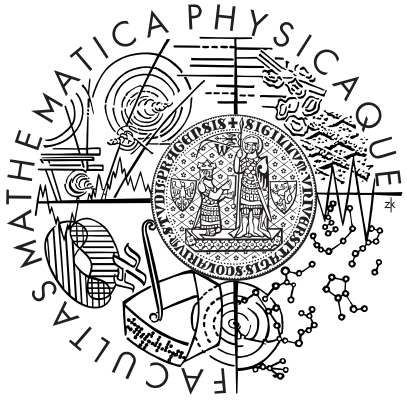
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Petr Kubát

Normalizace pojmenovaných entit v českých textech

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Barbora Vidová Hladká, Ph.D.

Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2014

Na tomto místě bych rád poděkoval všem lidem, bez nichž by tato práce nemohla vzniknout. Předně děkuji Mgr. Barboře Vidové Hladké, Ph.D., za téma práce, poskytnutou pomoc, ochotu a cenné připomínky. Dále Mgr. Vincentu Krížovi, Mgr. Martinu Popelovi a RNDr. Milanu Strakovi, Ph.D., za konzultace a rady související s nástroji Treex a MorphoDiTa, které jsou pro tuto práci velmi důležité. V neposlední řadě patří mé poděkování doc. RNDr. Vladislavu Kuboňovi, Ph.D., a všem studentům jeho semináře, kteří se podíleli na přípravě použitých testovacích dat. Děkuji také své rodině a přátelům za podporu při psaní této práce.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 22.5.2014

Název práce: Normalizace pojmenovaných entit v českých textech

Autor: Petr Kubát

Katedra / Ústav: Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Barbora Vidová Hladká, Ph.D.

Abstrakt: Pojmenované entity jsou slovní spojení, která v textu označují objekty reálného světa. Normalizací entit nazveme jejich převod do základního tvaru. Práce se zabývá vytvořením pravidlové procedury určené k normalizaci pojmenovaných entit v českých textech. Proces návrhu jednotlivých pravidel této procedury je důkladně zmapován. Důraz je kladen na to, aby každé pravidlo bylo motivováno příklady reálných entit. Za účelem dosažení co největší úspěšnosti jsou také analyzovány některé aspekty syntaxe českého jazyka. Na základě teoretického popisu procedury je dále implementován normalizační program a jeho úspěšnost je vyhodnocena srovnáním s ručně normalizovanými entitami. Ve spojení s již existujícími nástroji pro automatické rozpoznávání pojmenovaných entit v textu je možné tento normalizátor využít v jiných procesech strojového zpracování textu, například překladu do jiného jazyka, vyhledávání a kategorizaci apod.

Klíčová slova: pojmenované entity, normalizace, pravidlový systém

Title: Named Entity Normalization in Czech Texts

Author: Petr Kubát

Department: Institute of Formal and Applied Linguistics

Supervisor: Mgr. Barbora Vidová Hladká, Ph.D.

Abstract: Named entities are collocations used to refer to real world objects in text. Named entity normalization is a process of generating the basic form for a given named entity. The thesis is focused on creating a rule-based procedure for named entity normalization in Czech texts. The process of designing individual rules is closely examined. Stress is laid on the fact that each rule is motivated by entities from real-world texts. Additionally, some aspects of Czech language syntax are analyzed in order to achieve the highest possible accuracy. Based on the theoretical description of the procedure, a normalization application is implemented, and its accuracy is evaluated by comparison with manually normalized entities. Together with already existing tools for automatic named entity recognition, it is possible to use this normalizer in other text processing tasks, such as machine translation, searching and categorization, etc.

Keywords: named entities, normalization, rule-based system

Obsah

Úvod

V současné době se stává stále důležitějším strojové zpracování rozsáhlých textových dat. S rostoucím objemem textu nejen na internetu, ale i z jiných zdrojů, začíná být klíčová schopnost extrakce užitečných dat počítačem. Důkazem z prostředí World Wide Webu může být například nejnovější standard jazyka HTML (HyperText Markup Language), HTML 5.0, který tento značkovací jazyk obohacuje mimo jiné o mnoho prvků sloužících čistě k zachycení významu obsažených dat, nikoliv jejich vizuální podoby, právě pro usnadnění jejich strojové interpretace. Velice podobným způsobem zpřístupnění rozšiřující informace o významu textu, vhodným spíše pro souvislé dokumenty, je vyznačení pojmenovaných entit v něm.

Pojmenované entity jsou úseky textu, jež označují nějaké objekty reálného světa. Můžeme je považovat za identifikátory těchto objektů. Jeden konkrétní objekt pak bude často mít mnoho takovýchto identifikátorů, neboť jej můžeme označit různými názvy. Normalizace (i v zahraniční literatuře) je proces, jehož cílem je zjednoznačnit tyto identifikátory reálných objektů, tedy každému výskytu pojmenované entity přiřadit jednoznačný identifikátor objektu, který označuje.

Normalizace v pojetí této práce se omezí na aspekt českého jazyka, a sice skloňování. Jedno slovní spojení tvořící pojmenovanou entitu se může v českých větách vyskytovat v různých tvarech. Při normalizaci budou tato spojení převáděna do tvarů základních. Nebudeme tedy měnit slova samotná, pouze je budeme skloňovat.

Cílem je navrhnout proceduru, která bude schopna tyto entity převádět do základních tvarů s co největší úspěšností. K tomu bude využito obvyklých technik počítačového zpracování češtiny, jakými jsou analýza větné struktury a morfologie slov. Procedura by měla být prezentována formou pravidel, jež budou založeny na pozorování co největšího počtu pojmenovaných entit v reálných textech.

První kapitola této práce obsahuje stručné představení metod strojového zpracování přirozeného jazyka, především češtiny. Důraz je kladen zejména na zavedení pojmů, které budou ve zbytku textu často využívány, a seznámení čtenáře se základními koncepty této problematiky.

Druhá kapitola podrobněji popisuje, co to vlastně pojmenované entity jsou. Přibližuje jejich význam, využití a také možnosti jejich automatického rozpoznávání. V neposlední řadě je zde představen soubor CNEC (Czech Named Entity Corpus).

Třetí a nejobsáhlejší kapitola se věnuje konstrukci pravidlové procedury pro automatickou normalizaci pojmenovaných entit v českých textech. Proces návrhu jednotlivých pravidel je popisován iterativně tak, jak procedura opravdu vznikala.

Ve čtvrté kapitole se podíváme na technické aspekty implementace zmíněné pravidlové procedury - cílem bude navrhnout reálný program, normalizátor

pojmenovaných entit. Zvolíme pro tento úkol vhodné nástroje, prostředí a knihovny. Dále představíme jednotlivé nástroje, jež budou normalizátorem využívány, a podíváme se na jejich výhody, nevýhody a na způsob práce s nimi.

Cílem páté kapitoly potom bude zhodnotit úspěšnost a použitelnost implementovaného normalizátoru. K tomu využijeme několik reálných textů s vyznačenými pojmenovanými entitami, na nichž budeme srovnávat výsledky automatické a ruční normalizace. Zjistíme, v jakých typech textů je automatická normalizace úspěšná, a v jakých méně.

Šestá kapitola je formální dokumentací k normalizátoru. V uživatelské části je přiblížena instalace a používání nástroje. Programátorská dokumentace pak obsahuje rámcový přehled tříd programu a jeho celkového fungování.

Normalizátor implementovaný v rámci práce je k dispozici na přiloženém CD.

1 Počítačové zpracování češtiny

1.1 Zpracování přirozeného jazyka

Naše schopnost porozumět jazyku je založena na poměrně důkladné analýze syntaxe a tvarosloví, kterou provádíme zcela automaticky. Ta obnáší například dešifrování struktury jednotlivých vět, rozpoznání větných členů, zjištění vazeb mezi nimi, ale i identifikaci čísla a pádu podstatných jmen. Všechny uvedené informace je schopen lidský mozek interpretovat a na jejich základě přiřadit přečtené výpovědi smysl a význam. Bez nich by libovolný text byl pouze řetězcem slov a zcela by postrádal původní smysl. Podívejme se na příklad dvou vět:

Příklad 1: *Policista postřelil recidivistu, jenž minulý týden uprchl z vězení.*

Příklad 2: *Policistu postřelil recidivista, jenž minulý týden uprchl z vězení.*

Jediné, v čem se tyto dvě věty liší, je koncovka dvou podstatných jmen, tedy pouhá dvě písmena. Přesto mají zcela opačný význam. K pochopení nám dopomohla znalost koncovek pádů češtiny a také význam, v němž se jednotlivé pády používají.

Mohou nastat také situace, kde pouze na základě syntaktické a tvaroslovné analýzy nejsme schopni nějaké výpovědi jednoznačně přiřadit význam, obzvlášť v jazycích podobných češtině, kde u vět není pevně dán slovosled. Příkladem mohou být následující věty:

Příklad 3: *Představil tchýni hospodyni.*

Příklad 4: *Na nástupišti hlídají kamery batohy.*

V takové situaci musíme využít také znalosti kontextu, v němž se výpověď nachází. Tím většinou jednoznačně rozhodneme o významu vět podobných příkladu 3. U příkladu 4 už je potřeba schopnost chápat význam obou alternativ a na základě zkušenosti se pro jednu z nich rozhodnout.

Vidíme tedy, že proces porozumění textu je velice komplexní a vyžaduje jak schopnost analýzy slov a vět dle předem daných pravidel (syntaxe a tvarosloví), tak také zapojení přirozené intuice (význam výpovědi).

Nyní si představme, že chceme, aby byl podobným způsobem schopen textu porozumět i počítač, například pro účel překladu do jiného jazyka. V takovém případě se budeme muset obejít bez složky intuice. Analýzu syntaxe a tvarosloví však může počítač provádět podobně jako člověk s tím, že jeho úspěšnost nebude vždy stoprocentní. Výsledkem bude sada rozšiřujících údajů jednoznačně charakterizujících význam textu. Tyto údaje se obvykle ukládají k textu samotnému ve formě anotací určených k využití při dalším zpracování, například při zmíněném překladu. Proto se tyto procesy tvaroslovného a větného rozboru textu často souhrnně nazývají **anotace** textu a jsou základem počítačového zpracování přirozeného jazyka.

Pro úkol, jímž se bude zabývat tento text, se v této kapitole podrobněji zaměříme na vybrané aspekty zpracování češtiny, jež budou využity.

1.2 Koncepce Pražského závislostního korpusu

Počítačové zpracování přirozeného jazyka funguje v současnosti převážně na principu statistickém - všechny doplňující informace, které chceme v textech zkoumat, se musí počítač nejprve „naučit“ z nějakého ručně anotovaného textu dostatečného rozsahu. Takový text se pak nazývá **korpus**. Poté, co se počítač natrénuje na správných anotacích v korpusu, je schopen tyto anotace sám provádět na dalších textech s určitou úspěšností. Ta je u dnešních programů poměrně vysoká, ale ne stoprocentní.

Nejdůležitějším korpusem pro počítačové zpracování českého jazyka je Pražský závislostní korpus (Prague Dependency Treebank, PDT) [1]. Právě ten v podstatě definoval standardy, v nichž anotace v českých textech probíhají v Ústavu formální a aplikované lingvistiky MFF UK (ÚFAL). Podle jeho koncepce se tyto anotace dělí do několika rovin, které staví jedna na druhé a které postupně rozšiřují hloubku, do níž je text zkoumán. Jde o rovinu *slovní*, *morfologickou*, *analytickou* a *tektogramatickou*. Poslední zmíněné rovině se tento text věnovat nebude.

1.2.1 Slovní rovina

Slovní rovina (w-rovina) je první a nejjednodušší rovinou, neobsahující zatím žádnou anotaci (je tedy rovinou neanotační). Sestává z původního textu, jenž je rozdělen na dokumenty, odstavce a samostatné slovní jednotky (tokeny). Každá z těchto slovních jednotek může mít přiřazen jednoznačný identifikátor.

Procesu generování w-roviny říkáme **tokenizace**. Jeho podstatou je správné nalezení hranic jednotlivých úseků textu (dokumentů, odstavců a slovních jednotek) a rozdělení textu podle nich. Dělí se tedy podle mezer, konců řádků, ale také podle hranic interpunkce. Při tom již mizí původní formátování textu, například více mezer za sebou.

Uveďme příklad na následující větě:

Příklad 5: *Policista postřelil recidivistu, jenž minulý týden uprchl z vězení.*

W-rovina této věty uložená ve formátu XML (Extensible Markup Language) bude vypadat následovně:

```
<sentence>
<token>Policista</token> <token>postřelil</token>
<token>recidivistu</token><token>,</token> <token>jenž</token>
<token>minulý</token> <token>týden</token> <token>uprchl</token>
<token>z</token> <token>vězení</token><token>.</token>
</sentence>
```

Procedury provádějící anotaci textu na vyšších rovinách většinou pracují právě s w-rovinou, neboť zpracovávají již jednotlivé slovní jednotky či věty. Proto je většinou tokenizace prováděna jako první fáze při anotaci na jakékoliv rovině.

1.2.2 Morfologická rovina

Anotace na morfologické rovině (m-rovině) využívá rozdělení na jednotky z w-roviny. Těmto slovním jednotkám pak přiřazuje několik atributů popisujících jejich vlastnosti z hlediska tvaroslovného. Nejdůležitějšími atributy jsou lemma a morfologická značka.

Lemma obsahuje základní tvar dané jednotky (tj. 1. pád čísla jednotného u podstatných jmen, infinitiv u sloves apod.). Jde o jednoznačný identifikátor mezi všemi jejími tvary. K němu mohou být připojeny přípony čtyř různých typů (rozlišených podle způsobu připojení) rozšiřujících informaci o daném slově:

```
vlastnílemma_:P1_;P2_,P3_^(P4)
```

Přípona P1 nese informaci o vidu slovesa, případně udává, zda se jedná o zkratku. P2 určuje typ vlastního jména, P3 stylový příznak. V P4 pak může být uvedena libovolná rozšiřující informace.

Vzhledem k tomu, že často existuje více různých slov se stejným tvarem lemmatu, ale jiným skloňováním či časováním, je někdy nutné lemma opatřit číslem, jež jednoznačně určí konkrétní význam. Například lemma *stát* může mít hned pět významů (konkrétně jsou specifikovány v příponě typu P4):

Příklad 6:

```
stát-1_^(státní_útvar)
stát-2_^(něco_se_přihodilo)
stát-3_^(někdo/něco_stojí,_např._na_nohou)
stát-4_^(něco_stojí_peníze)
stát-5_^(sníh)
```

Proces, při němž je slovu přiřazeno jeho lemma, nazveme **lemmatizací**.

Morfologická značka (tag) je řetězcem sestávajícím z 15 znaků. Tyto znaky obsahují hodnoty jednotlivých morfologických kategorií shournně popisujících tvar, v němž se dané slovo vyskytuje. Jde o značku poziční, význam jednotlivých znaků tedy udává jejich pozice v rámci řetězce. Na prvním místě se například nachází identifikátor slovního druhu, na třetím rodu, na čtvrtém čísla a na pátém pádu. Některé pozice značky jsou momentálně nevyužité. Kompletní seznam používaných kategorií a jejich hodnot je uveden v dokumentaci k PDT [1].

V rámci jedné značky mohou mít některé kategorie nedefinovanou hodnotu, například slovesa v rámci pádu, rodu a čísla. Taková hodnota je reprezentována znakem '-' na daném místě. Viz několik příkladů značek:

Příklad 7:	`hrály`	`VpTP---XR-AA---`
Příklad 8:	`sousedů`	`NNMP2-----A----`
Příklad 9:	`neboť`	`J^------------`
Příklad 10:	`rychlými`	`AAMP7----1A----`

Příklad 7 udává sloveso (hodnota V na pozici 1) v ženském či mužském neživotném

rodě (T na pozici 3) plurálu (P na pozici 4) neurčené osoby (X na pozici 8) a minulého času (R na pozici 9). Příklad 8 ukazuje podstatné jméno (N na pozici 1) mužského rodu (M na pozici 3) v plurálu druhého pádu (2 na pozici 5). Příklad 9 obsahuje spojku (J na pozici 1). Příklad 10 pak přídavné jméno (A na pozici 1) rodu mužského v plurálu sedmého pádu (7 na pozici 5) prvního stupně (1 na pozici 10).

Procesu, při němž slovu přiřazujeme množinu možných kombinací lemmat a značek, říkáme **morfologická analýza**. Tu lze provést prohledáním morfologického slovníku, tedy slovníku obsahujícího všechny tvary všech slov daného jazyka, a vrácením shodných záznamů. Je obvyklé, že takových záznamů najdeme více, tj. že jeden tvar slova podle slovníku odpovídá různým dvojicím značka-lemma. Při anotaci na m-rovině však musí být slovu přiřazeno právě jedno lemma a právě jedna značka. K tomuto zjednoznačnění dochází při tzv. **značkování** (tagging). To využívá kontextu, ve kterém se ve větě slovo nachází, ke správně volbě jediné dvojice.

Jako příklad prozkoumejme slovo *hlavní* v následující větě:

Příklad 11: *Hlavní výhodou internetu je bezesporu jeho flexibilita.*

Část výsledku morfologické analýzy (z celkových 29 možností)[1]:

```
hlaveň  NNFP2-----A----
hlaveň  NNFS7-----A----
hlavní  AAFP1----1A----
hlavní  AAFP4----1A----
hlavní  AAFP5----1A----
```

Může jít například o podstatné jméno *hlaveň* v druhém pádě jednotného či sedmém pádě množného čísla. Nebo může jít o přídavné jméno *hlavní*. To z tvaru samotného nejsme schopni jednoznačně určit. Použijeme tedy kontext, procedura značkování vrátí tento výsledek:

```
hlavní  AAFS7----1A----
```

Bylo tedy správně určeno přídavné jméno rodu ženského v sedmém pádě.

S morfologickou analýzou souvisí ještě jedna důležitá procedura zpracování přirozeného jazyka - **morfologické generování** (či morfologická syntéza). Cílem je pro dané lemma a danou značku vygenerovat odpovídající tvar slova. K tomu lze opět využít morfologický slovník. Tímto směrem by obecně mělo být přiřazení jednoznačné[2], každé dvojici lemma-značka by mělo odpovídat vždy nejvýše jedno konkrétní slovo. Často se však stane, že značka je pro dané lemma nesmyslná, např. pro lemma *barevný* značka `VpYS---XR-AA---`, která je značkou slovesa a ne přídavného jména.

Formátů pro uložení m-roviny je mnoho, nejčastěji se využívají XML značky.

1 Pro všechny příklady výstupu morfologické analýzy v této kapitole byl použit software MorphoDiTa [8].

2 V praxi se u některých morfologických slovníků můžeme setkat s více možnými tvary pro jednu dvojici lemma-značka, např. u českých přepisů cizích slov.

Alternativou je čistý text, kde každá jednotka je na samostatném řádku s lemmatem a tagem oddělenými tabulátory. Pro úplnost si ukažme kompletní m-rovinu již použité věty z příkladu 5:

```
Token          Lemma                                Tag
Policista      policista                            NNMS1-----A----
postřelil      postřelit_:W                         VpYS---XR-AA---
recidivistu    recidivista                          NNMS4-----A----
,              ,                                    Z:-------------
jenž           jenž_^(který_[ve_vedl.větě])         PJYS1----------
minulý         minulý                               AAIS4----1A----
týden          týden_^(jednotka_času)               NNIS4-----A----
uprchl         uprchnout_:W                         VpYS---XR-AA--1
z              z-1                                  RR--2----------
vězení         vězení_,a_^(místo_výkonu_trestu)     NNNS2-----A----
.              .                                    Z:-------------
```

1.2.3 Analytická rovina

Anotace na analytické rovině (a-rovině) zpracovává větu jako celek na úrovni syntaxe. Rozlišuje tedy jednotlivé větné členy, jejich úlohu ve větě a vztahy mezi nimi. Takový rozbor nemůže být již zachycen pouze pomocí atributů u jednotlivých slovních jednotek (jako v případě m-roviny). Proto při anotaci na a-rovině dochází k uspořádání slovních jednotek do tzv. **závislostního stromu**. Jde o strom, jehož kořenem je pomocný uzel, zbylé uzly potom reprezentují jednotlivé slovní jednotky. Vztah otec-syn v tomto stromě vyjadřuje závislost větného členu reprezentovaného synem na větném členu reprezentovaném otcem. Potomky kořenu jsou všechny větné členy, jež na žádném jiném členu nezávisí, plus koncová interpunkce. U běžné věty jednoduché by potomkem kořenu měl kromě této interpunkce být pouze přísudek.

