MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Syntaktická analýza s využitím postupné segmentace věty

DIPLOMOVÁ PRÁCE

Vojtěch Kovář

Brno, podzim 2008

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: RNDr. Aleš Horák, Ph.D.

Poděkování

Děkuji Aleši Horákovi za odborné vedení práce, vstřícný přístup a cenné konzultace. Rovněž děkuji své rodině a přítelkyni za nenahraditelnou podporu při psaní práce.

Shrnutí

Tato práce se zabývá návrhem nové metody pro syntaktickou analýzu češtiny, založeném na postupné segmentaci věty. Popisujeme současné hlavní přístupy k řešenému úkolu a jejich nedostatky, návrh a implementaci nového systému pro syntaktickou analýzu češtiny a dosažené výsledky měření přesnosti na korpusových datech.

Klíčová slova

automatická syntaktická analýza, syntax, analyzátor, parser, analýza češtiny, segmentace věty, set

Obsah

Kapitola 1

Úvod

Automatická analýza přirozeného jazyka je odvětvím, které s rozmachem informační společnosti nabývá stále většího významu. Vzniká potřeba automaticky analyzovat velká množství dokumentů v přirozeném jazyce, dostupných volně na Internetu, v nejrůznějších databázích novinových či odborných článků, jazykových korpusech apod.

Tyto dokumenty je třeba dále třídit, získávat z nich informace a inteligentními metodami v nich vyhledávat tak, aby čtenář hledající informaci strávil minimum času zbytečným čtením textů, které jsou pro něj neužitečné. Ve vzdálenější budoucnosti mohou vznikat stroje, které se na základě textů v přirozeném jazyce budou učit fakta, budovat a rozšiřovat svou znalostní bázi a sloužit jako specializované dialogové systémy, případně dokonce inferenční stroje a generátory nových teorií.

V cestě za takto vyspělými inteligentními systémy jsme však zatím na samém počátku. Většina ze současných „inteligentních" přístupů využívá pouze jednoduché statistické informace získané ze slov, případně z morfologické analýzy slov obsažených ve vstupním textu. Důvodem tohoto stavu je nedostatečná kvalita nástrojů pro analýzu jazyka na vyšších úrovních – především syntaktické a sémantické. Kvalita takovýchto nástrojů je klíčová, chceme-li analyzovat texty skutečně do hloubky, tj. dosáhnout stavu, kdy příslušný stroj (například vyhledávací) obsahu zpracovávaných dokumentů de facto rozumí.

Tato práce se zabývá problémem automatické syntaktické analýzy vět v přirozeném jazyce, konkrétně v češtině, jako jedním z kroků komplexní automatické analýzy jazyka. V úvodních částech jsou na základě dostupných pramenů shrnuty základní přístupy používané k syntaktické analýze češtiny a problémy, s nimiž se tyto přístupy potýkají. Další části jsou věnovány hlavnímu přínosu práce, jímž je metoda postupné segmentace věty. Zmiňujeme myšlenky, které k navrhovanému konceptu vedly, vysvětlujeme principy metody a představujeme systém pro automatickou syntaktickou analýzu češtiny SET, který je na metodě postupné segmentace věty založen.

Kapitola 2

Syntaktická analýza přirozených jazyků

Úkolem syntaktické analýzy přirozených jazyků (dále jen analýzy) je odhalit povrchovou strukturu věty, tj. vztahy mezi slovy i většími jednotkami (konstituenty) a způsob, jakým se tyto jednotky skládají do větného celku. Takto získaná informace je klíčová v následujícím procesu sémantické (či logické) analýzy dané věty, případně pro extrakci dílčích informací z textu.

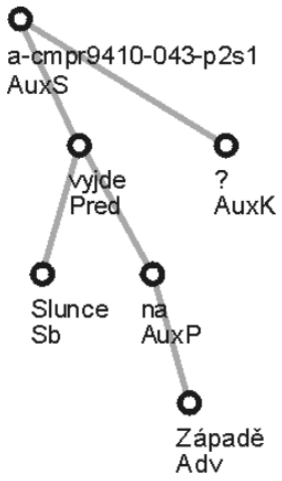
V současnosti existují dva základní přístupy k syntaktické analýze českých vět. Prvním z nich je přístup závislostní, jenž je rozvíjen v pražském Ústavu formální a aplikované lingvistiky (ÚFAL). Na jeho základech je mimo jiné postaven Pražský závislostní korpus (Prague Dependency Treebank, nebo též PDT [3], [4]) a množství analyzátorů (např. [5], [9], [10]). Druhým je přístup složkový, na jehož základě pracují analyzátory vyvíjené v Centru zpracování přirozeného jazyka na Fakultě informatiky Masarykovy univerzity, jejichž nejvýznamnějším reprezentantem je analyzátor `synt` [12].

V této kapitole stručně popíšeme charakteristické vlastnosti obou zmíněných přístupů a nastíníme problémy, které prozatím znemožňují výraznější nasazení vyvíjených analyzátorů na vyšších vrstvách analýzy jazyka, jako je logická analýza či extrakce informací.

2.1 Závislostní přístup

Práce probíhající na ÚFAL vychází z tradice pražské lingvistické školy, která zahrnuje poměrně komplexní analýzu jazyka na rovině morfologické, syntaktické a částečně i sémantické. Na základě těchto tradičních teorií, upravených pro potřeby současného výzkumu v oblasti počítačové lingvistiky, jsou anotována rozsáhlá korpusová data a vyvíjeny analyzátory využívající různých přístupů (viz dále).

V závislostním formalismu je za syntaktickou analýzu věty považován kořenový strom věty (acyklický orientovaný graf s vyznačeným kořeno-



Obrázek 2.1: Příklad závislostního stromu pro větu *Slunce vyjde na západě?*

vým vrcholem), jehož vrcholy tvoří právě slova vstupní věty,[1] spolu s jedním přidaným pomocným vrcholem, který je vždy kořenem stromu. S výjimkou tohoto pomocného vrcholu nejsou do věty přidávány žádné další strukturní informace, vše je kódováno do vzájemných vztahů slov ve vstupní větě. Vztahy mezi slovy jsou zachyceny hranami v grafu věty, přičemž každá hrana vyjadřuje závislost jednoho slova na jiném. Každá hrana je dále ohodnocena syntaktickou funkcí,[2] jež určuje typ závislosti dané hrany (např. funkce `Attr` vyjadřuje závislost přívlastku na řídícím slově). Platí, že z každého vrcholu s výjimkou kořene vede právě jedna závislostní hrana, tj. každé slovo je závislé na právě jednom dalším slově (nebo kořenovém uzlu).

Příklad závislostního stromu můžeme vidět na obrázku 2.1.[3]

2.1.1 Pražský závislostní korpus

Již zmíněný Pražský závislostní korpus (PDT [3]), vytvářený na ÚFAL, je korpus češtiny, manuálně anotovaný na více úrovních analýzy jazyka podle

1. Za slova zde považujeme všechny tzv.*tokeny* tj. slova, čísla a interpunkci.
2. Technicky jsou touto syntaktickou funkcí ohodnoceny uzly stromu, představu ohodnocených hran však považujeme za názornější.
3. Převzato z vizualizovaných příkladů pro PDT 2.0,
`http://ufal.mff.cuni.cz/pdt2.0/visual-data/sample/sample3_a_26.htm`

principů pražské lingvistické školy. Jeho celkový rozsah je téměř dva miliony slovních jednotek.

V této práci je pro nás podstatná syntakticky označkovaná část korpusu, tzv. analytická rovina anotace, která v nynější verzi PDT 2.0 obsahuje 87 913 vět s celkovým počtem 1 503 739 slovních jednotek. Tato část korpusu je tvořena syntaktickými stromy českých vět manuálně vytvořenými podle závislostního formalismu představeného výše. Je to jediný manuálně syntakticky označkovaný korpus češtiny srovnatelné velikosti (jediný zdroj „správných" syntaktických dat) a je tedy velmi důležitý z hlediska testování kvality vyvíjených automatických analyzátorů.

2.1.2 Závislostní analyzátory

Spolu s PDT je na stejném pracovišti vyvíjeno množství automatických závislostních analyzátorů, které korpus používají jako trénovací a testovací data.

Tyto analyzátory vesměs pracují ve dvou fázích. V první, trénovací fázi se analyzátor z označkovaných dat naučí pravidla (v různých formách podle konkrétního analyzátoru), která následně používá ve druhé fázi, kdy podle získaných pravidel analyzuje vstupní větu. Na tomto principu pracují například analyzátory popsané v [5], [8] a také *McDonnald's maximum spanning tree parser* [22], jenž dosahuje nejlepších výsledků v měření přesnosti analýzy (s výjimkou kombinace analyzátorů, viz dále).

Ne všechny analyzátory, se kterými se pracuje na ÚFAL, jsou ale tohoto charakteru. Uvedeme zde dva příklady odlišného přístupu k závislostní analýze. Prvním z nich je Holanův parser ANALOG [9], který nemá trénovací fázi a ve fázi analýzy vyhledává takovou lokální konfiguraci, která je nejvíce podobná datům v trénovací množině.

Druhým „atypickým" analyzátorem je Žabokrtského pravidlový závislostní analyzátor, popsaný v [10], který analyzuje vstupní větu s pomocí ručně vytvořených transformačních pravidel, nevyužívá tedy žádnou informaci z trénovacích dat. Pravidla jsou implementována přímo jako funkce v programovacím jazyce *Perl*, není využito žádného známého gramatického formalismu.

Všechny uvedené analyzátory předpokládají na vstupu morfologicky jednoznačně označkovanou (desambiguovanou) větu, před jejich použitím je tedy třeba aplikovat morfologický analyzátor a desambiguátor (tagger).

Seznam dostupných taggerů je zveřejněn na stránkách ÚFAL[4] i s příslušným komentářem a odkazy.

2.1.3 Měření úspěšnosti závislostní analýzy

Úspěšnost závislostní analýzy věty je dána počtem správně určených závislostí, hran ve stromu věty (za správně určené považujeme samozřejmě ty hrany, které se shodují s anotací v korpusu). Pro každou větu určujeme přesnost (precision) a pokrytí (recall) hran určených analyzátorem.

Vzhledem k tomu, že počet hran je pro danou větu vždy stejný (závislostní strom obsahuje právě tolik hran, kolik je slov ve větě), přesnost i pokrytí budou pro danou větu vždy nabývat stejné hodnoty. Úspěšnost analyzátoru na jedné větě můžeme tedy vyjádřit jediným číslem, v dalším textu nazývaným pouze *přesnost*. Úspěšnost analyzátoru na testovací sadě je pak vyjádřena průměrem (v některých případech případně mediánem) přesností analýzy pro jednotlivé věty.

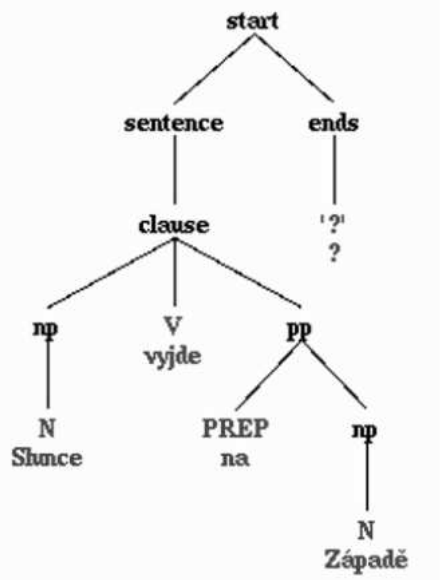
Přesnost analyzátorů uvedených v předchozí části se pohybuje od 53 do 84 procent [10]. Nejúspěšnějším samostatným analyzátorem je již zmíněný *McDonnald's maximum spanning tree parser*, dosahující přesnosti 83.98 %. Zajímavý je experiment s kombinací několika různých analyzátorů (též [10]), v němž se podařilo zvýšit dosahovanou přesnost o další téměř dvě procenta na 85.84 %. Přesnost Žabokrtského pravidlového analyzátoru je 75.93 %.[5]

2.2 Složkový přístup

Druhým z hlavním proudů v automatické syntaktické analýze je přístup složkový, rozvíjený v Centru zpracování přirozeného jazyka na Fakultě informatiky Masarykovy univerzity (CZPJ FI MU). Narozdíl od závislostního, složkový formalismus operuje při vyznačování syntaktických vztahů i s většími celky, než jsou slova. V procesu analýzy jsou rozpoznávány *konstituenty* ve vstupní větě a je odhalován způsob, jakým se tyto konstituenty skládají do větného celku.

Složkové stromy, výstup ze složkové analýzy, pak tento proces názorně reflektují – je explicitně vyznačeno, které části věty formují jmenné skupiny, které jmenné skupiny se pojí se slovesem apod. Jak již je z předchozího patrné, složkové stromy používají neterminální symboly pro označení rozpo-

4. `http://ufal.mff.cuni.cz/czech-tagging/`
5. Všechna uváděná čísla se vztahují k měření na oddílu PDT, který je k určen ke „slepému" testování úspěšnosti vyvíjených analyzátorů – *e-test*.



Obrázek 2.2: Příklad složkového stromu pro větu *Slunce vyjde na západě?*

znaných struktur, jsou tedy bohatší a složitější než stromy závislostní. Jejich podoba odpovídá odvození podle formální gramatiky bezkontextového typu a koresponduje s teoriemi syntaxe podle Noama Chomského.

Příklad složkového stromu můžeme vidět na obrázku 2.2.

