České vysoké učení technické v Praze
Fakulta elektrotechnická

Bakalářská práce

Analýza významu textových zpráv

Aleš Janda

Vedoucí práce: Ing. Jan Koutník

Studijní program: Elektrotechnika a informatika strukturovaný bakalářský

Obor: Informatika a výpočetní technika

září 2007

Poděkování

Tato bakalářská práce navazuje na programy Pokyd a později IQ Pokyd[1] Jednalo se o programy, které simulovaly člověka; dalo se s nimi hovořit česky prostřednictvím textu, tak jako se to dnes běžně děje mezi dvěma lidmi přes internet. Tyto programy, vydávané od roku 1999 do roku 2004, byly velice oblíbené a zejména okolo roku 2002 mezi českou a slovenskou veřejností i značně rozšířené. I když se vlastně jednalo o dost hloupou simulaci, dostal jsem několik stovek děkovných a podporujících e-mailů i klasických dopisů od lidí nejrůznějšího věku i vzdělání. Díky nim mě tato práce bavila. A i díky nim mě dodnes zajímá počítačové zpracování češtiny. A proto jsem si vybral tuto bakalářskou práci.

Chtěl bych tedy poděkovat všem lidem podporujícím můj dřívější program nebo stávající práci a všem, kteří mi v mém úsilí pomohli. Jmenovitě:

- *Petrovi Protušovi* za vynikající lokalizaci do slovenštiny
- *Michaele Chomátové* a *Markétě Lorenzové* za mnoho hodin strávených nad vytvářením seznamu českých slov – z jejich práce těžím dodnes
- *Ing. Janu Koutníkovi*, *Ing. Zdeňku Beranovi CSc.*, *Ing. Adamu Sporkovi* a dalším za umožnění pracovat na tomto projektu v rámci školy
- svým kamarádům, přítelkyni a rodičům za vynikající pasivní podporu v dlouhodobé práci

[1] viz [7]. Moje práce není ojedinělá; nejstarší takovou prací je *Eliza*[17] Josepha Weizenbauma. Dnes existuje mnoho klonů, v češtině kromě *IQ Pokydu* např. program *Pokec*, *Kecal* nebo vynikající slovenský *Ludvik* – žádný z nich se však již nevyvíjí. Ze zahraničních je to hlavně nadějná *Alice*[1] nebo komerční *VirtualFem* a *Virtual Woman*.

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Kralupech nad Vltavou dne 20. srpna 2007 .....

Abstract

In this work, I tried to practically implement the computer processing of natural language – in this case Czech. I concentre mainly to lexical side a grammatical process of input sentence. Via grammar I implements specifying main semantic word in sentence. The result is program which demonstrate ability of this alghorithms.

Abstrakt

V této práci jsem se snažil co nejlépe prakticky implementovat počítačové zpracování přirozeného jazyka – zde konkrétně češtiny. Zaměřil jsem se především na slovotvorbu (lexikální stránka) a gramatické zpracování zadané věty. Pomocí gramatiky jsem dále implementoval určení významového slova ve větě. Výsledkem je program, který demonstruje schopnosti těchto algoritmů.

Obsah

Seznam obrázků

1 Úvod

Tato bakalářská práce je pokračováním mé dlouhodobější práce v oblasti zpracování přirozeného jazyka.

Rád bych hned v úvodu shrnul, o co přesně se budu snažit:

- Pod pojmem „přirozený jazyk“ je v této práci myšlena *vždy* čeština. Tento fakt je zde zejména proto, že žádný jiný jazyk neumím tak dobře jako právě češtinu (včetně různých odchylek, výjimek), proto program bere v úvahu právě tento jazyk, i přesto, že je poměrně obtížný. Není ovšem z taktické vázat program příliš pouze na češtinu; měl by být maximálně modulární, aby do budoucna bylo snadné přidat podporu dalších jazyků; v takové míře a rozsahu, v jakém to je u češtiny.
- Tato práce *není rešeršní*. Nesnažil jsem se v ní popsat různé způsoby dosažení téhož výsledku, a už vůbec ne např. v historických souvislostech. Místo toho je vybráno jedno konkrétní řešení, které se mi zdá nejlepší (případně je jen upozorněno na ostatní řešení) a to je podrobně rozebráno. U všeho jsem snažil na co *nejpraktičtější výklad*, tedy takový, podle kterého by šel postup ihned implementovat. Cítím se být mnohem více programátorem než teoretikem a program „nějak“ zvládající danou problematiku je pro mě cennější než stohy papíru se s podrobnou studií na téže téma.
- Téma počítačového zpracování přirozeného jazyka není možno obsáhnout celé. Jednak proto, že je samo o sobě velice rozsáhlé a vyžaduje hodně času na implementaci, a jednak proto, že některé odchylky musí rozhodnout až člověk na základě svých zkušeností a znalostí okolního světa. Přesto je dobré se tomu ideálu – porozumět přirozenému jazyku – co nejvíce přiblížit. Na tomto tématu hodlám pracovat i nadále, nezávisle na tom, co se ještě vejde do bakalářské práce a co ne. I přesto, že je to běh na hodně dlouhou trať.

2 Motivace

2.1 Cíle práce

Hlavním cílem této práce je naučit počítač „porozumět“ přirozenému jazyku v psané formě. Naučit ho postupy a znalosti tak, aby text zpracovával stejně tak dobře jako člověk. V psaném textu se dnes vyskytuje ohromné množství informací, avšak není je možné nějak „rozumně“ strojově zpracovávat – počítač tomu množství dat prostě nerozumí. Ideálem, kterému se chce přiblížit i tato bakalářská prací, je právě umožnit alespoň zčásti zpracovat některá data počítačem.

Téma zpracování přirozeného jazyka strojem je však značně obsáhlé a přesahuje možnosti bakalářské práce i časové možnosti a znalosti jednoho člověka. Tato práce se tedy zaměřuje především na:

- základní rozebrání řetězce obsahující větu/věty do slov
- zjišťování, jakého slovního druhu jsou daná slova
- téma skloňování, časování a odvozování slov včetně výjimek a překlepů (potřebné ke zjišťování slovního druhu) [9]
- základy gramatiky přirozeného jazyka [13] [5]
- aplikace konkrétní věty na konkrétní gramatiku [12]

2.2 Konkrétní podoba výsledku práce

Jak tedy vypadá výsledný program? Je to běžná desktopová aplikace napsaná v jazyce Java, která:

- načte zadaný soubor obsahující český text
- z textu vytáhne jednotlivá slova po větách
- načte seznam českých slov
- všechna tato slova vygeneruje do všech tvarů a pádů[1] a zaindexuje je pro rychlé vyhledávání
- u každého slova ze zdrojového textu určí na základě rozskloňovaných tvarů jeho možné slovní tvary. Najde i ty tvary, které by vznikly opravou překlepu (a ty do dalšího zpracování označí menší pravděpodobností)
- z dalšího souboru načte gramatiku
- na základě možných tvarů hledá odpovídající stavbu gramatiky – z toho vzniknou tzv. „gramatické stromy“, tj. žádná až mnoho různých gramatik, tj. způsobů, jak danou větu gramaticky chápat
- nalezené gramatické stromy vypíše

[1] přesněji do všech *gramatických kategorií*, tedy zdaleka nejen pádů

2.3 Existující implementace

V současné době je mi známa pouze jedna úspěšně nasazená implementace pro češtinu – výborný Grammaticon [10] od společnosti Lingea. Je to program pro kontrolu českého pravopisu, včetně oprav gramatických a stylistických chyb. Protože se však jedná o komerční program, nejsou mi známy žádné podrobnosti imlementace. Mé pokusy o debugování a alespoň částečné rozkódování logiky Grammaticonu[2] nebyly bohužel úspěšné.

Pak existují open-source kontroly pravopisu – např. ASpell [3] – ale bez gramatické kontroly.

Z výzkumných laboratoří je mi známa pouze Laboratoř zpracování přirozeného jazyka na Fakultě informatiky Masarykovy [6] zabývající se celou řadou témat zpracování češtiny.

Pro angličtinu toho existuje mnohem více. Jednak jsou to nasazené aplikace chatbotů (např. Alicebot [**?**]) nebo některé laboratoře - např. IBM [15] nebo Microsoft [16] zabývající se strojovým překladem, hledáním údajů v textu atd.

