatomíra, Šebastian, Tichomíra, Velimíra, Vítoslava a Věslav.

4.2. Postavy světa J. R. R. Tolkiena

Ve fanfikci z fantasy světa anglického spisovatele J. R. R. Tolkiena se nám může hodit generovat jména postavám různých ras. Stáhneme proto z webu

behindthename.com seznam 51 jmen trpaslíků, 19 jmen elfek a 316 jmen lidských mužů.

```
$ download_tolkien_names() {
  wget -O- https://behindthename.com/namesakes/list/tolkien/alpha|
  xmllint -html -xpath '//div[@id="div_refinepage"]/table/
  tr[td[2]/text() = "'$2'" and td[3]/text() = "'$1'"]/td[1]' - |
  sed 's/<[^>]*>//g'; }
$ download_tolkien_names Dwarf m > tolkien-jmena-trpasliku.txt
$ download_tolkien_names Elf f > tolkien-jmena-elfek.txt
$ download_tolkien_names Man m > tolkien-jmena-lidskych-muzu.txt
```

Následně sestavíme tři modely, jeden pro jména trpaslíků, druhý pro jména elfek a třetí pro jména lidských můžů, a použijeme je v textu fanfikce:

```
\documentclass{article}
\usepackage[czech]{babel}
\usepackage{randomnames}
\newmodel[context_size=2,seed=35]{trpaslik}
\newmodel[max_context_size=2,seed=6]{elfka}
\newmodel[seed=2419]{clovek}
\inputnames{trpaslik}{tolkien-jmena-trpasliku.txt}
\inputnames{elfka}{tolkien-jmena-elfek.txt}
\inputnames{clovek}{tolkien-jmena-lidskych-muzu.txt}
\begin{document}
Tři hrdinové - trpaslík, elf a člověk - se vydali do temného lesa,
aby zastavili invazi nepřátelských vrrků. Trpaslík
\randomname{trpaslik} byl nastražen jako návnada na zuby těchto
dravých zvířat, elfka \randomname[suffix=iel]{elfka} připravovala
luk, zatímco člověk \randomname{clovek} divoce máchal mečem.
\end{document}
```

Tři hrdinové – trpaslík, elf a člověk – se vydali do temného lesa, aby zastavili invazi nepřátelských vrrků. Trpaslík Duri byl nastražen jako návnada na zuby těchto dravých zvířat, elfka Amariel připravovala luk, zatímco člověk Túrindor divoce máchal mečem.

Pokud neuvedeme náhodné semínko, každé použití příkazu `\randomname` vygeneruje nové jméno. Pro zachování vygenerovaných jmen můžeme použít parametr `save`, kterým jméno postavy trvale uložíme:

```
\randomname[save=Žoldnéř]{clovek} vyrostl v bohaté obchodnické
rodině. Když však jeho otec zesnul a rodinné jmění zpustlo,
\Žoldnéř\ se stal dobrodruhem a nechal se najímat jako žoldnéř.
Během svých dobrodružství si \Žoldnéř\ vydobyl pověst neohroženého
válečníka a jeho meč se stal jeho nejcennějším jměním.
```

Vstupní procesor LuaTEXu zná Unicode a můžeme tedy psát příkazy s diakritikou jako `\Žoldnéř`.

Túrindor vyrostl v bohaté obchodnické rodině. Když však jeho otec zesnul a rodinné jmění zpustlo, Túrindor se stal dobrodruhem a nechal se najímat jako žoldnéř. Během svých dobrodružství si Túrindor vydobyl pověst neohroženého válečníka a jeho meč se stal jeho nejcennějším jměním.

4.3. Bytosti v Cthulhu mýtu H. P. Lovecrafta

Ve fanfikci z hororového světa amerického spisovatele H. P. Lovecrafta se nám může hodit generovat jména kosmických božstev, tzv. *prastarých*. Stáhneme proto z anglické wikipedie seznam 157 jmen prastarých:

```
$ wget -O- https://en.wikipedia.org/wiki/List_of_Great_Old_Ones |
  xmllint -html -xpath '//table[2]//td[1]/text()' - | grep . \
  > lovecraft-jmena-prastarych.txt
```

Následně sestavíme model jmen prastarých a použijeme ho v textu fanfikce:

```
\documentclass{article}
\usepackage[czech]{babel}
\usepackage{randomnames}
\newmodel[seed=140]{prastary}
\inputnames{prastary}{lovecraft-jmena-prastarych.txt}
\begin{document}
Po západu slunce se skupina šílených kultistů shromáždila
kolem oltáře, aby přivolali svého pána. Prastarý
\randomname[prefix=C']{prastary}, jehož jméno nebylo možné
vyslovit lidskými ústy, se brzy objevil a jeho oči temně zářily.
\end{document}
```

Po západu slunce se skupina šílených kultistů shromáždila kolem oltáře, aby přivolali svého pána. Prastarý C'tya-Yg'Nalla, jehož jméno nebylo možné vyslovit lidskými ústy, se brzy objevil a jeho oči temně zářily.

4.4. Kouzla ve světě Harryho Pottera

Ve fanfikci ze světa Harryho Pottera se nám může hodit generovat nová jména kouzel. Stáhneme proto z webu harrypotter.fandom.com seznam 192 kouzel:

```
$ wget -O- https://harrypotter.fandom.com/wiki/List_of_spells |
  xmllint -html -xpath '//h3//i/a/text()' - \
  > harry-potter-kouzla.txt
```

Následně sestavíme model jmen kouzel a použijeme ho v textu fanfikce:

```
\documentclass{article}
\usepackage[czech]{babel}
\usepackage{randomnames}
```

```
\newmodel{kouzlo}
\inputnames{kouzlo}{harry-potter-kouzla.txt}
\begin{document}
Tesák, Hagridův pes, ležel před chatrčí a pozoroval okolní les.
Najednou uslyšel hlasitý křik z nedaleké cesty. Překvapeně se
postavil na nohy a natahoval uši. Když zpozoroval kouzelníka, jak
na něj mává hůlkou a křičí „\randomname[seed=2136]{kouzlo}!",
okamžitě sebou trhl a rozběhl se k němu s otevřenou tlamou.
\end{document}
```

Tesák, Hagridův pes, ležel před chatrčí a pozoroval okolní les. Najednou uslyšel hlasitý křik z nedaleké cesty. Překvapeně se postavil na nohy a natahoval uši. Když zpozoroval kouzelníka, jak na něj mává hůlkou a křičí „Furnunculus!“, okamžitě sebou trhl a rozběhl se k němu s otevřenou tlamou.

4.5. Kapesní příšerky Pokémon

Ve fanfikci ze světa kapesních příšerek Pokémon se nám může hodit generovat jména pokémonů. Stáhneme proto z webu bulbapedia.bulbagarden.net seznam 1015 jmen pokémonů:

```
$ wget -O- 'https://bulbapedia.bulbagarden.net/wiki/List_of_'\
  'Pokémon_by_name' | xmllint -html -xpath '//table//img/@alt' - |
  sed -r 's/ alt="(.*)"/\1/' > jmena-pokemonu.txt
```

Následně sestavíme model jmen pokémonů a použijeme ho v textu fanfikce:

```
\documentclass{article}
\usepackage[czech]{babel}
\usepackage{randomnames}
\newmodel[max_context_size=4,seed=4]{pokemon}
\inputnames{pokemon}{jmena-pokemonu.txt}
\begin{document}
Cestování po světě pokémonů může být plné nečekaných překvapení.
Onoho dne se trenér z Rumělkového města procházel po parku, když
tu náhle na něj z vysokého podrostu vyskočil psí pokémon
\randomname[prefix=Haf]{pokemon} a hlasitě zaštěkal. Od té doby
trenér věděl, že musí být vždy připraven na nečekané situace.
\end{document}
```

Cestování po světě pokémonů může být plné nečekaných překvapení. Onoho dne se trenér z Rumělkového města procházel po parku, když tu náhle na něj z vysokého podrostu vyskočil psí pokémon Hafeon a hlasitě zaštěkal. Od té doby trenér věděl, že musí být vždy připraven na nečekané situace.

5. Budoucí práce

V této sekci uvedeme možná vylepšení vyvinutého jazykového modelu a další aplikace jazykových modelů v TEXu, na které by se měla zaměřit budoucí práce.

5.1. Automatické skloňování českých a slovenských jmen

Pokud nemáme trénovací data v různých gramatických pádech a číslech, bude náš jazykový model generovat jména pouze v prvním pádě jednotného čísla. V analytickém jazyce, jako je angličtina, to není problém. U flektivních jazyků, jako jsou čeština a slovenština, bychom ale chtěli, aby náš model uměl jména nejen generovat, ale také skloňovat.

Tereza Vrabcová [12] navrhla následující algoritmus skloňování českých jmen:

1. Pro každé jméno J v trénovací množině:
   (a) Pokud jméno existuje ve wikislovníku, stáhni jeho skloňování odtud.
   (b) Jinak stáhni skloňování jména z webu sklonuj.cz.
2. Pro každé automaticky vygenerované jméno J':
   (a) Najdi v trénovací množině jméno J s nejpodobnější koncovkou:

$$J = \arg\max_{J} \sum_{i=1,\ldots,\min(|J|,|J'|)} \begin{cases} \frac{1}{2^i} & J[|J|-i+1] = J'[|J'|-i+1], \\ 0 & \text{jinak.} \end{cases}$$

   (b) Skloňuj jméno J' stejně jako jméno J.

Pro vyzkoušení algoritmu stáhneme do pracovního adresáře soubory `get_data.py`, `requirements.txt`, `declension.lua`, `declension.tex` a `declension.sty` [12]. Následně si stáhneme z webu rodina.cz česká křestní jména podle pokynů v sekci 4.1 na straně 27. Dále si pomocí skriptu `get_data.py` stáhneme skloňování jmen:

```
$ python3 -m pip install -r requirements.txt
$ python3 ./get_data.py krestni-jmena-zeny.txt krestni-jmena-zeny
$ python3 ./get_data.py krestni-jmena-muzi.txt krestni-jmena-muzi
```

Nakonec sestavíme dva modely, jeden pro mužská křestní jména a druhý pro ženská křestní jména, a použijeme je v textu povídky:

```
\documentclass{article}
\usepackage[czech]{babel}
\usepackage{randomnames}
\newmodel{zena}
\newmodel{muz}
\inputnames{zena}{krestni-jmena-zeny.txt}
\inputnames{muz}{krestni-jmena-muzi.txt}
\usepackage{declension}
\newdeclmodel{zena}
```

```
\newdeclmodel{muz}
\loaddata{zena}{krestni-jmena-zeny.decl}{krestni-jmena-zeny.names}
\loaddata{muz}{krestni-jmena-muzi.decl}{krestni-jmena-muzi.names}
\begin{document}
Když se setkali na výstavě psů, \randomname[prefix=Ro,save=On]
{muz} a \randomname[suffix=lie,save=Ona]{zena} si okamžitě
uvědomili, že se už nikdy nebudou chtít rozloučit. \decline{zena}
{\Ona}{1}{s} si uvědomila, že bez \decline{muz}{\On}{2}{s} nechce
žít. \decline{muz}{\On}{1}{s} si ve stejný okamžik uvědomil, že
s \decline{zena}{\Ona}{7}{s} chce žít navždy.
\end{document}
```

Když se setkali na výstavě psů, Rodan a Aurélie si okamžitě uvědomili, že se už nikdy nebudou chtít rozloučit. Aurélie si uvědomila, že bez Rodana nechce žít. Rodan si ve stejný okamžik uvědomil, že s Aurélií chce žít navždy.