2.2.1 Analyzátor Synt

Na základě složkového přístupu je v CZPJ FI MU vyvíjen analyzátor `synt` [12]. Tento analyzátor je založen na bezkontextové gramatice obohacené o kontextové akce; tato gramatika je přitom nejednoznačná, takže na výstup jsou předávány množiny možných stromů pro danou větu, nikoli jeden strom. Nicméně, stromy v této množině jsou ohodnoceny a setříděny a algoritmy implementované v analyzátoru dovolují vybrat N nejlepších stromů v polynomiálním čase, i když celkový počet stromů může být až exponenciální.

Vzhledem k tomu, že gramatika češtiny je velmi rozsáhlá (zejména kvůli relativně volnému pořádku slov), je vyvíjena ve formě tzv. metagramatiky [14], jež umožňuje zhustit informaci obsaženou v gramatice do řádově 200 pravidel, která jsou poté automaticky rozgenerována do gramatického formalismu používaného analyzátorem. Rozgenerovaná forma gramatiky již obsahuje tisíce pravidel.

Pro samotný proces analýzy je pak používán algoritmus *head-driven chart parsing* [17], který je velmi efektivní i pro rozsáhlé gramatiky. Po provedení analýzy je výstupní množina stromů dále prořezávána a tříděna s využitím různých metod ([15], [18], [19]).

Narozdíl od většiny závislostních analyzátorů zmíněných výše, analyzátor `synt` dokáže zpracovat i text s víceznačným morfologickým označkováním, bez výrazného nárůstu časové náročnosti analýzy.

V návaznosti na výsledky programu `synt` jsou též rozpracovány metody analýzy jazyka na vyšších vrstvách, zejména logická analýza v transparentní intenzionální logice [11]. Popis těchto metod by však byl již nad rámec této práce.

2.2.2 Měření úspěšnosti složkové analýzy

Pro měření úspěšnosti složkové analýzy je používáno podobnostních metrik na složkových stromech. V současnosti je pravděpodobně nejrozšířenější z nich metrika PARSEVAL a její varianty, v poslední době je však poukazováno na její nedostatečnost [7] a jsou navrhovány metriky nové. Velmi nadějným návrhem se jeví technika *Leaf Ancestor Assessment* (LAA), navržená v roce 2000 [24].

Problémem v případě měření úspěšnosti složkové analýzy češtiny je neexistence rozsáhlého korpusu složkových stromů pro češtinu. Protože existují techniky převodu závislostních stromů na složkové a naopak [2], bylo by možné využít PDT, jsou však problémy s převodem neprojektivních konstrukcí, které nelze zachytit ve složkovém formalismu. (O neprojektivní konstrukci v závislostním stromě mluvíme tehdy, když množina závislostí jednoho uzlu netvoří souvislý úsek v rámci věty, viz např. [27].)

Problém s převodem závislostních a složkových reprezentací syntaxe je také hlavní příčinou toho, že se dosud nepodařilo uspokojivě srovnat kvalitu pražských závislosních analyzátorů s analyzátorem `synt`. Výsledky zatím jediného pokusu o srovnání je možno nalézt v [13]. Na základě výsledků zde uvedených lze soudit, že přesnost analyzátoru `synt` (měřeno metrikou LAA) se pohybuje mezi 70 a 90 procenty. Tento údaj je však velmi neurčitý.

2.3 Parciální syntaktická analýza

V předcházejících částech jsme se zabývali syntaktickou analýzou úplnou, jejímž cílem je získat úplný syntaktický strom vstupní věty. Cílem parciální syntaktické analýzy (též *shallow parsing*) je získat z věty pouze některé syntaktické informace, např. hranice jmenných frází a jiných významných větných skupin, nikoli kompletní strom věty.

Základní techniky parciální syntaktické analýzy jsou shrnuty v [1], parciální syntaktickou analýzou češtiny a jejím využitím se zabývá práce [26]. Za jednu z technik parciální analýzy jazyka můžeme označit i formalismus pro tzv. *word sketches*, určený zejména pro vyhledávání častých kolokací v jazykových korpusech [23].

V této práci se zabýváme především úplnou syntaktickou analýzou, nebudeme tedy zabíhat do větších detailů. Metoda analýzy, kterou v dalších kapitolách navrhujeme, má však s technikami parciální analýzy některé rysy společné, jak uvidíme dále. V návrhu využíváme některých dílčích (parciálních) syntaktických informací v několika vrstvách k tomu, abychom získali úplnou analýzu vstupní věty. Navržený systém se dokonce dá částečně pro parciální syntaktickou analýzu použít, i když to nebylo primárním cílem.

2.4 Problémy v syntaktické analýze

Kromě již zmíněných problémů s měřením přesnosti a jejím srovnáváním existuje v oblasti syntaktické analýzy řada dalších. Některé z nich jsou omezeny na jednotlivé použité formalismy, jiné lze chápat jako komplexní problémy analýzy přirozených jazyků. V této sekci některé z takových problémů zmíníme.

2.4.1 Nízká přesnost analýzy

Nízká úspěšnost je klíčovým problémem v oblasti syntaktické analýzy přirozených jazyků. Nejlepší dosahované výsledky se pro češtinu pohybují okolo 85 procent, což při průměrné délce věty 17 slov[6] znamená zhruba 2,5 chyby na každou větu.

Takováto chybovost je neúnosná téměř pro všechny další potenciální aplikace výstupů analýzy. Z tohoto důvodu jsou také vyvíjeny stále nové přístupy k syntaktické analýze a z tohoto důvodu mimo jiné vznikla i tato práce.

6. Měřeno na PDT.

2.4.2 Víceznačnost analýzy

Víceznačnost (ambiguita) je problémem na všech úrovních analýzy přirozeného jazyka. Na úrovni syntaxe je často dokonce nemožné rozhodnout, které ze dvou možných čtení je správné.

Jako příklad nám může posloužit věta: *„Karel pronásledoval muže na kole."* Z takto samostatně uvedené věty nelze určit, zda se fráze *na kole* pojí se jménem *muž* či s dějem pronásledování. V tomto případě by člověku patrně pomohl širší kontext textu, ten ale syntaktické analyzátory zpravidla nevidí.

V případě brněnského analyzátoru `synt` způsobuje problém víceznačnosti extrémní množství stromů na výstupu (až miliardy) pro některé věty. Důvodem není jen vnitřní víceznačnost syntaxe, ale i fakt, že princip analýzy implementovaný v systému zohledňuje pouze morfologickou informaci vstupních slov – tedy např. fráze *„dívka od rána zpívala"* je pro něj nerozeznatelná od fráze *„dívka z vesnice zpívala"* a určuje tedy vždy všechny možnosti, což ve výsledku může způsobit exponenciální nárůst počtu stromů.

Pražské závislostní analyzátory víceznačnost prakticky neuvažují, jejich výstupem je vždy jediná analýza. Pro všechny současné myslitelné aplikace to asi dostačuje, nicméně v budoucnu se patrně bude muset vyřešit problém zpětné aplikace informací z vyšších vrstev analýzy jazyka na analýzu syntaktickou, neboť syntaktické čtení může být významně ovlivněno sémantikou výpovědi, sémantikou kontextu, situací promluvy a podobně. Vůbec zde přitom neuvažujeme takové případy jako je víceznačnost zamýšlená, objevující se například v anekdotách či v poezii, neboť analýza takovýchto jevů je záležitostí spíše vzdálenější budoucnosti.

2.4.3 Subjektivita syntaxe

Tento problém se dá s trochou nadsázky formulovat jako „co člověk, to názor". Tento princip platí i v syntaxi – i v rámci jednoho projektu či jedné pracovní skupiny se lidé často neshodnou, jak má vypadat správná analýza některých syntaktických jevů. Například pro větu *„Faxu škodí především přetížené telefonní linky"*[7] není zcela jasné, zda se slovo *především* pojí spíše ke slovu *linky*, ke slovesu *škodit*, či dokonce k adjektivu *přetížené*. Větší skupina lidí se na správném rozhodnutí jednoduše neshodne.[8]

7. Uvedený příklad je reálnou větou z PDT.
8. Podkladem pro toto tvrzení je autorovi diskuse, která proběhla v listopadu 2008 mezi členy CZPJ FI MU a drobný průzkum této otázky, který si následně provedl ve svém okolí.

Výše uvedené bohužel platí i o pracovní skupině anotátorů PDT. Částečným řešením v tomto konkrétním případě bylo sepsání rozsáhlého manuálu pro anotátory [6], který určuje, jak mají být řešeny některé sporné a nejednoznačné situace při anotaci. Manuál bohužel ponechává anotátorům značnou volnost v rozhodování, navíc ani zdaleka nepokrývá všechny sporné případy (což je také v principu zřejmě nemožné), například pro slovo *především* z věty v předchozím odstavci v něm žádné řešení nenajdeme. Ve výsledku se tedy v manuálně anotovaných korpusových datech mohou vyskytnout (a vyskytují) poměrně výrazné nekonzistence.

Vyvstává tedy otázka, zda je námi zvolený (a všeobecně používaný) přístup k syntaxi správný. Přestože se totiž lidé neshodnou na tom, co dělat se slovem *především* z diskutovaného příkladu, evidentně jsou schopni pro účely běžné komunikace větu správně pochopit a porozumět jí (podobně jako je tomu u mnoha dalších běžných jevů v jazyce, které se obtížně vyjadřují v tradičních formalismech pro syntaxi a které zde pro nedostatek prostoru neuvádíme).

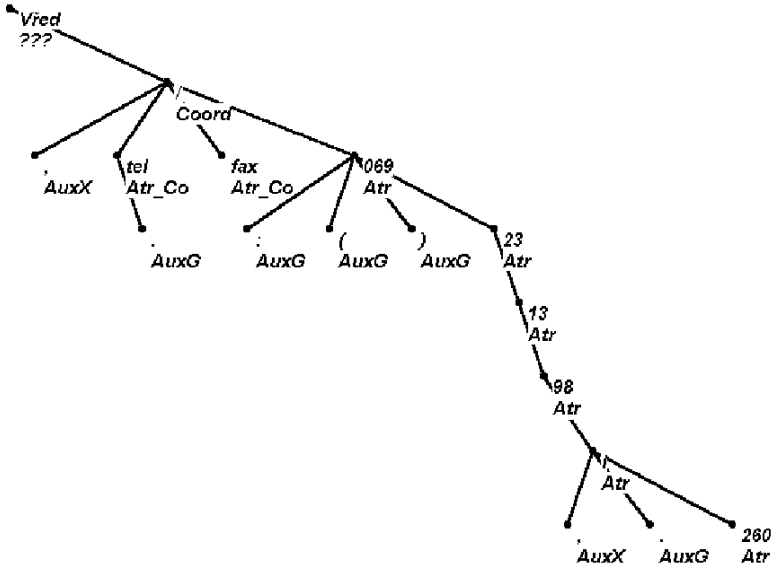
Na tomto místě se spokojíme s prohlášením, že žádné lepší pojetí syntaxe zatím bohužel neexistuje a návrh nového je úkolem, který značně přesahuje možnosti a rámec této práce. Nicméně, téma této sekce je jistě hodno dalšího zpracování. V dalším textu práce se k tomuto tématu částečně vrátíme v části 5.2.1.

2.4.4 Určování sporných a nadbytečných jevů

Tento poslední zmiňovaný nedostatek současného přístupu k syntaktické analýze je značně příbuzný předchozím dvěma. Jádrem problému je fakt, že zvolený syntaktický formalismus nás nutí nějakým způsobem rozhodovat o struktuře jevů, u nichž je sporné či irelevantní, která z nabízejících se možností je správná.

Příkladem může být slovo *především* z předchozí podkapitoly; tento problém jsme již zmínili a jeho řešení označili v rámci této práce za nedosažitelné. Existují však i jiné typy struktur, u kterých se problém určování nadbytečných jevů objevuje výrazněji a jejichž řešení mohou být přímočařejší – jedno z nich navrhujeme v sekci 3.2.4. Následují příklady z jednotlivých formalismů.

Za příklad nadbytečné informace v případě závislostního formalismu můžeme bez rozpaků označit téměř všechny problémy rozebírané v oddílu 3.7.2. Návodu pro anotátory [6]. Jedná se zde o zachycení struktury textu adres, názvů firem včetně telefonních čísel apod. Požadavek formalismu, podle něhož má být každé slovo (včetně interpunkce) zavěšeno právě na



Obrázek 2.3: Instrukce pro anotaci věty *„Vřed, tel. / fax : (069) 23 13 98, l. 260."* v PDT. Převzato z Návodu pro anotátory.

jednom dalším slově, zde vytváří komplexní struktury, které jsou však v drtivé většině případů zbytečné a pro člověka naprosto neintuitivní. Např. v případě věty *„Vřed, tel. / fax : (069) 23 13 98, l. 260."* – viz obrázek 2.3 – nám závislostní analýza zřejmě nedává téměř žádnou smysluplnou informaci. Je otázkou, zda by takovéto „věty" vůbec měly být zařazovány do jazykového korpusu typu PDT.

Příkladem nadbytečné informace ve složkovém formalismu je struktura složitějších jmenných frází. V případě fráze *„nízká rychlost přenosu"* nás formalismus (alespoň v jeho současné podobě v analyzátoru `synt`) nutí zvolit mezi dvěma možnými uzávorkováními – *(nízká rychlost) přenosu* vs. *nízká (rychlost přenosu)*. Je zřejmé, že obě uzávorkování jsou ekvivalentní, člověk není schopen rozhodnout, které z nich je lepší.

