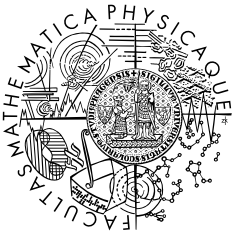
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Ondřej Bojar

Automatická extrakce lexikálně-syntaktických údajů z korpusu

Praha 2002

Ústav formální a aplikované lingvistiky
Vedoucí diplomové práce: RNDr. Vladislav Kuboň, Ph.D.
Studijní program: Informatika, Počítačová a formální lingvistika

Rád bych na tomto místě vyslovil poděkování pracovníkům Ústavu formální a aplikované lingvistiky a Centra komputační lingvistiky za laskavé přijetí a veškerou technickou podporu; zejména pak vedoucí katedry, prof. Evě Hajičové, za starostlivou péči o tento kolektiv.

Chtěl bych rovněž poděkovat dr. Vladislavu Kuboňovi za vedení této práce. Zdeňkovi Žabokrtskému pak patří zvláštní dík za nesčetné konkrétní náměty a připomínky a především za vždy inspirující způsob práce a uvažování.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 13. prosince 2002.

Ondřej Bojar

Obsah

Seznam tabulek

Seznam obrázků

Název práce: Automatická extrakce lexikálně-syntaktických údajů z korpusu
Autor: Ondřej Bojar
Katedra: Ústav formální a aplikované lingvistiky
Vedoucí diplomové práce: RNDr. Vladislav Kuboň, Ph.D.
E-mail vedoucího: vk@ufal.ms.mff.cuni.cz

Abstrakt: V práci studujeme možnosti automatizovaného získávání lexikálně-syntaktických údajů z korpusů, konkrétně se zaměřujeme na extrakci slovesných rámců. Upozornili jsme, že pro tento účel korpusy PDT ani ČNK svým rozsahem plně nepostačují. Implementovali jsme jednoduchý nástroj pro selektivní rozšiřování korpusů na základě textů z Internetu. Vyhodnotili jsme tři dostupné syntaktické analyzátory češtiny z často opomíjených, a přesto významných hledisek. Implementovali jsme vlastní systém filtrace vstupních vět, který identifikuje "velmi jednoduché věty". Na těchto větách parsery dosahují vyšší úspěšnosti. Vytvořený systém AX pro filtraci vět je obecný, filtry lze zaměřit na zisk příkladů pro extrakci libovolných typů údajů z korpusu. Systém lze použít mj. k částečné či úplné syntaktické analýze vět a v práci je představen formou uživatelské příručky. Dále studujeme možnosti zpracování pozorovaných rámců na rámce povrchové a valenční. Implementujeme řazení pozorovaných rámců do hierarchie, která je vhodným podkladem pro anotátora. Závěrem upozorňujeme na problémy plně automatického zpracování pozorovaných či povrchových rámců na rámce valenční.
Klíčová slova: korpus, extrakce, povrchové rámce, valenční rámce

Title: Automatic Extraction of Lexico-Syntactic Information from Corpora
Author: Ondřej Bojar
Department: Institute of Formal and Applied Linguistics
Supervisor: RNDr. Vladislav Kuboň, Ph.D.
Supervisor's email address: vk@ufal.ms.mff.cuni.cz

Abstract: The presented work investigates methods for semi-automatic extraction of lexico-syntactic information from corpora, particularly the information on subcategorization and valency frames. We document that at present time, PDT and CNC corpora are not sufficient for this task. We describe a simple method for a selective extension of corpora based on texts from Internet. We evaluate three parsers available for Czech with respect to the task of extracting verb frames. We have implemented a linguistically motivated filtration of input sentences to identify "very simple sentences", which helps the parsers to achieve better accuracy. The system AX designed in this work is more generic, any kind of linguistic filtration can be employed. The system is also suitable for creating partial or full parsers of natural languages. The thesis also presents a user's guide to the system AX. Furthermore, we compare methods for extraction of subcategorization frames from observed frames. We classify observed frames into a hierarchy suitable for human anotators. Finally, several problems of automatic extraction of valency frames are discussed.

Kapitola 1

Úvod

1.1 Motivace

Lingvistika jako věda studující přirozený jazyk v současné době prochází významným obdobím. V minulosti teoretická lingvistika vybudovala komplexní teorie a formalismy pro popis přirozeného jazyka. V relativně nedávné době se začala též významně rozvíjet specifická oblast lingvistiky – lingvistika korpusová. Ta díky technickým možnostem dnešních počítačů shromažďuje rozsáhlé zdroje textů přirozeného jazyka a snaží se pravidelnost v jazyce odhalit přímo z těchto dat. Konfrontace teorie a empirických pozorování prospívá oběma proudům lingvistiky.

Praktické aplikace lingvistiky jako celku zajišťuje obor zpracování přirozeného jazyka (natural language processing, NLP). Specifickým cílem NLP je budovat automatické systémy manipulující s texty (a dalšími projevy) přirozeného jazyka, ať již se jedná o systémy automatického překladu, zodpovídání otázek, automatické souhrny textu ap. Přitom celá řada z těchto aplikací se v současné době stává především díky rozkvětu celosvětové počítačové sítě, Internetu, velmi atraktivní a žádanou. Praxe však ukázala, že většiny svých cílů nemůže NLP dosáhnout, nevyužije-li poznatků lingvistiky.

Obor komputační lingvistiky pak tvoří most mezi NLP a teoretickou lingvistikou.

Mezi aktuální rozpracované otázky lingvistiky a aktuální palčivé problémy NLP se řadí syntax přirozeného jazyka. V současné době je zřejmé, že syntax přirozeného jazyka není možné adekvátně popsat, nevybudujeme-li rozsáhlé slovníky charakterizující lexikální jednotky z hlediska jejich chování v celkové skladbě věty. Podobně jsou pro obor NLP tyto lexikálně-syntaktické údaje velmi významnou, ne-li nutnou podmínkou úspěchu při snaze o automatickou syntaktickou analýzu vět, která je východiskem pro většinu z požadovaných aplikací.

Budování rozsáhlých a bohatých slovníků je však náročná a dlouhodobá práce, a je proto žádoucí část úlohy zautomatizovat a přenechat výpočetní síle počítačů. Právě korpusy pak představují vhodný zdroj vstupních dat pro

automatizované metody.

Cílem této práce je navrhout metodu získávání lexikálně-syntaktických údajů z korpusu. Z teoretické lingvistiky je přitom známo, že ústředním organizujícím prvkem ve větě je sloveso, a potřebu podrobnějších údajů o typickém chování jednotlivých sloves projevují i systémy automatické syntaktické analýzy. Z tohoto důvodu se tato práce konkrétněji zaměřuje na extrakci údajů charakterizujících právě slovesa.

Korpusy sestávají z autentických příkladů jazykových výpovědí, a systémy automatizované extrakce údajů proto čelí dvěma zásadním problémům[1]:

- Hledané obecně syntaktické údaje jsou v dostupných korpusech přítomny ve velmi "zředěné" podobě. Je třeba projít velké množství příkladů, aby bylo možné usuzovat na obecné pravidlo (a spolehlivého přesvědčení se nedočkáme nikdy).
- Každá jednotlivá výpověď zachycená v korpusu v sobě nese celou složitost jazykového systému. Při snaze o odpověď na jednu konkrétní otázku se proto potýkáme i s dalšími složitými otázkami, na než dosud odpovědi nalezeny nebyly.

V naší práci budujeme systém automatizované extrakce lexikálně-syntaktických údajů na základě korpusových dat, s oběma zmíněnými problémy tedy přicházíme do přímého kontaktu.

První z problémů, dostupnost dostatečného množství příkladů v korpusech, nemůže přímo tato práce řešit. Ve zúženém zadání, tj. extrakce informací charakterizujících slovesa, však práce tento problém alespoň podrobně studuje a varuje před ním. Náznak řešení, selektivní rozšiřování korpusů, je však přesto nastíněn.

Druhým z problémů, složitostí dostupných příkladů, se práce zabývá podrobněji a nabízí vlastní systém sofistikované filtrace dostupných dat tak, aby svou složitostí nebránila v získávání hledaných údajů.

Při řešení problémů automatizované extrakce údajů z korpusových dat práce důkladně čerpá z poznatků teoretické lingvistiky a záměrně tyto poznatky staví do kontrastu s konkrétními pozorováními.

Shrňme tedy zúžený cíl této práce:

Cílem práce je navrhnout metodu, pomocí níž bude možné částečně automaticky získávat lexikálně-syntaktické údaje o slovesech na základě dostupných korpusových dat. Metoda budovaná v této práci však počítá s tím, že otázka syntaxe přirozeného jazyka chováním sloves nekončí. Systém připravený v rámci této práce je proto navržen obecněji, tak, aby jej bylo možné později použít i pro další úlohy extrakce údajů z velkého množství příkladů vět.

Práce předpokládá rámcovou znalost Funkčního generativního popisu (FGP, viz Sgall (1967; Sgall, Hajičová, and Panevová (1986)) a rovin jazykového

[1] Pomineme-li klasickou otázku *reprezentativity* korpusu vzhledem k jazyku – komunikačnímu systému – jako celku.

popisu, jak je FGP definuje. Představu o rovinách jazykového popisu a závislostní syntaxi může rovněž dobře poskytnout Mel'čuk (1988).

1.2 Obsah práce

Úvodní kapitola charakterizuje současný stav znalostí potřebných pro automatickou analýzu přirozeného textu a podrobně popisuje zaměření této práce: získávání typických doplnění českých sloves.

Kapitola 3 na straně 22 představuje lingvistické těžiště této práce. Kapitola je konkrétním příspěvkem k automatizované extrakci slovesných rámců, obecný postup úvah v kapitole je však vhodné použít i při hledání odpovědí na jiné specifické otázky syntaxe přirozených jazyků.

Kapitola nejprve srovnává dostupné zdroje dat, jež lze pro extrakci syntaktických informací o slovese použít. V další části popisuje lingvisticky motivované filtrace vstupních dat. Na filtrovaných větách pak algoritmy extrakce budou probíhat úspěšněji. Součástí kapitoly je srovnání výsledků, které je možné pro účel extrakce typických doplnění slovesa získat za použití existujících syntaktických analyzátorů češtiny. Závěrem kapitoly studujeme možnosti zpracování pozorovaných doplnění slovesa tak, abychom získali povrchové rámce sloves.

V kapitole 4 na str. 46 konfrontujeme současné teoretické cíle s realitou pozorování anotovaných dat a studujeme obtížnost vytčených cílů teoretické lingvistiky.

Podstatná část této práce (kap. 5 na straně 53) je věnována popisu formalismu a systému automatizované extrakce, který byl pro účely práce vytvořen, ale svým zaměřením přesahuje rámec úzkého cíle extrakce slovesných rámců. Systém AX byl v této práci použit pro filtraci vstupních dat. Díky obecnosti návrhu systému je však možné v tomto systému implementovat obdobné filtrace vedené s jiným cílem, nebo i samotnou extrakci konkrétních údajů. Systém AX je rovněž vhodný pro budování částečných syntaktických analyzátorů řízených přesnými pravidly, např. jako preprocessing před nasazením jiných technik syntaktické analýzy.

Systém AX je proto představen formou uživatelské příručky, tak, aby jeho možnosti byly dostupné každému zájemci o extrakci konkrétních lexikálně-syntaktických údajů. Příklad konkrétních pravidel pro filtraci vět pro účely této práce je pak k dispozici v příloze A na str. 84.

Vedlejším produktem této práce je celá řada nepublikovaných nástrojů pro práci s tabulkami ap. Jeden z pomocných nástrojů, program na procházení strukturovaných údajů, však pro jeho zajímavost představujeme v příloze C na straně 100.

1.3 Současný stav znalostí

Rámec FGP, viz Sgall (1967), užívaný pro formální popis přirozeného jazyka dobře strukturuje informace obsažené ve větách přirozeného jazyka. FGP definuje několik rovin popisu: rovinu morfologickou, rovinu povrchové syntaxe (v konkrétní variantě označované též jako rovina analytická) a rovinu hloubkové syntaxe (též rovina tektogramatická).

V současné počítačové lingvistice budované v rámci FGP je pro češtinu rovina morfologická dobře zpracována. Existuje fungující systém pro převod vět přirozeného jazyka do zcela formalizovaného zápisu na morfologické rovině (avšak s morfologickou víceznačností) a slovník užívaný tímto systémem pokrývá drtivou část současné češtiny. Existuje několik variant statistických algoritmů (viz např. Hajič and Hladká (1997; Hajič and Hladká (1998a; Hajič and Hladká (1998b; Hajič et al. (2001)), které nejednoznačný morfologický zápis zjednoznačňují (tzv. *taggery*). Taggery pro češtinu, jejíž morfologický systém je velmi bohatý (teoreticky 4 000 různých značek), dosahují správného rozhodnutí pro 90 až 95 % slov. Existuje velmi rozsáhlý soubor textů (Český národní korpus, ČNK[2]) anotovaný na morfologické rovině s automaticky zjednoznačněným morfologickým zápisem.

Rovněž je dobře připraven formalismus pro zápis informací na analytické i tektogramatické rovině (datový formát FS a novější CSTS) a existuje soubor textů jednoznačně anotovaný na analytické rovině (Pražský závislostní korpus, Prague Dependency Treebank, PDT[3]).

Souběžně je budováno několik rozdílných systémů automatického zpracování vět přirozeného jazyka do analytické[4] či rovnou do tektogramatické roviny[5]. Všechny tyto systémy, i když třeba zpracovávají jen do nižší, tj. analytické, roviny, však čelí řadě složitých problémů. Největším z těchto problémů se v současné době jeví nedostatečnost stávajících slovníků, které by syntaktickým analyzátorů dávaly radu, jak z daných slov budovat syntaktickou stromovou strukturu.

Podle teorie FGP jsou pro syntaktickou analýzu věty klíčové tzv. *valenční rámce* sloves i dalších slovních druhů. Valenční rámec je pro dané sloveso slovníkovou informací a charakterizuje, jaké větné členy sloveso vyžaduje jako svá doplnění a jaké větné členy mohou sloveso rovněž rozvíjet. Teorie FGP, jak podrobně rozpracovává Panevová (1980), přesně rozlišuje především doplnění *volná* (např. časová či místní určení, která mohou rozvíjet prakticky libovolné sloveso, a to i v rámci jedné věty opakovaně) od *aktantů* (např. konatel ACT, pacient PAT ad., jejichž sestava je pro jednotlivá slovesa typická a které se nemohou u konkréního výskytu slovesa opakovat). Do valenčních rámců čistá teorie FGP pak řadí pouze aktanty a ta volná doplnění, která jsou pro dané sloveso *obligatorní*, tj. jejichž hodnotu v konkrétním případě musí původce věty nutně znát, aby mohl dané sloveso vůbec použít[6].

[2] `http://ucnk.ff.cuni.cz/`
[3] `http://ufal.mff.cuni.cz/pdt/`
[4] Podrobněji viz 3.1 na str. 24 a též Kuboň (2001).
[5] Přípravy viz Honetschläger (2002).
[6] Podrobně o tzv. *dialogovém testu* viz Panevová (1980)

Pojem rámce podle teorie FGP se odvolává především na tektogramatickou rovinu zápisu věty. Z hlediska praktického zpracování vět přirozeného jazyka je však nutné poznamenat, že i aktanty a *obligatorní* doplnění slovesa na povrchové rovině a tedy i na rovině povrchové syntaxe velmi často chybí. (Mluvčí i posluchač mohou doplnění znát z kontextu věty a ve větě je pro úsporu neuvádějí.) Naproti tomu určitá volná doplnění, jež podle FGP mohou rozvíjet libovolné sloveso, jsou pro určitá slovesa poměrně typická, a tato informace by systémům automatické syntaktické analýzy velmi pomohla. Z těchto důvodů je v budovaných valenčních slovnících[7] často k rámci slovesa připojena též informace o dalších druzích doplnění mimo aktanty (doplnění *kvazivalenční* a *typická* volná doplnění). Jak uvidíme v kapitole 2.3 na straně 19, stávající valenční slovníky zatím nejsou bohužel dostatečně rozsáhlé na to, aby pokrývaly významnou část českých sloves nebo dokonce dalších slovních druhů.

V této souvislosti je rovněž nutné podotknout, že ve valenčním rámci, tak jak je definován pro tektogramatickou rovinu, jsou doplnění slovesa charakterizována především svou větně členskou *funkcí*. Přitom funkce větného členu ve větě není z povrchového syntaktického rozboru věty vůbec zřejmá, jak rozpracovává i sama teorie a jak uvidíme též v kapitole 4.2 na str. 48. Valenční slovníky budované pro automatickou syntaktickou analýzu (již citovaný Straňáková-Lopatková and Žabokrtský (2002; Žabokrtský et al. (2002) a především brněnský BRIEF, Pala and Ševeček (1997; Horák (1998)), tedy zahrnují do popisu rámců nejen větně členskou funkci doplnění slovesa, ale též časté či povolené *formy* povrchové realizace doplnění. Brněnský BRIEF pak uvažuje právě jen povrchové valenční rámce.

1.4 Zaměření této práce

Cílem práce je navrhnout postup získávání lexikálně-syntaktických údajů o českých slovesech poloautomatickým způsobem. Získané informace vystihnou, jaká doplnění (volná i aktanty) jsou pro dané sloveso typická, a budou tak dobrou pomocnou informací pro automatické syntaktické analyzátory.

Postup získávání lexikálně-syntaktických údajů je navržen tak, aby jej bylo možné použít i na text, který *není* syntakticky analyzován a teoreticky nemusí být ani morfologicky zjednoznačněn. Otvírá se tím možnost použít velmi rozsáhlého Českého národního korpusu nebo teoreticky úplně libovolných textů získaných např. z Internetu. Kapitola 2.3 na straně 19 jasně dokládá, proč je pro vytyčený úkol nezbytné vycházet z většího množství byť méně anotovaných dat, než v současné době nabízí Pražský závislostní korpus.

Je třeba zdůraznit, že bezprostředně "pozorované rámce" pro česká slovesa není možné ani teoreticky považovat za *valenční rámce* podle teorie FGP. Jak již bylo naznačeno v předešlé kapitole, v české větě se ani obligatorní doplnění nemusí vyskytovat a na druhou stranu je u nalezeného doplnění automaticky velmi obtížné určit, zda jde o doplnění volné nebo ne. Navržený postup přitom vychází výhradně z dat, která lze v korpusech pozorovat. Kapitola 4 na str. 46

[7] Viz Skoumalová (2001) a Straňáková-Lopatková and Žabokrtský (2002; Žabokrtský et al. (2002).

sice naznačuje cestu, jak k valečním rámcům směřovat, přesahuje však dosah možností této práce. Praktickým důkazem obtížnosti tohoto úkolu je, že valenční slovník pro češtinu dosud neexistuje.

Kapitola 2

Dostupné zdroje dat

V práci usilujeme o zjištění typických doplnění sloves na základě pozorovaných českých vět. V následujících kapitolách srovnáme dostupnost a množství potřebných údajů v Českém národním korpusu a v Pražském závislostním korpusu. K dispozici je v současné době korpusů českých textů více[1], syntaktickou anotaci dat nabízí však pouze PDT, a z morfologicky anotovaných korpusů je ČNK nejrozsáhlejší.

2.1 Charakter a velikost ČNK a PDT

Nejprve stručně představme oba použité korpusy.

2.1.1 Český národní korpus (ČNK)

Český národní korpus (ČNK[2], Kocek, Kopřivová, and Kučera (2000)) obsahuje v oddíle současné psané češtiny (SYN2000) celkem 100 000 704 slov v 1 763 818 větách, anotovaných a automaticky zjednoznačněných na morfologické úrovni.

Většina textů ČNK pochází z denního tisku (Mladá fronta DNES, deníky vydavatelství Lidové noviny, Hospodářské noviny ad., celkem 60 %). Zbývající část tvoří odborná literatura (25 %,) a též beletrie (15 %).

Ústav Českého národního korpusu v současné době kromě neustálého rozšiřování ČNK připojuje i další typy korpusů, např. korpus mluvené řeči.

2.1.2 Pražský závislostní korpus (PDT)

Pražský závislostní korpus (Prague Dependency Treebank, PDT[3], Böhmová et al. (2001)) se opírá o texty pocházející z ČNK a anotuje je na analytické

[1] Viz např. DESAM, Pala, Rychlý, and Smrž (1997).
[2] `http://ucnk.ff.cuni.cz/`
[3] `http://ufal.mff.cuni.cz/pdt/`

(viz Hajič and others (1997; Hajič (1998)) a tektogramatické rovině popisu (viz Hajičová, Panevová, and Sgall (1999)).

Data anotovaná na analytické rovině v rozsahu 456 705 tokenů (slova a interpunkce) v 26 610 větách[4] byla vydána v roce 2001 jako PDT verze 1. V současné době probíhá anotace těchže vět až na tektogramatickou rovinu a vydání s označením PDT verze 2 je plánováno na rok 2004. V následujícím textu, není-li výslovně řečeno jinak, se tedy odvoláváme k PDT verze 1, anotovaném na analytické rovině popisu jazyka.

2.1.3 Typ anotace ČNK a PDT ve vztahu k extrakci slovesných rámců

Pro extrakci povrchových slovesných rámců nabízí PDT ve verzi 1 data anotovaná na velmi dobré úrovni. Ze stromových struktur povrchové syntaxe vět stačí nechat vypsat všechna slovesa a (po drobných formálních úpravách týkajících se především koordinace a dále např. předložek či vedlejších vět) všechny syny uzlů sloves. Tím přímo získáme seznam všech doplnění slovesa.[5]

Extrakce doplnění sloves z ČNK, který je anotován pouze morfologicky, je již lingvisticky i algoritmicky obtížnější záležitost a podrobně se jí věnuje kapitola 3 na str. 22.

Krok od rámců povrchových k rámců valenčním je při současných lingvistických znalostech zatím automatickým způsobem téměř nerealizovatelný a přesahuje rámec této práce. Přesto kapitola 4 na straně 46 rozvádí alespoň otázky, které je třeba pro tento úkol řešit.

2.2 Slovesa v ČNK a v PDT

Naše práce se zaměřuje na určení typických doplnění slovesa na základě velkých korpusových dat. Než samotnou práci extrakce zahájíme, je třeba si uvědomit, zda je pro daný konkrétní úkol k dispozici dostatek korpusových dat. Podívejme se proto podrobněji na množství výskytů jednotlivých českých sloves, která lze v ČNK a v PDT najít.

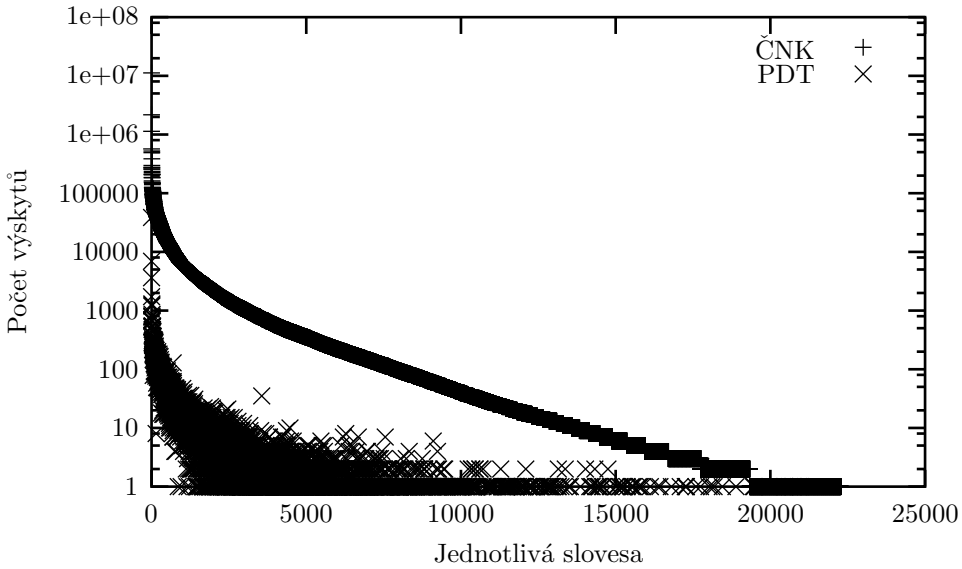
Pro přesnost je třeba uvést, že za jedno sloveso pokládáme jedno lemma, u něhož morfologická analýza označila slovní druh jako sloveso. Například pro složené slovesné tvary jsou zde tedy započítány všechny složky zvlášť. Naopak nejsou naprosto odlišena slovesa zvratná od svých nezvratných protějšků, jak by bylo žádoucí z lexikografického hlediska.

Graf 2.1 na následující straně srovnává počty výskytů sloves jednotlivých sloves v obou korpusech.

V ČNK je evidováno celkem 22 276 sloves, v PDT celkem 2 817. Odhadnout

[4] Korpus je účelně rozdělen na data určená k ladění a trénování a na dvě oddělené sady vět určených výhradně k závěrečnému testování (evaluaci) algoritmů.

[5] Viz však sekci 3.4 na straně 37 pro přehled problémů extrakce povrchových slovesných rámců z pozorovaných doplnění slovesa.



Obrázek 2.1: Graf srovnávající počet výskytů sloves v ČNK a PDT

celkový počet sloves v češtině je velmi obtížné, pravděpodobně by se jejich počet mohl pohybovat okolo 40 tisíc.

V grafu stojí za povšimnutí *výrazně neuniformní rozdělení* sloves podle výskytů. Křivka výskytů sloves v ČNK i obálka křivky výskytů sloves v PDT se velmi rychle přibližují nízkým hodnotám, a to i přes logaritmickou škálu použitou na ose y. Ukazuje se, že podobný charakter rozložení je možné spatřit prakticky i u všech jazykových jevů.

Jako jiný pohled na tytéž informace mohou posloužit dvě následující tabulky. Obě tabulky zařazují česká slovesa do skupin tak, aby souhrnný počet výskytů sloves byl pro všechny skupinky přibližně stejný. Tabulka 2.1 se věnuje všem slovesům evidovaným v ČNK, tabulka 2.2 na následující straně je pak jen jemnějším pohledem na poslední skupinu sloves.

		Výskyty v ČNK	**PDT**		
Skupina	**Sloves**	(min, ∅, max)	**Sloves**	**Výskytů**	**Příklady**
1	1	11 253 207	1 (100,0 %)	1 737	být
2	1	2 175 254	1 (100,0 %)	513	mít
3	2	570 234, 851 099,5, 1 131 965	2 (100,0 %)	116, 159,5, 203	moci, muset
4	21	140 362, 243 575,3, 522 307	17 (81,0 %)	21, 106,3, 524	říci, chtít, jít, dát, uvést
5	53	68 535, 92 773,1, 126 411	51 (96,2 %)	2, 45,2, 133	čekat, zůstat, znamenat
6	99	40 248, 50 606,4, 68 004	94 (94,9 %)	1, 31,8, 99	představit, věnovat, vyjít
7	164	23 011, 30 707,7, 40 210	157 (95,7 %)	1, 18,7, 79	číst, přicházet, končit
8	317	10 970, 15 861,7, 22 982	304 (95,9 %)	1, 10,5, 83	dokončit, svědčit, přejít
9	818	3 551, 6 137,0, 10 966	669 (81,8 %)	1, 4,5, 29	uznávat, reprezentovat
10	20 800	0, 241,7, 3 548	1 521 (7,3 %)	0, 1,8, 79	ztroskotat, rýsovat

Tabulka 2.1: Slovesa v ČNK ve skupinách dle četnosti. Zastoupení sloves i v PDT.

		Výskyty v ČNK	PDT		
Skupina	**Sloves**	(min, ∅, max)	**Sloves**	**Výskytů**	**Příklady**
10.1	153	3 050, 3 298,9, 3 548	106 (69,3 %)	1, 2,5, 9	ztroskotat, rýsovat, vyčistit
10.2	177	2 628, 2 835,6, 3 048	105 (59,3 %)	1, 2,6, 9	sklízet, napít, probouzet
10.3	211	2 172, 2 385,6, 2 627	114 (54,0 %)	1, 2,3, 9	vynášet, usnadňovat
10.4	257	1 732, 1 950,0, 2 172	121 (47,1 %)	1, 2,0, 11	navrátit, škrtnout
10.5	328	1 367, 1 535,0, 1 732	160 (48,8 %)	1, 1,9, 10	popíjet, odmlčet, zmapovat
10.6	426	1 028, 1 177,0, 1 366	165 (38,7 %)	1, 1,7, 7	rozčílit, tyčit, rezervovat
10.7	586	712, 858,0, 1 028	163 (27,8 %)	1, 1,4, 5	nashromáždit, čarovat
10.8	900	440, 558,5, 712	180 (20,0 %)	1, 1,7, 79	mrazit, maturovat, posít
10.9	1 629	208, 308,5, 440	190 (11,7 %)	1, 1,5, 61	pošpinit, rybařit, vyfouknout
10.10	16 133	0, 31,2, 208	217 (1,3 %)	0, 1,5, 32	ohrnout, korat, vyoperovat

Tabulka 2.2: Slovesa v ČNK s méně než 3 550 výskyty ve skupinách dle četnosti. Zastoupení těchto sloves i v PDT.

Ve druhém sloupci tabulek uvádíme údaj o počtu sloves, která se do dané skupiny zařadila. Nejčetnější sloveso *být* tedy s více než jedenácti miliony výskytů samo tvoří první skupinu. Naproti tomu ve skupině 10.10, kam se zařadila slovesa s nejvýše 208 výskyty v ČNK (průměrně však jen cca 31 výskytů), je celkem 16 133 sloves.

Ve třetím sloupci tabulky je uveden minimální a maximální počet výskytů jednotlivých sloves dané skupiny. Pro orientaci je k dispozici i průměrný počet výskytů sloves v této skupině.

Další sloupce tabulek jsou věnovány zastoupení sloves jednotlivých skupin v PDT. Ve sloupci PDT-Sloves je uvedeno, kolik sloves dané skupiny bylo v PDT alespoň jednou evidováno, a tentýž údaj vyjádřený procentuelně. Jak již bylo patrné z grafu 2.1 na předchozí straně, poměrné zastoupení jednotlivých sloves je PDT přibližně stejné jako v ČNK, PDT je však několikanásobně menší. Dokládají to i údaje o minimálním, průměrném a maximálním počtu výskytů sloves dané skupiny v PDT, jak uvádíme v předposledním sloupci tabulky. Z údajů je patrné, že již slovesa začleněná do skupiny 5 a dále se v PDT vyskytují průměrně méně než padesátkrát.

Poslední sloupec tabulky pro ilustraci uvádí příklady sloves zařazených do jednotlivých skupinek. Naprostá většina příkladů je pro rodilého mluvčího dobře známa[6] a řadu sloves ani z katerogie s nejmenším počtem výskytů nelze ještě označit za "periferii" jazyka. Potřebu připravit slovník slovesných rámců tak v teoretické lingvistice i v NLP není možné omezit jen na slovesa užívaná často.

[6] Výjimkou je sloveso *korat*. Jde o chybu v morfologickém značkování, ovšem proti očekáváním nejde o výskyty tohoto slovesa ve tvaru *korán*, ale zvláštní druh koček, koratů.

2.3 Poskytují korpusy dostatek dat?

Nahlédneme-li podrobněji do aktuální verze valenčního slovníku češtiny[7], která je používána pro anotaci tektogramatické roviny PDT, zjistíme tyto údaje[8]:

- Slovník eviduje cca 4 000 sloves (tj. cca desetinu odhadovaného celkového počtu).
- Slovník potvrzuje očekávání, častá slovesa mají více valenčních rámců než slovesa méně častá.
- Sloveso *mít* je evidováno s rekordním počtem 51 slovesných rámců.
- Slovník eviduje cca 35 sloves s více než 10 rámci.
- Tři rámce jsou evidovány pro cca 250 sloves, dva rámce pro 709 sloves[9].
- Přibližně 2 300 sloves je zatím uvedeno s jediným valenčním rámcem, a nebylo ještě dostatečně zpracováno.

K pohodlnému získání uvedených statistických údajů přímo ze zdrojového souboru slovníku byl použit program *xsh*, XML shell, Petra Pajase. Úvod do xsh je k dispozici v Žabokrtský et al. (2002), podrobná dokumentace je na Internetu[10].

Jakkoli je obtížné hovořit o "průměrném počtu rámců" českého slovesa (především k velmi výrazným rozdílům mezi slovesy častými a méně častými), lze očekávat, že pro jedno sloveso bude třeba evidovat nejméně dva až čtyři valenční rámce. Přitom máme na mysli tektogramatické valenční rámce, nikoli rozdílné povrchové realizace valenčních rámců; těch bude třeba ve slovníku pokrýt několikanásobně více. Úvahu provádíme proto, abychom mohli vyjádřit toto očekávání:

Pro automatizovanou extrakci valenčních rámců českých sloves bude třeba prostudovat nejméně 50 výskytů daného slovesa. Ve světle tohoto očekávání se ukazuje, že nejen menší PDT, ale ani velmi rozsáhlý ČNK zatím neposkytují dostatek dat pro extrakci valenčních rámců všech sloves.

Pokud vyjdeme z našeho odhadu, PDT postačuje pro přibližně *dvacet až třicet* nejčetnějších sloves a ČNK pro přibližně 6 000 sloves z 22 000 evidovaných. Naproti tomu polovina registrovaných sloves není ani v ČNK zastoupena v dostatečném počtu. Pro tento zjednodušený pohled jsme navíc předpokládali, že všechy výskyty sloves v ČNK je možné k automatizované extrakci použít. Kapitola 3.2 na str. 28 přitom jasně dokládá, že schopnosti dnešních syntaktických analyzátorů postačují spíše jen na velmi jednoduché věty a těch je v korpusu řádově méně.

[7] Viz Straňáková-Lopatková and Žabokrtský (2002; Žabokrtský et al. (2002), verze z listopadu 2002.

[8] Pro účely srovnání s výskyty sloves korpusech byly ve statistice valenčního slovníku záměrně sloučeny údaje pro zvratná slovesa i jejich nezvratné protějšky. Valenční slovník jinak tyto varianty odlišuje.

[9] Je však obtížné vyslovit se o definitivnosti záznamů pro tato slovesa.

[10] `http://xsh.sourceforge.net/`

Věty z PDT lze pro extrakci povrchových rámců naopak použít jistě všechny, díky povrchové syntaktické anotaci jsou doplnění slovesa zcela jasně v korpusu uvedena. Sloves dostatečně pokrytých v PDT je však zatím bohužel velmi málo, a jak ukazují kapitoly 3.4 na straně 37 a 4 na str. 46, cesta od doplnění sloves k povrchovým či dokonce valenčním rámcům je stále ještě velmi svízelná.

2.4 Slovesa z Internetu

Vzhledem k závěrům z předešlé kapitoly je zřejmé, že pro účely extrakce rámců sloves bude třeba i ČNK selektivně rozšiřovat. Navrhli a implementovali jsme proto jednoduchý systém pro získávání dalších příkladů použití sloves z nejbohatšího elektronického zdroje dat, celého Internetu.