Popsaný algoritmus funguje pouze pro česká jména a hledání jména s nejpodobnější koncovkou má lineární časovou složitost ve velikosti trénovací množiny. Budoucí práce by se měla zaměřit na rozšíření algoritmu o další flektivní jazyky jako slovenština a použít datovou strukturu jako trie pro efektivní hledání trénovacích jmen s nejdelší společnou koncovkou.

5.2. Omezující podmínky pro generovaná jména

Náš jazykový model umožňuje uživateli zadat buď předponu, nebo příponu vygenerovaného jména, ale ne obojí zároveň. Dále uživatel nemůže ovlivnit střed vygenerovaného jména ani omezit jeho délku. Ve chvíli, kdy uživatel není s vygenerovaným jménem spokojený, musí jméno generovat opakovaně s různými náhodnými semínky. Tento zdlouhavý proces bychom rádi urychlili tak, že uživateli umožníme zadat dodatečné omezující podmínky.

5.2.1. Permutermíny

Pro rozšířené vyhledávání v triích se využívají tzv. *permutermíny* [13, sekce 3.2.1]. Při sestavování trie použijeme místo slova „pejsek“ veškeré jeho rotace (permutermíny) doplněné o metasymbol $, který udává konec slova: „pejsek$“, „ejsek$p“, „jsek$pe“, „sek$pej“, „ek$pejs“, „k$pejse“ a „$pejsek“. Následně můžeme hledat slova a slovní spojení s předponou pe- a příponou -ek hledáním permutermínů s předponou ek$pe-. Dále můžeme hledat slova a slovní spojení obsahující text „ejs“ hledáním permutermínů s předponou ejs-.

Vzhledem k tomu, že jsme náš zapomnětlivý jazykový model odvodili od trií, mohlo by se zdát, že permutermíny budeme moci využít i v něm. Pokud ale do modelu vložíme místo jmen permutermíny, všimneme si několika nežádoucích jevů: ačkoliv náš jazykový model začne generovat permutermín s předponou ek$pe-,

nic mu nebrání v tom, aby v permutermínu vygeneroval několik metasymbolů $ a přestal generovat v bodě, který není pro požadovanou koncovku -ek vhodný:

```
local function add_permuterms(model_name, name)
  randomnames.add_name(model_name, name .. "$")
  for position, code in utf8.codes(name) do
    local character = utf8.char(code)
    local prefix = name:sub(1, position + #character - 1)
    local suffix = name:sub(position + #character, -1)
    randomnames.add_name(model_name, suffix .. "$" .. prefix)
  end
end

randomnames.new_model("perm", 1, 3, 17)  -- Vytvoř model.
add_permuterms("perm", "pes filipes")    -- Zanes do modelu
add_permuterms("perm", "pejsek")         -- permutermíny
add_permuterms("perm", "maxipes fík")    -- pohádkových psů.
local name = randomnames.                -- Vygeneruj permutermín.
  take_random_walk("perm", "forward", "ek$pe")
tex.print(name)
```

ek$pes fík$maxi

Popsané problémy souvisí se zapomnětlivostí našeho jazykového modelu a nemají přímočaré řešení, tj. bez úprav našeho modelu nelze permutermíny použít. Budoucí práce by se měla zaměřit na popis a implementaci potřebných úprav.

5.2.2. Regulární výrazy

Dalším způsobem, jak omezit vygenerované jméno, jsou regulární výrazy:

```
local name = nil
while name == nil or not name:match(  -- Vygeneruj jméno s danou
    "maxipes " .. (".?"):rep(60) ..   -- předponou, příponou a
    (".?"):rep(10) .. " fík") do      -- 60 až 70 znaky uprostřed.
  name = randomnames.take_random_walk("pes")
end
tex.print(name)
```

maxipes filipes filipes filipes filipes filipes filipes filipes filipes fík

Ve výše uvedeném příkladu jsme opakovaně generovali jména, dokud jsme nenašli takové, které by odpovídalo zadanému regulárnímu výrazu. Takovéto hledání je implementačně přímočaré, ale může být výpočetně náročné, pokud hledáme nepravděpodobné jméno.

Pokud sestavíme pro zadaný regulární výraz nedeterministický konečný automat, můžeme již během náhodné procházky navštívit pouze hrany, které automat

nedostanou do neakceptujícího stavu, a přestat generovat, pouze když je automat v akceptujícím stavu. Tento přístup je efektivnější, ale ani zde se nevyhneme opakovanému generování jmen: snadno se totiž dostaneme do slepé uličky, kdy náš model není schopný vygenerovat odpovídající jméno. Pokud bychom například, podobně jako ve výše uvedeném příkladu, chtěli vygenerovat jméno pohádkového psa, ale omezili bychom ho regulárním výrazem `maxipes␣.*␣pejsek`, náš model bude generovat nekonečný textový řetězec „maxipes filipes filipes filipes …“. Automat modelu nedovolí přestat generovat, dokud nezakončí jméno slovem „pejsek“, což ale nemůže nastat, vizte Obrázek 3b na straně 7. Jelikož je náš jazykový model také (pravděpodobnostní) konečný automat, který generuje regulární jazyky, mohli bychom podobné slepé uličky detekovat algoritmem, který zkontroluje, že regulární jazyk generovaný modelem a regulární jazyk akceptovaný automatem s aktuálním stavem nastaveným jako počátečním mají neprázdný průnik. Nejefektivnější známý algoritmus má ale exponenciální časovou složitost [14], což ho pro naše potřeby činí nepraktickým. Jako praktické řešení se jeví shora omezit počet kroků modelu během náhodné procházky a počet opakovaných pokusů o náhodnou procházku.

5.3. Vyhlazování pravděpodobnosti

V sekci 4 jsme pracovali s trénovacími množinami, které často obsahovaly jen malé desítky jmen. Jazykové modely trénované na těchto množinách jmen mají omezené schopnosti generalizace a nedokážou např. vygenerovat jména se znaky, které se neobjevily v trénovací množině. U mnoha jmen však známe dodatečné informace, kterými bychom mohli jazykové modely obohatit: jména prastarých v Cthulhu mýtu H. P. Lovecrafta (vizte sekci 4.3) jsou psána v latinkové transkripci r'lyehštiny, která byla pravděpodobně inspirována psanou velštinou [15] podobně jako jména v elfské sindarinštině [16] (vizte sekci 4.2). Anglická jména kapesních příšerek Pokémon (vizte sekci 4.5) jsou často anglické složeniny, např. bulbasaur = bulb (žárovka) + dinosaur a beedrill = bee (včela) + drill (vrták) [17].

Jedním způsobem, jak zlepšit generalizační schopnosti našich modelů, je pomocí tzv. *vyhlazování pravděpodobnosti* smíšením několika modelů. Pokud máme např. model sindarinštiny M_{s} a model velštiny M_{v}, můžeme pro výpočet pravděpodobnosti dalšího znaku $S[i]$ použít lineární vyhlazování:

$$P\big(S[i] \mid k\big) = \lambda \cdot P\big(S[i] \mid k; M_{\mathrm{s}}\big) + (1 - \lambda) \cdot P\big(S[i] \mid k; M_{\mathrm{v}}\big),$$

kde $k = S[i-n, \ldots, i-1]$ je kontext posledních n znaků a λ je relativní váha sindarinštiny oproti velštině. Díky velké trénovací množině pro velštinu bude výsledný model schopný lépe generalizovat mimo trénovací množinu pro sindarinštinu a generovat kreativní nová jména.

5.4. Možné aplikace jazykových modelů v TEXu

V tomto článku jsme se zaměřili na využití jazykových modelů pro generování jmen postav, což je zajímavá, ale okrajová aplikace jazykových modelů. Vzhledem k tematickému zaměření *Zpravodaje CSTUGu* by mohlo čtenáře zajímat, jakým způsobem mohou jazykové modely pomoci nejen autorovi textu, ale také sazeči.

5.4.1. Detekce jazyka pro účely dělení slov

V naší implementaci jsme se zaměřili na generativní využití jazykových modelů, kdy pomocí náhodných procházek vytváříme nová jména postav. Pokud bychom ale naše modely místo jmen trénovali na větách a pokud bychom rozšířili naši implementaci o výpočet pravděpodobnosti $P(V; M)$ vět V v daném jazykovém modelu M, pak můžeme jazykové modely využít pro detekci jazyka.

Detekce jazyka má přímočaré využití v TEXovém algoritmu řádkového zlomu. Můžeme např. připravit TEXové makro `\foreignlanguage{`věta`}`, které rozpozná jazyk věty a vysází ji se správnými vzory dělení slov. Podobný příkaz má internacionalizační LATEXový balíček babel, u kterého ale musíme kromě věty vždy ručně uvést i jazyk. Ještě vyšší úrovně uživatelského komfortu můžeme dosáhnout, když budeme pomocí callbacku `pre_linebreak_filter` LuaTEXu [18, sekce 9.5.4] automaticky přepínat vzory dělení slov podle jazyka aktuálního odstavce.

Pokud máme např. model češtiny M_{CZ} a model slovenštiny M_{SK}, pak můžeme rozhodnout, je-li věta V v češtině, nebo ve slovenštině porovnáním pravděpodobností $P(\text{CZ} \mid V)$ a $P(\text{SK} \mid V)$. Aplikací Bayesovy věty dostáváme:

$$P(? \mid V) = \frac{P(V; M_{?}) \cdot P(?)}{P(V)}.$$

Pokud předpokládáme, že všechny jazyky jsou apriori stejně pravděpodobné, pak $P(?)/P(V)$ je jen multiplikativní konstanta a dostáváme, že věta V je v češtině právě tehdy, když $P(V; M_{\text{CZ}}) > P(V; M_{\text{SK}})$, což dokážeme snadno spočítat pouze s využitím našich dvou modelů.

V praxi rozlišovat češtinu od slovenštiny nepotřebujeme, protože oba jazyky mají v TEXu společné vzory pro dělení slov [19, 20]. Máme ale mnoho jiných jazyků J, pro které společné vzory pro dělení slov (zatím) neexistují. Při detekci jazyka bychom vždy vybrali nejpravděpodobnější jazyk $? = \arg\max_{?\in J} P(V; M_{?})$.