Takovýchto příkladů je v obou představených formalismech více. Jejich důsledkem jsou mj. nekonzistence v anotovaných datech, a následně snížená objektivita hodnocení výsledků analyzátorů. Jiinak řečeno, v těchto případech nejsou lidé schopni správné řešení určit, přesto ale chceme po analyzátorech, aby volily správně.

Kapitola 3

Metoda postupné segmentace věty

V této kapitole popíšeme základní principy nového přístupu k syntaktické analýze, založeného na postupné segmentaci věty. Nejprve uvedeme několik pozorování o analýze jazyků a vyslovíme neformální závěry, ke kterým nás tato pozorování přivedla. Následně popíšeme navrhovanou metodu více podrobně a uvedeme její vztahy k ostatním formalismům. Konkrétní aplikace metody, analyzátor SET, bude náplní kapitoly následující.

3.1 Úvodní pozorování o syntaxi

V této sekci uvedeme několik pozorování o syntaxi nejen přirozených jazyků, která nám poslouží jako evidence pro některé závěry, které následně představíme. V dalších sekcích pak rozvedeme návrh metody analýzy, k níž nás tyto závěry přivedly.

3.1.1 Pozorování první: Obtíže v návrhu gramatiky

Jak již částečně vyplynulo z údajů uvedených v části 2.4, formulace pravidel v tradičních formalismech je velmi náročným úkolem. Složitost jevů v přirozeném jazyce nám neumožňuje obsáhnout plný rozsah jazyka relativně jednoduchou gramatikou bez vedlejších efektů, které se následně projevují nízkou přesností či vysokou víceznačností výstupu analýzy. Velkou komplikací v případě češtiny je též relativně volný pořádek slov ve větě.

Pro analyzátory založené na statistickém učení z korpusových dat můžeme učinit podobný závěr; tyto nástroje v trénovací fázi (zjednodušeně řečeno) vyvíjejí svou sadu pravidel, která pak aplikují ve fázi analýzy. Z výsledné nízké přesnosti lze soudit, že trénovací data nebo formalismus použitý pro naučená pravidla (případně obojí) jsou nedostatečné.

Pro ilustraci uvažme následující příklad návrhu jednoduché bezkontextové gramatiky pro analýzu českých jmenných frází (příklad je vykonstruován a značně zjednodušen, přesto však reflektuje reálný problém – v tomto případě problém analyzátoru `synt`):

- $NP \rightarrow N$ (např. „pes")
- $NP \rightarrow ADJ$ (např. „červená (je pěkná barva)")
- $NP \rightarrow ADJ \quad NP$ (např. „velký pes")
- $NP \rightarrow NP \quad NP$ (např. „královna krásy")

Tato gramatika dává pro analýzu elementární fráze *„velký černý pes"* v principu 5 možných analýz. Pokud namísto fráze o třech slovech budeme uvažovat dvacetislovnou větu, exponenciální počet stromů na výstupu nebude již žádným překvapením. Gramatiku by samozřejmě bylo možno optimalizovat, ovšem za cenu nárůstu její velikosti a složitosti, který by komplikoval její další vývoj.

3.1.2 Pozorování druhé: Negramatické konstrukce

Jak v případě formálně definovaných (např. programovacích) jazyků, tak v případě jazyků přirozených je známou pravdou, že lidé bez problémů dokáží analyzovat informaci obsaženou v negramatických kostrukcích, tj. takových, ve kterých jsou formální chyby. Bez této dovednosti by například programátoři nemohli opravovat chyby ve vytvořeném kódu a stejně tak práce jazykových korektorů by byla zcela nemožnou, neboť jedni ani druzí by nebyli schopni pochopit zamýšlený význam opravovaného textu.

Samozřejmě, tato schopnost je omezena jen na jisté druhy gramatických chyb – jak ve formálních, tak v přirozených jazycích existují chyby, které podstatně změní význam textu a znemožní čtenáři textu v původním významu porozumět. Dovolujeme si však tvrdit, že takovýchto chyb je v přirozených jazycích menšina.

Pro ilustraci uvádíme příklad dvou negramatických konstrukcí, kterým lidé ovládající daný jazyk bez problémů porozumí (a případně dokáží přítomné chyby opravit). První příklad se týká jazyka *C* jako reprezentanta formálních jazyků, druhá konstrukce je v češtině:

- if i % 3 == 0 printf(i);
- Když to neuděláš zbiju tě.

3.1.3 Pozorování třetí: Klíčová slova ve formálních jazycích

V tomto pozorování si vezmeme opět jazyk *C* jako reprezentanta formálních jazyků. Uvažujme následující jednoduchý program:

```
int i = 20;
while (i > 0) {
  if (i % 3 == 0) printf("%d", i);
  i--;
}
```

Pokusme se nyní o interpretaci procesu, jakým člověk, který do jisté míry ovládá jazyk *C*, čte daný program.

Na prvním řádku podle klíčového slova *int* pozná, že se zavádí nová proměnná. Podle dalších informací na tomtéž řádku usoudí, že proměnná se jmenuje *i* a její iniciální hodnota je 20. V dalším řádku s pomocí klíčového slova *while* rozezná, že na tomto místě začíná cyklus; podle příslušných složených závorek pak určí jeho rozsah. Podle podmínky v kulatých závorkách zase může udělat závěr o tom, kdy tento cyklus skončí. Takto bychom mohli dále pokračovat až do úplného porozumění programu.

Nyní si položme otázku, jakým způsobem provádí analýzu stejného programu kompilátor (počítač). Bezpochyby implementuje nějakou formu LR či LALR analyzátoru, což znamená, že načítá jedno vstupní slovo (token) po druhém a postupně nad nimi staví syntaktický strom podle gramatiky pro jazyk *C*.

O přesné podobě procesu lidského chápání programu by se jistě dalo dlouze diskutovat, nicméně můžeme s velkou mírou jistoty říci, že je značně odlišná od procesu, jakým programu rozumí kompilátor. Kdyby tomu tak nebylo, programátoři by nedělali chyby.

Dále můžeme říci, že lidské čtení programu je daleko více závislé na klíčových slovech jazyka. Výrazy *int*, *while*, *if* a *printf* spolu s příslušnými závorkami vyznačujícími rozsah přispívají k celkovému pochopení struktury programu, zatímco například podmínky v kulatých závorkách jsou relativně druhořadé a pro rámcové pochopení výpočetního toku programu nejsou zapotřebí. Pro zajímavost srovnejme informační hodnotu programu s vypuštěnými klíčovými slovy s hodnotou programu, obsahující pouze klíčová slova:

```
xxx i = 20;
xxx (i > 0)
  xxx (i % 3 == 0) xxx("%d", i);
  i--;

================================

int xxx;
while xxx {
  if xxx printf xxx;
  xxx;
}
```

Pokud můžeme mluvit o míře pochopení u těchto „programů", je tato míra u druhého z nich jistě vyšší, přestože tento obsahuje daleko menší zlomek textu původního programu (počítáno v tokenech).

Pro úplné pochopení samozřejmě potřebujeme informace úplné, nicméně výše uvedeným příkladem jsme ukázali, že v procesu lidského porozumění programu mohou mít různé tokeny různou váhu. Zejména je třeba všímat si nejprve klíčových slov programu a teprve potom zbylých konstrukcí. Z tohoto důvodu je také ve vývojových prostředích pro většinu programovacích jazyků hojně využíváno zvýraznění syntaxe, které významně pomáhá programátorovi při čtení a analýze zdrojového kódu programu.

3.1.4 Pozorování čtvrté: Klíčová slova v přirozených jazycích

Název této kapitoly může být poněkud zavádějící – výrazem *klíčová slova* se v souvislosti s přirozeným jazykem obvykle míní souhrn několika slov určujících téma textu. Zde budeme tento výraz používat pro slova *syntakticky významná*, tj. ve stejném významu jako v předchozí podkapitole, týkající se programovacích jazyků.

Klíčová slova v přirozených jazycích fungují ve skutečnosti obdobně jako klíčová slova ve formálních jazycích – aniž bychom viděli celou větu, dokážeme její rámcovou strukturu určit pomocí několika málo typů slov. Příklad věty:

Až přijdete domů, udělejte si malou módní přehlídku.

A její přepis pouze s „klíčovými slovy" ve dvou možných podobách (podle toho, která slova ještě považujeme za klíčová):

Až xxx , xxx .
Až přijdete xxx , udělejte xxx .

Poměrně zřetelně vidíme, že značnou část struktury věty lze odhalit pouze na základě některých význačných (klíčových) slov ve větě (to, jestli slovesa považujeme za klíčová slova, je v tuto chvíli nepodstatné). Přitom identifikace těchto klíčových slov je často v mnohých ohledech jednoznačná (což je v analýze jazyka velmi příjemná výjimka).

Detekcí a použitím některých klíčových slov v našem pojetí se zabývali autoři v [21]. Zde byla klíčová slova používána k segmentaci komplexních vět na menší celky. Bylo ukázáno, že proces segmentace je popsatelný několika jednoduchými pravidly a je do velké míry jednoznačný. Tento článek se částečně stal inspirací k našemu přístupu, popisovanému níže.

V dalším textu budeme pracovat s hypotézou, podle níž si lidský mozek při čtení textu nejprve všímá syntakticky klíčových slov, jak byla představena, a na jejich základě rozpoznává jisté strukturální informace ve větě obsažené. Teprve potom analyzuje další, podrobnější strukturu věty, podobně jako v případě analýzy kódu programovacích jazyků z předchozí podkapitoly.

Tato hypotéza zde nebyla nijak formálně dokázána (formální důkaz je patrně v otázkách průzkumu procesů v lidském mozku nemožný), nicméně byla předložena evidence dokládající, že některé prvky textu jsou pro lidský mozek významnější než jiné, ať již máme na mysli formální jazyky či jazyky přirozené.

3.2 Syntaktická analýza s využitím postupné segmentace věty

Formulace právě uvedené hypotézy nám nyní již relativně snadno pomůže vyjádřit principy, jichž chceme využít v návrhu nové metody syntaktické analýzy češtiny. V této sekci popíšeme návrh techniky analýzy spíše obecně, genericky, konkrétní realizací se zabýváme v následující kapitole. Začneme náčrtem obecných principů, následně budeme naši představu postupně konkretizovat.

3.2.1 Základní principy

Závěry z pozorování a základní principy metody v bodech:

- Člověk při analýze jazyka využívá detekci klíčových bodů ve větě, na základě získaných informací analyzuje větu „nahrubo" a poté pokračuje v detekci jemnějších podstruktur.

- Je velmi obtížné popsat tento proces tradičními formalismy, viz pozorování první.

- Architektura budoucího systému by se tedy měla co nejvíce blížit hypotéze o fungování lidského mozku při analýze jazyka.

- Nejprve se soustředíme na prvky textu, které lze určit relativně snadno a jednoznačně, dále pokračujeme komplikovanější analýzou.

- Detekce klíčových bodů probíhá v několika fázích či úrovních; v každé další fázi využíváme výsledků předchozí analýzy (např. postupná segmentace věty podle nalezených klíčových bodů zajistí menší rozsah nalezených segmentů a snazší další zpracování jednotlivých segmentů).

- Detekci klíčových slov a vyznačování nalezených syntaktických vztahů v segmentu obstarávají především manuálně vytvořená pravidla. Syntaktické vztahy jsou vyznačovány referencemi mezi slovy segmentu a přidáváním složkových elementů do segmentu (viz dále).

- Na každé úrovni analýzy připouštíme víceznačnost jako normální jev provázející přirozené jazyky – tuto víceznačnost budeme jistým způsobem promítat i do výstupu.

- Na každé úrovni analýzy však také zavedeme třídící (hodnotící) funkce, které vyberou nejpravděpodobnější z rozpoznaných struktur a v dalších úrovních pak pracujeme jen s touto nejlepší analýzou (důvodem tohoto postupu je efektivita výsledného algoritmu, větší samostatnost jednotlivých vrstev a transparentnost celého postupu analýzy)

- Při implementaci pravidel a třídících funkcí se budeme snažit o maximální přehlednost, deklarativnost, modularitu a rozšiřitelnost programu.

3.2.2 Schéma algoritmu

Nyní přistoupíme k více formálnímu popisu algoritmu analýzy.

Uvažujme posloupnost $U = [U_1, U_2, ..., U_n]$ úrovní analýzy. Každá z úrovní U_i představuje množinu pravidel dané úrovně, tedy

$U_i = \{P_{i,1}, P_{i,2}, ..., P_{i,m}\}$

Nechť na vstupu je věta (segment) S. Algoritmus analýzy prochází jednu úroveň po druhé a snaží se najít realizaci (též *match*) každého pravidla z příslušné množiny v daném segmentu. Pokud jsou nalezeny realizace, jsou vybrány nejlepší nekonfliktní z nich a v segmentu jsou vyznačeny příslušné syntaktické vztahy.

Podle aktuální úrovně U_i mohou být v segmentu vytvořeny subsegmenty $S_{i,1}, S_{i,2}..., S_{i,p}$, na kterých pak analýzu spouštíme rekurzivně. Důsledkem vytvoření subsegmentů je mj. to, že nemohou být přidávány žádné další vztahy mezi prvky z odlišných subsegmentů $S_{a,b}, S_{c,d}$, kde $a \neq c$ nebo $b \neq d$. Toto může být užitečné např. při analýze vsuvek či relativních vět – vytvoření subsegmentu pro relativní větu způsobí její oddělení od zbytku věty, který dále není v analýze uvažován (a symetricky, relativní věta není dále uvažována v analýze zbytku segmentu).

Precizněji ve formě pseudokódu je idea algoritmu znázorněna na obrázku 3.1.