Systém pro dané lemma slovesa nejprve vytvoří všechny morfologicky přípustné formy slovesa a tyto formy postupně nechá hledat některým z internetových vyhledávačů[11]. Systém pak projde všechny doporučené odkazy a příslušné stránky stáhne.

Získané texty jsou (nejde-li již o prostý text) očištěny od značek jazyka HTML a rozděleny do vět[12].

Ze získaných vět jsou vybrány jen ty, kde se vyskytuje sledované sloveso v libovolné z přípustných forem. Duplicitní věty jsou navíc odstraněny. Získané věty by v další fázi bylo možné morfologicky anotovat a případně zjednoznačnit některým z dostupných taggerů. Systém morfologické analýzy umožňuje rozebrat i text, který byl v Internetu nalezen bez diakritických znamének, víceznačnost forem slov tím však přirozeně stoupá.

Pokusně byl systém použit pro sloveso *bacit*, které se v ČNK vyskytuje v 207 ukázkách. Toto sloveso se podle morfologického slovníku dr. Hajiče může vyskytovat ve 106 různých formách (včetně negací, archaismů ap., např. *nebacímť*), z toho pro 32 forem byl vyhledávačem nalezen alespoň jeden odkaz. Celkem vyhledávač doporučil 1 854 unikátních URL. (V případě šesti forem vrátil vyhledávač vždy právě 300 odkazů, což bude pravděpodobně nějaký interní limit vyhledávače.)

Při stahování 1 854 doporučených URL (proces trval cca hodinu) se 21 stránek nepodařilo získat. V těchto textech bylo identifikováno 1574 unikátních vět s některou z forem slovesa *bacit*. Při ruční kontrole se ukázalo, že část vět (především s formami *bac* a *bacil*) není vhodným příkladem. Filtrací těchto vět (z vět obsahujících tvar začínající řetězcem *bacil* jsme ponechaly jen ty, kde byl tento tvar doprovázen vhodným tvarem slovesa *být*) jsme získali cca 490 nadějných vět, tj. zhruba dvojnásobek počtu výskytů v ČNK.

[11] Konkrétně byl použit český vyhledávač `http://www.jyxo.cz/`.

[12] K tomu byl připraven jednoduchý program v jazyce Perl. Program při dělení textu na věty využívá seznamu zkratek, který pro účely této práce poskytnut společně s analogickým nástrojem *bcpmark* dr. Hajičem.

2.5 Shrnutí

V této kapitole jsme se zaměřili na rozbor zdrojů dat vhodných pro automatizovanou extrakci rámců sloves. Ukázali jsme, že PDT je dobrým zdrojem pro přibližně 20 nejčetnějších sloves. ČNK bude možné použít pro nejvýše 6 000 nejčetnějších sloves. Přitom celkový počet sloves zastoupených v ČNK přesahuje 22 000 a v češtině bude celkem možná i dvakrát více různých sloves.

Kapitola tedy dokládá, že pro účely získání rámců sloves bude nutné použít i další zdroje dat, a představuje jednoduchý systém získání příkladů použití sloves z českého Internetu.

Zjištěné výsledky nejsou pro korpusové lingvisty překvapením. K charakteristickým rysům jazykových jevů patří i výrazně neuniformní a velmi "hladké" rozdělení pravděpodobnosti výskytu. Bylo by naopak překvapením najít v kvantitativním popisu jazykového jevu nějaký ostrý zlom.

Při extrakci lexikálně-syntaktických údajů na základě korpusových dat je třeba s tímto problémem počítat a v každé konkrétní situaci prověřit, pro které lexikální jednotky je vstupních pozorování k dispozici ještě dostatek.

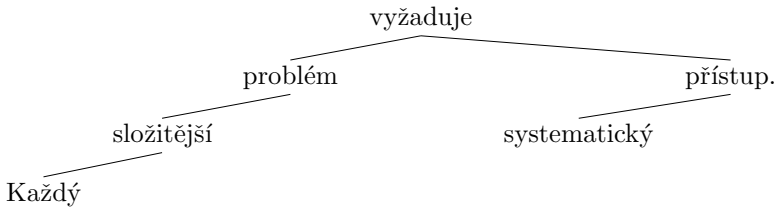
Kapitola 3

Extrakce povrchových rámců slovesa

Povrchovým rámcem slovesa rozumíme seznam typických forem synů slovesa v povrchovém syntaktickém zápisu věty.

Například ve větě v příkladu 1 spatřujeme náznak toho, že jeden z povrchových rámců slovesa *vyžadovat* bude dvojčlenný: <*kdo*> *vyžaduje* <*co*>.

(1)



(S/J/1998/150:001-p5s5)

Jak však víme z kapitoly 2.3 na str. 19, syntakticky rozebraných vět, jako byl předchozí příklad, zatím však v současné době bohužel není dostatek. Navíc je většina vět dostupných v korpusech výrazně komplikovanějších. V obrázku 3.1 na následující straně uvádíme pro srovnání několik příkladů vět z ČNK. Z příkladu je rozdíl v náročnosti úlohy extrahovat slovesné rámce ze syntakticky rozebraných a nerozebraných vět velmi zřetelný. U autentických vět čelíme jednak celkově složitější větné stavbě, a dále pak četným volným doplněním slovesa, tj. zhruba řečeno takovým doplněním sloves, která nejsou pro sloveso nijak typická, a není je proto vhodné uvádět ve slovníku.

Postup získání povrchových rámců sloves lze tedy rozdělit na dva kroky: identifikace všech doplnění slovesa (říkejme též *pozorovaných rámců*) a zpracování pozorovaných rámců tak, abychom identifikovali doplnění, která jsou pro sloveso typická. Seznam typických doplnění slovesa nazývejme *povrchovým rámcem*.

Jak již bylo uvedeno výše, zisk pozorovaných rámců z PDT, kde jsou větám jednoznačně přiřazeny syntaktické rozbory, je triviálním úkolem (stačí vypsat syny slovesa), PDT však svým rozsahem pro tento účel zdaleka nepostačuje

(S/J/1994/abc94:377-p23s1) Pro osoby, jejichž zdravotní stav vyžaduje důkladného chlazení těla, tedy pro osoby trpící vysokou horečkou nebo které utrpěly výron některého kloubu, místo dosavadních ledových obkladů, které vyžadují stálou obsluhu, sestrojil doktor M. Dumontpalliér z Nemocnice de la Pitié v Paříži aparát, který takové ošetření usnadňuje samotnému pacientu. (S/J/1994/abc94:388-p9s2) Přestože vyžaduje dobré znalosti o pěstování, soustavnou péči a studený skleník, krása této rostliny nám zvýšené úsilí určitě vynahradí. (S/J/1994/abc94:388-p30s2) *V letních a zimních měsících* vyžaduje relativně suché prostředí a dobře propustnou půdu, ale *na jaře před dobou květu a během ní* vyžaduje velké množství srážek, takže se doporučuje v jarním období tyto kosatce ještě dostatečně zalévat. (S/J/1994/abc94:332-p6s5) Vyžadují *denně* dostatek čisté pitné vody (ve které se s oblibou i koupou). (S/J/1994/abc94:176-p13s3) Tato teplota *však* vyžaduje stále odolnější žáruvzdorné a pevnější materiály, z nich jsou vyrobeny lopatky turbinových kol. (S/J/1998/150:011-p9s4) Nové předpisy vyžadují *mj. též* povinnost určit u každého kvantitativního výsledku měření jeho nejistotu. (S/J/1994/abc94:010-p7s2) *Bezpodmínečně* vyžadují volný výběh, a pokud nemáme velkou voliéru, budeme je muset pouštět k proběhnutí v bytě.

Obrázek 3.1: Ukázka zdrojových neanotovaných vět z ČNK pro extrakci slovesných rámců. V ukázce jsou podtržena ta rozvití slovesa, která by měl systém extrakce v textu identifikovat jako typická, *kurzíva* označuje volná rozvití slovesa.

(viz 2.3 na str. 19). První část této kapitoly proto podrobně rozpracovává postup, jak extrahovat povrchové rámce z vět, které nejsou syntakticky rozebrány, ale jsou anotovány pouze na morfologické úrovni.

Při snaze o extrakci syntaktické informace z vět, pro něž tento rozbor není dostupný, se nabízí několik možností; v jádru se však nijak výrazně neodlišují, neboť hrubý odhad syntaktické struktury věty musíme udělat stejně.

Nejprostší aproximací syntaktických vazeb slovesa je kolokace: "Vyskytla se v okolí pěti slov studovaného slovesa často předložka *na* následovaná jménem ve čtvrtém pádě?". Sofistikovanější postup představuje částečná syntaktická analýza věty – identifikujeme jmenné skupiny, jednotlivé klauze (tj. jednotlivé věty souvětí), části sloves ap., a test kolokace budeme provádět na nich. Dospějeme tak nepochybně k lepším výsledkům.

Poslední možností je nechat některému ze stávajících syntaktických analyzátorů provést úplný rozbor vstupní věty a klasickým způsobem pak vypsat syny sloves. Úspěch totoho postupu závisí především na kvalitě dostupných parserů s ohledem na složitost vět, které jim předložíme.

V naší práci jsme zvolili poslední jmenovanou možnost a to z několika důvodů: pokud by se analyzátory ukázaly obecně uspokojivě kvalitní, bude možné jejich výstup přímo použít i k extrakci jiných typů lexikálně-syntaktických údajů. Druhým důvodem je fakt, že koneckonců základní snahou v této rovině popisu jazyka je vytvořit spolehlivý syntaktický analyzátor. Z hlediska tohoto

obecnějšího pohledu je tedy zajímavé dostupné parsery vyzkoušet v praxi, navzájem porovnat. Výhledově by pak bylo jistě zajímavé odhalit jejich silnější i slabší stránky, tak, aby je bylo možné postupně vylepšovat.

S ohledem na současné poznatky jsme očekávali poměrně nízkou úspěšnost parserů na všech větách přirozeného jazyka. V této práci jsme se proto soustředili na filtraci vstupních vět tak, abychom parserům nedávali příliš složitý vstup a aby parsery pracovaly úspěšněji. Lingvistické těžiště této práce tedy představuje právě systém filtrace dostupných vět.

V první části této kapitoly stručně představíme tři odlišné parsery pro češtinu a provedeme srovnání jejich úspěšnosti z hlediska cíle této práce. Druhá část kapitoly, 3.2 na straně 28, podrobně popisuje lingvisticky motivovanou filtraci vstupních vět, tak aby parsery pracovaly úspěšněji. Závěrem této kapitoly je tedy učiněn odhad, nakolik jednodušší vstupní věty současným parserům pomohou.

Závěrečná část kapitoly, 3.4 na str. 37, se pak vrací k cíli této práce a studuje možnosti zpracování získaných pozorovaných rámců slovesa tak, abychom identifikovali typická doplnění slovesa.

3.1 Současné syntaktické analyzátory

Z teoretického hlediska je třeba mít na paměti rozdíl mezi pojmy *syntaktický analyzátor* a *parser*.

Pojem *syntaktický analyzátor* typicky označuje zařízení, které pro vstupní větu přirozeného jazyka anotovanou na morfologické úrovni provede syntaktickou analýzu, tj. připraví seznam všech možností syntaktického rozboru dané věty. Pojem *parser* pochází původně z teorie formálních gramatik a označuje zařízení, které pro danou vstupní větu a danou formální gramatiku rozhodne, zda je věta správnou větou jazyka definovaného danou gramatikou. Syntaktický analyzátor je v tomto smyslu také parserem: pokud pro danou větu nenajde žádnou možnost, jak větu syntakticky rozebrat, není věta korektní větou jazyka, pokud alespoň jednu možnost najde, je věta větou jazyka.

Situace zpracování textů přirozeného jazyka je však ve srovnání s formálními jazyky významně složitější. Pro přirozené jazyky se dosud nepodařilo vybudovat žádnou formální gramatiku, a i pro rodilé mluvčí často hranice mezi syntakticky správnou a nesprávnou větou jazyka není ostrá. Z těchto důvodů se dosud nepodařilo žádný *parser* v původním slova smyslu vytvořit.

Ve snaze zpracovat syntax vět přirozeného jazyka však byla již (pro různé jazyky) připravena celá řada zařízení, která provádějí tzv. *robustní syntaktickou analýzu*[1]. Robustnost metody spočívá v tom, že zařízení vytvoří syntaktickou strukturu i pro věty, které nejsou správnými větami jazyka.

Termín *parser* vykonal od svého zavedení v teorii formálních jazyků okružní cestu přes NLP zpět ke komputační lingvistice. V současné době tedy ztratil původní jednoznačný význam a často nyní označuje robustní zařízení, které pro danou vstupní větu jazyka připraví jednu nejpravděpodobnější variantu syntaktického rozboru. Takové zařízení není tedy v původním smyslu

[1] Viz např. Kuboň (2001).

ani *parserem* (je robustní, zpracuje i nesprávnou větu), ani (robustním) *syntaktickým analyzátorem* (vrací jediné řešení, nikoli všechny přípustné varianty).

V této práci jsme se ze stylistických důvodů rozhodli používat oba pojmy zaměnitelně. Toto zkreslení si můžeme dovolit především díky tomu, že s ohledem na cíl práce není tento rozdíl významný. Parsery či syntaktické analyzátory představují pouze jeden z nástrojů používaných na cestě extrakce lexikálně-syntaktických údajů z korpusových dat a bez ohledu na uvedený rozdíl by tomuto účelu posloužily prakticky stejně.

3.1.1 Zemanův parser

Parser Dana Zemana byl vytvořen v rámci diplomové práce (Zeman, 1997) v roce 1997. Analyzátor založen na statistických metodách opírajících se o myšlenku závislostní syntaxe (hledá nejpravděpodobnější zapojení slov do syntaktických stromů, vychází přitom pouze ze zjednodušených morfologických značek). Předpokládá, že vstupní data jsou již morfologicky anotována a zjednoznačněna. Parser vrací (jednu) nejpravděpodobnější možnost syntaktické analýzy věty. Natrénován byl na datech PDT.

Později byl ryze statistickému parseru předřazen pravidlový systém popisující pomocí regulárních výrazů strukturu jmenných skupin (viz Zeman (2001b; Zeman (2001a)). Gramatická pravidla pro jmenné skupiny jsou totiž až na řídké výjimky poměrně dobře formulovatelná, a statistické části se tak snadno zabrání dělat "hloupé chyby".

V dalším vylepšení (viz Zeman (2002)) byl parser obohacen o subkategorizační rámce a rovněž byl statistický model natrénován s tzv. *selektivní lexikalizací*, tzn. že pro určité slovní druhy (předložky ad.) model uvažuje nejen morfologickou značku, ale i lemma daného slova. Díky těmto vylepšením se přesnost parseru podle Zemana (2002) zlepšila z 77,0 % na 78,7 % správně zapojených uzlů.

Pro srovnání v této práci byla použita aktuální verze, která nezahrnuje gramatická pravidla ani subkategorizační rámce, ale používá selektivní lexikalizaci.

Parser je implementován v jazyce Perl.

3.1.2 Collinsův parser

Parser Michaela Collinse (1996) je statistický parser původně určený pro angličtinu. Z toho důvodu se parser snaží vybudovat pro vstupní větu bezprostředně složkový strom. Statistický model parseru však interně pracuje s pojmem závislosti mezi dvojicemi slov ve větě. Kritérium pravděpodobnosti samotné závislosti je však vhodně doplněno kritériem vzdálenosti mezi slovy, výskytem speciálních symbolů mezi slovy (interpunkce) ad. Díky tomu, že je model z lingvistického hlediska dosti adekvátní, dosahuje parser pro angličtinu vynikajících výsledků.[2]

[2] Collins (1996) uvádí pro věty do 40 slov *pokrytí (recall)* 85,8 % správně vytvořených složek a *přesnost (precission)* 86,3 %, tj. poměr složek, které parser vytvořil správně, k celkovému počtu vytvořených složek. V Collins (1997) je model parseru obohacen o subkategorizaci a

Parser očekává na vstupu věty jednoznačně anotované na morfologické rovině popisu. Parser vrací (jeden) nejpravděpodobnější syntaktický rozbor věty.

Pro češtinu byl parser přizpůsoben v roce 1999, viz Collins et al. (1999). Úprava spočívala jednak v modifikaci vstupních morfologických značek, a především pak v překladu složkových stromů věty na stromy závislostní (pro trénování i v závěru parsingu). Pro češtinu uvádí Collins et al. (1999) úspěšnost průměrně 80,0 % správně zapojených a srovnává tento výsledek s poslední verzí parseru pro angličtinu, která dosahuje 91 % správně identifikovaných hlav složek, tj. de facto správně zapojených uzlů v závislostním zápisu věty.

Stručný návod k použití a popis aktuální práce je k dispozici na Internetu[3]. Je však třeba poznamenat, že tento dosud nejlepší dostupný parser pro češtinu je v současné době z čistě technických důvodů obtížné plně zprovoznit a v některých případech se nepodaří vstup analyzovat vůbec. Z tohoto důvodu v uvedených srovnáních parserů vždy uvádíme přesný počet vět, které se jednotlivým systémům podařilo zpracovat.

3.1.3 Parser Zdeňka Žabokrtského

Syntaktický analyzátor Zdeňka Žabokrtského byl vytvořen v průběhu tří týdnů v jazyce Perl a dosud nebyl publikován. Analyzátor je ryze pravidlový a současná gramatika obsahuje cca šedesát ručně vytvořených pravidel. Pravidla určují, jak má být které dosud nepoužité vstupní slovo přivěšeno do stávajících podstromů. Aplikace pravidel je zcela deterministická a jejich pořadí je pevně dáno. Po aplikaci pravidla se parser již nikdy nevrací zpět (žádný backtracking), i když gramatika může obsahovat a obsahuje i opravná pravidla, která přemisťují ve stromové struktuře i uzly zavěšené dříve.

Parser předpokládá, že vstupní věta je morfologicky zjednoznačněna.

3.1.4 Srovnání úspěšnosti parserů

Popsané syntaktické analyzátory v této kapitole srovnáme podle několika kritérií úspěšnosti. Ve všech případech byly analyzátory spuštěny na morfologicky jednoznačně anotované věty z evaluační části PDT.

Vzhledem k tomu, že statistické parsery vždy pro vstupní větu vracejí právě jedno řešení, je možné jejich úspěšnost měřit prostým počtem chyb. Stejně je tomu i pro pravidlový parser Zdeňka Žabokrtského, který obsahuje záchytná pravidla a v každém případě rovněž vrátí právě jeden výstupní strom.

Úspěšnost podle počtu uzlů a s ohledem na délku vstupní věty

Nejčastějším kritériem pro srovnání úspěšnosti parserů do závislostní reprezentace povrchové syntaxe věty je počet správně syntakticky zapojených slov věty.

tzv. *wh-movement* a dosahuje pokrytí 87,5 % a přesnosti 88,1 %.

[3] `http://ckl.mff.cuni.cz/~honet/collins`

Tabulka 3.1 toto kritérium uvádí jednak souhrnně, a jednak jednotlivě pro věty do délky dvaceti slov.

Z hlediska praktické použitelnosti pro další aplikace (nikoli např. jako předzpracování, které pouze šetří práci anotátorům PDT, ale např. pro strojový překlad), je významné též jiné kritérium, a sice počet zcela správně analyzovaných vět. Není překvapením, že toto kritérium hodnotí všechny analyzátory podstatně méně příznivě než procento správně zapojených uzlů. Průměrná česká věta (textu z korpusu) má cca 15 slov, tím se vysvětluje velká pravděpodobnost, že ve větě analyzátory udělají alespoň jednu chybu. Tento údaj již neuvádíme jednotlivě pro věty různých délek.

		Collins	**Zeman**	**Žabokrtský**
Celkem vět		7 316	7 319	7 319
Celkem uzlů		125 937	126 030	126 030
Úspěšnost podle počtu uzlů		82,5 %	69,2 %	73,8 %
Správně analyzovaných vět		30,9 %	15,0 %	18,4 %
		Úspěšnost podle počtu uzlů		
Délka věty	**Vět**	**Collins**	**Zeman**	**Žabokrtský**
1	55	100,0 %	100,0 %	100,0 %
2	199	99,5 %	95,5 %	94,5 %
3	151	92,3 %	83,9 %	84,1 %
4	209	93,8 %	83,5 %	84,0 %
5	274	89,8 %	68,5 %	79,9 %
6	220	88,3 %	82,7 %	79,3 %
7	276	87,3 %	78,4 %	80,5 %
8	248	87,2 %	81,2 %	82,9 %
9	267	88,2 %	78,2 %	82,5 %
10	313	87,6 %	78,3 %	81,3 %
11	282	87,6 %	76,6 %	80,0 %
12	289	86,8 %	75,9 %	79,5 %
13	289	86,1 %	75,0 %	79,5 %
14	297	85,1 %	73,1 %	77,4 %
15	280	85,4 %	72,6 %	78,2 %
16	262	85,1 %	69,8 %	75,9 %
17	273	83,6 %	71,9 %	76,4 %
18	275	84,0 %	70,2 %	76,4 %
19	249	85,1 %	69,8 %	75,4 %
20	231	84,3 %	69,2 %	73,3 %

Tabulka 3.1: Úspěšnost parserů podle počtu správně zapojených uzlů a s ohledem na délku vstupní věty.

Z uvedených měření je zřejmé, že parser Michaela Collinse je v současnosti stále zdaleka nejlepším parserem pro češtinu. Velmi zajímavé je však srovnání parserů Zdeňka Žabokrtského a Dana Zemana. Ukazuje se, že oba parsery, ač jsou založeny na zcela odlišných přístupech, dosahují velmi podobných přesností, a rozdíl v přesnostech je zcela v rámci statistické chyby.

Není překvapením, že Collinsův parser dosahuje téměř stoprocentní přesnosti

na větách o dvou slovech. Jako jediný ze srovnávaných parserů totiž zohledňuje při konstrukci hrany konkrétní lexikální jednotky vždy, když danou dvojici slov měl možnost zaznamenat v trénovacích datech. K odhadu závislosti na základě morfologické značky přistupuje jen v případě nedostatku znalosti o daných konkrétních slovech.

Zajímavé by jistě bylo zjistit, čím jsou věty o pěti resp. šesti slovech typicky obtížnější než věty o několik málo slov delší, neboť tento rozdíl se dotkl úspěšnosti všech parserů. (U Collinse k této odchylce došlo až pro věty o 7 a 8 slovech, věty o několik slov delší zpracovává úspěšněji. Rozdíl v úspěšnosti však již není tak významný.)

Jak ukazujeme v kapitole 3.3 na straně 35, je však též velmi zajímavé srovnání úspěšnosti parserů na větách různé složitosti z jazykového hlediska.

Úspěšnost podle synů sloves

Pro účely naší práce je podstatnější jiné kritérium než celkový počet správně zapojených uzlů, a sice procento správně identifikovaných pozorovaných rámců slovesa, viz tabulka 3.2 na následující straně. Jako správně identifikovaný je označen takový výskyt slovesa, u něhož množina (bezprostředních) synů neobsahuje žádné uzly navíc ani nepostrádá žádné uzly proti správné syntaktické analýze věty. Celek je přirozeně celkový počet výskytů sloves. Pro výpočet tohoto údaje bylo ovšem odhlédnuto od výskytů slovesa být.

Nejlepšího výsledku podle správně identifikovaných pozorovaných rámců dosáhl opět Collins. Správně připraví rámec pro více než polovinu výskytů sloves, a je tak o 15 až 20 % lepší než ostatní parsery. Pro věty délek do 10 uzlů pak úspěšnost Collinsova parseru v počtu správně pozorovaných rámců přesahuje 60 %.

Z hlediska extrakce povrchových slovesných rámců je zajímavé též kritérium, kdy parser žádné syny sloves *nepřehlédne*. Pokud by následná fáze zpracování pozorovaných rámců na povrchové rámce (viz 3.4 na straně 37) byla dostatečně kvalitní, bylo by možné i na základě nesprávně bohatých pozorovaných rámců získat rámce povrchové.

Úspěšnost parserů podle kritéria sloves, u nichž žádný ze synů *nechybí* (ale může přebývat), uvádíme v tabulce 3.3 na str. 30 již pouze souhrnně bez ohledu na délku vět. Překvapivě je tentokrát vítězem parser Zdeňka Žabokrtského, o procento lepší než Collins. Zemanův parser si zachovává odstup 20 %. Toto zjištění lze vysvětlit tak, že záchytné pravidlo "na předchozí sloveso" v parseru Zdeňka Žabokrtského je jako nouzové řešení velmi kvalitní. Statistické parsery dají pravděpodobně přednost svým "málo podloženým odhadům", a dopustí se tak většího množství chyb.

3.2 Složitost vět

Jak ukázala předešlá kapitola, předložíme-li současným syntaktickým analyzátorům libovolné vstupní věty, které najdeme v korpusu, nemůžeme s ohledem na

			Collins	Zeman	Žabokrtský
Celkem vět			7 316	7 319	7 319
Celkem výskytů sloves			16 311	16 329	16 329
Sloves se správnými syny			55,3 %	33,1 %	39,5 %
		Sloves	**Sloves se správnými syny**		
Délka věty	**Vět**	**ve větě**	**Collins**	**Zeman**	**Žabokrtský**
1	55	0,0	-	-	-
2	199	0,0	100,0 %	100,0 %	100,0 %
3	151	0,2	80,0 %	80,0 %	74,3 %
4	209	0,6	91,9 %	74,0 %	80,5 %
5	274	0,6	75,3 %	52,6 %	65,6 %
6	220	0,9	72,0 %	59,6 %	58,0 %
7	276	1,1	64,5 %	46,6 %	52,8 %
8	248	1,2	63,9 %	44,0 %	49,0 %
9	267	1,4	65,1 %	41,4 %	47,8 %
10	313	1,5	63,1 %	41,2 %	45,8 %
11	282	1,6	59,6 %	38,9 %	44,4 %
12	289	1,7	60,1 %	38,9 %	44,4 %
13	289	1,9	59,1 %	37,0 %	45,7 %
14	297	1,9	55,1 %	34,5 %	40,8 %
15	280	2,1	57,7 %	33,8 %	43,3 %
16	262	2,1	59,3 %	31,1 %	41,5 %
17	273	2,4	55,4 %	34,4 %	41,7 %
18	275	2,4	56,5 %	31,9 %	39,7 %
19	249	2,5	59,3 %	34,6 %	40,7 %
20	231	2,7	55,7 %	32,5 %	41,4 %

Tabulka 3.2: Úspěšnost parserů podle správných pozorovaných rámců slovesa a s ohledem na délku vstupní věty.

značnou složitost českých vět očekávat příliš přesné výsledky.

V této kapitole proto popíšeme jednoduchá kritéria, která umožní ze vstupních dat vybrat jednodušší věty tak, aby je současné syntaktické analyzátory dokázaly zpracovat dostatečně kvalitně.

Současně odhadneme, jaké procento vět z korpusových dat do této kategorie zpracovatelných vět ještě zapadne a nakonec srovnáme zlepšení úspěšnosti pro jednotlivé parsery.

Samotná pravidla identifikující dostatečně jednoduché věty byla implementována v systému AX, který byl vytvořen pro účely této práce a podrobně se jím zabýváme v kapitole 5 na straně 53. Detailní výpis pravidel použitých pro filtraci je pak uveden v příloze A na str. 84.

Úvodem vzpomeňme ještě jeden demotivující příklad věty z korpusu, která je sice dostatečně jednoduchá, ale která nás při hledání rámců sloves přivede na špatnou stopu.

(2) V sirkové hře mnozí nevěřili, že se dá vyhrát.(S/J/1994/abc94:008-p15s3)

	Collins	Zeman	Žabokrtský
Sloves se správnými syny	55,3 %	33,1 %	39,5 %
Sloves, jimž nechybí syn	73,7 %	55,9 %	74,7 %
Rozdíl	18,47 %	22,87 %	35,27 %

Tabulka 3.3: Úspěšnost parserů podle pozorovaných rámců slovesa, kde nechybí žádné doplnění.

Větě lze naštěstí vytknout nejen stylistickou neobratnost, ale i gramatickou chybu (chybí čárka před slovem *mnozí*). Tváří v tvář takovým příkladům nezbývá než doufat, že nesprávné věty neovlivní výsledky příliš. V kapitole 3.4 na straně 37 ukážeme, že i věty zcela správné přinášejí dostatek problémů.

3.2.1 Složitá interpunkce, číslice ad.

Jako první vyloučíme z dalšího zpracování věty obsahující interpunkční znaménka s obtížným syntaktickým výkladem. Jedná se například o pomlčku, dvojtečku, liché závorky a uvozovky, lomítka, středníky a další symboly. Podobně též zamítneme věty obsahující číslice.

Důvodem k zamítnutí těchto vět je zkušenost, že současné syntaktické analyzátory, především jsou-li založeny na ručně psaných pravidlech, nejsou dostatečně kvalitně připraveny na tyto "okrajové" jevy. U statistických analyzátorů by bylo třeba toto tvrzení ještě experimentálně ověřit. V každém případě je však jisté, že správné řešení syntaktické struktury v těsném okolí atypické interpunkce našemu úkolu extrahovat povrchové rámce sloves nijak nepomáhá. U číslic je zase problém identifikovat správně morfologické kategorie pádu a rodu, výslednou analýzu získaných rámců by tedy jen komplikovaly.

3.2.2 Složení analytických slovesných tvarů

Pro další fáze rozhodování, zda větu přijmout k dalšímu zpracování nebo ne, je nutné pokusit se složit analytické slovesné tvary. V obecném případě syntaktického rozboru věty je tento úkol obtížný, neboť mezi jednotlivými částmi složeného slovesného tvaru se může vyskytnou např. vsuvka nebo celá vložená věta:

(3) Včera jsem, a moc rád bych se za to omluvil, dočista zapomněl na naši schůzku.

Vložené klauze však samy o sobě představují složitý problém a v dalším zpracování věty s vloženou klauzí pro jistotu v každém případě zamítneme (viz níže). Už při skládání složených slovesných tvarů tedy můžeme pravidla formulovat tak, aby nedokázala dohromady složit části složeného slovesa, jsou-li odděleny vloženou klauzí.

V této fázi jsou též sloučena modální slovesa jako *muset*, *moci* ad. s plnovýznamovými slovesy. Údaj o modalitě je k hlavnímu slovesu připojen

pouze jako příznak.

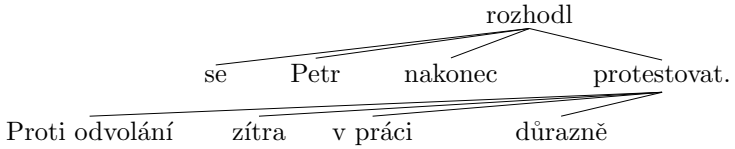
3.2.3 Věty s více plnovýznamovými slovesy

Je významné uvědomit si, že z hlediska úkolu extrakce povrchových rámců sloves jsou v případě češtiny jako zdrojová data naprosto nevhodné všechny klauze, které obsahují více než jediné plnovýznamové sloveso. Relativní volnost českého pořádku slov totiž dovoluje doplnění jednotlivých sloves téže klauze prakticky libovolně promíchat. Pro správnou identifikaci, které určení patří ke kterému slovesu, by však bylo nutné opřít se často nejen o syntaktická kritéria (slovesné rámce, jež se však právě pokoušíme získat), ale i o kritéria sémantická.

Jako příklad 4 uveďme mírně upravenou větu z práce Holan et al. (2001). Je třeba zdůraznit, že volná doplnění ve větě (*zítra*, *v práci*, *nakonec* a *důrazně*) je nepochybně možné zapojit na základě ryze syntaktických kritérií k libovolnému ze sloves. Doplnění *proti odvolání* lze k oběma slovesům připojit rovněž; preferované čtení závisí již na lexikální hodnotě v tomto doplnění (srov. s *proti očekávání*). Naše analýza tedy v tomto směru uvádí čtení preferované i na základě sémantických podmínek.

Zapojení větných členů *se* a *Petr* je však jednoznačné již na základě výhradně syntaktických kritérií.

(4)



Do dalšího zpracování tedy nevpustíme věty, které obsahují dvě slovesa, mezi nimiž není žádná čárka.[4]

3.2.4 Koordinace ve jmenných či předložkových skupinách

Koordinace ve větách přirozeného jazyka je velmi složitý jev a týká se samozřejmě nejen jmenných a předložkových skupin. I na úrovni větně členských koordinací je možné do spojení dávat různé a odlišné typy členů. Některé větné členy pak mohou rozvíjet celou koordinaci najednou, některé mohou rozvíjet jen její členy. V závislostním rozboru při analýze věty s koordinací tak často nevystačíme se stromovou strukturou a musíme zavést nějaká rozšíření.[5]

[4] Zatím nám mohou "proklouznout" věty, které obsahují dvě slovesa v jedné klauzi, pokud se mezi slovesy vyskytuje např. koordinace obsahující čárku. Proto tento filtr provedeme ještě jednou po zpracování koordinací. Stejného pochybení bychom se dopustili, pokud by ve větě byla vsuvka. Vsuvkami se však již naše gramatika nezabývá vůbec. Věty se vsuvkami jsou zamítnuty na samém konci zpracování, v rámci filtru vět s "nevysvětlenými" čárkami, tj. čárkami, jež nebyly identifikovány jako oddělovač v koordinaci ani jako hranice klauzí.

[5] Výstižně koordinaci pro němčinu charakterizuje již Kunze (1972), když říká: Jsou-li P, Q, R úseky rozdělující nějakou správnou větu jazyka a jsou-li P, Q', R úseky rozdělující jinou správnou větu jazyka (věty se liší jen v Q a Q'; P či R mohou být případně prázdné), je i věta tvaru P, Q *a* Q' R správnou větou jazyka. Příklady lze uvést i v češtině:

Z naznačených důvodů je zřejmé, že provést řádně syntaktickou analýzu věty s koordinací může být velmi obtížné. Syntaktické analyzátory se v koordinaci s nejvyšší pravděpodobností dopustí nějaké chyby.

Do dalšího zpracování tedy povolíme pouze "prototypické" koordinace, které nastávají ve jmenných a předložkových skupinách. Postupně náš systém pravidel zredukuje:

- příslovce v koordinaci na jediného zástupce,
- příslovce k následujícímu přídavnému jménu,
- přídavná jména v koordinaci na jediného zástupce
- přídavná jména k následujícími syntaktickému jménu[6],
- syntaktická jména v koordinaci na jediného zástupce,
- předložku k následujícímu syntaktickému jménu.

Této redukci nevyhoví jakékoli složitější formy koordinace (např. *na dřevěný stůl, starou židli i pod velkou skříň*) a záměrně též koordinace dvou předložkových skupin. Zpracování dvou koordinovaných předložkových skupin s odlišnými předložkami by bylo výrazně komplikovanější, i když by koordinace na druhou stranu přinesla poměrně spolehlivou informaci o tom, že funkci daného doplnění lze ekvivalentně vyjádřit dvěma odlišnými formami.