[2] samozřejmě pouze pro studijní účely

3 Analýza a návrh řešení

Jak už bylo zmíněno výše, tento projekt už více či méně tvořím cca 8 let. To je výhoda ve zkušenostech, ale i nevýhoda v používaných technologiích. Některé „kostlivce ve skříni“ ve skříních jsem odstranil, některé na odstranění teprve čekají.

3.1 Použitý programovací jazyk

Projekt zatím nemá žádné jedno konkrétní použití (a ani by ho neměl mít). Může proto běžet na desktopu stejně dobře jako na serveru nebo třeba v mobilním zařízení. Rozhodně by tedy použitý jazyk měl být *multiplatformní*. Na základě Ing. Jana Koutníka jsem zvolil Javu, na kterou jsem stávající program v C++ předělal.

3.2 Používané prostředí pro tvorbu

V současnosti používám pro tvorbu v Javě prostředí Eclipse běžící na Linuxu (Fedora 7). V dřívějších dobách jsem pro vývoj v C++ používal Borland C++ 3 (Caldera DR-DOS) a Microsoft Visual C++ 6 (Windows NT, 2000, XP). Použité prostředí dostatečně vyhovuje požadavkům na vývoj i na hladký běh programu.

3.3 Modulárnost kódu

Porozumění textu je velice obecné – text může být např. psán v různém jazyku nebo je různého typu. Každý jazyk má různou slovní zásobu, různý způsob tvoření slov, různé používané tvary slov i různou gramatiku. Všechny tyto věci se navíc neustále vyvíjejí[1]. Takže je vhodné toho co nejvíce „vystrčit ven“ z programu do datových souborů; udělat tak program univerzálnější a lépe spravovatelný. Nicméně přitom také stoupá doba prvotního naprogramování – a taky ne vždy, zejména na začátku, mě ta myšlenka napadla – takže nyní je na tom projekt jen o něco lépe.

3.3.1 Slovní zásoba

Slovní zásoba, jinak řečeno seznam slov v jazyce používaných, (naštěstí) příliš zakomponovat do kódu nejde. Slova jsou tedy načítána ze souboru. U každého je napsán jeho slovní druh, u ohebných druhů ještě vzor a popř. výjimky při skloňování/časování. U neohebných druhů je napsáno, do kterého poddruhu dané slovo patří (např. u předložky s kterým je spojeno pádem).

Zde je ten problém, že existuje spousta slov, jejichž gramatický význam je de facto jedinečný a lze ho špatně popsat. Jsou to typicky částice a různé ustrnulé tvary – např. *ahoj*, *proboha* nebo hovorové slovo *na* (ve smyslu, když někdo někomu něco

[1] Nejrychleji se mění slovní zásoba, pomaleji už stavba slov a jen velmi zřídka gramatika [18].

dává). U těchto slov se musí sáhnout kromě standardního zatřídění do vzoru i po hlubším rozlišení možného použití.

3.3.2 Možné druhy slov

Seznam druhů slov (jako podstatné jméno, zájmeno, sloveso . . .) je v kódu napevno. Nejsem si vědom toho, že by nějaký jazyk mohl mít ještě nějaký jiný, češtinou nepoužívaný druh, takže zde ta závislost (zatím) nevadí. Je ale možné, že se časem objeví případ, kdy 10 slovních druhů přestane stačit.

3.3.3 Možné vzory slov

Možné vzory pro každý slovní druh jsou také v kódu napevno. Plně si ale uvědomuji, že je to chyba – každý jazyk má svoje vzory. Při návrhu programu jsem proto dal tyto „jazykově závislé“ třídy do speciálního balíčku jen pro češtinu, nicméně načítání vzorů z externího souboru by bylo mnohem lepší. Cesta tvoření balíčků speciálních pro každý jazyk je obecně špatná a zde není bezpodmínečně nutná.

3.3.4 Možné vlastnosti vzorů slov

Těmi „vlastnostmi“ myslím to, že např. podstatné jméno má svůj pád a číslo, sloveso má osobu, číslo, čas a vid[2], ale také např. to, že pád může být číslo 1 až 7. Tyto vlastnosti jsou také „zadrátovány“ přímo v kódu. Sice jsou taktéž ve speciálním balíčku tříd pro češtinu, ale uvědomuji si, že z dlouhodobého hlediska je tento návrh špatný.

3.3.5 Skloňování jednotlivých vzorů

Zde mám na mysli popis toho, jak se skloňuje podle jednotlivých vzorů, jak se třeba utvoří 2. pád množného čísla u vzoru muž. Zde program částečně prošel evolucí – zpočátku jsem skloňování jednotlivých vzorů měl také přímo v kódu, postupným přepisováním jsem si ale uvědomil svou chybu a tak jsem popis dalších vzorů dal do externího souboru. Výsledkem je „mišmaš“, kdy všechny vzory až do vzoru „bere“ jsou v kódu napevno, ale ostatní vzory (slovesa) jsou v externím souboru a program umí pracovat s obojím. Přepisu těch prvních vzorů do externího souboru mi bránilo jednak zatím nevyřešení některých nepříjemných výjimek při skloňování a jednak relativní zbytečnost takového úkonu.

Do budoucna by toto dvojité načítání chtělo sjednotit.

[2] a další vlastnosti, to však zde není podstatné

3.3.6 Gramatika stavby věty

Gramatika stavby věty se zcela načítá z externího souboru, na použitém jazyku to vůbec nezávisí. Jediné, co je zatím nedořešené, je seznam možných terminálů v gramatice – ten je zatím určen napevno.

3.4 Formát dat

3.4.1 Formát datových souborů

Program využívá několik různých datových souborů. Každý z nich je uložen trochu jinak:

- **databáze slov jazyka a jejich vzorů** – je to čistě textový soubor, jedno slovo na řádek. Každé slovo pak má u sebe nula až několik doplňujících parametrů (vzor, výjimky atd.). Slova jsou setříděna podle abecedy, ale ne nutně

  Příklad formátu:
  ```
  .prales: 1,hrad,{2p-jc:pralesa,pralesu}
  .pramen: 1,hrad
  ```

- **databáze slov ve všech možných tvarech** – tento soubor vzniká automaticky z databáze slov jejich rozskloňováním. Protože se sám přepisuje při změně seznamu slov, není žádoucí jeho ruční úprava. Pro uložení je zvolena binární podoba dat zabalená kompresí *gzip*. Slova jsou zde striktně setříděná podle ASCII kódu, využívá se zde také komprese díky velké podobnosti dvou po sobě jdoucích slov a dále komprese nahrazením atributů slova jejich zaindexováním a uváděním pouze pozice vlastnosti v seznamu (díky omezenému počtu vlastností)

- **soubor se seznamem vzorů a jejich skloňováním** – textový soubor, který obsahuje jednotlivé vzory např. pro slovesa. U každého vzoru pak je pomocí maker popsán způsob vytvoření tvaru. Jsou zde možné i podmínky např. na koncovku slova (pro rozlišení poddruhů slov)

  Příklad formátu:
  ```
  .maže:
   orizni_samohlasku_na_konci
   zdrsni_koncovku_slovesa
     [cm-jc-rm]: -al
     [cm-jc-rz]: -ala
     [cm-jc-rs]: -alo
     [cm-mc-rm]: -ali
     [cm-mc-rz]: -aly
     [cm-mc-rs]: -ala,-aly(n)
  ```

- **soubor s gramatikou jazyka** – jedná se o textový soubor, který popisuje gramatická pravidla ve tvaru `Neterminál` $\Rightarrow$ *pravá strana*. První neterminál na levé straně značí kořenový neterminál věty

Příklad formátu:

```
Veta -> <cro>Podmet <cro>Prisudek1a = člověk dělá
Veta -> Prisudek1b                  = dělá; prší
Veta -> <cro>Prisudek1b <cro>Podmet = jede vlak

Podmet -> PodstJmeno1[1p]#[g]
```

Pro čtení některých textových souborů používám vlastní implementaci lexikálního a syntaktického analyzátoru [5].