5.4.2. Výplňový text

Při přípravě dokumentových šablon se nám může hodit generovat výplňový text, který do šablon umístíme při přípravě ukázkových maket. Pokud budeme, stejně jako v předchozí sekci, trénovat naše modely místo jmen na větách, můžeme snadno generovat věrohodný výplňový text. Stáhněme si například 114 vět z knihy *Dášeňka čili život štěněte* spisovatele Karla Čapka [21]:

```
$ for NAME in dasenka-cili-zivot-stenete{,-{2..5}}.html; do
  wget -O- https://www.cesky-jazyk.cz/citanka/karel-capek/$NAME |
  xmllint -html -xpath '//div[@id="dbtext"]//p[2]/text()' -
  done | grep '^\w' | sed -e 's/&#13;//g' -e 's/\. /.\n/g' \
  > karel-capek-dasenka.txt
```

Následně sestavíme model vět o psech a použijeme ho k vygenerování odstavce výplňového textu:

```
\documentclass{article}
\usepackage[czech]{babel}
\usepackage{randomnames}
\newmodel[context_size=9]{veta-o-psu}
\inputnames{veta-o-psu}{karel-capek-dasenka.txt}
\begin{document}
\randomname[seed=417]{veta-o-psu}
\randomname[seed=10204]{veta-o-psu}
\randomname[seed=9335]{veta-o-psu}
\end{document}
```

Ba, lidi, takové bílé nic, do hrsti se to vešlo; ale anžto to mělo pár černých ušisek a vzadu ocásek, ocáskem se nedá chodit. Nevyhne se žádné lidské náruči; pak dostane pro útěchu špičku člověčího nosu, aby se naučila sedět a chodit a ledacos jiného důležitého pro život. A musí se olíznout jazejčkem umývala, česala a hladila, pěstovala ji, líbala a lízala, čistila a jazejčkem, a namoutě kutě, ono je to dobré.

Vygenerovaný text splňuje požadavky kladené na výplňový text: je nonsensový, ale zachovává charakter trénovacích vět do té míry, že je dobře zalomitelný algoritmem řádkového zlomu. Vzhledem k tomu, že u výplňového textu nečekáme přítomnost nových slov, může být vhodnější (a efektivnější) použít jazykový model, který negeneruje věty po znacích, ale po celých slovech.

6. Závěr

V tomto článku jsme ukázali, jak můžeme pomocí jazykových modelů automaticky generovat jména postav při tvůrčím psaní v LuaTEXu. Dále jsme vyvinuli balíček, který je přístupný laickým uživatelům LATEXu, a zveřejnili jsme ho online [6]. Nakonec jsme analyzovali možná vylepšení našeho modelu a další možné aplikace jazykových modelů v LuaTEXu. Článek názorně ukazuje vývoj LATEXového balíčku, který kombinuje kód v TEXu s kódem v jazyce Lua, a demonstruje možnosti využití jazykových modelů v TEXu.

Odkazy

1. KNUTH, Donald. 6.3: Digital Searching. In: *The Art of Computer Programming Volume 3: Sorting and Searching*. 2. vyd. Addison-Wesley, 1997, s. 492. ISBN 0-201-89685-0.
2. *Check out this transparent Huckleberry Hound hello PNG image* [online]. [cit. 2023-02-22]. Dostupné z: `https://cartoongoodies.com/png_images/huckleberry-hound-hello/`.
3. ČAPEK, Josef. *Povídání o pejskovi a kočičce: jak spolu hospodařili a ještě o všelijakých jiných věcech*. 13. vyd. Albatros, 1972.
4. *GoGEN Maxipes Fík: Česká značka elektroniky pro děti* [online]. [cit. 2023-02-22]. Dostupné z: `https://www.gogen.cz/gogen-maxipes-fik/`.
5. KATZ, S. M. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*. 1987, roč. 35, č. 3, s. 400–401.
6. NOVOTNÝ, Vít. *Nápadovník jmen postav pro tvůrčí psaní v LuaTEXu: Release The latest version* [online]. GitHub, 2023-05-03 [cit. 2023-05-03]. Dostupné z: `https://github.com/witiko/character-name-generator-for-creative-writing-in-luatex/releases/tag/latest`.
7. IERUSALIMSCHY, Roberto. *Programming in Lua*. 4. vyd. Lua.org, 2016. ISBN 978-8590379867.
8. IERUSALIMSCHY, Roberto; FIGUEIREDO, Luiz Henrique de; CELES, Waldemar. *Lua 5.3 Reference Manual* [online]. 2020-07-14. [cit. 2023-03-10]. Dostupné z: `https://www.lua.org/manual/5.3/manual.html`.
9. PRESS, W.H.; TEUKOLSKY, S.A.; VETTERLING, W.T.; FLANNERY, B.P. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3. vyd. Cambridge University Press, 2007. ISBN 978-0-521-88068-8.
10. L'ECUYER, Pierre. Tables of linear congruential generators of different sizes and good lattice structure. *Mathematics of Computation*. 1999, roč. 68, č. 225, s. 249–260.
11. THE LATEX PROJECT. *The LATEX3 Interfaces* [online]. CTAN, 2023-03-30 [cit. 2023-04-09]. Dostupné z: `http://mirrors.ctan.org/macros/latex/contrib/l3kernel/interface3.pdf`.
12. VRABCOVÁ, Tereza. *Model pro automatické skloňování českých jmen v LuaTEXu: Release The latest version* [online]. GitHub, 2023-05-03 [cit. 2023-05-03]. Dostupné z: `https://github.com/xvrabcov/declension_names/releases/tag/latest`.
13. MANNING, Christopher D.; RAGHAVAN, Prabhakar; SCHÜTZE, Hinrich. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN 0521865719. Dostupné také z: `https://nlp.stanford.edu/IR-book/`.

14. RABIN, Michael O; SCOTT, Dana. Finite automata and their decision problems. *IBM J. Res. Dev.* 1959, roč. 3, č. 2, s. 114–125.
15. NOVOTNÝ, Vít; STARÁ, Marie. Cthulhu Hails from Wales: *N*-gram Frequency Analysis of R'lyehian. In: HORÁK, Aleš; RYCHLÝ, Pavel; RAMBOUSEK, Adam (ed.). *Recent Advances in Slavonic Natural Language Processing*. Tribun EU, 2020. ISBN 978-80-263-1600-8. ISSN 2336-4289. Dostupné také z: `https://nlp.fi.muni.cz/raslan/2020/paper12.pdf`.
16. HOOKER, Mark T. *Tolkien and Welsh (Tolkien a Chymraeg): Essays on J. R. R. Tolkien's Use of Welsh in his Legendarium*. 1. vyd. Llyfrawr, 2012. ISBN 978-1477667736.
17. CREATIVEBLOGGER. *Pokémon in Translation: Where Do Their Names Come From?* [online]. 2017-08-11. [cit. 2023-04-17]. Dostupné z: `https://creativetranslation.com/blog-pokemon-names-in-translation/`.
18. LUATEX DEVELOPMENT TEAM. *LuaTEX Reference Manual* [online]. CTAN, 2023-04-06 [cit. 2023-04-17]. Dostupné z: `https://ctan.org/pkg/luatex`. Verze 1.16.
19. SOJKA, Petr; SOJKA, Ondřej. The Unreasonable Effectiveness of Pattern Generation. *Zpravodaj CSTUGu*. 2019, roč. 29, č. 1–4, s. 73–86. ISSN 1211-6661. Dostupné z DOI: `10.5300/2019-1-4/73`.
20. SOJKA, Petr; SOJKA, Ondřej. Towards New Czechoslovak Hyphenation Patterns. *Zpravodaj CSTUGu*. 2020, roč. 30, č. 3–4, s. 118–126. ISSN 1211-6661. Dostupné z DOI: `10.5300/2020-3-4/118`.
21. ČAPEK, Karel. *Dášeňka čili život štěněte*. 1. vyd. Fr. Borový, 1933.

Summary: Character Name Generator for Creative Writing in LuaTEX

A famous dictum of the computer scientist Phil Karlton says that there are only two difficult things in computer science: cache invalidation and naming things. This is also true in creative writing, where authors have to come up not just with a story and a setting but also the names of all their fictional characters. In this article, we develop a language model in LuaTEX, which allows authors to automatically generate names for their characters. Besides creative writing, we also discuss other uses of language models in LuaTEX, namely the automatic switching of hyphenation patterns based on the current language and blind text generation. For the TEXnically-minded users, the article acts as an introduction to the programming languages of Lua and expl3, and also the xparse LATEX package for defining document commands in LATEX.

Keywords: creative writing, trie, language models, LuaTEX, Lua, expl3, xparse

Vít Starý Novotný, witiko@mail.muni.cz

Sazba hudebních skladeb

Karel Šebela

TEX je užitečný nástroj pro sazbu textu, ale nemá v základu dobrou podporu pro sazbu skladeb. Na konferenci TUG 2022 zazněla přednáška o sazbě not, která porovnávala sazbu pomocí balíku *MusiXTEX* s nástroji *Musescore* a *Flat*, ale nezmínila preprocesory *PMX* a *M-Tx*, které sazbu zjednodušují. V tomto článku porovnám sazbu nástrojem *MusiXTEX* s jeho preprocesory a popíšu jejich užití. Dále rozeberu začlenění notových značek do textu odstavce. Po přečtení článku bude čtenář schopný vytvořit jednoduchý krátký úryvek hudebního díla a začlenit jej do TEXového dokumentu, či doplnit psaný text notovými značkami.

Klíčová slova: notový zápis, *MusiXTEX*, *PMX*, *M-Tx*, *LilyPond*, *MuseScore*, *Flat*

1. Úvod

V tomto článku navazuji na prezentaci *Musical Composition Typesetting* představenou na konferenci TUG 2022 [1]. Samostatná prezentace sice zmiňovala některé nedostatky sazby konvenčními programy, ale celkově byla příliš stručná a mnohé principy vůbec nerozváděla. Rád bych v tomto článku prozkoumal možnosti, které TEX v rámci sazby hudby umožňuje, a následně prodiskutoval jejich výhody a omezení.

Součástí tvorby hudebních děl je jejich zápis podle určitého systému. Takřka výlučně se dnes používá způsob zápisu formou not do notových osnov s pěti linkami. Tento způsob vychází již z počátku 11. století a přes svůj historický vývoj, například v počtu linek nebo trámcích mezi notami, se jeho podstata zásadně nezměnila [2].