3.2.3 Pravidla a realizace

V této části upřesníme pojem pravidla a jeho realizace v daném segmentu. Jako motivační příklad nám poslouží následující představa. Chceme, aby zápis typu

```
adj ... noun   AGREE 0 2 gnc   MARK 0   DEP 2
```

umožnil nalézt ve vstupním segmentu všechna přídavná jména následovaná (ne nutně bezprostředně) podstatným jménem taková, že obě slova se shodují v rodě, čísle a pádě. Zároveň má uvedený zápis vyznačit závislost přídavného jména na příslušném substantivu.

Nyní formálněji. Každé pravidlo $P_{i,j}$ se skládá ze šablony $T_{i,j}$ a množiny akcí $A_{i,j}$.

```
function parse(segment S):
--------------------------

init(U);     // inicializuj množinu úrovní

for level in U˜do begin
    found_matches := {};
    for rule in level do begin
        new_matches := find_matches(rule, S);
        // najdi realizace pravidla v˜segmentu
        found_matches := found_matches + new_matches;
    end;

    best_matches := select_best_matches(
                                found_matches, level);
    // vyber nejlepší nekonfliktní realizace

    for match in best_matches do begin
        // přidej do segmentu příslušné vztahy
        add_relationships(S, match);
        if creates_subsegments(level) then begin
            SS := create_subsegment(S, match);
            parse(SS);
        end;
    end;
end;
```

Obrázek 3.1: Pseudokód algoritmu analýzy

Šablona je posloupnost značek $[z_1, z_2, ..., z_n]$, kde každá značka reprezentuje jedno nebo více slov segmentu (u každé značky je přitom pevně určeno, zda může či nemůže reprezentovat více slov). Každá značka dále definuje podmínky určující, která slova může reprezentovat. Tyto podmínky mohou omezovat tvar slova, lemma, morfologickou značku a případně další atributy dostupné pro prvky vstupního segmentu.

Před objasněním pojmu akce nejprve definujeme realizaci pravidla. Budeme uvažovat segment S jako posloupnost vstupních slov $[s_1, s_2, ..., s_m]$. Realizace pravidla je pak poslounost uspořádaných dvojic

$$[(z_{i1}, s_j), (z_{i2}, s_{j+1}), ..., (z_{iq}, s_{j+q-1})]$$

taková, že $i_1 = 1$, $i_q = n$, dále $i_{k+1} = i_k + 1$ (pokud z_{ik} reprezentuje právě jedno vstupní slovo) nebo $i_{k+1} = i_k$ (jen pokud z_{ik} reprezentuje více vstupních slov) a konečně pro každou dvojici posloupnosti (z, s) platí, že s splňuje všechny podmínky definované značkou z. Definovali jsme tedy relaci, která souvislé podposloupnosti vstupních slov přiřadí podle určitých pravidel značky ze šablony pravidla. Tuto relaci budeme nazývat realizace.

Množina akcí A obsahuje akce prováděné nad realizací pravidla, resp. nad prvky segmentu, kterým je v realizaci přiřazena nějaká značka. Akce mohou být trojího typu:

- *Další omezení na realizaci.* Takováto pravidla vyjadřují další podmínky, které musí realizace splňovat a které nelze vyjádřit šablonou pravidla. Příkladem je test na gramatickou shodu u některých prvků vstupního segmentu.

- *Vyznačení dalších atributů realizace*. Tyto akce mohou z realizace vybrat některá (důležitá) vstupní slova nebo nastavit pravděpodobnostní ohodnocení příslušného pravidla. Atributy vyznačené těmito akcemi mohou být dále použity v třídících funkcích a mohou je též využívat akce třetího typu –

- *Akce přidávající vztahy do segmentu.* Tyto akce řídí činnost algoritmu v případě, že je příslušná realizace vybrána do další analýzy výběrovou funkcí *select_best_matches*. Mohou reprezentovat přidání závislosti do segmentu nebo přidání složkového uzlu do segmentu (viz též dále).

Na tomto místě se spokojíme s takto obecnou podobou definice pravidel a jejich realizací. Aktuální reprezentace pravidel v programu je popsána

v části 4.3, je však možné, že konkrétní podoba jejich formátu se bude dále vyvíjet, proto považujeme za vhodné uvést ji odděleně.

3.2.4 Formy výstupu

V této části konkretizujeme formy výstupu navrhovaného analyzátoru.

Hybridní syntaktické stromy

Výstupem z algoritmu představeného v předchozím textu bude segment obohacený o strukturní vztahy mezi jeho prvky, dle realizací pravidel vybraných v procesu analýzy. Abychom se v maximální míře vyhnuli problémům popsaným v části 2.4.4, navrhli jsme jako formát výstupu kombinaci závislostního a složkového formátu syntaktických stromů, v dalším textu jej budeme nazývat *hybridní formát*.

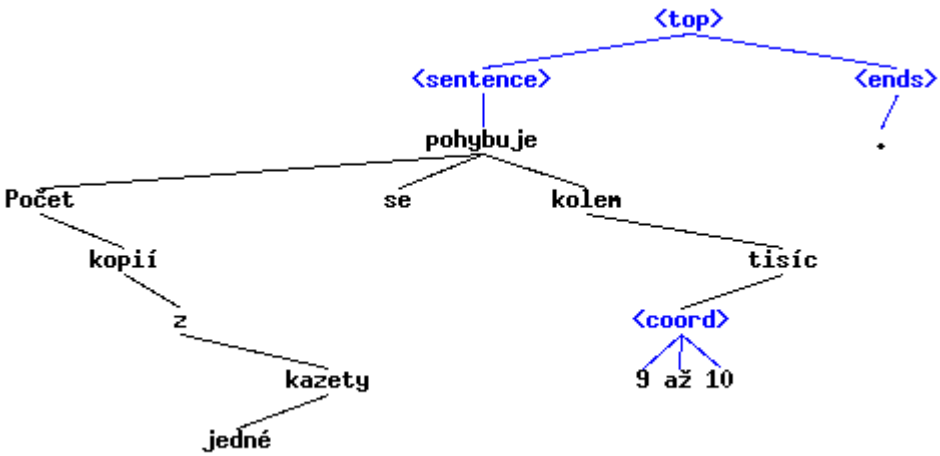
Syntaktický strom v hybridním formátu obsahuje dva typy uzlů: povrchové, které představují slova vstupní věty, a složkové, představující složkové elementy. Jeho hrany se rovněž dělí do dvou skupin. Hrany závislostní vyznačují závislostní vztahy mezi uzly, stejně jako je tomu u pražského závislostního přístupu. Hrany složkové vyznačují příslušnost uzlu do složky. Řídící uzel složkové hrany musí být vždy složkový.

Tímto hybridním formátem se snažíme reflektovat různorodost syntaktických jevů a každý z nich zachycovat způsobem, který je pro něj vhodnější. Tedy například pro analýzu jmenné skupiny (jakou je např. *„nízká rychlost přenosu"*) zvolíme závislostní formalismus. Vyhneme se tak problému s dvojím možným uzávorkováním, jak byl popsán v části 2.4.4.

Naopak pro informace typu adres, kódů, jmen firem a osob apod. volíme formát složkový (motivace opět pochází z části 2.4.4). Věříme, že dobrý analyzátor nemusí umět určit vnitřní strukturu všech možných označení kódového charakteru, jako to vyžaduje závislostní formalismus (zabýváme se analýzou *přirozeného* jazyka), stačí, když v textu kódová označení rozpozná a označí (např. připojením všech slov kódového označení pod jeden přidaný složkový element).

Složkový přístup používáme i v případě analýzy koordinací, neboť vyznačení koordinace závislostí na spojovacím výrazu (jako je tomu v PDT) považujeme za značně matoucí jak pro člověka, jenž čte výsledky analýzy, tak pro vlastní analyzátor.

Ukázku hybridního stromu můžeme vidět na obrázku 3.2. Závislostní hrany jsou vyznačeny černě, složkové modře. Složkové uzly jsou vyznačeny rovněž modrou barvou a názvem v lomených závorkách.



Obrázek 3.2: Ukázka hybridního stromu pro větu *„Počet kopií z jedné kazety se pohybuje kolem 9 až 10 tisíc."*

Závislostní výstup

Protože je velmi žádoucí, aby správnost výstupů z navrhovaného analyzátoru bylo možno změřit na korpusu PDT, bude analyzátor schopen poskytovat výstup i v čistě závislostním formátu. Toho docílíme tak, že u každého složkového elementu, přidaného do segmentu, určíme nejdůležitější slovo (hlavu) tohoto elementu a při převodu do závislostního formátu zavěsíme všechny prvky dané složky na tuto hlavu.

Víceznačnost ve výstupu

Kromě jednoznačného výstupu ve formě nejlepších vybraných realizací a syntaktického stromu, který je jimi určen, chceme na výstupu zachytit i informaci o víceznačnostech v segmentu.

Toho dosáhneme výpisem všech nalezených realizací následovaným výstupem z funkce *select_best_matches*, vybírající nejlepší z nich. Tento přístup nám umožní velmi názorný pohled na proces analýzy a kromě zachycení případných víceznačností umožní efektivní ladění pravidel a třídících funkcí.

3.3 Zařazení formalismu

Z předchozího textu, v němž jsme představili převod stromů na výstupu navrhovaného analyzátoru do závislostního formalismu, již vyplývá, že všechny struktury, které lze reprezentovat závislostním formalismem, můžeme reprezentovat i v námi navrhovaném formalismu.

V opačném směru tato převoditelnost nutně platit nemusí – námi představený formalismus dovoluje kódovat vztah mezi třemi a více slovy bez nutnosti rozlišení jemnější struktury těchto vztahů (pomocí složkových elementů), zatímco závislostní formát syntaktických stromů povoluje pouze binární vztahy. Podobně náš formalismus narozdíl od závislostního umožňuje vyjádřit souřadný vztah, opět pomocí složkových elementů. V případě anotace PDT se tyto nedostatky částečně řeší zavedením syntaktických (analytických) funkcí, jak bylo popsáno v sekci věnované závislostnímu formalismu, závislostní analyzátory však se syntaktickými funkcemi nepracují.

Srovnání našeho formalismu s formalismem složkovým vychází obdobně jako srovnání formalismů závislostního a složkového (viz úvodní kapitoly). Předností našeho návrhu oproti složkovému formalismu je zejména schopnost kódovat neprojektivní konstrukce. Převoditelnost mezi oběma formáty je možná v jistých mezích, stejných jako jsou meze převodu mezi formátem závislostním a složkovým [2].

Kapitola 4

Systém SET

V této kapitole představujeme konkrétní aplikaci metody popsané v kapitole předchozí – systém pro automatickou syntaktickou analýzu češtiny SET. Popíšeme celkový návrh systému, konkrétní formát pravidel a implementaci hodnotících funkcí s vazbou na teoretický popis z předchozí kapitoly. V závěru kapitoly podáme stručnou informaci o použití výsledného programu, který je přiložen na CD.

4.1 Návrh systému

Základní činnost systému spočívá v realizaci algoritmu specifikovaného v předchozí kapitole. Návrh odděluje znalosti ve formě pravidel od výkonného kódu programu, pravidla jsou z tohoto důvodu uchovávána ve vyhrazeném textovém souboru, odděleně od zdrojových kódů programu.

V návrhu systému jsme si stanovili několik prakticky motivovaných omezení řešeného problému. Předně se budeme věnovat analýze češtiny; návrh pravidel a případná uzpůsobení programu pro jiné jazyky ponecháváme dalšímu vývoji. Dále, na vstupu programu očekáváme správně utvořenou českou větu. Nechceme se zabývat ani rozlišováním, zda je daná věta správně utvořena,[1] ani opravami chyb ve větách či hledání správných analýz vět s chybami. Věty s gramatickými chybami algoritmus samozřejmě zpracuje, na výstupu ovšem mohou být nepřesnosti způsobené chybami ve vstupní větě.

Posledním praktickým omezením je rozhodnutí analyzovat pouze věty, které jsou jednoznačně morfologicky označkovány. Před použitím programu je tedy třeba aplikovat morfologický analyzátor a desambiguátor, stejně jako je tomu u pražských závislostních analyzátorů. Vzhledem k tomu, že desambiguátor nutně musí provádět alespoň základní povrchovou syntaktickou analýzu, dochází zde k částečnému překryvu řešených úloh. V čás-

1. Rozhodování správnosti, příslušnosti daných vět do jazyka, naráží opět na problém nenalezení shody mezi větší skupinou lidí.

ti 5.4.1 proto navrhujeme rozšíření analyzátoru o analýzu morfologicky víceznačných vstupů a aplikaci navrhovaného algoritmu na desambiguaci morfologické informace.

Vlastní činnost analyzátoru – aplikace pravidel na vstupní segment – je rozdělena do několika modulů, které popisujeme v následující části. Byl navržen objektový model reprezentace věty, pravidel, i vlastní analýzy.

Z důvodu rozšiřitelnosti, rychlosti vývoje, čitelnosti zápisu a celkové přehlednosti programu byl pro implementaci zvolen jazyk *Python*. Tato volba se může negativně projevit na rychlosti analýzy, která však pro nás v současné fázi vývoje není prioritou, navíc je problém rychlosti v případě potřeby relativně snadno řešitelný reimplementací některých klíčových modulů např. v jazyce *C*.

4.2 Implementace

V této části popíšeme vlastní implementaci systému. Popis budeme organizovat podle rozdělení systému do jednotlivých modulů.