3.4.1.1 Textový formát vs. XML

Proč pro textové soubory nepoužívám formát XML? Formát XML je sice široce rozšířený a všeobecně uznávaný standard, pro který existuje nativní podpora ve většině programovacích jazycích, avšak má jednu vadu – sám o sobě je špatně čitelný pro člověka. Zatímco pro stroj je takový formát výborný, způsob zápisu je pro člověka nepřirozený. A protože tyto soubory jsou určeny k přímé editaci, ne prostřednictvím editoru, je vhodnější čistý text.

3.4.1.2 Binární formát

Pro uložení rozskloňované databáze využívám binární formát, značně komprimovaný. Toto je možná z dnešního hlediska přežitek – binární formát ztěžuje ladění a kontrolu databáze, protože do značné míry znemožňuje nahlédnutí do dat. Tento způsob vychází z dřívějšího stavu, kdy desítky megabajtů potřebné pro uložení databáze byly dost. Myslím ale, že i dnes má toto zhuštění díky binárnímu kódování a umožnění další komprese opravdu smysl, zvláště když je komprese oproti rozbalené verzi cca stonásobná.

Právě menší velikost jinak velkého souboru je tedy důvodem ke zvolení uložení v binární podobě.

3.4.2 Kódování souborů

Protože se všechny české znaky nevejdou do základní ASCII tabulky, je nutno použít nějaký způsob pro uložení *ne*ASCII znaků. Zde je veškerý text v souborech uložen v kódování *windows-1250* (označovaného také jako *cp1250*). Volba tohoto kódování není zrovna šťastná, a to především z toho důvodu, že v něm lze uložit pouze znaky používáné ve střední Evropě. Tím pádem je prakticky znemožněna podpora jiných jazyků nebo i uložení cizího slova obsahujícího nějaký nestandardní znak. Nejlepší volbou pro kódování se proto zdá být *utf-8*, které při zachování minimální velikosti umožňuje uložit veškeré znaky Unicode.

Stávající kódování je dáno technickým omezením v době začátku vývoje programu. Později již nebyla změna kódování tak jednoduchá (zejména kvůli proměn-

nému počtu bajtů reprezentující jeden znak), takže zatím tento nedostatek není opraven. Nicméně ho zatím chápu jako minoritní.

4 Realizace

V této kapitole tedy ukážu, jak z běžného textu dostat nějakou počítačově zpracovatelnou informaci. Tento úkol samozřejmě není nijak snadný. Je třeba text zpracovat na několika úrovních, *vrstvách*. Podíváme se tedy na jednotlivé vrstvy, jak jdou za sebou od hrubého textu až po strom gramatiky ukazující návaznosti slov na sebe.

4.1 První vrstva: Základní rozbor

Zapsat text není v počítačových programech problém. Je to jednoduše posloupnost znaků, kde každý znak je z nějaké konečné množiny znaků. Dejme tomu, že již máme takovouto posloupnost znaku někde uloženu.

Text se skládá z vět a věty se skládají ze slov. Ač se to na první pohled nemusí zdát, nelze na této vrstvě od sebe bezpečně rozeznat věty. Slova ano[1] – jednoduše tak, že jsou od sebe oddělená mezerou, čárkou atd. Tímto způsobem můžeme vytáhnout z textu všechna slova a uložit je do pole řetězců. Každé slovo je jeden řetězec. Někam „vedle“ si pak můžeme uložit i informaci o tom, kde byla v textu čárka, tečka středník, pomlčka atd.; tyto informace budeme potřebovat na třetí vrstvě. Vhodné je uložit si i jestli bylo na začátku velké písmeno, celé slovo je velkými písmeny atd. a velikost písmen pro další zpracování pak sjednotit.

4.2 Druhá vrstva: Slovotvorba

Na druhé vrstvě potřebujeme ke každému jednotlivému slovu zjistit, o jaký *tvar*[2] slova se může jednat. Tím mám na mysli, jestli je to podstatné jméno, přídavné jméno, sloveso atd., v jakém je pádě, čísle, rodu... U většiny slov navíc může být možných tvarů více (např. jednomu tvaru může odpovídat více pádů téhož slova[3]). Nás zajímají všechny tvary, které teoreticky připadají v úvahu pro každé slovo.

Tady je vhodné říci, jakým způsobem lze vlastně zjistit, o jaký tvar slova se jedná. Pro zjištění existují dva způsoby:

- Pomocí formátu slova: každý druh má specifický tvar, např. příponu, předponu, a podle toho lze od sebe jednotlivé druhy bezpečně poznat. Příkladem takových jazyků je např. *esperanto* nebo *interlingua*. Nespornou výhodou těchto jazyků je, že zde nemusí být žádný slovník, velmi lehce se vytváří slova nová a vůbec – velice dobře se s nimi pracuje a proto jsou pro potřeby počítačového zpracování tyto jazyky velice vhodné.

[1] někdy ani slova od sebe nelze bezpečně rozpoznat – např. u slov typu *česko-německý*, *bude-li*, *sci-fi* apod. Pro další rozbor tyto výjimky vynechám, řešily by se pravděpodobnostní analýzou podobně jako překlepy

[2] označení *tvar* je zde trochu nepřesné, lepší termín by byl *gramatická kategorie*. Vzhledem k jednoduchosti však budu používat termín *tvar*

[3] a často také připadá – např. slovo *hradu* může být od slova *hrad* 2., 3. nebo 6. pád

Nevýhodou je, že se tyto jazyky v praxi příliš nevyužívají (a kdyby začaly, svoji pravidelnost by ztratily[4]).

- Pomocí seznamu slov: Pro každý jazyk je definován seznam slov, resp. jeho podmnožina[5]. U každého slova zvlášť je pak definován jeho slovní druh, popř. vzor, výjimky skloňování/časování atd.

  I u takto definovaného jazyka lze s úspěchem[6] rozeznat správný tvar dosud neznámého slova; to lze zpravidla na základě koncovky, předpony a porovnáním se vzory známých slov a postavením ve větě, tj. porovnáním s předpokládanými tvary slov. Nicméně prvotní tvorba slovníku zabere vždy určitou námahu, až později se může slovník doutvářet na základě vstupů.

Pro běžný přirozený jazyk – a tedy náš případ – je jasné, že každé slovo musí být analyzováno druhým způsobem, tj. pomocí předem daného seznamu slov.

4.2.1 Vytváření seznamu všech možných slov

Před prvním zjišťováním tvaru slova je nutné všechna slova rozskloňovat, tedy každé slovo se převede do všech možných[7] podob[8]. Takže potom vznikne velký slovník, jehož každý prvek je dvojicí tvar : atributy. Těch atributů bývá dost – kromě gramatických je vhodné zapsat i takové vlastnosti slova jako „nespisovné“, „nevhodné“ nebo „toto zájmeno nelze použít na začátku věty nebo v důrazu“ atd. Vhodné je také napsat gramatický původ slova, např. slovo *nedodělanost* je odvozeno jako „dělá“ ⇒ „dělaný“ ⇒ „dodělaný“ ⇒ „nedodělaný“ ⇒ „nedodělanost“.

Je jasné, že se tyto atributy značně liší jazyk od jazyka; třeba pro češtinu jich u každého slova mám 19[9].

Tato slova se potom kvůli lepšímu indexování setřídí, stejné tvary se seskupí do jednoho. Na vyhledávání je možno použít klasické binární půlení, v případě vlastního vysoce optimalizovaného hledání na výkon (porovnávání jen některých znaků ve slově, detekce neexistující části slova, cache . . .) lze dosáhnout nalezení/nenalezení slova za milisekundu. Výkon je u tohoto hledání důležitější, než by se mohlo zdát – musí se zde počítat i s možností překlepů.

---

[4] každý používaný přirozený jazyk se nutně vyvíjí; vývoj je jedna z podmínek jeho užívání[18]. Vývojem okolního světa a změnami četnosti používání jednotlivých slov však nutně jazyk přejímá různé výjimky. Snad je tedy dobře, že se tyto jazyky nikde nevyužívají jako hlavní jazyk.