Dříve manuální úpravy v současnosti usnadňují specializované editory. Na typografické úrovni řeší správné umístění not a rozestupů, sklony trámců, legat a dalších linek. Mimo jiné také zajišťují, že počet dob v každém z taktů souhlasí s uvedeným taktovým označením. Většina populárních programů (*MuseScore*, *Sibelius*, *Finale*, …) umožňuje jednotlivé noty vkládat interaktivně přes WYSIWYG rozhraní. Jiný přístup využívá *LilyPond*, který, podobně jako TEX, výsledné dokumenty sestavuje ze zdrojového kódu.

V sekci 2 shrnuji prezentaci Musical Composition Typesetting [1] z TUGu 2022 a popisuji nedostatky představených nástrojů *MuseScore* a *Flat*. Následně v sekci 3 představuji sazbu pomocí balíku maker *MusiXTEX* a následně v sekci 4

DOI: 10.5300/2023-1-2/39

představuji preprocesory, které zápis v *MusiXTEX*u usnadňují. V sekci 5 představuji doplňkové LaTeXové balíky, pomocí kterých lze sázet notové symboly v psaném textu. Článek uzavírám v sekci 6 popisem nedostatků *MusiXTEX*u a souvisejících preprocesorů a krátkým představením dávkového nástroje *LilyPond*, který je na TeXu nezávislý a netrpí uvedenými nedostatky.

2. Motivace

Původní prezentace [1] připomínkuje nedostatky interaktivních editorů *MuseScore* a *Flat*. *MuseScore* je desktopová aplikace s otevřeným zdrojovým kódem, *Flat* je webová aplikace dostupná na doméně `flat.io`. Prezentace nezmiňuje, ve které aplikaci se vyskytují které problémy, připomínky jsem tak ověřoval v obou programech: v *MuseScore* verze 4.0.1 a ve zdarma dostupné edici *Flat*. Většinu chyb zmiňovaných v prezentaci se mi podařilo reprodukovat pouze v aplikaci *Flat*.

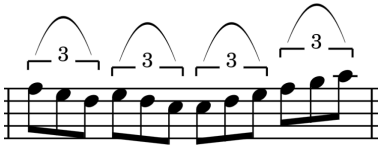
2.1. Nedostatky *MuseScore* a *Flat*

Připomínky k problémům výstupu sazby se týkají několika různých oblastí:

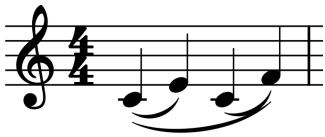
Nastavení rozestupů not Prezentace připomínkuje nemožnost volby vlastních rozestupů mezi jednotlivými notami. *Flat* nastavení rozestupů sice umožňuje, ale pouze pro celou skladbu a pouze v placené verzi [3]. Tyto rozestupy zůstávají stejné i po přidání textu písně, díky čemuž se jednotlivé slabiky překrývají. *MuseScore* těmito nedostatky netrpí.

Sazba legat *Flat* u not s trámci sází nadbytečné triolové svorky a legata nad nimi nereflektují informaci o výšce jednotlivých not. Výsledkem jsou nevzhledná legata s příliš prohnutým obloukem, uvedená na obrázku 1.

Dále jsou omezeny možnosti sazby vnořených legat, používaných u skladeb pro smyčcové nástroje. *Flat* umožňuje noty zároveň svázat pouze legatem a ligaturou. *MuseScore* umožňuje spojit legatem libovolné dvě noty a většinou i správně brání jejich kolizi. Případ lehké kolize je uveden na obrázku 2.



Obrázek 1: Nevzhledná legata nad triolami v programu *Flat*. *MuseScore* sází trioly pod trámci bez svorek.



Obrázek 2: Kolize dvou legat v *MuseScore*. Legata se v tomto případě lehce dotýkají, nikdy se ale nepřekrývají.

Export části skladby *MuseScore* ani *Flat* neumožňují exportovat výběr dokumentu do grafického formátu [4]. Nelze tedy například vzít část skladby a zahrnout ji jako obrázek do jiného dokumentu; stránku je třeba ručně oříznout grafickým editorem.

3. Balík *MusiXTEX*

MusiXTEX obsahuje sadu maker a písem umožňujících sazbu hudby v systému TEX. Zahrnuje symboly osnov, not, trámců, linek a dalších značek. *MusiXTEX* na základě posloupnosti TEXových maker určuje umístění jednotlivých prvků v rámci partitury. Na rozdíl od sazby lineárního textu umožňuje práci s několika notovými osnovami současně, což mimo jiné komplikuje řešení velikosti mezer mezi notami.

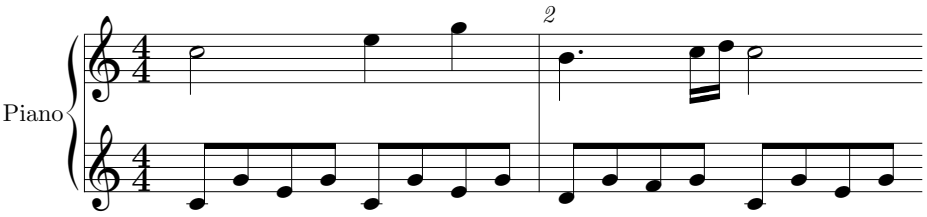
MusiXTEX kvůli řešení zalamování řádků zpracovává vstupní soubory třemi průchody. V prvním průchodu zpracuje vstupní soubor příkazem `etex` a poznačí si rozestupy taktů do souboru s příponou `.mx1`. Ve druhém průchodu procesor `musixflx` řeší optimální velikost rozestupů not a výstup zapisuje do souboru s příponou `.mx2`. Ten je následně zahrnut ve finálním průchodu `etex`em.

Zdrojový kód pro zahrnutí do LATEXového dokumentu, který po sestavení zobrazuje úryvek Mozartovy Sonáty C-dur, vypadá následovně [5]:

```
\begin{music}
\parindent10mm
\instrumentnumber{1}        % 1 nástroj
\setname1{Piano}            % pojmenovaný Piano
\setstaffs1{2}              % se 2 notovými osnovami
\generalmeter{\meterfrac44} % 4/4 taktové označení
\startextract               % začátek úryvku
\Notes\ibu0f0\qb0{cge}\tbu0\qb0g|\hl j\en
\Notes\ibu0f0\qb0{cge}\tbu0\qb0g|\ql l\sk\ql n\en
\bar
\Notes\ibu0f0\qb0{dgf}|\qlp i\en
\notes\tbu0\qb0g|\ibbl1j3\qb1j\tbl1\qb1k\en
\Notes\ibu0f0\qb0{cge}\tbu0\qb0g|\hl j\en
\zendextract               % konec úryvku
\end{music}
```

V úryvku mezi příkazy `\startextract` a `\zendextract` se jednotlivé noty uvádí po řádcích mezi makry `\Notes` a `\en`. Nejprve je uvedena část not (ne nutně celý takt) pro spodní hlas (levou ruku), poté svislice `|` a následně odpovídající noty pro horní hlas (pravou ruku).

Vysázené dílo poté vypadá takto:



4. Preprocesory pro *MusiXTEX*

Popis not *MusiXTEX*ovými makry je poměrně zdlouhavý a u rozsáhlejších skladeb poněkud nečitelný. Namísto přímého psaní maker lze notový vstup psát ve zkráceném formátu a následně jej přeložit na prostý TEXový soubor obsahující jednotlivá *MusiXTEX*ová makra. Konkrétně lze k tomu využít jeden z preprocesorů, buď preprocesor *PMX* nebo z něj vycházející procesor *M-Tx*.

4.1. Preprocesor *PMX*

Vstup preprocesoru *PMX* se zapisuje do odděleného souboru s příponou `.pmx`. TEXový výstup lze generovat příkazem `pmxab`, snadnější je však zdroje sestavovat obalujícím luaskriptem `musixtex`, který ve výchozím chování dokument přímo vysází do formátu `.pdf`.

Výslednou partituru ve formátu `ps` nebo `pdf` pak lze použít samostatně nebo zahrnout do jiného dokumentu LATEXovým příkazem `\includegraphics` jako vektorovou grafiku. Protože má však vysázená skladba rozměry celé strany, je žádoucí ji před zahrnutím oříznout – buď ruční specifikací přes možnost `trim`, nebo automaticky například externím nástrojem `pdfcrop`.

Preambule vstupního souboru `.pmx` je velmi minimální. Začíná posloupností 12 čísel, které určují parametry celé partitury. Popisují například počet notových osnov, nástrojů, definice taktových označení nebo počet stran. Následuje seznam jmen nástrojů a určení klíčů pro jednotlivé osnovy. Poslední řádek preambule tvoří adresářová cesta, do které má být zapsán TEXový výstup [6]. Celá preambule včetně komentářů může vypadat například takto:

```
% počet osnov, nástrojů, 4/4 takt, označený C, délka předtaktí,
       2          1          4 4        0 6           0
% předznamenání, počet stran, řádků, velikost, předsazení
        0             1          1       20        0.07
Piano
```

```
bt
./
```

Tělo vstupního souboru *PMX* poté určuje výšky a délky jednotlivých not. Noty se zapisují písmeny `a-g`, následovanými číslicemi určujícími délku (`0` 𝅝, `2` 𝅗𝅥, `4` ♩, `8` ♪, `1` 𝅘𝅥𝅯, `3` 𝅘𝅥𝅰) a oktávu. Oddělení taktů svislou čarou není povinné, ale usnadňuje kontrolu chyb. Dalšími příkazy lze přidávat posuvky (`s` ♯, `f` ♭, `n` ♮), dynamiku a další značky. Oproti balíku *MusixTEX* je většina příkazů zkrácena na jeden znak, což zmíněný úryvek Sonáty C-dur zkracuje na pouhé dva řádky (zarovnání mezerami zde nehraje roli):

```
c83 g83 e g c83 g83 e  g | c83 g83 f g     c83 g83 e g /
c25         e4      g4   | bd-       c1 d1 c2          /
```

Délku not i oktávu si preprocesor udržuje podle předchozí noty, lze je tedy místy vynechat. U větších intervalových skoků je nutné uvést konkrétní oktávu absolutně číslem nebo relativně symboly `+`/`-`. Poněkud neintuitivní je u *PMX* zápis hlasů v obráceném pořadí – levý hlas klavíru je v tomto příkladu popsán horním řádkem.

4.2. Preprocesor *M-Tx*

Ve starších verzích procesor *PMX* nepodporoval sazbu textů písní. V současné verzi lze texty písní v *PMX* psát v uvozovkách, které jsou poté obaleny do TEXového makra `pmxlyr`, dále rozvíjeného na *MusiXTEX* ové příkazy.