4.2.1 Modul `grammar`

Tento modul zajišťuje načítání pravidel z definičního souboru a jejich organizaci do jednotlivých úrovní analýzy. Obsahuje definici třídy *Rule*, reprezentující pravidlo, a třídy *Grammar*, jež představuje soubor všech pravidel rozčleněných do jednotlivých úrovní analýzy.

Součástí těchto tříd je mj. analyzátor syntaxe pravidel, který převádí textovou podobu pravidel do paměťových struktur. Struktura pravidla je dána šablonou, tj. seznamem značek, a množinou akcí. Každá značka je vnitřně reprezentována jako množina podmínek zapsaných instancí objektu slovník (Python Dictionary). Akce jsou rovněž reprezentovány slovníkovými objekty, kde klíči jsou jména akcí, hodnotami seznamy argumentů akce.

4.2.2 Modul `token`

Modul `token` modeluje základní prvky vstupní věty – slova neboli tokeny. Obsahuje definici základní třídy *Token* a tří odvozených tříd: *SurfaceToken*, používané pro reprezentaci skutečných vstupních slov, *PhrToken*, reprezentující přidávané složkové elementy, a *LinkToken*, jejíž instance zajišťují vazby mezi segmentem a vytvořenými subsegmenty.

Každá instance třídy *Token* obsahuje odkaz na mateřský segment či subsegment, dále pak odkaz na token, na němž závisí, a odkaz na seznam všech potomků (tokenů, které závisí na tomto tokenu). Tyto dva poslední údaje jsou na počátku prázdné, jsou naplňovány v průběhu analýzy. Na základě údajů v nich obsažených je po skončení analýzy vykreslován výsledný syntaktický strom.

Kde je to smysluplné, instance třídy *Token* mají atributy *word*, *lemma* a *tag* určující příslušné morfologické kategorie. Morfologická značka (*tag*) je očekávána v atributovém formátu, jaký používá morfologický analyzátor `ajka` [25], vyvinutý v CZPJ FI MU, a v principu není omezena na morfologickou informaci jednotlivých slov. Vnitřní struktura analyzátoru s konkrétní podobou značky nijak nepracuje a přizpůsobení analyzátoru případným novým značkám by tedy znamenalo pouze úpravu množiny pravidel.

4.2.3 Modul `segment`

Tento modul reprezentuje vstupní větu, je v něm definována třída *Segment*. Instance této třídy obsahují seznam instancí třídy *Token* (slov, prvků segmentu) a (zpočátku prázdný) seznam svázaných subsegmentů.

Jeho metody pokrývají načítání slov z vertikálního souboru ve formátu *brief* a tisk výstupních stromů v závislostním nebo hybridním formátu na základě vztahů mezi obsaženými instancemi typu *Token*. Objekt také implementuje některé pomocné metody využívané v procesu analýzy, jako např. přidání závislosti mezi prvky segmentu nebo přidání složkového elementu spolu s příslušnými vazbami.

4.2.4 Modul `matcher`

Úkolem modulu `matcher` je reprezentace a vyhledávání realizací pravidel v segmentu. Obsahuje definice tříd *Match* a *Matcher*.

Třída *Match* představuje jednu konkrétní realizaci pravidla v daném segmentu. Obsahuje všechny informace potřebné pro vyznačení příslušných vztahů v segmentu – přiřazení jednotlivých prvků segmentu značkám v šabloně pravidla a výsledky výpočtu akcí nad danou realizací.

Třída *Matcher* zajišťuje vyhledávání všech realizací daného pravidla nad segmentem a výpočet akcí pravidla nad nalezenými realizacemi. Zde je nutno poznamenat, že časová náročnost vyhledání všech realizací může být až kvadratická vzhledem k délce segmentu. Důvodem je jednoduchý fakt, že i počet nalezených realizací může být v principu kvadratický. Důležité však je, že se jedná pouze o extrémní případy a u všech rozumně defino-

vaných pravidel je složitost nalezení realizací lineární (algoritmus je založen na průchodu segmentem a testování, zda vstupní slova splňují podmínky značek šablony daného pravidla).

Modul `matcher` též obsahuje pomocné funkce, jež testují, zda určité slovo vstupního segmentu splňuje podmínky dané určitou značkou šablony pravidla.

4.2.5 Modul `parser`

Tento modul integruje činnost všech dříve popsaných modulů a realizuje vlastní činnost algoritmu popsaného v části 3.2.2. Je v něm definována třída *Parser* se základní metodou *parse()*, přidávající strukturální vztahy do segmentu.

Obsahem třídy *Parser* jsou též hodnotící funkce pro každou z úrovní analýzy. Tyto funkce jsou vesměs založeny na jednoduchém výpočtu zahrnujícím velikost realizace (počet slov segmentu, která jsou pokryta danou realizací) a pravděpodobnost pravidla (výsledek jedné z pravidlových akcí). Podle výsledku těchto hodnotících funkcí jsou všechny realizace setříděny a do další analýzy jsou vybírány nejlepší z nich, které nejsou v konfliktu (vztahy v segmentu jimi určené nekolidují).

4.2.6 Další moduly

Implementaci doplňují moduly `set`, `tree_view` a `utils`. Modul `set` je kořenovým modulem celého programu, obsahuje analýzu příkazové řádky (pro tento úkol byl použit modul `getopt`) a spouštění analýzy podle zadaných parametrů.

Modul `tree_view` slouží ke zobrazení jednoduchého grafického výstupu výsledných stromů (program samozřejmě dává výstup i v textové podobě, tento však není pro člověka příliš přehledný). Konečně modul `utils` obsahuje několik pomocných generických funkcí, zejména pro usnadnění výpisů programu.

4.3 Systém pravidel

V této sekci popíšeme konkrétní implementaci systému pravidel podle principů uvedených v části 3.2.3. Definice pravidel pro češtinu ve formátu popisovaném v této kapitole jsou uchovávány v souboru `grammar.set` a jsou dostupné na přiloženém CD.

V analyzátoru SET jsme se rozhodli implementovat pravidla v šesti vrstvách. Každá z vrstev odhaluje odlišnou strukturní informaci. Popisujeme zde původní návrh pravidel, v dalším vývoji se může ukázat, že některé z vrstev jsou nepotřebné, či naopak je třeba přidat další. První tři vrstvy popisují jevy relativně jednoduché, další tři se postupně zabývají složitějšími a složitějšími fenomény. Analýza podle jednotlivých vrstev pravidel probíhá následovně:

1. Detekce tzv. „tvrdých" (jednoznačných) vsuvek. Tyto vsuvky vytvoří subsegment závislý na vrcholovém uzlu hlavní věty. Mohou to být např. dodatky uvedené v závorkách.

2. Detekce větných ukončení. Ukončení umisťujeme do jedné složky s vrcholovým uzlem hlavní věty.

3. Detekce „tvrdých" (jednoznačných) oddělovačů. Tyto oddělovače rozdělí větu (segment) na subsegmenty souřadně spojené složkovým elementem. Mohou to být např. středníky.

4. Detekce vedlejších vět. Vedlejší věty opět vytváří subsegmenty.

5. Detekce koordinací, méně jednoznačných vsuvek (např. oddělených čárkami) a označení kódového charakteru (data, telefonní čísla). Tyto struktury jsou reprezentovány složkovými elementy.

6. Zbylé závislostní (a tedy binární) vztahy ve větě. Tyto struktury jsou reprezentovány závislostmi mezi prvky segmentu.

V následující sekci popíšeme konkrétní formát pravidel analyzátoru, tedy kódování šablon a akcí pravidel, včetně výčtu všech dostupných akcí.

4.3.1 Formát zápisu pravidel

Pravidla zapisujeme jako šablonu následovanou seznamem akcí. Celá šablona musí být na jednom řádku a musí být uvedena klíčovým slovem `TMPL:` (včetně dvojtečky). Seznam akcí může být rozdělen v místě před klíčovým slovem (viz dále) tak, aby nový řádek začínal klíčovým slovem.

V dalším textu podrobně popíšeme formát zápisu jednotlivých komponent pravidla. Část 4.3.6 je věnována konkrétním příkladům pravidel, s nimiž může čtenář obecný popis srovnávat.

4.3.2 Formát zápisu značek šablony

Jak již bylo řečeno, šablona je seznam značek; takto ji také zapisujeme. Každá značka může být zapsána v několika formách:

- *Jediná podmínka*. Takováto značka se zapisuje v hranatých závorkách; levou závorku bezprostředně následuje atribut, na který je podmínka kladena, mezera (případně jiný bílý znak) a omezení uveideného atributu bezprostředně následované pravou hranatou závorkou.
  Schéma: `[atribut podmínka]`
  Příklad: `[lemma třeba]` – tato značka bude vyhovovat vstupním slovům, jejichž základní tvar je *třeba*.

- *Pojmenovaná proměnná*. Tato značka se zapisuje jako znak dolar (*$*) bezprostředně následovaný názvem proměnné. Jméno proměnné může obsahovat alfanumerické znaky, tečku a podtržítko. Podmínky dané značky jsou pak vyjádřeny na zvláštních řádcích pod vlastní definicí pravidla. Tyto řádky (nazývejme je *definice proměnných*) mají následující formát: `$jméno[atribut]: seznam omezení oddělený mezerami`
  Příklad: `$SPOJKA` – tato značka bude vyhovovat slovům *a, i, ani, nebo,* pokud jeden z dalších řádků bude
  `$SPOJKA[word]: a i ani nebo`

- *Alias*. Tato značka je jednou z množiny značek předdefinovaných v modulu `grammar`. Zapisuje se svým jménem a její význam je dán definicí (vždy jako jediná podmínka) v uvedeném modulu. Seznam všech aliasů dostupných v současnosti uvádíme v tabulce 4.1, seznam je však velmi flexibilní (přidání nového aliasu se provede přidáním jedné řádky kódu v souboru `grammar.py`).
  Příklad: `infinitive` – vyjadřuje totéž jako `[tag k5mF]`, sloveso v infinitivním tvaru.

Doplňující informace k prvnímu způsobu zápisu. V tomto případě může být podmínka uvedena také jako disjunkce omezení. Jednotlivá omezení jsou oddělena znakem „`|`". Například značka

```
[word a|i|ani|nebo]
```

bude mít stejný význam jako značka `$SPOJKA` z příkladu ve druhé odrážce.

Alias	Ekvivalent	Slovní popis
comma	`[word ,|-]`	čárka nebo pomlčka
like_noun	`[tag k1|k2|k3]`	substantivum, adjektivum či zájmeno
verb	`[tag k5m(IBRAP)]`	sloveso v určitém tvaru
verb_all	`[tag k5]`	sloveso v jakémkoli tvaru
adj	`[tag k2]`	adjektivum
noun	`[tag k1]`	substantivum
prep	`[tag k7]`	předložka
num	`[tag k4]`	číslovka
adv	`[tag k6]`	příslovce
infinitive	`[tag k5mF]`	sloveso v infinitivu

Tabulka 4.1: Seznam aliasů pro šablony pravidel s příslušnou sémantikou

Speciální možnost vyjádření je u omezení atributu *tag* – po uvedení libovolného atributu morfologické značky lze místo jediné hodnoty zapsat seznam hodnot v kulatých závorkách. Tento seznam je opět chápán jako disjunkce omezení. Tedy například značka šablony

```
[tag k(123)c2]
```

vyjadřuje substantivum, adjektivum nebo zájmeno, vždy však ve druhém pádě.

Doplňující informace ke druhému způsobu zápisu. K proměnné v šabloně pravidla se vždy vztahuje první výskyt definice dané proměnné následující dané pravidlo. Definice proměnných lze tedy sdílet více šablonami, což může v některých případech výrazně zestručnit zápis pravidel. V definici proměnné je možno uvést omezení na více atributů, a to zápisem na více řádků. Všechny řádky s definicí jedné proměnné musí následovat bezprostředně za sebou, v případě přerušení jiným řádkem (např. definicí jiné proměnné) je podstatná pouze první souvislá část definice. Řádky s definicemi jsou brány jako konjunkce podmínek (dané slovo musí vyhovovat všem řádkům definice). Je též možno uvádět negativní podmínky, tedy hodnoty, kterých daný atribut nabývat nesmí. Příklad:

```
$SPOJKA
...
```

```
$SPOJKA[tag]: k8xC
$SPOJKA[word not]: a i ani nebo
```

Tato značka reprezentuje všechny souřadící spojky (k8xC) s výjimkou slov *a, i, ani, nebo*.

Seznam všech atributů použitelných v zápisu pomocí pojmenovaných proměnných je následující (s příslušnou sémantikou omezení):

- `word` – tvar slova musí být jedním z řetězců ze seznamu
- `lemma` – základní tvar slova musí být jedním ze seznamu
- `tag` – morfologická značka musí souhlasit alespoň s jednou morf. značkou ze seznamu
- `word not` – tvar slova nesmí být v seznamu
- `lemma not` – základní tvar slova nesmí být v seznamu
- `tag not` – morfologická značka nesmí souhlasit s žádnou ze seznamu

V závěru popisu značek šablony zmíníme konstrukt, který je rozšířením teoretického modelu z předchozí kapitoly.

V některých případech můžeme chtít některé značky v šabloně provázat, např. chceme-li rozpoznávat koordinace a vyjádřit, že v koordinaci mohou být dvě substantiva, dvě adjektiva, ale nikoli substantivum a adjektivum. Tohoto docílíme použitím konstruktu `MATCH`, kterým je možno definovat více proměnných najednou a provázat seznamy jejich podmínek. Řešení pro uvedený příklad dostaneme následovně:

```
$C1 [word a] $C2
...
MATCH $C1[tag] $C2[tag]
k1 k1
k2 k2
END
```

Konstrukce `MATCH` plně nahrazuje definice obou zahrnutých proměnných.