Jednotlivé uzly závislostního stromu jsou dále očíslovány čísly 0 až n, kde n je celkový počet slov ve větě. Přitom platí, že kořen má vždy číslo 0, a že číslo ostatních uzlů udává pořadí jejich slovní jednotky v rámci původní věty. V souvislosti s číslováním v závislostním stromě rozlišujeme **levé** a **pravé syny** daného uzlu podle toho, zda je číslo synovského uzlu menší či větší než číslo otce, tedy zda v původní větě slovní jednotka odpovídající synovi předchází slovní jednotce odpovídající otci, či následuje až po ní.

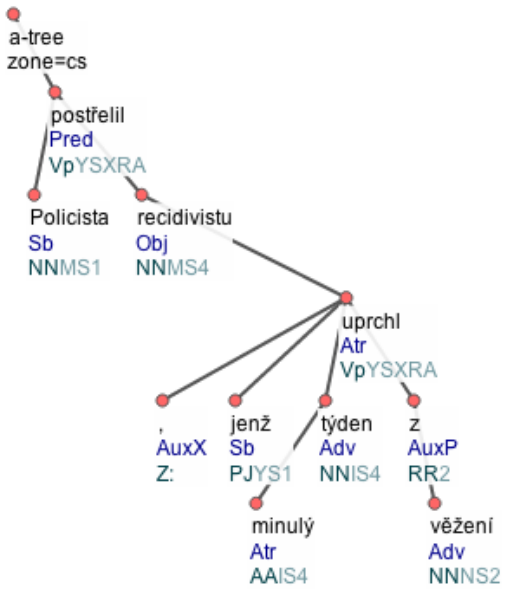
Dodejme také, že pojmem **řídící uzel** (**řídící člen**) budeme označovat ve stromové hierarchii nejbližší logicky nadřazený uzel (větný člen, který reprezentuje), který není spojkou, předložkou či interpunkcí, tedy fakticky uzel, který by u shodných větných členů řídil jejich tvar.

V závislostním stromě je dále každému uzlu přidělena hodnota **analytické funkce**. Ta vyjadřuje způsob, jakým uzel rozvíjí svého rodiče, tedy jakou funkci ve větě má. Nejdůležitější hodnoty[3] této funkce uvádí tabulka 1.

3 Všechny hodnoty, jichž může analytická funkce nabývat, jsou uvedeny v dokumentaci k PDT [1].

Funkce	Popis
Pred	Predikát, resp. uzel, který nezávisí na jiném uzlu; věší se na # (kořen stromu).
Sb	Subjekt (podmět).
Obj	Objekt (předmět).
Adv	Adverbiale (příslovečné určení, bez dalšího rozlišení).
Atv	Doplněk (jen tzv. určující), technicky zavěšen na neslovesném členu.
AtvV	Doplněk (jen tzv. určující), visící na slovese (chybí druhý řídící člen).
Atr	Atribut (přívlastek).

Tab. 1: Vybrané hodnoty analytické funkce



Obr. 1: Závislostní strom věty „Policista postřelil recidivistu, jenž minulý týden uprchl z vězení“

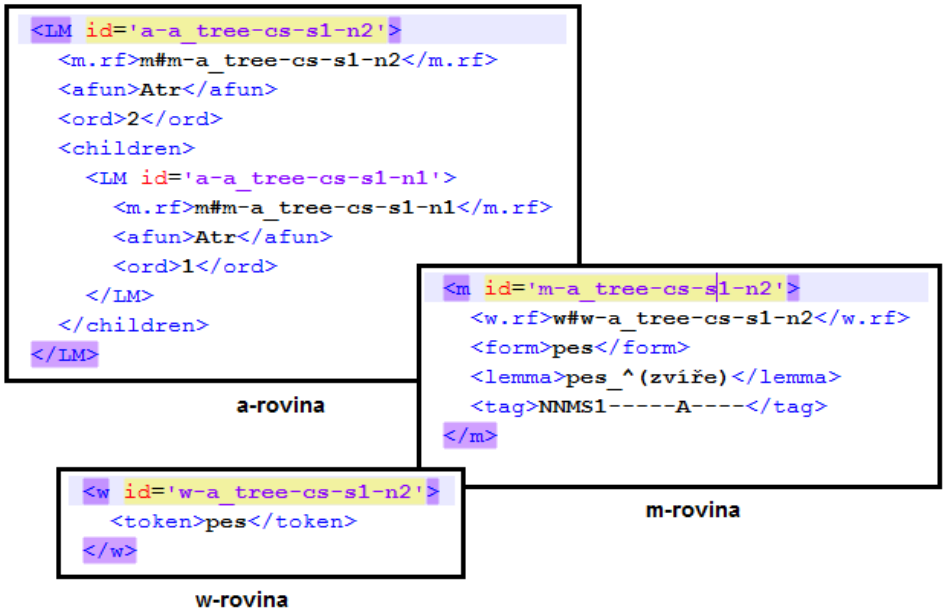
Nástroj, který provádí automatickou anotaci textu na a-rovině, se nazývá **parser**, jeho činnost je tedy **parsing** textu. Parsery pracují s textem již předem anotovaným na m-rovině. Přístupů, jak parsing provádět, je více, příkladem může být velmi rozšířený MSTParser. MST je zkratka pro Maximum Spanning Tree, česky maximální kostry grafu. Takovýto parser tedy pracuje na principu hledání maximální kostry v úplném grafu všech slovních jednotek, přičemž ohodnocení jednotlivých koster v závislosti na morfologických vlastnostech těchto jednotek je dáno právě statistickou funkcí získanou z nacvičených dat. MSTParser je v tomto textu využit pro všechny ukázky anotací na a-rovině a závislostních stromů.

Pro uložení anotace na a-rovině se prakticky výlučně používá formát XML, jenž umožňuje velmi jednoduše a názorně zachytit stromovou strukturu. Závislostní strom pak lze různými nástroji vizualizovat. Na obr. 1 si můžeme prohlédnout závislostní strom věty rozebírané již dříve. Můžeme si všimnout, že MSTParser si s problémem uvedeným v sekci 1.1 (příklady 1 a 2) dokázal poradit a správně určil činitele děje.

1.2.4 Propojení rovin

Mají-li všechny roviny anotace pro jednotlivé slovní jednotky uchovány jednoznačné identifikátory, mohou být navzájem propojeny. Propojení bývá realizováno jednosměrnými odkazy ve směru od roviny logicky vyšší k nejbližší rovině nižší. Tedy uzel a-roviny má odkaz na slovní jednotku v m-rovině a ta zpravidla na odpovídající jednotku ve w-rovině.

Toto propojení umožní pracovat naráz s aspekty více rovin současně, tedy například měnit strukturu m-roviny (lemma či morfologickou značku) na základě pozice odpovídající jednotky v a-rovině.



Obr. 2: Sdílení identifikátoru mezi uzly jednotlivých rovin ve formátu PML

Realizace propojení vyžaduje jednotný formát anotací na všech rovinách. Ten poskytuje např. formát **PML** (Prague Markup Language) [2], což je nadstavba XML, která uchovává každou rovinu ve zvláštním souboru (přípony .w, .m a .a). Každému uzlu (slovní jednotce) je přiřazen jednoznačný identifikátor, který je mezi rovinami sdílen (viz obr. 2). Jiným příkladem je formát **treex**, jenž má původ v nástroji Treex [3] (viz 4.4), a jenž sdružuje anotace všech rovin v jediném stromě (jedna slovní jednotka se tedy vyskytuje jako jeden uzel obsahující údaje všech rovin).

Příklad takového uzlu:

Příklad 12:

```
<LM id=a_tree-cs-s1-n5">
  <form>jenž</form>
  <lemma>jenž_^(který_[ve_vedl.větě])</lemma>
  <tag>PJFS1----------</tag>
  <no_space_after>0</no_space_after>
  <ord>5</ord>
  <afun>Sb</afun>
</LM>
```

2 Pojmenované entity

2.1 Definice pojmu

Tato práce se zabývá specifickými úseky textu, jež jsou označovány jako **pojmenované entity** (angl. **named entity**). Už samotné vymezení tohoto výrazu je poněkud problematické. Žádná exaktní definice, na základě níž bychom mohli o jakémkoliv úseku textu jednoznačně prohlásit, že takovouto entitu tvoří, totiž neexistuje. Technická zpráva o zpracování pojmenovaných entit v českých textech [4] využívá k jejich zavedení seznam konkrétních typů slovních spojení (jména osob, geografické názvy, jména produktů, časové údaje apod.). Tento přehled je úzce spojen s obvyklou kategorizací pojmenovaných entit, která bude zmíněna později.

Abychom však byli schopni zahrnout všechna odvětví, v nichž se s konkrétním pojmenováním můžeme setkat, musel by takový seznam být mnohonásobně delší. Například v tomto textu narazíme na pojmenované entity z legislativní domény, u nichž se bude jednat zejména o pojmenování zákonů, směrnic, vyhlášek, správních orgánů či dokumentů. Proto se v rámci práce nebudeme omezovat na zavedené kategorie podchycující nejobvyklejší příklady a vyhneme se konkrétnímu vymezování. Vystačíme si s jednoduchým shrnutím – za entitu budeme považovat vše, co v rámci textu označuje nějakou konkrétní osobu, věc, místo či událost. Přitom věc může být i nefyzického charakteru, ale v určitém smyslu konkrétně pojmenovatelná, tedy například matematická věta, legislativní zákon, báseň či článek.

2.2 Význam

Základním odvětvím, v němž pojmenované entity využijeme, je strojové zpracování přirozeného jazyka. Jsme-li v textu schopni rozlišit pojmenovanou entitu, případně dokonce typ informace v ní obsažené, umožní nám to tento text (nebo jeho úsek) klasifikovat a poskytnout zjištené údaje k dalšímu zpracování.

Taková „přidaná“ informace může výrazně pomoci v mnoha obvyklých činnostech počítačového zpracování textu. Umožní například v rozsáhlých dokumentech z různých specializovaných odvětví vyhledávat mnohem efektivněji na základě kategorizace klíčových slov – hledáme-li např. konkrétní událost, můžeme hledat primárně (či dokonce zcela výlučně) mezi entitami reprezentujícími právě události. Podobně se zjednoduší vyhledávání podle jmen osob, dat, míst a mnoha dalších snadno klasifikovatelných kategorií.

Dále je možné entit využít pro efektivní strojovou extrakci užitečných dat z rozsáhlých textů. Na základě toho je možné porovnávat různé dokumenty, hledat shodu mezi nimi a podobně.

V neposlední řadě mohou pojmenované entity v textech výrazně pomoci při strojovém překladu. Se spojením tvořícím entitu se obvykle při překladu bude

nakládat jinak, než se zbylým textem. Entita by se měla vždy překládat jako celek, neboť jinak je možné, že zcela pozbude svého významu. Její ekvivalent v daném jazyce by se měl volit na základě reálného pojmenování objektu, který označuje.

K tomu, aby pojmenovaná entita mohla být při strojovém zpracování textu využita, se musí o její existenci dozvědět systém provádějící toto zpracování. Musí dostat informaci o tom, kde pojmenovaná entita v textu začíná, kde končí, případně do jaké kategorie spadá. V úvahu připadají dva základní způsoby získání této informace:

1. Ruční vyznačení entit v textu
2. Automatické rozpoznávání entit

Hlavní výhodou ručního vyznačování je velmi vysoká přesnost. Prakticky je při něm dokument anotován pomocí speciálně určené sady značek, která snadno umožní i klasifikaci jednotlivých entit. Příkladem tohoto způsobu anotace je soubor Czech Named Entity Corpus (CNEC), kterému se budeme věnovat více v souvislosti s entitami v českých textech.

2.3 Automatické rozpoznávání entit

Ruční anotace rozsáhlejších textů je značně komplikovaná a časově náročná. Proto by bylo vhodnější, kdyby počítač dokázal sám nejprve v čistém textu pojmenované entity rozpoznat, interně vyznačit a poté jej dále zpracovat. V takovém případě hovoříme o automatickém rozpoznávání entit a počítačové programy, které jej provádí, nazýváme systémy NER (Named Entity Recognition). Fungování systémů NER by se dalo přirovnat k práci taggeru či parseru (viz sekce 1.2) - jde také o anotaci textu.

První systémy NER (ačkoliv se tehdy tak nenazývaly) paradoxně předcházely zavedení termínu pojmenované entity. Vznikaly v rámci série konferencí MUC (Message Understanding Conference) v 80. a 90. letech dvacátého století, jejichž hlavním tématem byla automatická extrakce informací z textu. Prvních pět konferencí se zabývalo vždy nějakým konkrétním úkolem, např. extrakcí informací z vojenských textů. Na šesté konferenci MUC-6 byl poprvé předložen úkol extrahovat z textu různé druhy jmen a názvů, a to nezávisle na jeho charakteru. Právě zde byl v roce 1995 poprvé použit termín named entity, tedy pojmenovaná entita, pro tato různorodá pojmenování. Dá se tudíž říct, že význam pojmenovaných entit vyplynul ze schopnosti je automaticky rozpoznávat.

Cílem nástrojů vyvíjených v rámci této konference bylo nejen entity v textu identifikovat, ale také správně určit jejich typ. Proto zde byl vyvinut první formát pro anotování a kategorizaci entit v rámci textu. Rozeznával tři základní druhy entit (*entity*, *časové údaje* a *množstevní údaje*), z nichž každý zahrnoval ještě několik poddruhů. Celkem bylo možné entitu zařadit do 7 různých kategorií.

Brzy se ukázalo, že 7 kategorií je pro klasifikaci entit velmi málo. Vznikala rozšíření této MUC koncepce přidávající nové druhy entit, ale také kompletně nové formáty, jež nabízely mnohem bohatší škálu typů. Například hierarchie sestavená firmou

BBN zahrnovala 29 typů a 64 podtypů[4]. Jiným přístupem byla směrnice vydaná TEI (Text Encoding Initiative)[5]. Ta nabízí široké možnosti značkování textu, mimo jiné právě i označování názvů a jmen. Pro jejich typy však není pevně stanovena množina možných hodnot - je možné si dodefinovat vlastní a tím tedy např. přizpůsobit tento formát specifickým požadavkům konkrétního úkolu. Tento přístup se uchytil a byl využit i v mnoha reálných projektech.

Systémy NER většinou fungují na principech strojového učení a jako podklady jsou pro ně použity právě rozsáhlé textové dokumenty, na nichž bylo dříve provedeno ruční vyhledání a vyznačení entit. Takovýto systém je typicky nezávislý na konkrétním jazyce a jeho úspěšnost bývá poměrně vysoká (okolo 90%). Byly vyvinuty i jiné typy rozpoznávacích algoritmů, fungující na různých specifických principech.

K porovnávání jednotlivých systémů NER a jejich úspěšnosti při rozpoznávání byla zavedena hodnota F-measure. Ta je závislá jak na poměru počtu správně určených entit vůči počtu všech určených entit, tak na jeho poměru vůči počtu všech entit v textu. Pohybuje se mezi 0 a 1, lze ji tedy snadno vyjádřit procentuálně. Vzorec pro její výpočet a podrobnější informace jsou spolu s dalšími informacemi o principech používaných NER uvedeny v Technické zprávě [4].

2.4 Entity v českých textech

2.4.1 Klasifikace

Podmnožina pojmenovaných entit v podobě vlastních jmen byla vyznačena již v několika korpusech českého jazyka. Jedná se např. o Pražský závislostní korpus [1], který ve svojí první verzi vznikl v roce 2001. V tomto souboru byla ručně provedena anotace na morfologické a analytické rovině, při níž byla vyznačena vlastní jména (v rámci obou rovin). Byla k tomu využita přípona lemmatu, umožňující specifikovat také typ vlastního jména, případně speciální atribut uzlu a-roviny (více o anotaci v sekci 1.2.3).

Výše popsaná technika je značně omezená, zejména v případě víceslovných entit. Proto byl pro potřeby automatického zpracování pojmenovaných entit v rámci projektu *Integrace jazykových zdrojů za účelem extrakce informací z přirozených textů*[6] navržen nový způsob jejich značkování a klasifikace. Každá entita má dle tohoto schématu svůj typ, nadtyp a rozsah. V textu je uzavřena mezi dvojici značek < a > spolu se svým typem, který jednoznačně určuje i nadtyp. Entity se do sebe mohou zanořovat, případně mohou být sdruženy do kontejneru, souvisí-li spolu. Ukažme si konkrétní příklad:

Příklad 13:

```
Mluvčí společnosti <P<pf Jaroslav> <ps Krejčí>> nám poskytl
rozhovor.
```

4 Technická zpráva [4], str. 13
5 Technická zpráva [4], sekce 2.7
6 Technická zpráva [4], str. 8

V tomto textu byla vyznačena dvojice entit - *Jaroslav* typu *křestní jméno* a *Krejčí* typu *příjmení* (obojí se společným předkem *jména osob*). Tyto dvě entity jsou sdruženy do kontejneru typu *jména osob*.

Seznam všech nadtypů je uveden v tabulce 2. Kompletní seznam všech 63 typů entit, typů kontejnerů a pomocných značek lze najít v Technické zprávě [4].

Symbol	Nadtyp
a	Čísla jako součásti adres
c	Součásti bibliografických údajů
g	Geografické názvy
i	Názvy institucí
m	Názvy médií
n	Čísla se specifickým významem
o	Názvy věcí
p	Jména osob
q	Čísla s významem počtu a pořadí
t	Časové údaje

Tab. 2: Seznam nadtypů entit

Výše uvedená klasifikace je dnes zcela běžným způsobem anotace pojmenovaných entit v českých textech. Prošla již jedním významným rozšířením, které bylo motivováno zkušenostmi vzešlými z anotace reálných textů.

2.4.2 CNEC

Czech Named Entity Corpus (CNEC) [5] je soubor obsahující ve verzi 2.0 cca 9 000 českých vět s vyznačenými pojmenovanými entitami. Anotace těchto vět proběhla ručně dle schématu popsaného v sekci 2.4.1, entity jsou tedy zařazeny do rozsáhlé dvouúrovňové hierarchie.

CNEC 2.0 je hlavním zdrojem pojmenovaných entit použitých pro testovací a ladící účely při konstrukci normalizační procedury. Více o volbě testovacích dat a statistikách provedených za tímto účelem viz sekce 3.3. V příloze G jsou dále shrnuty poznatky plynoucí z práce s korpusem.

2.4.3 Automatické rozpoznávání v českém jazyce

Pro český jazyk byly vyvíjeny také systémy NER. Obecně lze říct, že je jejich úspěšnost horší než úspěšnost systémů NER v angličtině. Toto lze snadno demonstrovat hodnotou F-measure (viz sekce 2.3). Nejlepší systémy pro anglický jazyk dosahují F-measure okolo 90%.

V současnosti nejlepším systémem NER pro český jazyk je NameTag [6]. Algoritmus, na němž je založen, je popsán v dokumentu *A New State-Of-The-Art Czech Named Entity Recognizer* [7]. Jednotlivé entity NameTag také zařazuje do

kategorií podle klasifikace popsané v sekci 2.4.1. Český model použitý v NameTagu byl trénován na CNEC (viz 2.4.2) ve verzích 1.1 a 2.0. Úspěšnost NameTagu dle F-measure je 82.82%[7], čímž výrazně překonává všechny ostatní současné systémy NER pro český jazyk.

NameTag byl použit při generování testovacích dat pro tuto práci, více viz kapitola 5.

7 Údaj platí pro klasifikaci entit pouze dle nadtypů, tedy kategorií uvedených v tab. 2. Správnost určení konkrétního typu se pro toto měření nebrala v potaz.

3 Normalizace entit v českých textech

3.1 Normalizovaný tvar entity

Oproti angličtině, s níž pracuje většina projektů zabývajících se strojovým zpracováním pojmenovaných entit, má čeština (a další podobné flektivní jazyky) významnou nevýhodu. Díky skloňování se může stát, že více různých entit označených v textu má stejný význam. Jde vlastně jen o různé tvary téže entity podle toho, v jakém pádě a čísle se vyskytuje. Například entity *krajským soudem v Ostravě*, *krajského soudu v Ostravě* či *krajskému soudu v Ostravě* zřejmě označují jednu jedinou instituci zvanou *krajský soud v Ostravě*, ale jejich tvar závisí na větě, v níž byly použity.

Chceme-li provádět klasifikaci textů právě podle pojmenovaných entit, je v češtině velmi důležité těmto entitám přiřadit **normalizovaný tvar**, který bude možné jednoznačně spojit s každým výskytem téže entity. Právě podle tohoto tvaru se budou následně řídit všechny systémy pro extrakci dat a strojové zpracování a bude podle něj možno např. vyhledávat, zjišťovat podobnosti v textech apod. Proces převodu do této neutrální formy pak nazveme **normalizací pojmenované entity** a jde o problém, jímž se budeme zabývat ve zbytku tohoto textu.