Alternativní rozhraní pro sazbu textů nabízí preprocesor *M-Tx* (z anglického „Music from Text“) [7]. Formát souboru *M-Tx* je založen na formátu *PMX*; podstatněji se liší ve formátu preambule, kde místo seznamu hodnot používá seznam příkazů ve formátu `Klíč: hodnota`. Vstupní soubory se zpracovávají luaskriptem `musixtex`, který postupně volá jednotlivé procesory. Soubor je nejprve přeložen do formátu `.pmx`, poté zpracován preprocesorem *PMX* a vysázen TEXem. Oproti *PMX* se zde jednotlivé osnovy zapisují v přirozeném vertikálním pořadí.

S ohledem na český text mi přišlo žádoucí zkusit sazbu českých písní včetně diakritiky. Bohužel, písmena s diakritikou nelze do *MusiXTEX*ových zdrojových souborů jednoduše zadávat. Částečné řešení, které umožňuje psát text s diakritikou v UTF-8, jsem přejal z dokumentace *MusiXTEX*u [5, sekce 23.4]. Spočívá ve změně kategorie písmen s diakritikou a následném překladu LuaTEXovým backendem. Takto lze text psát přirozenou češtinou:

```
%%\catcode`\á\active \defá{\'{a}}
%%\catcode`\č\active \defč{\v{c}}
%%\catcode`\ě\active \defě{\v{e}}
%%\catcode`\ř\active \defř{\v{r}}
```

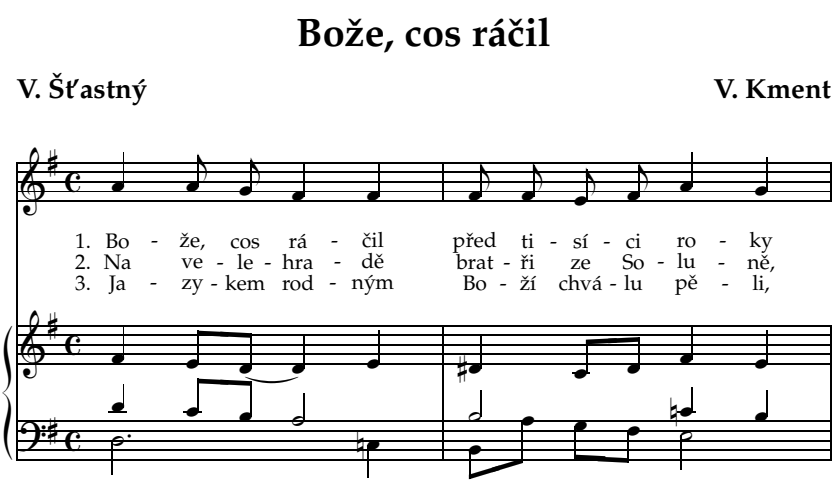
```
%%\catcode`\í\active \defí{\'{i}}
%%\catcode`\Š\active \defŠ{\v{S}}
%%\catcode`\ť\active \defť{\v{t}}
%%\catcode`\ž\active \defž{\v{z}}
Title: Bože, cos ráčil
Composer: V. Kment
Poet: V. Šťastný
Sharps: 1
Space: 7
Style: Singer Organ
Organ: Voices RH LH,Ped; Clefs G F; Continuo
Width: 150mm

a a8 g  f4 f             | f8 f e f      a4 g   |
L: 1. Bo-že, cos rá-čil    před ti-sí-ci ro-ky
L: 2. Na ve-le-hra-dě      brat-ři ze So-lu-ně,
L: 3. Ja-zy-kem rod-ným    Bo-ží chvá-lu pě-li,
f  e8 ( d  d4 ) e        | ds        c8 d f4  e |
d+ c8    b  a2           | b2             c4n b |
d2d         c4n          | [ b8 a+ ] g8 f e2    |
```

LuaTEXový backend lze zvolit zadáním příslušného argumentu:

```
musixtex -F "luatex --output-format=pdf" soubor.mtx
```

Vysázená skladba poté vypadá následovně:



5. Notové značky uvnitř textu

Mimo sazbu not do notových osnov lze i psaný text doplnit notovými symboly, například pro ilustraci hudební teorie. Symboly, které používá *MusiXTEX*, mají při použití v textu nesprávnou velikost a nelze je tak do textu jednoduše začlenit.

Umístění hudebních značek do odstavců nabízí balíky `musicography` a `lilyglyphs`. Balík `musicography` má omezenější výběr, navíc jeho makra pro taktová označení kolidují s těmi *MusiXTEX*ovými.

Balík `lilyglyphs` využívá k sazbě notových symbolů glyfy písma Emmentaler z programu *LilyPond*. K vyhledání jednotlivých glyfů přitom závisí na balíku `fontspec`, díky čemuž vyžaduje k překladu LuaLaTeX. Pro porovnání vzhledu notových značek z obou balíků vizte tabulku 1.

Noty 𝅝 𝅗𝅥 ♩ ♪ 𝅘𝅥𝅯,	posuvky ♯ ♭ ♮,	taktová označení $\frac{3}{4}$ $\frac{4}{4}$ $\frac{6}{8}$ 𝄴 𝄵.
Noty 𝅝 𝅗𝅥 ♩ ♪ 𝅘𝅥𝅯,	posuvky ♯ ♭ ♮,	taktová označení $\frac{3}{4}$ $\frac{4}{4}$ $\frac{6}{8}$ 𝄴 𝄵.

Tabulka 1: Porovnání vzhledu a rozestupu některých symbolů z balíků `musicography` (nahoře) a `lilyglyphs` (dole) při oddělování mezerami `\thinspace = \,`. Vysázené prvky z jednotlivých balíků mají odlišné mezery, taktová označení `musicography` mají pouze základní vzhled.

Zahrnutí balíku `lilyglyphs` konkrétně u šablony `csbulletin` zahlásí chybu při duplicitní definici `centerbox`u. Jako workaround lze makro `centerbox` před načtením balíku `lilyglyphs` oddefinovat příkazem `\let\centerbox\relax`. Zároveň je u této šablony první z taktových označení `lilyglyphs` mírně nezarovnané, konkrétní příčinu jsem již ale nedohledal.

6. Nedostatky TEXových systémů

V porovnání s konvenčními WYSIWYG editory jsou současné TEXové systémy poměrně obtížné. Začínající uživatele může odradit nedostatek zdrojů, často omezený na oficiální dokumentace, které jsou často příliš dlouhé nebo neobsahují dostatek názorných příkladů. Názornější mohou být příklady skladeb a dodatečné příručky dostupné z webu *Werner Icking Music Archive* [8], konkrétně například návod *Typesetting Music With PMX* [9].

Dalším nedostatkem TEXových systémů a jejich preprocesorů jsou místy nečitelné chybové zprávy. Jako příklad uvádím zprávu, která může uživatele zaskočit při zpracování souboru *PMX* s nesprávným formátem hlavičky. Hlášení v tomto případě nevypisuje kód *PMX*, ale přímo formátovací funkce Fortranu, ve kterém je *PMX* implementován [6]. Každopádně hledání příčiny problému příliš neusnadňuje:

```
fmt: read unexpected character
apparent state: internal I/O
last format: (f1.0)
lately reading sequential formatted internal IO
```

Mimoto je také možné, že samotná podstata textové sazby nemusí být pro řadu uživatelů zcela intuitivní, což může bránit širšímu rozšíření těchto nástrojů.

6.1. Program LilyPond

Alternativu k TEXovým makro balíkům může představovat program *LilyPond*. Na rozdíl od představených procesorů není vázán na TEX a existují k němu grafické editory, například *Frescobaldi* či *Denemo*. Zatímco Frescobaldi ze zadaného zdrojového kódu generuje náhled skladby, Denemo umožňuje naopak noty vkládat interaktivně a zpětně generuje kód pro LilyPond [10].

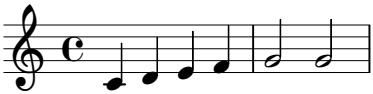
Kód LilyPondu lze zahrnout i do TEXových souborů, konkrétně buďto inline prostřednictvím makra `lilypond` nebo častěji přes stejnojmenné prostředí:

```
\begin{lilypond}[quote,fragment,staffsize=26]
  c' d' e' f' g'2 g'2
\end{lilypond}
```

Pro překlad dokumentu lze použít makrobalík *LyLuaTEX* krátce představený v minulém čísle Zpravodaje [11, sekce 5]. Po zahrnutí příkazem `\usepackage{lyluatex}` lze sázet dokument přímo příkazem `lualatex`:

```
lualatex --shell-escape document.tex
```

Vysázený úryvek poté vypadá takto. Pokud je cílem uvést úryvek do jiného dokumentu (jako zde) a využívá se pro ořez `pdfcrop`, je potřeba vypnout LATEXové číslování stran příkazem `\pagenumbering{gobble}`.



Odkazy

1. PARK, Christopher; PARK, Emily. *Musical composition typesetting* [online]. 2022-07-24. [cit. 2022-11-12]. Dostupné z: `https://youtu.be/bwbk2xiMLSg`.
2. PATERSON, Jim. *A Short History of Musical Notation* [online]. [cit. 2023-03-08]. Dostupné z: `https://www.mfiles.co.uk/music-notation-history.htm`.

3. TUTTEO LIMITED. *Flat: Number of measures per system and notes spacing* [online]. [cit. 2023-03-13]. Dostupné z: https://flat.io/help/en/music-notation-software/layout-measures.
4. WEISS, Isaac; SABATELLA, Marc. *Export Selection File menu option* [online]. 2014-12-26. [cit. 2022-11-21]. Dostupné z: https://musescore.org/en/node/42336. Feature request.
5. VOGEL, Oliver et al. *MusixTEX: Using TEX to write polyphonic or instrumental music* [online]. 2021-08-29. Ver. 1.35 [cit. 2022-11-21]. Dostupné z Package documentation na https://ctan.org/pkg/musixtex.
6. SIMONS, Don. *PMX: a Preprocessor for MusiXTEX* [online]. 2022-02-04. Ver. 2.98 [cit. 2022-11-22]. Dostupné z Package manual na https://ctan.org/pkg/pmx.
7. LAURIE, Dirk. *M-Tx: Music from Text* [online]. 2019-01-15. Ver. 0.63c [cit. 2022-11-21]. Dostupné z Package documentation na https://ctan.org/pkg/m-tx.
8. *MusiXTEX and related software: Introduction* [online]. [cit. 2023-03-13]. Dostupné z: https://www.icking-music-archive.org/software/htdocs/Introduction.html.
9. NOACK, Cornelius C. *Typesetting music with PMX* [online]. 2013-05. Ver. 2.821 [cit. 2023-03-13]. Dostupné z: http://icking-music-archive.org/software/pmx/pmxccn.pdf.
10. *Front-end Applications* [online]. [cit. 2023-03-13]. Dostupné z: https://lilypond.org/easier-editing.html.
11. NOVOTNÝ, Vít. Vysokoúrovňové jazyky pro TEX. *Zpravodaj CSTUGu*. 2022, roč. 32, č. 1–4, s. 35–48. Dostupné z DOI: 10.5300/2022-1-4/35.