[5] obsáhnout všechna slova není u běžných jazyků možné

[6] samozřejmě většinou, problémem jsou zejména různá cizí slova mající koncovky podobné např. jiným slovním druhům. Pro naše účely je ale pozitivní, že obecně se nová slova skloňují pravidelně; právě proto, že jsou nová[18]

[7] téměř všech – některá nejsou nutná a možná ani žádoucí – např. u sloves veškeré přípony a záporky *ne*

[8] naprosto neocenitelným pomocníkem při skloňování a časování je knížka [9]. Výborně (i když už ne v takové míře) se ale také může hodit takřka jakákoli mluvnice pro základní školy, např. [8]

[9] na toto číslo jsem přišel empiricky

4.2.2 Překlepy a dohledání správného slova

Samotné vyhledávání by takhle fungovalo, ale co když se člověk při psaní uklepne a místo „schůzka“ napíše třeba „schůkza“ nebo „zchůzka“? Nebo co když píše bez diakritiky – je pak nutno udržovat v paměti obě verze celého rozskloňovaného jazyka?

To se samozřejmě může stát a je vhodné to ošetřit. Zde je ovšem potřeba výkon. Nepřišel jsem totiž na žádný lepší způsob než zkoušení vkládání/ubírání/prohazování/záměna všech písmen ve slově, a to ještě rekurzivně (když jsou ve slově chyby třeba 2). Při každé záměně písmen je také snižována pravděpodobnost takové možnosti (je pravděpodobnější, že se člověk neuklepl). Každý typ záměny má navíc přiřazenu jinou pravděpodobnost (např. člověk si všimne překlepu spíše na začátku než uprostřed slova[10]). Při určité míře snižující se pravděpodobnosti pak už prakticky nemá smysl dál pokračovat – vznikají tak úplně odlišná slova.

Při tomto prohledávání tak vznikají trojice tvar : atributy : pravděpodobnost[11]

I když je tato operace také zefektivněna (např. nejsou prohledávány kombinace, když první část slova neexistuje), u každého slova se proto slovníku porovnává několik set tisíc kombinací (při hloubce rekurze 3). Tím se dá dosáhnout rychlosti asi 6 slov za vteřinu – to věru není mnoho[12], věřím však, že do budoucna přijdu na další optimalizační metody.

Takto nalezená slova postupují ke zpracování gramatikou, viz další podkapitola. K tomu je však někdy vhodné slova uzpůsobit. V češtině např. existuje skupina slov *-bych/-bys...* (např. ”kdybych”, ”abyste” atd.), které se sice tváří jako jedno slovo, ale gramaticky to jsou slova dvě – *kdyby jsem*, *aby jste* atd. Je to trochu taková češtinářská zrada[13], ale rozdělit tato slova na dvě před gramatickým zpracováním je vhodné. S tím ale samozřejmě vzniká další problém – chyby ve vstupu, překlepy. Pak je nutno generovat gramatiku z rozděleného tvaru a možného nerozděleného (za předpokladu, že by se jednalo o překlep) a každý případ by si s sebou nesl příslušnou počáteční pravděpodobnost.

4.3 Rozbor gramatiky přirozeného jazyka

Gramatika je způsob, jakým jsou poskládána slova, aby vznikla smysluplná věta, která něco označuje. My se budeme snažit o opačný způsob – získání významu nebo spíše návazností ve větě, přičemž tvar jednotlivých slov už známe.

O co nám tedy přesně jde? Nejedná se ještě o význam věty jako takový, ale spíše o „seskupování dat“. Nyní nám tedy stačí zjistit, že např. věta *Navrhuji tedy schůzku*

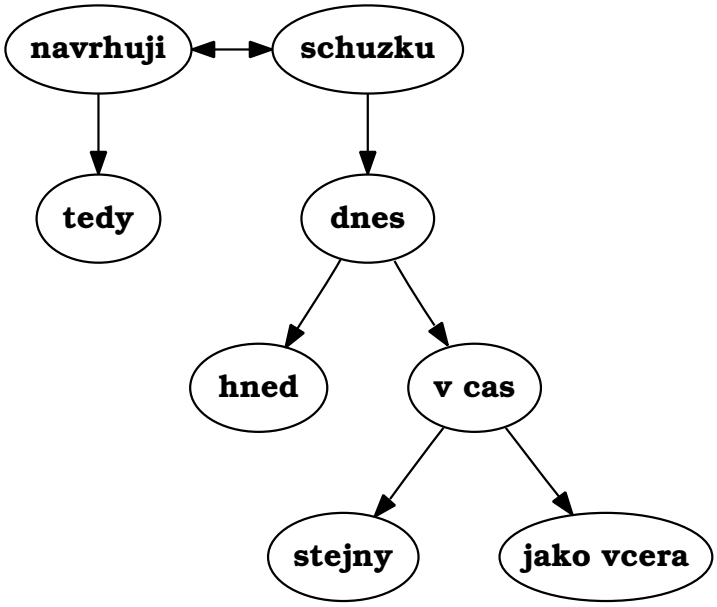
[10] dobré pojednání o tom, jak vlastně člověk čte psaný text je na [2]

[11] S pravděpodobností se počítá i dále při gramatice; vlastně jakýkoli vstup na jakékoli vrstvě je zatížen nějakou možnou nepravděpodobností. Je to daň za to, že přirozený jazyk sám o sobě není jednoznačný.

[12] A bude hůř – ještě není implemetováno testování předpon. To bude mít na výkon také velký (negativní) vliv

[13] V jiných jazycích se nepochybně vyskytuje něco obdobného. Pravopisně chybné, ale gramaticky správné tvary „kdyby jste“ atd. jsou častou chybou. Protože se však čeština neustále vyvíjí směrem k benevolenci a k logičtějším tvarům (jako každý jazyk [18]), věřím, že pravopisně nyní zavrhované tvary se dříve či později stanou „oficiálně“ doporučovanými.

hned dnes ve stejný čas jako včera lze podle závislostí slov zařadit do stromu podle obrázku 4.1.



Obrázek 4.1: Gramatický strom věty

4.3.1 Základní fakta o přirozené gramatice

Jeden z problémů analýzy gramatiky je fakt, že žádná gramatika přirozeného jazyka není jednoznačná. A mnohdy se lehké odchylky dají odlišit jen tím, že člověk jako takový ví, co z toho „dává smysl“.

Vezměme si např. větu

```
Viděl jsem lampu na poličce, která byla špinavá.
```

To je sice špatně formulovaná, ale gramaticky správná věta. Co bylo vlastně špinavé – lampa nebo polička? To z věty vůbec není jasné, nicméně gramatika by v tomto případě zřejmě dala přednost rozvinutí příslovečného určení před rozvinutí předmětu. Takže by byla špinavá ta polička. Je to velmi jemná nuance, která se ale ještě dá odchytit na úrovni gramatiky.

Že se ale dá odchytit jen velice zběžně, dokazuje další příklad:

```
Viděl jsem lampu na poličce, která svítila.
```

Gramaticky se nezměnilo prakticky nic. I nadále by gramatika dala (velice mírnou) přednost tady už svítící poličce před svítící lampou. Ovšem rozum a hlavně zkušenost a znalost okolního světa by dala přednost svítící lampě. Svítící poličku totiž zřejmě nikdo neviděl, zato špinavé může být téměř cokoliv.

Z důvodu této nejednoznačnosti dostaneme na výstupu gramatické analýzy něco podobného jako u rozboru slov – seznam možných stromů závislostí a u toho jejich

pravděpodobnosti. Která z nich je správná, musí rozhodnout až zkušenost. Jenže počítač může těžko aplikovat nějaké předchozí zkušenosti, jelikož zpravidla žádné nemá[14].

4.3.2 Tvar gramatiky přirozeného jazyka

Gramatiku přirozeného jazyka se snažilo popsat mnoho lingvistů[15] a filosofů[16]).