Otázkou tedy nyní je, který tvar pojmenované entity zvolit jako normalizovaný. Při návrhu vyjdeme z již existujícího procesu používaného při klasifikaci neutrálních forem jednotlivých slov pro potřeby skloňování a časování. Tomuto procesu se říká lemmatizace a jeho výsledkem je tzv. lemma. Jde o převedení daného slovního tvaru do prvního pádu čísla jednotného (v případě jmen), či do infinitivu (v případě sloves) - podrobněji viz 1.2.2.

Lemma může velmi dobře posloužit jako normalizovaný tvar pojmenované entity tvořené jediným slovem. Entita ale může sestávat z více po sobě jdoucích slov (pak hovoříme o entitě víceslovné) a v tom případě lemma k dispozici nemáme. Zcela přímočaré řešení, jež se samo nabízí, je prohlásit za normalizovaný tvar prosté spojení lemmat. V případě dříve uvedené entity *krajským soudem v Ostravě* bychom dostali spojení *krajský soud v Ostrava*. Tento tvar by však netvořil jednoznačný identifikátor - např. entity *strýc Petra* a *strýc Petr*, označující dvě různé osoby, by jej sdílely. Navíc je cílem, aby tento identifikátor odpovídal reálnému použití dané entity.

Za normalizovaný tvar dané entity budeme tedy dále považovat její přirozený základní tvar, který bude pro uvedenou entitu *krajský soud v Ostravě*. U některých entit vyskytujících se v množném čísle se naskytne otázka, zda bude základní tvar v čísle jednotném či zda si zachová své původní číslo. Tímto problémem se budeme podrobněji zabývat dále v textu. Prozatím můžeme uvažovat normalizované tvary zachovávající původní číslo dané entity.

3.2 Automatická normalizační procedura a její varianty

Cílem zbylé části této kapitoly bude navrhnout pravidlovou proceduru pro automatickou normalizaci pojmenovaných entit v českých textech s maximální možnou úspěšností.

Tato procedura bude existovat ve dvou základních variantách:

- Normalizace samostatné pojmenované entity
- Normalizace pojmenovaných entit vyznačených v textu

První varianta bude na vstupu předpokládat textový řetězec reprezentující jednu pojmenovanou entitu sestávající z libovolně mnoha slov a v libovolném tvaru. Na výstupu potom vrátí normalizovaný tvar této entity. Vezměme jako vstup např.:

```
Kulturním domě v Dolním Benešově
```

Procedura by měla vrátit následující výstup:

```
Kulturní dům v Dolním Benešově
```

Druhá varianta procedury obdrží textový řetězec reprezentující souvislý úsek textu, v němž budou vyznačeny pojmenované entity nějakou formou anotace (definovanou konkrétní implementací). Na výstupu pak bude tentýž úsek textu, kde budou anotace jednotlivých entit rozšířeny o atribut specifikující normalizovaný tvar. Uveďme příklad vstupu pro konkrétní způsob anotace:

```
<doc>Nebezpečí nástupu diktatury v <ne>Sovětském svazu</ne> stále
trvá.</doc>
```

Výstupem pak bude:

```
<doc>Nebezpečí nástupu diktatury v <ne normalized_name="Sovětský
svaz">Sovětském svazu</ne> stále trvá.</doc>
```

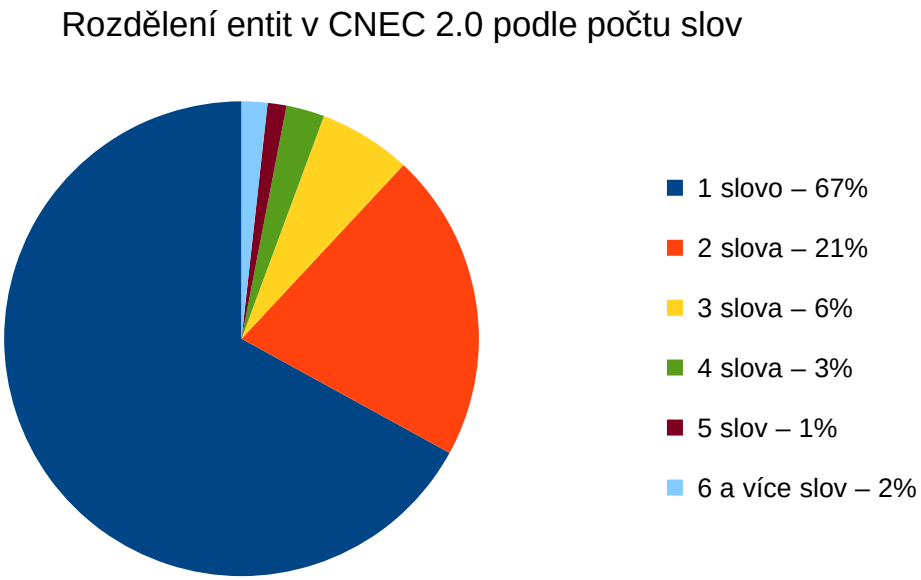
3.3 Použitá data, ruční normalizace

Pro konstrukci normalizační procedury byla zvolena následující strategie - vznikne množina ladících a množina testovacích dat. Ladící data budou tvořena výrazně menší množinou a budou použita při vytváření jednotlivých pravidel a při tvorbě samotné procedury. Tento vzorek by měl podchytit co největší škálu různých typů pojmenovaných entit, různých kontextů a tvarů. Teoreticky by výsledný algoritmus měl vypadat tak, že je schopen normalizovat každou entitu z této ladící množiny.

Testovací data budou sestávat z výrazně většího množství entit a na jejich základě bude poté vyhodnocena úspěšnost již hotové normalizační procedury. Tím se bude zabývat kapitola 5, zde se zaměříme pouze na prvotní zpracování ladících dat.

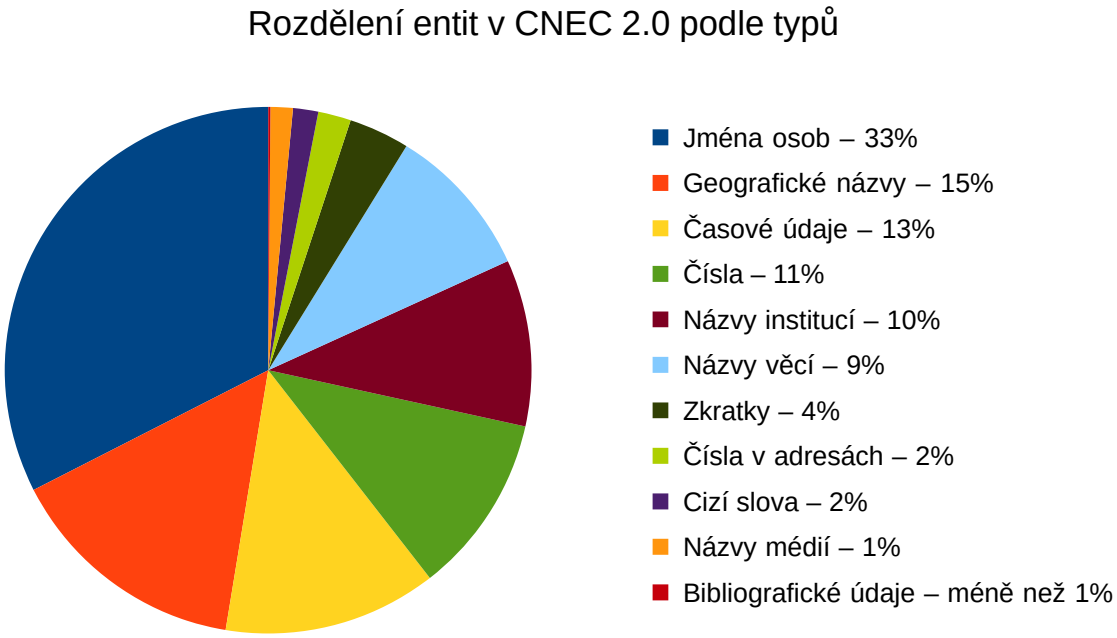
Jako základní zdroj dat byl zvolen soubor CNEC (viz 2.4.2). Obr. 3 a 4 zobrazují

statistiku[8] jednotlivých typů a délek (co do počtu slov) pojmenovaných entit v tomto souboru. Je vidět, že s počtem slov výrazně ubývá počet entit. Z pohledu konstrukce normalizační procedury jsou však zajímavé hlavně víceslovné entity, u nichž bude normalizace komplikovanější. Proto byly pro vzorek použity pouze troj a víceslovné pojmenované entity v CNEC.



Obr. 3: Diagram zobrazující rozdělení entit v CNEC 2.0 dle jejich délky

8 Statistika nezahrnuje entity speciálních typů *lower*, *upper*, *segm*, *cap*, *?*, *!*, zahrnuje však speciální typy *s a f*. Dále také zahrnuje veškeré kontejnery, které řadí mezi entity odpovídajícího typu. Skripty použité při generování této statistiky jsou k dispozici na přiloženém CD, viz přílohu F.



Obr. 4: Diagram zobrazující rozdělení entit v CNEC 2.0 dle jejich typu

Generováno[9] bylo celkem 5 vzorků obsahujících dohromady cca 800 různých pojmenovaných entit. Pro první dva vzorky se volily rovnoměrně entity ze všech nadtříd hierarchie použité v CNEC (viz 2.4.1), pro zbylé tři byly již vypuštěny nadtřídy *t*, *q* a *a*, tedy čísla ve významu času, čísla ve významu kvantit a čísla v adresách, u nichž normalizace nemá smysl.

Pro všechny vybrané vzorky byla nejprve provedena ruční normalizace. K uchovávání ručně i automaticky normalizovaných vzorků byl zvolen jednotný formát. Jde o textový soubor sestávající z po sobě jdoucích záznamů pro jednotlivé entity. Jeden takový záznam má následující podobu:

```
ID entity
Původní tvar entity
Normalizovaný tvar entity
- prázdný řádek -
```

ID entity je nějaký jednoznačný identifikátor v rámci souboru, z nějž entita pochází. V případě CNEC je to ID uzlu v treex formátu dat (viz 1.2.4). Příklad takového reálného záznamu z CNEC 2.0:

Příklad 14:

```
n_tree-cs-s1953-n50096
Úřadu vlády SR
Úřad vlády SR
```

9 Skripty použité při generování jednotlivých vzorků i vzorky samotné jsou k dispozici na přiloženém CD, viz přílohy C a F.

Při ruční normalizaci ladících dat bylo možné získat hrubou představu o tom, jak normalizace entit probíhá a jaké techniky jsou při ní použity, což poskytlo základ k proceduře rozebírané v následujících částech textu. Uvedeme několik postřehů plynoucích z normalizace všech uvedených vzorků:

- Většina entit se v textu vyskytuje již v normalizovaném tvaru
- Velké množství entit je číselného charakteru, zde normalizace nemá smysl
- Velké množství entit i v českém textu je v cizím jazyce
- Většina nečíselných víceslovných entit má podobu rozvitých podstatných jmen
- Malé množství entit tvoří příslovečná určení či přívlastky
  - Např: *Ze zásuvky a bloku*, *Červený a černý*
- Velmi malé množství entit tvoří slovesná spojení
  - Např: *Bratříčku , zavírej vrátka*

3.4 Vstupní předpoklady automatické normalizace

Automatická normalizace bude algoritmický proces, který bude zpracovávat větnou strukturu českého textu. K tomu je potřeba, aby tento text (věta či slovní spojení) byl řádně zanalyzován co do tvarosloví a větné struktury a aby byl na základě této analýzy také anotován.

Předpokládáme tedy schopnost nad textem provést anotaci na m-rovině a na a-rovině (viz 1.2). V samotné proceduře již budeme využívat možnost procházet závislostní strom dané entity a pro jednotlivé slovní jednotky také zjišťovat jejich lemma a tag. Z hlediska zpracování a úprav anotovaného textu bude také nutná schopnost morfologického generování. Budeme tedy také předpokládat, že pro zadané lemma a tag jsme schopni toto generování provést. Konkrétní detaily vyřešíme až v kapitole 4.

3.5 Základní koncept algoritmu

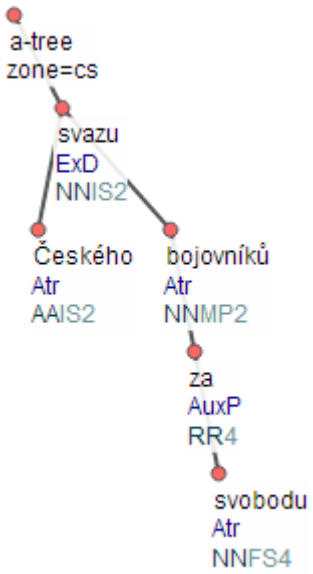
Nyní tedy předpokládejme, že máme na vstupu samostatnou pojmenovanou entitu. Zkonstruujeme základní proceduru pro normalizaci této entity. Výstupem bude opět pouze samotný normalizovaný tvar.

Při ručním zpracování víceslovných entit se prokázalo, že jde povětšinou o rozvitá podstatná jména. Příkladem mohou být entity jako *Český svaz bojovníků za svobodu* (jméno *svaz* je rozvito atributy *Český* a *bojovníků*), či *Taneční studio pardubického divadla*. Nyní zkusme některou z těchto entit použít ve větě v roli předmětu, tedy v jiném než prvním pádě:

Příklad 15: *Dlouholetý člen Českého svazu bojovníků za svobodu poskytl našemu časopisu exkluzivní rozhovor.*

V takovéto větě v reálném textu bude za entitu označeno spojení *Českého svazu bojovníků za svobodu*. Po provedení anotace na m-úrovni a a-úrovni získáme

závislostní strom[10] zachycený na obr. 5. Pro potřeby algoritmu normalizace budeme ignorovat univerzální kořen stromu a za kořen budeme považovat až jeho potomka (případně potomky). Později se nám totiž stane, že normalizační proceduru budeme muset spouštět na podstromy závislostních stromů, kde tento univerzální kořen již k dispozici mít nebudeme.



Obr. 5: Závislostní strom entity „Českého svazu bojovníků za svobodu“

V kořeni stromu je základní prvek entity - podstatné jméno *svazu*. Toto podstatné jméno je ve druhém pádě a pro normalizaci musí být převedeno do tvaru prvního pádu. Zbylé charakteristiky zachováme. Dále je toto jméno rozvíjeno dvěma způsoby - shodně (větným členem nacházejícím se ve větě před ním samotným) a neshodně (větným členem nacházejícím se až za ním).

U shodně rozvíjejících větných členů musíme provést shodu v čísle, pádu a rodu. Proto převedeme *českého* také do prvního pádu, zbylé vlastnosti už shodné jsou. Korektní shoda musela totiž existovat již před zahájením normalizace a my jsme ji mohli rozbít pouze normalizací kořene, tedy změnou jeho pádu.

Naopak neshodně rozvíjející člen shodu s řídícím členem mít nebude. Tvar takovýchto členů se při skloňování jejich řídícího prvku (a tedy i celé entity) nemění, proto s nimi nemusíme nic dělat.

Aplikací popsaných kroků dostáváme přesně normalizovaný tvar entity:

českého svazu bojovníků za svobodu

10 Všechny závislostní stromy použité v rámci práce jsou generovány reálným parserem. Nezobrazují tedy skutečný strom entity, ale strom, jež bude mít k dispozici normalizační procedura. Často jsou v nich (stejně jako v tomto případě) chybně určeny závislostní funkce.

českého svaz bojovníků za svobodu

český svaz bojovníků za svobodu

Nyní tento postup dále zobecníme - je-li jakýkoliv větný člen dále rozvíjen, pak po opravě jeho shody zpracujeme jeho potomky stejně jako potomky kořene. Získáváme tímto postupem rekurzivní proceduru:

```
Procedura NormalizujUzel X
  pokud X je kořen
    převeď X do prvního pádu
  pokud X není kořen a předchází ve větě svého otce
    ProveďShodu X a otec(X)
  na všechny potomky X postupně spusť proceduru NormalizujUzel
```

Po její aplikaci na kořen závislostního stromu pojmenované entity získáme závislostní strom normalizovaného tvaru entity. Poté již stačí z tohoto stromu zpětně zrekonstruovat text např. průchodem jednotlivých uzlů stromu zleva doprava.

Popsaný algoritmus bude výchozí variantou procedury automatické normalizace zpracovávající samostatnou entitu. Jde o proceduru rekurzivní, procházející závislostní strom a upravující jej podle určitých pravidel. Výchozí pravidla jsou tedy dvě:

(1) Pokud je uzel kořenem, převeď jej do prvního pádu

(2) Pokud uzel není kořen a předchází svého otce (tj. je jeho levým synem), proveď s ním shodu v pádě

Základní kostru průchodu už dále měnit nebudeme, upravovat budeme pouze množinu pravidel, jež budou na každý uzel aplikována.

Doplňme ještě způsob, jakým se bude provádět shoda:

```
Procedura ProveďShodu X a Y
  uprav pád v tagu uzlu X podle tagu uzlu Y
  generuj tvar pro lemma uzlu X a tag uzlu X
  uspěje-li generování
    nastav obsah uzlu X na vygenerovaný tvar
  jinak
    vrať tag uzlu X do původního stavu
```

Tedy nepodaří-li se vygenerovat odpovídající tvar pro shodu, ponecháme tvar původní.

3.6 Složitější případy a jejich řešení

V této části si rozebereme jednotlivé případy, v nichž základní procedura z předchozí sekce selhává, a přidáme nová pravidla, která budou tyto situace řešit. Zdrojem těchto případů budou většinou použité ladíci vzorky dat (viz 3.3).

3.6.1 Rozlišení shodného a neshodného přívlastku

V úvodním příkladu jsme pro rozlišení toho, zda je větný člen rozvíjen shodně či neshodně, použili pouze jeho pozici vůči členu rozvíjejícímu. Tvrdili jsme, že shodně rozvíjející člen se nachází před členem řídícím, neshodně rozvíjející člen pak až za ním. V češtině je toto schéma velice obvyklé, ale existují také výjimky. Někdy totiž může shodný přívlastek stát i za členem řídícím, přesto se s ním musí v čísle i pádu shodovat. Uveďme si několik příkladů, které jsou celkem běžné:

Příklad 16: *Spojených států amerických*

Příklad 17: *kyselinou sírovou*

Příklad 18: *medvěda hrajícího na housle*

Ve všech těchto případech se podtržený přívlastek musí shodovat v rodu, čísle i pádu s řídícím podstatným jménem ležícím před ním. Závislostní funkce podle Pražského závislostního korpusu nemá žádné speciální hodnoty pro rozlišení shodnosti a neshodnosti přívlastku, všechna podtržená slova jsou v závislostním stromě jen atributy. Proto musíme situaci vyřešit jinak. Podíváme se na jednotlivé slovní druhy a na situace[11], za kterých rozvíjí přívlastek shodně a neshodně. Je důležité podotknout, že čeština je jazyk s velice volnými pravidly a mnoha výjimkami, proto se nám nejspíš nepodaří podchytit všechny možné případy.

Přívlastek v české větě vždy rozvíjí podstatné jméno a je nejčastěji reprezentován **jménem přídavným**. Jak jsme již viděli v předchozím příkladu, přídavná jména shodně rozšiřující mohou stát před i za řídícím členem. Ve velmi malém počtu případů můžeme ovšem narazit i na přídavná jména rozšiřující neshodně. Příkladem mohou být názvy *Liga výjimečných* či *schůze pracujících* kde přídavné jméno v podstatě zastupuje podstatné jméno ve funkci přívlastku neshodného.

Velmi podobná situace nastává u **číslovek** a **zájmen**. Obvyklé je rozvíjení shodné předcházející řídícímu členu, např. *Tři mušketýři*, *druhý termodynamický zákon*, nebo *Ten nejlepší*. Objevíme také případy shodného rozvíjení za členem, např. *Karel pátý*. A příklady jako *pokoj osmnáct* a *osud všech* dokazují, že i tyto slovní druhy mohou rozvíjet řídící člen neshodně.

Podstatné jméno může být ve větě také přívlastkem. Vždy se ale vyskytuje až za řídícím členem, a to jak ve variantě shodné (*otec Goriot*), tak také ve variantě neshodné (*otec Goriota*, *hrad Kost*). Na podstatné jméno v roli přívlastku shodného v rámci pojmenovaných entit narazíme také velmi často u jmen a příjmení, např. *Jiřina Vysoudilová*.

Přívlastek může být vyjádřen také **příslovcem**, to je však slovní druh neohebný a proto u něj nemá smysl uvažovat shodnou variantu. Neshodná varianta se vyskytuje opět vždy až za rozvíjeným členem, např. *Místo nahoře*. Stejně se chová v roli přívlastku i **slovesný infinitiv** (*Umění lhát*) či předložkové vazby (*daň z příjmu*).