Summary: Musical Composition Typesetting

TEX is a useful tool for text typesetting; however, it doesn't feature a good support for composition typesetting at its core. The TUG 2022 conference featured a talk on notation typesetting which compared the *MusiXTEX* package with the *MuseScore* and *Flat* tools, but did not mention the *PMX* and *M-Tx* preprocessors. In this article, I compare typesetting using *MusiXTEX* and its preprocessors and describe its usage. In addition, I describe the incorporation of note symbols into a paragraph text. After reading the article, the reader will be able to create a short simple excerpt of a piece of music and incorporate it into a TEX document, as well as add musical symbols to a written text.

Keywords: sheet music, music engraving, *MusiXTEX*, *PMX*, *M-Tx*, *LilyPond*, *MuseScore*, *Flat*

Karel Šebela, 514509@mail.muni.cz

Pohľad TEXového nováčika na prezentáciu „Bricks and Jigsaw Pieces“ z TUGu 2022

Matúš Vančík

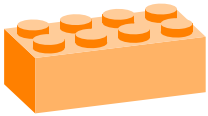
V tomto článku sa bližšie pozrieme na dva nové LATEXové balíčky, *TikZbricks* a *jigsaw*. Ako ešte len čerstvý používateľ TEXu by som chcel pomocou vlastných experimentov preskúmať dizajn, jednoduchosť práce s balíčkami ako aj možnosti uplatnenia. Nakoniec by som pomocou tohto článku chcel vyzdvihnúť, prečo sú takéto balíčky v TEXu potrebné.

Kľúčové slová: *TikZ*, *TikZbricks*, *jigsaw*

Úvod

Pri pozeraní prezentácií z každoročnej konferencie používateľov TEXu (TUG 2022), ma hneď zaujala prezentácia balíkov *TikZbricks* a *jigsaw* od Sam Carterovej [1]. Tieto balíčky boli pre mňa, nového používateľa TEXu, veľkým oživením inak monotónneho LATEXu. Do tej chvíle som si nevedel predstaviť, že niečo takéto kreatívneho je možné v LATEXu vytvoriť. Vedel som o rôznych možnostiach na vytvorenie grafov a vkladaní fotografií do dokumentov, ale táto možnosť, vytvárať niečoho kreatívne a nové pomocou LATEXových príkazov, bola pre mňa doposiaľ neznáma. Zároveň, „čisto“ náhodná podoba kociek balíčku *TikZbricks* s kockami Lego ma vtiahla naspäť do mojich detských čias. Mohol som sa hrať s puzzle a legom v TEXu!

Zaujímalo ma, či je možné praktické použitie týchto balíčkov. Je možné použiť ich v akademickej sfére pri písaní vedeckých článkov alebo si nájdu svoje uplatnenie ako prostriedky na vzdelávanie nových používateľov TEXu? Je takýto balík vôbec potrebný vo svete TEXu? To boli len niektoré otázky, ktoré mi prebehli hlavou pri pozeraní prezentácie Sam Carterovej. Pri hľadaní odpovedí na moje otázky som sa dostal na stránku CTAN, táto skratka odkazuje na zložitú sieť archívov TEXu. Na tejto stránke som sa po prvýkrát ponoril do rôznych zaujímavých a vizuálne kreatívnych balíčkov TEXu. Preto odporúčam každému novému používateľovi TEXu navštíviť túto stránku a konkrétne by som odporúčal laické preskúmanie balíčku *TikZ*, do ktorého patria taktiež balíky *TikZbricks* a *jigsaw*.

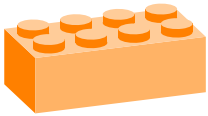


doi: 10.5300/2023-1-2/48

Stav balíkov *TikZbricks* a *jigsaw*

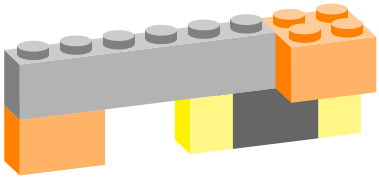
Počas písania tohto článku bol balík *TikZbricks* dostupný vo verzii 0.4. V tejto verzii ide o malý LaTeXový balíček na kreslenie Lego kociek pomocou *TikZ*. Používateľ môže ľubovoľne meniť farbu, tvar, veľkosť a uhol pohľadu jednotlivých kociek. Taktiež môže zobrazovať len jednu kocku alebo vytvoriť celú stenu z viacerých jednotlivých kociek. Stena sa stavia smerom zdola nahor a sprava doľava, inými slovami, začína sa z pravého dolného rohu. Balík obsahuje aj skript `img2bricks` od Scotta Pakina [2], ktorý umožňuje zaradiť jednoduché PNG obrázky do štruktúr *TikZbricks*, čo hlavne ocenia používatelia, ktorí by chceli z kociek vytvoriť obrázok na základe nejakej predlohy.

Jednu kocku je možné vytvoriť pomocou príkazu `\brick`, pričom prvý argument špecifikuje dĺžku kocky a druhý argument ovplyvňuje šírku kocky. Taktiež je možné nastaviť farbu kocky pomocou argumentu `color` a celková veľkosť kocky sa dá nastaviť pomocou argumentu `scale`.



```
\begin{tikzpicture}
\brick[color=orange,scale=0.6]{4}{2}
\end{tikzpicture}
```

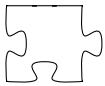
Stenu z kociek môžeme vytvoriť pomocou prostredia `wall` a príkazu `\wallbrick`, ktorým je možné stavať jednotlivé kocky vedľa seba. Ak chceme v stene vytvoriť dieru, ktorá môže slúžiť ako pomyslné dvere, používame príkaz `\addtocounter` s argumentmi `brickx`, `bricky` a `brickz`, ktorými určujeme smer diery.



```
\begin{wall}[scale=0.5]
\wallbrick[color=yellow]{1}{1}
\wallbrick[color=black]{2}{1}
\wallbrick[color=yellow]{1}{1}
\addtocounter{brickx}{2}
\wallbrick[color=orange]{2}{1}
\newrow
\wallbrick[color=orange]{2}{2}
\wallbrick[color=gray]{6}{1}
\end{wall}
```

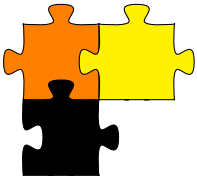
Balíček *jigsaw* sa počas písania tohto článku nachádzal vo verzii 0.3. Ide o malý LaTeXový balík na kreslenie dielikov skladačky puzzle pomocou *TikZ*. Pomocou tohto balíčku je možné kresliť jednotlivé kúsky, pričom sme schopní upravovať ich tvar, vytvárať rôzne vzory jednotlivých kusov skladačky alebo vygenerovať celú skladačku. Následne sa dá na takto vygenerovanú skladačku vložiť obrázok.

Samostatný kúsok puzzle je možné vytvoriť pomocou príkazu `\piece`. Príkaz má štyri premenné ⟨*spodná*⟩, ⟨*pravá*⟩, ⟨*vrchná*⟩ a ⟨*lavá*⟩, ktoré môžu nadobúdať tieto hodnoty: 0 = hrana, −1 = výbežok, 1 = priehlbina.



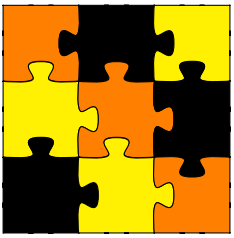
```
\begin{tikzpicture}
\piece{1}{-1}{0}{1}
\end{tikzpicture}
```

Pomocou prostredia `scope` môžeme manuálne vytvoriť celé puzzle. Jedno prostredie `scope` reprezentuje jeden kúsok skladačky a pomocou argumentov ako napríklad `xshift` sa vieme posunúť na miesto, kde chceme vytvoriť ďalší kúsok. Taktiež môžeme použiť voliteľný argument na zmenu farby jednotlivých puzzlí.



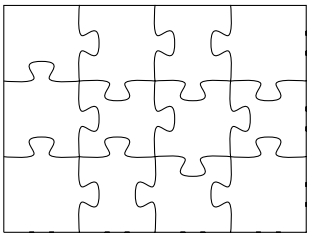
```
\begin{tikzpicture}
\begin{scope}
\piece[orange]{0}{0}{-1}{-1}
\end{scope}
\begin{scope}[xshift=1cm]
\piece[yellow]{0}{-1}{-1}{-1}
\end{scope}
\begin{scope}[yshift=-1cm]
\piece[black]{0}{-1}{-1}{1}
\end{scope}
\end{tikzpicture}
```

Tento spôsob však môže byť zdĺhavý a chybový. Preto ja osobne preferujem príkaz `\tile[`⟨*farba*⟩`]{`⟨*spodná*⟩`}{`⟨*pravá*⟩`}{`⟨*vrchná*⟩`}{`⟨*ľavá*⟩`}`, ktorý môže byť použitý aj mimo prostredia `tikzpicture`. Pomocou tohto príkazu môžeme jednotlivé riadky skladačky vyskladať.



```
\tile[orange]{1}{1}{0}{0}%
\tile[black]{1}{-1}{0}{-1}%
\tile[yellow]{1}{0}{0}{1}
\tile[yellow]{1}{-1}{-1}{0}%
\tile[orange]{1}{-1}{-1}{1}%
\tile[black]{-1}{0}{-1}{1}
\tile[black]{0}{-1}{-1}{0}%
\tile[yellow]{0}{-1}{-1}{1}%
\tile[orange]{0}{0}{1}{1}
```

Ak chceme automaticky vygenerovať celú skladačku, stačí nám použiť len príkaz `\jigsaw{`⟨*šírka*⟩`}{`⟨*výška*⟩`}`. Pomocou čísel zvolíme počet jednotlivých dielikov, ktoré budú určovať šírku a výšku výsledného puzzle.



```
\begin{tikzpicture}
\jigsaw{4}{3}
\end{tikzpicture}
```

O ďalších možnostiach týchto balíčkov, ktoré som nespomenul vyššie, si môžete prečítať v dokumentáciách napísaných Sam Carterovou [3–6].

Experimentovanie

Ako bolo možné vidieť v predchádzajúcej časti článku, práca s balíčkami je veľmi jednoduchá a intuitívna. V tejto sekcii s balíkmi *TikZbricks* a *jigsaw* experimentujem s cieľom vytvoriť zložitejšie obrázky.

Balíček *TikZbricks*

Medzi najzaujímavejšie experimenty z tohto balíčku považujem skript `img2bricks`, ktorý sa nachádza v githubovom repozitári *TikZbricks* [2]. Ide o pythonový script na premenu PNG obrázkov na TEXové príkazy, ktoré replikujú daný PNG obrázok pomocou balíka *TikZbricks*. Na spustenie scriptu je potrebné mať pythonový balíček Pillow, ktorý nainštalujeme príkazom `pip install Pillow`. Príkazom `./img2bricks` ⟨*PNG obrázok*⟩ `-o obrazok.tex` vznikne samostatný LATEXový dokument *obrazok.tex*. Tento dokument môže následne používateľ preložiť LATEXom, vďaka čomu získa obrázok vo formáte PDF. Alternatívne je možné skopírovať kód v prostredí `wall` a vložiť ho do svojho vlastného LATEXového dokumentu.