Na závěr jsou proto zamítnuty všechny věty, v nichž zůstala "nevysvětlená" koordinační spojka nebo čárka.[7]

Při tomto postupu jsme mimochodem místo každé (nerekurzivní, ve smyslu genitivních přívlastků) jmenné skupiny získali pouze jeden uzel věty[8]. V závěru fáze ještě připojíme celý řetízek genitivních přívlastků k předcházející jmenné či předložkové skupině. Později tak můžeme zvažovat, zda riziko syntaktické víceznačnosti v zapojení jmenných skupin mezi sebou či ke slovesu nepřekračuje rozumnou mez.

(5) a. Petr koupil knihu Martinovi.

b. Petr daroval knihu Martinovi.

c. Petr koupil a daroval knihu Martinovi.

Větě je třeba rozumět tak, že kniha byla koupena i darována, v syntaktickém rozboru věty by tedy slovo *kniha* mělo záviset současně na obou slovesech. To samozřejmě překračuje možnosti zápisu syntaktické struktury ve formě stromu.

[6]Syntaktickým jménem rozumíme takové slovní druhy, které mají podobné syntaktické vlastnosti jako podstatná jména. Jedná se především o některé druhy zájmen a o "zpodstatnělá" přídavná jména.

[7]Zamítnutí na základě nevysvětlené čárky je samozřejmě nutné odložit až za identifikaci klauzí, viz dále.

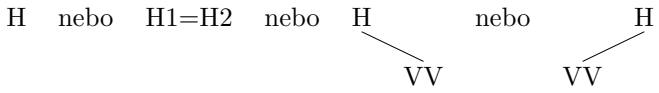
[8]Náš analyzátor pro jednoduchost v této fázi pracuje deterministicky (byť to systém AX nevyžaduje) a jako jmennou skupinu označí nejdelší přijatelný úsek věty. Je samozřejmě možné najít příklad věty, kde se dopustíme chyby, nejde však o častý případ:

(6) V okolních lesích roste tajuplná rostlina jílek mámivý, lidově <zvaná mátonoha> nebo opilka. (ln95040:062-p5s9)

3.2.5 Jednoduchá větná skladba

V této fázi je již možné odhadnout hranice klauzí, neboť analytické slovesné tvary byly již sloučeny. Pokud je však struktura klauzí v souvětí složitější, stává se odhad především konce klauze velmi obtížným úkolem. Čárka na konci klauze totiž může i nemusí hrát roli koncové čárky ještě pro nadřazenou či předcházející klauzi.

V našem jednoduchém přístupu proto pro další zpracování ponecháváme pouze věty se strukturou (H značí větu hlavní, VV značí větu vedlejší):



Ve větě hlavní i větě vedlejší je vyžadováno právě jedno sloveso (viz výše) a hranice mezi větami musí být správně vyznačena čárkou nebo spojkou *a*, *i*, *ani*, *nebo*. V případě H\VV je za čárkou oddělující klauze navíc vyžadována podřadicí spojka nebo vztažné zájmeno. V případě VV/H je spojka či zájmeno vyžadováno na začátku celé věty.

3.2.6 Problémy v důsledku homonymie předložkových skupin

Syntaktickou víceznačnost při zapojení předložkových skupin podrobně rozebírá Straňáková-Lopatková (2001). Práce zaměřená na jasná lingvisticky motivovaná kritéria řešící syntaktickou víceznačnost předložkových skupin a použitelná v algoritmech automatické syntaktické analýzy mj. formuluje "základní podezřelé", "základní jednoznačné" a "základní preferované" slovosledné vzorce (word order patterns, WOP). Při řešení identifikovaných potenciálně víceznačných konstrukcí práce především předpokládá existenci valenčního slovníku sloves i dalších slovních druhů, dále pak formuluje kritéria (především "princip separace"), která zamítnou nepřípustné syntaktické konstrukce ve větě, kde již určité závislosti spolehlivě identifikovány byly.

Je zřejmé, že v našem případě se nemůžeme opřít o existenci valenčního slovníku. "Základní podezřelé" slovosledné vzorce však můžeme velmi snadno prověřovat při filtraci vět a věty, u nichž syntaktická víceznačnost zcela jistě ohrožuje správnou identifikaci doplnění slovesa, nepřipustit k dalšímu zpracování. Jde o tyto slovosledné vzorce:

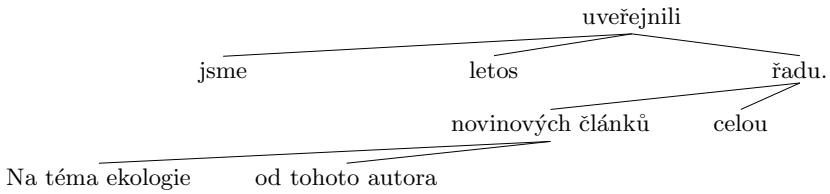
- **VNPg** – sloveso následované syntaktickým jménem (v předložkovém či nepředložkovém pádě) a předložkovou skupinou, v tomto pořadí. Předložková skupina může rozvíjet sloveso i jméno.
- **NPgV** – syntaktické jméno (opět bez ohledu na pád a předložku), předložková skupina a sloveso. Předložková skupina může rozvíjet sloveso i jméno.

- **VPgA** a **PgAV** – sloveso, předložková skupina a přídavné jméno, resp. předložková skupina, přídavné jméno a sloveso. Předložková skupina může rozvíjet sloveso i jméno.[9]

Pro naše účely není třeba filtrovat věty s posledním "základním podezřelým" slovosledným vzorcem: **ˆNNPg** – dvě syntaktická jména v libovolném pádě na začátku klauze následovaná předložkovou skupinou. Chyba syntaktického analyzátoru v připojení skupiny k jednomu či druhému jménu neovlivňuje zisk pozorovaných slovesných rámců.

Příklad 7 Karla Olivy (citováno podle Straňáková-Lopatková (2001)) varuje, že i věty bez základních podezřelých slovosledných vzorců a o jediném (složeném) slovese mohou být syntakticky velmi komplikované[10]. Podobně jako v příkladu 4 na str. 31 i zde vychází bezespornost syntaktického rozboru věty z valence, tentokrát však jde o valenci podstatných jmen.

(7)



3.2.7 Dostupnost dostatečně jednoduchých vět v korpusech

Výsledky aplikace výše popsaného vícestupňového filtru na prvních 144 839 vět z ČNK shrnuje tabulka 3.4 na následující straně. Údaje pro tabulku byly přímo převzaty z výstupu programu AX, viz 5.7.1 na str. 72.

V tabulce je přehledně uvedeno, do které fáze zpracování jednotlivé věty směly ještě postoupit. Ve druhém sloupci jsou uvedeny počty vět, které byly na jednotlivých filtrech zamítnuty, ve třetím sloupci je tento údaj vyjádřen kumulativně, s ohledem na to, že zamítnuté věty již nebyly podrobeny další filtraci. Pro zdůraznění stupňovitosti filtrace jsou jednotlivé filtry pojmenovány též postupně písmeny abecedy.

Podobnou tabulku uvádíme i pro věty z evaluační části PDT, viz. tabulka 3.5 na straně 36. Část těchto vět se nepodařilo načíst pro drobné nesrovnalosti ve formátu CSTS.

Ukazuje se, že přibližně 15 % vět projde všemi popsanými filtry. Pokud bychom neuvažovali věty se základními podezřelými slovosledными vzorci, bude

[9] Jak podrobně popisuje původní práce, Straňáková-Lopatková (2001), těmto slovosledným vzorcům je třeba rozumět tak, že nesmí být spatřeny v žádném kroku nedeterministické redukční analýzy. Pro dostatečnou rychlost zpracování však v naší práci při filtrování vět volíme přímočarý deterministický postup. Slovosledný vzorec **VPgA** testujeme bohužel již ve fázi, kde mohlo být koncové přídavné jméno připojeno pod následující podstatné jméno. Domníváme se, že touto chybou závěrečné výstupní věty nezatížíme příliš, tuto otázku by však jistě bylo vhodné studovat podrobněji, nebo se rozhodnout pro plnou nedeterministickou redukční analýzu.

[10] Komplikované především z hlediska míry *neprojektivity*, tj. stručně řečeno počtu "děr" v závislostním zápisu věty. Podrobněji viz Holan et al. (1998; Holan et al. (2001).

Část vět z ČNK	**Počet vět**	**Postupně zamítnuto**
Odpovídá	22 203 (15,3 %)	
Zamítnuto-A pomlčka	18 156 (12,5 %)	18 156 (12,5 %)
Zamítnuto-B apostrof	33 (0,0 %)	18 189 (12,6 %)
Zamítnuto-C lomítko	1 369 (0,9 %)	19 558 (13,5 %)
Zamítnuto-D dvojtečka ne na konci	1 030 (0,7 %)	20 588 (14,2 %)
Zamítnuto-F středník ne na konci	947 (0,7 %)	21 535 (14,9 %)
Zamítnuto-G číslice	22 458 (15,5 %)	43 993 (30,4 %)
Zamítnuto-H nepárová závorka	548 (0,4 %)	44 541 (30,8 %)
Zamítnuto-J příliš mnoho sloves	3 594 (2,5 %)	48 135 (33,2 %)
Zamítnuto-K větná skladba	66 098 (45,6 %)	114 233 (78,9 %)
Zamítnuto-L podezření **VNPg**	5 661 (3,9 %)	119 894 (82,8 %)
Zamítnuto-M podezření **NPgV**	1 860 (1,3 %)	121 754 (84,1 %)
Zamítnuto-N podezření **PgAV**	218 (0,2 %)	121 972 (84,2 %)
Zamítnuto-O podezření **VPgA**	664 (0,5 %)	122 636 (84,7 %)
Celkem vět	144 839 (100,0 %)	122 636 (84,7 %)

Tabulka 3.4: Zastoupení "velmi jednoduchých vět" v ČNK.

počet "velmi jednoduchých vět" (u PDT) o necelých deset procent vyšší, tj. dosáhne přibližně 25 %.

Souhrnné charakteristiky vybraných vět ve vztahu k celku srovnává tabulka 3.6 na následující straně. Není překvapením, že průměrná délka velmi jednoduchých vět je téměř poloviční ve srovnání se všemi větami evaluační části PDT. Podobně nepřekvapí ani stejně významné snížení počtu sloves.

Následující kapitola ukazuje podrobně, jak se zlepší úspěšnost stávajících parserů, omezíme-li se na profiltrované věty.

3.3 Úspěšnost parserů na "velmi jednoduchých větách"

V této kapitole srovnáme přesnost extrakce povrchových valenčních rámců pomocí jednotlivých syntaktických analyzátorů pro češtinu, ovšem pouze z vět, které prošly filtrací (viz výše), a jsou tedy "velmi jednoduché".

Aby bylo možné úspěšnost analyzátorů měřit, byly opět použity věty z evaluační části PDT, u nichž je správná syntaktická struktura k dispozici.

Kritéria hodnocení jednotlivých analyzátorů jsou stejná jako v kapitole 3.1.4 na straně 26. Hodnotíme správně zapojené uzly, správně zpracované celé věty, a pro účely naší práce pak správné a nezapomenuté syny sloves.

Z tabulky 3.7 na str. 37 je význam odfiltrování příliš složitých vět pro úspěšnost současných syntaktických analyzátorů dobře patrný. Srovnáme-li klasický údaj úspěšnosti, procento správně zavěšených uzlů, pro různě složité věty, zjistíme, že parserům Dana Zemana a Zdeňka Žabokrtského filtrace pomohla o přibližně 10 %, z čehož zlepšení díky odstranění vět s podezřelými

Věty evaluační části PDT	**Počet vět**	**Postupně zamítnuto**
Odpovídá	1 113 (15,6 %)	
Zamítnuto-A pomlčka	725 (10,1 %)	725 (10,1 %)
Zamítnuto-B apostrof	30 (0,4 %)	755 (10,6 %)
Zamítnuto-C lomítko	14 (0,2 %)	769 (10,7 %)
Zamítnuto-D dvojtečka ne na konci	30 (0,4 %)	799 (11,2 %)
Zamítnuto-F středník ne na konci	71 (1,0 %)	870 (12,2 %)
Zamítnuto-G číslice	841 (11,8 %)	1 711 (23,9 %)
Zamítnuto-H nepárová závorka	9 (0,1 %)	1 720 (24,0 %)
Zamítnuto-J příliš mnoho sloves	289 (4,0 %)	2 009 (28,1 %)
Zamítnuto-K větná skladba	3 360 (47,0 %)	5 369 (75,0 %)
Zamítnuto-L podezření **VNPg**	484 (6,8 %)	5 853 (81,8 %)
Zamítnuto-M podezření **NPgV**	164 (2,3 %)	6 017 (84,1 %)
Zamítnuto-N podezření **PgAV**	9 (0,1 %)	6 026 (84,2 %)
Zamítnuto-O podezření **VPgA**	16 (0,2 %)	6 042 (84,4 %)
Celkem vět	7 155 (100,0 %)	6 042 (84,4 %)

Tabulka 3.5: Zastoupení "velmi jednoduchých vět" v evaluační části PDT.

	Uzlů	Sloves
V průměrné větě	17,22	2,26
Ve velmi jednoduché větě bez podezřelých WOP	9,91	1,39

Tabulka 3.6: Počet uzlů a sloves ve "velmi jednoduchých větách".

slovosledn ými vzorci (viz 3.2.6 na straně 33) tvořilo jen nepatrnou část (jen 1,3 % u Žabokrtského; u Zemana dokonce došlo k mírnému poklesu úspěšnosti). U Collinsova parseru nedosáhlo zlepšení úspěšnosti podle uzlů takové míry, výsledná úspěšnost 87,9 % je však poměrně zajímavá.

Není překvapením, že úspěšnost parserů měřená podle počtu zcela správně rozpoznaných vět, se díky filtraci složitých vět zvýšila velmi výrazně. Opět s výjimkou Collinsova parseru šlo dokonce o více než zdvojnásobení úspěšnosti. Collins však podle tohoto kritéria stále vítězí, analyzuje více než 50 % velmi jednoduchých vět správně.

Význam filtrace vět s podezřelými slovosledn ými vzorci je z hlediska zcela správně rozpoznaných vět podle očekávání vyšší. Pro Collinsův a Zemanův parser přinesla zlepšení o cca 5 %, pro parser Žabokrtského dokonce o 10 %.

Velmi uspokojivé zlepšení je vidět též u posledních dvou kritérií, procenta sloves se syny identifikovanými zcela správně (Synové sloves správně) a procenta sloves, u nichž žádný syn nechybí, ale někteří mohou přebývat (Synové sloves nechybí). Žabokrtského a Zemanův parser syny zcela správně určí u přibližně poloviny výskytů sloves, Collins u téměř 65 % sloves. Filtrace podezřelých slovosledn ých vzorců v tomto kritériu opět hrála významnou roli.

Uzly správně	**Uzlů**	**Collins**	**Zeman**	**Žabokrtský**
Všechny věty	126 030	82,51 %	69,15 %	73,8 %
Velmi jednoduché věty	20 028	87,70 %	79,40 %	82,3 %
... navíc bez podezřelých WOP	11 030	87,89 %	79,31 %	83,6 %
Celé věty správně	**Vět**	**Collins**	**Zeman**	**Žabokrtský**
Všechny věty	7 319	30,95 %	15,00 %	18,4 %
Velmi jednoduché věty	1 786	47,14 %	29,00 %	31,6 %
... navíc bez podezřelých WOP	1 113	52,83 %	34,41 %	41,5 %
Synové sloves správně	**Sloves**	**Collins**	**Zeman**	**Žabokrtský**
Všechny věty	16 329	55,32 %	33,11 %	39,5 %
Velmi jednoduché věty	2 472	61,37 %	41,87 %	44,8 %
... navíc bez podezřelých WOP	1 546	64,68 %	47,02 %	53,8 %
Synové sloves nechybí	**Sloves**	**Collins**	**Zeman**	**Žabokrtský**
Všechny věty	16 329	73,73 %	55,86 %	74,7 %
Velmi jednoduché věty	2 472	78,52 %	67,76 %	84,1 %
... navíc bez podezřelých WOP	1 546	81,44 %	71,28 %	84,9 %

Tabulka 3.7: Srovnání parserů na "velmi jednoduchých větách".

3.4 Zpracování a filtrace povrchových rámců

V této kapitole se zaměříme na problematiku zpracování *pozorovaných rámců*, tj. přehledu všech doplnění spatřených u jednotlivých výskytů slovesa, s cílem získat přehled forem doplnění, které jsou pro dané sloveso typické. Pokusíme-li se typická doplnění uvádět nikoli izolovaně, ale navíc v typických sestavách, můžeme hovořit o *povrchových rámcích*[11].

V tabulce 3.8 na následující straně uvádíme příklad pozorovaných rámců pro sloveso *dát* seřazených podle četnosti. Pro jednoduchost pocházejí pozorované rámce prozatím z vět v PDT, aby byla míra chyb v pozorovaných rámcích co nejnižší. Z výskytů slovesa *dát* byly odfiltrovány též všechny výskyty v pasivu; slovesa v pasivu mají strukturu doplnění typicky odlišnou. Záměrně však nebyly odlišeny výskyty slovesa ve zvratných variantách (*dát se*, *dát si*) od nereflexivních výskytů, neboť na základě analýzy jednotlivých vět není možné rozhodovat, zda je sloveso reflexivum tantum, nebo zda částice *se* není pouze krátký tvar zvratného zájmene v roli běžného doplnění slovesa. Pro větší přehlednost uvozujeme v uvedených rámcích číslo pádu doplnění znakem `#`.

Z přehledu jsou dobře patrné oba závažné problémy extrakce povrchových rámců:

- Sloveso se často vyskytuje s volnými doplněními (též *adjuncts*), o nichž nelze prohlásit, že by byla pro sloveso typická. (Zde např. předložka *v* se šestým pádem.)

[11] V literatuře zaměřené na angličtinu se užívá pojmu *subkategorizační rámce (subcategorization frames)*. Pojmům povrchového a subkategorizačního rámce lze ve většině případů rozumět stejně, neboť pro angličtinu přehled typických *forem* slovesných doplnění postačuje. V případě češtiny s významně bohatším repertoárem forem vyjadřujících tutéž funkci (viz 4.2 na str. 48) si však distinkci mezi rámcem *povrchovým* a *valenčním* uvědomujeme více, a proto budeme raději užívat tato ostře odlišující označení.

238 (100 %)	**Celkem výskytů slovesa *dát* v aktivním tvaru**
22 (9,2 %)	`infin se`
21 (8,8 %)	`#1 infin se`
6 (2,5 %)	`infin se VV`
5 (2,1 %)	`infin se že`
4 (1,7 %)	`#1 infin podle-2#2 se`
4 (1,7 %)	`#1 #4`
3 (1,3 %)	`#1 infin se v-1#6`
3 (1,3 %)	`#1 infin se VV`
3 (1,3 %)	`#1 #3 #4`
3 (1,3 %)	`#4`
3 (1,3 %)	`#1 #3 za-1#4`
3 (1,3 %)	`#3 #4`
3 (1,3 %)	`#1 #4 v-1#6`
2 (0,8 %)	`si`
2 (0,8 %)	`se`
151 (63,4 %)	**Ostatní (143 dalších typů pozorovaných rámců)**

Tabulka 3.8: Pozorované rámce slovesa *dát* v PDT. V přehledu byla všechna určitá slovesa závislá na slovese dát označena jako `VV` (vedlejší věta) a jako `infin`, pokud byla v infinitivu.

- Pozorované rámce často neobsahují celý povrchový rámec slovesa, některé členy jsou elidovány. (Zde nejvýrazněji např. *dát <co>*.)

Oba problémy vedou ke značnému počtu rozdílných typů pozorovaných rámců (zde celkem 158) a i nejčetnější pozorovaný rámec byl spatřen v méně než 10 % výskytů slovesa *dát*.

3.4.1 Metody statistické filtrace

V literatuře (Brent (1991; Manning (1993; Sarkar and Zeman (2000; Briscoe and Carroll (1997) ad.) lze najít řadu odlišných statistických metod, jejichž cílem je ze seznamu rámců pozorovaných získat rámce povrchové. Většina metod je zaměřena na angličtinu[12] a předpokládá, že:

- Správný povrchový rámec bude mezi pozorovanými rámci zastoupen.
- Správný povrchový rámec bude zastoupen v dostatečné míře na to, aby jej statistické metody filtrace rámců dokázaly najít.

Jak správně poukazuje Sarkar and Zeman (2000), volná doplnění se ovšem (pro češtinu) nezřídka vyskytnou ve všech pozorovaných rámcích a *žádný*

[12]Pro angličtinu se uvádí cca 19 typů povrchových rámců, z nichž některé jsou parametrizovány předložkami (např. TV tranzitivní sloveso, NP_P(<PREP>) přímý objekt a předložková fráze s předložkou <prep>). Naproti tomu Sarkar and Zeman (2000) tvrdí, že česká slovesa mohou mít více než 130 různých typů povrchových rámců.

pozorovaný rámec tedy nemůže být právem prohlášen za povrchový rámec slovesa.

Citované metody statistické filtrace se pohybují v nejlepším případě u hranice cca 80 % správných rozhodnutí, zda jedno konkrétní doplnění slovesa v konkrétním výskytu je volným doplněním, nebo členem povrchového rámce. Ve srovnání metod filtrace, jak jej podrobně provádí Korhonen, Gorrell, and McCarthy (2000), se však ukazuje, že podle významnějšího kritéria – procenta rámců, které daná metoda v datech nalezne z celkového počtu rámců, které byly slovesům přiřazeny ručně – dosahují metody jen cca 50% úspěšnosti. Ještě závažnějším výsledkem Korhonen, Gorrell, and McCarthy (2000) je však zjištění, že o celých 25 procentních bodů, tj. cca 75 %, je úspěšnější metoda, která z pozorovaných rámců vezme prvních n nejčastějších, až do určitého ručně zvoleného prahu.

S ohledem na poznatky z literatury nebudeme tedy v této práci zvažovat nasazení metod statistické filtrace a dáme přednost strukturálnímu přístupu k pozorovaným rámcům.

3.4.2 Náznak lingvistické filtrace

V některých výjimečných případech je možné na základě lexikálního obsazení v doplnění slovesa určit povahu tohoto doplnění. Především je žádoucí identifikovat doplnění, která typicky *nejsou* součástí slovesných rámců. Pokud se z pozorovaných synů slovesa podaří včas odstranit doplnění, která jsou nad vší pochybnost volná, významně se tím ulehčí následným metodám spojování pozorovaných rámců.

V naší práci proto z řady pozorovaných doplnění ihned odstraňujeme:

- rematizátory (přehled pochází z Hajičová, Panevová, and Sgall (1999))
- částice
- časová příslovce[13]
- vedlejší věty příčiny a podmínky (v prvním hrubém přiblížení uvažujeme všechny věty uvozené podřadicími spojkami *protože*, *proč*, *když*, *kdyby*, *pokud* a *aniž*; bylo by vhodné tento seznam připravit pečlivěji)

Odhad časových určení na základě lexikálního obsazení

U doplnění realizovaných (rozvitou) jmennou či předložkovou skupinou se rovněž můžeme pokusit o rozhodnutí, zda jde o časové určení. Následující tabulka shrnuje výsledky úspěšnosti algoritmu identifikace časových určení mezi doplněními slovesa na základě seznamu typicky časových lexikálních jednotek, který byl připraven ručně úpravou seznamu připraveného Zdeňkem

[13] Přehled typů příslovcí připravil a spravuje Zdeněk Žabokrtský. Původně tento seznam pro potřeby anotátorů PDT připravovala Dr. Eva Buráňová. Seznam je využíván především při ruční anotaci PDT.

Žabokrtským pro práci (2002) na základě české verze EuroWordNetu[14]. Seznam obsahuje substantiva, u kterých byl alespoň jeden ze synsetů, do něhož patří, zařazen v EWN Top Ontology (základní sémantické třídění slov) do třídy Time.

Algoritmus dostane na svůj vstup podstrom syntaktického rozboru věty, v jehož kořeni se nachází podstatné jméno (předložka je tentokrát zavěšena pod řídicím jménem). Na základě tohoto podstromu a pevného slovníku algoritmus rozhodne, zda je toto doplnění časové, nebo ne. Není již přitom činěn rozdíl mezi typem časového určení (kdy, odkdy, dokdy, jak dlouho ap.).

Byly studovány čtyři odlišné algoritmy: první pro své rozhodnutí studoval pouze řídicí jméno v doplnění, druhý procházel celý podstrom a hledal, zda mezi uzly nenajde jakýkoli náznak, jedno "časové" substantivum. Třetí algoritmus požadoval, aby všechna plnovýznamová slova doplnění s výjimkou číslovek byla "časová". Konečně čtvrtý algoritmus používal heuristiku a požadoval, aby alespoň polovina uzlů doplnění byla v daném seznamu. Pokud algoritmy neuspěly s hledání důkazu *pro* časové určení, vydaly pro doplnění zápornou odpověď.

Úspěšnost na podstromech kdekoli	**TP+TN**	**FP**	**FN**
Řídicí slovo doplnění	96,31 %	2,24 %	1,46 %
Stačí jeden náznak	95,68 %	3,01 %	1,31 %
Plnovýznamová slova vyjma číslovek	95,42 %	0,64 %	3,94 %
Polovina uzlů doplnění	95,07 %	0,00 %	4,93 %
Celkem studováno	124 457 podstromů kdekoli		
Úspěšnost na podstromech pod slovesem	**TP+TN**	**FP**	**FN**
Řídicí slovo doplnění	95,74 %	1,82 %	2,45 %
Stačí jeden náznak	95,13 %	2,68 %	2,19 %
Plnovýznamová slova vyjma číslovek	92,97 %	0,33 %	6,70 %
Polovina uzlů doplnění	91,58 %	0,00 %	8,42 %
Celkem studováno	55 090 podstromů pod slovesem		

Tabulka 3.9: Úspěšnost identifikace časových určení na základě lexikálního obsazení v podstromu pod podstatným jménem.

Měření úspěšnosti algoritmů s různou citlivostí bylo provedeno na větách, které byly dosud označkovány na tektogramatické rovině. Výsledky měření jsou uvedeny v tabulce 3.9 odděleně pro všechny podstromy řízené substantivem a odděleně pro takové podstromy, které se navíc vyskytovaly pod slovesem.

Ve sloupci **TP+TN** je vyjádřen podíl správně identifikovaných časových doplnění (tj. správně rozhodnodnutá kladná odpověď, true positive, **TP**, a správně rozhodnutá záporná odpověď, true negative, **TN**) z celkového počtu studovaných doplnění. Ve sloupci **FP** je uvedeno procento doplnění, u nichž se algoritmus zmýlil, když vydal kladnou odpověď (false positive, **FP**), v posledním sloupci pak opačný případ, kdy algoritmus neprávem vydal zápornou odpověď

Ukazuje se, že správně identifikovat časové určení pod slovesem je mírně

[14] `http://www.elda.fr/cata/text/M0015.html`

[14] S výjimkou číslovek a číslic.

obtížnější, než identifikovat časové určení kdekoli. Ze čtyřech studovaných algoritmů je přitom nejjednodušší algoritmus založený na kontrole řídicího jména zároveň nejúspěšnější. Přitom případy, kdy tento algoritmus omylem mezi časová určení zařadí doplnění jiného druhu, představují pouze necelá dvě procenta. Automatizované extrakci povrchových rámců sloves tato míra chyby významná doplnění slovesa tedy neskryje.

Popsaná lingvistická filtrace celkem snížila počet typů pozorovaných rámců ze 158 na 127 a počet typů doplnění z 97 na 65. Počet typů rámců i doplnění je pro ruční zpracování stále nevhodný, i když má lingvistická filtrace nepochybně význam.

3.4.3 Spojování rámců

Jako protiklad k metodám statistického rozhodování, který pozorovaný rámec označit jako povrchový rámec slovesa, lze uvést metodu spojování pozorovaných rámců s cílem najít povrchové rámce s ohledem na "typické překryvy" v doplněních.

Jak se ukázalo v průběhu práce, není tato myšlenka vůbec nová, i když samostatně a soustavně se jí dosud nevěnoval podle našich znalostí žádný z autorů. Metodu spojování pozorovaných rámců lze najít jednak v Skoumalová (2001), a jednak v Basili and Vindigni (1998). Přitom Skoumalová (2001) pracuje na datech pocházejících nikoli z korpusů, ale ze slovníku BRIEF, a díky tomu jí stačí čelit významně nižšímu počtu různých typů pozorovaných doplnění. Algoritmus popsaný v této práci s ohledem na množství typů doplnění identifikovaných v textových korpusech bohužel není možné úspěšně použít.

Druhá práce, Basili and Vindigni (1998), spojuje pozorované rámce do hierarchie částečného uspořádání (lattice), podobně jako uvažujeme i zde. Práce se však především zaměřuje na automatické omezení daného valenčního slovníku na poddoménu jazyka charakterizovanou daným textovým korpusem. Z tohoto důvodu opět navržený algoritmus výběru povrchových rámců z hierarchie pozorovaných rámců není pro naši situaci bez vhodného valenčního slovníku relevantní. Navíc práce studuje angličtinu, kde je repertoár typů doplnění opět nižší než v češtině, a získané hierarchie pozorovaných rámců je proto ještě účelné graficky prezentovat.

Vyjdeme-li z vět PDT obsahujících sloveso *dát* (pro jednoduchost studujeme zatím metodu na datech, kde není pochyb o správnosti syntaktické analýzy), najdeme mezi 238 výskyty slovesa po provedení výše popsané filtrace doplnění celkem 127 pozorovaných rámců s celkem 65 různými typy doplnění. (Přehled nejčetnějších z těchto rámců není příliš odlišný od příkladu uvedeného na str. 38.)

Za zmínku stojí metoda slučování rámců, která se ukázala jako neúspěšná a dobře ilustruje míru "šumu" volných doplnění v pozorovaných rámcích. V první fázi jsme se snažili studovat hierarchii pozorovaných rámců ve směru od rámců s nejvyšším počtem doplnění. Pod tyto nejširší rámce jsme v hierarchii zařazovali jako podmnožiny další pozorované rámce. Hierarchie pozorovaná z tohoto směru

je však již od první úrovně velmi nepřehledná, jak ilustrujeme příkladem[15] 3.2 na str. 45.

V příkladu je u každého pozorovaného rámce uveden jednak počet bezprostředních následníků v hierarchii (počet podrámců), a jednak počet výskytů. Přitom jsou zvlášť uvedeny výskyty získané díky libovolnému z následníků a zvlášť výskyty samotného citovaného rámce. Vrchol hierarchie je v tomto případě přidán uměle (viz `+0` vlastních výskytů) a představuje úplné sjednocení všech pozorovaných doplnění; některá doplnění byla u jednoho výskytu slovesa spatřena i vícekrát (např. překvapivě `2x#1`).

Hierarchie pozorovaných rámců se jeví mnohem přehlednější, viz příklad 3.3 na straně 45, pokud budeme postupovat směrem od nejchudších rámců k rámcům s více doplněními. Na vrchol hierarchie se při tomto pohledu dostane prázdný rámec, výskyt slovesa bez jakýchkoli doplnění (která prošla předchozí filtrací). I v tomto příkladu jsou rámce v hierarchii utříděny podle klesajícího počtu výskytů rámce či jeho následníků v hierarchii, do předních pozic se tak dostanou nejčastější typy doplnění.

Tento způsob prezentace pozorovaných povrchových rámců považujeme za velmi vhodný pro anotátora. Interaktivní hierarchické prohlížení by dále mohlo být rozšířeno o konkrétní příklady rámců ze zdrojových dat a anotátor by byl při přípravě valenčního slovníku veden od nejčetnějších rámců k méně četným.

Ve snaze identifikovat povrchové rámce na základě této hierarchie pozorovaných rámců bez asistence člověka jsme implementovali jednoduchý algoritmus "rozbalování" hierarchie až do "rozumné" hloubky. Stručně lze postup shrnout takto:

- Každý rámec, který byl (včetně výskytů potomků) spatřen alespoň n krát, zkusme "rozbalit" a raději studovat jako kandidáty na povrchové rámce jeho syny.
- Uvažujme pouze ty syny, kteří svým výskytem dosahují alespon p procent celkového počtu výskytů rámce. (Výskyty vždy včetně všech potomků.)
- Rekurzivně "rozbalujme" hierarchii, dokud nebudou mít všechny dosavadní kandidátské rámce méně než n výskytů nebo každý z jejich synů nedosáhne p procent výskytů svého otce.

Za povrchové rámce slovesa algoritmus prohlásí množinu kandidátských rámců, u nichž se rozbálovaní hierarchie zastavilo. (Duplicitní rámce je nutno odstranit; k jednomu rámci se dalo dostat více cestami v hierarchii.)

Použijeme-li popsaný algoritmus na rámce získané sloveso pro *dát*, dostaneme pro parametry $n = 20$ a $p = 10$ resp $p = 5$ rámce uvedené v tabulce 3.10 na následující straně. Ve sloupci *Vlastní výskyty* je uveden počet výskytů slovesa právě s daným povrchovým rámcem. Ve sloupci *Výskyty bohatších rámců* je uveden počet výskytů rámců, které jsou nadmnožinou sledovaného rámce (tj. obsahují nějaká doplnění navíc, teoreticky volná doplnění). Ve sloupci *Výskyty*

[15] Příklad hierarchie je obtížné prezentovat na omezené ploše stránky, při naší práci jsme použili náš jednoduchý nástroj na interaktivní prohlížení, "rozbalování", orientovaných grafů v textovém režimu, viz přílohu C na straně 100.

Rámce odhadnuté pro p=10	Vlastní výskyty	Výskyty bohatších rámců	Výskyty chudších rámců	Součet výskytů realizací rámce	Procento realizací rámce v celkovém počtu výskytů
`#1 #3 #4`	8	8	19	35	15,0 %
`#1 VV infin se`	5	2	66	73	31,3 %
`#1 #4 časové_určení`	2	4	14	20	8,6 %
`#1 infin podle-2#2 se`	5	1	59	65	27,9 %
Rámce odhadnuté pro p=5					
`#1 #3 #4`	8	8	19	35	15,0 %
`infin se že`	5	5	32	42	18,0 %
`#1 VV infin se`	5	2	66	73	31,3 %
`#1 #4 časové_určení`	2	4	14	20	8,6 %
`#1 najevo#Db že`	3	3	2	8	3,4 %
`#1 infin podle-2#2 se`	5	1	59	65	27,9 %
`#1 #3 za-1#4`	3	0	4	7	3,0 %
`#1 infin se v-1#6`	3	0	58	61	26,2 %
`#1 #4 v-1#6`	1	2	15	18	7,7 %
`#1 #4 do-1#2`	1	1	15	17	7,3 %
`#3 #4 VV`	1	1	14	16	6,9 %
`#1 #4 #7`	1	1	16	18	7,7 %
`#1 #4 na-1#4`	1	1	14	16	6,9 %
`#1 #4 z-1#2`	1	1	13	15	6,4 %

Tabulka 3.10: Ukázkový výstup automatického rozbalení hierarchie rámců.

chudších rámců jsou naopak započteny jen výskyty podmnožin sledovaného rámce. Není sledován údaj o výskytu rámců ostatních případů (neprázdný průnik s daným rámcem, ale také nějaká doplnění navíc). Tři uvedené počty jsou sečteny v předposledním sloupci tabulky. Tento údaj je odhadem výskytů "realizací" sledovaného rámce, ve smyslu: byl pozorován rámec sám, jeho nadmnožina (výskyt "s volnými doplněními") nebo jeho podmnožina (výskyt "s elidovanými členy"). Poslední sloupec tabulky pak udává, jakou část celkového počtu výskytů slovesa *dát* lze takto zjednodušeně interpretovat jako "realizaci" daného rámce. Přirozeně nemusí součet procent dát dohromady 100 %.