4.3.3 Značky s větším rozsahem

Doposud jsme představili značky šablony, které reprezentují pouze jedno vstupní slovo. V následujících odstavcích představíme takové značky šablony, které mohou reprezentovat žádné nebo více slov vstupního segmentu. Význam těchto značek tkví v možnosti vyjádřit jevy, kdy mezi dvěma slovy v nějakém vztahu (který dané pravidlo odhaluje) jsou jiná slova, přičemž neumíme zachytit jejich přesnou podobu.

Základní značkou, reprezentující nula nebo více vstupních slov, jsou tři tečky (`...`). Význam této značky je „libovolný počet libovolných slov".

V praxi však potřebujeme tyto „mezery" v pravidle omezovat různými podmínkami. Například nechceme, aby v mezerách mezi prvky koordinace byly další souřadící spojky. Toho docílíme speciální formou značky pro pojmenovanou proměnnou:

```
$SPACE*
```

Hvězdička za pojmenováním proměnné značí libovolný počet výskytů. V jednom z dalších řádků pak uvedeme definici této proměnné (hvězdička je brána jako součást názvu proměnné, proto se vyskytuje i zde):

```
$SPACE*[tag not]: k8xC
```

Tímto způsobem jsme tedy definovali mezeru, která neobsahuje souřadící spojku. Její využití v pravidle pro koordinaci dvou substantiv může vypadat například takto:

```
TMPL: noun $SPACE* [word a] $SPACE* noun
$SPACE*[tag not]: k8xC
```

4.3.4 Značky `bound` a `rbound`

Z praktické potřeby vyplynula nutnost definovat značky, kterými lze vyjádřit začátek a konec segmentu bez vazby na konkrétní slova v segmentu. Tyto značky tedy opět překračují teoretický rámec pravidla, definovaný v předchozí kapitole.

Jsou to značky se speciálními aliasy, `bound` a `rbound`. První z nich označuje začátek segmentu, druhá z nich jeho konec. Kromě hranic segmentu se tyto značky mohou vázat také na oddělovače, např. čárku.

4.3.5 Formát akcí

Deklarace akcí následuje v zápisu pravidla deklaraci šablony. Akcí může být libovolný počet a vždy jsou zapsány ve formě klíčového slova (jména akce) následovaného seznamem argumentů akce. Některé akce se mohou odkazovat na značky šablony – děje se tak indexy příslušných značek (tedy celými čísly, vyjadřujícími pořadí dané značky v šabloně), počítáno od nuly. Odkazovány mohou být pouze značky reprezentující jedno slovo segmentu.

Počet argumentů akce může být proměnný, každá akce má však minimálně jeden argument. Následuje seznam všech v současnosti implementovaných akcí spolu s popisem argumentů, komentářem k sémantice akce a příklady:

- `MARK` – tato akce je používána pro vyznačení slov segmentu. Možnosti použití jsou dvě: první z nich způsobí přidání složkového elementu do segmentu, v takovém případě je posledním argumentem jméno nové složky, předchozí argumenty vyznačují indexy slova budoucí složky. Druhý způsob použití je vyznačení jediného prvku segmentu. Výsledku akce (označenému slovu či složkovému elementu) se poté přidává závislost akcí `DEP`.
  Příklad: `MARK 0 2 4 <coord>` – vyznačí koordinaci na slovech odpovídajících značkám s indexy 0, 2 a 4.

- `AGREE` – test na gramatickou shodu. Argumenty jsou tři: dva indexy značek, jimž příslušná slova mají být testována na shodu, třetím argumentem je řetězec tvořený názvy atributů morfologických značek. Seznam argumentů může v případě potřeby obsahovat i více trojic.
  Příklad: `AGREE 0 2 gnc` – vyjadřuje shodu v pádě, v čísle a rodě na slovech odpovídajících značkám 0 a 2.

- `DEP` – vyznačí závislost výsledku akce `MARK`. Má jediný argument, jímž je index řídícího slova.
  Příklad: `DEP 5` – přidá závislost výsledku akce `MARK` na slovu odpovídajícímu značce 5.

- `PROB` – vyjadřuje váhu pravidla, dále používanou hodnotícími funkcemi. Má jediný argument, jímž je kladné přirozené číslo (případně 0 pro některé speciální účely).
  Příklad: `PROB 5` – sníží váhu pravidla na 5 (výchozí hodnota je 100).

- `HEAD` – vyznačí hlavu složkového elementu, který byl vytvořen akcí `MARK`. Má opět jediný argument, jímž je index slova, které má být

označeno jako hlava.
Příklad: `HEAD 2`

- `IMPORTANT` – označuje některá slova segmentu za „důležitá". Tato slova se dále mohou využívat v hodnotících funkcích nebo podle nich mohou být vytvářeny subsegmenty. Argumenty akce je libovolný počet indexů značek.
  Příklad: `IMPORTANT 2 4`

4.3.6 Reálné příklady pravidel

V tomto oddílu uvedeme několik příkladů reálných pravidel systému se stručným komentářem. Kompletní soubor pravidel je přiložen spolu s programem na CD (soubor `grammar.set`). Následují příklady pravidel:

```
TMPL: noun $...* comma [tag k3yR] $...* verb $...* rbound
MARK 2 7 <relclause> DEP 0 AGREE 0 3 gn
$...*[tag not]: k3yR
```

Toto poměrně komplexní pravidlo pokrývá vedlejší větu vztažnou. Můžeme z něj vyčíst, že řídícím prvkem vztažné věty je podstatné jméno a že věta samotná je uvozena čárkou a vztažným zájmenem a obsahuje sloveso v určitém tvaru. Akce vytváří složkový element *relclause*, který závisí na řídícím substantivu. Též musí být splněna shoda v rodě a čísle mezi vztažným zájmenem a řídícím substantivem. Případná další slova v mezerách jsou zachycena značkou `$...*` s libovolným rozsahem; podle definice příslušné proměnné se v mezerách nemohou vyskytovat jiná vztažná zájmena.

```
TMPL: $1 num        MARK 0 1 <code>   HEAD 0
$1[word]: A B C D T
```

Toto pravidlo je o poznání jednodušší a přehlednější (podobně jako většina ostatních). Rozpoznává některá kódová označení, např. *A 4* pro formát papíru. Vidíme, že obě detekovaná slova jsou přidána do složkového elementu *code* a hlavou je zvoleno první z nich (podle příslušné konvence v PDT).

```
TMPL: verb ... $AND ... verb
MARK 0 2 4 <coord> HEAD 2 IMPORTANT 2 4
$AND[word]: , a ani nebo
```

Pravidlo vyjadřující koordinaci dvou sloves pomocí některých spojek. Vidíme, že obě slovesa jsou i se spojkou přidána do složkového elementu pro koordinaci a že hlavou této koordinace je zvolena (opět podle konvencí PDT) spojka. Akce *IMPORTANT* vyznačuje slova, která budou použita jako argumenty hodnotící funkce; hodnotící funkce v tomto konkrétním případě zohledňuje vzdálenost mezi vyznačenými slovy. Pro případná výplňová slova je použita obecná značka tří teček.

```
TMPL: noun $...* [tag k1c2]        MARK 2 DEP 0 PROB 500
$...*[tag not]: k5
```

Typické závislostní pravidlo. Vyjadřuje genitivní vazbu (např. *„ministr školství"*) závislostí druhého slova na prvním. Váha pravidla je zvýšena na 500, neboť se jedná o jev velmi frekventovaný. Připouští se, aby fráze byla rozdělena slovem, které není slovesem (resp. více takovými slovy).

Ačkoliv nám obecný popis pravidlového formalismu zabral mnoho místa, z uvedených příkladů lze vidět, že pravidla mají velmi vysokou expresivitu a relativně dobrou čitelnost. Fungování pravidel na principu hledání realizací v segmentu (lidově řečeno „napasování" pravidla na segment) poskytuje velmi dobrou představu o tom, co vlastně pravidla se vstupním segmentem dělají.

4.4 Použití programu

V této podkapitole uvedeme několik praktických instrukcí k použití programu `SET`. Stručně též popíšeme formát vstupu a výstupu programu.

Program se spouští z příkazové řádky příkazem `set.py` a jako jediný (povinný) argument očekává jméno souboru se vstupní větou. Před spuštěním není třeba provádět žádnou instalaci, je pouze nutné mít nainstalován interpret jazyka *Python* a knihovnu *Tkinter* (dostupnou volně v balíčku *python-tk*).

Repertoár přepínačů je poměrně malý, vzhledem k tomu, že cílem práce bylo vytvořit prototypovou implementaci metody postupné segmentace věty, nikoli široce použitelný program. Ve výchozím nastavení bez přepínačů program provede analýzu vstupní věty a na výstup vypíše hyb-

```
Šetřete <l>šetřit <c>k5eAp2nPt_mRaP
peníze <l>peníze <c>k1gInPc4
, <l>, <c>kI
netelefonujte <l>telefonovat <c>k5eNp2nPt_mRaP
, <l>, <c>kI
faxujte <l>faxovat <c>k5eAp2nPt_mRaP
! <l>! <c>kI
```

Obrázek 4.1: Ukázka vstupu ve formátu *brief* – věta *Šetřete peníze, netelefonujte, faxujte!*

ridní strom v textovém formátu. Přepínač `-d` přepne výstup na závislostní formát. Přepínač `-g` způsobí po vypsání stromu v textové podobě grafický výstup v podobě jednoduchého okna s vykresleným stromem (závislostním nebo hybridním).

V průběhu analýzy program vypisuje na standardní chybový výstup podrobné informace o výpočtu: aktuální úroveň analýzy, nalezené realizace, nejlepší vybrané realizace, vytvořené subsegmenty. Tento výpis dává vývojáři pravidel značnou kontrolu nad výpočtem programu a poskytuje velmi užitečnou a podrobnou informaci o možných nedostatcích v pravidlech. Ukázka spuštění programu na větě *Šetřete peníze, netelefonujte, faxujte!* (4. věta v PDT 1.0) je obsahem přílohy A.

4.4.1 Formát vstupu

Jak již bylo naznačeno, soubor se vstupní větou je očekáván ve vertikálním formátu *brief*. Tento formát je tvořen třemi sloupci. První z nich obsahuje slovní tvar, jak byl ve větě použit (atribut *word*). Ve druhém je uveden základní tvar, lemma, uvozené značkou `<l>`. Třetí, poslední sloupec obsahuje morfologickou značku slova a je uvozen značkou `<c>`. Příklad věty v uvedeném formátu můžeme vidět na obrázku 4.1.

4.4.2 Formát výstupu

Formát grafického výstupu programu i výpisu průběhu analýzy na standardní chybový výstup považujeme za intuitivní. Zastavíme se zde krátce jen u formátu výstupních stromů v textové podobě.

Závislostní i hybridní stromy jsou popisovány stejným formátem. Jedná se o čtyři tabulátorem oddělené sloupce, na každém řádku je popis jednoho

vrcholu výstupního stromu. První sloupec určuje identifikátor daného vrcholu, přirozené číslo. Druhý obsahuje řetězec popisující vrchol – u slovních vrcholů je to tvar slova, u složkových elementů jejich pojmenování. Ve třetím sloupci je uveden identifikátor vrcholu, na němž aktuální vrchol závisí (-1 je používáno pro kořenový vrchol). Poslední sloupec vyjadřuje typ této závislosti, obsahuje znak `p` pro složkový typ vztahu, `d` pro závislosti.

Kapitola 5

Dosažené výsledky a další vývoj

V této kapitole popíšeme experimenty a měření, které jsme s navrženým systémem SET prováděli. Pokusíme se lokalizovat časté chyby, jichž se analyzátor dopouští, a navrhnout způsoby řešení. Naznačíme též možné směry dalšího vývoje programu.

5.1 Přesnost závislostního výstupu

V této části prezentujeme měření přesnosti závislostního výstupu analyzátoru ve srovnání s dostupnými syntakticky anotovanými daty. V následujícím textu nejprve popisujeme množiny dat, které jsme zahrnuli do testování, následně uvádíme a interpretujeme dosažené výsledky.

5.1.1 Testovací data

Základní testovací množinou je pro nás podmnožina PDT, určená k slepému testování analyzátorů – *PDT e-test*.

Vzhledem k tomu, že PDT obsahuje věty, o nichž je diskutabilní, zda jsou vůbec větami (seznamy fotbalových výsledků, příklad uvedený v části 2.4.4), rozhodli jsme se do vyhodnocení zahrnout další testovací množiny vět. První z nich je tvořena větami oddílu *PDT e-test* takovými, které obsahují alespoň jedno sloveso v určitém tvaru (u těchto je větší pravděpodobnost, že budou správnými českými větami). Tuto testovací sadu budeme v dalším označovat jako *e-test-sel*.

Další dodatečnou množinou, kterou jsme se rozhodli zahrnout do vyhodnocení, je prvních 2000 vět z malého brněnského korpusu složkových stromů. Tento korpus byl vytvořen pro účely testování analyzátoru `synt` [20] a mezi jeho vlastnosti patří mj. to, že byl s pomocí analyzátoru `synt` vytvořen a obsahuje pouze věty, které tento analyzátor akceptoval. Zdrojem vět je PDT, je tedy možné věty brněnského korpusu v datech identifikovat a změřit přesnost jejich závislostní analýzy. Motivací pro tuto testovací množinu je nám povaha vět akceptovaných analyzátorem `synt` –

Testovací sada	Přesnost – průměr	Přesnost – medián
PDT e-test	75,55 %	77,27 %
e-test-sel	76,21 %	77,27 %
BPT2000	81,99 %	85,71 %
PDT100	84,08 %	88,56 %

Tabulka 5.1: Přesnost závislostního výstupu programu SET na různých testovacích sadách

jeho pravidlový formalismus by měl akceptovat pouze správně utvořené české věty a tedy korpus pravděpodobně obsahuje ve značné míře „učesanou" češtinu (bez krkolomných konstrukcí typu sportovních výsledků apod.). Tuto sadu vět dále označujeme *BPT2000*.