Prekážkou bolo nájsť vhodné rozmery vstupných obrázkov, ktoré by so scriptom pracovali najlepšie. Pomocou experimetov som zistil, že najvhodnejšie obrázky majú rozmer 30×30 pixelov. Z takýchto obrázkov bolo možné vygenerovať detailné TEXové príkazy. Pri väčších obrázkoch vznikne kód, ktorý sa neoplatí prenášať do TEXu, pretože vytvorený obrázok bude až moc veľký na to, aby bol použiteľný v bežnom dokumente. Na obrázku 1 vidíte obrázok stromu, ktorý som vygeneroval príkazom `img2bricks` počas svojich experimentov.

Obr. 1: PNG obrázok (vľavo), z ktorého som generoval skriptom `img2bricks` TEXové príkazy balíka *TikZbricks* (vpravo).

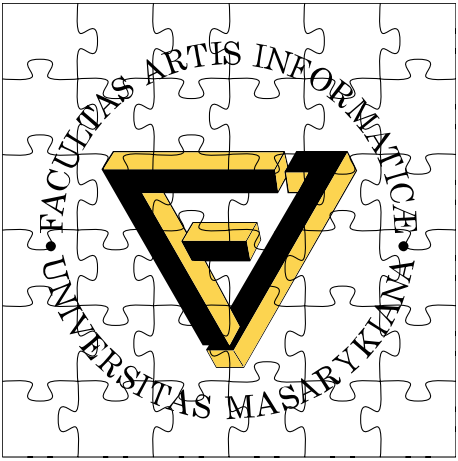
Balíček *jigsaw*
Experimentovanie pomocou balíčku *jigsaw* považujem za menej príjemné, hlavne z dôvodu monotónnosti a časovej náročnosti, keď som sa snažil vygenerovať puzzle podľa svojich vlastných predstáv. Na druhej strane je automatická generácia pomocou príkazu jigsaw veľmi jednoduchá a bezproblémová. Na obrázku 2 vidíte puzzle s logom Fakulty informatiky Masarykovej univerzity, ktoré som vygeneroval pomocou balíka *jigsaw* počas svojich experimentov.

Myšlienky

Obidva balíky sa majú ešte kam posunúť v kvalite používania, ale už v týchto skorých štádiách vývoja sú použiteľné. Majú rôzne možnosti uplatnenia, najviac však na vizuálne skrášlenie textu. Balíky, ako tieto, posúvajú hranicu možností TEXu, ako som ho doteraz poznal, a hrajú významnú úlohu pri zaujatí nováčikov ako aj ľudí, ktorí predtým nemali ani poňatia o TEXu.

Referencie

1. CARTER, Sam. *Bricks and jigsaw pieces* [online]. YouTube, 2022-08-29 [cit. 2022-11-23]. Dostupné z: `https://youtu.be/rplZ4zZ2IuE`.
2. PAKIN, Scott. *img2bricks* [online]. 2021-09-12. [cit. 2023-04-30]. Dostupné z: `https://github.com/samcarter/TikZbricks/blob/main/img2bricks`.



Obr. 2: Logo Fakulty informatiky Masarykovej univerzity vytvorené pomocou balíka *jigsaw*.

3. CARTER, Sam. *TikZbricks* [online]. TUG, 2022-07-23 [cit. 2023-02-13]. Dostupné z: `https://tug.org/tug2022/assets/served/samcarter-TUG2022-samcarter-bricks-bricks.pdf`.
4. CARTER, Sam. *TikZbricks* [online]. CTAN, 2022-07-21 [cit. 2022-11-23]. Dostupné z: `https://ctan.org/pkg/tikzbricks`.
5. CARTER, Sam. *The jigsaw package* [online]. TUG, 2022-07-23 [cit. 2023-02-13]. Dostupné z: `https://tug.org/tug2022/assets/served/samcarter-TUG2022-samcarter-bricks-jigsaw.pdf`.
6. CARTER, Sam. *jigsaw* [online]. CTAN, 2022-07-18 [cit. 2022-11-23]. Dostupné z: `https://ctan.org/pkg/jigsaw`.

Summary: TEX Newbie Reports on the "Bricks and Jigsaw Pieces" Talk at TUG 2022

The article deals with *TikZbricks* and *jigsaw* packages: their use, design, and possible applications. I look at these aspects of the packages through my own experimentation as a newish TEX user and I show why such packages are needed in the world of TEX.

Keywords: *TikZ*, *TikZbricks*, *jigsaw*

Matúš Vančík
514505@mail.muni.cz

Mělo by to fungovat XIII

Peter Wilson

Článek ukazuje několik způsobů, jak lze v LaTeXu použít verbatim text uvnitř argumentu některých maker. Dále jsou ukázány možnosti, jak zkrátit sazbu textu na danou šířku či výšku.

Klíčová slova: Makra, verbatim argumenty, zkracování textu, LaTeX

> . . . Cloath'd all in glistering coats,
> which made a shew. . .
>
> *Poems and Fancies*
> Margaret Cavendish

Cílem tohoto seriálu je ukázat čtenáři krátké kousky kódu, které mohou vyřešit některé z jeho problémů. Doufám, že situaci ještě více nezkomplikuji v důsledku mých chyb. Opravy, poznámky a návrhy na změny budou vždy vítány.

> Sir, I have found you an argument,
> but I am not obliged to find you an understanding.
>
> Samuel Johnson

1. Verbatim argumenty

Nedávno jsem byl upozorněn na problém, že verbatim text nemůže být použitý jako argument makra. Když například chceme vysázet text v zarámované `minipage`, lze to udělat jednoduše použitím prostředí `minipage` uvnitř argumentu makra `\fbox`.

```
\fbox{\begin{minipage}{0.97\columnwidth}
    Text uvnitř zarámované minipage
 \end{minipage}}
```

Toto funguje bez problémů až do doby, kdy uvnitř `minipage` použijeme verbatim text. Potom dostaneme zmatené chybové zprávy, a to i přesto, že bez použití rámečku vše uvnitř `minipage` funguje.

LaTeX disponuje makrem `\newsavebox`, s jehož pomocí lze uložit vysázený text a ten pak použít prostřednictvím makra `\usebox`.

Z anglického originálu *Glisterings* [1] přeložil Jan Šustek.

 doi: 10.5300/2023-1-2/54

Zde je uvedena definice prostředí `framedminipage`, které umožňuje použít verbatim text uvnitř rámečku.[1]

```
\newsavebox{\minibox}
\newenvironment{framedminipage}[2][c]{%
  \begin{lrbox}{\minibox}
    \begin{minipage}[#1]{#2}}%
  {\end{minipage}\end{lrbox}
    \noindent\fbox{\usebox{\minibox}}}
```

Při sazbě tohoto rámečku mělo prostředí `minipage` šířku rovnu 97 % šířky okolní sazby.

```
\begin{framedminipage}{0.97\columnwidth}
  ...
\end{framedminipage}
```

Prostředí `lrbox` je analogií LaTeXových maker `\savebox` a `\sbox`. Zatímco dovnitř argumentu makra nelze přímo vložit verbatim text, dovnitř těla prostředí jej vložit lze. V uvedeném kódu se nejprve deklaruje registr `\minibox` typu box, do nějž lze uložit vysázený text. Dále je definováno prostředí `framedminipage`, a to podobně jako prostředí `minipage`, včetně nepovinného argumentu určujícího vertikální umístění sazby. Na začátku prostředí `framedminipage` se zahájí prostředí `lrbox` a `minipage` a na jeho konci se zase obě tato prostředí ukončí. Vysázený obsah `minipage` se tak uloží do registru `\minibox`, přičemž si musíme uvědomit, že verbatim text už je uvnitř `\minibox` vysázený. Poté se obsah registru `\minibox` jen vloží do makra `\fbox`.

TeXbook [2, str. 363] definuje makro `\footnote` tak, aby jeho argument mohl obsahovat verbatim text. Knuth píše, že makro je propracované a používá několik triků. Já tomu makru nerozumím, ale ukážu zde jeho hlavní myšlenku. Pro jednoduchost makro `\verbtext` pouze vysází svůj argument. Nejsem si jistý, zda jsem správně umístil makra `\color@...`, protože nic podobného v původních Knuthových makrech nebylo.

```
\makeatletter
\long\def\verbtext{\vtintro\futurelet\next\vte@t}
\def\vte@t{\ifcat\bgroup\noexpand\next
    \let\next\vt@@t
  \else \let\next\vt@t\fi \next}
\def\vt@@t{\bgroup\aftergroup\vtend\let\next}
\def\vt@t#1{%
```

[1] V původním článku nebyl na řádku 9 použit příkaz `\noindent`, což mělo za následek, že makro `\fbox` při přechodu z vertikálního módu zahájilo odstavec a vložilo odstavcovou zarážku, což posunulo celý rámeček doprava. (pozn. překl.)

```
  \color@begingroup
  #1\vtmid
  \color@endgroup}
\let\vtintro\relax
\let\vtmid\relax
\let\vtend\relax
\makeatother
```

Makra `\vtintro` a `\vtend` se expandují před a za argumentem a jejich nadefinováním je možné dělat různé triky. Může se stát, že i definovat makro `\vtmid` bude užitečné.[2]

Jednoduché použití makra `\verbtext` může být

```
\verbtext{Argument makra \verb-\verbtext- může
  obsahovat \verb-\verb- text.}
```

s očekávaným výsledkem

Argument makra `\verbtext` může obsahovat `\verb` text.

Následující kód ukazuje použití maker `\vtintro` a `\vtend` pro sazbu textu kapitálkami.

```
\makeatletter
\newcommand*{\fred}[1][\@empty]{Frederick%
  \ifx\@empty #1\else~#1\fi}
\makeatother
\def\vtintro{\begingroup\scshape}
\def\vtend{\endgroup}
\verbtext{Makro \verb-\fred[III]-
  vypíše \uv{\fred[III]}, zatímco makro
  \verb-\fred- vypíše pouze \uv{\fred}.}
```

[2]Na řádku 15 se testuje, zda je argument makra `\verbtext` uzavřen do složených závorek (tj. za tokenem `\verbtext` následuje otevírací závorka), nebo zda je argumentem jediný token neuzavřený do složených závorek (v tomto případě samozřejmě nemůže nastat situace, že by argument obsahoval verbatim text). Na základě toho se pak provede buď makro `\vt@@t`, nebo makro `\vt@t`.