Gramatiky obecně lze rozdělit na 4 typy [13]:

- **Typ 0 – neomezená přepisovací pravidla**: nejobecnější zápis gramatických pravidel ($\alpha \Rightarrow \beta$)
- **Typ 1 – kontextové gramatiky**: pravidla ve tvaru $\alpha A\beta \Rightarrow \alpha\gamma\beta$, přičemž neterminál A nelze nahradit prázdným řetězcem – tj. gramatiku nelze expandováním zkrátit
- **Typ 2 – nekontextové gramatiky**: pravidla ve tvaru $A \Rightarrow \gamma$, kde γ může obsahovat terminály i neterminály
- **Typ 3 – regulární gramatiky**: pravidla ve tvaru $A \Rightarrow xB$. Tyto gramatiky se počítačově zpracovávají nejlépe, protože jsou *jednoznačné*

Ukázalo se, že nejvhodnější reprezentací pro gramatiku přirozených jazyků jsou pravidla *nekontextové gramatiky*. Ačkoli některé práce (např. od Chomského [4]) ukazují, že nekontextové gramatiky jsou nedostačující a zavádí se zde *transformační gramatiky*, další vývoj ukázal, že tyto nové gramatiky nejsou potřeba [13].

V této práci jsem tedy použil nekontextovou gramatiku. Z praktických důvodů je použita pouze její podmnožina – program nepřipouští zkracování gramatik expandováním, tj. počet pravidel na pravé straně každého pravidla je vždy menší nebo rovno počtu slov, která může obsáhnout. Toto omezení je zde z toho důvodu, že se při postupné expanzi lépe filtrují nesmyslné stromy, protože jejich gramatický zápis je příliš dlouhé. Ačkoli je toto omezení „umělé“, zatím jsem ho nemusel porušit. Některé pravidla se však zapisují těžkopádněji.

4.3.3 Princip expandování gramatiky

Každý gramatický člen se tak může rozpadat na jeden nebo více dalších gramatických členů, ať už přímých (*terminály*) nebo nepřímých (*neterminály*). „Nejvyšším“ gramatický členem je *Věta*, resp. obecněji *Text*.[17]

V praxi to funguje takto: Mějme např. text

[14] více o možných předchozích „zkušenostech“ počítače viz [14]
[15] např. [13] nebo [12]
[16] velice dobře popsal gramatiku přirozeného jazyka Noam Chomsky[4]
[17] Asi nemá smysl popisovat princip fungování LL gramatik. Pro další informace viz např. [5]

```
Ty tmavé štafle jsou na chodbě.
```

Základním (tím „prvním“) neterminálem je *Věta*. Ten můžeme expandovat na neterminály *Podmět*[18], *Přísudek*. *Podmět* dále expandujeme na *zájmeno* a *PodmětBezZájmena*. Zbývající neterminál *PodmětBezZájmena* expanfuji na *Přísudek* a *PodmětBezZájmena* a ty potom nahradím terminály *přídavné-jméno* a *podstatné-jméno*[19]. Část s neterminálem *Přísudek* se rozpadne opět na *Přísudek* a *PříslovečnéUrčení*. Neterminál *PříslovečnéUrčení* se dále rozpadne na *předložku* a *PodstatnéJménoVŠestémPádě*, to se potom rozpadne na příslušný terminál.

Každý terminál se pak skládá z vlastního druhu slova a volitelně i z atributu, který blíže určuje např. pád nebo číslo, kterých může nabývat.

4.3.4 Pokročilé vlastnosti gramatiky

4.3.4.1 Shoda podmětu s přísudkem

Takovýto jednoduchý zápis ale bohužel pro gramatický popis jazyka nestačí. Čeština například uplatňuje shodu podmětu s přísudkem. To znamená, že podmět a přísudek musí mít stejný osobu, číslo a rod. Jak toho docílit?

Existují dva způsoby [12]:

- **Vypsáním všech možností zvlášť** – pro každou kombinaci osoby, čísla a rodu bude speciální pravidlo. Při použití tohoto řešení se nijak nemusí měnit gramatika jako taková, ale vzhledem k počtu kombinací (např. 7 pádů, 2 čísla, 3 osoby atd.) dojde ke značnému nárůstu počtu pravidel. Gramatika se tak stane velice nepřehlednou, navíc kvůli počtu pravidel se i časově ztíží její derivace.

- **Zavedením atributů do gramatiky**[20] – do gramatiky se zavede nějaký způsob popisu shodnosti. Můžeme tak zapsat, že např. neterminál *Věta* lze expandovat do *Podmět Přísudek*, ale Podmět a Přísudek se shodují v osobě, rodu a čísle.

V programu je použit druhý způsob, jsou tedy zavedeny atributy do gramatiky. V praxi to tedy vypadá např. takto:

```
Veta -> <cro>Podmet <cro>Prisudek1a
```

kde *<cro>* před neterminály na pravé straně znamená „shoda v čísle, rodu a osobě“. Je třeba mít na paměti, že jeden (ne)terminál může být závislý i na několika různých dalších slovech zároveň a to různými způsoby.

Otázka je, jak pak takovou gramatiku derivovat. Existují zase dva způsoby:

[18] Mohli bychom ho nazvat také jako *PodstatnéJménoVPrvnímPádě*

[19] u každého terminálu se ještě definuje možný tvar, tj. v jakém je pádu atd. Pro jednoduchost neuvádím, ale je to potřeba vědět

[20] Termín „atribut“ se u gramatik používá v lehce odlišném významu

- **Rozepsáním všech možností** – před použitím gramatiky program automaticky všechny pravidla s atributy automaticky rozdělí na sadu pravidel, kde každé pravidlo bude představovat jednu kombinaci. Toto je řešení poměrně nenáročné na další zpracování, ale vnitřně se zvyšuje počet pravidel a s tím i doba běhu programu kvůli derivaci.

- **„Probubláním“ (zděděním) atributů do nižších vrstev gramatiky** – při expanzi se atribut shodnosti, který s neterminálem svázán, dostane do nižší vrstvy, na kterou se aplikuje.

V programu je opět použit druhý (a pravděpodobně lepší) způsob. Vyvstává zde ale další problém – ke kterému (ne)terminálu na pravé straně pravidla přiřadit zděděný atribut. To je také potřeba v gramatice určit.

4.3.4.2 Zpracování gramatických závislostí mezi slovy na několika vrstvách

V gramatice programu je tedy zaveden další atribut – *gramatická závislost*. Tento atribut říká, že terminál/neterminál, u kterého je, přebírá veškeré závislosti svého předka, tj. expandujícího terminálu.

Se závislostmi a atributy již tedy lze jednoduše popsat gramatickou shodu několika slov, např. shody podmětu s přísudkem nebo shodu podstatného jména s jeho rozšiřujícím přídavným jménem.

V praxi je to v gramatice popsáno takto:

```
PodstJmeno3 -> <pcr>PridJmeno1#[g] <pcr>podstJmeno#[g]
```

kde $\#[]$ znamená *shodu* a g znamená *gramatickou*.

4.3.4.3 Klíčová slova ve větě

Pomocí dědění u gramatiky lze také popsat možná klíčová slova. Pomocí statistiky typických vět (viz kapitola 4.4) lze vysledovat, v jakém kontextu gramatiky se nejčastěji vyskytují nějaká klíčová slova (tedy slova s předmětem sdělení).

Z gramatického pohledu se klíčová slova chovají stejně jako atributy, s tím rozdílem, že klíčové slovo existuje samo o sobě. Takže pro něj stačí zavést další kategorii, např. *závislost klíčového slova* a pak je třeba ho také nějak definovat. V praxi to vypadá podobně jako u atributové závislosti – deklarace:

```
Prisudek2b -> Prisudek3bab#[g] {klicove_slovo}PodstJmeno1[4p]
```

a použití (modifikace předchozího případu):

```
PodstJmeno3 -> <pcr>PridJmeno1#[g] <pcr>podstJmeno#[gk]
```

4.3.5 Způsob generování gramatiky strojem

Možností, na které se může daný neterminál rozpadnout, bývá více[21]. Stroj ale samozřejmě neví, podle jakého pravidla má neterminál expandovat, aby mu to přesně „vycházelo“ do vstupní věty.

Klasické gramatiky[22] mají tu vlastnost, že pravidlo, na které se bude neterminál expandovat, lze jednoznačně určit pomocí „nahlédnutí“ na to, jaký terminál je první v expandované části. Takže výslednou posloupnost pravidel lze vytvořit pomocí jednoho průchodu. Těmto gramatikám se říká *regulární gramatiky*[23]. Bohužel, gramatiky přirozených jazyků tuto vlastnost *nemají*, tj. nelze v okamžiku expanze jednoznačně určit, podle kterého pravidla se tak má stát.

