AX – Systém pro automatizovanou extrakci lexikálně-syntaktických údajů z korpusu.

Ondřej Bojar
obo@cuni.cz

Abstrakt

Systém AX je určen ke zpracování morfologicky analyzovaných vět přirozeného jazyka s cílem věty syntakticky předzpracovat a vybrat pouze ty, které jsou dostatečně jednoduché a vhodné pro získávání konkrétních lexikálně syntaktických údajů. Pravidla filtrace je možné pohodlně formulovat na základě lingvistických kritérií. Jazyk AX je mj. dostatečně silný i pro přípravu částečných či úplných syntaktických analyzátorů vět přirozeného jazyka. Vstupní věty nemusí být morfologicky zjednoznačněny. Systém byl použit k výběru vět vhodných pro získávání slovesných rámců a na vybraných větách dostupné parsery pro češtinu dosáhly významně lepší úspěšnosti.

Klíčová slova: částečná syntaktická analýza, filtrace vět na základě lingvistických kritérií

1 Motivace

Úspěšnost syntaktických analyzátorů češtiny i dalších přirozených jazyků je v současné době omezena především nedostatkem konkrétních slovníkových údajů, které by vystihovaly typické chování slov ve skladbě věty, například typicky vyžadované typy doplnění. Budování slovníků je přitom časově náročná práce a bylo by vhodné lexikálně-syntaktické údaje získávat částečně automatickou metodou z dostupných korpusů nebo libovolných textů přirozeného jazyka. Ukazuje se však (Bojar, 2002), že dosavadní korpusy syntakticky analyzovaných vět nejsou pro tento účel dostatečně rozsáhlé. Aby bylo možné na základě příkladů použití sledované syntaktické jednotky odvodit její typické chování ve skladbě věty, bude nutné příklady hledat i v „nestromečkových" korpusech či libovolných dostupných textech. Chybějící syntaktickou anotaci je pak možné doplnit pomocí existujících parserů[1] přirozeného jazyka.

Stávající parsery pro češtinu bohužel nedosahují obecně dostatečné přesnosti. Necháme-li však analyzovat věty jednodušší (např. věty do délky deseti slov ap.), průměrná úspěšnost parserů se významně zlepší.

Z hlediska cíle extrahovat konkrétní typ lexikálně syntaktických údajů je však vhodnější vybírat věty s příklady nikoli na základě délky, ale na základě konkrétních lingvistických kritérií. Systém AX byl navržen tak, aby formulaci lingvisticky motivovaných kritérií maximálně usnadnil. Pro vyhodnocení řady kritérií je však nutné provést též částečnou syntaktickou analýzu vstupní věty, a proto systém AX umožňuje rovněž pohodlnou formulaci potřebných pravidel.

Kapitola 2 popisuje souhrnné schéma systému. Kapitola 3 je věnována základní datové struktuře reprezentující vstupní slova věty i pracovní jednotky v průběhu filtrace. Kapitola 4 se zaměřuje na způsob formulace filtrů určených k zamítnutí nevhodných vět a následující kapitola 5 popisuje schéma pravidel pro částečnou syntaktickou analýzu vět. Závěrečná kapitola 6 ukazuje příklad použití systému AX pro výběr vět vhodných pro extrakci doplnění sloves a výsledky zlepšení parserů na takto vybraných větách.

[1] Pro přesnost je třeba rozlišovat mezi pojmy *syntaktický analyzátor* a *parser*. Syntaktický analyzátor pro danou vstupní větu najde všechny dostupné rozbory věty založené výhradně na syntaktických kritériích; pro nesprávnou větu vrátí prázdnou množinu analýz. Parser v původním slova smyslu pro danou vstupní větu rozhodne, zda je věta správnou větou jazyka; nově se však tohoto pojmu používá i pro nástroje, které pro vstupní větu vybudují jednu nejpravděpodobnější syntaktickou strukturu věty, a to s ohledem i na jiná než výhradně syntaktická kritéria a naopak bez ohledu na správnost či nesprávnost vstupní věty. Pro naše účely není tento rozdíl nijak podstatný.