3.4.4 Shrnutí

V této kapitole jsme se zaměřili na získávání povrchových syntaktických údajů z vět, které nejsou syntakticky rozebrány. Konkrétně šlo o extrakce slovesných

rámců.

Ukázali jsme, že typicky udávané údaje o úspěšnosti parserů (pro češtinu přibližně 70 až 80 % správně zapojených uzlů) nejsou z hlediska konkrétních úkolů příliš relevantní. Pro účely našeho cíle, extrakce slovesných rámců, dosahují parsery úspěšnosti přibližně pouze 30 až 50 % správně zpracovaných výskytů sloves. Pro aplikace, kde je třeba pracovat se správně rozebranou celou větou, lze pak očekávat od parserů správně zpracovaných pouze 15 až 30 % vstupních vět.

Představili jsme lingvisticky motivovaná kritéria filtrace vstupních vět tak, abychom mohli očekávat kvalitněji zpracované věty. Naše filtry identifikují "velmi jednoduché věty" a teprve ty jsou předloženy parserům. Ukázali jsme, že průměrně 15 až 20 % vět (podle restriktivnosti filtru) z korpusů zapadne ještě do této kategorie.

Při použití studovaných parserů pouze na "velmi jednoduché věty" se úspěšnost měřená počtem správně zavěšených uzlů zlepší o 5 až 10 % (nejlepší parser dosáhne necelých 88 % správně zavěšených uzlů). Z hlediska kritéria extrakce slovesných rámců filtrace vstupních vět přinese zlepšení o 10 až cca 15 % (nejlepší parser dosáhne téměř 65 % správně pozorovaných slovesných rámců). Nejvýznamnější posun úspěšnosti filtrace vstupních vět přináší poslednímu kritériu, počtu správně analyzovaných vět. Dva ze sledovaných parserů dosáhly zdvojnásobení úspěšnosti podle tohoto kritéria. Nejlepší ze studovaných parserů je pak schopen více než 50 % vstupních vět analyzovat bezchybně.

V poslední části kapitoly jsme upozornili na obtížnost dalšího úkolu, zpracování pozorovaných rámců tak, abychom získali rámce povrchové. Navrhli jsme několik jednoduchých lingvisticky motivovaných filtrací členů v pozorovaných rámcích. Rovněž jsme vyhodnotili jednoduchý algoritmus identifikace časových doplnění slovesa. Na základě lexikální jednotky v kořeni jmenné skupiny algoritmus správně identifikuje časové určení v cca 96 % případů. Navrhli jsme postup probírání pozorovaných rámců pro anotátora a představili jednoduchý automatický způsob. Automaticky získané povrchové rámce však nepokládáme za lingvisticky adekvátní.

```
- (78 podrámců s 232+0 observacemi) 2x(#1) #3 #4 #6 #7 ADJ#- ADJ#1 ADJ#4 NUMER#- NUMER#1 ...
  - (2 podrámců s 77+1 observacemi) #1 VV infin podle-2#2 se
  - (1 podrámců s 70+1 observacemi) #1 VV infin se tak-3#Db
    - (2 podrámců s 65+5 observacemi) #1 VV infin se
      - (2 podrámců s 32+24 observacemi) #1 infin se
        - (2 podrámců s 4+27 observacemi) infin se
        - (0 podrámců s 0+1 observacemi) #1
      - (1 podrámců s 31+9 observacemi) VV infin se
  - (3 podrámců s 60+1 observacemi) #1 #7 infin se
  - (3 podrámců s 60+1 observacemi) #1 časové_určení infin se
  - (2 podrámců s 57+3 observacemi) #1 infin se v-1#6
  - (2 podrámců s 57+1 observacemi) #1 infin se totiž#Db
  - (2 podrámců s 57+1 observacemi) #1 infin k-1#3 se však#J^
  - (2 podrámců s 57+1 observacemi) #1 infin na-1#6 se
  - (1 podrámců s 56+1 observacemi) #1 infin ovšem-1#J^
  ...
```

Obrázek 3.2: Hierarchie pozorovaných rámců slovesa *dát* ve směru od nejobsažnějších rámců.

Nultá a první úroveň hierarchie rámců slovesa *dát*: (Výpis je úplný.)

```
- (10 podrámců s 231+1=232 observacemi)
  - (5 podrámců s 124+2=126 observacemi) se
  - (4 podrámců s 122+2=124 observacemi) infin
  - (11 podrámců s 111+1=112 observacemi) #1
  - (16 podrámců s 59+6=65 observacemi) #4
  - (9 podrámců s 33+2=35 observacemi) #3
  - (5 podrámců s 5+3=8 observacemi) si
  - (3 podrámců s 7+1=8 observacemi) do-1#2
  - (4 podrámců s 5+1=6 observacemi) na-1#4
  - (0 podrámců s 0+1=1 observacemi) jasně_^(*1ý)#Dg najevo#Db že
  - (0 podrámců s 0+1=1 observacemi) 2x(časové_určení)
```

Hluší pohled do hierarchie:

```
- (10 podrámců s 231+1=232 observacemi)
  - (5 podrámců s 124+2=126 observacemi) se
    - (17 podrámců s 92+27=119 observacemi) infin se
    - (1 podrámců s 2+1=3 observacemi) #7 se
    - (1 podrámců s 1+1=2 observacemi) do-1#2 se
    - (0 podrámců s 0+1=1 observacemi) #1 na-1#4 se
    - (0 podrámců s 0+1=1 observacemi) #1 pak#Db se v-1#4
  - (4 podrámců s 122+2=124 observacemi) infin
    - (17 podrámců s 92+27=119 observacemi) infin se
    - (1 podrámců s 1+1=2 observacemi) #1 časové_určení infin
    - (0 podrámců s 0+1=1 observacemi) #1 #4 aby infin podle-2#2
    - (0 podrámců s 0+1=1 observacemi) ADJ#- infin
  - (11 podrámců s 111+1=112 observacemi) #1
  - (16 podrámců s 59+6=65 observacemi) #4
  - (9 podrámců s 33+2=35 observacemi) #3
  - (5 podrámců s 5+3=8 observacemi) si
    - (0 podrámců s 0+1=1 observacemi) #4 během#2 si
    - (0 podrámců s 0+1=1 observacemi) #1 #4 časové_určení si
    - (0 podrámců s 0+1=1 observacemi) #4 k-1#3 si
    - (0 podrámců s 0+1=1 observacemi) #4 kromě#2 na-1#6 si z-1#2
    - (0 podrámců s 0+1=1 observacemi) #4 číslovka#- si spolu#Db
  - (3 podrámců s 7+1=8 observacemi) do-1#2
  ...
```

Obrázek 3.3: Hierarchie pozorovaných rámců slovesa *dát* ve směru od nejchudších rámců.

Kapitola 4

Od pozorovaných rámců k valenci

V úvodním souhrnu dosavadních znalostí (kapitola 1.3 na straně 12) jsme stručně zavedli nejvýznamnější pojmy z teorie valence v rámci FGP. Shrňme a doplňme ještě několik zásadních pojmů úvodem této kapitoly. V další části se zaměříme na praktické problémy vztahu teorie a korpusových dat v otázce hledání valenčních rámců sloves.

4.1 Základní pojmy

Aktanty a volná doplnění.

Hranici mezi *aktanty* ("povinnými" doplněními slovesa) a *volnými doplněními* definují různé lingvistické teorie různě. FGP definuje oba pojmy na rovině hloubkové syntaxe (tektogramatická rovina) a používá tato kritéria (podrobnosti viz Panevová (1980)):

- Aktant je pro sloveso charakteristický.
- Aktant nemůže sloveso rozvíjet vícenásobně.[1]
- Volné doplnění daného typu může rozvíjet prakticky libovolné sloveso.
- Volné doplnění může sloveso rozvíjet i vícenásobně.[2]

Ve shodě s ostatní literaturou věnovanou FGP uvádí manuál pro značkování na tektogramatické rovině, viz Hajičová, Panevová, and Sgall (1999), jako aktanty tyto typy doplnění:

ACT, PAT, ADDR, ORIG, EFF[3]

Všechny ostatní typy (větně členské funkce) doplnění sloves jsou zařazeny v kategorii volných doplnění. Na tomto místě je vhodné upozornit na

[1] Tj. daný typ aktantu se může pod daným slovesem objevit nejvýše jedenkrát.
[2] Např. *Sešli se <v Praze> <na Václavském náměstí>.*
[3] Přesný popis jednotlivých typů doplnění viz citovaná literatura.

skutečnost, že pojmu "volné doplnění" se velmi často užívá i pro označení doplnění slovesa ještě na rovině povrchové syntaxe. FGP přitom tento pojem definuje až na rovině tektogramatické, a na rovinu povrchové syntaxe je formálně vzato nutno tento pojem převádět.

Z hlediska automatického rozhodování tohoto kritéria je dobré shrnout: Podaří-li se nám správně určit funkci daného doplnění, je otázka aktant vs. volné doplnění zodpovězena. O obtížnosti automatického odhadu funkce na základě formy však viz 4.2 na následující straně.

Doplnění obligatorní a fakultativní.
Teorie valence v rámci FGP odlišuje doplnění *obligatorní* a *fakultativní*. Odlišení se provádí pro aktanty i pro volná doplnění a kritériem je tzv. *dialogový test*[4].

Stojí za poznámku, že kritérium obligatornosti není možné při současných znalostech žádným způsobem rozhodovat automaticky, neboť se rozhodnutí opírá velmi hluboce o sémantiku daného slovesa. Z tohoto důvodu se kritériem obligatornosti nebudeme v této práci podrobněji zabývat.

Do kontrastu s pojmem fakultativnosti doplnění je nutno uvést pojem *vypustitelnosti* doplnění, který se vyjadřuje k rovině povrchové syntaxe a nikoli k rovině tektogramatické.[5]

Pozorovaný rámec slovesa.
Tímto pojmem označujeme množinu synů slovesa v konkrétním povrchovém syntaktickém rozboru konkrétní věty.

Povrchový rámec slovesa.
Tímto pojmem (v přímé korespondenci s anglickým *subcategorization frame*) označujeme takový seznam forem doplnění slovesa, který je pro dané sloveso typický. Vztahem mezi pozorovaným a povrchovým rámcem jsme se zabývali již v kapitole 3.4 na straně 37.

Valenční rámec slovesa.
Valenční rámec v teorii valence v rámci FGP je definován jako výčet těch doplnění slovesa, která jsou *aktanty* nebo *obligatorními* volnými doplněními.

Provedené shrnutí pojmů je záměrně omezeno na užší otázky valence sloves s cílem budování valenčních slovníků sloves a např. valence ostatních slovních druhů je zcela záměrně ponechána stranou. V případě zájmu v tomto směru odkazujeme čtenáře na text Panevová (1980). Shrnutí je k dispozici též v dalších pracích, např. Skoumalová (2001) či Straňáková-Lopatková (2001).

[4] Může-li původce věty nevědět danou informaci (*Rodiče přijeli. Kdy? Nevím.*), jde o doplnění fakultativní. Není-li s ohledem na použité sloveso přípustné, aby informaci nevěděl (*Kam přijeli? *Nevím.*), jde o doplnění obligatorní. Podrobněji Panevová (1980).

[5] Ani povrchovou vypustitelnost není možné na základě syntakticky analyzovaných vět automaticky rozhodnout bez zpřesnění lingvistické definice tohoto pojmu, neboť přinejmenším v krátkých odpovědích na otázky je možné vypustit všechna doplnění slovesa a pojem *vypustitelnosti* tak ztratí svůj smysl. (*Potkal Petr Pavla? Potkal.*)

4.2 Vztah formy a funkce

Asymetrický dualismus vztahu formy a funkce jazykových znaků je ve strukturalistické tradici hluboce vžit a formuloval jej již Karcevskij (1929). V našem případě jde o to, že doplnění slovesa vyjadřující jednu funkci (např. konatel ACT, určení místa kde LOC) mohou být i pro jedno sloveso vyjádřena více různými formami (např. nominativem či infinitivem či genitivem, resp. předložkou *v* či *na* v lokativu či předložkou *u* v genitivu).

Díky právě probíhající anotaci PDT na úroveň tektogramatické roviny je možné vztah formy a funkce studovat podrobněji. V příloze B na str. 92 lze najít detailní výpis z dosud anotovaných cca 26 000 vět – souhrn nejčastějších povrchových realizací nejčastějších funkcí a naopak nejčastějších rolí nejčastějších forem.

Už dosavadní rozsah anotovaných dat nabízí zajímavé možnosti ověřovat tvrzení teorie. Např. se ukazuje, že tzv. *primární funkce* dané formy[6] jsou v některých případech voleny diskutabilně: Primární funkcí předložky *na* s akuzativem je podle Hajičová, Panevová, and Sgall (1999) funkce DIR3. Tuto funkci má však v anotovaných větách (v roli doplnění slovesa) forma *na+4* jen v cca 30 % výskytů, častěji (téměř 40 %) však forma hraje roli PAT.

Při snaze o automatické odvození funkce doplnění na základě jeho formy, což je první nutnou podmínkou pro možnost automatického získávání valenčních rámců[7], si musíme uvědomit, že funkce doplnění je určena několika faktory:

- Povrchová realizace doplnění. (Pro jmenná doplnění především pád a předložka.)
- Lexikální hodnota slovesa (pokud jsme se již omezili na aktivní formy slovesa, jinak též slovesný rod).
- Lexikální obsazení v doplnění.

Automatickou identifikací funktorů při převodu vět z roviny povrchové syntaxe na rovinu tektogramatickou se zabývá též Žabokrtský, Džeroski, and Sgall (2002). Práce se však zabývá funkcemi všech větných členů a nesoustřeďuje se na role doplnění u sloves. Na všech uzlech věty (v tektogramatickém zápisu; počet uzlů se na jednotlivých rovinách popisu mírně liší) dosahuje cca 80% úspěšnosti a významně usnadňuje práci anotátorům.

V následující části se podíváme na obtížnost automatického zpracování jednotlivých faktorů ovlivňujících funkci doplnění slovesa podrobněji.

Přirozeně je však velmi snadné vytvořit takové příklady vět, které pro systém automatizované identifikace funkcí doplnění slovesa budou představovat nevyřešitelnou hádanku:

[6] Viz Hajičová, Panevová, and Sgall (1999). Daná forma by v bezpříznakovém případě, kde nejde o omezení na určitou třídu kontextů, měla vyjadřovat danou funkci.

[7] Druhou nutnou podmínkou ve stávající definici valenčních rámců je kritérium obligatornosti; jak však víme, není toto kritérium strojově rozhodnutelné a bude třeba ještě zvážit jiné možnosti, jak kritérium aproximovat, nebo odlišně definovat.

(8) Čeká na strážnici.

Pokud systém například bude pracovat se vstupem, který je morfologicky nejednoznačný, nelze v příkladu 8 vůbec rozhodnout, zda jde o doplnění s funkcí LOC (čeká kde), což předložka *na* s šestým pádem signalizuje, nebo zda jde o doplnění s funkcí PAT (čeká na koho), jak signalizuje (i když již méně jednoznačně) předložka *na* se čtvrtým pádem.

4.2.1 Faktor povrchové realizace doplnění

Faktor povrchové realizace doplnění (pád a předložka) bohužel ve většině případů nemá rozhodující vliv – daná forma sice některé funkce vyřazuje ze hry, ale stále velmi mnoho funkcí na výběr zbývá, jak je zřejmé z přílohy B.2 na str. 97.

Ukazuje se, že i "forma nejjistější", tj. nominativ závislý na slovese, vyjadřuje podle dosavadních dat svou primární funkci ACT jen v 91 % výskytů, a to i přesto, že byly do přehledu zařazeny pouze výskyty sloves v činném rodě. Nepříjemný odhad horní hranice úspěšnosti jakýchkoli automatických postupů podporují i dosavadní rámce shromážděné v zárodku valenčního slovníku češtiny.[8] Nominativ jako doplnění slovesa je uveden v roli ACT u 97 % sloves, ve 2 % hraje roli PAT a dále pak výjimečně další role aktantů i volných doplnění (EFF, COMPL, MANN a BEN).

4.2.2 Faktor lexikální hodnoty slovesa

Faktor lexikální hodnoty slovesa není přirozeně možné v algoritmu identifikace funkce doplnění pod daným slovesem zohlednit, protože pro dané sloveso právě zjišťujeme, jaké funkce vyžaduje. Pokud by však byl tento faktor "slabý", tj. vstupoval do hry jen u velmi malého procenta sloves, mohli bychom jej zanedbat.

Již velmi jednoduchý příklad nás však varuje, že faktor lexikální hodnoty slovesa může být v řadě případů právě jediný odlišující prvek:

(9) a. Posuň ten obrázek <ke konci stránky>$_{\texttt{DIR3}}$.

b. Nechej ten obrázek <ke konci stránky>$_{\texttt{LOC}}$.[9]

Pro podrobnější studium vlivu lexikální hodnoty slovesa na výklad jednotlivých forem doplnění jsme připravili (opět na základě vět anotovaných na tektogramatickou rovinu) tabulku 4.1 na následující straně.

V tabulce je uvedeno prvních patnáct forem doplnění, která se vyskytla u největšího počtu sloves.

[8] Je třeba zdůraznit, že v tomto slovníku jsou shromážděna opravdu jen valenční a typická doplnění sloves, a údaj tedy není zatížen zkreslením volných doplnění.

[9] Jakkoli negramaticky může příklad působit, výraz *ke konci stránky* anotovaný jako LOC je doslova uveden v Hajičová, Panevová, and Sgall (1999).

Je třeba zdůraznit, že tabulka studuje vztah formy a funkce dle četnosti u *sloves*, nikoli dle četnosti *výskytů* daných forem. Z tabulky na straně 97 je však zřejmé, že všechny formy citované zde byly spatřeny jako doplnění slovesa alespoň pětsetkrát.

Údaj o počtu sloves, u nichž se doplnění vyskytlo, je uveden včetně podílu z celkového počtu 3 036 sloves ve druhém sloupci. V dalších sloupcích jsou slovesa, u nichž byla spatřena daná forma doplnění, rozdělena do kategorií podle toho, jaké všechny funkce dané doplnění u různých výskytů daného slovesa hrálo. Rozlišujeme přitom, zda daná forma doplnění daného slovesa hrála výhradně roli jednoho konkrétního aktantu (sloupce ACT až EFF), nebo zda hrála roli výhradně aktantů, ale i v rámci výskytů daného slovesa několika různých (sloupec Směs aktantů), nebo zda hrála zásadně roli libovolného z volných doplnění, nebo zda i v rámci výskytů jednoho slovesa hrála daná forma v některých případech roli aktantu či aktantů a v jiných případech roli volných doplnění.

Například nominativ *#1* tak byl u 79,3 % sloves z celkového počtu 2 487 spatřen výhradně v roli ACT. U 2,5 % sloves však hrál výhradně roli PAT, atd.

	Celkem	**Výhradně aktanty**					**Směs**	**Volná doplnění**	
Forma	**u sloves**	ACT	PAT	ADDR	ORIG	EFF	**aktantů**	Jen volné	I aktanty
#1	2 487 (81,9 %)	79,3 %	2,5 %	-	-	0,1 %	10,9 %	0,6 %	6,7 %
#4	1 819 (59,9 %)	1,4 %	78,9 %	2,2 %	-	0,4 %	4,5 %	3,0 %	9,6 %
VV	1 168 (38,5 %)	1,0 %	9,8 %	-	0,1 %	1,4 %	1,1 %	64,6 %	22,0 %
v#6	988 (32,5 %)	-	0,9 %	-	-	-	-	96,4 %	2,7 %
se	940 (31,0 %)	-	4,7 %	0,4 %	-	-	-	90,5 %	4,4 %
že	610 (20,1 %)	-	-	-	-	-	-	100,0 %	-
#7	585 (19,3 %)	0,3 %	7,2 %	-	-	2,1 %	0,2 %	83,4 %	6,8 %
#2	545 (18,0 %)	19,1 %	40,9 %	1,8 %	-	0,6 %	5,7 %	17,6 %	14,3 %
#3	476 (15,7 %)	1,7 %	18,1 %	45,2 %	0,2 %	-	6,7 %	22,9 %	5,3 %
na#6	459 (15,1 %)	-	3,5 %	0,4 %	0,7 %	-	0,2 %	91,5 %	3,7 %
na#4	402 (13,2 %)	0,7 %	21,9 %	1,7 %	-	2,7 %	1,0 %	62,9 %	9,0 %
i-1	374 (12,3 %)	-	-	-	-	-	-	100,0 %	-
infin	365 (12,0 %)	3,6 %	38,1 %	-	-	6,6 %	5,2 %	30,4 %	16,2 %
do#2	363 (12,0 %)	-	1,9 %	-	-	1,9 %	-	92,3 %	3,9 %
#X	351 (11,6 %)	61,8 %	14,5 %	4,6 %	-	0,6 %	7,1 %	7,1 %	4,3 %
s#7	348 (11,5 %)	-	19,5 %	6,6 %	-	2,3 %	2,9 %	62,1 %	6,6 %
z#2	317 (10,4 %)	-	6,0 %	-	6,3 %	-	0,3 %	82,0 %	5,4 %
však	292 (9,6 %)	-	-	-	-	-	-	100,0 %	-
po#6	284 (9,4 %)	-	2,1 %	-	0,7 %	-	-	94,0 %	3,2 %
podle#2	282 (9,3 %)	-	-	-	-	-	-	100,0 %	-

Tabulka 4.1: Vliv lexikální hodnoty slovesa na určení funkce doplnění na základě jeho formy. Jsou uvažovány pouze výskyty sloves v činném rodě.

Z velikých rozdílů mezi jednotlivými formami doplnění je patrné, že se "individuální" přístup k jednotlivým formám doplnění vyplatí.

Abychom však odpověděli na naši původní otázku: pro danou formu doplnění má lexikální hodnota slovesa, pod nímž bylo toto doplnění spatřeno, vliv na funkci, kterou toto doplnění hraje. Například akuzativ *#4* je možné bezpečně pokládat za PAT jen u 78,9 % sloves. Připravíme-li tedy systém, který bude

při převádění povrchových rámců sloves na valenční hledět pouze na formu doplnění, dopustí se tento systém chyby pro přibližně pětinu sloves, která ve svém povrchovém rámci akuzativ mají. Systém na základě pozorovaného akuzativu rozhodne, že jde o PAT, přitom u celkem 4 % sloves hraje akuzativ jednoznačně roli jiného z aktantů.

Podobně, pokud systém pro každý nominativ pod slovesem označí toto doplnění jako ACT, dopustí se opět u přibližně pětiny sloves chyby. Přitom u 2,6 % sloves šlo o jasný PAT, resp. EFF.

Z uvedených forem doplnění je neobtížnější určit funkci infinitivu, který pro 38,1 % sloves hraje ve všech výskytech roli PAT, u 30,4 % sloves však hraje ve všech výskytech roli některého z volných doplnění. Naopak jako nejsnáze identifikovatelná se jeví doplnění *že*, *i*, *však* a *podle*, která vždy hrají roli volného doplnění (nerozlišujeme však, zda jde o jediný druh volného doplnění, nebo směs různých druhů doplnění, to lze nahlédnout v příloze B.1 na str. 92).

4.2.3 Faktor lexikálního obsazení v doplnění

Lexikální obsazení ve studovaném doplnění slovesa je algoritmicky zatím rovněž obtížně uchopitelný faktor.

Prvním přiblížením, o než se pokouší slovník BRIEF, je odlišení doplnění (především na základě jmenného rodu) na osoby a předměty. Dále by bylo možné např. použít Eurowordnet k převodu konkrétních lexikálních jednotek na více či méně zobecněné skupiny významů. Počet ruzných typů jednotek by se tak snížil a manuálně či automaticky by bylo možné i za současného množství tektogramaticky anotovaných vět získat zobrazení mezi hrubým významem doplnění a jeho větně členskou funkcí. Výsledky tohoto postupu by však zatím pravděpodobně nedosáhly očekávání.

Metodu identifikace funkce na základě lexikálního obsazení doplnění lze zatím tedy použít prakticky jen na typická časová určení, kde je seznam užívaných slov poměrně malý a především uzavřený. Výsledky této metody jsme předvedli v kapitole 3.4.2 na straně 39.

4.2.4 Shrnutí

V této kapitole jsme se zaměřili na otázku automatického získávání valenčních rámců sloves. V úvodu jsme nejprve v definicích odlišili valenční rámce od dosud uvažovaných pozorovaných a povrchových rámců sloves.

Pro rozhodnutí, které doplnění do valenčního rámce patří a které ne, je nutné vyhodnotit dvě kritéria: určit větně členskou funkci doplnění (nebo alespoň odlišit aktant a volné doplnění) a ověřit kritérium obligatornosti. Druhé uvedené kritérium zatím není možné vyhodnocovat automaticky, a proto jsme se jím podrobněji nezabývali.

Druhá část kapitoly byla věnována přehledu problémů při snaze o vyhodnocení prvního kritéria, automatické identifikaci funkce doplnění. Na datech, která byla pro PDT dosud anotována na tektogramatickou rovinu popisu, jsme ukázali, že i

forma "nejjistější" svou rolí ve větě, nominativ pod slovesem v aktivním tvaru, hraje roli konatele jen v 91 % výskytů. Upozornili jsme navíc, že v přepočtu na slovesa může být tento problém ještě významnější: nominativ byl jako jednoznačný aktor spatřen jen u 79 % z 2 487 pozorovaných sloves.

Soudíme však, že selektivně, pro jednotlivé formy doplnění, je možné připravovat samostatná kritéria a dosáhnout tak lepších výsledků. Rovněž by bylo zajímavé na základě korpusových dat studovat vzájemné vztahy mezi jednotlivými doplněními slovesa a získat přehled o platných "lingvistických implikacích". (Např. je-li sloveso rozvito akuzativem a současně předložkou *na* s kauzativem, pak ...) Tuto otázku však zatím necháváme otevřenou dalšímu výzkumu.

Vzhledem k tomu, že podkladová data zatím procházejí fází anotace, může se v nich dosud objevit množství chyb. V takovém případě je náš souhrnný pohled na doplnění sloves jedním z vhodných podnětů k nalezení a případné korekci chyb v anotaci.

Kapitola 5

Systém AX

5.1 Cíle systému a jazyka AX, možnosti aplikace

Systém a jazyk AX je navržen s cílem pohodlně formulovat pravidla pro zpracovávání vět přirozeného jazyka (*ne*jednoznačně) anotovaných na morfologické rovině popisu. Jazyk a systém AX umožňují snadno formulovat:

- pravidla pro úpravy (zjednodušování) vstupní věty,
- filtry pro odstranění vět, které svou složitostí přesahují požadované omezení.

Vzhledem k obecnosti návrhu jazyka je však v systému AX možné implementovat částečný či plnohodnotný syntaktický analyzátor řízený pravidly a autor gramatiky může podle svého uvážení gramatiku formulovat jak v závislostním, tak bezprostředně složkovém pojetí. Analyzátor rovněž plně podporuje nedeterministické zpracování vstupu a pro jednu vstupní větu může vrátit libovolný počet řešení.

Nový systém jsme v této práci budovali proto, že žádný z dosavadních systémů (např. ALE, PATR či Holan (2001)) nám nepřipadal dostatečně flexibilní a s dostačně silnými vyjadřovacími prostředky pro *pohodlnou a elegantní* formulaci našich požadavků na věty přirozeného jazyka. Netvrdíme přitom, že stejné požadavky by nebylo možné ve stávajících systémech nebo přímo v libovolném z programovacích jazyků implementovat, to rozhodně ne. Díky vyjadřovací síle filtrů a pravidel systému AX je však formulace lingvistických filtrů velmi pohodlná. Implementací tohoto systému jsme si rovněž připravili celou řadu komplexních datových struktur a základních algoritmů, které bude v budoucnu možné použít i ve velmi odlišných lingvistických systémech, mj. např. syntaktických analyzátorech založených na automatické extrakci platných gramatických pravidel a jejich aplikaci stochastickým způsobem.

Naznačme nyní souhrnné schéma běhu systému AX. Vstupní věta je reprezentována jako posloupnost sestav rysů, z nichž každá odpovídá jednomu vstupnímu slovu věty. (Podrobněji viz kapitola 5.2 na následující straně.) Postup

zpracování je pro přehlednost rozdělen do několika navazujících bloků. Na vstupu každého bloku je množina posloupností sestav rysů, jinými slovy množina dosavadních *čtení věty*. Blok je buď to *filtrem* (viz 5.4 na straně 62), jehož úkolem je některá z dosavadních čtení věty zamítnout a nedovolit další zpracování, nebo skupinou *pravidel*, jejichž úkolem je čtení věty nějak upravit a generovat novou množinu čtení. Do prvního bloku vždy vstupuje vstupní věta, množina čtení na výstupu z posledního bloku je výstupem celého systému. Pořadí i typ jednotlivých bloků je zcela na autorovi programu pro systém AX, tzv. *gramatiky*. Příklad chodu je naznačen ve schématu 5.1.

Fáze:	Vstup	Filtr_1	Generování_1	F_2	G_2	F_3	G_3	F_4
Počet čtení ve hře:	1	$\emptyset$						
Počet čtení ve hře:	1	1	20	10	50	$\emptyset$		
Počet čtení ve hře:	1	1	30	12	35	17	34	16

Obrázek 5.1: V příkladu byla první vstupní věta zamítnuta filtrem 1, druhá vstupní věta byla zamítnuta filtrem 3 a teprve třetí vstupní věta byla přijata a gramatika pro ni našla 16 různých čtení.

Blokové schéma zpracování vět vycházelo z představy o gramatice pro extrakci povrchových rámců slovesa a pro ilustraci ji uvedeme i zde:

- Negativní fáze zamítne věty s nevhodnými slovními jednotkami (nerozpoznaná slova, složitá interpunkce ap.),
- Regulární filtr spojí pevné řetízky slov do nedělitelných jednotek (např. nepravé předložky, idiomy, skupiny čísel ap.),
- Následuje složení analytických slovesných tvarů a identifikace klauzí,
- Zamítnutí vět s příliš složitou strukturou klauzí,
- Redukce jmenných a předložkových skupin,
- Vyšetření rizika syntaktické homonymie jmenných a předložkových skupin, další možnost zamítnutí vět,
- Souhrn výsledků.

Detailně se blokovým zpracováním množin čtení vět zabývá kapitola 5.6 na straně 70. Kapitola 5.8.2 na str. 77 popisuje možnost, jak podstatným způsobem obohatit základní schéma zpracování vět a nabídnout autorovi gramatiky plnou kontrolu nad řízením běhu.

Přílohou k této práci je i CD s poslední verzí systému AX.

5.2 Základní datová struktura: Variantová sestava rysů

Sestavy rysů (feature structures, atribute-value matrices) představují velmi známou datovou strukturu. Významného uplatnění dostaly sestavy rysů ve

formalismu HPSG (viz Pollard and Sag (1994)), a odtud také pochází inspirace pro systém AX.

Pro detailní charakteristiku typovaných sestav rysů doporučujeme práci Penn (2000). Pro účely této práce postačí jednodušší a netypované zavedení této datové struktury, jak je předvedeme v této kapitole.

5.2.1 Definice variantové sestavy rysů

Variantová sestava rysů je jedno z:

- Jednoduchá hodnota (např. symbol sg pro jednotné číslo, nebo int(312) pro číslo hodnoty 312 ap.)
- Seznam dvojic tvaru (název položky - hodnota položky), kde název položky je z předem definované množiny položek a hodnota položky je variantová sestava rysů. Na pořadí dvojic v seznamu přitom nezáleží a žádné jméno položky se nesmí v seznamu vyskytnout vícekrát. V podstatě se tedy jedná o (částečné) zobrazení z množiny jmen do množiny variantových sestav rysů.
- Seznam možností sestav rysů, tj. seznam tvaru { ...prvky seznamu... }, kde každý prvek seznamu je variantová sestava rysů.

Právě poslední část definice, seznam možností, je tím, co naši variantovou sestavu rysů odlišuje od (obyčejných) sestav rysů. Variantové sestavy rysů umožňují v jediné struktuře popsat více přípustných možností.

Pro jednoduchost užívejme v dalším textu termín "sestava rysů" i pro variantovou sestavu rysů.

5.2.2 Operace nad sestavami rysů

Operace unifikace

Základní operací s (variantovými) sestavami rysů je unifikace. Unifikací dvou sestav rysů vznikne sestava rysů obsahující informace z obou vstupních sestav. Např.

$$\begin{bmatrix} \mathsf{jmeno} & \mathsf{Kamil} \\ \mathsf{prijmeni} & \mathsf{Horak} \end{bmatrix} \text{ a } \begin{bmatrix} \mathsf{prijmeni} & \mathsf{Horak} \\ \mathsf{vek} & \mathsf{int(32)} \end{bmatrix}$$

unifikují do struktury

$$\begin{bmatrix} \mathsf{jmeno} & \mathsf{Kamil} \\ \mathsf{prijmeni} & \mathsf{Horak} \\ \mathsf{vek} & \mathsf{int(32)} \end{bmatrix}$$

Unifikace může selhat, pokud obě vstupní sestavy rysů obsahují atribut téhož jména ale rozdílné hodnoty:

$$\begin{bmatrix}\mathsf{jmeno} & \mathsf{Kamil}\\ \mathsf{prijmeni} & \mathsf{Horak}\end{bmatrix} \quad \text{a} \quad \begin{bmatrix}\mathsf{jmeno} & \mathsf{Josef}\\ \mathsf{prijmeni} & \mathsf{Horak}\\ \mathsf{vek} & \mathsf{int(32)}\end{bmatrix}$$

neunifikují.