Poslední testovací množinou je 100 prvních vět z PDT 1.0. Důvodem této volby je skutečnost, že množina pravidel analyzátoru SET byla s pomocí těchto vět vyvíjena. Měřením přesnosti proti takovéto množině tedy můžeme odhadnout přesnost, jaké může analyzátor potenciálně na daných datech dosáhnout bez podstatných rozšíření, například tím, že by ve vývoji pravidel byli angažováni lingvističtí specialisté, nebo dalším studiem korpusových dat. Tuto testovací sadu značíme *PDT100*.

Pro všechny testovací množiny jsme využívali dostupnou morfologickou anotaci, neboť jsme nechtěli do výsledků měření zanášet chyby nižších vrstev analýzy. Chtěli jsme se omezit čistě na měření kvality analýzy syntaxe, nikoli komplexní analýzy jazyka s využitím našeho syntaktického modulu.

5.1.2 Výsledky a interpretace

V tabulce 5.1 jsou shrnuty výsledky měření přesnosti vytvořeného analyzátoru na uvedených testovacích množinách. Vidíme, že přesnost závislostního výstupu se v souhrnu pohybuje mezi 75 a 90 procenty.

Rovněž můžeme vypozorovat, že přesnost je vyšší na těch množinách, z nichž jsou odfiltrovány některé typy vět a potenciál navrhovaného analyzátoru bez výrazných rozšíření (přesnost na vývojové množině vět) je téměř 90 procent. Z vyšších hodnot mediánů lze soudit, že problémy způsobuje spíše malé množství vět, s nimiž má analyzátor větší problémy. To může být zapříčiněno absencí některých potřebných pravidel, viz též dále.

Ve srovnání s ostatními závislostními analyzátory se přesnost programu SET nejvíce blíží Žabokrtského pravidlovému analyzátoru [10]. To může

souviset s faktem, že oba programy staví na stejných základech – pro analýzu používají člověkem vytvořená pravidla a pokud můžeme z citovaného článku soudit, jejich pravidlové systémy byly vyvíjeny srovnatelnou dobu. Dovolujeme si však tvrdit, že námi navržený pravidlový formalismus má oproti Žabokrtského analyzátoru výhodu snadnější rozšiřitelnosti. Vzhledem k tomu, že pravidla jsou relativně čitelná, je též možné do procesu jejich vývoje zapojit lingvistické odborníky, kterým by psaní pravidel jako procedur v jazyce *Perl* (jak jsou implementovány v Žabokrtského analyzátoru) patrně dělalo nemalé problémy.

Některé z analyzátorů založených na učení z anotovaných dat dosahují lepších výsledků než oba pravidlové analyzátory. Domníváme se, že je to způsobeno převážně nekonzistencemi v datech, vůči nimž jsou automaticky se učící algoritmy mnohem odolnější než manuálně vytvořené systémy pravidel (nekonzistencemi a chybami v datech se dále zabýváme v další části, věnované analýze chyb). Tato odolnost je však vyvážena tím, že zmíněné analyzátory jsou velmi svázány s učícími daty, je obtížné je dále rozšiřovat a zpřesňovat a je nemožné je použít k jiným účelům, než je analýza syntaxe podle konvencí označkovaných dat (PDT). Jsme přesvědčeni o tom, že nepřítomnost těchto nedostatků kompenzuje nižší přesnost na testovacím oddílu PDT.

5.2 Analýza chyb

Při analýze chyb jsme respektovali přání anotátorů PDT, aby oddíl *e-test* nebyl přístupný při vývoji programu a podrobné výsledky analyzátoru na těchto datech jsme nezkoumali. Namísto toho jsme se důkladněji věnovali analýze chyb na testovacích datech z PDT 1.0. V následujících podkapitolách se podrobněji věnujeme nejčastějším typům chyb, s nimiž jsme se při testování setkali.

5.2.1 Nepřesnosti v PDT

Poměrně velká část chybně určených závislostí má původ jinde než v samotném analyzátoru – v testovacích datech, PDT 1.0.

Může se jednat o chybně určenou morfologickou značku slova, chybně určenou závislost, případně nedodržení konvencí anotace definovaných v Návodu pro anotátory [6] nebo nekonzistence v anotaci některých syntaktických jevů.

První dva případy jsou relativně řídké; počet zjevných chyb v datech netvoří statisticky významné procento, i když chyby jsou samozřejmě přítomny.

Podstatně závažnější jsou problémy s nedodržováním daných konvencí a s nekonzistencemi v anotaci sporných jevů, pro které konvence neexistují. Jako příklad si vezměme čtvrtou větu PDT 1.0, *„Šetřete peníze, netelefonujte, faxujte!"*. Výstup z analyzátoru SET je znázorněn na obrázku 5.1 a tuto analýzu můžeme bez rozpaků označit za stoprocentní a správnou. Podle konvencí uvedených v Návodu pro anotátory je uvedený hybridní strom převeden do čistě závislostní formy, jak je vidět na obrázku 5.2. Na obrázku 5.3 vidíme reprezentaci uvedené věty v PDT. Jak je vidět, anotátor nerespektoval doporučení Návodu (znázornění koordinace), což má v tomto případě destruktivní vliv na hodnocení naší analýzy – její přesnost vůči uvedené reprezentaci v PDT je pouhých 57 procent.

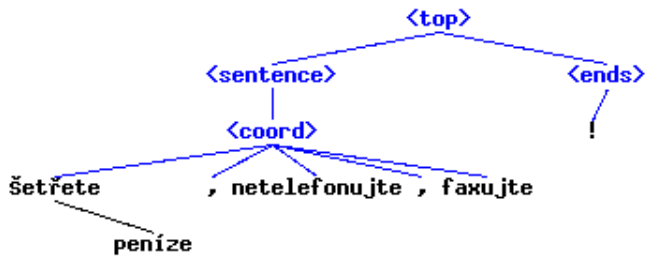
Dalším příkladem nekonzistence jsou případy přísudku vyjádřeného v trpném rodě, jako např. *„je nakupován"* (věta PDT č. 28) nebo *„je uváděn"* (věta 13). Nekonzistence je v označení řídícího slova přísudkové fráze; někdy je označen tvar slovesa *být*, jindy příčestí trpné. Vzhledem k důležitosti přísudkové fráze ve větě může tato nekonzistence postihnout i další závislostní hrany (mnoho větných členů závisí na přísudku) a výrazně ovlivnit celkové hodnocení správnosti analýzy.

Výskyt chyb a nekonzistencí podobných výše uvedeným je poměrně častý, zde uvádíme pouze špičky ledovce. Velmi hrubým odhadem (odvozeným z pozorování v průběhu několikatýdenní práce s korpusem) mohou chyby, nekonzistence a sporné závislosti pokrývat až 10, možná dokonce 15 procent celkového počtu závislostních hran v korpusu. Odpovídajícím způsobem jsou potom zkresleny výsledky vyhodnocení přesnosti syntaktické analýzy.

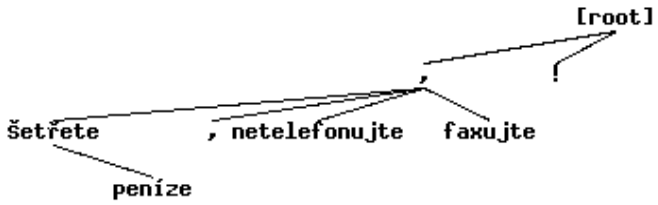
Za tohoto stavu mají výhodu algoritmy založené na strojovém učení, neboť tyto se s nekonzistencemi v datech dokáží *nějak* vyrovnat. Raději bychom ovšem dali přednost konzistentnějšímu obrazu syntaxe a konzistentním korpusovým datům. Takovéto řešení je ale bohužel zatím v nedohlednu.

5.2.2 Méně časté syntaktické jevy

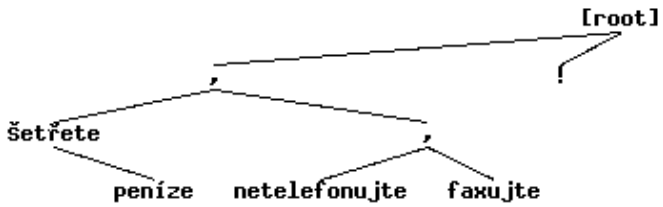
Druhý typ chyb, kterých se analyzátor dopouští, je způsoben omezenou dobou vývoje pravidel. Pravidla v současné podobě pokrývají všechny zásadní syntaktické jevy v češtině, precizní pokrytí všech fenoménů by však vyžadovalo mnohem delší vývoj, pokud je vůbec v principu dosažitelné.



Obrázek 5.1: Hybridní výstup analyzátoru SET pro větu *„Šetřete peníze, netelefonujte, faxujte!"*



Obrázek 5.2: Závislostní výstup analyzátoru SET pro větu *„Šetřete peníze, netelefonujte, faxujte!"*



Obrázek 5.3: Reprezentace věty *„Šetřete peníze, netelefonujte, faxujte!"* v PDT 1.0

Řešení těchto chyb může být dvojího charakteru:

- *Další vývoj pravidel.* Vzhledem k tomu, že základní syntaktické jevy jazyka jsou již pokryty, přidávání dalších pravidel by dále zvyšovalo přesnost jen velmi pomalu. Přínosem by jistě byla spolupráce s jazykovými odborníky, neboť ti mají o českém jazyce mnohem větší znalosti a přehled, než autor práce.
- *Automatické učení z korpusových dat.* Pro málo frekventované jevy můžeme v budoucnu využít techniky učení z příkladů pro natrénování nových pravidel z označkovaných dat a automatické doplňování pravidlové množiny.

5.2.3 Nedostatečná lexikální informace

Tento typ chyb je způsoben faktem, že na vstupu syntaktické analýzy uvažujeme pouze informace z analýzy morfologické. V některých případech totiž potřebujeme informací více. Například k tomu, abychom mohli správně analyzovat větu *„Skákal pes přes oves."* potřebujeme nějaký typ informace o tom, že *„pes přes oves"* je jiný typ fráze než např. *„pes od sousedů"*.

Informaci tohoto druhu můžeme v principu získat z různých zdrojů: lze využít např. kolokační statistiky z korpusů, data z valenčních slovníků, sémantické typy a vztahy zachycené v sémantických sítích typu WordNetu či FrameNetu a další. Je samozřejmě otázkou, jak velký vliv na přesnost analýzy bude integrace konkrétního zdroje dat mít. Nemůžeme zde uvést žádný rozumný odhad, neboť nemáme k dispozici informace o žádných podobných experimentech. Odpověď je tak pravděpodobně otázkou budoucích experimentů.

5.3 Časová náročnost

Při testování systému na korpusových datech byla měřena i doba analýzy. Jak již bylo řečeno dříve, efektivita programu v časové oblasti pro nás není prvořadou prioritou, proto jen krátce:

Asymptotická složitost algoritmu je $O(N R + R \log R)$, kde N je délka segmentu, R celkový počet pravidel. Za (rozumného) předpokladu, že každé pravidlo má konstantní počet realizací, pro každé pravidlo procházíme celý segment (první složka součtu). Nalezené realizace poté třídíme podle hodnotících funkcí s konstantní složitostí (druhá složka součtu).

Reálně spotřebovaný čas byl měřen na stroji s procesorem Intel Xeon na frekvenci 2,0 GHz a s RAM pamětí 2 GB. Pro 10 148 vět z testovací

množiny *PDT e-test* trval výpočet celkem 23 minut. To znamená průměrný čas analýzy 0,14 sekundy na jednu větu; inverzně za jednu sekundu program analyzuje v průměru 7,35 věty, což odpovídá přibližně 125 slovům.

5.4 Další vývoj

V této části krátce rozvedeme některé dříve zmíněné myšlenky týkající se rozšíření programu a navrhneme tak možné cesty dalšího vývoje analyzátoru v blízké budoucnosti.

5.4.1 Analýza víceznačných morfologických vstupů

Aby nebylo nutné před samotnou syntaktickou analýzou zanášet do vstupů chybu ve formě automaticky zjednoznačněného morfologického označkování, je třeba přizpůsobit analyzátor pro práci s víceznačnou morfologickou informací. Tato úprava též odstraní částečně duplicitní práci na syntaktické analýze, kterou nutně provádí desambiguátor.

Technická úprava analyzátoru pro práci s víceznačnými vstupy je poměrně jednoduchý úkol. Stačí při hledání realizací zohlednit všechny možné morfologické značky slov na vstupu. Pro zachování přesnosti bude však patrně třeba upravit funkce vybírající nejlepší realizace a zahrnout do nich také informaci o pravděpodobnosti dané morfologické značky (např. ve formě frekvence v korpusu).

Podle toho, které realizace budou ve výsledku vybrány, můžeme dále zpětně zpřesňovat výsledky morfologické analýzy, podobně jak je to popsáno v [16] pro analyzátor `synt`. Vzhledem k jednoznačnému výstupu analyzátoru SET můžeme pravděpodobně očekávat výraznější zjednoznačnění morfologické analýzy, než je uvedeno v odkazovaném článku, ovšem také o něco větší chybovost.