Otázkou nyní je, jak obecně identifikovat, kdy je přívlastek shodný. Z rozboru vyplynulo, že stojí-li přívlastek před rozvíjeným členem, bude vždy shodný. Proto

11 Inspirací byla dokumentace k PDT [1].

můžeme zachovat pravidlo (2). V případě přívlastku stojícího za členem je situace velmi komplikovaná. Neshodné přívlastky se mohou vyskytovat v různých pádech, ačkoliv nejobvyklejší je pád druhý. Shodný přívlastek se vyskytuje vždy ve stejném pádě jako řídící člen. Existují situace, kde bez znalosti kontextu nelze rozlišit, zda se jedná o shodný či neshodný přívlastek, např:

Příklad 19: *Znáš otce Goriota?*

Zde se může jednat jak o otce jménem Goriot, tak o otce jiné postavy tohoto jména.

Z předchozího vyplývá, že ať budou pravidla vypadat jakkoliv, všechny případy správně neznormalizujeme. Pokusíme se však o co možná nejlepší řešení použitím následujícího pravidla:

(3) Je-li uzel pravým potomkem a zároveň atributem tvořeným podstatným jménem, přídavným jménem, zájmenem či číslovkou a byl-li v původní větě ve shodě čísla, pádu a rodu se svým rodičem a následoval-li ve větě přímo po něm, považuj jej za levého potomka tohoto rodiče

S pravidlem (3) správně normalizujeme v podstatě všechny shodné přívlastky, nezávisle na jejich pozici vůči rodiči. Může se však stát, že budeme nesprávně považovat přívlastek neshodný za přívlastek shodný a dojde k chybné normalizaci.

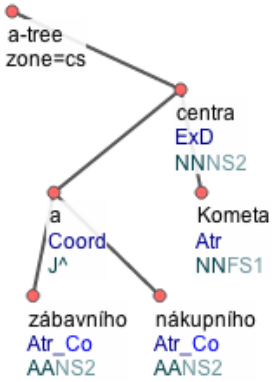
Dalším důsledkem pravidla (3) je nutnost pamatovat si původní formu (lemma a tag) každého uzlu a při provádění shody ji nepřepisovat, ale ukládat si ji stranou jako formu novou, z níž pak bude vygenerován výsledek.

3.6.2 Spojky a interpunkce

Prakticky jakýkoliv větný člen v rámci pojmenované entity může být násobný, oddělený spojkou, případně čárkou (v úvahu připadá i jiná interpunkce). V závislostním stromě takové entity poté všechna slova násobného členu závisí na spojce či interpunkci a teprve ta závisí na členu, který je ve skutečnosti rozvíjen. Uveďme si konkrétní příklad:

Příklad 20: *zábavního a nákupního centra Kometa*

Závislostní strom této entity najdeme na obr. 6. Jak vidíme, atributy *zábavního* a *nákupního* shodně rozvíjí slovo *centra*. Ve stromě jsou však potomky spojky *a*, podle stávající procedury by tedy probíhala shoda s touto spojkou, a to pouze u slova *zábavního*, jež spojce předchází. Ve stromové hierarchii je tedy nutné spojky a interpunkci přeskakovat.



Obr. 6: Závislostní strom entity „zábavního a nákupního centra Kometa“

Musíme si však také uvědomit, že všechny členy, které rozvíjí násobný člen, budou v závislostním stromě také potomky spojky či interpunkce. Např. ve spojení *obrazárna a muzeum zámku v Nelahozevsi* na spojce *a* závisí podstatná jména *obrazárna, muzeum* a *zámku*. Poslední zmíněné však rozvíjí obě předešlé, logicky tedy závisí na nich. Proto zkusíme identifikovat prvky násobného členu a zbylé potomky „pověsíme“ v hierarchii až pod ně.

Pro násobné členy je v PDT zavedena přípona závislostní funkce `_Co` (tzv. koordinace). Využijeme ji a zkonstruujeme následující pravidla:

(4) Je-li uzel spojkou či interpunkcí a není-li mezi jeho potomky žádný člen v koordinaci (závislostní funkce s příponou `_Co`), považuj všechny jeho potomky za:

- Levé potomky jeho otce, byl-li sám levým potomkem
- Pravé potomky jeho otce, byl-li sám pravým potomkem
- Kořeny, byl-li sám kořenem

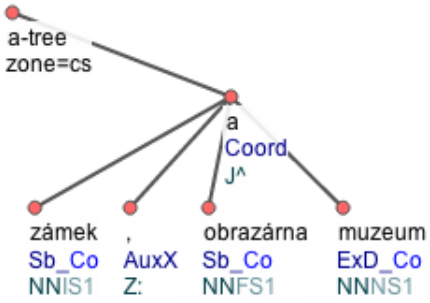
(5) Je-li uzel spojkou či interpunkcí a jsou-li mezi jeho potomky členy v koordinaci, považuj jeho levé potomky bez koordinace za levé potomky jeho prvního potomka s koordinací, jeho pravé potomky bez koordinace za pravé potomky jeho prvního potomka s koordinací a všechny jeho potomky s koordinací za:

- Levé potomky jeho otce, byl-li sám levým potomkem
- Pravé potomky jeho otce, byl-li sám pravým potomkem
- Kořeny, byl-li sám kořenem

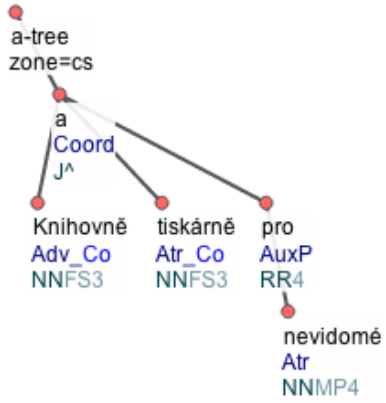
Pravidlo (4) není založeno na žádném reálném příkladu. Situace, kterou popisuje, by při korektním parsingu vůbec neměla nastat - každá spojka či interpunkce musí mít některé potomky v koordinaci. Je proto použito spíše pro úplnost, využito bude například v situacích, kdy parser nedoplní správně přípony závislostních funkcí.

Důsledně vzato může násobný člen sestávat z více než jen dvou slov, např. *zámek, obrazárna a muzeum*. Potom budou všechny členy záviset na jedné ze spojovacích

slovních jednotek (zde slovo *a*) a budou mít uvedenu koordinaci u závislostní funkce. Zbylé spojovací slovní jednotky (symbol čárky) budou záviset také na této spojovací jednotce, ale již bez koordinace (viz obr. 7). Pravidla (4) a (5) by měla korektně řešit i tyto situace.



Obr. 7: Závislostní strom spojení „zámek, muzeum a obrazárna“



Obr. 8: Závislostní strom entity „Knihovně a tiskárně pro nevidomé“

Podívejme se ještě na jeden příklad analogického problému:

Příklad 21: *Knihovně a tiskárně pro nevidomé*

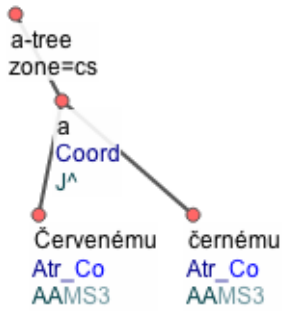
Závislostní strom této entity (obr. 8) má v kořeni spojku. Násobný je zde řídící člen, entita samotná je tvořena rozvitou dvojicí podstatných jmen. Potřebujeme tudíž normalizaci „spustit“ z obou těchto potomků. Pravidla (4) a (5) podobné situaci již řeší, konkrétně zde budou podle (5) oba uzly *Knihovně* a *tiskárně* vnitřně považovány za kořeny nových stromů.

3.6.3 Chybějící řídící člen

Některé pojmenované entity jsou tvořeny pouze přívlastkem či příslovečným určením, kterému chybí řídící větný člen. Mohou nastat dva různé případy -

chybějící člen může být rozvíjen shodně nebo neshodně. Podívejme se na reálný příklad demonstrující první variantu:

Příklad 22: *Červenému a černému*



Obr. 9: Závislostní strom entity „Červenému a černému“

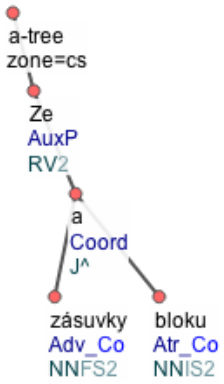
Jde o název známého literárního díla, vyskytující se však v textu ve třetím pádě. V kořeni závislostního stromu (obr. 9) máme spojku *a*, což pravidlo (5) převede na dva stromy s kořeny v obou přídavných jménech. Ty jsou poté normalizovány převedením do prvního pádu - zde je vše v pořádku. Jediný problém spočívá v taggingu takovéto entity - mnoho přídavných jmen má stejné tvary v různých pádech a rodech a bez řídícího členu není možné jednoznačně určit lemma a tag.

Varianta s neshodným rozvíjením chybějícího větného členu zahrnuje použití nějaké předložkové vazby. Například:

Příklad 23: *Ze zásuvky a bloku*

U této entity najdeme v kořeni (obr. 10) předložku. Ta jednoznačně určuje pády všech ostatních větných členů, entita jako celek se tedy neskloňuje. Není ani třeba ji normalizovat a proto zavedeme pravidlo:

(6) Je-li uzel kořenem a zároveň předložkou, ukonči normalizační proceduru



Obr. 10: Závislostní strom entity „Ze zásuvky a bloku“

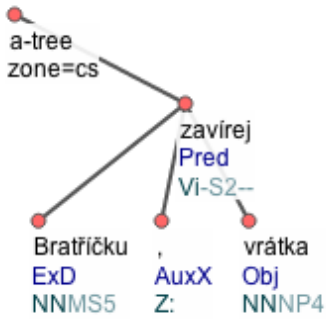
3.6.4 Slovesné konstrukce

Ve výjimečných případech může být obsahem pojmenované entity také sloveso v určitém tvaru. Půjde pak o plnohodnotnou větu. Jako příklad si uveďme následující entitu:

Příklad 24: *Bratříčku, zavírej vrátka*

Takováto entita se při použití ve větě nebude nijak skloňovat, proto s ní nebudeme nic dělat ani při normalizaci. V závislostním stromě celé jednoduché věty bude v kořeni přísudek (obr. 11), tedy jediný určitý slovesný tvar věty. Stačí tudíž přidat následující pravidlo:

(7) Je-li uzel kořenem a zároveň slovesem v určitém tvaru, ukonči normalizační proceduru



Obr. 11: Závislostní strom entity „Bratříčku, zavírej vrátka“

V případě souvětí bude v kořeni stromu spojka či interpunkce, což použití pravidla ze sekce 3.6.2 převede na množinu stromů odpovídajících předchozímu případu. Tato situace bude tudíž také korektně vyřešena.

3.7 Další pravidla a vylepšení

Dokončíme základní koncept pravidlové procedury pro normalizaci samostatné pojmenované entity přidáním několika pravidel a drobných úprav, které nesouvisí přímo s určitým jazykovým jevem, ale mohou obecně pomoci dosáhnout co nejlepších výsledků.

3.7.1 Zachování entit v normalizovaném tvaru

Normalizační procedura je vždy procesem, který může potenciálně skončit chybou, minimálně díky závislosti na správném parsingu textu, který sám o sobě nedosahuje stoprocentní úspěšnosti. Navíc v běžném textu se velmi významné množství entit vyskytuje již v normalizovaných podobách. Proto není žádoucí „pokoušet štěstí“ při opětovné normalizaci těchto entit. Zavedeme tedy následující pravidlo:

(8) Je-li uzel kořenem a jde-li o podstatné jméno v prvním pádě, ukonči normalizační proceduru

Z podobného důvodu, tedy abychom napáchali co nejméně škod, ukončíme normalizaci také v případě, že je kořenem stromu nerozpoznaný slovní druh. To signalizuje nějaký problém při taggingu a pravděpodobně také parsingu a bude lepší ponechat takovou entitu v původním stavu. Přidáme tedy pravidlo:

(9) Je-li uzel kořenem a jde-li o nerozpoznaný slovní druh (poziční značka má hodnotu 'X' pro slovní druh a '@' pro slovní poddruh), ukonči normalizační proceduru

3.7.2 Zachování velikosti písmen

Původní tvary slov v pojmenované entitě jsou při procesu normalizace často nahrazovány tvarem jiným, vygenerovaným pomocí morfologického generování. Tento vygenerovaný tvar nereflektuje velikost písmen v původním tvaru, což je často žádoucí, např. u velkého písmene na začátku názvu, či u slov psaných čistě velkými písmeny. Proto se pokusíme co možná nejlépe přenést základní schéma velikosti jednotlivých písmen původního tvaru do tvaru nového.

Upravíme proto algoritmus provádění shody. Po vygenerování tvaru nového vždy zkontrolujeme tvar původní a rozlišíme následující případy:

- Původní tvar byl psán velkými písmeny
  - Všechny znaky nového tvaru převedeme na velká písmena
- Původní tvar byl psán malými písmeny
  - Všechny znaky nového tvaru převedeme na malá písmena
- Původní tvar měl první písmeno velké a zbytek obsahoval alespoň jedno malé písmeno
  - Novému tvaru převedeme první znak na velké písmeno, zbytek ponecháme

3.7.3 Optimalizace algoritmu

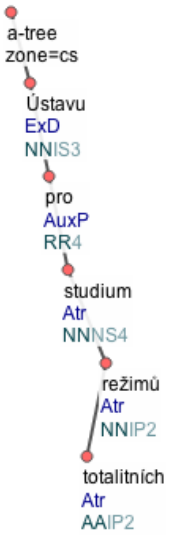
Předpokládáme-li, že máme na vstupu korektní pojmenovanou entitu v češtině, můžeme algoritmus do značné míry zoptimalizovat. Změny probíhající uvnitř stromu v podání jednotlivých pravidel se totiž propagují od kořene pouze prostřednictvím uzlů představujících shodné větné členy, či prostřednictvím spojek a interpunkce. Při průchodu stromem narazíme na první uzel reprezentující neshodný větný člen (mimo spojek a interpunkce), nemusíme jeho potomky již vůbec procházet, protože všechna pravidla by u nich způsobovala pouze zajištění shody s tímto neshodným větným členem (případně s jeho potomky). Tvar těchto členů se ale při normalizaci nijak nezměnil, předpokládáme-li tedy, že korektní shoda byla dodržena již v původním tvaru entity, naše procedura s žádným potomkem nic neudělá.

Můžeme proto přidat následující pravidlo:

(10) Je-li uzel pravým potomkem a nesplňuje-li podmínky pravidel (3), (4) a (5), ignoruj jeho potomky (tj. ukonči průchod touto větví stromu)

Jako příklad si vezměme následující entitu:

Příklad 25: *Ústavu pro studium totalitních režimů*



Obr. 12: Závislostní strom entity „Ústavu pro studium totalitních režimů“

Závislostní strom této entity najdeme na obr. 12. Při její normalizaci stačí převést do prvního pádu slovo *Ústav*. Zbylé větné členy rozvíjí neshodný přívlastek *pro studium*, případně jeho potomky, proto zůstanou v původní podobě. Normalizační průchod stromem tedy můžeme podle pravidla (10) zastavit hned v uzlu s předložkou *pro*, ušetříme tím zbytečný průchod zbylých tří uzlů.

3.8 Problémy a chyby

V sekcích 3.5 - 3.7 byla podrobně popsána základní procedura pro normalizaci samostatné pojmenované entity. Nyní se podrobněji podíváme na situace, s nimiž tato procedura může mít problém a vracet nesprávné či nejednoznačné výsledky, ať už kvůli nedokonalostem pravidel samotných, nebo kvůli problémům s procesy, na nichž procedura závisí.

3.8.1 Pluralita

Jak již bylo zmíněno v sekci 3.1, definice normalizované entity vede k zachování čísla u řídícího členu v našem algoritmu. Díky tomu se normalizované entity mohou vyskytovat jak v jednotném tak v množném čísle v závislosti na čísle jejich původního tvaru. Otázkou nyní je, zda je tento výsledek vždy správný.

Někdy je množné číslo nutnou součástí dané entity, např. *Spojené státy americké*. V tomto případě rozhodně musí být pluralita zachována, tvar *Spojený stát americký* je chybný. Na základě tohoto a podobných příkladů byla také definice normalizované entity zvolena tímto způsobem.

Na druhé straně však existuje mnoho jiných příkladů entit, které jsou v množném čísle proto, že označují větší množství věcí, které jsou ale principiálně entitou jednotnou. Příkladem může být entita *Pobytové zájezdy léto 2013*. Přirozenější by zde nejspíše byl normalizovaný tvar *Pobytový zájezd léto 2013*.

Obě uvedené situace od sebe však není možné jednoduše rozeznat a chyba při zachování plurality je přijatelnější, neboť alespoň nedostaneme zcela nesmyslný výsledek.

3.8.2 Cizojazyčné entity

Značné množství pojmenovaných entit v českých textech tvoří entity cizojazyčné. U těch většinou není žádoucí normalizovat, neboť nejsou v českém textu nijak skloňovány. Problémem však je, že ještě před aplikací normalizačního algoritmu se nad těmito entitami pokoušíme provádět anotaci pomocí taggeru a parseru pro český jazyk. Je tedy téměř jisté, že výsledkem bude zcela nesmyslný strom. Ve většině případů bude kořen tohoto stromu mít v tagu jako slovní druh hodnotu 'X', tedy nerozpoznán, a podle pravidla z 3.7.1 bude normalizace ukončena, což bude korektní řešení.

Narazíme však také na zvláštní případy, kdy se cizí výrazy v textu skloňují, mají počeštěnou podobu. Pak se musíme spoléhat na to, že byly zavedeny do použitého morfologického slovníku. Může se pak stát, že tyto skloňované tvary cizího slova jsou spojeny s lemmatem českého ekvivalentu a výsledkem generování je pak úplně jiné slovo. Příkladem může být následující proces normalizace:

Příklad 26: *Cherry coke clubu Prostějov --> Cherry coke klub Prostějov*

3.8.3 Zkratky, speciální symboly, oddělovače

V pojmenovaných entitách se velice často vyskytují zkratky. Typickým příkladem

jsou např. *nám.*, *ul.* a podobné součásti adres. U těchto zkratek, jež jsou běžně používané a zažité, ale nejsou jednoslovné, obvykle selhává tokenizace[12]. Tečka je považována za oddělovač vět a např. spojení *nám. J. Berana* je tokenizováno jako dvě věty:

```
<sentence>
  <token>nám</token>
  <token>.</token>
</sentence>
<sentence>
  <token>J</token>
  <token>.</token>
  <token>Berana</token>
</sentence>
```

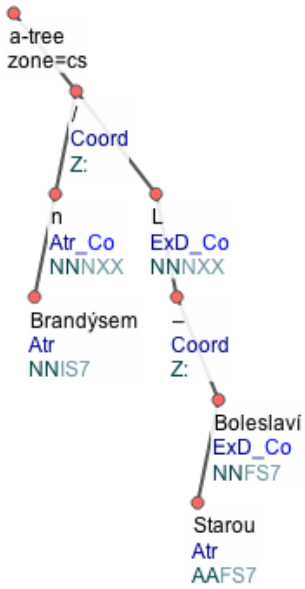
V tomto konkrétním případě pak navíc typicky tagging označí *nám* za tvar zájmena *my*. Při normalizaci potom vznikají opravdu bizarní tvary, z nichž se bohužel kompletně vytrácí původní význam, např *my. J. Berana.*

Podobné problémy způsobují další speciální symboly, používané např. v geografických názvech, jako jsou lomítka a pomlčky. Podívejme se na příklad opravdu komplikovaného názvu:

Příklad 27: *Brandýsem n / L – Starou Boleslaví*

U této entity dochází k selhání parsingu. Kořenem závislostního stromu by zde pravděpodobně měl být symbol pomlčky, jehož potomky by pak byly objekty *Brandýsem* a *Boleslaví* v koordinaci. Reálný výstup parseru[13] pak ukazuje obrázek 13. V hierarchii se na nejvyšší místa dostalo nesprávně spojení *n* / *L* tvořené zkratkami a spojovacím symbolem. Takto naparsovanou entitu pak naše pravidlová procedura nedokáže korektně normalizovat.