Je-li hodnotou nějakého atributu jedné vstupní sestavy rysů vnořená sestava rysů, je tato vnořená sestava unifikována s hodnotou téhož atributu druhé vstupní sestavy:

$$\begin{bmatrix}\mathsf{id} & \begin{bmatrix}\mathsf{jmeno} & \mathsf{Kamil}\\ \mathsf{prijmeni} & \mathsf{Horak}\end{bmatrix}\\ \mathsf{udaje} & \begin{bmatrix}\mathsf{stav} & \mathsf{svobodny}\end{bmatrix}\end{bmatrix} \quad \text{a} \quad \begin{bmatrix}\mathsf{id} & \begin{bmatrix}\mathsf{prijmeni} & \mathsf{Horak}\end{bmatrix}\\ \mathsf{udaje} & \begin{bmatrix}\mathsf{vek} & \mathsf{int(32)}\end{bmatrix}\end{bmatrix}$$

unifikují za vzniku:

$$\begin{bmatrix}\mathsf{id} & \begin{bmatrix}\mathsf{jmeno} & \mathsf{Kamil}\\ \mathsf{prijmeni} & \mathsf{Horak}\end{bmatrix}\\ \mathsf{udaje} & \begin{bmatrix}\mathsf{stav} & \mathsf{svobodny}\\ \mathsf{vek} & \mathsf{int(32)}\end{bmatrix}\end{bmatrix}$$

Rozšíření operace unifikace pro variantové sestavy rysů je přímočaré. Pokud vstupní sestavy neobsahují výčet variant, jde o obyčejnou unifikaci sestav rysů. Pokud alespoň jedna ze vstupních sestav připouští varianty, budou postupně unifikovány všechny přípustné kombinace vstupních variant. Výstupem bude taková sestava rysů, která připouští právě úspěšné výsledky unifikací.

$$\{\mathsf{int(32)},\ \mathsf{int(35)}\} \quad \text{a} \quad \{\mathsf{int(35)},\ \mathsf{int(40)}\}$$

unifikují za vzniku

$\mathsf{int(35)}$

Operace sloučení, merge

Pro variantové sestavy rysů je navíc definována operace sloučení (merge) dvou sestav rysů tak, aby výsledná sestava unifikovala právě s těmi sestavami rysů, které unifikují s libovolnou ze vstupních sestav.

$$\begin{bmatrix}\mathsf{jmeno} & \mathsf{Kamil}\\ \mathsf{prijmeni} & \mathsf{Horak}\end{bmatrix} \quad \text{a} \quad \begin{bmatrix}\mathsf{jmeno} & \mathsf{Josef}\\ \mathsf{prijmeni} & \mathsf{Horak}\end{bmatrix}$$

lze sloučit do sestavy rysů

$$\begin{bmatrix}\mathsf{jmeno} & \{\mathsf{Kamil},\ \mathsf{Josef}\}\\ \mathsf{prijmeni} & \mathsf{Horak}\end{bmatrix}$$

Ale např.

$$\begin{bmatrix}\mathsf{jmeno} & \mathsf{Kamil}\\ \mathsf{prijmeni} & \mathsf{Horak}\end{bmatrix} \quad \text{a} \quad \begin{bmatrix}\mathsf{jmeno} & \mathsf{Josef}\\ \mathsf{prijmeni} & \mathsf{Klement}\end{bmatrix}$$

lze sloučit pouze do sestavy rysů:

$$\left\{\begin{bmatrix}\mathsf{jmeno} & \mathsf{Kamil}\\ \mathsf{prijmeni} & \mathsf{Horak}\end{bmatrix}, \begin{bmatrix}\mathsf{jmeno} & \mathsf{Josef}\\ \mathsf{prijmeni} & \mathsf{Klement}\end{bmatrix}\right\}$$

a nikoli

$$\begin{bmatrix}\mathsf{jmeno} & \left\{\mathsf{Kamil},\ \mathsf{Josef}\right\}\\ \mathsf{prijmeni} & \left\{\mathsf{Horak},\ \mathsf{Klement}\right\}\end{bmatrix}$$

protože tato sestava by unifikovala *navíc* se sestavou "Josef Klement", která však neunifikovala s ani jednou ze vstupních sestav.

Operace sloučení se užívá při načítání nejednoznačných morfologických informací o daném slově, tak aby všechna morfologická čtení byla uchována v jediné (variantové) sestavě rysů a bylo možné se všemi čteními pracovat najednou (viz 5.2.3 na straně 59).

Operace extrakce hodnoty atributu

Při operaci extrakce atributu získáme z dané vstupní sestavy podsestavu (či jednoduchou hodnotu) uloženou v daném atributu. Pro variantové sestavy rysů je však třeba mít na paměti, že operace extrakce hodnoty atributu je nedeterministická, nemusí mít řešení vůbec (pokud atribut v sestavě není), ale vzhledem k možným variantám sestavy může mít i řešení několik.

Můžeme rovněž uvažovat o extrakci hodnoty hlouběji ve struktuře sestavy – v jejích podsestavách podle dané "cesty" atributy.

Budeme-li chtít definovat operaci extrakce alespoň polodeterministicky[1], nezbývá, než za jednoznačný výsledek prohlásit spojení všech výsledků, které je extrakcí možné získat; pokud alespoň jeden takový vysledek získat lze.

Je zřejmé, že deterministickou extrakcí všech hodnot v sestavě a jejich opětovným sloučením získáme sestavu novou, neekvivalentní se sestavou původní.

Operace obalení sestavy

Operace obalení sestavy slouží jako protiklad k operaci extrakce. Pro danou vstupní sestavu (např. [case - gen]) a vstupní atribut či cestu atributy (např agr) získáme strukturně hlubší sestavu rysů:

$$\begin{bmatrix}\mathsf{agr} & \begin{bmatrix}\mathsf{case} & \mathsf{gen}\end{bmatrix}\end{bmatrix}$$

[1] Tj. tak, aby vracela nejvýše jedno řešení, ale směla neuspět a nevrátit žádné. Podrobněji viz Somogyi, Henderson, and Conway (1995) a dokumentace k Mercury.

Operace mazání atributu

Ve variantových sestavách je rovněž širší prostor pro to, jak definovat operaci mazání atributu. Chceme-li např. v sestavě:

$$\left\{\begin{bmatrix}\mathsf{jmeno} & \mathsf{Kamil}\\ \mathsf{prijmeni} & \mathsf{Horak}\end{bmatrix}, \begin{bmatrix}\mathsf{jmeno} & \mathsf{Josef}\\ \mathsf{prijmeni} & \mathsf{Klement}\end{bmatrix}\right\}$$

smazat atribut jmeno, můžeme operaci mazání definovat jako deterministickou nebo jako nedeterministickou. V prvním případě zní úkol: Smaž ve všech variantách sestavy atribut jméno, a jako výsledek dostaneme:

$$\left\{\left[\mathsf{prijmeni} \quad \mathsf{Horak}\right], \left[\mathsf{prijmeni} \quad \mathsf{Klement}\right]\right\}$$

nebo stručněji $\left[\mathsf{prijmeni} \quad \left\{\mathsf{Horak}, \mathsf{Klement}\right\}\right]$

Při nedeterministické definici operace (s nepatrně jednodušší implementací) dostaneme postupně jako výsledky jednotlivé varianty sestavy, vždy se smazaným atributem jmeno:

$$\left[\mathsf{prijmeni} \quad \mathsf{Horak}\right] \quad \text{a} \quad \left[\mathsf{prijmeni} \quad \mathsf{Klement}\right]$$

Rozhodnutí, jak operaci mazání atributu definujeme, má především technické důsledky při manipulacemi se sestavami rysů ve složitějších částech kódu. Programátor může chtít nedeterminismu využívat, nebo naopak může chtít získat hned jedinou sestavu. Je však důležité uvědomit si, že výsledek determinického smazání atributu podle implementace *nemusí být* identický s výsledkem sloučení všech řešení, která poskytl nedeterministický postup. (Slučováním totiž vznikají přednostně "kompaktnější" sestavy, takové, které mají výčty možných variant co nejhlouběji ve své struktuře.) Z hlediska další unifikace těchto odlišných výsledků s jinými sestavami jsou však oba výsledky ekvivalentní.

Podobně jako u extrakce lze uvažovat o mazání atribu hlouběji ve struktuře sestavy rysů.

Lze též definovat operace restrikce a projekce nad sestavou rysů. Při restrikci smažeme ze sestavy rysů všechny uvedené atributy či cesty k atributům. Při projekci ze vstupní sestavy všechny uvedené atributy či cesty k atributům naopak ponecháme a ostatní odstraníme.

Poznámky k implementaci

Základní sestavy rysů jsou uchovávány v podobě seznamu dvojic atribut-hodnota uspořádaného podle názvů atributů (pořadí podle definice povolených atributů). Uspořádání umožňuje efektivněji implementovat operace nad sestavami, především často užívanou unifikaci. Navíc, jsou-li definovány povolené názvy atributů uspořádaně tak, aby atributy, které nejsnáze odlišují sestavy rysů byly mezi prvními, je běh unifikace v konkrétních situacích ještě rychlejší.

Rovněž je třeba upozornit na to, že stávající definice variantové sestavy

rysů dovoluje zapsat jedinou sestavu více různými zápisy, více či méně "kompaktními", podle toho, zda jsou jednotlivé varianty hodnot uvedeny hlouběji či výše ve struktuře podsestav. Z hlediska implementace je zde tedy riziko zbytečného plýtvání systémovými zdroji, pokud by sestavy byly obsažnější a nebyly zapisovány v dostatečně "kompaktním" tvaru. V praxi však k tomuto problému zatím nedochází.

5.2.3 Reprezentace slovního tvaru variantovou sestavou rysů

Pro každé vstupní slovo je k dispozici informace o možných základních tvarech (lemmatech) tohoto slova a o morfologických příznacích, které konkrétní tvar ve větě oproti základnímu tvaru nese. Každé vstupní slovo budeme reprezentovat jedinou variantovou sestavou rysů, lemma i morfologické kategorie budou uchovávány v samostatných atributech. Vstupní větu pak můžeme reprezentovat jako seznam sestav rysů jednotlivých slov.

Pro názornost uvedeme příklad reprezentace jednoho slova:

Slovo *má* např. ve větě: *Jakou barvu má stará židle?* je morfologickou analýzou zpracováno do formátu CSTS (zalomení řádek přidáno pro přehlednost):

```
<f>má<MMl>mít<MMt>VB-S---3P-AA---
     <MMl>můj<MMt>PSFS1-S1------1<MMt>PSFS5-S1------1
              <MMt>PSNP1-S1------1<MMt>PSNP4-S1------1
              <MMt>PSNP5-S1------1
```

Slovo *má* může být buď tvarem slovesa *mít*, nebo tvarem zájmena *můj* (a to hned v několika různých pádech jednotného i množného čísla ženského i středního rodu).

Při načítání tohoto vstupu se všechna možná čtení slova *má* na morfologické rovině uloží do jediné (variantové) sestavy rysů, viz obrázek 5.2 na následující straně.

Podrobný přehled o vztahu použitých atributů a morfologických značek, které jsou výstupem z morfologické analýzy nebudeme v této dokumentaci uvádět, k dispozici je na přiloženém CD. Názvy atributů i hodnot jsou poměrně mnemotechnické. Jmenujme však nejvýznamnější z atributů:

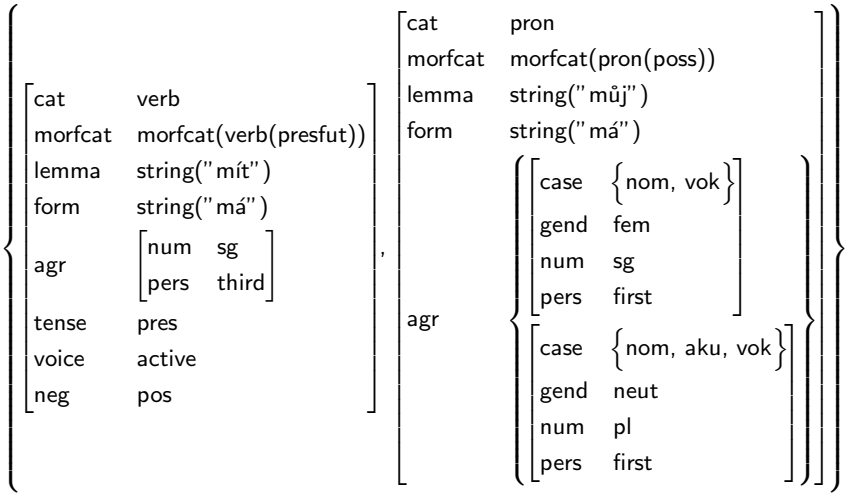
cat uchovává hrubou kategorii reprezentovaného slova (podstatné jméno, přídavné jméno, ..., interpunkce, ...).

morfcat nese podrobnou morfologickou kategorii, tak, jak je uložena ve druhé pozici pozičního systému morfologických značek.

form obsahuje konkrétní slovní tvar.

lemma obsahuje základní tvar slova.

agr obsahuje jako podsestavu rysů atributy case, gend, num a pers nesoucí údaje o jmenném pádu, rodu a čísle a (slovesné) osobě. Tyto atributy



Obrázek 5.2: Sestava rysů pro reprezentaci slovního tvaru *má*.

jsou reprezentovány v rámci společné podsestavy agr proto, že především u podstatných a přídavných jmen dochází v těchto atributech k četným nejednoznačnostem, přičemž kategorie slovního tvaru i lemmatu jsou pro všechny možnosti identické. Je tedy úspornější uchovat v sestavě společné atributy jen jednou a nejednoznačnost zaznamenat až hlouběji datové struktuře sestavy rysů.

5.2.4 Syntax variantových sestav rysů v jazyce AX

Předešlá kapitola podrobně představila datovou strukturu variantové sestavy rysů a základní operace definované nad touto strukturou. V této kapitole popíšeme část syntaxe jazyka AX týkající se variantových sestav rysů.

5.3 Zápis sestav rysů

Obrázek 5.2 ilustruje, jak je sestavou rysů reprezentován slovní tvar *má*, jenž může být chápán jako tvar slovesa *mít* i jako tvar zájmene *můj* v několika možných rodech, číslech a pádech.

Stručně se tato sestava rysů v syntaxi jazyka AX zapisuje takto:

```
[
  cat - verb, morfcat-'da(morfcat(verb(presfut)))', lemma - "mít", form - "má",
  agr - [ num-sg, pers-third ],
  tense - pres, voice - active, neg-pos
|
  cat - pron, morfcat - 'da(morfcat(pron(poss)))', lemma - "můj", form - "má",
  agr - [ case - nom;vok, gend - fem, num - sg, pers - first
        | case - nom; aku; vok, gend - neut, num - pl, pers - first ]
```

```
]
```

V příkladu jsme se setkali se všemi významnými prvky syntaxe:

- Sestava rysů je ohraničena hranatými závorkami.
- Jednotlivé varianty v rámci jedné sestavy rysů odděluje znak `|`.
- Dvojice atribut - hodnota se odděluje znakem `-`.
- Jednotlivé páry atribut - hodnota jsou odděleny čárkou.
- Jednotlivé varianty jednoduchých hodnot pro jeden atribut odděluje středník.
- Textové hodnoty je třeba uzavřít do uvozovek.
- Složité hodnoty (zde morfcat) je z technických důvodů (viz 5.9 na str. 79) zatím nutno uvádět v apostrofech a ve výrazu `'da(<hodnota>)'`. Tento nedostatek bude v nejbližší uveřejněné verzi systému odstraněn.

5.3.1 Zkratky

Pro konstatní sestavy rysů, které se v gramatice vyskytují často, je účelné zavést zkratky. Direktiva pro zavedení zkratek v jazyce AX buď zavádí jedinou zkratku, nebo zavádí hned skupinu zkratek najednou; takto:

```
shortcut verb = [cat-verb]

shortcuts
  noun = [cat-noun|morfcat-'da(morfcat(pron(pers)))';
                           'da(morfcat(pron(pers_short)))'],
  adj = [cat-adj|morfcat-'da(morfcat(pron(poss)))';
                         'da(morfcat(pron(poss_refl)))']
end
```

Zkratky lze od okamžiku zavedení používat na všech místech, kde je možné uvést konstatní sestavu rysů (tj. ve výrazech z konstatních sestav rysů, ve filtrech i v pravidlech, viz níže).

Jako jméno zkratky není možné použít žádné klíčové slovo jazyka AX. Seznam klíčových slov viz dokumentace na přiloženém CD.

5.3.2 Morfologická analýza na místě, "instantní sestavy rysů"

Jelikož i sestava pro jeden slovní tvar může být poměrně složitá a obsahuje řadu atributů, jejichž názvy je obtížné si zapamatovat, byl jazyk AX obohacen o možnost zavést sestavu rysů přímým příkladem slovního tvaru. V průběhu

kompilace gramatiky je tento slovní tvar (víceznačně) morfologicky analyzován a všechny přípustné varianty čtení daného slova jsou vyjádřeny variantovou sestavou rysů.

Požadavek o instantní morfologickou analýzu je vyjádřen zpětnými apostrofy (‘). Instantní sestava rysů se může objevit všude, kde je možné zapsat konstantní sestavu rysů, tj. ve zkratkách, konstatních výrazech, filtrech i pravidlech. Takto lze např. zavést zkratku pro konkrétní tvar slovesa být:

```
shortcut jsem = ‘jsem‘
```

5.4 Filtry: Regulární výrazy nad sestavami rysů

Pojem regulárního výrazu jistě není třeba čtenáři představovat. Na tomto místě tedy popíšeme pouze syntax jazyka AX týkající se regulárních výrazů. Stručně lze rozdíl mezi klasickými regulárními výrazy a regulárními výrazy v jazyce AX vystihnout takto:

- V jazyce AX je základním stavebním kamenem regulárního výrazu nikoli jeden symbol abecedy, ale jedna variantová sestava rysů.
- Při hledání podposloupnosti sestav rysů, která odpovídá danému výrazu, se neověřuje rovnost symbolů abecedy ve výrazu a ve vstupu, ale ověřuje se, zda vstupní sestava a sestava vyžadovaná výrazem unifikují.

5.4.1 Syntax regulárního výrazu

Regulárním výrazem v jazyce AX tedy je:

Jedna sestava rysů. Vstup odpovídá dané sestavě rysů, pokud jej tvoří právě jedna sestava rysů unifikující se zadanou sestavou. Speciálním případem je prázdná sestava rysů “`[]`”, s níž unifikuje libovolná vstupní sestava. Prázdnou sestavu je v regulárních výrazech možné podle konvence též zapsat jako tečku “`.`”.

Sestava může být zapsána buď přímo jako konstanta (např. `[cat-verb]`) nebo jako jméno zavedené zkratky (např. `verb`).

Odmítnutí sestavy rysů.
Vstup odpovídá, je-li tvořen jedinou sestavou rysů a ta neunifikuje s danou sestavou.

Odmítnutí sestavy se zapisuje v případě použití konstatní sestavy jako `![cat-verb]` nebo v případě zkratky jako `!verb`.

Odmítnutí libovolné ze sestav rysů.
Vstup odpovídá, je-li tvořen jednou sestavou rysů a tato sestava sestava neunifikuje se žádnou z daného seznamu sestav.

Odmítnutí libovolné ze sestav se zapisuje např. takto: `!{[cat-verb], zkratka_sestavy, ‘slovo‘}`.

Konkatenace regulárních výrazů.
Vstup odpovídá, pokud jej lze rozdělit na takové úseky, že postupně odpovídají daným regulárním výrazům.

Konkatenace se zapisuje jako prostá posloupnost regulárních výrazů, není potřeba používat žádný symbol, povolena je samozřejmě mezera či jiné bílé místo.

Varianty regulárních výrazů.
Vstup odpovídá, pokud odpovídá některé z daných variant.

Varianty je třeba oddělit znakem svislé čáry, "`|`".

Opakování regulárního výrazu s omezením minimálního a maximáního počtu opakování.
Vstup odpovídá, lze-li jej rozdělit na úseky požadovaného počtu, z nichž každý jednotlivě odpovídá danému regulárnímu výrazu.

Povolený počet opakování se připojuje hned za zápis regulárního výrazu a lze použít jednu z těchto variant:

	Počet opakování	
Syntax	**Min.**	**Max.**
`{m, n}`	m	n
`{m,}`	m	∞
`{m}`	m	∞
`*`	0	∞
`+`	1	∞
`?`	0	1

Pokud je regulární výraz použit v pravidle aplikovaném deterministickým způsobem (viz 5.5.4 na straně 68), je možné před požadovaný počet opakování vložit znak $<$ nebo $>$. Při determinické aplikaci je pak použit nejmenší ($<$) nebo nejvyšší ($>$) počet opakování, který vystupu vyhovuje. Implicitně se hledá nejdelší možný úsek.

Začátek a konec řetězce.
Tradičně regulární výraz `^` odpovídá začátku řetězce a výraz `$` odpovídá konci řetězce.

Operátory pro konstrukci regulárních výrazů mají standardní priority: `!` váže nejtěsněji (lze stejně použít jen u jediné sestavy rysů, nelze negovat celý regulární výraz), následují operátory počtu opakování, konkatenace a nejmenší prioritu má operátor alternativy (`|`).

Je-li třeba, je možné uzavírat regulární výrazy do kulatých či složených závorek. Složené závorky však mají navíc speciální význam vyznačení hledané skupiny, viz 5.5.1 na str. 65.

5.4.2 Syntax a sémantika filtru

Filtr má vždy tvar regulárního výrazu nad sestavami rysů.

Vstupem filtru je čtení věty, tj. posloupnost sestav rysů. Filtr prověřuje, zda tato posloupnost *jako celek* odpovídá zadanému regulárnímu výrazu. Je tedy lhostejno, zda na začátku výrazu uvedete `^` a na konci `$`, nebo ne.

Filtr se v jazyce AX zapisuje např. takto:

```
filter  .* verb .* verb .*  end

keep  !verb* verb !verb* | !conj*  end
```

Klíčové slovo `filter` říká: zamítni větu, pokud odpovídá danému regulárnímu výrazu. Klíčové slovo `keep` říká: zamítni větu, pokud neodpovídá danému regulárnímu výrazu. Klíčové slovo `end` označuje konec regulárního výrazu.

Pro účely ladění a závěrečné statistiky je velmi vhodné filtr pojmenovat. Jméno je možné uvést buď ve tvaru identifikátoru, tj. bez mezer a speciálních znaků, nebo uzavřít v uvozovkách. Za jménem je nutné uvést znak `:`.

```
filter zamitni_vic_nez_jedno_sloveso:
  .* verb .* verb .*
end

keep "Ponech jen věty s jediným slovesem nebo bez jakékoli spojky":
  !verb* verb !verb* | !conj*
end
```

5.5 Pravidla

Způsob aplikace pravidel v systému AX je inspirován klasickými pravidly Chomského gramatik, hlavní motivace však vychází z myšlenky tzv. *redukční analýzy*, analýzy zjednodušováním. Tento postup je dobře znám z literatury věnované závislostním gramatikám, viz např. Tesnière (1959; Sgall (1967; Mel'čuk (1988; Sgall and Panevová (1990), pěkně je popsán též v Straňáková-Lopatková (2001). Speciální formální prostředky pro analýzu jazyků do závislostní syntaxe pak nabízí v podobě gramatik např. Holan et al. (1998) (*Free Order Dependency Grammars*) a v podobě automatů např. Jančar et al. (1995; Jančar et al. (1996; Plátek (1999) (*restartovací automaty*).

Redukční pravidla jazyka AX mají vždy tvar:

`rule` <název> `:`
<náhrada> `--->` <vstupní regulární výraz> `::`
<omezení>
`end`

Pokud v pravidle nejsou formulována žádná <omezení>, ani nechceme pravidlo pojmenovat, je možné uvést pravidlo stručněji:

`rule` <náhrada> `--->` <vstupní regulární výraz> `end`

Hrubě lze postup aplikace pravidla shrnout takto:

- Ve vstupu je nalezen úsek (podřetez sestav rysů), který odpovídá <vstupnímu regulárnímu výrazu>.
- Je ověřeno, zda úsek splňuje též pořadavky dodatečných <omezení>.
- Úspěšně nalezený úsek je ve větě nahrazen řetězcem sestav rysů <náhrada>.

Aby bylo možné <náhradu> (říkejme též výstup pravidla) nějak "vypočítat" z nalezeného podřetězu vstupu pravidla, jsou v rámci jednoho pravidla k dispozici *proměnné*. Proměnné se vyskytují v hledaném <regulárním výrazu>, v <omezení> jsou na ně kladeny další požadavky a v <náhradě> jsou jejich aktuální hodnoty otištěny do výstupu pravidla.

Všechny proměnné jsou zásadně typu sestava rysů, tj. nesou jako svou hodnotu nějakou sestavu rysů. Všechny proměnné jsou *lokální* pro jednu aplikaci pravidla. Proměnné tedy nesdílejí svou hodnotu mezi různými pravidly, ale ani mezi více postupnými aplikacemi jednoho pravidla[2]. Podrobněji se proměnným věnujeme dále.

5.5.1 Regulární výrazy pro nahrazování a náhrada

Syntax regulárních výrazů je stejná jako syntax výrazů pro filtry (viz 5.4 na straně 62). Navíc je syntax obohacena o:

Proměnné.

Názvy proměnných se zapisují stejně jako názvy zkratek; proměnné stejně jako zkratky odpovídají právě jedné sestavě rysů. Proměnné není třeba nijak deklarovat, jsou automaticky alokovány při prvním výskytu.

Ve skutečnosti i zkratky jsou v rámci pravidla chápány jako proměnné. V <omezení> i <náhradě> je tedy možné odvolávat se ke zkratkám i k "čistým proměnným". Dá se tedy říci, že zkratka je obyčejná proměnná, která navíc musí unifikovat s konstantou definovanou ve zkratce; nebo jinak: každá proměnná ma počáteční hodnotu stejnojmenné zkratky.

V průběhu hledání výskytu vzorku, který odpovídá regulárnímu výrazu s proměnnými, dochází k tzv. *navazování* proměnných na vstupní sestavy. V jednoduchém případě jde o to, že se dosavadní hodnota proměnné unifikuje se vstupní sestavou. Pokud unifikace neuspěje, zkouší se regulární výraz napasovat na vstup jinak, pokud unifikace uspěje, je hodnota proměnné upravena tak, aby zahrnovala údaje zjištěné ze vstupní sestavy (výjimku však viz 5.5.5 na str. 69).

Pojmenované části nalezeného výrazu.

Podobně jako v klasických regulárních výrazech použitých pro nahrazování, i zde je možné jednotlivé nalezené části "vyznačit" a v náhradě dále použít.

Pro vyznačení části řetezu se v regulárním výrazu užívají složené závorky. Aby však při použití vyznačených částí nevznikaly pochybnosti o tom, která část je která, je vhodné každou vyznačenou část pojmenovat, např.:

[2] Motivace z restartovacích automatů: po úspěšné náhradě se automat restartuje.

...`o_zavorka {mezi_zavorkami: !{o_zavorka,z_zavorka}*} z_zavorka`...

Jméno pro vyznačenou část lze volitelně uvést mezi otvírací závorkou a dvojtečkou. Není-li jméno uvedeno, jsou vyznačené části jako obvykle číslovány od 1 dále.

<Náhrada> je tvořena seznamem (oddělovačem je mezera) sestav rysů a citací vyznačených částí nalezeného vzorku. Sestavy rysů lze zapsat jako konstanty, názvy zkratek či proměnných. Citace částí vzorků se zapisují za zpětným lomítkem (podobně, jako u klasických regulárních výrazů).

Uveďme příklad pravidla, které smaže otvírací a zavírací závorku, ale jen pokud se je podaří správně spárovat:

```
rule \mezi_zavorkami --->
  o_zavorka { mezi_zavorkami: !{o_zavorka,z_zavorka}* } z_zavorka
end
```

5.5.2 Dodatečné podmínky na regulární výrazy a náhradu

Dodatečná <omezení> účinnosti pravidla se zapisují jako neuspořádaná posloupost požadavků na proměnné. Omezení jsou chápána *deklarativně*, proto na jejich pořadí nezáleží. Požadavky jsou zásadně formulovány jako unifikace (pod)sestav dvojic proměnných. Uvažme pravidlo redukce přídavného jména a podstatného jména na pouhé podstatné jméno:

```
rule out_noun ---> adj noun ::
  adj.agr = noun.agr,
  out_noun = noun
end
```

Požadavek `adj.agr = noun.agr` garantuje, že k aplikaci pravidla dojde, jen pokud se přídavné a podstatné jméno shodnou v atributech potřebných pro jmennou shodu (viz 5.2.3 na straně 59). Požadavek `out_noun = noun` navíc zajišťuje, že výstupní sestava ponese právě všechny atributy, které původně neslo jméno; ovšem s přihlédnutím k případnému omezení variant v důsledku unifikace atributů čísla, rodu a pádu s hodnotami přípustnými pro přídavné jméno.

Jazyk AX navíc obsahuje jednoduchou syntaktickou konstrukci pro stručnější vyjádření více požadovaných vztahů na dvojici proměnných:

```
usnul <- cat, agr.num, agr.gend -> jsem
```

je ekvivalentní se zápisem

```
usnul.cat = jsem.cat,
usnul.agr.num = jsem.agr.num,
usnul.agr.gend = jsem.agr.gend
```

5.5.3 Způsob aplikace pravidel

Deklarativní chápání pravidel je vhodné především pokud chceme *vysvětlovat* vazby, které jsou konstrukcemi přirozeného jazyka respektovány, nebo pokud naopak známe respektované vazby a chceme pravidlo použít, jen pokud jsou vazby ve vstupní posloupnosti slov dodrženy.

Technicky je však ověřování platnosti podmínek prováděno již v průběhu hledání řetězu, který odpovídá danému regulárnímu výrazu. Pro lepší porozumění pravidlům je vhodné mít i tento postup zpracování na paměti.

Aplikace hledaného regulárního výrazu začne od libovolného vstupního slova věty (viz 5.5.4 na následující straně). Regulární výraz je aplikován na následující vstupní sestavy s možností backtrackingu (a případně nedeterministicky), aplikace se přitom provádí strukturální rekurzí podle regulárního výrazu, nikoli podle vstupní posloupnosti slov. To znamená, že jednotlivé varianty regulárního výrazu jsou postupně probírány a postupně aplikovány, dokud to unifikace aktuálních hodnot proměnných se vstupními sestavami (tj. *navazování* vstupních sestav na aktuální hodnoty proměnných) dovolují; současně jsou ověřována a prosazována i omezení na vzájemné vztahy proměnných.

Pokud se podaří dojít až na konec některé z větví regulárního výrazu, jsou již všechny podmínky ověřeny a všechny proměnné nastaveny na aktuální hodnoty. Ve vstupní větě je pak úsek, který odpovídal regulárnímu výrazu, nahrazen <náhradou> z definice pravidla.

Při *navazování* vstupní sestavy na danou proměnnou se přesně odehrává toto:

1. Ze zásobníku proměnných je získána aktuální hodnota proměnné daného jména.
2. Vstupní sestava je unifikována s touto hodnotou. (Selže-li unifikace, backtrackuje se do jiné větve regulárního výrazu.) Získá se tak "přesnější", "omezenější" hodnota proměnné.
3. Následně jsou prověřeny a aplikovány všechny podmínky, které pravidlo na vzájemné vztahy hodnot proměnných klade. Hodnoty dotčených proměnných i hodnota výchozí proměnné se tím mohou dále omezit.
4. Po prověření všech podmínek jsou nové hodnoty dotčených proměnných uloženy opět na zásobník (výjimkou je právě navázaná proměnná, pokud je tzv. *vstupní*, viz 5.5.5 na straně 69).
5. Nový zásobník proměnných je předán do dalšího zpracování této větve regulárního výrazu.

Aplikace podmínek kladených na vztahy proměnných není zcela jednoduchá záležitost. Pro ilustraci si představme pravidlo pro zpracování dvojice přídavných jmen a jména podstatného s cílem omezit na základě jmenné shody přípustné morfologické varianty jednotlivých slov:

```
rule adj1 adj2 noun ---> adj1 adj2 noun ::
```

```
  adj1 = [cat-adj], adj2 = [cat-adj], noun = [cat-noun],
  adj1.agr = adj2.agr,
  adj2.agr = noun.agr
end
```

Aplikujeme-li nyní postupně regulární výraz a podmínky tohoto pravidla na morfologicky nejednoznačný vstup *lesní žlutá kuřátka*, dojde ihned po navázání slova *lesní* v důsledku podmínek pravidla k omezení přípustných hodnot podsestavy `agr` proměnných `adj2` i `noun` na hodnotu:

$$\begin{bmatrix} \text{cat} & \text{adj} \\ \text{morfcat} & \text{'morfcat(adj(adj))'} \\ \text{lemma} & \text{"lesní"} \\ \text{agr} & \begin{bmatrix} \text{case} & \{\text{instr,lok,vok,aku,dat,gen,nom}\} \\ \text{gend} & \{\text{inanim,neut,fem,masc}\} \\ \text{num} & \{\text{pl,sg}\} \end{bmatrix} \\ \text{neg} & \text{'neg(pos)'} \end{bmatrix}$$

Pokud by následovalo např. slovo *radostnému*, bude pro neshodu v kategorii pádu tato analýza ihned zamítnuta. Při navázání slova *žlutá* analýza může pokračovat, omezení na vztahy mezi proměnnými si však vynutí "zpřesnit" nejen proměnnou `noun`, ale zpětně i proměnnou `adj1`. Nepřipadá totiž již v úvahu, aby slovo *jarní* v tomto kontextu (*jarní žlutá*) bylo např. mužského rodu. Stejně dojde k omezení i vlivem posledního slova, takže výsledně všechny tři proměnné ve své podsestavě `agr` ponesou již jen tyto možnosti:

$$\begin{bmatrix} \text{case} & \{\text{vok,aku,nom}\} \\ \text{gend} & \text{neut} \\ \text{num} & \text{pl} \end{bmatrix}$$

Stručně lze tedy shrnout: při ověřování podmínek jsou po každém navázání vstupní sestavy na některou z proměnných postupně omezovány i opakovaně *všechny* proměnné, do nichž se danými podmínkami přesnější hodnoty šíří.

5.5.4 Determinismus začátku a konce aplikace pravidel

Dosud jsme málo popsali nederministický chod pravidel. Uvedli jsme jen, že aplikace hledaného regulárního výrazu začne od libovolného místa ve vstupní posloupnosti slov, a v sekci 5.4.1 na str. 63 zavedli odlišení pro minimální resp. maximální dostupný počet opakování (značky $<$ a $>$ před operátorem opakování).

Systém tedy postupně probírá všechna možná místa zleva doprava, kde by bylo možné pravidlo aplikovat. Pokud uspěje, přidá k dosavadním výstupům výsledek aplikace pravidla od daného místa a zkouší aplikaci dále.

Chceme-li přijmout jen řešení od první úspěšné pozice, kterou při hledání zleva doprava systém najde, je třeba "nad" šipku pravidla vepsat požadavek *deterministického začátku pravidla*:

```
rule hvezdicka_misto_prvni_zavorky: ‘*‘ --detstart--> ‘(‘ end
```

Podobně se systém v základním nastavení nezastaví po první úspěšné aplikaci pravidla od dané pozice o dané délce, ale zkouší i další větve regulárního výrazu – např. větší či menší počet opakování podvýrazu – dokud nevyčerpá všechny možnosti.