A toto je hlavní problém generování stromů gramatiky obecné LL gramatiky. Abychom je tedy mohli vytvářet, musíme tento problém nějak obejít. Tu vlastnost „nahlédnutí“ zde můžeme použít taky; když je např. první neterminál po *Přísudku* přídavné jméno, je jasné, že se nebude expandovat na příslovečné určení, protože to má vždy na začátku předložku. Tím tedy můžeme zúžit výběr „kandidátů“ pouze na ty, které můžou mít na začátku daný terminál.

Nicméně, to může být stále více než jedno pravidlo. Do dalšího generování tedy neterminál expandujeme *všemi přípustnými pravidly*, čímž tedy z jedné výstupní gramatiky uděláme několik možných a dále pracujeme se všemi z nich.

Pokud zjistíme, že nějakou gramatiku už nelze dále expandovat (neexistuje příslušné pravidlo), dostali jsme se do slepé uličky – příslušný strom tedy zahodíme jako neplatný. Tím se nám výběr opět zúží. Pokud už nezbývá žádný platný strom, zadaná věta není gramaticky správná.

4.3.6 Problémy generování gramatických stromů

I přes všechny prořezávání možných kombinací stromů zde často vzniká mnoho stromů – někdy řádově i stovky. To je poměrně hodně z hlediska výsledné rychlosti algoritmu. A co hůře; může zde snadno vzniknout *zacyklení*, tj. že jedno pravidlo lze expandovat do nekonečna.

Podívejme se na následující příklad pravidla:

```
    PřídavnéJméno ⇒ PřídavnéJméno přídavné-jméno
PřídavnéJméno ⇒ přídavné-jméno
```

První pravidlo za předpokladu, že první terminál čekající na expanzi, může (a bude) expandovat do nekonečna; tj. bude vznikat nekonečně dlouhé pravidlo, kde budou samé neterminály *PřídavnéJméno*. Přitom je to tak správně; podstatné jméno lze *teoreticky* upřesnit jakýmkoli množstvím přídavných jmen vložených před podstatné jméno. Otočit neterminál a terminál na pravé straně také nelze, protože to zase

[21] jinak by ten neterminál v podstatě neměl smysl; mohl by být expandován rovnou
[22] např. na popis počítačových jazyky
[23] popis klasických jazyků viz [5]

neodpovídá sémantické hierarchii slov ve větě[24]. Co tedy s tím?

Lze udělat několik věcí:

- **omezit počet uplatnění jednoho pravidla** – poměrně účinné, ale v nějakých extrémních případech se může stát, že program gramatiku nerozezná, protože člověk jedno pravidlo uplatnil vícekrát než je limit
- **omezit maximální počet minimálního počtu terminálů gramatiky** – jinými slovy, počítat kolik minimálně terminálů v gramatice může být a v případě, že je to počet vyšší než je počet slov ve větě, přestat expandovat. Toto omezení je ideální v případě, pokud je věta krátká (cca do patnácti slov). Pak už ztrácí na významu, pro proudové zpracování předem neznámého textu nepoužitelné
- **při každé „podexpanzi" o něco snížit pravděpodobnost takového stromu** – je nutné expandovat stromy prioritně podle pravděpodobnosti, pak takovéto nepravděpodobné stromy přijdou na řadu až po vygenerování elegantnějších (a zřejmě správných) stromů. Nicméně i tak *někdy* přijdou na řadu, je proto vhodné zároveň omezit minimální pravděpodobnost nebo aplikovat nějaké z výše uvedených dalších omezení

Ani jedno řešení není ideální, nicméně kombinací omezení lze i tak dosáhnout dobrých výsledků.

4.3.7 Omezení počítačového generování gramatických stromů

Ačkoli jsou současné gramatické popisy přirozených jazyků poměrně dost přesné, nikdy v praxi nebyly na rozpoznání gramatických stromů moc dobře použitelné.

Je to hlavně ze dvou důvodů:

- **Gramatika vytváří příliš mnoho stromů** – v gramatice jsou správně popsána všechna pravidla a kdy je možno je použít. Z důvodu nejednoznačnosti, kterou člověk rozliší jen významově, nikoli gramaticky, často vznikají mnoho stromů, které jsou sice také gramaticky správné, ale v daném významu/kontextu absolutně nepoužitelné. To však stroj není schopen posoudit. Z tohoto důvodu může někdy počítač upřednostnit nesmyslný význam věty před smysluplným, ale netradičně vyjádřeným; ze stejného důvodu bývají počítačové kontroly pravopisu (gramatiky) podivně „toleratní"[25].
- **Gramatika je málo tolerantní** – protipólem k tomu je případ, kdy se počítači gramatiku nepodaří rozluštit, i když má k dispozici všechna gramatická pravidla. Někdy se totiž objeví gramaticky chybná věta, ale přesto natolik srozumitelná,

[24] Příklad: *malý bílý kámen* je *bílý kámen*, který je navíc *malý*. Nelze to však chápat jako *malý kámen*, který je *bílý*, ačkoli je to ve výsledku totéž

[25] poučné čtení, jaká jsou omezení, resp. tolerance u kontroly českého pravopisu je na [11]

že člověk nemá nejmenší problém ji přečíst a porozumět[26]. Zavést nějakou „toleranci“ do gramatiky je poměrně problematické, zvlášť když to vede na ještě více výsledných gramatických stromů.

I přes uvedené neduhy zmíněné v této kapitole – které se zejména ze sémantického hlediska špatně řeší[27] – jsou gramatické postupy na větší část případů celkem dobře použitelné.

4.4 Základy zpracování významu

Jedním z úkolů počítačového zpracování přirozeného jazyka je tvorba rešerší nebo hledání klíčových slov. Lze to dělat několika způsoby[28]:

- **Jen podle slov** – téma textu se odhaduje jen podle výskytu slov, přičemž se téměř nebere v úvahu kontext těchto slov. Na tomto principu[29] pracují současné internetové vyhledávače.
- **Na základě gramatických vazeb** – text se převede podle vztahů na gramatické stromy a v gramatice se určí, které typy pravidel obvykle nesou významovou informaci.
- **Na základě významu textu** – text se *nějakým* způsobem převede na význam a z toho se pak udělá výtah. Takto pracuje člověk.

V této práci je jednoudše implementována druhá možnost – tedy hledání *klíčových slov* podle gramatických vztahů.

4.4.1 Způsob hledání gramatického pravidla pro klíčové slovo

Na základě nějaké statistiky je potřeba zjistit, který gramatický vztah (které gramatické pravidlo) obvykle expanduje na to stěžejní slovo ve větě.

Při vytváření statistiky jsem postupoval následovně:

- Sepsal jsem si různá klíčová slova, která by mohla být eventuálním obsahem sdělení.
- Na každé slovo jsem vytvořil několik vět, ve kterých opravdu byly tím podstatným. Tyto věty musely být derivovatelné stávajícími gramatickými pravidly.
- Spustil jsem derivaci gramatiky na každou větu a díval se, jaké společné pravidlo mají v oblasti klíčového slova, ideálně které se jinde nevyskytuje

[26] Např. u věty *Ten na to by se vykašlal* jsou slova *by se* na špatném místě. Gramaticky je to špatně, člověku to však příliš nevadí. Našly by se ale i lepší příklady. Viz též [2]

[27] viz [14]

[28] Toto rozdělení je spíše orientační.