12 Testováno v Treex bloku W2A::CS::Tokenize a v MorphoDiTě (viz sekce 4.4 a 4.5).
13 Použit parser ParseMSTAdapted jako součást Treex



Obr. 13: Závislostní strom vygenerovaný pro entitu „Brandýsem n / L – Starou Boleslaví“

3.8.4 Obecné chyby parsingu a taggingu

Parsing věty při anotaci na a-rovině je proces, na němž staví celý náš algoritmus. Jeho úspěšnost však není stoprocentní. Pokud selže, pak má normalizační procedura k dispozici špatný závislostní strom a normalizace nebude provedena úspěšně. U běžných souvislých vět mají MSTParsery poměrně vysokou úspěšnost. Při zpracování samostatné entity je však nutné parsovat nevětnou strukturu, např. rozvité podstatné jméno (které samotné často také chybí, viz 3.6.3). Zde jejich úspěšnost rapidně klesá a chybně zpracované stromy se začínají objevovat poměrně často.

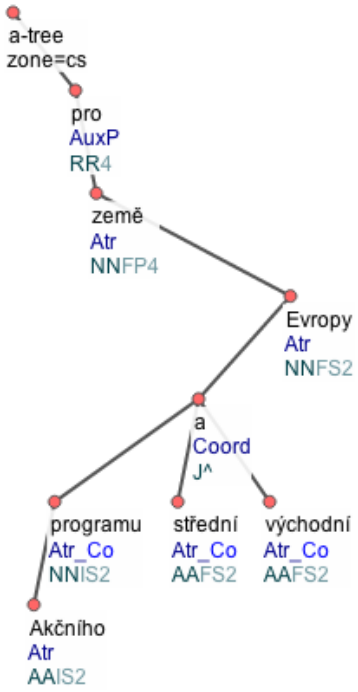
Velmi podobně je na tom tagging, jenž využívá větného kontextu k zjednoznačnění tagu a lemmatu pro daný tvar. Pokud je kontext nedostatečný, roste pravděpodobnost, že tagging vydá chybný výsledek. Díky tomu selhává normalizace i u entit, které zcela odpovídají doposud uvedeným schématům, např:

Příklad 28: *Akčního programu pro země střední a východní Evropy*

Příklad 29: *Plzeňských městských dopravních podniků*

Podívejme se důkladněji na příklad 28. Výstup taggeru a parseru[14] je zobrazen na obr. 14. V závislostním stromě se vůbec neodráží reálná struktura věty. Do kořene se dostala předložka *pro*, která by však měla rozvíjet větný člen *program*. Díky pravidlu (6) zastavujícímu normalizaci entit s předložkami v kořeni algoritmus okamžitě vrátí původní tvar jako normalizovaný. Ani vynechání pravidla (6) by nám v tomto případě příliš nepomohlo, neboť všechny důležité větné členy jsou v pravém podstromě kořenu a nebyly by tudíž stejně normalizovány.

14 Použit byl Treex scénář uvedený v sekci 4.4 obsahující ParseMSTAdapdted a TagFeaturama



Obr. 14: Závislostní strom vygenerovaný pro entitu „Akčního programu pro země střední a východní Evropy“

Tento problém je bohužel na úrovni algoritmu normalizace samostatné entity neřešitelný. Dodejme jen, že oba příklady (28 a 29) vyřeší provádění parsingu a taggingu na celých větách obsahujících tyto entity.

3.9 Shrnutí normalizační procedury

Postupně jsme poskládali kompletní podobu pravidlové procedury pro normalizaci jedné pojmenované entity za použití jejího závislostního stromu. Nyní si tuto proceduru stručně shrneme.

K dispozici máme závislostní strom reprezentující danou pojmenovanou entitu. Může jít o samostatný strom získaný při parsingu této entity, případně o podstrom nějakého jiného závislostního stromu. Ke každému jeho uzlu dále potřebujeme znát tag, lemma a konkrétní formu, a to ve dvou kopiích, jednou pro původní hodnoty a jednou pro generování hodnot nových, výstupních.

Pravidlová procedura probíhá tak, že strom je procházen rekurzivně do hloubky a v každém uzlu je aplikována sada pravidel. Ty uvádíme v pořadí jejich výskytu v předchozím textu, žádné z nich by nemělo být závislé na pořadí provedení. Předpokládáme, že některé akce popsané v pravidlech mohou selhat, s čímž v algoritmu počítáme. Např. pravidlo (1) se pokusí převést do 1. pádu jakýkoliv kořen, pokud je jím slovní druh bez schopnosti rozlišovat pád (neohebný slovní druh či sloveso), tato akce selže, procedura však pokračuje dál.

3.9.1 Seznam všech pravidel

(1) Pokud je uzel kořenem, převeď jej do prvního pádu

(2) Pokud uzel není kořen a předchází svého otce (tj. je jeho levým synem), proveď s ním shodu v pádě

(3) Je-li uzel pravým potomkem a zároveň atributem tvořeným podstatným jménem, přídavným jménem, zájmenem či číslovkou a byl-li v původní větě ve shodě čísla, pádu a rodu se svým rodičem a následoval-li ve větě přímo po něm, považuj jej za levého potomka tohoto rodiče

(4) Je-li uzel spojkou či interpunkcí a není-li mezi jeho potomky žádný člen v koordinaci (závislostní funkce s příponou _Co), považuj všechny jeho potomky za:

- Levé potomky jeho otce, byl-li sám levým potomkem
- Pravé potomky jeho otce, byl-li sám pravým potomkem
- Kořeny, byl-li sám kořenem

(5) Je-li uzel spojkou či interpunkcí a jsou-li mezi jeho potomky členy v koordinaci, považuj jeho levé potomky bez koordinace za levé potomky jeho prvního potomka s koordinací, jeho pravé potomky bez koordinace za pravé potomky jeho prvního potomka s koordinací a všechny jeho potomky s koordinací za:

- Levé potomky jeho otce, byl-li sám levým potomkem
- Pravé potomky jeho otce, byl-li sám pravým potomkem
- Kořeny, byl-li sám kořenem

(6) Je-li uzel kořenem a zároveň předložkou, ukonči normalizační proceduru

(7) Je-li uzel kořenem a zároveň slovesem v určitém tvaru, ukonči normalizační proceduru

(8) Je-li uzel kořenem a jde-li o podstatné jméno v prvním pádě, ukonči normalizační proceduru

(9) Je-li uzel kořenem a jde-li o nerozpoznaný slovní druh (poziční značka má hodnotu 'X' pro slovní druh a '@' pro slovní poddruh), ukonči normalizační proceduru

(10) Je-li uzel pravým potomkem a nesplňuje-li podmínky pravidel (3), (4) a (5), ignoruj jeho potomky (tj. ukonči průchod touto větví stromu)

3.9.2 Příklad normalizace samostatné entity

Ukážeme si přesný průběh normalizace jedné konkrétní samostatné pojmenované entity:

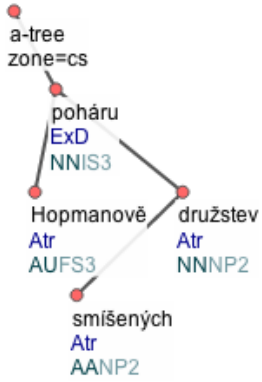
Příklad 30: *Hopmanově poháru smíšených družstev*

Závislostní strom této entity najdeme na obr. 15. Procedura zahájí průchod stromem v kořeni, tj. v uzlu se slovem *poháru*. Postupně vykoná následující kroky:

1. V kořeni podle pravidla (1) převede tvar *poháru* na *pohár* a pokračuje do jeho

levého potomka.

2. V levém potomkovi převede dle pravidla (2) tvar *Hopmanově* na *Hopmanův*. Uzel nemá vlastní potomky, proto se vrací do kořene a pokračuje do jeho pravého potomka.
3. V pravém potomkovi (uzel se slovem *družstev*) kontroluje dle pravidla (3) shodu mezi tvary *poháru* a *družstev* - shoda neplatí, jelikož první tvar je v singuláru a druhý v plurálu. Proto je aplikováno pravidlo (10) a normalizace je ukončena.
4. Procedura vrací výsledný tvar *Hopmanův pohár smíšených družstev*



Obr. 15: Závislostní strom entity „Hopmanově poháru smíšených družstev“

3.10 Použití kontextu při normalizaci

Podívejme se nyní na několik příkladů pojmenovaných entit tak, jak byly extrahovány z reálného textu (zdrojem je opět CNEC 2.0):

Příklad 31: *Americké společnosti stavebních inženýrů*

Příklad 32: *Evropské banky pro obnovu a rozvoj*

Příklad 33: *Centrální etické komise*

Příklad 34: *Zpravodaje Muzea Prostějovska*

Společným rysem těchto entit je, že bez znalosti kontextu, v němž se nacházely, nemůžeme rozhodnout, zda jsou v normalizovaném tvaru či nikoliv. Pro velké množství podstatných jmen napříč všemi rody v češtině platí, že jejich tvar v prvním pádě plurálu je shodný s tvary některých pádů singuláru. Například u slova *společnost* je tvar *společnosti* společný pro první a čtvrtý pád plurálu a druhý, třetí a šestý pád singuláru. Je-li prezentovaný tvar v původním kontextu skutečně plurálem, pak jde o tvar normalizovaný a není třeba jej měnit. Pokud však půjde o libovolný pád singuláru, je nutné tvar normalizovat převedením do prvního pádu singuláru.

Na základě citu pro jazyk a znalosti dostupných reálií asi většina lidí správně usoudí, že ani jedna z prezentovaných entit v plurálu není, není tedy také ani normalizována. Striktně vzato však může jít o názvy, které budou v rozporu s touto přirozenou intuicí, takže není možné činit na jejím základě jakékoliv závěry. Navíc platí, že běžné nástroje provádějící morfologickou analýzu textu obvykle mají tendenci takovýmto nejasným případům přiřazovat spíše první pád plurálu, neboť je tento neutrální tvar pravděpodobnější (a technicky vzato také jediný správný) pro rozvité jméno stojící samostatně mimo větu.

Příkladem je výstup procedury tagging nástroje MorphoDiTa [8] na entitu *Centrální etické komise* (příklad 33):

Token	Lemma	Tag
Centrální	centrální	AAF**P1**----1A----
etické	etický	AAF**P1**----1A----
komise	komise_,h_,x	NNF**P1**-----A----

Tag všech slov obsahuje sekvenci P1 (ve výstupu zvýrazněna), tvar byl tedy identifikován jako první pád plurálu.

V podstatě jediným faktorem, který nám umožní správně identifikovat použitý tvar, je analýza větného kontextu, ve kterém se entita v původním textu nacházela. Z celé věty již dokážeme v naprosté většině případů rozpoznat pluralitu a případně i pád dané entity. Podívejme se například na část věty, z níž pochází entita *Centrální etické komise*:

Příklad 35: *Jeho nedůvěru v instituce sdílí i předseda Centrální etické komise Jan Payne, ...*

Při analýze této věty zjistíme, že slovo *komise* rozvíjí slovo *předseda*. Ze znalosti vazeb v češtině je pak jasné, že komise musí být ve druhém pádě singuláru.

Podobně jsou schopny tvar rozeznat i taggery, např. opět MorphoDiTa:

Token	Lemma	Tag
Jeho	jeho_^(přivlast.)	PSXXXZS3-------
nedůvěru	nedůvěra	NNFS4-----A----
v	v-1	RR--4----------
instituce	instituce	NNFP4-----A----
sdílí	sdílet_:T	VB-S---3P-AA---
i	i-1	J^-------------
předseda	předseda	NNMS1-----A----
Centrální	**centrální**	**AAFS2----1A----**
etické	**etický**	**AAFS2----1A----**
komise	**komise_,h_,x**	**NNFS2-----A----**
Jan	Jan_;Y	NNMS1-----A----
Payne	Payne_;S	NNMS1-----A----

Zde je již korektně určen tvar S2 (ve výstupu zvýrazněn), tedy druhý pád singuláru.

Analýzu celé věty s pojmenovanou entitou můžeme použít pouze ve scénáři, v němž normalizujeme entity vyznačené v souvislém textu. V jednodušším případě, kdy je vstupem pouze samostatná entita, tento postup aplikovat nemůžeme.

Výše popsaný problém se vyskytuje ještě v jednom případě, konkrétně u vlastních jmen a jejich mužských a ženských forem. Například pro tvar *Karla* či *Pavla* nejsme schopni určit, zda se jedná o první pád ženské formy jména, nebo o pád druhý formy mužské. Dojde-li v tomto případě k omylu při morfologické analýze, je špatně určen nejen tag daného tvaru, ale také jeho lemma, což výrazně zkomplikuje jakékoliv snahy o dodatečnou nápravu problému. Platí však opět, že zohlednění kontextu při morfologické analýze výrazně zvýší pravděpodobnost správného určení.

3.11 Rozšíření algoritmu pro entity vyznačené v textu

Nyní se zaměřme na rozšíření stávající procedury pro normalizaci entit vyznačených v souvislém textu. Zcela přímočarým řešením by bylo entity v textu identifikovat, extrahovat je z něj a normalizovat jednu po druhé aplikací této procedury. V rozboru problémů jsme však několikrát narazili na situaci, kdy by mohla práce s kontextem zlepšit výsledky normalizace a umožnit jednoznačné výsledky. Proto pojmeme toto rozšíření o něco komplexněji.

Problémem u samostatných entit je parsing a tagging, který často nemá k dispozici dostatek informací pro to, aby dokázal správně část souvislého textu anotovat. Základní požadavek na toto rozšíření proto bude, aby provádělo anotaci textu na m-rovině a a-rovině nad celým textem, v němž se entita nachází. To přinese několik nových problémů:

- Pojmenovaná entita nebude tvořit samostatný závislostní strom, ale bude tvořena množinou uzlů ve stromě větším
- Tato množina uzlů bude zpravidla souvislým podstromem, ale ve výjimečných případech tomu tak být nemusí
- Uzly této množiny budeme muset pro každou entitu ručně identifikovat
- Na každý souvislý úsek patřící entitě budeme muset spustit naši proceduru `NormalizujUzel`, budeme muset sledovat její průběh a také ji včas zastavit
- Normalizovaný tvar entity budeme muset složit z množiny jejích uzlů

3.11.1 Zpracování vstupu, párování uzlů s entitami

Na vstupu očekáváme text s vyznačenými pojmenovanými entitami. Konkrétní způsob vyznačení entit ponechme na dané implementaci, pro naši potřebu předpokládejme, že již známe množinu počátečních a koncových indexů jednotlivých entit. Dále pro jednoduchost předpokládejme, že entity se v textu nepřekrývají, mohou do sebe však být vnořeny.

Prvním krokem bude provedení anotace vstupního textu na m-rovině a a-rovině. Získáme tím množinu závislostních stromů, jeden pro každou větu tohoto textu. Nyní musíme pro všechny uzly těchto stromů zjistit, zda leží v některých vyznačených entitách a případně ve kterých. Budeme postupně každý uzel párovat s odpovídajícím úsekem původního textu a zjišťovat, zda do tohoto úseku nezasahuje vyznačení nějaké pojmenované entity.

Začneme průchodem závislostních stromů jednotlivých vět. Každý z nich projdeme podle očíslování jejich uzlů zleva doprava. Toto očíslování odpovídá pořadí, v němž se tokeny patřící do těchto uzlů vyskytovaly ve výchozí větě. Při tom paralelně procházíme tuto větu v původním textu a kontrolujeme shodu jednotlivých znaků se znaky tokenu daného uzlu. Vynecháváme bílé znaky, neboť ty se při tokenizaci ztrácí. Pro každý uzel tedy získáme rozsah znaků, jimž odpovídá v původním textu, a můžeme snadno porovnat, zda náleží do některé pojmenované entity, a případně ji k tomuto uzlu poznačit. Je třeba podotknout, že vyznačená entita by měla vždy sestávat z celých tokenů, neboť není možné uzel odpovídající tokenu rozdělit mezi dvě entity, případně entitu přiřadit pouze jeho části.

3.11.2 Spouštění normalizační procedury

Máme tedy množinu stromů, z nichž každý má u uzlů vyznačeno, do kterých entit patří. Problémem je, že jeden uzel může být pro každou entitu, do níž náleží, normalizován do jiné podoby. Vezměme si následující úsek textu:

Příklad 36: *Budeme projíždět Ústím nad Labem.*

Vyznačíme-li jako entitu jak *Ústím nad Labem*, tak i *Labem*, chceme aby si uzel reprezentující slovo *Labem* v první z nich zachoval svůj původní tvar, ale ve druhé by jej měl nahradit tvarem *Labe*. Budeme tedy muset upravit vnitřní podobu závislostního stromu a ke všem uzlům si pamatovat množinu výsledných tvarů, jeden pro každou entitu, do níž patří.

Nyní musíme nad každou entitou spustit normalizační proceduru. K tomu je třeba vědět, který uzel dané entity je ve stromové hierarchii nejvýše (není-li entita tvořena souvislým podstromem, potřebujeme takový uzel znát pro každou souvislou část entity). Dále potřebujeme zajistit, že normalizační procedura bude ignorovat uzly, jež do entity nepatří.

Naše řešení bude spočívat v jediném průchodu každým stromem. Pro všechny uzly zavedeme množinu pomocných příznaků ukazujících, zda již byly pro danou entitu normalizovány - výchozí hodnota je negativní. Stromy poté jeden po druhém projdeme do hloubky a v každém uzlu provedeme následující akci:

- Pokud uzel patří do nějaké pojmenované entity a nebyl v ní již normalizován, spusť z tohoto uzlu upravenou proceduru `NormalizujUzel`

Úprava procedury spočívá v tom, že si při spuštění na kořen zapamatuje, do které entity náleží, a při průchodu podstromu bude brát v potaz jen uzly této entity. Pracovat pak bude v každém uzlu vždy s tvarem slova pro danou entitu. Dále bude každému uzlu, jímž prošla, nastavovat příznak, že již byl v dané entitě normalizován. Nesmíme zapomenout, že v případě ukončení normalizace jedním z pravidel ((6), (7), (8), (9)) či při vypuštění jednoho podstromu (pravidlo (10)) musíme dodatečně projít zbytek podstromu a pouze nastavit příznaky. Tím se bohužel značně zhoršuje efektivita optimalizace uvedené v 3.7.3.

Popsaný algoritmus zajistí, že každý podstrom náležící nějaké entitě bude normalizován pouze jednou, a to zavoláním procedury `NormalizujUzel` na jeho kořen.

Často se stane, že entita je do více nesouvislých podstromů naparsována chybně. V těchto případech většinou spuštění procedury nad každým podstromem zvlášť vydá špatný výsledek. Alespoň částečně těmto problémům předejdeme, budeme-li každé dva podstromy příslušející téže entitě, které odděluje pouze spojka či interpunkce, považovat za jediný. Toho docílíme tím, že se v proceduře `NormalizujUzel` pro každý uzel se spojkou či interpunkcí mimo entitu, na který narazíme, podíváme, zda jeho potomkem není uzel téže entity. Pokud ano, s touto spojkou či interpunkcí zacházíme tak, jako by sama byla součástí entity.

3.11.3 Rekonstrukce textu entit

Na výstupu chceme ke každé vyznačené entitě přidat její normalizovaný tvar. Potřebujeme tedy postupně pro každou entitu získat její textovou reprezentaci. Proto si již při párování uzlů s entitami budeme ke každé entitě vytvářet seznam uzlů, které ji tvoří. Budeme-li chtít tuto entitu vypsat, setřídíme tento seznam podle očíslování uzlů a postupně budeme slepovat jednotlivé tokeny oddělené mezerami.

4 Implementace normalizátoru

4.1 Cíle implementace

V této sekci se budeme věnovat konkrétní implementaci výše popsaných procedur pro normalizaci pojmenovaných entit. Smyslem bude získat sadu aplikací, které bude možné prakticky použít k provedení normalizace při práci s českými texty.

Cílem je tedy implementovat následující:

- Command-line nástroj pro normalizaci samostatné entity
- Command-line nástroj pro normalizaci entit anotovaných v textu
- Webové rozhraní umožňující normalizaci samostatné entity
- Webové rozhraní umožňující normalizaci entit anotovaných v textu

Webové rozhraní by mělo sloužit jako nadstavba command-line nástrojů, nemělo by nijak zasahovat do funkcionality samotné. Oba zmíněné command-line nástroje reprezentují přímo dvě normalizační procedury představené v sekci 3.2 a popsané dále v kapitole 3.

4.2 Volba nástrojů a platformy

Dvěma základními omezujícími podmínkami při volbě prostředí, pro které bude normalizátor vyvinut, je dostupnost již existujících nástrojů potřebných k provedení anotace textu na m-rovině a a-rovině (viz sekce 2.3) a možnost dalšího praktického využití normalizátoru v souvisejících projektech. Zejména druhé jmenované omezení prakticky vyžadovalo, aby byl projekt řešen v prostředí operačního systému Unix, neboť právě na této platformě je realizována většina projektů týkajících se počítačového zpracování českého jazyka na ÚFAL.