Chceme-li přijmout jako jediné řešení od dané startovní pozice již první vhodný konec dosahu aplikace pravidla (s ohledem na přepínače < a > to může být nejkratší či nejdelší dostupná možnost), vepíšeme do šipky pravidla klíčové slovo `detend`.

Současně můžete zjednoznačnit začátek i konec pravidla šipkou ve tvaru `--detstart-detend-->` nebo ekvivalentně `-!->`.[3]

Jako příklad uveďme ještě pravidlo, které nahradí pouze poslední závorku hvězdičkou:

```
rule hvezdicka_misto_posledni_zavorky:
      \pred_zavorkou ‘*‘ --!--> ^ {pred_zavorkou: .>*} ‘)‘
end
```

Je však třeba zdůraznit, že v rámci fázového zpracování (viz 5.6 na následující straně) může být i pravidlo s deterministickým začátkem i koncem aplikováno postupně několikrát, takže v našem příkladu by, neuvedeme-li jinak, nakonec opakovaným provedením nahradilo všechny závorky hvězdičkami.

5.5.5 Proměnné vstupní, pracovní a výstupní

Až dosud jsme neuvažovali možnost *opakovaného navázání* jedné proměnné v regulárním výrazu na více vstupních sestav, které po sobě ve vstupu následují[4]. Na problém však narazíme, pokud podle zásad uvedených dosud zkusíme např. na vstup *velký černý pes* aplikovat pravidlo:

```
shortcuts
  noun = [cat-noun], adj = [cat-adj]
end
rule nounph ---> adj* noun ::
  adj.agr = noun.agr,  # ověřujme jmennou shodu
  nounph = noun        # naplňme výstupní sestavu
end
```

V prvním kroku se ověří, že *velký* odpovídá sestavě `[cat-agr]` a výsledek unifikace se uloží do proměnné `adj`. Pak jsou prověřeny podmínky aplikace a do proměnných `noun` (dosud obsahovala `[cat-noun]`) a `nounph` (dosud byla prázdná) se “přiunifikuje” hodnota podsestavy `agr` proměnné `adj`. Ve druhém

[3] V šipce je možné bez rozdílu uvádět jeden či více spojovníků.

[4] Tuto situaci je třeba nezaměňovat s opakovaným *pokusem* o navázání proměnné na různé vstupní sestavy při backtrackování regulárního výrazu.

kroku však proti intuici *selže* navázání vstupního slova *černý* na aktuální hodnotu proměnné `adj`, protože `adj` již nese nejen morfologické kategorie potřebné pro shodu, ale také lemma a formu slova *velký*!

Abychom v interpretaci pravidel následovali intuici, rozlišujeme proměnné *vstupní*, *pracovní* a *výstupní*.

Vstupní proměnná se vyskytuje ve vstupním <regulárním výrazu> a případně v <omezeních>. Nesmí se však vyskytnout v <náhradě>.

Výstupní proměnná se vyskytuje v <náhradě>, <omezeních> a případně v <regulárním výrazu>.

Pracovní proměnná se vyskytuje pouze v <omezeních> a není přítomna ani v regulárním výrazu, ani v <náhradě>.

V našem příkladu jsou tedy `adj` a `noun` proměnné vstupní a `nounph` je proměnná výstupní.

Rozdíl mezi vstupními a výstupními či pracovními proměnnými spočívá v navazování na vstupní sestavy. Po úspěšném navázání vstupní sestavy na počáteční hodnotu proměnné a propagaci omezení do ostatních proměnných je výsledek unifikace u *vstupních proměnných* zapomenut, aby byly proměnné "čerstvé" pro další navázání v rámci opakování regulárního výrazu. Po navázání vstupní sestavy na *pracovní* či *výstupní proměnnou* a propagaci omezení se výsledek unifikací naopak uloží jako nová hodnota proměnných.

V našem příkladu tedy intuitivně *velký* bude úspěšně navázáno na vstupní proměnnou `adj`. Po prověření (a uskladnění) morfologických kategorií jmenné shody bude proměnná `adj` opět vyčištěna na počáteční hodnotu `[cat-adj]` a i slovo *černý* se na ni podaří úspěšně navázat.

Pracovní proměnné jsou zavedeny pro případy, kdy je třeba v pravidle kontrolovat unifikací nějakou shodu, ale není žádoucí výsledek této unifikace nahlas uvést ve výstupu. Pokud bychom např. v našem příkladu trvali na první (neintuitivní) interpretaci i v novém schématu, musíme pro kontrolu *velký*≠*černý* zavést pracovní proměnnou `adj_storage`:

```
rule nounph ---> adj* noun ::
  adj.agr = noun.agr,
  adj = adj_storage,
  nounph = noun
end
```

5.6 Fáze běhu systému AX

Jak již bylo naznačeno v samém úvodu této kapitoly (viz str. 53), při běhu systému AX se podle dané gramatiky střídají fáze filtrace a fáze generování pomocí pravidel.

Přitom fáze filtrace je přirozeným dělítkem v gramatice. Maximální skupiny pravidel, mezi nimiž není uveden žádný filtr, pak tvoří fázi generování. Jako hranici mezi dvě fáze generování, nemají-li být odděleny filtrem, je možné vložit buď prázdný filtr `keep .* end`, nebo čáru tvořenou alespoň pěti spojovníky (`-----` či delší), která hranici vyznačuje rovněž.

Pro účely ladění je možné jako dělítko fáze použít též klíčové slovo `commit`. Tento příkaz nejen vytváří hranici fáze, ale také po skončení běhu předchozí fáze generování či filtrace ukončí celý běh gramatiky a vrátí celou množinu dosavadních přípustných čtení.

Fáze filtrace již byla popsána dostatečně přehledně v kapitole 5.4 na str. 62. Popišme nyní podrobně i fázi generování pomocí skupiny pravidel, kdy (jak zmiňují předešlé kapitoly) systém AX může intenzivně pracovat s nedeterminismem.

Příklad běhu systému AX lze najít v podobě ladicího výpisu v kapitole 5.5 na straně 81.

5.6.1 Pořadí aplikace pravidel v rámci fáze

Víme, že vstupem do fáze je množina přípustných čtení věty. Všechny prvky množiny jsou postupně předkládány jednotlivým pravidlům v rámci jedné fáze, a to v pořadí, v jakém jsou pravidla uvedena v gramatice.

Řekneme, že pravidla v rámci dané fáze jsou aplikována *v pevně daném pořadí* v případě, že vstup, který některé z pravidel již alespoň jednou úspěšně zpracovalo[5], není již dále prověřován žádným dalším pravidlem.

Naopak řekneme, že pravidla v rámci dané fáze jsou aplikována *v libovolném pořadí* v případě, že každý vstup je bez ohledu na úspěchy či neúspěchy předchozích pravidel předán i následujícím pravidlům dané fáze gramatiky.

V obou případech je *výstup* z úspěšné aplikace libovolného pravidla předán znovu na začátek fáze a je znovu (s ohledem na determinismus pořadí) zkoušen všemi dostupnými pravidly. (Je však možné omezit povolený počet opakování pravidla, viz níže.)

Není-li uvedeno jinak, jsou pravidla aplikována v pevně daném pořadí. Vyskytne-li se v rámci fáze mezi libovolnými pravidly, před prvním či za posledním pravidlem klíčové slovo `nondetorder`, budou pravidla této fáze aplikována v libovolném pořadí.

5.6.2 Deterministická a nedeterministická fáze

O fázi (tj. skupině pravidel) řekneme, že je *deterministická*, v případě, že žádný ze vstupů fáze, který byl kterýmkoli pravidlem či postupně více pravidly jakkoli úspěšně zpracován, nebude zahrnut do výsledků fáze. (Tj. fáze si je "jistá" náhradami, které provádí. Když nějakou náhradu provede, zahodí původní tvar věty.)

[5] Nezapomeňme možnost nedeterministického začátku a konce pravidel, viz 5.5.4 na str. 68.

O fázi řekneme, že je *nedeterministická*, v případě, že každý vstup i mezivýsledek fáze po aplikaci libovolné posloupnosti pravidel, bude zahrnut i mezi závěrečné výsledky fáze. Tento způsob aplikace pravidel fáze je vhodný, pokud si nejsme kvalitou pravidel jisti – víme, že pravidlo může provést redukci, která by se po ukončení fáze mohla ukázat jako nesprávná. V deterministické fázi by však již nebylo původní čtení věty k dispozici.

Není-li uvedeno jinak, je fáze deterministická. Vyskytne-li se v rámci fáze mezi libovolnými pravidly, před prvním či za posledním pravidlem klíčové slovo `nondetphase`, bude fáze prováděna nedeterministickým způsobem.

Je zřejmé, že nedeterministické pořadí aplikace pravidel i nedeterministická fáze výrazným způsobem zvyšují množství zpracovávaných čtení věty. Doporučujeme proto obě možnosti používat velmi střídmě a nedeterministické fáze ohraničit filtry, které počty vstupních i výstupních čtení významně sníží.

5.6.3 Omezení počtu opakování, ochrana před zacyklením

Vzhledem k tomu, že jsou pravidla v rámci jedné fáze aplikována opakovaně, hrozí při neopatrné formulaci pravidel gramatiky zacyklení běhu interpretu. Problém zastavení programu (Turingova stroje) je však algoritmicky nerozhodnutelný, a interpret proto nemůže zacyklení v obecném případě sám identifikovat.

Zacyklení gramatiky proto zabraňuje již kompilátor, a to poměrně jednoduchým způsobem: pokud by všechna pravidla vždy vstupní větu zkrátila, cyklický běh gramatiky se po jistém počtu kroků zastaví nad prázdným vstupem. Obecně však pravidla dovolují nejen větu nezkrátit, ale dokonce i libovolně prodlužovat. To je ovšem možné poznat již v průběhu kompilace. Kompilátor tedy nedovolí gramatiku spustit, pokud není u nezkracujícího pravidla s potenciálním rizikem zacyklení explicitně omezen povolený počet opakování.

Omezení počtu opakování se vyjadřuje poznámkou v šipce pravidla, podobně jako deterministický začátek a konec pravidla (viz 5.5.4 na straně 68. Povolený počet opakování je možné vyjádřit libovolným z následujících způsobů:

once	pravidlo lze použít v rámci celé fáze nejvýše jednou
times(X)	pravidlo lze použít nejvýše nX krát, kde n je počet slov vstupní věty
pow(X)	pravidlo lze použít nejvýše X^n krát
exp(X)	pravidlo lze použít nejvýše n^X krát

5.7 AX pro uživatele

5.7.1 Spuštění, vstup a výstup systému AX

Program AX je implementován pro prostředí OS typu Unix. Spouští se typicky na příkazové řádce zadáním příkazu:

`ax <gramatika.ax>`

Povinným argumentem programu je název souboru s gramatikou. Další

```
ID:  cmpr9410:009-p7s1
IN:  Máme zaměstnance , které občas vysíláme na služební cestu .
OUT: mít [lemma-"zaměstnanec",form-"zaměstnance",
          agr-[case-aku,gend-masc,num-pl],prep-"BLANK"]
OUT: vysílat [lemma-"který",form-"které",
              agr-[case-aku,gend-inanim;masc,num-pl]]
             občas
             [lemma-"cesta",form-"cestu",
              agr-[case-aku,gend-fem,num-sg],prep-"na"]
ID:  cmpr9410:013-p5s4
IN:  Přesně k tomu slouží naše rubrika .
OUT: sloužit Přesně
             [lemma-"ten",form-"tomu",
              agr-[case-dat,gend-neut;masc;inanim,num-sg],prep-"k"]
             [lemma-"rubrika",form-"rubrika",
              agr-[case-nom,gend-fem,num-sg,pers-first],prep-"BLANK"]
ID:  cmpr9410:028-p16s2
IN:  Ceny jejich obrazů šplhají do statisíců a dobře se prodávají v cizině .
OUT: prodávat dobře [lemma-"se",form-"se",agr-[case-aku]]
              [lemma-"cizina",form-"cizině",
               agr-[case-lok,gend-fem,num-sg],prep-"v"]
OUT: šplhat [lemma-"cena",form-"Ceny",
             agr-[case-nom,gend-fem,num-pl],prep-"BLANK"]
            [lemma-"stotisíc",form-"statisíců",
             agr-[case-gen,gend-inanim,num-pl],prep-"do"]
```

Obrázek 5.3: Ukázkový výstup programu AX. Zalomení a odsazení řádek přidáno pro přehlednost.

parametry jsou popsány níže.

Program očekává na standardní vstup posloupnost vstupních vět ve formátu CSTS, s tím omezením, že na každném řádku vstupu musí být uvedena právě jedna celá věta[6].

Výstup programu je v současném stavu vývoje pouze textový. Příklad výstupu je na obrázku 5.3. program pro každou vstupní větu, která nebyla během zpracování zamítnuta, vrací skupinu řádků

Pro každou vstupní větu, která nebyla během zpracování žádným filtrem zamítnuta, program vypíše její identifikátor (`ID`) a vstupní tvar (`IN`). Pak následují jednotlivá přípustná čtení věty, tj. prvky výsledné množiny čtení věty, vždy uvozená textem `OUT`. Výstupních čtení může být více, pokud to gramatika pro zpracování dovolila. Výstupní čtení jsou opět vytištěna ve stručné textové podobě. Detailní výpis by představovala posloupnost sestav rysů; v závěru gramatiky uvedené v příloze A na straně 90 je popsán technický trik, jak detailní výpis sestav získat.

Po svém spuštění provede AX nejprve kompilaci zadané gramatiky.[7] Pokud se gramatiku z nějakých důvodů nepodaří kompilovat, program oznámí důvod

[6] Tzn. každý řádek musí mít tvar: `<s id="..."><f>...<f>...<f>...`

[7] V průběhu kompilace může program též načíst morfologický slovník češtiny, a to v případě, že gramatika obsahuje sestavy rysů zadané vzorem slovního tvaru, viz 5.3.2 na str. 61.

```
% Informace o načítání
8157  Vstup
1002  Vstup-Selhal
162   Vstup-Selhal-Chyba ve formátu SGML.
840   Vstup-Selhal-Formát SGML v pořádku, neporozuměl CSTS.
7155  Vstup-V pořádku; načtena věta.

% Informace o zpracování
7155  Stats
1051  Stats-Odpovída
910   Stats-Odpovída-1
141   Stats-Odpovída-2
6104  Stats-Zamítnuto
30    Stats-Zamítnuto-apostrof
841   Stats-Zamítnuto-cislice
30    Stats-Zamítnuto-dvojtecka_ne_na_konci
14    Stats-Zamítnuto-lomitko
3360  Stats-Zamítnuto-nech_jen_H_VVH_nebo_HVV
9     Stats-Zamítnuto-neparova_zavorka
164   Stats-Zamítnuto-podezreni_NPgV
9     Stats-Zamítnuto-podezreni_PgAV
484   Stats-Zamítnuto-podezreni_VNPg
16    Stats-Zamítnuto-podezreni_VPgA
725   Stats-Zamítnuto-pomlcka
289   Stats-Zamítnuto-prilis_mnoho_partic_active
71    Stats-Zamítnuto-strednik_ne_na_konci
62    Stats-Zamítnuto-vyrad_vety_s_pasivnimi_tvary_sloves
```

Obrázek 5.4: Závěrečná statistika běhu programu AX.

chyby na standardní chybový výstup a ukončí svůj běh.

Po skončení běhu (po uzavření standardního vstupu a zpracování všech vět) program rovněž vypíše statistiku o tom, kolik vět bylo zpracováno a kolik vět bylo z jakých důvodů zamítnuto. Odděleně program uvádí přehled o tom, kolik vět se nepodařilo načíst z důvodu nesrovnalostí ve vstupním řádku, a odděleně přehled o tom, které filtry zamítly kolik vět. Obě statistiky jsou přitom mírně "hierarchické", tzn. že uvádějí nejen počet vět zamítnutých jednotlivými filtry, ale též souhrnný počet zamítnutých vět. V této hierarchii jsou roztříděny i přijaté věty, a to podle toho, kolik výstupních čtení se pro ně podařilo gramatikou získat. V našem příkladu bylo 910 vět přijato s jedním výstupním čtením, pro 141 vstupních vět se podařilo najít čtení dvě, celkem tedy bylo přijato 1051 vět.

5.7.2 Parametry na příkazové řádce

Program AX akceptuje tyto parametry:

`--help`
: Program vypíše stručnou nápovědu.

`--input=<manual|ambig|desam|a|b|...>`
Určuje, které vstupní morfologické značkování má program ze vstupního souboru ve formátu CSTS načítat (podrobnosti viz popis CSTS v rámci dokumentace PDT):

Hodnota	Volí lemma a značky
`manual`	<l> a <t>
`ambig`	<MMl> a <MMt> (výchozí nastavení)
`desam`	první dostupná značka <MDl> a <MDt>
<zdroj>	<MDl src="<zdroj>"> a <MDt src="<zdroj>">

`--verbose`
V průběhu kompilace jsou zpracovaná pravidla vypisována.

`--progress-ids`
Program bude na standardní chybový výstup postupně vypisovat ID vět, které právě začíná zpracovávat.

`--log=<název>`
Program bude ukládat ladicí výpis do souborů <název>.trex a <název>.graph; viz níže.

`--flush=<n>`
Ladicí výpis bude aktualizován každých <n> vět. (Výchozí hodnota je 10.)

`--debug`
Ladicí výpis bude aktualizován při *každém* kroku algoritmu. Chod systému bude proto velmi pomalý.

Další parametry programu nejsou určeny běžnému uživateli.

5.7.3 Ladění gramatiky pro systém AX

Stručné výpisy

K ladění jednoduchých problémů v gramatice pro jazyk AX obvykle postačí zapnout ladicí výpisy pro jednotlivá pravidla či filtry.

O ladicí výpis požádáme připsáním klíčového slova `debug` před filtr či pravidlo, např.:

```
debug rule ne_zcela_dobre_pravidlo: ...

debug keep az_sem_postoupilo_cteni_vety: .* end
```

Do standardního výstupu jsou pak *nad* klasické řádky `ID`, `IN` a `OUT` vypsána všechna čtení, která prověřovaný filtr zpracovával společne s poznámkou, zda čtení filtrem prošla nebo ne. Výpis u pravidel je proveden pouze tam, kde se sledované pravidlo podařilo aplikovat. Ve výpisu je vyznačen začátek i konec aplikace pravidla a rovněž výsledná náhrada.

Detailní ladicí výpis

S ohledem na potenciální složitost chodu programu AX a množství drobných chyb v pravidlech a filtrech gramatiky byl vybudován rovněž velmi podrobný systém ladicích výpisů o průběhu analýzy.

Je třeba důrazně varovat, že ladicí výpisy jsou *velmi* rozsáhlé. Podrobný ladicí výpis doporučujeme proto používat pouze na velmi malý počet vstupních vět, v nichž chceme hledat problém.

Celý systém ladicích výpisů je uveden do chodu, pokud spustíme AX s parametrem `--log=<logname>`. Detailní ladicí výpisy jsou postupně ukládány do souborů `<logname>.graph` a `<logname>.trex`. Frekvenci ukládání výpisů lze určit parametrem `--flush=<počet vět>`, implicitně je výpis uložen po každých 10 zpracovaných větách.

Ladicí výpis je rozdělen do dvou souborů, jelikož v úplnosti by byl velmi nepřehledný. Soubor `<logname>.graph` obsahuje výpis v interní reprezentaci orientovaného grafu.

Soubor `<logname>.trex` je určen pro uživatele. Otevřeme-li tento soubor v textovém editoru *vim*, který je řádně nastaven pro prohlížení souborů `*.trex` (viz příloha C na straně 100), můžeme několika stisky klávesy **Return** postupně rozbalit hierarchický pohled na průběh analýzy zpracovávaných vět, až k detailům aplikace jednotlivých pravidel. Ukázka záznamu je uvedena na obrázku 5.5 na str. 81.

5.8 Náměty na rozšíření

Tato kapitola stručně uvádí několik námětů, jak by bylo vhodné jazyk AX obohatit. V pozdějších verzích jazyka se tato rozšíření s nejvyšší pravděpodobností objeví.

5.8.1 Operace nad konstantními sestavami rysů

Vzhledem k možnosti zavedení zkratek a instantních sestav rysů se ukázalo jako účelné dát autoru gramatiky možnost manipulovat s konstatními sestavami rysů a "předpočítat" si z nich jiné sestavy.

Pro standardní operace nad konstantními sestavami rysů (viz 5.2.2 na straně 55) by tedy byly v syntaxi jazyka AX zavedeny tyto operátory:

Operátor	Provádí operaci
sestava `.` atribut	Deterministická extrakce atributu
atribut `-` sestava	Obalení sestavy
sestava `\` atribut	Odstranění všech výskytů atributu
sestava `/\` sestava	Unifikace
sestava `\/` sestava	Sloučení, merge

Operátory v tabulce jsou uvedeny s klesající prioritou. Je možné uzavírat výrazy nad konstatními sestavami rysů do kulatých závorek.

Například by takto bylo možné díky instantní sestavě rysů a operaci unifikace zavést velmi pohodlně zkratku pro sloveso *být* ve třetí osobě jednotného čísla přítomného času. Unifikaci je nutno použít, neboť slovo *je* může být chápáno též jako zájmeno.

```
shortcut je = ‘je‘ /\ [cat-verb]
```

Konstantní výrazy ze sestav rysů by bylo možné použít všude tam, kde je možné použít konstantní sestavy rysů. Výrazy by byly vyhodnoceny ještě v době kompilace. Pokud by byla požadována nemožná operace (např. unifikace sestav, které si neodpovídají), šlo by o chybu při kompilaci gramatiky.

Toto rozšíření bude do jazyka AX nepochybně zahrnuto, jakmile bude ze systému odstraněna samostatná část kompilace vstupní gramatiky pomocí externího modulu v Perlu (viz 5.9 na str. 79).

5.8.2 Plný repertoár řízení běhu

V průběhu práce se systémem AX se ukázalo, že zpracování vět v pevné sekvenci bloků nemusí být vždy pro autora gramatiky nejpohodlnější. Tato kapitola proto navrhuje, jak jazyk AX významně obohatit o plnou možnost kontroly řízení běhu gramatiky.

Základní myšlenka reprezentovat v každém kroku programu dosavadní přijatelná čtení věty jako množiny posloupností sestav rysů zůstává zachována i v novém návrhu, proti filtrům a pravidlům (jak byly popsány v kapitolách 5.4 na straně 62 a 5.5 na str. 64) je však přidána:

- možnost definice a volání podprocedur,
- možnost větvení gramatiky.

Implementace tohoto rozšíření není technicky příliš obtížná, rozšíření by však bylo vhodné spojit s “úklidem a očistou” syntaxe jazyka AX, a proto přesahuje možnosti této práce.

V nové definici by tedy platilo: *program* v jazyce AX ze vstupní množiny posloupností sestav rysů (čtení věty) vytvoří výstupní množinu posloupností sestav rysů. Pokud je výstupní množina čtení věty prázdná, říkáme, že program *selhal*. *Programem* přitom je:

- filtr (jak jej známe z kapitoly 5.4 na straně 62),
- skupina pravidel (jak ji známe z kapitoly 5.5 na str. 64),
- podprocedura,
- podmínka `if-then-else`.

Podprocedury

Podprocedura v jazyce AX je sekvence *programů*. Množina čtení věty vstupující do podprocedury je předána prvnímu z programů. Pokud program uspěje a vrátí nějaká výstupní čtení věty, jsou předána na vstup dalšímu v sekvenci programů.

Výstup posledního z programů je odevzdán jako výstup celé podprocedury. Podprocedura selže a vrátí prázdnou množinu výstupních čtení, pokud libovolný program ze sekvence selže.

Větvení gramatiky

Větvení typu `if-then-else` je programem jazyka AX. Jako podmínka pro větvení slouží regulární výraz nad sestavou rysů.[8]

Množina čtení věty vstupujících do podmínky `if-then-else` je rozdělena na ta čtení, která odpovídají regulárnímu výrazu v podmínce, a na ta, která tomuto výrazu neodpovídají. Čtení vyhovující podmínce jsou zpracována programem uvedeným ve větvi `then`, čtení nevyhovující podmínce jsou zpracována programem uvedeným ve větvi `else`. Výstupem celé konstrukce `if-then-else` je pak sjednocení výstupních čtení z větve `then` a větve `else`. Pokud větev `else` chybí, je výstupem sjednocení výsledků z větve `then` a vstupních čtení, která nevyhověla podmínce větvení (tj. jako by větev `else` se čteními neprovedla žádné úpravy).

5.8.3 Kontrola zachování závislostí

Z hlediska přípravy gramatik provádějících syntaktickou analýzu věty českého jazyka se nabízí zajímavá možnost: ověřovat správnost chodu gramatiky na větách, které již syntakticky analyzovány byly, totiž na větách PDT.

V duchu závislostních gramatik a redukční analýzy lze o konstrukci automatu na rozpoznání správné české věty uvažovat takto: V každém kroku redukční analýzy je možné ze vstupní věty odstranit pouze ta slova, na nichž žádné další části věty nezávisí, jinak by dosud platná věta jazyka byla redukcí převedena na větu, která součástí jazyka není.

Autor gramatiky systému AX při formulování redukčních pravidel typicky usiluje o zjednodušení vstupní věty, ale tak, aby neporušil její syntaktickou strukturu. Pokud vstupní data závislostní syntaktickou strukturu věty explicitně obsahují (jako PDT), mohl by systém AX po aplikaci každého pravidla ověřit, že výsledné čtení tuto strukturu neporušilo. Přesný postup kontroly zde nebudeme uvádět, podstata tkví v tom, že úpravu, kterou pravidlo provedlo na uzlech vstupní věty, musí být možné vysvětlit jako posloupnost kontrakcí hran syntaktického stromu věty. Syntaktická struktura věty by naopak byla porušena, pokud by pravidlo spojením dvou uzlů vytvořilo ve stromu věty cyklus. V takovém případě by systém mohl autorovi gramatiky hlásit varování

[8] Bylo by možné uvážovat i regulární výraz s omezujícími podmínkami, viz 5.5.2 na straně 66.

ve smyslu: úprava, kterou toto pravidlo na této větě provedlo, je v rozporu se syntaktickou strukturou věty.

5.9 Poznámky k implementaci

Systém AX je implementován kombinovaně v jazyce Perl a v jazyce Mercury (viz Somogyi, Henderson, and Conway (1995) a aktuální dokumentaci[9]).

Volba neprocedurálního programovacího jazyka Mercury pro podstatnou část systému AX se ukázala jako velmi šťastná. Mezi příznivé stránky tohoto jazyka patří především:

- propracovaný typový systém, typované prologovské termy, včetně typového polymorfismu,
- práce s pamětí zcela v režii kompilátoru,
- deklarativní způsob programování umožňuje psát funkce "reversibilní", kompilátor sám se postará o správné uspořádání volání tak, aby producent vždy předcházel konzumenta,
- vestavěný backtracking jako v prologu, ovšem s kontrolou determinismu ještě v době kompilace (predikáty jsou prověřovány, a pokud nemohou vracet více řešení, nikdy nejsou spouštěny opakovaně),
- programování vyššího řádu (předávání nejen hodnot, ale i funkcí),
- výjimky a možnost jejich bezpečného ošetření,
- přímá kompilace do jazyka C s možností přímé integrace s kódem napsaným v C/C++, a to včetně "nedeterminického kódu" (tj. funkcí vracejících postupně více různých řešení) v C i pro C.

Bohatý typový systém umožňuje pohodlné vytváření a velmi pohodlnou manipulaci i se značně složitými datovými strukturami. Důslednou typovou kontrolu programátor ocení rovněž v případě, že později navrhne např. mírně odlišnou definici základních datových struktur. Po změně definice typů kompilátor sadou chybových hlášek programátora doslova provede po *všech* místech v kódu, která změna zasáhla. Přirozená modularita prologových programů na druhou stranu napomáhá tomu, aby takových míst bylo v celém programu velmi málo.

Navíc striktní typová kontrola, kontrola determinismu a především úplná péče o paměť (programátor nic nealokuje a nedealokuje, všechny proměnné jsou lokální, na zásobníku volání predikátů) jsou vlastnosti, které prakticky znemožňují napsat havarující kód.

Syntaktická analýza vstupní gramatiky je v současné době zajišťována za pomoci modulu `Parse::RecDescent`[10] pro Perl. Pro jazyk Mercury existuje

[9] `http://www.cs.mu.oz.au/research/mercury/`

[10] `http://search.cpan.org/author/DCONWAY/Parse-RecDescent-1.80/lib/Parse/RecDescent.pod`

sice v současné době k dispozici použitelný zárodek syntaktického analyzátoru (program *moose*, analogický ke známému programu *bison*), jeho schopnosti a pohodlí pro programátora se však zatím s modulem `Parse::RecDescent` nedají srovnávat. Například lexikální analýzu zdrojového kódu by bylo nutné provádět ve vlastní režii, kdežto modul do Perlu umožňuje popsat tokeny pomocí regulárních výrazů přímo v gramatice.

Syntaktická analýza v Perlu tedy vstupní gramatiku předzpracuje a přepíše do syntaxe shodné se syntaxí ISO Prologu, kterou již Mercury umí přímo načítat do svých datových struktur.

V současném stavu vývoje systému je toto rozdělení na dva samostatné bloky již mírně na obtíž; například je nutné mezi oběma částmi předávat řadu přepínačů. Nejbližší změnou, kterou systém AX projde, bude proto právě reimplementace kompilátoru gramatiky.

```
graphfile:bug.graph
-a Běh zahájen: Thu Dec 12 23:00:16 2002
  -c Vstup cmpr9410:028-p6s2: Nedávno jsem byl ve starožitnictví a některé věci se mi tam líbily .
    -d Filtr: pomlcka: 1 --> 1
    -t Filtr: apostrof: 1 --> 1
    -v Filtr: lomitko: 1 --> 1
    -x Filtr: dvojtecka_ne_na_konci: 1 --> 1
    -z Filtr: zlomitko: 1 --> 1
    -a1 Filtr: strednik_ne_na_konci: 1 --> 1
    -a3 Filtr: cislice: 1 --> 1
    -a5 Výsledky fáze: 1 --> 1 (smaz_kulate_zavorky_kolem_cele_vety, smaz_hranate_zavorky_kolem_cele_...
    -eN Filtr: neparova_zavorka: 1 --> 1
    -eP Výsledky fáze: 1 --> 1 (smaz_uvozovku_na_zacatku, smaz_uvozovku_na_konci).
    -fr Filtr: uvozovka_uvnitr: 1 --> 1
    -ft Filtr: prilis_mnoho_partic_active: 1 --> 1
    -fv Výsledky fáze: 1 --> 1 (complex_pasv, complex_modal, complex_condit, ...).
      -fw Detaily vstupu
      -hz Pravidlo complex_pasv nebylo možno použít.
      -jJ Pravidlo complex_modal nebylo možno použít.
      -lU Pravidlo complex_condit nebylo možno použít.
      -mO Pravidlo complex_past_tense: -..----------  -->  -.----------
    -vg Výsledky fáze: 1 --> 1 (adv_coord_adv).
    -wm Výsledky fáze: 1 --> 1 (adj_coord_adj, make_nounph, make_pronph).
      -wn Detaily vstupu
      -xP Pravidlo adj_coord_adj nebylo možno použít.
      -yj Pravidlo make_nounph: --..--------  -->  --.--------
        -xe Detail aplikace pravidla make_nounph
        -yk Detail: Nedávno byl starožitnictví a některé věci se mi tam líbily .
        -zU Pravidlo adj_coord_adj nebylo možno použít.
        -a0w Pravidlo make_nounph: ----..-----  -->  ----.-----
          -zi Detail aplikace pravidla make_nounph
            -zj Pozice 0: Nedávno
            -a03 Pozice 1: byl
            -a0C Pozice 2: starožitnictví
            -a0L Pozice 3: a
            -a0U Pozice 4: některé
              -a0V Navazuji vstup 'některé' na proměnnou prep: [cat-prep]
              -a0X Navazuji vstup 'některé' na proměnnou adv: [cat-adv]
              -a0Z Navazuji vstup 'některé' na proměnnou adjpronnum: [cat-pron,morfcat-'da(morfcat(pron...
                -a0a První navázání uspělo: [cat-pron,morfcat-'da(morfcat(pron(indef)))',lemma-"některý...
                  -a0b Uspěla i revize vztahů mezi proměnnými, detail nových proměnných zde.
                  -a0i Navazuji vstup 'věci' na proměnnou adv: [cat-adv]
                  -a0k Navazuji vstup 'věci' na proměnnou adjpronnum: [cat-pron,morfcat-'da(morfcat(pro...
                  -a0m Navazuji vstup 'věci' na proměnnou noun: [cat-noun,agr-[case-aku,gend-fem,num-pl]]
              -a0v Nalezene nahrazuji: věci
              -a4a Navazuji vstup 'některé' na proměnnou prep: [cat-prep]
          -a0x Detail: Nedávno byl starožitnictví a věci se mi tam líbily .
          -a1d Pravidlo adj_coord_adj nebylo možno použít.
          -a3f Pravidlo make_nounph nebylo možno použít.
          -a4N Pravidlo make_pronph nebylo možno použít.
    -a4c Výsledky fáze: 1 --> 1 (oznac_nonprepgroup).
    -a5s Výsledky fáze: 1 --> 1 (zpatky_vrat_cat_nounph).
    -a78 Výsledky fáze: 1 --> 1 (noun_coord_noun, noun_gennounph).
    -a9S Filtr: nech_jen_H_VVH_nebo_HVV: 1 --> 1
    -a9U Filtr: PRED_PODEZRELYMI: 1 --> 1
    -a9W Filtr: podezreni_VNPg: 1 --> 1
    -a9Y Filtr: podezreni_NPgV: 1 --> 1
    -a9a Filtr: podezreni_PgAV: 1 --> 1
    -a9c Filtr: podezreni_VPgA: 1 --> 1
    -a9e Filtr: PO_PODEZRELYCH: 1 --> 1
    -a9g Výsledky fáze: 1 --> 2 (extrahuj_H_z_HV, extrahuj_V_z_HV, extrahuj_H_z_VH, ...).
    -aKF Filtr: nech_jen_klauze: 2 --> 2
    -aKI Filtr: vyrad_vety_s_pasivnimi_tvary_sloves: 2 --> 2
    -aKL Výsledky fáze: 2 --> 2 (1).
    -aMM Filtr: OCISTENO: 2 --> 2
    -aMP Výsledky fáze: 2 --> 2 (sloveso_dopredu, hezky_vyjadri_doplneni).
    -aP7 Výstupy: 2
-b Výstupy:
  -aPG 1: být Nedávno [lemma-"starožitnictví",form-"starožitnictví",agr-[case-lok,gend-neut,num-sg],pre...
  -aPH 1: líbit [lemma-"věc",form-"věci",agr-[case-aku,gend-fem,num-pl],prep-"BLANK"] [lemma-"se",form-...
```

Obrázek 5.5: Ukázka ladicího výpisu běhu systému AX.