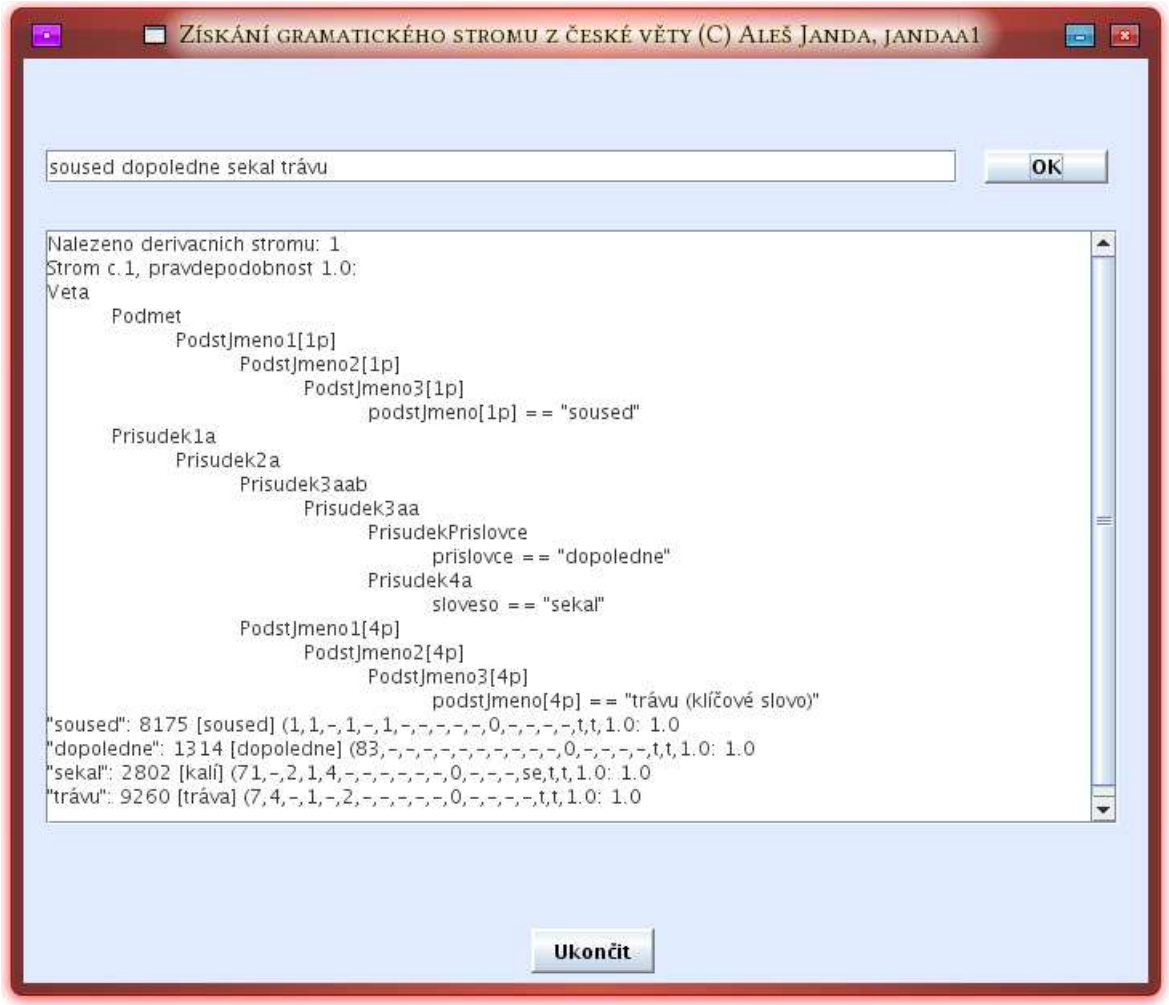
[29] Minimálně u obecných stránek.

Až na pár výjimek bylo takovým společným pravidlem pravidlo, které definovalo větný člen *předmět*. Tento větný člen se asi jen ve 3 případech vyskytoval jinde než u obsahu sdělení, dá se tedy s poměrně velkou pravděpodobností říci, že pokud program ve větě narazí právě na předmět, jedná se o podstatu sdělení.

V gramatice jsem toto pravidlo příslušně označil (viz kapitola 4.3.4.3) a upravil závislosti klíčových slov v dalších pravidlech.

4.5 Popis výsledného programu

Výsledný program má poměrně jednoduché ovládání. Jedná se o grafický program, ale některé interní údaje zapisuje do konzole.



Obrázek 4.2: Náhled okna programu s vygenerovaným stromem

Na obrázku 4.2 je zobrazeno hlavní okno programu. Do políčka nahoře lze napsat libovolnou větu a po kliknutí na OK se ve velkém textovém poli dole ukáží vygenerované gramatické stromy. V něm se i vyznačí, pokud se jedná o klíčové slovo (zde je klíčové slovo vyznačeno u slova „trávu“). U každého stromu se pak ukáže jeho prav-

děpodobnost (pravděpodobnost se snižuje překlepy a špatným umístěním čárky ve větě.

4.5.1 Demonstrace schopností programu

Pro lepší pochopení, co program umí, uvádím několik demonstračních vět, které dostal na vstupu. Tyto věty rozebral a vybral z nich klíčová slova. Tato klíčová slova jsou uvedena tučně.

chci navrhnout domluvit krátkou **schůzku** ke včerejšímu problému třeba na zítřek
na zítřek ráno navrhuji **schůzku**
navrhuji na zítřek **schůzku**
vzal jsem **práci** do vlaku
tu **práci** jsem nechal
kolega, který chce lenošit, bere lehkou **práci**
soused dopoledne sekal **trávu**
současný kolega často pěstuje v práci **růže**
vařím k večeři **maso** s rýží
v rádiu teď hrajou fakt pěknou **písničku**
program generuje možné **stromy** gramatik
obhájce u soudu obhajuje **zločince**

Samozřejmě existuje i mnoho vět, které program nezvládne (zpravidla vůbec nevygeneruje žádný strom), a to zejména z důvodu benevolence české gramatiky ke slovosledu, která je špatně implementovatelná, nebo kvůli chybějícímu slovu/tvaru slovu.

Uvedené věty ale program zná a dokáže je zpracovat.

5 Testování

Během vývoje byl a na konci každé větší části samozřejmě program průběžně testován. Během něj se odhalilo nejen možné chyby v zápisu programu, ale i některá opomenutí autorů zdrojů, ze kterých jsem čerpal. Jedná se zejména o neúplný popis daného problému nebo nalezení nějaké specifické výjimky, která se vyskytuje[1] V takových případech jsem hledal v jiných zdrojích po příčině nebo spíše vysvětlení tohoto chování, anebo – zejména když se nejednalo o složitý problém – jsem si poradil sám.

Testování bych obecně rozdělil na 3 části:

5.1 Testování slovotvorby

K testování slovotvorby patří ověřování, zda program ze slova správně vytvoří jiné tvary, např. jiný pád, osobu atd. Za dobu vývoje jsem si udělal několik „udělátek“, zejména různé náhledy, autodetekce atd., které mi v tom pomáhaly. Ruční kontrola všeho je prakticky nemožná.

Ačkoli jsou výjimky zejména u slov s dlouhou historií nebo slov znějících „divně“[2], u některých slov jejich výjimečnost na první pohled znát není (např. 2. pád množného čísla od slova *vejce* nebo *brána*). Nová slova jsou většinou pravidelná, na druhou stranu mají často neustálený tvar (*displej* × *display*) nebo neobvyklou koncovku (např. *idea*, *Billy* atd.). Zde je správné skloňování nejasné často i jazykovědcům (resp. při dotazu na tato slova připouští užívání více tvarů, ale přesto doporučují použít úplně jiný výraz).

K dalšímu testování jsem dále použil např. standardní kontrolu pravopisu v programu *Word* a *OpenOffice Writer* (využívající *ASpell*).

5.2 Testování tvorby gramatiky

Program jsem testoval tak, že jsem si vytvořil soubor s různými případy vět (viz kapitola 4.5.1), které by program měl umět gramaticky zpracovat. Po každé změně jsem pak zkontroloval, jestli všechny stávající věty program dokáže rozebrat správně.

Automatická kontrola pravděpodobně není možná (vzhledem k tomu, že ideální „žádoucí“ strom se bude lišit s každou úpravou gramatických pravidel), při testování jsem si tedy musel vystačit s optickou kontrolou. Z tohoto důvodu se strom zobrazuje ve formátování, kdy je každý následující stupeň stromu reprezentován odsazením – čím větší hloubka, tím větší odsazení.

[1] Např. české slovo *spát* má v záporu tvar *nespat*, tj. krátí se dlouhé *á* na *a*. Totéž slovo *brát* × *nebrat* aj. Navzdory tomu, že existuje podtyp sloves obsahující tuto výjimku, nikde jsem přesně chování zdokumentované nenašel (ani v jinak výborné příručce [9]).