Pro maximalizaci kompatibility byla jako ideální prostředí zvolena Java – program napsaný v tomto jazyce a přeložený do bytecode může být spuštěn na jakékoliv platformě s nainstalovanou Java Virtual Machine (JVM), která je součástí Java Runtime Environment (JRE) přítomného na naprosté většině dnešních Unixových systémů. Java navíc umožňuje psát kód na poměrně vysoké úrovni, bez nutnosti starat se o technické detaily implementace. Poskytuje také rozsáhlou sadu snadno použitelných standardních knihoven. Výsledný kód je při správném objektovém návrhu dobře čitelný a udržovatelný.

Alternativy k jazyku Java byly dvě - jazyky C a Perl. C je velice nízkoúrovňovým jazykem s omezenou škálou standardních knihoven. Existují jeho kompilátory pro naprostou většinu dnes používaných platforem.

Hlavní nevýhoda jazyka C pro využití v řešení našeho problému je jeho těžkopádnost při zpracování textových řetězců. Všechny textové řetězce v tomto jazyce jsou konstantní, při jakékoliv manipulaci s nimi je třeba řešit uvolňování a

alokaci paměti, což vede k velice nepřehlednému a rozsáhlému kódu i pro vyřešení algoritmicky poměrně jednoduchého problému.

Výhodou jazyka C je pak rychlost výsledného programu způsobená tímto nízkoúrovňovým návrhem. Pro problém normalizace entit však rychlost není kritickým faktorem, neboť zpracovaný text musí vždy nejprve projít taggingem a parsingem, což jsou časově mnohem náročnější operace než následující normalizační algoritmus. Jazyk C byl tedy vyhodnocen jako nevhodný.

Další možností byl jazyk Perl. Jde o interpretovaný skriptovací jazyk, který je určen primárně pro unixové platformy. Poskytuje velmi mnoho funkcí pro snadné zpracování textových řetězců (např. práci s regulárními výrazy podobnou unixovým nástrojům jako je sed). Proto jej také využívá velké množství v současnosti používaných nástrojů pro strojové zpracování textu na ÚFAL. Jde však o jazyk dynamický a netypovaný, který příliš vhodný pro objektové programování. Z tohoto důvodu nebyl použit ani Perl.

V průběhu testování a vytváření programu bylo nutné vytvořit mnoho malých skriptů automatizujících určitou jednodušší činnost. Tyto skripty byly vytvořeny ve dvou jazycích - v Unix shellu a v Pythonu. Unix shell byl zvolen pro nutně přenositelné skripty provádějící spouštění hlavního normalizátoru, Python potom spíše pro jednorázové akce, zejména pro generování testovacích vstupů a zpracování dat z CNEC (viz příloha F).

4.3 Text s vyznačenými entitami

4.3.1 Volba vstupního formátu

Pro normalizaci entit v textu je nutné zvolit formát, jímž budou tyto entity v textu vyznačeny. V sekci 2.4.1 je uveden původní způsob značkování navržený pro české texty a hierarchii kategorií popsanou tamtéž. Jde také o jeden z formátů, v němž je šířen CNEC. Další možností je výstupní formát nástroje NameTag (2.4.3), tedy jednoduchý XML dokument s entitami uzavřenými mezi dvojice tagů `<ne>` a `</ne>`. I v tomto formátu je k dispozici CNEC.

Z důvodu kompatibility s NameTagem byl jako vhodnější formát vyhodnocen druhý zmíněný. Je možné jej zpracovat jako XML pomocí dostupných knihoven, navíc se v textu mohou vyskytovat i jiné značky, které budou prostě ignorovány.

4.3.2 Zpracování textu

Načtení textu z XML formátu probíhá pomocí DOM interface (více viz 4.4.1). Cílem je získat čistý text a množinu pozic jednotlivých entit v něm. Procházíme DOM strom do hloubky a kdykoliv narazíme na uzel obsahující text, připojíme jej na konec pomocného řetězce. Když při tom projdeme uzel reprezentující element `<ne>` směrem dolů, uložíme si k odpovídající entitě jako počáteční index aktuální délku pomocného řetězce. Při návratu tímto uzlem směrem nahoru potom připojíme aktuální délku jako koncový index a přidáme dvojici do množiny entit. Výsledkem je datová struktura obsahující textový řetězec zbavený všech značek a seznam jednotlivých entit reprezentovaných odkazy do tohoto řetězce na počáteční

a koncový znak. Každé entitě v této reprezentaci je navíc přidělen unikátní kladný celočíselný identifikátor.

Při průchodu uzly reprezentujícími elementy `<ne>` navíc do textu přidáváme mezery. Tím zajistíme, že po tokenizaci bude každá pojmenovaná entita samostatným tokenem. Pokud bychom to neudělali a entita nebude začínat na začátku tokenu, nebude uzel odpovídající tomuto tokenu k entitě vůbec přiřazen. Bohužel se může stát, že se přidané mezery projeví i na výstupu, ovšem jen v případě vnořených entit, např. pro vstup:

Příklad 37:

```
<doc><ne><ne>Jičín</ne>, město pohádky</ne></doc>
```

Na výstupu dostaneme:

```
<doc><ne normalized_name="Jičín , město pohádky"><ne
normalized_name="Jičín">Jičín</ne>, město pohádky</ne></doc>
```

Mezera před znakem čárky se totiž při tokenizaci promítne jako hodnota 0 atributu `no_space_after` v treex formátu a při rekonstrukci nadřazené entity je mezera před čárku znovu vložena. Text mimo atributy `normalized_name` je zachován v původní podobě, do něj se tudíž žádná změna nepromítne.

Výstup programu je vygenerován opět do stejného formátu serializací DOM struktury, do níž byly přidány atributy `normalized_name` k elementům `<ne>`.

4.4 Anotace na m-rovině a a-rovině pomocí Treex

K samotnému anotování textu na potřebných rovinách je nutné spustit nad vstupním textem řetězec několika procedur. Konkrétně se jedná o tokenizaci, tagging a parsing. Ke každé z těchto procedur existují pro české texty volně dostupné nástroje - bylo by tedy možné tento proces spouštět ručně. My však využijeme možnost provést anotaci pomocí jediného programu.

Treex [3] je komplexní a modulární platformou pro zpracování přirozeného jazyka. Je implementován v jazyce Perl a je složen z řady tzv. **bloků**, menších programových modulů určených k provádění jednotlivých akcí (anotace různého druhu, rozpoznávání entit apod.). Bloky jsou vždy aplikovány na pracovní dokument, který si Treex drží interně v paměti. Proces práce Treex je následující:

1. Pracovní dokument je načten pomocí jednoho z Reader bloků
2. Pracovní dokument je postupně modifikován soustavou bloků
3. Pracovní dokument je uložen pomocí jednoho z Writer bloků

Vstupem Treex je mimo názvu vstupního a výstupního souboru (je možná i práce se standardním vstupem a výstupem) tzv. **scénář**, neboli sekvence bloků, jež budou postupně spuštěny. V souladu s předchozím popisem procesu práce Treex by měl první blok být typu Reader a poslední blok typu Writer.

Uvedeme příklad Treex scénáře, který provede na textovém dokumentu anotace na m-rovině a a-rovině a uloží jej ve formátu treex (viz sekce 1.2.4):

```
Util::SetGlobal language=cs
Read::Text
W2A::CS::Segment

# m-layer
W2A::CS::Tokenize
W2A::CS::TagFeaturama lemmatize=1
W2A::CS::FixMorphoErrors

# a-layer
W2A::CS::ParseMSTAdapted
W2A::CS::FixAtreeAfterMcD
W2A::CS::FixIsMember
W2A::CS::FixPrepositionalCase
W2A::CS::FixReflexiveTantum
W2A::CS::FixReflexivePronouns

Write::Treex
```

Tento scénář je použit k anotaci v rámci implementace normalizační procedury.

Treex je využíván jako externí program. Je tedy vyžadováno, aby byl nainstalován na počítači, na němž budeme normalizační proceduru spouštět.

Vstup je Treex schopen číst buďto ze souboru, nebo ze standardního vstupu. Výstup do treex formátu dat je možný pouze do souboru. Proto je nutné, aby normalizátor s Treex komunikoval prostřednictvím pomocných souborů. Z důvodu možnosti paralelního běhu více procesů normalizátoru je potřeba zajistit, aby tyto soubory byly pro každou instanci normalizátoru unikátní. K tomu je využito funkce knihoven JDK umožňující vytvoření dočasného souboru omezeného pouze pro běh programu.

Ze vstupního souboru programu je tedy vygenerován dočasný soubor. Nad ním je spuštěn Treex jako externí proces, na jehož skončení normalizátor čeká. Podle návratové hodnoty pak normalizátor pozná, zda běh skončil úspěchem, a pokud ano, může načíst výstupní soubor, jehož název systém Treex automaticky odvozuje ze souboru vstupního (tedy toho dočasně vygenerovaného).

Hlavní nevýhodou Treexu je velmi pomalé načítání rozsáhlých modelů pro jednotlivé bloky. Díky tomu může typicky dokončení jednoho scénáře zabrat okolo minuty. Normalizátor pak musí tuto celou minutu čekat na výstup Treex, čímž je i doba jeho běhu výrazně prodloužena.

4.4.1 Formát treex a jeho zpracování

Výstupní soubor nástroje Treex je ve formátu treex (viz 1.2.4). Ten vychází z formátu PML [2], jenž je sám založen na jazyce XML. Dokument je v něm popsán prostřednictvím stromově hierarchicky uspořádaných elementů. K načtení všech informací získaných při anotaci textu tedy musíme umět zpracovávat formát XML.

Pro zpracování XML existují v Javě dvě základní rozhraní. Prvním z nich je SAX

(Simple API for XML), který funguje na principu sekvenčního průchodu dokumentem XML. Zpracování obsahu se pak děje v obsluze událostí vyvolaných v jednotlivých elementech. Hlavní výhodou tohoto přístupu je rychlost a šetření pamětí - celý dokument není třeba do paměti načítat naráz. My však v normalizátoru stejně informace z dokumentu potřebujeme do paměti načíst, takže dáme přednost druhému, jednoduššímu rozhraní.

Tím je DOM (Document Object Model), který zpracuje a načte celý dokument do paměti a vytvoří z něj strom strukturou odpovídající původnímu XML. Na stejném modelu fungují např. dnešní webové prohlížeče při reprezentaci HTML stránky v paměti. DOM potom poskytuje metody pro průchod tímto stromem, případně jeho změny. Zde si vystačíme s rekurzivním průchodem celým stromem.

Na základě DOM struktury treex dokumentu si vnitřně vygenerujeme závislostní strom dané věty, s nímž pak budou pracovat samotné normalizační procedury. Jednotlivé uzly získají z atributů uzlů v treex formátu obsah (původní slovo), lemma, tag a hodnotu závislostní funkce. Dále si k nim uložíme seznam potomků, aby se stromem dalo procházet do hloubky.

4.4.2 Spuštění Treex nad více dokumenty

Chceme-li normalizovat více samostatných entit naráz, můžeme prostě několikrát spustit normalizátor. Pak se ale bude pokaždé znovu inicializovat Treex, což zabere velké množství času. Výhodnější by bylo umožnit uživateli vložit do normalizátoru libovolné množství entit a nechat Treex je zpracovat naráz.

Treex tuto techniku naštěstí umožňuje - dá se, prostřednictvím parametru `lines_per_doc` reader bloku, nastavit, že každá řádka vstupního souboru má být interpretována jako nový dokument a zpracována zvlášť. Výsledky jsou v tomto případě uloženy do souborů, jejichž názvy jsou vygenerovány z názvu vstupního souboru a pořadového čísla dokumentu zarovnaného na 3 číslice. Např. pro vstupní soubor `treex_input.txt` budou výstupní soubory následující:

```
treex_input001.treex      treex_input002.treex      ...
```

Při implementaci normalizátoru samostatné entity tedy budeme počítat s tím, že na vstupu bude soubor s množinou entit (jedna na řádek). Treex budeme spouštět výše popsaným způsobem a v případě, že vygeneruje více než jeden soubor, zpracujeme postupně každý z nich a spustíme normalizační proceduru nad každým stromem.

4.5 Generování a MorphoDiTa

Pro generování (viz sekce 1.2.2) je využit open-source nástroj MorphoDiTa (Morphologic Dictionary and Tagger) [8]. Jde o nativní aplikaci v jazyce C++ pro platformy Unix a Windows. Funguje jednak jako samostatný command-line nástroj, ale také jako knihovna s rozhraním pro jazyky Java, Python a Perl. Právě první z těchto rozhraní je využito v implementaci našeho normalizátoru.

Nástroj samotný je při použití s jiným jazykem distribuován ve formě sdílené knihovny (.so na Unixu, .dll na Windows) obsahující potřebný kód. Jednotlivé

funkce jsou pak uživatelům zpřístupněny v případě Javy pomocí rozhraní Java Native Interface (JNI) a wrapper javovské knihovny, která obsahuje třídy zahrnující veškerou funkcionalitu. Použití knihovny samotné tedy obnáší pouze volání funkcí přímo z javovského kódu. Narozdíl od Treex tedy není nutné používat pomocné soubory ani spouštět nové procesy.

MorphoDiTa podporuje při generování tzv. tag wildcards, tedy tagy dle PDT, v nichž na některých pozicích mohou být speciální znaky určující množinu možných hodnot. Např. znak `?` symbolizuje libovolnou hodnotu na dané pozici. Při generování potom MorphoDiTa dosazuje za wildcardy všechny konkrétní hodnoty, pro ně provádí generování a vrací všechny získané výsledky dohromady.

Wildcardy jsme využili i v našem normalizátoru. Pokud se při provádění shody nepodaří vygenerovat tvar podle konkrétního tagu, zkusíme v tomto tagu všechny neurčené poziční značky (s hodnotou `-`) nahradit za wildcard `?` a generovat znovu. Může se totiž stát, že některé tvary slov mají nastaveny např. stylové příznaky (značka na pozici 15) a pro hodnotu `-` této značky se nic nevygeneruje.

Generování samotné může do procesu normalizace přinést i další problémy. Použitý morfologický slovník totiž pro jednu hodnotu tagu a lemmatu často obsahuje více různých tvarů, které jsou při volání generující funkce vráceny všechny. Náš program potom nemá jak vybrat tvar správný, takže volí vždy prostě první ze seznamu. Obvykle jde například o počeštěné tvary cizích slov. Např. pro lemma *software* a tag `NNIS1-----A----` je vrácena dvojice *softver*, *software*, použije se tedy první tvar, *softver*. Bohužel však např. pro tag `NNIS2-----A----` jsou tvary vráceny v jiném pořadí, konkrétně *software*, *softveru*. Nemůže být tedy dodržena ani konzistence použitých tvarů.

4.6 Implementace pravidlové procedury

K normalizaci samotné bude sloužit normalizační procedura popsaná v sekci 3.9. Ta bude pracovat se stromem získaným z treex dokumentu (viz 4.4.1) a ke generování využívat nástroj MorphoDiTa (viz 4.5). Abychom nemuseli mít různé procedury pro normalizaci samostatného stromu a pro normalizaci podstromu v rámci normalizace entit v souvislém textu, pokusíme se případy sjednotit. Procedura si vždy pamatuje identifikátor entity, kterou normalizuje, aby mohla rozlišit uzly, jež má brát v potaz, a tagy a tvary slov, s nimiž má pracovat (viz 3.11.2). Jako identifikátory entit jsou použita kladná celá čísla. Zavedeme dále speciální identifikátor -1 značící, že má procedura projít strom celý a pracovat s výchozí hodnotou tagu a slovního tvaru. Díky tomu můžeme spouštět jednu pravidlovou proceduru jak v případě normalizace samostatné entity, tak v případě normalizace entit v souvislém textu.

Velké množství pravidel normalizační procedury vyžaduje, aby bylo možné za běhu měnit pro potřeby dalšího průchodu závislostní strom věty - vždy ale pouze přesouváním potomků aktuálního uzlu. Možnost mít strom modifikovatelný by značným způsobem zkomplikovala provádění algoritmu, neboť podstrom můž být sdílen více různými entitami vyžadujícími rozdílné úpravy. Proto v implementaci používáme jiné řešení - svoji aktuální polohu ve stromě uzel nezjišťuje sám o sobě,

ale je mu předána jeho otcem ve formě použité rekurzivní metody. Uzel navíc nemá přímý přístup ke svému otci, odkaz na něj je mu předán v parametru této metody. To umožňuje „měnit“ polohy synovských uzlů aniž by se skutečně modifikovala struktura stromu.

4.7 Command-line nástroje

Normalizační procedura je uživatelům zpřístupněna pomocí jednoduchého nástroje příkazové řádky. Proces jeho práce pro samostatnou entitu je následující:

- Načíst vstupní text
- Spustit Treex nad vstupním textem (4.4)
- Načíst výstup Treexu a vytvořit v paměti závislostní strom (4.4.1)
- Spustit nad stromem normalizační proceduru (4.6)
- Zrekonstruovat text entity a vrátit jej uživateli

Při normalizaci entit vyznačených v textu je pak třeba provést následující:

- Načíst vstupní XML dokument do DOM
- Ze vstupního XML dokumentu získat čistý text a množinu entit (4.3.2)
- Spustit Treex nad čistým textem (4.4)
- Načíst výstup Treexu a vytvořit v paměti závislostní strom (4.4.1)
- Propojit závislostní strom a množinu entit
- Projít strom a nad každou entitou spustit normalizační proceduru (4.6)
- Do XML DOM dokumentu vložit zrekonstruovaný text entit
- Serializovat XML do výstupního souboru

Text k normalizaci je tomuto nástroji předán formou vstupního souboru. Možnost předávání entit jako argumentů příkazové řádky se neosvědčila vzhledem k problémům s kódováním UTF-8. Výsledek normalizace je při normalizaci samostatné entity předán na standardní výstup, v případě entit v textu pak zapsán do výstupního souboru.

4.8 Webové rozhraní

Používání nástrojů v příkazové řádce není příliš uživatelsky přívětivé. Navíc se může stát, že různé terminálové aplikace budou mít problém s českými znaky v kódování UTF-8. Proto vznikla nad základními nástroji jednoduchá nadstavba ve formě webové aplikace s grafickým uživatelským rozhraním.

Účelem webového rozhraní je převzít od uživatele vstup, odeslat jej na server, tam spustit normalizátor, počkat na výsledek a ten odeslat zpátky klientovi a zobrazit jej. Vzhledem k tomu, že normalizátor pracuje poměrně dlouho (kvůli načítání modelů v Treexu), není rozumné celý tento proces řešit synchronně. Uživateli by se totiž pak mohla odpověď serveru načítat několik minut, což dost často vede k zavření

prohlížeče či opakovaným pokusům o odeslání požadavku.

Použijeme tedy asynchronní webové požadavky - technologii AJAX (Asynchronious JavaScript and XML). Jakmile bude chtít uživatel spustit normalizaci, klientská část webové aplikace (fungující v JavaScriptu) vezme vstupní text, zabalí jej do těla asynchronního HTTP (Hyper Text Transfer Protocol) POST požadavku a ten odešle prostřednictvím AJAXu. Na serveru pak požadavek příjme PHP (PHP Hypertext Preprocessor) skript, který obsah jeho těla uloží do pomocného souboru a spustí nad ním jako samostatný proces normalizační program. Výstup normalizátoru, obsah chybového výstupu a návratovou hodnotu potom skript serializuje do JSON (JavaScript Object Notation) objektu, který vrátí jako tělo stránky v HTTP odpovědi. JavaScript pak tento objekt opět dekóduje a zobrazí uživateli příslušný výstup, případně oznámí chybu a zobrazí obsah chybového výstupu. Po celou dobu čekání na výsledek je přitom webová stránka plně funkční a uživateli se pouze zobrazuje výzva k vyčkání.

PHP skript je také nastaven tak, aby chybové výstupy ukládal do souborů, jejichž názvy jsou vygenerovány z aktuálního data. Tím je možné uchovávat na serveru logy jednotlivých požadavků a případně zpětně dohledat příčinu chyby v normalizaci či pádu programu.

Hlavním problémem webového rozhraní je, že uživatel, pod kterým je spuštěn webový server (a tudíž se také spouštějí PHP skripty), musí být schopen spustit Treex. Treex je přitom závislý na instalaci pro daného uživatele, tj. potřebuje mít konfiguraci v jeho domovském adresáři a také nastavené proměnné prostředí.

Je technicky možné tento problém vyřešit ručním nastavením proměnné `HOME` a všech ostatních proměnných pro běh normalizátoru spouštějícího Treex na hodnoty jiného uživatele s instalovaným Treexem. Potom však musí mít uživatel, pod nímž je spuštěn webový server, možnost čtení i zápisu nad domovským adresářem tohoto uživatele.