Kapitola 6

Závěr

V této práci jsme studovali možnosti automatizovaného získávání lexikálně-syntaktických údajů z korpusových dat.

Ukázali jsme, že stávající korpusy češtinu i přes svůj rozsah nemusí pro specifické otázky poskytovat dat dostatek. Navrhli a implementovali jsme jednoduchý postup selektivního rozšiřování korpusů na základě textů dostupných v Internetu.

Pro otázky týkající se syntaxe přirozeného jazyka je žádoucí studovat data anotovaná na syntaktické rovině popisu. Ukázali jsme však, že stávající syntaktický korpus pro češtinu, Pražský závislostní korpus, poskytuje v současném rozsahu pro některé účely dat skutečně velmi málo; naše práce byla konkrétné zaměřena na získávání slovesných rámců.

V práci jsme proto vyhodnotili dostupné syntaktické analyzátory pro češtinu, které by bylo možné pro získání většího množství syntakticky anotovaných dat použít. Ukázali jsme, že v konkrétních případech je třeba měřit úspěšnost parserů podle vhodnějších kritérií, než je počet správně zapojených slov v závislostním rozboru věty. Konkrétně pro náš účel extrakce slovesných rámců je třeba uvádět pro nejlepší z dostupných parserů úspěšnost cca 55 % správně identifikovaných rámců, na rozdíl od typicky udávané úspěšnosti 70 až 83 % správně zapojených uzlů.

Abychom zlepšili úspěšnost stávajících parserů češtiny, navrhli a implementovali jsme vlastní systém pro formulaci lingvistických kritérií filtrace vstupních vět. Systém AX je obecný a lze jej použít k identifikaci vhodných příkladů pro libovolné lingvistické účely. Věty musejí být analyzovány na morfologické úrovni, zjednoznačnění morfologické informace není podmínkou. Systém AX je v práci představen formou uživatelské příručky.

Pro naše účely extrakce slovesných rámců jsme v tomto systému implementovali konkrétní filtr. Tímto filtrem projde přibližně 15 až 20 % vstupních vět. Stávající parsery jsme porovnali z hlediska úspěšnosti na filtrovaných větách a dosáhli zlepšení o 5 až 10 % v počtu správně zavěšených uzlů. Pro náš účel bylo podstatnější druhé kritérium, počet správně identifikovaných slovesných rámců, kde parsery dosáhly zlepšení o přibližně 10 až 15 %. Nejzajímavější přitom bylo

zlepšení úspěšnosti v počtu zcela správně analyzovaných vět: nejlepší z parserů zcela správně analyzuje více než 50 % vět, které prošly naší filtrací.

V další části práce jsme se zaměřili na problematiku identifikace povrchových rámců sloves na základě pozorovaných doplnění slovesa. Upozornili jsme na nespolehlivost statistických metod použitých pro tyto účely a navrhli zatím velmi jednoduchou variantu lingvisticky motivované filtrace a identifikace typů slovesných doplnění. Dále jsme studovali možnosti klasifikace pozorovaných slovesných rámců do hierarchie a navrhli vhodný pohled na tuto hierarchii.

Samostatná kapitola byla věnována otázce získávání valenčních rámců sloves. V této kapitole jsme především dokumentovali obtížnost této úlohy. Jedno z potřebných kritérií, test obligatornosti doplnění, není v současné formulaci vůbec možné strojově vyhodnotit. Druhé kritérium, odlišení aktantů a volných doplnění, lze vyhodnocovat jen velmi obtížně a závisí též na konkrétním studovaném slovese. (Např. nominativ hraje jednoznačně roli aktora jen u přibližně 79 % z 2 487 pozorovaných sloves, u cca 7 vystupuje i v roli volného doplnění.) Při automatickém získávání valenčních rámců bude proto nutné mít tato úskalí na paměti a studovat podrobněji i jednotlivé typy sloves. Jako podkladový materiál pro tuto kapitolu byly použity věty PDT, které jsou postupně anotovány na tektogramatickou rovinu jazykového popisu. Je možné, že se v anotaci dosud vyskytuje množství chyb; v takovém případě je však náš souhrnný pohled na doplnění sloves vhodné použít jako podnět k hledání chyb a případně korekci anotace.

Jak již bylo řečeno, podstatná část této práce je věnována popisu našeho systému AX, který slouží k filtraci vět přirozeného jazyka na základě lingvisticky motivovaných podmínek.

Předkládanou práci je vhodné považovat za první přiblížení k úkolu automatické extrakce lexikálně-syntaktických údajů z korpusů. Práce především mapuje problémy, které bude při automatickém získávání např. studovaných slovesných rámců třeba vyřešit. První náznaky řešení jsou představeny zde, pro praktické získávání rámců však řada významných a zajímavých otázek zůstává otevřena a bude třeba věnovat se jí i v dalším výzkumu.

Dodatek A

Identifikace velmi jednoduchých vět

Příloha obsahuje přesnou gramatiku pro systém AX, která byla použita pro identifikaci dostatečně jednoduchých vět, aby syntaktické analyzátory dokázaly ve větě poznat povrchový rámec slovesa dostatečně kvalitně. Lingvistický význam pravidel je popsán v kapitole 3.2 na str. 28, popis syntaxe jazyka AX je součástí kapitoly 5 na straně 53.

```
grammar

## Základní definice zkratek za slovní druhy, významná slova
# a symboly a některé neterminály

shortcuts
  noun = [cat-noun],
  adv = [cat-adv],
  adj = [cat-adj],
  prep = [cat-prep],
  aianinebo = [form-"a";"i";"ani";"nebo";"A";"I";"Ani";"Nebo";"či";"Či"],
  verb = [cat-verb;complexmodal;complexpassive;
          complexpast;complexcond;complexfut],
  comma = [form-","],
  punct = [form-","; ";"; "-"; ":"; "."; "?"; "!"],
  nounph = [cat-nounph],
  trace = [cat-trace]
end

################################################################
## Zamítnutí složité interpunkce a vět s číslicemi

shortcuts
  uvozovka = `"`,
  pomlcka = `-`,
  apostrof = `'`,
  lomitko = `/`,
  dvojtecka = `:`,
  zlomitko = `\`,
  strednik = `;`
end
```

```
filter pomlcka: .* pomlcka .* end
filter apostrof: .* apostrof .* end
filter lomitko: .* lomitko .* end
filter dvojtecka_ne_na_konci: .* dvojtecka $ end
filter zlomitko: .* zlomitko .* end
filter strednik_ne_na_konci: .* strednik $ end
filter cislice: ^.* [morfcat-'da(morfcat(numer(digits)))';
                            'da(morfcat(numer(roman)))'] .*$
end

##
# Vymazání textu v závorkách
# Odstranění párovych závorek

shortcuts
  openpar = [form-"("], openbr = [form-"["], opencur = [form-"{"],
  closepar = [form-")"], closebr = [form-"]"], closecur = [form-"}"]
end

rule smaz_kulate_zavorky_kolem_cele_vety: \contents ---!->
  ^ openpar {contents:.*} closepar $ end
rule smaz_hranate_zavorky_kolem_cele_vety: \contents ---!->
  ^ openbr {contents:.*} closebr $ end
rule smaz_slozene_zavorky_kolem_cele_vety: \contents ---!->
  ^ opencur {contents:.*} closecur $ end

rule delete_parentheses:  ---!->
    openpar !{openpar,closepar,openbr,closebr,opencur,closecur}<* closepar end
rule delete_brackets:     ---!->
    openbr !{openpar,closepar,openbr,closebr,opencur,closecur}<* closebr end
rule delete_braces:       ---!->
    opencur !{openpar,closepar,openbr,closebr,opencur,closecur}<* closecur end

##
# Zamítnutí vět s (nepárovými) závorkami

keep neparova_zavorka:
  ^ !{openpar, closepar, openbr, closebr, opencur, closecur}* $
end

##
# Zamítnutí vět s nepříjemnou uvozovkou

rule smaz_uvozovku_na_zacatku: --!-> ^ uvozovka end
rule smaz_uvozovku_na_konci: --!-> uvozovka $ end
filter uvozovka_uvnitr: .* uvozovka .* end

############################################################
## Složení analytických slovesných tvarů

shortcut particactive = [cat-verb, morfcat-'da(morfcat(verb(particactive)))']

## Z důvodu efektivity zamítneme věty s příliš mnoha určitými
# slovesy ihned.
filter prilis_mnoho_partic_active:
  ^ .* particactive .* particactive .* particactive .* $
end


## Spojování složených slovesných tvarů provádíme zcela deterministickou fází
## Jen počátky a konce aplikace pravidel dávají možnost volby.
# V pravidlech jsou komentářem vyznačena místa, kde je možné
```

```
# ponechat ve zpracovávané větě na místě zpracované části slovesa
# "stopu" (trace)

# pasivum
rule complex_pasv:
  complex \gap #trace
  ---->
     jsem {gap:!{verb, comma, aianinebo}*} zalit
   | zalit {gap:!{verb, comma, aianinebo}*} jsem ::
  zalit <- cat, morfcat, voice -> ‘zalit‘,
  jsem = [cat-verb;complexpast,
          lemma-"být",
          morfcat-’da(morfcat(verb(presfut)))’;
          ’da(morfcat(verb(particactive)))’],
  #jsem <- lemma -> ‘jsem‘,
  zalit.agr.num = jsem.agr.num,
  zalit <- agr.gend, agr.pers -> jsem,
  complex = [cat-complexpassive],
  complex <- lemma, form, mood -> zalit,
  complex.neg2 = zalit.neg,
  complex.neg = jsem.neg,
  complex <- agr.num, agr.gend, agr.pers, morfcat -> jsem
#  trace = [cat-trace, form-"XPASIVUMX"]
end

# modální slovesa
rule complex_modal:
  complex \gap #trace
  ---->
     muset {gap:!{verb, comma, aianinebo}*} prosit
   | prosit {gap:!{verb, comma, aianinebo}*} muset ::
  prosit <- morfcat -> ‘prosit‘, # musí to být infinitiv
  prosit = [cat-verb;complexpassive], # ale může to být infinitiv pasiva
                                  # muset být oholen
  muset = [cat-verb, lemma-"muset";"začít";"směť";"moci";"chtít"],
  complex = [cat-complexmodal],
  complex <- lemma, form -> prosit,
  complex <- agr, mood, neg, morfcat -> muset,
  complex.neg2 = prosit.neg
#  trace = [cat-trace, form-"XMODALX"]
end

# podmiňovací způsob
rule complex_condit:
  complex \gap #trace
  ---->
     zalil {gap:!{verb, comma, aianinebo}*} bych
   | bych {gap:!{verb, comma, aianinebo}*} zalil ::
  zalil <- morfcat,voice -> ‘zalil‘,
  bych <- cat, morfcat -> ‘bych‘,
  zalil = [cat-verb;complexpassive;complexmodal],
    # bych <zalil>; bych <musel zalít>; bych <byl zalit>
  complex = [cat-complexcond],
  complex <- lemma, form, neg, agr.num, agr.pers, voice -> zalil,
  complex <- agr, neg,agr.num,agr.pers -> bych
  #trace = [cat-trace, form-"XCONDITX"]
end

# složený minulý čas
rule complex_past_tense:
  complex \gap #trace
  ---->
```

```
     zalil {gap:!{verb,comma,aianinebo}*} jsem
   | jsem {gap:!{verb, comma, aianinebo}*} zalil
  ::
  zalil <- morfcat, voice -> ‘zalil‘,
  zalil = [cat-verb;complexpassive;complexmodal],
    # jsem <zalil>; jsem <byl zalit>; jsem <musel zalít>; jsem <musel být zalit>
  jsem.cat = ‘jsem‘.cat,
  jsem <- morfcat, lemma, neg -> ‘jsem‘,
  jsem.agr = [pers-first;second],
  zalil <- agr.num, agr.gend -> jsem,
  complex = [cat-complexpast],
  complex <- lemma, form, neg, morfcat -> zalil,
  complex <- agr.pers, agr.num, agr.gend, mood -> jsem
#  trace = [cat-trace, form-"XMINULY’Ja’TyX"]
end

# budoucí čas
rule complex_fut:
  complex \gap #trace
  ---->
     budu {gap:!{verb, comma, aianinebo}*} prosit
   | prosit {gap:!{verb, comma, aianinebo}*} budu ::
  prosit <- morfcat, neg -> ‘prosit‘,
  prosit = [cat-verb;complexmodal],
    # budu <prosit>; budu <muset prosit>
  budu.cat = ‘budu‘.cat,
  budu <- morfcat, voice, lemma -> ‘budu‘,
  complex = [cat-complexfut],
  complex <- lemma, form -> prosit,
  complex <- agr, voice, mood, neg -> budu
#  trace = [cat-trace, form-"XFUTURX"]
end



################################################################
## Zjednodušení jmenných a předložkových skupin příp. s koordinací

--------------

rule adv_coord_adv: outadv --!-> adv1 aianinebo adv2 ::
  adv1 = [cat-adv],
  adv2 = [cat-adv],
  outadv = adv1
end

----------------

rule adj_coord_adj: outadj --!-> adj1 (comma? aianinebo|comma) adj2 ::
  adj1 = [cat-adj], adj2 = [cat-adj],
  adj1 <- agr -> adj2,
  outadj = adj1
end

rule make_nounph: nounph -!-> prep?(adv{0,2}adjpronnum)+noun | prep noun ::
  adjpronnum = [cat-adj;num
               |cat-pron,
                  morfcat-’da(morfcat(pron(poss)))’;
                          ’da(morfcat(pron(poss_refl)))’;
                          ’da(morfcat(pron(ten)))’;
                          ’da(morfcat(pron(vsechen)))’;
                          ’da(morfcat(pron(neg)))’;
```

```
  'da(morfcat(pron(indef)))'],
  noun.agr = adjpronnum.agr,
  noun.agr = prep.agr,
  nounph <- lemma, form, agr -> noun,
  nounph.prep = prep.lemma
end

rule make_pronph: pronph -!-> prep pron ::
  pron = [cat-pron,morfcat-'da(morfcat(pron(pers)))';
                           'da(morfcat(pron(poss_refl)))';
                           'da(morfcat(pron(ten)))';
                           'da(morfcat(pron(indef)))';
                           'da(morfcat(pron(neg)))';
                           'da(morfcat(pron(vsechen)))'
   ],
  pron.agr = prep.agr,
  pronph = [cat-pronph],
  pronph <- morfcat, lemma, form, agr -> pron,
  pronph.prep = prep.lemma
end

------------
## Technický trik: Pro další zpracováni potřebujeme, aby v každé sestavě
# reprezentující zpracovanou jmennou či předložkovou skupinu, bylo
# v rámci jednoho atributu explicitně uvedeno, zda skupinu předcházela
# předložka, nebo ne. (Dosud máme přiznak pouze pozitivní: byla-li
# předložka, je v sestavě atribut prep s lemmatem předložky a atribut
# has_prep s hodnotou yes.)
# V této fázi projdeme všechny sestavy reprezentující jmenné či
# předložkové skupiny, a pokud se je podaří unifikovat se sestavou
# [prep-BLANK, has_prep-no], doplnili jsme potřebný přiznak.
# Aby pravidlo zpracovalo všechny sestavy (a ne několikrát opakovaně první
# sestavu), musíme při této úpravě rovněž změnit kategorii sestavy na
# nějakou odlišnou hodnotu. Proto následuje druhá fáze, která kategorii
# vrací zpět na nounph.

## Přidej příznak pro nepředložkové skupiny
rule oznac_nonprepgroup: nonprepgroup ---times(1)-!--> group ::
  group = [cat-nounph; noun; pronph],
  group = [prep-BLANK, has_prep-no],
  nonprepgroup = [cat-nonprepgroup],
  nonprepgroup <- morfcat, prep, agr, lemma, form -> group
end
----------
## Vrať zpátky nounph
rule zpatky_vrat_cat_nounph: nounph --times(1)-!-> nonprepgroup ::
  nonprepgroup <- morfcat, prep, agr, lemma, form, has_prep -> nounph,
  nonprepgroup = [cat-nonprepgroup]
end
------------

## Spojení koordinovaných jmenných či předložkových skupin.
# Opakovanou aplikací pravidlo zpracuje všechny členy koordinace.
rule noun_coord_noun: outnounph --!-> nounph1 (comma? aianinebo | comma) nounph2 ::
  nounph1 = [cat-nounph;pronph;noun],
  nounph2 = [cat-nounph;pronph;noun],
  outnounph = nounph1,
  nounph1.prep = nounph2.prep,
  nounph1.agr.case = nounph2.agr.case
end

## Spoj řetízek genitivních přívlastků s předcházející jmennou skupinou
```

```
# Opakovanou aplikací pravidlo zpracuje celý řetízek
rule noun_gennounph: outnounph --!-> innounph gennounph ::
  innounph = [cat-nounph; noun],
  outnounph = innounph,
  gennounph = [cat-nounph;noun,
               agr-[case-gen],
               prep-BLANK]
end


### Konec zjednodušování jmenných skupin



############################################################
### Identifikace VV/H nebo H\VV nebo H nebo H=H
### (v každé větě potřebujeme právě jedno sloveso)

shortcuts
  subord = [cat-conj, morfcat-'da(morfcat(conj(subord)))'],
  coord = [cat-conj, morfcat-'da(morfcat(conj(coord)))'],
  kdo = [cat-pron, morfcat-'da(morfcat(pron(kdo)))'],
  ktery = [cat-pron, morfcat-'da(morfcat(pron(jaky)))']
end

# Filtr rovněž zamítne věty, kde by byla nečekaná čárka
# Neřeší závěrečnou interpunkci věty
keep nech_jen_H_VVH_nebo_HVV:
    !{comma,coord,subord,verb,aianinebo}* verb !{comma,coord,subord,verb,aianinebo}*
  |
    !{comma,coord,subord,verb,aianinebo}* verb !{comma,coord,subord,verb,aianinebo}*
    ( comma? aianinebo
    | comma (
        subord
        | coord
        | prep? (kdo|ktery)
      )
    )
    !{comma,coord,subord,verb,aianinebo}* verb !{comma,coord,subord,verb,aianinebo}*
  |
    subord
    !{comma,coord,subord,verb,aianinebo}* verb !{comma,coord,subord,verb,aianinebo}*
    comma
    !{comma,coord,subord,verb,aianinebo}* verb !{comma,coord,subord,verb,aianinebo}*
end


############################################################
```

Na tomto místě byl běh gramatiky ukončen příkazem `commit` při identifikaci "velmi jednoduchých vět" bez ohledu na podezřelé slovosledné vzorce, viz 3.2.6 na str. 33.

```
############################################################
## Filtrace vět s podezřelými WOP

shortcuts
  noun = [cat-noun;nounph;pronph],
  prepgroup = [cat-nounph;pronph;noun,has_prep-yes]
end

filter podezreni_VNPg:  .* verb noun prepgroup .* end
filter podezreni_NPgV:  .* noun prepgroup verb .* end
```

```
filter podezreni_PgAV:  .* prepgroup adj verb .* end
filter podezreni_VPgA:  .* verb adj prepgroup .* end


############################################################
```

Zde byl běh gramatiky ukončen příkazem `commit` při identifikaci "velmi jednoduchých vět" včetně filtrace vět s podezřelými slovoslednými vzorci.

Následuje ukázková gramatika, která extrahuje pozorované rámce přímo v systému AX a neopírá se o další zpracování úplnými parsery. V práci tento dodatek nebyl využit, neboť by ještě bylo třeba gramatiky prověřit a zpřesnit z lingvistického hlediska. Pravidla jsou však zajímavou ukázkou technik práce v jazyce AX.

```
############################################################
------------------------------
## Nedeterministické pořadí aplikace pravidel.
# Vstupní souvětí je jednotlivými pravidly zpracováno, některá pravidla
# nahradí souvětí jen první z vět, některá je nahradí jen druhou z vět.
# V závěru fáze tak díky nedeterministickému pořadí aplikace získáme
# jako samostatná čtení věty všechny klauze věty.

# Pravidla navíc na standardní výstup zvýrazňují, kde byly jednotlivé
# klauze identifikovány (viz klíčové slovo debug).

nondetrules

debug rule extrahuj_H_z_HV:
  subord -!--> comma subord !verb* verb !verb* $
end
debug rule extrahuj_V_z_HV:
  \vedlejsi -!--> ^ {hlavni: .*} comma subord { vedlejsi: .*} $
end

debug rule extrahuj_H_z_VH:
  subord -!--> ^ subord !verb* verb !verb* comma
end
debug rule extrahuj_V_z_VH:
  \vedlejsi -!--> ^ subord {vedlejsi: .*} comma { hlavni: .*} $
end

debug rule extrahuj_vztaznou_V_z_HV:
  \vedlejsi -!--> ^ {hlavni: .*} comma { vedlejsi: (prep?(kdo|ktery)) .*} $
end
debug rule extrahuj_H_pri_vztazne_V_z_HV:
  \hlavni -!--> ^ {hlavni: .*} comma { vedlejsi: (prep?(kdo|ktery)) .*} $
end

debug rule extrahuj_vztaznou_V_z_VH:
  \vedlejsi -!--> ^ { vedlejsi: (prep?(kdo|ktery)) .*} comma {hlavni: .*} $
end
debug rule extrahuj_H_pri_vztazne_V_z_VH:
  \hlavni -!--> ^ { vedlejsi: (prep?(kdo|ktery)) .*} comma {hlavni: .*} $
end

debug rule extrahuj_H1_z_HH:
  \hlavni1 -!-> ^ {hlavni1: .*} comma? (coord|aianinebo) {hlavni2: .* } $
end
debug rule extrahuj_H2_z_HH:
  \hlavni2 -!-> ^ {hlavni1: .*} comma? (coord|aianinebo) {hlavni2: .* } $
end
```

```
filter nech_jen_klauze: .* (comma|coord|aianinebo) .* end


----------------------
# Nyní z každé klauze extrahujme pozorovaný rámec aktivních tvarů sloves.
# Nejprve zamítneme klauze s pasivem a dále přemístíme sloveso na začatek
# posloupnosti sestav rysů ve čtení věty. V úplném závěru vypíšeme pro
# jmenné a předložkové skupiny získanou informaci detailně.

debug filter vyrad_vety_s_pasivnimi_tvary_sloves:
          .*
          [cat-verb;complexmodal;complexpassive;
   complexpast;complexcond;complexfut,
   morfcat-'da(morfcat(verb(transgrpast)))';
                  'da(morfcat(verb(particpassive)))']
          .*
end

# Očista od zbytečností
rule ---> `.` | `!` | `?` end

------------

rule sloveso_dopredu: infin \1 --!-once-> {.*} verb ::
  infin.form = verb.lemma,
  infin = [cat-infin]
end

## Trik pri vypsání detailu pro jmenné skupiny:
# Systém ax pro každou výstupní sestavu vypisuje hodnotu v atributu form.
# V následujícim pravidle proto místo každé sestavy reprezentující
# jmennou či předložkovou skupinu připravíme sestavu, kde v atributu form
# umístíme všechny sledované atributy.
# Proměnné v pravidle mají tento význam:
#   watch ... spatřená sestava jmenné či předložkové skupiny
#   observe ... pracovní sestava, shromažď'uje, co nás o skupině zajímá
#   dump ... výstupní sestava, dump.form = observe

rule hezky_vyjadri_doplneni: dump ---!-times(1)-> watch ::
  watch = [cat-pron;noun;nounph;pronph], # sleduj tyto kategorie slov
  observe.form = watch.form,
  observe.agr = watch.agr,
  observe <- lemma, prep -> watch,
    # shromazduj o nich uvedene atributy
  dump.form = observe, # a atributy nakonec hezky vytiskni
  dump = [cat-dump] # zajisti unikatni kategorii, abys nedumpoval cyklicky
end



end
## Konec gramatiky pro extrakci slovesných doplnění z velmi jednoduchých vět
#################################################################
```

Dodatek B

Formy a funkce slovesných doplnění

Na základě cca 26 000 vět, které byly dosud anotovány na tektogramatickou rovinu v připravovaném PDT verze 2, byly připraveny následující přehledné tabulky nejčastějších forem a funkcí slovesných doplnění. Z vět byly vynechány všechny tvary slovesa *být* a též všechny pasivní tvary sloves, neboť pasivní tvary sloves mají typicky jinou strukturu doplnění. Rovněž byla ve větách správně ošetřena koordinace a v přehledu jsou zahrnuta i společná rozvití koordinovaných sloves.

V tabulkách značka *#číslo* znamená pád číslo. Značka VV značí vedlejší větu.

B.1 Od funkce k formě

Nejprve jsou uvedeny tabulky aktantů a dále pak abecedně tříděné tabulky volných doplnění. V tabulkách nejsou zahrnuta ta doplnění, kde si anotátoři volbou funkce nebyli jisti. (V pracovní verzi anotace se více možných anotací vyskytuje.

Formy ACT	Zastoupení	Výskytů
#1	86,5 %	18 450
infin	3,5 %	743
#2	2,3 %	486
VV	2,0 %	436
#X	1,9 %	412
#3	0,9 %	198
o#4	0,9 %	185
#4	0,7 %	142
k#3	0,5 %	113
jako#1	0,2 %	47
než#1	0,2 %	43
Ostatní (21 forem)	0,3 %	66

Formy ADDR	Zastoupení	Výskytů
#3	55,5 %	898
#4	14,6 %	236
s#7	10,0 %	162
si	7,7 %	124
#X	5,8 %	93
na#4	1,2 %	20
#2	1,2 %	19
se	0,7 %	11
nad#7	0,5 %	8
na#6	0,4 %	7
#1	0,4 %	6
Ostatní (15 forem)	2,0 %	33

Formy EFF	Zastoupení	Výskytů
VV	56,6 %	1 104
#4	14,9 %	290
infin	9,9 %	193
za#4	8,9 %	174
#7	2,0 %	39
na#4	1,3 %	26
#1	1,0 %	20
s#7	1,0 %	19
do#2	0,6 %	12
na#2	0,6 %	12
k#3	0,4 %	8
Ostatní (17 forem)	2,7 %	52

Formy ORIG	Zastoupení	Výskytů
od#2	46,4 %	70
z#2	36,4 %	55
na#6	6,6 %	10
po#6	4,0 %	6
než#2	1,3 %	2
#X	1,3 %	2
#2	0,7 %	1
z#X	0,7 %	1
#3	0,7 %	1
VV	0,7 %	1
od#X	0,7 %	1
Ostatní (1 forem)	0,7 %	1

Formy PAT	Zastoupení	Výskytů
#4	47,3 %	8 631
#7	8,4 %	1 525
#1	7,8 %	1 417
VV	7,4 %	1 345
infin	6,4 %	1 167
#2	5,2 %	943
#3	2,6 %	465
o#6	2,2 %	393
na#4	2,1 %	389
s#7	1,7 %	314
k#3	1,5 %	266
Ostatní (60 forem)	7,5 %	1 376

Formy ???	Zastoupení	Výskytů
se	46,2 %	4 209
že	16,8 %	1 527
VV	13,9 %	1 267
aby	4,6 %	420
si	4,1 %	375
když	2,8 %	257
protože	1,8 %	161
pokud	1,2 %	112
zda	1,2 %	110
než-2	0,9 %	80
kdyby	0,7 %	68
Ostatní (67 forem)	5,8 %	527

Formy ACMP	Zastoupení	Výskytů
s#7	62,1 %	343
bez#2	16,7 %	92
spolu_s#7	4,7 %	26
souvislosti_s#7	2,7 %	15
společně_s#7	1,6 %	9
VV	1,4 %	8
včetně#2	1,1 %	6
s#X	0,9 %	5
Spolu_s#7	0,9 %	5
sebou	0,7 %	4
v_spolupráce_s#7	0,7 %	4
Ostatní (25 forem)	6,3 %	35

Formy ADVS	Zastoupení	Výskytů
ale	100,0 %	1

Formy AIM	Zastoupení	Výskytů
k#3	30,6 %	98
VV	30,0 %	96
na#4	16,6 %	53
pro#4	13,1 %	42
infin	5,3 %	17
v_zájmu#2	0,6 %	2
v_zájem#2	0,6 %	2
kvůli#3	0,6 %	2
#3	0,3 %	1
za#4	0,3 %	1
na#6	0,3 %	1
Ostatní (5 forem)	1,6 %	5

Formy APP	Zastoupení	Výskytů
#2	40,0 %	2
#X	20,0 %	1
infin	20,0 %	1
#1	20,0 %	1

Formy APPS	Zastoupení	Výskytů
jako	32,5 %	26
tj-1	13,8 %	11
#1	12,5 %	10
a-1	11,2 %	9
že	7,5 %	6
čili-1	5,0 %	4
neboli	3,8 %	3
když	2,5 %	2
až-2	2,5 %	2
ať	1,2 %	1
aby	1,2 %	1
Ostatní (5 forem)	6,2 %	5

Formy ATT	Zastoupení	Výskytů
v#6	62,1 %	41
ovšem-1	10,6 %	7
k#3	9,1 %	6
s#7	4,5 %	3
naproti#3	3,0 %	2
#1	3,0 %	2
#7	1,5 %	1
vždyť	1,5 %	1
infin	1,5 %	1
de#X	1,5 %	1
bez#2	1,5 %	1

Formy BEN	Zastoupení	Výskytů
pro#4	37,4 %	293
si	24,9 %	195
#3	23,6 %	185
proti#3	5,7 %	45
pro#X	2,4 %	19
ve_prospěch#2	1,0 %	8
sebe	0,9 %	7
v#4	0,8 %	6
rozporu_se#7	0,4 %	3
rozporu_s#7	0,4 %	3
v_neprospěch#2	0,3 %	2
Ostatní (14 forem)	2,2 %	17

Formy CAUS	Zastoupení	Výskytů
VV	35,1 %	206
díky#3	10,6 %	62
z#2	9,9 %	58
za#4	8,9 %	52
kvůli#3	6,8 %	40
na#4	3,7 %	22
#7	3,7 %	22
pro#4	3,4 %	20
proto-1	2,7 %	16
infin	1,7 %	10
v_důsledku#2	1,2 %	7
Ostatní (36 forem)	12,3 %	72

Formy CNCS	Zastoupení	Výskytů
VV	76,4 %	168
přes#4	13,6 %	30
infin	3,2 %	7
navzdory#3	3,2 %	7
při#6	1,4 %	3
#3	0,9 %	2
#4	0,5 %	1
I_přes#4	0,5 %	1
proti#3	0,5 %	1

Formy COMPL	Zastoupení	Výskytů
#1	48,0 %	170
jako#1	18,1 %	64
#4	12,4 %	44
VV	11,6 %	41
jako#4	6,5 %	23
coby#1	0,6 %	2
jako#2	0,6 %	2
v#6	0,6 %	2
coby#4	0,3 %	1
infin	0,3 %	1
jako#3	0,3 %	1
Ostatní (3 forem)	0,8 %	3

Formy COND	Zastoupení	Výskytů
VV	50,9 %	255
v#6	16,4 %	82
při#6	11,2 %	56
za#2	9,8 %	49
infin	5,8 %	29
v_případě#2	2,0 %	10
pod#7	0,8 %	4
na#4	0,8 %	4
V_případě#2	0,6 %	3
#6	0,4 %	2
v_případ#1	0,2 %	1
Ostatní (6 forem)	1,2 %	6

Formy CONJ	Zastoupení	Výskytů
a-1	100,0 %	13

Formy CPR	Zastoupení	Výskytů
rozdíl_od#2	19,1 %	22
oproti#3	17,4 %	20
srovnání_s#7	16,5 %	19
proti#3	13,0 %	15
naproti#3	4,3 %	5
jako#1	4,3 %	5
VV	4,3 %	5
srovnání_se#7	4,3 %	5
v_porovnání_s#7	1,7 %	2
porovnání_s#7	1,7 %	2
#1	1,7 %	2
Ostatní (11 forem)	11,3 %	13

Formy CRIT	Zastoupení	Výskytů
podle#2	95,3 %	588
podle#X	2,1 %	13
VV	1,1 %	7
podle#1	0,3 %	2
dle#2	0,3 %	2
#7	0,2 %	1
podle#4	0,2 %	1
#2	0,2 %	1
podle#6	0,2 %	1
kolem#7	0,2 %	1

Formy CSQ	Zastoupení	Výskytů
a-1	66,7 %	2
VV	33,3 %	1

Formy CTERF	Zastoupení	Výskytů
VV	82,3 %	51
infin	14,5 %	9
zatímco	3,2 %	2

Formy DENOM	Zastoupení	Výskytů
#1	60,0 %	3
#7	20,0 %	1
navzdory#3	20,0 %	1

Formy DIFF	Zastoupení	Výskytů
o#2	65,7 %	69
o#4	23,8 %	25
#7	5,7 %	6
#2	1,9 %	2
#X	1,0 %	1
o#X	1,0 %	1
na#2	1,0 %	1

Formy DIR1	Zastoupení	Výskytů
z#2	89,5 %	400
od#2	6,5 %	29
z#X	0,9 %	4
#7	0,7 %	3
od#X	0,4 %	2
sebe	0,4 %	2
mimo#4	0,4 %	2
k#3	0,2 %	1
než#2	0,2 %	1
zpoza#2	0,2 %	1
při#6	0,2 %	1
Ostatní (1 forem)	0,2 %	1

Formy DIR2	Zastoupení	Výskytů
#7	60,6 %	60
přes#4	15,2 %	15
po#6	11,1 %	11
skrz#4	2,0 %	2
napříč#7	2,0 %	2
mezi#7	2,0 %	2
směrem_k#X	1,0 %	1
mimo#4	1,0 %	1
před#7	1,0 %	1
přes#X	1,0 %	1
nad#7	1,0 %	1
Ostatní (2 forem)	2,0 %	2

Formy DIR3	Zastoupení	Výskytů
do#2	48,7 %	629
na#4	21,6 %	279
k#3	16,3 %	210
mezi#4	2,7 %	35
do#X	2,4 %	31
pod#4	1,3 %	17
#2	0,9 %	11
za#4	0,7 %	9
za#7	0,6 %	8
na#X	0,5 %	6
před#4	0,5 %	6
Ostatní (26 forem)	3,9 %	51

Formy DISJ	Zastoupení	Výskytů
nebo	100,0 %	2

Formy DPHR	Zastoupení	Výskytů
k#3	18,9 %	53
na#4	11,0 %	31
na#6	10,3 %	29
v#6	10,0 %	28
#4	9,3 %	26
za#4	7,8 %	22
v#4	4,6 %	13
#7	3,2 %	9
infin	2,5 %	7
z#2	2,5 %	7
do#2	2,5 %	7
Ostatní (23 forem)	17,4 %	49