[2] Jsou to většinou slova, jejichž vzor je *neproduktivní*, tj. nová slova se už k tomuto vzoru nepřiřazují. Viz [18]

6 Závěr

Zadání jsem splnil – program, který ze zadané vstupní věty vygeneruje gramatický strom a označí možná klíčové slova v textu. Počítačová lingvistika je (podobně jako jiné obory umělé inteligence) však obor, ve kterém lze jen velice těžko dosáhnout nějakého „ideálního“ stavu. I proto jsou zde nějaká omezení a „nedodělky“.

6.1 Poučení z této bakalářské práce

Tato práce mi hodně dala. V posledním roce jsem si uvědomil několik skutečností, které jsem do té doby poměrně dost nebo zcela podceňoval. Některé z nich v této práci stále chybí.

6.1.1 Modulárnost programu, rozšiřitelnost

Každý program, který není naprogramován jen jako rychlé „udělátko“, ale je zde reálná šance, že by se mohl rozvinout v něco většího, musí být dostatečně rozšiřitelný. Toto hledisko jsem (zejména na začátku) dost podcenil. Ačkoli jsem jednou slyšel, že by se měl psát jen ten kód, který je zrovna potřeba a moc do budoucnosti nekoukat[1], u větších projektů, jaký je tento, je dobrý návrh skutečně potřeba.

6.1.2 Použití cizích řešení ve vlastním projektu

Tento program obsahuje vlastní seznam českých slov a vlastní definici jeho skloňování. Dá se říci, že pouze toto mi zabralo nejméně 60 % času z celé tvorby programu. Přitom existuje otevřený, zdokumentovaný a velice dobře fungující systém ASpell [3]. Proč jsem nepoužil ten a neušetřil si množství času a nervů?

V začátcích jsem totiž měl názory jako „Přece nebudu používat nějaký cizí kód“ a „Bůhví, jak je to napsané, radši si to udělám sám“. Ztratil jsem tak mnoho času jen tím, že jsem neměl odvahu přijmout cizí řešení, které by mi tolik pomohlo. Nevyužíval jsem cizí zdroje. A i teď je využívám jen minimálně, ale zkusím ASpell zaintegrovat.

6.1.3 Rychlost psaní kódu x rychlost vykonávání kódu

V začátcích jsem příliš dal na hledisko „aby to bylo rychlé“. To, že se při tom třeba nadřu já, jsem příliš nevnímal. Proto přechod z C++ na Javu byl pro mne ze začátku nepřijatelný. Nyní jsem ovšem pochopil, že mnohem důležitější je mít možnost program dobře odladit, a k tomu velice rychle najít chyby. To mi Java umožňuje. Bohužel jen samotným přechodem jsem strávil mnoho času, který jsem si mohl ušetřit.

[1] Myslím, že to bylo na přednášce z Řízení softwarových projektů. Ačkoli nám celý semestr tvrdili, že dobrý návrh je základ, potom přišel na přednášky někdo z praxe a tuto teorii částečně vyvrátil.

6.2 Pokračování práce na bakalářské práci

Protože mě téma počítačového zpracování češtiny baví, chtěl bych i nadále v projektu pokračovat. Je to však ještě dlouhá práce. Co je nejvíce potřeba dodělat:

- Odstranit všechny „nedodělky“, které čekají na přepis do Javy
- Zaintegrovat některá stávající řešení (zejména ASpell, později i jiná řešení) a více využívat zdroje o zpracování přirozeného jazyka, abych znovu „nevytvářel kolo“
- Rozšířit možnosti gramatiky a extrakce klíčových slov v textu
- Zavést pojem „význam slova a sdělení“ do programu. Alespoň v takové míře, v jaké to jen bude možné
- Dát programu takovou hodnotu (zaměření), aby byl užitečný co nejvíce lidem

6.3 Poslání (nejen) bakalářské práce

Tato bakalářská práce je součástí dlouhodobé práce. Pokud pomohla mé snaze o vytvoření dokonalý počítačový rozbor českého jazyka – a já věřím, že pomohla – splnila svůj úkol.

Jsem rád, že jsem mohl tuto práci zlepšovat i v rámci školních a jiných povinností.

7 Literatura

[1] Alicebot.org – program na hovor s počítačem angličtině, zdroj informací. http://www.alicebot.org/.

[2] *Boumův mýtus* – jak člověk čte psaný text. http://www.magtypo.cz/buxus/generate_page.php?page_id=300.

[3] Technologie *ASpell* pro kontrolu překlepů slov včetně českých. http://aspell.net/.

[4] N. Chomsky. Oficiální stránky tohoto amerického jazykovědce. http://www.chomsky.info/.

[5] Doc. Ing. K. Müller, CSc. *Programovací jazyky*. České vysoké učení technické, 2005.

[6] F. informatiky Masarykovy univerzity. Laboratoř zpracování přirozeného jazyka. http://nlp.fi.muni.cz/.

[7] A. Janda. Program IQ Pokyd pracující s českou gramatikou, 2003. http://iqpokyd.kyblsoft.cz/, *vlastní zdroj*.

[8] V. S. Jiří Melichar. *Český jazyk*. Státní pedagogické nakladatelství, 1990.

[9] kolektiv Ústavu českého jazyka filosofické fakulty Masarykovy univerzity v Brně. *Příruční mluvnice češtiny*. Nakladatelství Lidové noviny, 2001.

[10] Lingea. Aplikace *Grammaticon* pro kontrolu českého pravopisu. http://www.lingea.cz/cz/nastroje.asp?sp=gr.

[11] Lingea. Příručka aplikace *Grammaticon* – omezení kontroly českého pravopisu. http://www.lingea.cz/download/prirucka.pdf.

[12] K. Oliva. *Úvod do formalismu*. Fakulta filosofie Masarykovy. http://utkl.ff.cuni.cz/~rosen/public/KO_formalismus_skripta.pdf .

[13] K. Pala. *Počítačové zpracování přirozeného jazyka*. Fakulta informatiky Masarykovy univerzity, 2000.

[14] J. Peregrin. *Význam a struktura*. OIKOYMENH, 1999.

[15] I. Research. Natural language processing. http://domino.research.ibm.com/comm/research.nsf/

[16] M. Research. Natural language processing. http://research.microsoft.com/nlp/.

[17] J. Weizenbaum. ELIZA – první program pro hovor s počítačem. http://i5.nyu.edu/~mm64/x52.9265/january1966.html/.

[18] J. Černý. *Úvod do studia jazyka*. Rubico, 1998.

A Přílohy

A.1 Obsah přiloženého CD

Na přiloženém CD je kromě tohoto dokumentu i vlastní program.
Popis jednotlivých adresářů a některých souborů:

- **/** *(kořenový adresář)*
  - **install.bat** – spouštěcí dávka programu (použitelná ve Windows i v Linuxu)
  - **install.txt** – technické informace o programu a jeho spuštění
  - **readme.txt** – popis jednotlivých adresářů
- **/exe** – vlastní program ve spustitelné podobě
  - **gramatika.txt** – textový soubor s definicí gramatiky
  - **program.jar** – vlastní program v Javě
  - **sklonovani.txt** – textový soubor s definicí způsobu skloňování
  - **slovnik.bin** – binární soubor se seznamem všech rozskloňovaných slov. Tento soubor program vytvoří sám ze souboru *zakladnislovnik.txt*
  - **zakladnislovnik.txt** – textový soubor se seznamem českých slov
- **/html** – popis ovládání programů ve formátu HTML
  - **index.html**
  - **screenshot-program1.png ...** – screenshoty k popisu
- **/src** – zdrojový kód programu v jazyce Java
  - **HlavniSpusteni.java** – vstupní metoda (main class)
  - **Rozhrani.java**
  - **/Gramatika**
    - **....java**
  - **/Obecne**
    - **....java**
  - **/Rozpoznani**
    - **....java**
  - **/TvorbaSlov**
    - **/cs**
      - **....java**
    - **....java**
- **/text**
  - **BP.pdf** – tato bakalářská práce ve formátu PDF
  - **/src** – zdrojový kód textu k bakalářské práce v LaTeXu
    - **bak.tex** – vlastní zdrojový kód (kódování utf-8)
    - **LogoK336.eps ...** – obrázky vkládané do textu
    - **strom.dot** – zdrojový kód grafu v GraphViz