5 Vyhodnocení normalizátoru

K vyhodnocení úspěšnosti implementovaného normalizátoru byly využity reálné texty, v nichž byly ručně či automaticky vyznačeny pojmenované entity. Testována byla vždy procedura, která pracuje nad entitami v rámci celého textu.

Pro referenční srovnání výstupu normalizátoru byla ve všech použitých textech provedena nejprve ruční normalizace. Autory této ruční normalizace jsou lidé, kteří nebyli s problematikou normalizace probíranou v této práci obeznámeni a kteří ani neměli přístup k samotnému normalizátoru. Nemohli tedy být ovlivněni použitými pravidly a technikami. Výstup z ruční normalizace byl upraven do formátu identického s výstupním formátem normalizátoru (konkrétní skript viz příloha F), při té příležitosti byly odstraněny zjevné překlepy a chyby.

Vyhodnocení úspěšnosti bylo prováděno postupným porovnáváním hodnot atributů `normalized_name` u všech elementů `ne` obou dokumentů. Toto porovnávání nebralo v potaz velikost písmen, neboť v ručně normalizovaných tvarech často nebylo zachováno stejné schéma jako v původních entitách. Dále byly ignorovány všechny entity, jež byly při ruční normalizaci označeny za chybné (tj. byl jim přiřazen speciální normalizovaný tvar `!!!`). Použitý skript je přiblížen v příloze F.

Všechna data použitá při vyhodnocení jsou k dispozici na přiloženém CD (viz příloha B).

5.1 Testování na běžném textu

První test byl proveden na překladu textu krátkého blogu pojednávajícího o výstavbě silnice v Richmondu[15]. Entity v něm byly rozpoznány automaticky pomocí software NameTag [6]. Výsledky jsou následující:

Celkový počet entit:	54
Správně normalizováno:	54
Úspěšnost:	100%

Na základě tohoto testu tedy normalizátor vypadá velice dobře. Příčinou stoprocentního výsledku je však fakt, že procedura automatického rozpoznávání v textu vyznačuje převážně vlastní jména, názvy a čísla. To jsou obvykle entity, které jsou buďto přímo v základním tvaru, nebo jejich normalizace není příliš složitá. Zajímavější případy potom NameTag vůbec v textu neoznačil. Příkladem může být fráze *výdajích na dopravu*, která by v našem pojetí nejspíše pojmenovanou entitou byla.

Vysoká úspěšnost je v tomto případě dále způsobena tím, že jde o souvislý text, takže tagging a parsing na něm probíhá s velice dobrou přesností.

15 http://www.baconsrebellion.com/2006/05/288-chickens-coming-home-to-roost.html, autorem překladu je Jan Hajič, jr.

5.2 Testování na CNEC

Dalším použitým zdrojem dat byl CNEC zmíněný již v sekcích 2.4.2 a 3.3. Z tohoto souboru byla čerpána také ladící data, tedy entity sloužící k rozboru možných případů a konstrukci samotné procedury. Tentokrát byl však zvolen souvislý úsek testu, konkrétně začátek části označené jako `dtest`. Pro úplnost dodejme, že CNEC obsahuje ručně vyznačené entity a že v použitém vzorku byly odstraněny vnořené entity. Výsledky vypadají následovně:

Celkový počet entit:	262
Správně normalizováno:	238
Úspěšnost:	90,8%

Úspěšnost se zdá být dobrá, ale při prozkoumání konkrétních chyb zjistíme, že se často staly i v entitách, které už byly v základních tvarech. Příčinou patrně bude nesouvislost některých úseků textu tvořících např. pouhý výčet pojmenovaných entit. Parsing takovéto úseky zpracovává jako celé věty a často vrací nesprávné výsledky. Dalším faktorem způsobujícím chyby je velký počet cizích slov v pojmenovaných entitách, podrobněji je tomuto problému věnována část 3.8.2.

5.3 Testování na legislativních dokumentech

V tomto testu se zaměříme na odvětví, kde by normalizace mohla být prakticky využívána. Půjde o text legislativního charakteru, konkrétně Předpis č. 500/2002 Sb[16], v něm byly entity vyznačeny ručně. Výsledky testu:

Celkový počet entit:	333
Správně normalizováno:	305
Úspěšnost:	91,6%

Zajímavé je, že jsme dosáhli prakticky stejného výsledku jako v případě CNEC. Použitý text je však pro naše účely mnohem více vypovídající. Obsahuje totiž velké množství dlouhých a z pohledu normalizace zajímavých entit, např. *nehmotnými výsledky výzkumu a vývoje*. Zmíněná entita byla procedurou normalizována správně jako *nehmotné výsledky výzkumu a vývoje*.

Některé chyby v tomto testu jsou navíc způsobeny nesprávným vyznačením pojmenované entity (důsledek ruční normalizace), např. *ke dni účinnosti rozhodnutí o úpadku* místo *dni účinnosti rozhodnutí o úpadku*. Ručně normalizující v tomto případě nebral předložku *ke* v potaz a normalizoval entitu jako *den účinnosti rozhodnutí o úpadku*. Automatická normalizační procedura však chybně zařazenou předložku ignorovat nemůže. Další chyby jsou pak způsobeny opět taggingem a parsingem, který v tomto typu dokumentu naráží na problémy s odrážkami a oddělovači sekcí.

U tohoto typu textu je dále velmi názorně vidět problém plurality, o němž se hovoří v sekci 3.8.1. Většina entit by pravděpodobně měla být normalizována do jednotného čísla (např. *účetní jednotka* místo *účetní jednotky*, *závazek* místo *závazky*

16 Jde o data využívaná v rámci projektu INTLIB (id. č. TA02010182, http://ufal.mff.cuni.cz/intlib)

apod.). Entita *přeměna společnosti* se v textu dokonce vyskytuje v obou číslech, proto má v textu také dvě normalizované podoby, přestože jde o označení téhož jevu. Takový případ je v rozporu s původní myšlenkou přiřazení jednoznačného identifikátoru.

5.4 Testování na policejních zprávách

Pro poslední test byla použita policejní zpráva[17], v níž byly pojmenované entity detekovány opět pomocí NameTagu [6]. Výsledky testu:

Celkový počet entit:	143
Správně normalizováno:	136
Úspěšnost:	95,1%

Výsledek je opět velice pozitivní. Dodejme jen, že v tomto textu převládají díky zdroji dat a metodě vyznačení entit vlastní jména a čísla v rámci adres a dat, jde tedy spíše o méně významný test.

5.5 Shrnutí výsledků automatických testů

Výsledky všech testů jsou shrnuty v tabulce 3. Je z nich patrné, že úspěšnost normalizace u automaticky rozpoznaných entit je mírně vyšší než u ručně vyznačených entit. To je způsobeno jednak tím, že automaticky rozpoznat lze většinou pouze entity jednoduššího charakteru (vlastní jména, číselné údaje apod.), a dále také častým výskytem chyb v ručních anotacích.

Celkově jsme normalizátor otestovali na 792 pojmenovaných entitách, z nichž 733 byly normalizovány správně. Dosáhli jsme tak úspěšnosti cca 92,6%.

Dokument	Způsob vyznačení	Entit celkem	Entit správně	Úspěšnost
Překlad blogu	automatické	54	54	100%
CNEC	ruční	262	238	90,8%
Předpis	ruční	333	305	91,6%
Policejní zpráva	automatické	143	136	95,1%
Celkem		**792**	**733**	**92,6%**

Tab. 3: Shrnutí výsledků testů normalizátoru

17 Zpráva pochází z dat, s nimiž pracuje softwarový projekt Textan na MFF UK (http://ksvi.mff.cuni.cz/~holan/SWP/zadani/textan.pdf).

6 Dokumentace CNEN

6.1 Uživatelská dokumentace

Czech Named Entity Normalizer (CNEN) je jednoduchým nástrojem umožňujícím provádět automatickou normalizaci pojmenovaných entit v českých textech. Je založen na principech popsaných v kapitolách 3 a 4 této práce.

Aktuální verze CNEN je udržována ve formě git repozitáře na adrese:

https://github.com/ips1/CNEN

6.1.1 Požadavky

Pro běh CNEN je potřeba, aby počítač splňoval následující požadavky:

- OS Linux (pravděpodobně i jiné OS založené na UNIXu), testováno na Ubuntu verze 12.04
- Nainstalovaný JRE (Java Runtime Environment) verze minimálně 1.6[18]
- Nainstalovaný a fungující systém Treex[19] včetně všech potřebných bloků[20], spustitelný soubor treex musí být v proměnné PATH[21]

6.1.2 Instalace

CNEN je šířen ve formě tar archivu. K jeho rozbalení slouží příkaz:

```
tar -xvf cnen.tar
```

Vznikne adresář `cnen` obsahující všechny potřebné soubory. Nyní je nutné v tomto adresáři nastavit patřičná oprávnění spouštěcím souborům:

```
cd cnen
chmod u+x CNEN.sh
chmod u+x CNEN-text.sh
```

CNEN ke svému běhu dále potřebuje knihovnu MorphoDiTa. Ta je k dispozici ve formě binárních souborů na přiloženém CD, aktuální verze je pak ke stažení na adrese:

http://github.com/ufal/morphodita/releases/latest

18 Ke stažení na adrese: http://www.oracle.com/technetwork/java/javase/downloads/jre7-downloads-1880261.html

19 Podrobný návod k instalaci je k dispozici zde: http://ufal.mff.cuni.cz/treex/install.html. Je třeba provést také kroky č. 5 a 6 tohoto návodu.

20 Info viz přílohu E

21 Lze snadno otestovat zadáním `treex -v` do okna terminálu, vypíše-li se informace o verzi, vše je v pořádku.

Po stažení a rozbalení archivu je třeba zkopírovat obsah adresáře `morphodita-1.2.0-bin\bin-linux32\java` (případně `morphodita-1.2.0-bin\bin-linux64\java`, v závislosti na architektuře používaného systému) do adresáře `cnen`. Číslo 1.2.0 je aktuální verze MorphoDiTa, může se tedy měnit.

Posledním krokem je získání morfologického slovníku pro MorphoDiTu. Nalezneme jej na přiloženém CD, případně na adrese:

http://hdl.handle.net/11858/00-097C-0000-0023-68D8-1

Z poměrně rozsáhlého archivu je potom nutné zkopírovat soubor `czech-morfflex-131112.dict` do adresáře `cnen` a přejmenovat jej na `morphodita.dict`.

Tím je instalace programu dokončena a CNEN je připraven k použití.

6.1.3 Používání programu

Ke spustění programu CNEN jsou určeny dva skripty v adresáři cnen. Pro normalizaci jedné nebo více samostatných pojmenovaných entit slouží příkaz:

```
./CNEN.sh VSTUPNI_SOUBOR
```

`VSTUPNI_SOUBOR` reprezentuje cestu k souboru se vstupními daty. Ten musí obsahovat na každém řádku jednu pojmenovanou entitu a musí být v kódování UTF-8. Výsledek normalizace se vypíše na standardní výstup, opět ve tvaru jedna entita na jeden řádek. Pro uložení výsledku normalizace do souboru je možné použít operátoru přesměrování výstupu v shellu:

```
./CNEN.sh VSTUPNI_SOUBOR > VYSTUPNI_SOUBOR
```

Pro normalizaci entit vyznačených v textu použijeme příkaz:

```
./CNEN-text.sh VSTUPNI_SOUBOR VYSTUPNI_SOUBOR
```

`VSTUPNI_SOUBOR` je cesta k souboru obsahujícímu entity vyznačené v textu. Tento soubor musí splňovat formát popsaný v 6.1.4 a musí být v kódování UTF-8. `VYSTUPNI_SOUBOR` je pak cesta k souboru, do něhož se uloží výsledek normalizace (tento soubor by neměl existovat, jinak bude přepsán).

Běh obou skriptů může být poměrně dlouhý. Díky náročnosti systému Treex je možné, že bude pracovat i několik minut. Na standardní chybový výstup je přitom přesměrováván standardní chybový výstup Treex, takže je možné sledovat jeho průběh a případné chyby.

Program za běhu vypisuje velké množství informací na standardní chybový výstup. Tento výpis lze potlačit přidáním následujícího přesměrování za libovolný ze spouštěcích příkazů:

```
2> /dev/null
```

6.1.4 Formát vstupního a výstupního souboru

Při normalizaci entit v textu je třeba poskytnout programu CNEN soubor s vyznačenými pojmenovanými entitami v textu. Jde o validní soubor ve formátu XML v kódování UTF-8. Konkrétní XML elementy použité v souboru mohou být libovolné, ani kořenový element není pevně určen (ale musí existovat, jinak nepůjde o validní XML a soubor se nepodaří načíst). Za pojmenované entity bude považován textový obsah elementů `<ne>`, tj. text mezi značkami `<ne>` a `</ne>` (včetně textu ve vnořených elementech). Tyto elementy mohou mít také atributy, jejich obsah bude přenesen do výstupu. Výstupním souborem bude XML totožné struktury, jedinou změnou v něm budou nové atributy `normalized_name` u elementů `<ne>` obsahující výsledky normalizace jednotlivých entit.

Pro umístění značek `<ne>` a `</ne>` platí jedno omezení - značky by měly vždy ohraničovat celá slova, neměly by zahrnovat pouze jejich části.

Tedy např. toto je příklad správného vstupu:

```
<doc>Souhlas k montáži tohoto zařízení ve východních a severních
<ne type="gr">Čechách</ne> získala v těchto dnech soukromá firma
<ne type="if">HROMO</ne> v <ne type="gu">Hradci
Králové</ne>.</doc>
```

Odpovídajícím výstupem pak bude:

```
<doc>Souhlas k montáži tohoto zařízení ve východních a severních
<ne type="gr" normalized_name="Čechy">Čechách</ne> získala v
těchto dnech soukromá firma <ne type="if"
normalized_name="HROMA">HROMO</ne> v <ne type="gu"
normalized_name="Hradec Králové">Hradci Králové</ne>.</doc>
```

Nesprávným vstupem je např.:

```
<ne>Byzanc</ne> je známá také jako Východo<ne>římská</ne> říše
```

Chybí zde kořenový element (tj. nejde o validní XML) a navíc je označení entity vnořeno doprostřed slova.

6.1.5 Použití vlastního Treex scénáře

CNEN využívá k provedení lingvistických anotací textu nástroj Treex (viz 4.4). Scénář spouštěný v Treexu je uložen v následujícím souboru:

```
cnen/cz/cuni/mff/kubatpel/java/cnen/parsing/treex_scen.txt
```

Tento scénář je možné nahradit vlastním, např. za účelem použití novějších bloků. Je však bezpodmínečně nutné, aby bylo dodrženo schéma původního scénáře, tedy:

- Scénář neobsahuje Reader ani Writer bloky (ty si program doplňuje sám)
- Scénář provede nad textem segmentaci, tokenizaci, tagging a parsing

Chybí-li soubor úplně, program použije výchozí scénář.

6.1.6 Webové rozhraní

Webové rozhraní umožňuje snadné používání programu CNEN prostřednictvím grafického uživatelského rozhraní z prostředí libovolného operačního systému. Stačí mít k dispozici webový prohlížeč podporující JavaScript.

Webové rozhraní je momentálně dostupné na následující adrese:

http://ufallab.ms.mff.cuni.cz/~kubat/CNEN/

Podobu výchozí obrazovky ukazuje obr. 16. Mezi módy programu (normalizace samostatné entity, normalizace entit v textu) se můžete přepínat prostřednictvím záložek v horní části okna. Vstup programu vložte do textového pole v prostřední části okna. Samotnou normalizaci pak zahájíte klepnutím na tlačítko Normalizovat. Po dobu normalizace budou veškeré ovládací prvky dané záložky neaktivní. Normalizace může trvat až několik minut, proto okno prohlížeče nezavírejte, neobnovujte ani nepřecházejte na jinou stránku.

Po dokončení normalizace se zobrazí nové textové pole obsahující její výstup. Stala-li se při normalizaci chyba, bude zobrazena červená chybová hláška a v textovém poli pak bude zobrazen chybový výstup normalizačního programu (obr. 17).

Obr. 16: Výchozí obrazovka webového rozhraní CNEN

Obr. 17: Chybový výstup webového rozhraní CNEN

6.2 Programátorská dokumentace

Programátorská dokumentace k programu CNEN bude velice stručným přehledem balíčků a tříd a jejich zapojení v celém procesu normalizace. Detailnější úvahu o použitých technikách je možné najít v kapitole 4, podrobný popis tříd a metod pak poskytují dokumentační komentáře přímo v kódu, případně přehled vygenerovaný nástrojem javadoc (viz příloha D).

Při tvorbě programu byl kladen hlavní důraz na komplexní objektový návrh a jednoduchá rozhraní jednotlivých tříd, a to zejména z těchto důvodů:

- Jednoduchost, čitelnost a udržitelnost zdrojového kódu
- Znovupoužitelnost některých částí programu - s pojmenovanými entitami můžeme v budoucnu chtít provádět i další akce, pro něž budou potřeba podobné datové struktury
- Snadná upravitelnost a rozšiřitelnost - povaha implementované pravidlové procedury připouští úpravy množiny pravidel, případně jejich kompletní nahrazení

6.2.1 Základní třídy a proces normalizace

Pravidlová procedura pro normalizaci pojmenovaných entit implementovaná v CNEN pracuje primárně se závislostním stromem věty. Nejdůležitější datovou strukturou celého programu je tedy právě tento strom reprezentovaný třídou **SentenceTree** z balíčku **cnen.sentencetree**[22]. Jde o stromovou datovou strukturu, která si pro každý uzel (třída **TreeNode**) drží jeho pořadí ve stromě, závislostní funkci dle PDT (třída **AnalyticalFunction**), seznam potomků, původní tvar slova, současný tvar slova a množinu tvarů slova pro jednotlivé entity, do nichž patří. Tvar slova reprezentuje třída **WordForm** obsahující slovo samotné, lemma a tag dle PDT (třída **Tag**) a příznak, zda jde již o tvar normalizovaný.

K získání instance SentenceTree z externího souboru slouží interface **SentenceTreeParser** z balíčku cnen.parsing. Obsahuje metody pro načtení množiny stromů reprezentující jeden dokument, ale také více množin pro více dokumentů. Jeho konkrétní implementace použité v projektu jsou dvě. **TreexParser** zpracuje treex soubor. **PlaintextTreexParser** je v podstatě wrapperem TreexParseru. Načte čistý text, spustí nad ním Treex jako externí proces (za použití třídy **TreexInterface**), počká na jeho ukončení a poté zpracuje výsledný soubor TreexParserem. Tento parser se v programu reálně používá.

Ke zpracování a modifikaci instance SentenceTree slouží interface **TreeAction** z balíčku **cnen.actions**. V tomto balíčku jsou dále jeho dvě konkrétní implementace - **SingleEntityNormalizer** a **EntitySubtreeNormalizer**. První z nich je přímá implementace pravidlové procedury pro normalizaci jedné entity. Druhá z nich potom slouží k normalizaci více entit v rámci jednoho stromu podle algoritmu popsaného v 3.11.2, k čemuž sama použije instancí SingleEntityNormalizeru.

SingleEntityNormalizer využívá morfologické generování tvarů slov pomocí interface **MorphologyGenerator** z balíčku **cnen.morphology**. V CNEN jsou zahrnuty dvě implementace tohoto generátoru - **MorphoditaGenerator** využívající vazby přes JNI na program MorphoDiTa (viz 4.5) a **ManualGenerator**, který slouží spíše pouze testovacím účelům a generování provádí prostřednictvím dialogu přes standardní vstup a výstup.

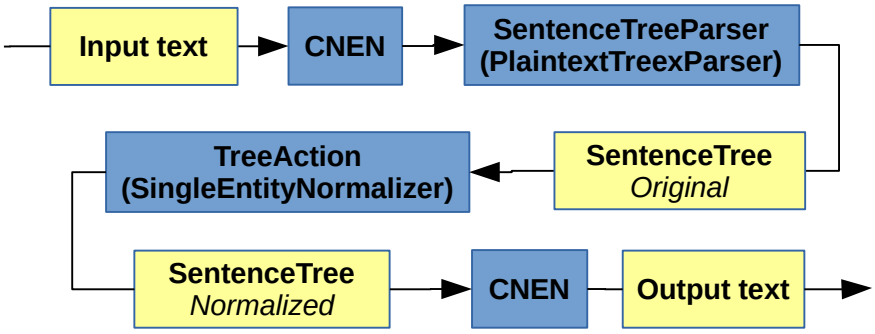
Doplňkovým balíčkem, který slouží ke zpracování textu s vyznačenými entitami, je **cnen.annotations**. Obsahuje třídu **AnnotatedText** sloužící k uchování textu a množiny entit v něm vyznačených (instance **EntityAnnotation**). Její instance se načítá ze vstupního souboru prostřednictvím třídy **AnnotatedTextParser**. Pomocí třídy **TreeTextMatcher** se poté páruje s odpovídající množinou SentenceTree a k jednotlivým EntityAnotation se přiřazují seznamy jim odpovídajících uzlů TreeNode.