5.4.2 Využití korpusových statistik

Kolokační statistiky slov získané z korpusů mohou tvořit velmi zajímavý zdroj dat pro zpřesnění syntaktické analýzy. Základní myšlenka je jednoduchá: čím častěji se slova vyskytují v korpusu blízko sebe, tím větší je pravděpodobnost, že mají vztah i na syntaktické úrovni.

Modifikace analyzátoru by tedy spočívala pouze v úpravě hodnotících funkcí pro realizace. Bylo by ovšem také třeba vyřešit možný technický problém s velikostí kolokační databáze a efektivitou vyhledávání v ní.

Použitá statistika pro vyhledávání kolokací by přitom nemusela být klíčová – lze vyzkoušet jednoduché frekvenční statistiky, data z tzv. *word sketch* tabulek [23] nebo dokonce využít víceznačná data z průběhu analýzy pro získání korpusových kolokací. Podstatné u všech těchto možností je, že použitý korpus nemusí být syntakticky označkován a může tedy být dostatečně velký na to, aby z něj bylo možné extrahovat statisticky významné výsledky.

Kapitola 6

Závěr

Cílem práce bylo ověřit možnosti využití postupné segmentace věty v syntaktické analýze přirozeného jazyka, konkrétně češtiny, a navrhnout systém pro syntaktickou analýzu, který tuto metodu bude využívat.

Práce obsahuje tři hlavní celky. Prvním z nich je obecný přehled o současných hlavních proudech v syntaktické analýze češtiny. V této části jsme uvedli formalismy používané k zachycení syntaxe češtiny, diskutovali jejich výhody i nedostatky a představili některé analyzátory vyvíjené na základě zmíněných formalismů.

Ve druhém tematickém celku popisujeme teoretický rámec nově navrhované metody pro syntaktickou analýzu, poučeni z nedostatků diskutovaných v části první.

V poslední rámcové části se zabýváme návrhem konkrétního systému pro syntaktickou analýzu češtiny. Popisujeme jeho celkový návrh a implementaci, použitý pravidlový systém a použití výsledného programu. V závěru se zabýváme měřením přesnosti, rozborem chyb analýzy a předjímáme směry dalšího vývoje programu.

Za hlavní přínos práce považujeme návrh a realizaci nového přístupu k syntaktické analýze přirozeného jazyka. Navrhovaný přístup má četné výhody; jmenujme zde alespoň přehledný a současně velmi expresívní pravidlový formalismus, rozšiřitelnou a přehlednou implementaci a univerzální použitelnost. Z výsledků měření též vyplývá, že navržený analyzátor je srovnatelný se současnými analyzátory češtiny a po implementaci některých navrhovaných rozšíření může soupeřit o první místo mezi nimi.

Sekundárním přínosem práce jsou obsažené diskuse o reprezentaci syntaxe, vhodnosti jednotlivých formalismů pro kódování syntaxe přirozených jazyků a připomenutí obecného problému subjektivity syntaxe. Píšeme zejména o potřebě konzistentnějšího pohledu na reprezentaci syntaxe přirozených jazyků a ukazujeme konkrétní příklady, kdy současné přístupy selhávají. Věříme, že podobné úvahy povedou ke zkvalitnění formálního přístupu k přirozeným jazykům a že nás opět o krůček přiblíží ideálu umělé inteligence.

Literatura

[1] S. Abney. Part-of-speech tagging and partial parsing. *Corpus-Based Methods in Language and Speech Processing*, 2, 1997.

[2] M. Collins. dep2phr – conversion between dependency and phrase structures, 1998.
`http://ufal.mff.cuni.cz/pdt/Utilities/dep2phr/`.

[3] J. Hajič. Complex Corpus Annotation: The Prague Dependency Treebank. Bratislava, Slovakia, 2004. Jazykovedný ústav Ľ. Štúra, SAV.

[4] J. Hajič. Building a syntactically annotated corpus: The Prague Dependency Treebank. In *Issues of Valency and Meaning*, pages 106–132, Prague, 1998. Karolinum.

[5] J. Hajič, M. Collins, L. Ramshaw, and C. Tillmann. A Statistical Parser for Czech. In *Proceedings ACL'99*, Maryland, USA, 1999.

[6] J. Hajič, J. Panevová, E. Buráňová, Z. Urešová, J. Štěpánek, P. Pajas, and J. Kárník. Anotace na analytické rovině – Návod pro anotátory, 2005.
`http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/cz/a-layer`.

[7] P. Harrison, S. Abney, E. Black, D. Flickinger, C. Gdaniec, R. Grishman, D. Hindle, R. Ingria, M. Marcus, B. Santorini, and T. Strzalkowski. Evaluating syntax performance of parser/grammars of English. In J. G. Neal and S. M. Walter, editors, *Natural Language Processing Systems Evaluation Workshop: Final Technical Report RL-TR-91-362*, pages 71–77, Griffiss Air Force Base, NY, 1991. Rome Laboratory.

[8] T. Holan. Tvorba závislostního syntaktického analyzátoru. In *Sborník semináře MIS 2004*. Matfyzpress, Prague, Czech Republic, 2004.

[9] T. Holan. Genetické učení závislostních analyzátorů. In *Sborník semináře ITAT 2005*. UPJŠ, Košice, 2005.

[10] T. Holan and Z. Žabokrtský. Combining Czech Dependency Parsers. In *Lecture Notes in Artificial Intelligence, Proceedings of TSD 2006*, pages 95–102, Brno, Czech Republic, 2006. Springer Verlag.

[11] A. Horák. *The Normal Translation Algorithm in Transparent Intensional Logic for Czech*. PhD thesis, Masaryk University, 2002.

[12] A. Horák. *Computer Processing of Czech Syntax and Semantics*. Librix.eu, Brno, Czech Republic, 2008.

[13] A. Horák, T. Holan, V. Kadlec, and V. Kovář. Dependency and Phrasal Parsers of the Czech Language: A Comparison. In *Lecture Notes in Artificial Intelligence, Proceedings of Text, Speech and Dialogue 2007*, pages 76–84, Plzeň, Czech Republic, 2007. Springer-Verlag.

[14] A. Horák and V. Kadlec. New Meta-grammar Constructs in Czech Language Parser synt. In *Lecture Notes in Artificial Intelligence, Proceedings of Text, Speech and Dialogue 2005*, pages 85–92, Karlovy Vary, Czech Republic, 2005. Springer-Verlag.

[15] A. Horák and P. Smrž. Best analysis selection in inflectional languages. In *Proceedings of the 19th international conference on Computational linguistics*, pages 363–368, Taipei, Taiwan, 2002. Association for Computational Linguistics.

[16] M. Jakubíček. Extraction of syntactic structures based on the Czech parser synt. In *Proceedings of Recent Advances in Slavonic Natural Language Processing 2008*, pages 56–62, Brno, Czech Republlic, 2008. Masaryk University.

[17] V. Kadlec. *Syntactic analysis of natural languages based on context-free grammar backbone*. PhD thesis, Masaryk University, 2008.

[18] V. Kovář and A. Horák. Reducing the Number of Resulting Parsing Trees for the Czech Language Using the Beautified Chart Method. In *Proceedings of 3rd Language and Technology Conference*, pages 433–437, Poznań, 2007. Wydawnictwo Poznańskie.

[19] V. Kovář, A. Horák, and V. Kadlec. New Methods for Pruning and Ordering of Syntax Parsing Trees. In Proceedings of Text, Speech and Dialogue 2008. In *Lecture Notes in Artificial Intelligence, Proceedings of Text, Speech and Dialogue 2008*, pages 125–131, Brno, Czech Republic, 2008. Springer-Verlag.

[20] V. Kovář and M. Jakubíček. Test suite for the Czech parser synt. In *Proceedings of Recent Advances in Slavonic Natural Language Processing 2008*, pages 63–70, Brno, Czech Republlic, 2008. Masaryk University.

[21] V. Kuboň, M. Lopatková, M. Plátek, and P. Pognan. Segmentation of complex sentences. In *Proceedings of the 9th International Conference, TSD 2006*, number 4188 in Lecture Notes In Computer Science, pages 151–158. Springer-Verlag Berlin Heidelberg, 2006.

[22] R. McDonald. *Discriminative learning and spanning tree algorithms for dependency parsing*. PhD thesis, University of Pennsylvania, 2006.

[23] P. Rychlý and P. Smrž. Manatee, Bonito and Word Sketches for Czech. In *Proceedings of the Second International Conference on Corpus Linguisitcs*, pages 124–132, Saint-Petersburg, 2004. Saint-Petersburg State University Press.

[24] G. Sampson. A Proposal for Improving the Measurement of Parse Accuracy. *International Journal of Corpus Linguistics*, 5(01):53–68, 2000.

[25] R. Sedláček. *Morphemic Analyser for Czech*. PhD thesis, Masaryk University, 2005.

[26] E. Žáčková. *Parciální syntaktická analýza (češtiny)*. PhD thesis, Masaryk University, 2002.

[27] D. Zeman. Neprojektivita v Pražském závislostním korpusu (PDT). Technical Report TR-2004-22, ÚFAL/CKL MFF UK, Prague, 2004.

Dodatek A

Příloha A: Ukázka spuštění programu

```
$ set.py -g ../pdt/brief/00004

Parsing segment 0: Šetřete peníze , netelefonujte , faxujte !
--------------
interjections ...
ends ...
  Match found: !
  Sub-segment created: Šetřete peníze , netelefonujte , faxujte
  Sub-segment created: !
  Phrase created: <top> ::: <sentence> <ends> ::: head = <sente
nce>

Parsing segment 0.0: Šetřete peníze , netelefonujte , faxujte
--------------
interjections ...
ends ...
hard delimiters ...
relative clauses ...
coordinations and other constructs ...
  Match found: peníze , netelefonujte ,
    Rule: noun $...* comma $...* verb $...* rbound MARK 2 4 6 <
intr> DEP 0 HEAD 4 IMPORTANT 0 2 4 PROB 2
  Match found: Šetřete , netelefonujte
    Rule: [tag k5m(IBRAP)] ... $AND ... [tag k5m(IBRAP)] MARK 0
 2 4 <coord> HEAD 2 IMPORTANT 2 4 PROB 10000
  Match found: Šetřete , faxujte
    Rule: [tag k5m(IBRAP)] ... $AND ... [tag k5m(IBRAP)] MARK 0
 2 4 <coord> HEAD 2 IMPORTANT 2 4 PROB 10000
  Match found: Šetřete , faxujte
    Rule: [tag k5m(IBRAP)] ... $AND ... [tag k5m(IBRAP)] MARK 0
```

```
 2 4 <coord> HEAD 2 IMPORTANT 2 4 PROB 10000
  Match found: netelefonujte , faxujte
    Rule: [tag k5m(IBRAP)] ... $AND ... [tag k5m(IBRAP)] MARK 0
 2 4 <coord> HEAD 2 IMPORTANT 2 4 PROB 10000
  Match found: Šetřete , netelefonujte
    Rule: $1 $...* comma $...* $3 MARK 0 2 4 <coord> HEAD 2 AGR
EE 0 4 c PROB 0
  Match found: netelefonujte , faxujte
    Rule: $1 $...* comma $...* $3 MARK 0 2 4 <coord> HEAD 2 AGR
EE 0 4 c PROB 0
  Match selected: netelefonujte , faxujte :: <coord> :: netelef
onujte , faxujte
  Match selected: Šetřete , netelefonujte :: <coord> :: Šetřete
 , netelefonujte
  Phrase created: <coord> ::: Šetřete , netelefonujte , faxujte
 ::: head = ,
dependencies ...
  Match found: Šetřete peníze
    Rule: verb_all ... noun MARK 2 DEP 0
  Match found: Šetřete peníze
    Rule: verb_all noun MARK 1 DEP 0 PROB 200
  Match found: peníze netelefonujte
    Rule: noun ... verb_all MARK 0 DEP 2 PROB 80
  Match found: peníze faxujte
    Rule: noun ... verb_all MARK 0 DEP 2 PROB 80
  Match found: Šetřete peníze
    Rule: verb_all ... [tag k(13)c4] MARK 2 DEP 0 PROB 300
  Match found: peníze netelefonujte
    Rule: [tag k(13)c4] ... verb_all MARK 0 DEP 2 PROB 150
  Match found: peníze faxujte
    Rule: [tag k(13)c4] ... verb_all MARK 0 DEP 2 PROB 150
  Match found: peníze ,
    Rule: [tag k.] [tag kI] MARK 1 DEP 0 PROB 20
  Match found: netelefonujte ,
    Rule: [tag k.] [tag kI] MARK 1 DEP 0 PROB 20
  Match selected: Šetřete peníze
    Rule: verb_all ... [tag k(13)c4] MARK 2 DEP 0 PROB 300
segment head selection ...
  Head selected: <coord>
======================
```

```
Parsing segment 0.1: !
--------------
interjections ...
ends ...
hard delimiters ...
relative clauses ...
coordinations and other constructs ...
dependencies ...
segment head selection ...
  Head selected: !
=======================

Back to segment 0: <sentence> <ends> <top>
--------------
hard delimiters ...
relative clauses ...
coordinations and other constructs ...
dependencies ...
segment head selection ...
  Head selected: <top>
=======================
0 Šetřete 6 p
1 peníze 0 d
2 , 6 p
3 netelefonujte 6 p
4 , 6 p
5 faxujte 6 p
6 <coord>7 p
7 <sentence>10 p
8 ! 9 p
9 <ends>10 p
10 <top>-1 p
```