V makru `\vt@@t` se pomocí `\bgroup` otevře skupina, která se pak uzavře uživatelovou pravou závorkou. Nastaví se, že po uzavření této skupiny se provede `\vtend`. Konstrukce `\let\next` odstraní uživatelovu levou závorku. Poté se běžným způsobem načte a zpracuje vnitřek uživatelových složených závorek.

Makro `\vt@t` by mělo provést totéž, avšak bez nutnosti práce se skupinami. Pro stejnou funkci jako makro `\vt@@t` by stačilo definovat `\def\vt@t#1{#1\vtend}`.

Autor zřejmě chtěl ukázat, že je možné definovat různé chování makra na základě způsobu jeho zavolání. (pozn. překl.)

Makro `\fred[III]` vypíše „Frederick III“, zatímco makro `\fred` vypíše pouze „Frederick“.

V tomto konkrétním příkladě by šlo jednodušeji psát

```
{\scshape\verbtext{...}}
```

a nemuseli jsme se vůbec zatěžovat makry `\vtintro` a `\vtend`. Nicméně se určitě najdou situace, kdy se tato makra budou hodit.

> Wickedness is always easier than virtue;
> for it takes a short cut to everything.
>
> Samuel Johnson

2. Zkracování textu

Další z dotazů na `comp.text.tex` se týkal problému, kdy je třeba dlouhý text zkrátit například na dva nebo tři řádky.

Již v tu dobu existoval balíček truncate Donalda Arseneaua [3], který umožňuje zkrátit text na danou šířku. Na konci zkráceného textu se vytiskne „. . . “ (`\ldots`), případně jiná značka, aby se čtenář dozvěděl, že text ještě pokračuje. Například

```
\truncate{0.9\columnwidth}{Balíček \textsf{truncate} definuje
  makro pro zkracování textu tak, aby nepřekročil zadanou šířku.}
```

vysází

> Balíček truncate definuje makro pro zkracování textu tak, aby . . .

Nicméně dotaz se týkal vertikální analogie makra `\truncate`. V odpovědi [4] na dotaz Donald definoval makro `\vtruncate` následovně.

```
\newsavebox\descbox
\newsavebox\partialbox
\newcommand{\vtruncate}[2]{%
  \setbox\descbox\vbox{{#2\par}}%
  \setbox\partialbox\vsplit\descbox to #1\relax
  \vtop{\unvbox\partialbox}%
  % or use
  % \par\unvbox\partialbox
  }
```

Makro má dva argumenty. Druhým argumentem je text, který se zkrátí na výšku danou prvním argumentem.

V další odpovědi [5] Will Robertson definoval prostředí `cutlines`.

```
\makeatletter
\newbox\cut@desc
\newenvironment{cutlines}[1][2]{%
  \@tempcnta=#1\relax
  \setbox\cut@desc\vbox\bgroup
  \parskip=0pt}{%
  \egroup
  \vsplit\cut@desc to \@tempcnta\baselineskip}
\makeatother
```

Prostředí má jeden nepovinný argument (uzavřený do hranatých závorek), který udává, na kolik řádků se má text uvnitř prostředí zkrátit.

Vyzkoušel jsem obě dvě řešení a narazil jsem na následující možné problémy.

1. Textový argument makra `\vtruncate` nemůže obsahovat žádný verbatim text. (To nemusí až tolik vadit.)
2. Jestliže v prostředí `cutlines` je zadaný počet řádků větší než počet řádků textu, potom je pod text vložena vertikální mezera taková, aby celková výška odpovídala zadanému počtu řádků.
3. V obou řešeních se stává, že výsledná výška je jiná, než bylo zadáno. Vždy se však liší maximálně o jeden řádek. Zdá se, že `cutlines` je přesnější než `\vtruncate`.
4. Zkrácený text je umístěn ve `\vbox`u, v němž nemůže dojít ke stránkovému zlomu.

Po dlouhém ladění jsem dospěl k definici prostředí `vcutlines`, které je kombinací obou řešení a které řeší první dva problémy a možná i třetí problém. Čtvrtý problém se týká všech řešení.[3] Prostředí `vcutlines` má jeden nepovinný argument, který udává, na jakou výšku se má text uvnitř prostředí zkrátit.

```
\newsavebox\descbox
\newsavebox\partialbox
\newlength{\vcutl}% for the limit height
\newlength{\Vcutl}% height of full text
\newenvironment{vcutlines}[1][2\baselineskip]{%
  \setlength{\vcutl}{#1}%
  \setbox\descbox\vbox\bgroup
  \parskip=0pt\relax
  }{%
  \egroup
  \Vcutl=\ht\descbox
  \advance\Vcutl \dp\descbox
```

[3] I původní článek obsahoval řádek 48. Pokud by se tento řádek použil namísto řádků 46 a 72, pak by ke čtvrtému problému nedošlo. (pozn. překl.)

```
  \setbox\partialbox\vsplit\descbox to \vcutl\relax
  \vtop{\unvbox\partialbox}
  \ifdim \vcutl<\Vcutl \vtruncont \fi}
\newcommand*{\vtruncont}{\noindent\strut\ldots}
```

V následujících příkladech pro testování použijeme text

```
{\itshape Donald Arseneau vytvořil makro \verb-\vtruncate- a
Will Robertson vytvořil prostředí \verb-cutlines-. Obě řešení
zkrátí text, pokud by po vysázení měl větší než zadanou výšku.
V~tomto článku je vytvořeno nové prostředí \verb-vcutlines-,
které je kombinací obou předchozích řešení.}
```

který obsahuje i několik verbatim textů.

Pro začátek vyzkoušejme prostředí `vcutlines` s omezením na 20 řádků (tj. na výšku `20\baselineskip`).

Donald Arseneau vytvořil makro `\vtruncate` *a Will Robertson vytvořil prostředí* `cutlines`*. Obě řešení zkrátí text, pokud by po vysázení měl větší než zadanou výšku. V tomto článku je vytvořeno nové prostředí* `vcutlines`*, které je kombinací obou předchozích řešení.*

A nyní stejný text s omezením na 3 řádky.

Donald Arseneau vytvořil makro `\vtruncate` *a Will Robertson vytvořil prostředí* `cutlines`*. Obě řešení zkrátí text, pokud by po vysázení měl větší než zadanou výšku. V tomto článku je vytvořeno nové prostředí* `vcutlines`*, které je kombinací* ...

Jestliže dojde ke zkrácení textu, vloží se na konec prostředí makro `\vtruncont`, které implicitně vloží řádek obsahující pouze „...“ (`\ldots`). Pro zjištění, zda došlo ke zkrácení textu, se výška celého vysázeného textu porovnává se zadanou výškou.

Makro `\vtruncont` můžeme například nadefinovat na nic.

```
\renewcommand*{\vtruncont}{}
```

čímž dostaneme

Donald Arseneau vytvořil makro `\vtruncate` *a Will Robertson vytvořil prostředí* `cutlines`*. Obě řešení zkrátí text, pokud by po vysázení měl větší než zadanou výšku. V tomto článku je vytvořeno nové prostředí* `vcutlines`*, které je kombinací*
V tom případě se ale čtenář nedozví, zda text ještě pokračuje, nebo ne.

Odkazy

1. WILSON, Peter. Glisterings. *TUGboat* [online]. 2011, roč. 32, č. 3, s. 339–341 [vid. 2011-12]. Dostupné z: `https://tug.org/TUGboat/tb32-3/tb102glister.pdf`.

2. KNUTH, Donald E. *The TEXbook*. Sv. A. Reading, MA: Addison-Wesley, 1984. Computers & Typesetting. V současnosti jsou dostupné 35. výtisk (měkká vazba, 2017) a 23. výtisk (pevná vazba, 2021).
3. ARSENEAU, Donald. truncate.sty*: Truncate text to a specified width* [online]. CTAN, 2001 [vid. 2011-12]. Dostupné z: `https://ctan.org/pkg/truncate`. Verze 3.6.
4. ARSENEAU, Donald. *Re: How to limit/cut off text after a number of lines?* `comp.text.tex` newsgroup, 2008-07-16.
5. ROBERTSON, Will. *Re: How to limit/cut off text after a number of lines?* `comp.text.tex` newsgroup, 2008-07-16.

Summary: It Might Work XIII

This paper shows several ways in LaTeX how to use verbatim text within argument of some macros. We also show how to truncate text to a given width or height.

Keywords: Macros, verbatim arguments, truncating text, LaTeX

Peter Wilson, herries.press@earthlink.net
12 Sovereign Close
Kenilworth, CV8 1SQ, UK

Zpravodaj Československého sdružení uživatelů TEXu
ISSN 1211-6661 (tištěná verze), ISSN 1213-8185 (online verze)

Vydalo:	Československé sdružení uživatelů TEXu vlastním nákladem jako interní publikaci
Obálka:	Antonín Strejc
Ilustrace na obálce:	Vít Starý Novotný
Počet výtisků:	250
Uzávěrka:	30. 6. 2023
Odpovědný redaktor:	Jan Šustek
Redakční rada:	Pavel Haluza, Lukáš Novotný, Vít Starý Novotný, Michal Růžička a Jan Šustek (šéfredaktor)
Vědecká rada:	Ján Buša (předseda), Jiří Demel, Jaromír Kuben (zástupce předsedy), Jiří Rybička a Petr Sojka
Technická redakce:	Vít Starý Novotný
Evidenční číslo MK:	E 7629
Adresa:	CSTUG, Nejedlého 373/1, 638 00 Brno
Email:	cstug@cstug.cz

Zřízené poštovní aliasy sdružení CSTUG:

`bulletin@cstug.cz`, `zpravodaj@cstug.cz`
korespondence ohledně Zpravodaje sdružení

`board@cstug.cz`
korespondence členům výboru

`cstug@cstug.cz`, `president@cstug.cz`
korespondence předsedovi sdružení

`gacstug@cstug.cz`
grantová agentura CSTUGu

`secretary@cstug.cz`, `orders@cstug.cz`
korespondence administrativní síle sdružení, objednávky CD a DVD

`cstug-members@cstug.cz`
korespondence členům sdružení

`cstug-faq@cstug.cz`
řešené otázky s odpověďmi navrhované k zařazení do dokumentu CSFAQ

`bookorders@cstug.cz`
objednávky tištěné TEXové literatury na dobírku

ftp server sdružení:
`ftp://ftp.cstug.cz`

www server sdružení:
`https://www.cstug.cz`

CONTENTS

Petr Sojka: Introductory Word 1

Vít Starý Novotný: Character Name Generator for Creative Writing in LuaTEX 3

Karel Šebela: Musical Composition Typesetting 39

Matúš Vančík: TEX Newbie Reports on the “Bricks and Jigsaw Pieces” Talk at TUG 2022 48

Peter Wilson: It Might Work XIII 54