Třída **DOMLoader** z balíčku **cnen.dom** slouží ke snadnému načítání DOM (instance třídy org.w3c.dom.Document) ze souborů XML a také k jejich ukládání. Je využívána jak při parsování treex pomocí TreexParseru, tak při načítání XML anotací pomocí AnotatedTextParseru.

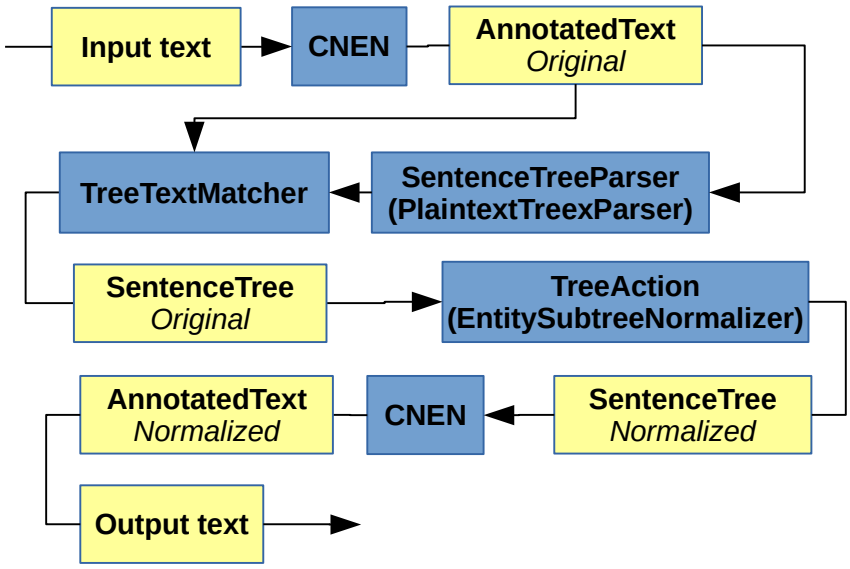
22 Všechny Java balíčky použité v tomto programu mají prefix cz.cuni.mff.kubatpe1.java, jenž není uveden. Plný název zmíněného balíčku by tedy byl: cz.cuni.mff.kubatpe1.java.cnen.sentencetree

Hlavní třída programu, třída **CNEN**, je spustitelná (obsahuje metodu main) a stará se o spouštění celého normalizačního řetězce. Ten je pro scénář normalizace samostatné entity vyobrazen na obr. 18 a pro scénář normalizace entit v textu na obr. 19. Jde v podstatě o schéma popsané v sekci 4.7.

V parametrech příkazové řádky je třídě CNEN oznámeno, zda se normalizuje samostatná entita či entity v textu, dále je předána cesta k morfologickému slovníku pro MorphoDiTu a nakonec také cesta ke vstupnímu souboru (pro normalizaci v textu také k výstupnímu souboru). Pro zjednodušení spouštění programu jsou připraveny shell skripty `CNEN.sh` a `CNEN-text.sh`, jejichž úkolem je mimo předání popsaných parametrů také nastavení proměnné prostředí `LD_LIBRARY_PATH` a proměnné JVM `CLASSPATH` na aktuální adresář z důvodu načtení knihovny MorphoDiTa.



Obr. 18: Schéma zpracování samostatné entity – tmavé jsou akce, světlá jsou data



Obr. 19: Schéma zpracování entit vyznačených v textu - tmavé jsou akce, světlá data

6.2.2 Implementace vlastní normalizační procedury

Normalizační procedura bude třídou implementující interface TreeAction. V tomto interface je třeba pouze implementovat metodu `runOnTree(SentenceTree)`. Ta obdrží v parametru závislostní strom věty a očekává se, že jej vhodným způsobem modifikuje. Modifikace spočívá ve změně atributu `content` jednotlivých uzlů (TreeNode) prostřednictvím metody `setContent(String)`.

Průchod stromem lze realizovat dvěma způsoby - buďto pomocí metody `getRoot()` získáme odkaz na kořen stromu, který poté projdeme hierarchicky (využitím metod na TreeNode, viz javadoc reference v příloze D), nebo použitím metody `getLinearRepresentation()`, která vrátí `List<TreeNode>` obsahující uzly stromu seřazené podle jejich pořadí (atribut order).

K morfologickému generování lze získat přístup pomocí interface MorphologyGenerator. Jeho jediná metoda `generateForTag(String, Tag)` provede generování tvaru ze zadaného lemmatu a pro zadaný tag. Přesné chování metody závisí na konkrétní implementaci. MorphoditaGenerator použitý ve výchozí normalizační proceduře vrací při více výsledcích vždy první z nich. Naopak není-li výsledek žádný, je vyvolána výjimka MorphologyGeneratingException.

Pokud při normalizaci dojde k nějakému chybovému stavu, díky kterému výstup nelze považovat za korektní, je možné vyvolat výjimku TreeActionException.

Závěr

V tomto textu jsme poměrně podrobně prozkoumali základní lingvistický přístup k normalizaci pojmenovaných entit. Zmapovali jsme velké množství různých typů entit a s nimi souvisejících problémů. Na jejich základě jsme navrhli normalizační pravidlovou proceduru. Ta byla také za použití dalších pomocných nástrojů implementována a je nyní k dispozici i prostřednictvím webového rozhraní.

Ukázalo se, že problém normalizace jako takový není příliš komplikovaný. Pro vyřešení naprosté většiny běžných případů nám stačí okolo desítky jednoduchých pravidel. Jedinými vážnějšími komplikacemi, na něž jsme narazili, a které se nám v obecné podobě nepodařilo vyřešit, jsou identifikace shodných a neshodných přívlastků v pozici za podstatným jménem a otázka zachování plurality pro entity vyskytující se v množném čísle. Je samozřejmě velice pravděpodobné, že existují některé další případy, jež jsme vůbec nezkoumali, ale vzhledem k množství probíraných entit by šlo zřejmě o okrajové a velice málo časté situace.

Většina chyb, které bohužel výrazně snižují použitelnost normalizačního nástroje pro reálné účely, plyne z nedokonalostí nástrojů pro morfologickou a syntaktickou anotaci textu a nástrojů pro morfologické generování. Tuto skutečnost jsme se pokusili vyřešit návrhem procedury pro normalizaci entit v rámci souvislého textu. Vycházíme při tom z předpokladu, že analýza pojmenované entity proběhne lépe, bude-li entita v reálné větě. Jak ukázalo vyhodnocení normalizátoru na několika reálných testech, toto řešení je pouze částečné a nesprávných výsledků je stále velmi mnoho.

Je možné, že efektivita pomocných nástrojů se bude časem zlepšovat, v tom případě by neměl být problém použít v normalizátoru novější verze a tím zvýšit i jeho účinnost. Statisticky fungující nástroje však stoprocentní úspěšnosti nemohou dosáhnout nikdy. Proto by tedy bylo vhodné zaměřit se při dalším zkoumání problému normalizace tímto směrem a pokusit se např. zmapovat obvyklé problémy, které tyto nástroje způsobují, a poskytnout metodu, jak problémy identifikovat a vyřešit.

Zajímavým rozšířením normalizačního nástroje by mohlo být vytvoření jednoduchého jazyka popisujícího vlastní pravidla a možnost přidání těchto pravidel do procedury. Tím by se program stal rozšiřitelným a snadněji udržovatelným bez zásahů do zdrojového kódu. Navíc by se jeho funkcionalita dala adaptovat specifickým způsobům použití se zvláštními požadavky na normalizaci. Cílem této práce však byl návrh konkrétní procedury, proto by podobná technika přesahovala její rámec.

Další možností rozšíření by bylo zakomponování normalizátoru do Treex jakožto samostatného bloku. K tomu by musel být vytvořen wrapper pro Javovský program v jazyce Perl a musely by se také změnit vstupní a výstupní formáty na formát vnitřně používaný Treex pro reprezentaci dokumentu v paměti. Ve výsledku by

potom bylo možné normalizátor používat kdekoliv, kde je nainstalován Treex, a navíc by bylo možno data předávát v libovolném formátu, pro nějž existuje Reader blok, a naopak vracet v libovolném formátu, pro nějž existuje Writer blok. Navíc nástroj pro automatické rozpoznávání entit v textu NameTag byl již do Treex zakomponován, takže by také bylo možné nad čistým textem najednou provést rozpoznání i normalizaci entit.

Seznam použité literatury

[1] HAJIČ, Jan, et al. 2012. *Prague Czech-English Dependency Treebank 2.0*. Linguistic Data Consortium, Philadelphia (http://ufal.mff.cuni.cz/pdt2.0).

[2] PAJAS, Petr, ŠTĚPÁNEK, Jan. *Recent Advances in a Feature-Rich Framework for Treebank Annotation.* The 22nd International Conference on Computational Linguistics - Proceedings of the Conference, Manchester, pp. 673-680, 2008 (http://ufal.mff.cuni.cz/jazz/pml).

[3] Platforma Treex (http://ufal.mff.cuni.cz/treex).

[4] ŠEVČÍKOVÁ, Magda, ŽABOKRTSKÝ, Zdeněk, KRŮZA, Oldřich. *Zpracování pojmenovaných entit v českých textech.* Technical Report TR-2007-36. 2007.

[5] ŠEVČÍKOVÁ, Magda, ŽABOKRTSKÝ, Zdeněk, STRAKOVÁ, Jana, STRAKA, Milan. *Czech Named Entity Corpus 2.0* (http://ufal.mff.cuni.cz/cnec).

[6] STRAKA, Milan, STRAKOVÁ, Jana. NameTag (http://ufal.mff.cuni.cz/nametag).

[7] STRAKOVÁ, Jana, STRAKA, Milan, HAJIČ, Jan: *A New State-of-The-Art Czech Named Entity Recognizer.* Lecture Notes in Computer Science, Vol. 8082, Text, Speech and Dialogue: 16th International Conference, TSD 2013. pp. 68-75, 2013

[8] STRAKA, Milan, STRAKOVÁ, Jana. MorphoDiTa (https://ufal.mff.cuni.cz/morphodita).

Seznam tabulek a obrázků

Seznam použitých zkratek

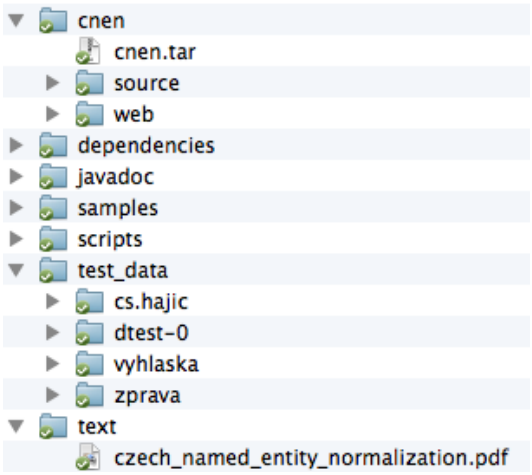
- **HTML** - HyperText Markup Language, značkovací jazyk používaný v prostředí World Wide Webu
- **CNEC** - Czech Named Entity Corpus [5], korpus obsahující texty s ručně vyznačenými pojmenovanými entitami
- **PDT** - Prague Dependency Treebank [1], Pražský závislostní korpus, korpus obsahující texty s ručními anotacemi na několika rovinách
- **XML** - Extensible Markup Language, značkovací jazyk určený k uchování a přenosu strukturovaných dat
- **w-rovina** - slovní rovina lingvistické anotace textu
- **m-rovina** - morfologická (tvaroslovná) rovina lingvistické anotace textu
- **a-rovina** - analytická rovina lingvistické anotace textu
- **PML** - Prague Markup Language [2], značkovací jazyk založený na XML určený k uchování lingvisticky anotovaných dat
- **MST** - maximum spanning tree, maximální kostra grafu
- **NER** - Named Entity Recognition, označení systémů pro automatické rozpoznávání pojmenovaných entit
- **MUC** - Message Understanding Conference, řada konferencí v 80. a 90. letech 20. století zaměřených na extrakci informací z textu
- **TEI** - Text Encoding Initiative, skupina vytvářející formát pro uchování textů se zaměřením na jejich sémantiku
- **JVM** - Java Virtual Machine, virtuální stroj, v němž jsou spouštěny programy napsané v jazyce Java
- **JRE** - Java Runtime Environment, balík obsahující JVM a standardní knihovny nutný pro spuštění programů napsaných v jazyce Java
- **DOM** - Document Object Model, paměťová hierarchická reprezentace HTML či XML dokumentu, zároveň také rozhraní pro práci s ním
- **SAX** - Simple API for XML, rozhraní pro práci s XML dokumentem založené na průchodu jeho strukturou a použití událostí
- **JNI** - Java Native Interface, rozhraní pro spouštění nativního kódu z prostředí Javy
- **AJAX** - Asynchronious JavaScript and XML, technologie pro vývoj interaktivních webových aplikací
- **HTTP** - HyperText Transfer Protocol, protokol využívaný v prostředí World Wide Webu

- **PHP** - PHP Hypertext Preprocessor (rekurzivní zkratka), původně Personal Home Page, skriptovací programovací jazyk určený k použití na serverové části webových aplikací
- **JSON** - JavaScript Object Notation, formát pro výměnu strukturovaných dat mezi klientskou a serverovou aplikací
- **CNEN** - Czech Named Entity Normalizer, implementace normalizátoru pojmenovaných entit prezentovaná v rámci této práce

Přílohy

A. Obsah přiloženého CD

Všechna potřebná data v elektronické podobě jsou k práci přiložena na CD. Adresářová struktura tohoto CD vypadá následovně:



- `cnen` - adresář obsahující program CNEN a jeho zdrojové kódy (soubor `readme.txt` stručně popisuje způsob kompilace programu)
- `dependencies` - adresář obsahující programy a data třetích stran, na nichž CNEN závisí, konkrétně knihovnu MorphoDiTa a morfologický slovník, obojí v aktuálních verzích v době psaní textu (Treex přiložen není, neboť se instaluje přímo z webového úložiště)
- `javadoc` - viz příloha D
- `samples` - viz příloha C
- `scripts` - viz příloha F
- `test_data` - viz příloha B
- `text` - adresář obsahující pdf verzi tohoto textu

B. Data k vyhodnocení normalizátoru

Data použitá v kapitole 5 se nachází v kompletní podobě na přiloženém CD v adresáři `test_data`. Jednotlivé adresáře odpovídají textům, na nichž byl normalizátor testován:

- `cs.hajic` - 5.1 Testování na běžném textu
- `dtest-0` - 5.2 Testování na CNEC
- `vyhlaska` - 5.3 Testování na legislativních dokumentech
- `zprava` - 5.4 Testování na policejních zprávách

Každý adresář obsahuje následující soubory:

- `nazev.raw.xml`
  - čistý text s vyznačenými entitami
- `nazev.rucni.xml`
  - text s ručně normalizovanými entitami
- `nazev.auto.xml`
  - text s automaticky normalizovanými entitami (výstup normalizátoru)
- `nazev.result.txt`
  - výsledek porovnání obou normalizací, vygenerován automatickým skriptem

C. Ladící vzorky

Ladící vzorky dat z CNEC popsané v sekci 3.3 jsou dostupné na přiloženém CD v adresáři `samples`. Je zde celkem 5 souborů ve výše zmíněném formátu. Ke generování těchto souborů byl použit skript uvedený v příloze E.

D. Automatická dokumentace javadoc

V elektronické podobě je k práci přiložena dokumentace programu CNEN vygenerovaná nástrojem javadoc z dokumentačních komentářů přímo ve zdrojovém kódu programu. Tato dokumentace obsahuje přehled všech tříd programu a jejich veřejných metod. U každé metody jsou navíc rozepsány parametry, návratová hodnota a výjimky, které může vyvolávat.

Dokumentace se nachází na přiloženém CD v adresáři `javadoc`. Jde o soubor navzájem propojených HTML stránek. Výchozí stránkou je `index.html`.

E. Seznam potřebných Treex bloků

Pro spuštění programu CNEN (při použití výchozího Treex scénáře) je třeba, aby byly v Treex na cílovém počítači nainstalovány následující bloky:

```
Read::Text
Write::Treex
W2A::CS::Segment
```

```
W2A::CS::Tokenize
W2A::CS::TagFeaturama lemmatize=1
W2A::CS::FixMorphoErrors
W2A::CS::ParseMSTAdapted
W2A::CS::FixAtreeAfterMcD
W2A::CS::FixIsMember
W2A::CS::FixPrepositionalCase
W2A::CS::FixReflexiveTantum
W2A::CS::FixReflexivePronouns
```

F. Pomocné skripty

Při tvorbě této práce bylo potřeba některé činnosti více či méně zautomatizovat. Proto vznikla řada pomocných jednoduchých skriptů zajišťujících tuto činnost. Všechny skripty jsou k dispozici na přiloženém CD v adresáři `scripts`. Účelem těchto skriptů je pouze jejich jednorázové použití, jsou tedy psány co možná nejúsporněji a nedochází v nich např. ke kontrole vstupů atd. Často nemají skripty ani interface a očekává se, že pro každé použití bude upraven jejich zdrojový kód (tj. ručně zavolány funkce v nich definované s vlastními parametry). Jsou uvedeny pouze pro úplnost, aby bylo možno zrekonstruovat některé kroky popsané v textu práce. Nejsou v žádném případě určeny pro koncového uživatele normalizační procedury.

Konkrétní seznam skriptů:

- `nestats.py`
  - Skript slouží k vytváření statistik nad CNEC ve formátu treex a ke generování vzorků entit. Nemá uživatelské rozhraní.
- `generate.py`
  - Skript slouží k odstraňování vnořených entit (zachová pouze vnější označení) z XML souborů ve vstupním formátu CNEN. Dále umožňuje rozdělit jeden soubor na více menších. Byl určen ke generování dat k ruční normalizaci nad CNEC. Nemá uživatelské rozhraní.
- `evaluate.py`
  - Skript slouží k vyhodnocení rozdílů mezi normalizovanými XML soubory (tj. soubory, kde každý element `ne` má atribut `normalized_name`). Porovnává obsah atributů k jednotlivým entitám, a to case-insensitive. Na standardní výstup tiskne výsledky porovnání. Dá se spustit přímo s cestami k oběma souborům jako parametry příkazové řádky.

Poslední pomocný program není v adresáři s ostatními skripty, jelikož je implementován v Javě a využívá tříd CNEN. Jde o následující třídu:

```
cz.cuni.mff.kubatpe1.java.cnen.tools.AnotationFormatter
```

Tato třída slouží k převodu ručně normalizovaných souborů do porovnatelného formátu. Je spustitelná a jejím vstupem (první parametr příkazové řádky) je cesta

k XML souboru s vyznačenými entitami, do nějž byla ručně vepsána normalizovaná jména entit na řádky začínající symbolem # (je-li jich na řádku víc, pak jsou odděleny středníky). Výstupem (druhý parametr příkazové řádky) je soubor, v němž jsou jména již korektně uložena do atributů `normalized_name`.

G. Práce s CNEC

V rámci přípravy této bakalářské práce byl velmi intenzivně využíván jako zdroj dat CNEC (viz 2.4.2). Při tom bylo objeveno několik chyb v ruční anotaci pojmenovaných entit. Chyby pochází z treex formátu dat. Zde je jejich přehled:

- *Vyhlášku o zdravotnické záchranné službě a nerespektováním nadřazených zákonů*
  - Do této entity nemá patřit druhá část, „*a nerespektováním nadřazených zákonů*“, ta už je součástí zbytku věty
- *Václava ps Havla*
  - Do obsahu entity se připletl identifikátor kategorie *ps*

U v současnosti (květen 2014) dostupných variant CNEC 2.0 se dále vyskytuje drobná komplikace – text ve všech zdrojových formátech se nevyskytuje v původní podobě, ale byl zřejmě zrekonstruován z textu tokenizovaného přidáním jedné mezery mezi každé dva tokeny. To se v praxi projeví přebytečnými mezerami např. okolo interpunkce či jiných rozdělovacích symbolů, které jsou obvykle psány bez mezer okolo. Uveďme několik příkladů:

Knihovně a tiskárně pro nevidomé K . E . Macana v Praze

Městské noviny Brandýsa n / L - Staré Boleslavi

Problémem se potom může stát opětovná tokenizace takového textu, zejména pokud je použita jiná metoda tokenizace.