Formy ETHD	Zastoupení	Výskytů
si	82,4 %	14
#3	17,6 %	3

Formy EXT	Zastoupení	Výskytů
#2	31,3 %	63
v#6	12,4 %	25
do#2	10,0 %	20
#X	9,5 %	19
z#2	7,0 %	14
#4	6,5 %	13
#7	4,5 %	9
kolem#2	2,5 %	5
na#4	2,5 %	5
i-1	2,5 %	5
za#4	2,0 %	4
Ostatní (14 forem)	9,5 %	19

Formy HER	Zastoupení	Výskytů
po#6	50,0 %	1
po#X	50,0 %	1

Formy ID	Zastoupení	Výskytů
#1	100,0 %	2

Formy INTF	Zastoupení	Výskytů
#1	67,3 %	35
#4	26,9 %	14
že	3,8 %	2
když#1	1,9 %	1

Formy INTT	Zastoupení	Výskytů
infin	49,2 %	32
na#4	36,9 %	24
k#3	3,1 %	2
na#6	3,1 %	2
VV	3,1 %	2
pro#4	1,5 %	1
na#X	1,5 %	1
do#2	1,5 %	1

Formy LOC	Zastoupení	Výskytů
v#6	60,3 %	2 433
na#6	21,0 %	848
u#2	5,3 %	213
v#X	2,4 %	96
mezi#7	2,0 %	82
před#7	1,4 %	55
za#7	0,9 %	35
na#X	0,6 %	24
pod#7	0,5 %	20
kolem#2	0,5 %	19
sebou	0,4 %	18
Ostatní (53 forem)	4,7 %	190

Formy MANN	Zastoupení	Výskytů
#7	30,3 %	132
v#6	17,7 %	77
VV	9,9 %	43
pod#7	5,7 %	25
na#4	5,3 %	23
#1	3,7 %	16
jak-2	2,8 %	12
na#6	2,3 %	10
s#7	2,1 %	9
infin	1,8 %	8
z#2	1,8 %	8
Ostatní (32 forem)	16,7 %	73

Formy MAT	Zastoupení	Výskytů
#2	100,0 %	1

Formy MEANS	Zastoupení	Výskytů
#7	86,7 %	618
VV	2,8 %	20
prostřednictví#2	2,2 %	16
na#6	1,0 %	7
za#2	0,8 %	6
na#4	0,7 %	5
pomocí#2	0,7 %	5
přes#4	0,7 %	5
#2	0,6 %	4
za#4	0,6 %	4
o#4	0,3 %	2
Ostatní (17 forem)	2,9 %	21

Formy MOD	Zastoupení	Výskytů
v#6	66,7 %	4
do#2	16,7 %	1
s#7	16,7 %	1

Formy NORM	Zastoupení	Výskytů
podle#2	41,9 %	44
na_základě#2	25,7 %	27
souladu_s#7	7,6 %	8
Na_základě#2	6,7 %	7
shodě_s#7	2,9 %	3
podle#X	2,9 %	3
v_světlo#2	1,9 %	2
#2	1,9 %	2
na#6	1,9 %	2
dle#2	1,9 %	2
souladu_se#7	1,9 %	2
Ostatní (3 forem)	2,9 %	3

Formy OPER	Zastoupení	Výskytů
až-1	55,6 %	5
až-2	44,4 %	4

Formy PAR	Zastoupení	Výskytů
VV	78,0 %	259
#1	9,3 %	31
#X	6,3 %	21
infin	2,7 %	9
#2	0,9 %	3
#7	0,6 %	2
rozdíl_ode#2	0,6 %	2
podle#2	0,3 %	1
za#2	0,3 %	1
jak-2	0,3 %	1
#4	0,3 %	1
Ostatní (1 forem)	0,3 %	1

Formy PREC	Zastoupení	Výskytů
však	37,9 %	572
ale	12,1 %	183
a-1	10,9 %	164
ovšem-1	6,8 %	103
proto-1	6,4 %	97
pokud	3,4 %	51
když	3,2 %	49
tak-2	2,5 %	37
jenže	1,8 %	27
nicméně	1,7 %	26
protože	1,5 %	23
Ostatní (35 forem)	11,7 %	176

Formy PRED	Zastoupení	Výskytů
VV	52,8 %	142
infin	47,2 %	127

Formy REG	Zastoupení	Výskytů
v#6	24,9 %	98
z#2	5,8 %	23
na#6	5,6 %	22
VV	5,3 %	21
vzhledem_k#3	5,1 %	20
na#4	4,8 %	19
v_rámci#2	4,8 %	19
u#2	4,3 %	17
k#3	3,6 %	14
pro#4	2,8 %	11
Vzhledem_k#3	2,8 %	11
Ostatní (43 forem)	30,2 %	119

Formy RESL	Zastoupení	Výskytů
VV	33,3 %	9
na#2	22,2 %	6
infin	14,8 %	4
k#3	11,1 %	3
na#4	11,1 %	3
#7	3,7 %	1
#4	3,7 %	1

Formy RESTR	Zastoupení	Výskytů
kromě#2	65,2 %	86
vedle#2	21,2 %	28
s_výjimkou#2	4,5 %	6
infin	2,3 %	3
na#4	2,3 %	3
mimo#4	1,5 %	2
vyjma#2	1,5 %	2
VV	0,8 %	1
#2	0,8 %	1

Formy RHEM	Zastoupení	Výskytů
i-1	79,5 %	786
ani	18,3 %	181
až-2	0,7 %	7
až-1	0,6 %	6
než-2	0,5 %	5
ovšem-1	0,2 %	2
ale	0,1 %	1
#7	0,1 %	1

Formy RSTR	Zastoupení	Výskytů
VV	66,4 %	77
#1	11,2 %	13
#2	8,6 %	10
infin	6,9 %	8
#4	3,4 %	4
#6	0,9 %	1
jako#1	0,9 %	1
kromě#2	0,9 %	1
o#6	0,9 %	1

Formy SUBS	Zastoupení	Výskytů
za#4	41,4 %	24
místo#2	24,1 %	14
namísto#2	6,9 %	4
za#X	5,2 %	3
VV	5,2 %	3
za#2	1,7 %	1
jméno#2	1,7 %	1
na_místo#2	1,7 %	1
místo_na#4	1,7 %	1
místo#7	1,7 %	1
/#X	1,7 %	1
Ostatní (4 forem)	6,9 %	4

Formy TFHL	Zastoupení	Výskytů
na#4	40,0 %	20
pro#4	22,0 %	11
po#4	18,0 %	9
na#2	14,0 %	7
#2	4,0 %	2
pro#2	2,0 %	1

Formy TFRWH	Zastoupení	Výskytů
z#2	87,5 %	7
od#2	12,5 %	1

Formy THL	Zastoupení	Výskytů
#4	34,0 %	89
#2	26,3 %	69
za#4	16,8 %	44
za#2	6,1 %	16
po#4	5,3 %	14
VV	1,9 %	5
po#6	1,9 %	5
přes#4	1,9 %	5
po#2	1,1 %	3
než#2	0,8 %	2
přes#2	0,8 %	2
Ostatní (8 forem)	3,1 %	8

Formy THO	Zastoupení	Výskytů
#4	58,8 %	20
v#6	23,5 %	8
#7	5,9 %	2
VV	5,9 %	2
do#2	2,9 %	1
#2	2,9 %	1

Formy TILL	Zastoupení	Výskytů
do#2	100,0 %	4

Formy TOWH	Zastoupení	Výskytů
na#4	67,9 %	19
na#2	17,9 %	5
do#2	7,1 %	2
#4	7,1 %	2

Formy TPAR	Zastoupení	Výskytů
během#2	51,3 %	78
za#2	11,2 %	17
za#4	5,3 %	8
v_průběhu#2	4,6 %	7
VV	3,9 %	6
v#6	3,3 %	5
o#6	2,6 %	4
pod#7	2,6 %	4
V_průběhu#2	2,6 %	4
přes#4	2,6 %	4
s#7	1,3 %	2
Ostatní (10 forem)	8,6 %	13

Formy TSIN	Zastoupení	Výskytů
od#2	94,0 %	219
od#X	3,9 %	9
z#2	1,7 %	4
počínat#7	0,4 %	1

Formy TTILL	Zastoupení	Výskytů
do#2	90,0 %	135
VV	5,3 %	8
na#4	1,3 %	2
k#3	0,7 %	1
#2	0,7 %	1
po#6	0,7 %	1
infin	0,7 %	1
do#X	0,7 %	1

Formy TWHEN	Zastoupení	Výskytů
v#6	33,8 %	981
po#6	15,5 %	449
v#4	8,0 %	233
při#6	7,8 %	226
VV	6,4 %	186
před#7	6,0 %	175
#2	4,8 %	140
#4	4,5 %	130
na#6	1,7 %	49
za#2	1,1 %	33
konec#2	1,1 %	33
Ostatní (59 forem)	9,1 %	265

Formy VOC	Zastoupení	Výskytů
#5	100,0 %	3

Formy VOCAT	Zastoupení	Výskytů
#5	70,0 %	7
#1	30,0 %	3

B.2 Od formy k funkci

Tabulky pro jednotlivé formy jsou utříděny sestupně podle celkové četnosti výskytů. Uvádíme jen 30 nejčastějších forem, konkrétně:

Forma	Výskytů	Forma	Výskytů
#1	20 206	#4	9 672
VV	5 812	se	4 353
v#6	4 015	#7	2 515
infin	2 402	#2	1 799
#3	1 768	že	1 542
na#6	1 165	na#4	1 007
s#7	885	do#2	876
k#3	814	i-1	798
z#2	756	si	715
#X	687	podle#2	642
však	575	po#6	527
o#4	485	za#4	441
aby	435	o#6	430
pro#4	397	od#2	359
když	308	při#6	299

Funkce #1	Zastoupení	Výskytů
ACT	91,3 %	18 450
PAT	7,0 %	1 417
COMPL	0,8 %	170
INTF	0,2 %	35
PAR	0,2 %	31
EFF	0,1 %	20
MANN	0,1 %	16
RSTR	0,1 %	13
APPS	0,0 %	10
???	0,0 %	9
ADDR	0,0 %	6
Ostatní (15 funkcí)	0,1 %	29

Funkce #4	Zastoupení	Výskytů
PAT	89,2 %	8 631
EFF	3,0 %	290
ADDR	2,4 %	236
ACT	1,5 %	142
TWHEN	1,3 %	130
THL	0,9 %	89
COMPL	0,5 %	44
DPHR	0,3 %	26
THO	0,2 %	20
INTF	0,1 %	14
EXT	0,1 %	13
Ostatní (17 funkcí)	0,4 %	37

Funkce VV	Zastoupení	Výskytů
PAT	23,1 %	1 345
???	21,8 %	1 267
EFF	19,0 %	1 104
ACT	7,5 %	436
PAR	4,5 %	259
COND	4,4 %	255
CAUS	3,5 %	206
TWHEN	3,2 %	186
CNCS	2,9 %	168
PRED	2,4 %	142
AIM	1,7 %	96
Ostatní (36 funkcí)	6,0 %	348

Funkce se	Zastoupení	Výskytů
???	96,7 %	4 209
PAT	2,9 %	126
ADDR	0,3 %	11
ACT	0,1 %	3
DPHR	0,1 %	3
PAT—???	0,0 %	1

Funkce v#6	Zastoupení	Výskytů
LOC	60,6 %	2 433
TWHEN	24,4 %	981
REG	2,4 %	98
PAT	2,0 %	82
COND	2,0 %	82
MANN	1,9 %	77
LOC—???	1,5 %	59
ATT	1,0 %	41
DPHR	0,7 %	28
EXT	0,6 %	25
MANN—???	0,5 %	21
Ostatní (31 funkcí)	2,2 %	88

Funkce #7	Zastoupení	Výskytů
PAT	60,6 %	1 525
MEANS	24,6 %	618
MANN	5,2 %	132
DIR2	2,4 %	60
EFF	1,6 %	39
CAUS	0,9 %	22
TWHEN	0,9 %	22
???	0,4 %	10
DPHR	0,4 %	9
EXT	0,4 %	9
REG	0,3 %	8
Ostatní (30 funkcí)	2,4 %	61

Funkce infin	Zastoupení	Výskytů
PAT	48,6 %	1 167
ACT	30,9 %	743
EFF	8,0 %	193
PRED	5,3 %	127
INTT	1,3 %	32
COND	1,2 %	29
AIM	0,7 %	17
CAUS	0,4 %	10
CTERF	0,4 %	9
PAR	0,4 %	9
MANN	0,3 %	8
Ostatní (20 funkcí)	2,4 %	58

Funkce #2	Zastoupení	Výskytů
PAT	52,4 %	943
ACT	27,0 %	486
TWHEN	7,8 %	140
THL	3,8 %	69
EXT	3,5 %	63
ADDR	1,1 %	19
DIR3	0,6 %	11
RSTR	0,6 %	10
LOC	0,4 %	8
EFF	0,4 %	7
MEANS	0,2 %	4
Ostatní (26 funkcí)	2,2 %	39

Funkce #3	Zastoupení	Výskytů
ADDR	50,8 %	898
PAT	26,3 %	465
ACT	11,2 %	198
BEN	10,5 %	185
ACT—???	0,2 %	4
ETHD	0,2 %	3
CAUS	0,2 %	3
PAT—???	0,1 %	2
ADDR—???	0,1 %	2
CNCS	0,1 %	2
ORIG	0,1 %	1
Ostatní (5 funkcí)	0,3 %	5

Funkce že	Zastoupení	Výskytů
???	99,0 %	1 527
PREC	0,5 %	7
APPS	0,4 %	6
INTF	0,1 %	2

Funkce na#6	Zastoupení	Výskytů
LOC	72,8 %	848
PAT	12,3 %	143
TWHEN	4,2 %	49
DPHR	2,5 %	29
REG	1,9 %	22
MANN	0,9 %	10
ORIG	0,9 %	10
LOC—???	0,9 %	10
ADDR	0,6 %	7
MEANS	0,6 %	7
PREC	0,5 %	6
Ostatní (13 funkcí)	2,1 %	24

Funkce na#4	Zastoupení	Výskytů
PAT	38,6 %	389
DIR3	27,7 %	279
AIM	5,3 %	53
DPHR	3,1 %	31
EFF	2,6 %	26
INTT	2,4 %	24
TWHEN	2,4 %	24
MANN	2,3 %	23
CAUS	2,2 %	22
TFHL	2,0 %	20
ADDR	2,0 %	20
Ostatní (26 funkcí)	9,5 %	96

Funkce s#7	Zastoupení	Výskytů
ACMP	38,8 %	343
PAT	35,5 %	314
ADDR	18,3 %	162
EFF	2,1 %	19
MANN	1,0 %	9
PAT—???	0,8 %	7
TWHEN	0,6 %	5
???	0,6 %	5
DPHR	0,5 %	4
ATT	0,3 %	3
TPAR	0,2 %	2
Ostatní (9 funkcí)	1,4 %	12

Funkce do#2	Zastoupení	Výskytů
DIR3	71,8 %	629
TTILL	15,4 %	135
PAT	4,8 %	42
EXT	2,3 %	20
EFF	1,4 %	12
MANN	0,8 %	7
DPHR	0,8 %	7
TILL	0,5 %	4
DIR3—???	0,5 %	4
MEANS	0,2 %	2
TOWH	0,2 %	2
Ostatní (10 funkcí)	1,4 %	12

Funkce k#3	Zastoupení	Výskytů
PAT	32,7 %	266
DIR3	25,8 %	210
ACT	13,9 %	113
AIM	12,0 %	98
DPHR	6,5 %	53
TWHEN	2,6 %	21
REG	1,7 %	14
EFF	1,0 %	8
ATT	0,7 %	6
PAT—???	0,5 %	4
???	0,4 %	3
Ostatní (11 funkcí)	2,2 %	18

Funkce i-1	Zastoupení	Výskytů
RHEM	98,5 %	786
???	0,8 %	6
EXT	0,6 %	5
PREC	0,1 %	1

Funkce z#2	Zastoupení	Výskytů
DIR1	52,9 %	400
PAT	21,7 %	164
CAUS	7,7 %	58
ORIG	7,3 %	55
REG	3,0 %	23
EXT	1,9 %	14
MANN	1,1 %	8
TFRWH	0,9 %	7
DPHR	0,9 %	7
TSIN	0,5 %	4
DIR1—???	0,4 %	3
Ostatní (9 funkcí)	1,7 %	13

Funkce si	Zastoupení	Výskytů
???	52,4 %	375
BEN	27,3 %	195
ADDR	17,3 %	124
ETHD	2,0 %	14
DPHR	0,4 %	3
BEN—???	0,3 %	2
PAT	0,1 %	1
ADDR—???	0,1 %	1

Funkce #X	Zastoupení	Výskytů
ACT	60,0 %	412
PAT	17,5 %	120
ADDR	13,5 %	93
PAR	3,1 %	21
EXT	2,8 %	19
LOC	0,6 %	4
???	0,4 %	3
EFF	0,3 %	2
ORIG	0,3 %	2
APP	0,1 %	1
TWHEN	0,1 %	1
Ostatní (9 funkcí)	1,3 %	9

Funkce podle#2	Zastoupení	Výskytů
CRIT	91,6 %	588
NORM	6,9 %	44
MANN	1,1 %	7
CRIT—NORM	0,2 %	1
PAR	0,2 %	1
COND	0,2 %	1

Funkce však	Zastoupení	Výskytů
PREC	99,5 %	572
???	0,5 %	3

Funkce o#4	Zastoupení	Výskytů
PAT	54,0 %	262
ACT	38,1 %	185
DIFF	5,2 %	25
EFF	0,8 %	4
???	0,8 %	4
MEANS	0,4 %	2
MANN	0,4 %	2
TWHEN	0,2 %	1

Funkce aby	Zastoupení	Výskytů
???	96,6 %	420
PREC	3,2 %	14
APPS	0,2 %	1

Funkce pro#4	Zastoupení	Výskytů
BEN	73,8 %	293
AIM	10,6 %	42
CAUS	5,0 %	20
REG	2,8 %	11
TFHL	2,8 %	11
PAT	2,5 %	10
ADDR	0,5 %	2
CAUS—???	0,3 %	1
???	0,3 %	1
BEN—???	0,3 %	1
AIM——	0,3 %	1
Ostatní (4 funkcí)	1,0 %	4

Funkce když	Zastoupení	Výskytů
???	83,4 %	257
PREC	15,9 %	49
APPS	0,6 %	2

Funkce po#6	Zastoupení	Výskytů
TWHEN	85,2 %	449
PAT	6,1 %	32
LOC	2,1 %	11
DIR2	2,1 %	11
ORIG	1,1 %	6
THL	0,9 %	5
MANN	0,6 %	3
CPR	0,2 %	1
TTILL	0,2 %	1
DPHR	0,2 %	1
EXT	0,2 %	1
Ostatní (6 funkcí)	1,1 %	6

Funkce za#4	Zastoupení	Výskytů
EFF	39,5 %	174
PAT	15,2 %	67
CAUS	11,8 %	52
THL	10,0 %	44
SUBS	5,4 %	24
DPHR	5,0 %	22
TWHEN	2,3 %	10
DIR3	2,0 %	9
TPAR	1,8 %	8
MANN	1,1 %	5
MEANS	0,9 %	4
Ostatní (14 funkcí)	5,0 %	22

Funkce o#6	Zastoupení	Výskytů
PAT	91,4 %	393
TWHEN	4,9 %	21
TPAR	0,9 %	4
REG	0,9 %	4
???	0,7 %	3
DPHR	0,2 %	1
PAT—???	0,2 %	1
COMPL	0,2 %	1
RSTR	0,2 %	1
ACT	0,2 %	1

Funkce od#2	Zastoupení	Výskytů
TSIN	61,0 %	219
ORIG	19,5 %	70
PAT	8,6 %	31
DIR1	8,1 %	29
TWHEN	1,1 %	4
MANN	0,3 %	1
TFRWH	0,3 %	1
CAUS	0,3 %	1
???	0,3 %	1
TPAR	0,3 %	1
DPHR	0,3 %	1

Funkce při#6	Zastoupení	Výskytů
TWHEN	75,6 %	226
COND	18,7 %	56
LOC	1,0 %	3
CNCS	1,0 %	3
TWHEN—COND	0,7 %	2
LOC—???	0,7 %	2
TPAR	0,7 %	2
REG	0,7 %	2
COND—???	0,3 %	1
DIR1	0,3 %	1
COND—TWHEN	0,3 %	1

Dodatek C

Prohlížení orientovaných grafů, program TREX

Program TREX byl připraven pro účely pohodlného prohlížení orientovaných grafů v obyčejném textovém editoru (např. *vim* nebo *emacs*). Název je zkratkou za "tree explorer", neboť výpis z grafu je zásadně stromovitý.

Uzly orientovaného grafu jsou označeny textově, posloupností číslic a písmen; vyloučeny jsou mezery i všechny další symboly. Navíc všechny uzly nesou libovolnou textovou informaci, která se rozumně vejde na jeden řádek. (Textová hodnota uzlů tedy nesmí obsahovat znak konce řádku.) Uzly jsou propojeny orientovanými hranami, na hranách již žádná informace uložena není. (Hrana je tedy uspořádaná dvojice identifikátorů uzlů, které propojuje.)

Orientovaný graf, který chceme procházet, je uložen ve speciálním formátu souboru `*.graph`. Soubory tohoto typu umí vytvářet sám TREX (viz níže), nebo jsou vedlejším výstupem jiných programů, např. ladicí výpisy programu AX (viz 5.5 na straně 81).

Soubor `*.graph` je vždy doplněn "startovním" souborem `*.trex`. Tento soubor je určen k prohlížení v textovém editoru. Soubor `*.trex` obsahuje na svém prvním řádku odkaz na datový soubor s grafem, a dále obsahuje libovolný z uzlů grafu, od něhož chceme graf postupně "rozbalovat".

Je-li textový editor vybaven správnými makry pro práci se souborem `*.trex` (viz obrázek C.1 na následující straně pro makra vhodná pro editor *vim*), stačí na zvoleném uzlu stisknout klávesu **Return**, a editor s pomocí programu TREX uzel "rozbalí", tj. připíše k uzlu všechny uzly sousedící po hranách s daným uzlem.

Zdrojový kód programu TREX i potřebná makra pro editor *vim* jsou k dispozici na CD, které je přílohou této práce.

Makro pro "rozbalení" další úrovně hierarchie (uveďte na jediném řádku):

```
map <CR> ^r+0i>><ESC>
         :%!env MERCURY_OPTIONS="--detstack-size 40000" trex<CR>
         />><CR>2x<ESC>
```

Makro napřed na aktuálním řádku znakem + naznačí, že má být tento uzel expandován, dále si na řádku pomocí znaků >> poznačí aktuální pozici kurzoru. Takto upravený soubor je systémovou rourou předán programu TREX a opět načten. Nakonec editor umístí kurzor na řádek, kde zanechal značku >> a značku odstraní. Program TREX zajistil expanzi označených uzlů.
Jiné makro umožňuje expandované uzly pohodlně opět zabalit. Makro řádky pouze označí jako určené k zabalení a teprve při nejbližší expanzi nějakého uzlu program TREX provede všechny požadované úpravy:
`map <F2> ^r.`

Obrázek C.1: Makra pro práci se soubory `*.trex` v editoru *vim*.

C.1 Vlastní orientované grafy

Kromě základní funkce sloužit textovému editoru v rozbalování uzlů grafu program TREX umožňuje snadno připravit vlastní datový soubor s libovolným orientovaným grafem.

Při spuštění programu TREX s parametrem `--make` <`soubor`> bude vytvořena dvojice souborů <`soubor`>`.graph` a <`soubor`>`.trex` pro prohlížení zadaného orientovaného grafu. Při této variantě spuštění TREX očekává na standardní vstup zadání grafu ve tvaru:

```
uzel1: Textová informace v uzlu jedna
uzel2: Textová informace v uzlu dva
uzel3: Textová informace v uzlu tři
uzel1=uzel2
uzel2=uzel3
```

Jednotlivé řádky buď nesou informaci o uzlu (identifikátor uzlu, dvojtečka, textová hodnota uzlu), nebo nesou informaci o hraně grafu (výchozí uzel, rovnítko, cílový uzel). Program TREX odmítne popis grafu a ohlásí chyby, pokud zadání nebude odpovídat tomuto formátu.

C.2 Současná omezení

Ve stávající jednoduché implementaci mezi nejvýznamnější omezení patří především:

Nemožnost procházení grafem pozpátku. Tzn. že při rozbalování uzlů postupujeme vždy ve směru orientace hran. Po hranách se nevracíme.

Načítání datového souboru při každé expanzi uzlu. Program TREX je zatím textovým editorem spouštěn při každé expanzi uzlu znovu. TREX tedy musí vždy načíst celý datový soubor s grafem, aby dokázal k uzlu

najít a vypsat sousedy. Pro malé grafy toto implementační omezení nepředstavuje problém, prohlížení větších grafů je ovšem zdlouhavé a velmi rozsáhlé grafy se nemusí podařit prohlédnout vůbec. Problém pomalého prohlížení by bylo možné elegantně vyřešit při lepší spolupráci s editorem; TREX by byl po celou dobu běhu editoru připraven a graf by načetl pouze jednou. Např. v editoru *vim* by to však vyžadovalo implementovat TREX jako dvojici programů typu klient-server. V editoru *emacs* by stačilo připravit speciální makra.

Literatura

Basili, Roberto and Michele Vindigni. 1998. Adapting a Subcategorization Lexicon to a Domain. In *Proceedings of the ECML'98 Workshop TANLPS: Towards adaptive NLP-driven systems: linguistic information, learning methods and applications*, Chemnitz, Germany, April.

Böhmová, Alena, Jan Hajič, Eva Hajičová, and Barbora Hladká. 2001. The Prague Dependency Treebank: Three-Level Annotation Scenario. In Anne Abeillé, editor, *Treebanks: Building and Using Syntactically Annotated Corpora*. Kluwer Academic Publishers.

Brent, M. 1991. Automatic acquisition of subcategorization frames from untagged text. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 209–214, Berkeley, CA.

Briscoe, E. J. and J. Carroll. 1997. Automatic Extraction of Subcategorization from Corpora. In *Proceedings of the 5th ACL Conf. on Applied Nat. Lg. Proc.*, pages 356–363, Washington, DC.

Collins, Michael. 1996. A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 184–191.

Collins, Michael. 1997. Three Generative, Lexicalised Models for Statistical Parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (jointly with the 8th Conference of the EACL)*, Madrid.

Collins, Michael, Jan Hajič, Eric Brill, Lance Ramshaw, and Christoph Tillmann. 1999. A Statistical Parser of Czech. In *Proceedings of 37th ACL Conference*, pages 505–512, University of Maryland, College Park, USA.

Hajič, Jan. 1998. Building a Syntactically Annotated Corpus: The Prague Dependency Treebank. In Eva Hajičová, editor, *Issues of Valency and Meaning. Studies in Honor of Jarmila Panevová*. Prague Karolinum, Charles University Press, pages 12–19.

Hajič, Jan et al. 1997. A Manual for Analytic Layer Tagging of the Prague Dependency Treebank. Technical Report TR-1997-03, ÚFAL MFF UK, Prague, Czech Republic. In Czech.

Hajič, Jan and Barbora Hladká. 1997. Probabilistic and Rule-Based Tagger of an Inflective Language - a Comparison. In *Proceedings of the 5th ANLP Conference*, pages 111–118, Washington, USA.

Hajič, Jan and Barbora Hladká. 1998a. Czech Language Processing - POS Tagging. In *Proceedings of the First LREC Conference*, pages 931–936, Granada, Spain.

Hajič, Jan and Barbora Hladká. 1998b. Tagging Inflective Languages: Prediction of Morphological Categories for a Rich, Structured Tagset. In *Proceedings of COLING-ACL Conference*, pages 483–490, Montreal, Canada.

Hajič, Jan, Pavel Krbec, Pavel Květoň, Karel Oliva, and Vladimír Petkevič. 2001. Serial Combination of Rules and Statistics: A Case Study in Czech Tagging. In *Proceedings of ACL Conference*, Toulouse, France.

Hajičová, Eva, Jarmila Panevová, and Petr Sgall. 1999. Manuál pro tektogramatické značkování. Technical Report TR-1999-07, ÚFAL MFF UK, Prague, Czech Republic.

Holan, T., V. Kuboň, K. Oliva, and M. Plátek. 1998. Two Useful Measures of Word Order Complexity. In A. Polguere and S. Kahane, editors, *Proceedings of the Coling '98 Workshop: Processing of Dependency-Based Grammars*, Montreal. University of Montreal.

Holan, T., V. Kuboň, K. Oliva, and M. Plátek. 2001. Complexity of Word Order. *Special Issue on Dependency Grammar of the journal TAL (Traitement Automatique des Langues)*, 41(1):273–300.

Holan, Tomáš. 2001. *Nástroj pro vývoj závislostních analyzátorů přirozených jazyků s volným slovosledem.* Ph.D. thesis, Charles University, Prague. In Czech.

Honetschläger, Václav. 2002. Analytical and Tectogrammatical Syntactical Parsing. Charles University, Prague. WDS.

Horák, Aleš. 1998. Verb valency and semantic classification of verbs. In Petr Sojka, Václav Matoušek, Karel Pala, and Ivan Kopeček, editors, *Text, Speech and Dialogue - TSD 98*, Brno.

Jančar, P., F. Mráz, , M. Plátek, and J. Vogel. 1995. Restarting automata. In *FCT'95*, volume 965 LNCS, pages 283–292. Springer, August.

Jančar, P., F. Mráz, , M. Plátek, and J. Vogel. 1996. Restarting automata with rewriting. In K. G. Jeffery, J. Král, and M. Bartošek, editors, *Theory and Practice of Informatics, SOFSEM'96*, volume 1175 LNCS, pages 401–408, Milovy, Czech Republic, November. Springer.

Karcevskij, Sergej. 1929. Du dualisme asymétrique du signe linguistique (Asymetrický dualismus lingvistického znaku). *TCLP*, 1:88–93.

Kocek, Jan, Marie Kopřivová, and Karel Kučera, editors. 2000. *Český národní korpus - úvod a příručka uživatele.* FF UK - ÚČNK, Praha.

Korhonen, Anna, Genevieve Gorrell, and Diana McCarthy. 2000. Statistical Filtering and Subcategorization Frame Acquisition. In *Proc. of SIGDAT EMNLP and Very Large Corpora*, Hong Kong, China.

Kuboň, Vladislav. 2001. *Robust parser for Czech.* Ph.D. thesis, Charles University, Prague.

Kunze, J. 1972. *Die Auslassbarkeit von Satzteilen bei koordinativen Verbindungen im Deutschen.* Akademie-Verlag, Berlin.

Manning, Christopher D. 1993. Automatic acquisition of a large subcategorization dictionary from corpora. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 235–242, Columbus, Ohio.

Mel'čuk, Igor A. 1988. *Dependency Syntax - Theory and Practice.* Albany: State University of New York Press.

Pala, K., P. Rychlý, and P. Smrž. 1997. DESAM - Annotated Corpus for Czech. In *Proceedings of SOFSEM'97*, New York, Hamburg. Springer Verlag.

Pala, Karel and Pavel Ševeček. 1997. Valence českých sloves. In *Sborník prací FFBU*, pages 41–54, Brno.

Panevová, Jarmila. 1980. *Formy a funkce ve stavbě české věty [Forms and functions in the structure of the Czech sentence].* Academia, Prague, Czech Republic.

Penn, Gerald. 2000. *The Algebraic Structure of Attributed Type Signatures.* Ph.D. thesis, School of Computer Science, Carnegie Mellon University.

Plátek, M. 1999. Strict monotonicity and restarting automata. *Prague Bulletin of Mathematical Linguistics*, 72:11–27.

Pollard, C. and I. Sag. 1994. *Head-driven Phrase Structure Grammar.* Chicago.

Sarkar, Anoop and Daniel Zeman. 2000. Automatic Extraction of Subcategorization Frames for Czech. In *Proceedings of the 18th International Conference on Computational Linguistics (Coling 2000)*, Saarbrücken, Germany. Universität des Saarlandes.

Sgall, P. and J. Panevová. 1990. Dependency syntax - a challenge. *Theoretical Linguistics*, 15(1/2):213–232, Jan.

Sgall, Petr. 1967. *Generativní popis jazyka a česká deklinace.* Academia, Prague, Czech Republic.

Sgall, Petr, Eva Hajičová, and Jarmila Panevová. 1986. *The Meaning of the Sentence and Its Semantic and Pragmatic Aspects.* Academia/Reidel Publishing Company, Prague, Czech Republic/Dordrecht, Netherlands.

Skoumalová, Hana. 2001. *Czech syntactic lexicon.* Ph.D. thesis, Univerzita Karlova, Filozofická fakulta.

Somogyi, Zoltan, Fergus Henderson, and Thomas Conway. 1995. Mercury: an efficient purely declarative logic programming language. In *Proceedings of the Australian Computer Science Conference*, pages 499–512, Glenelg, Australia, February.

Straňáková-Lopatková, Markéta. 2001. Homonymie předložkových skupin v češtině a možnost jejich automatického zpracování (Ambiguity of prepositional phrases in Czech and possibilities for automatic treatment). Technical Report TR-2001-11, ÚFAL/CKL, Prague, Czech Republic. In Czech.

Straňáková-Lopatková, Markéta and Zdeněk Žabokrtský. 2002. Valency Dictionary of Czech Verbs: Complex Tectogrammatical Annotation. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC 2002)*, volume 3, pages 949–956. ELRA.

Tesnière, L. 1959. *Eléments de Syntaxe Structurale.* Klincksieck, Paris. 2. edition (1969).

Žabokrtský, Zdeněk, Václava Benešová, Markéta Lopatková, and Karolina Skwarská. 2002. Tektogramaticky anotovaný valenční slovník českých sloves. Technical Report TR-2002-15, ÚFAL/CKL, Prague, Czech Republic.

Žabokrtský, Zdeněk, Sašo Džeroski, and Petr Sgall. 2002. A Machine Learning Approach to Automatic Functor Assignment in the Prague Dependency Treebank. In M. González Rodríguez and C. Paz Suárez Araujo, editors, *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC 2002)*, volume 5, pages 1513–1520, Las Palmas de Gran Canaria, Spain. ELRA. 405/96/K214 (GAČR), CKL - LN00A063 (MŠMT).

Zeman, Daniel. 1997. A Statistical Parser of Czech. Master's thesis, ÚFAL, MFF UK, Prague, Czech Republic. In Czech.

Zeman, Daniel. 2001a. How Much Will a RE-based Preprocessor Help a Statistical Parser? In *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT 2001)*, number ISBN 7-302-04925-4, Beijing Daxue, Beijing, Čína. Tsinghua University Press.

Zeman, Daniel. 2001b. Parsing with Regular Expressions: A Minute to Learn, a Lifetime to Master. *Prague Bulletin of Mathematical Linguistics*, 75:29–37.

Zeman, Daniel. 2002. Can Subcategorization Help a Statistical Parser? In *Proceedings of the 19th International Conference on Computational Linguistics (Coling 2002)*, Taibei, Tchaj-wan. Zhongyang Yanjiuyuan (Academia Sinica).