2 Schéma systému AX

Systém AX načte skript popisující filtrace a částečné analýzy vět a dále očekává na svém vstupu morfologicky anotované věty. Vstupní věty mohou i nemusí být morfologicky zjednoznačněny, AX interně pracuje i s více možnými analýzami. Výstupem je pro každou vstupní větu především odpověď, zda věta úspěšně prošla filtracemi nebo byla některým z filtrů zamítnuta. Pro úspěšné vstupní věty je výstupem též poslední dosažené čtení věty po provedení dané částečné syntaktické analýzy. V některých případech může být již výsledek této částečné analýzy dobrým podkladem pro získání lexikálně-syntaktických údajů a není vůbec třeba používat žádný z parserů.

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 3.) 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 buďto *filtrem* (viz 4 na str. 4), 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 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 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.

3 Základní datová struktura: variantová sestava rysů

Sestavy rysů (feature structures, atribute-value matrices) představují velmi známou datovou strukturu. Pro detailní charakteristiku typovaných sestav rysů doporučujeme práci Penn (2000). Pro účely systému AX postačí jednodušší a netypované zavedení této datové struktury. *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.

Základní operací s (variantovými) sestavami rysů je unifikace. Unifikací dvou sestav rysů vznikne sestava rysů obsahující informace z obou vstupních sestav. Z více variant se přitom vybere průnik z obou sestav. Např.:

$$\begin{bmatrix} \textsf{jmeno} & \textsf{Kamil} \\ \textsf{prijmeni} & \left\{\textsf{Horak, Klement}\right\} \end{bmatrix} \text{ a } \begin{bmatrix} \textsf{prijmeni} & \textsf{Horak} \\ \textsf{vek} & \textsf{int(32)} \end{bmatrix} \quad \begin{array}{c}\text{unifikují} \\ \textsf{za vzniku}\end{array} \quad \begin{bmatrix} \textsf{jmeno} & \textsf{Kamil} \\ \textsf{prijmeni} & \textsf{Horak} \\ \textsf{vek} & \textsf{int(32)} \end{bmatrix}$$

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

$\begin{bmatrix} \textsf{jmeno} & \textsf{Kamil} \\ \textsf{prijmeni} & \textsf{Horak} \end{bmatrix}$ a $\begin{bmatrix} \textsf{jmeno} & \textsf{Josef} \\ \textsf{prijmeni} & \textsf{Horak} \\ \textsf{vek} & \textsf{int(32)} \end{bmatrix}$ neunifikují.

Pro každé vstupní slovo věty přirozeného jazyka 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 pouze 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 2.

Podrobný přehled o vztahu použitých atributů a morfologických značek, které jsou výstupem z morfologické analýzy v tomto textu uvádět, k dispozici je v rámci práce Bojar (2002). Názvy atributů i hodnot jsou poměrně mnemotechnické.

$$\left\{ \begin{bmatrix} \textsf{cat} & \textsf{verb} \\ \textsf{morfcat} & \textsf{morfcat(verb(presfut))} \\ \textsf{lemma} & \textsf{string("mít")} \\ \textsf{form} & \textsf{string("má")} \\ \textsf{agr} & \begin{bmatrix} \textsf{num} & \textsf{sg} \\ \textsf{pers} & \textsf{third} \end{bmatrix} \\ \textsf{tense} & \textsf{pres} \\ \textsf{voice} & \textsf{active} \\ \textsf{neg} & \textsf{pos} \end{bmatrix}, \begin{bmatrix} \textsf{cat} & \textsf{pron} \\ \textsf{morfcat} & \textsf{morfcat(pron(poss))} \\ \textsf{lemma} & \textsf{string("můj")} \\ \textsf{form} & \textsf{string("má")} \\ \textsf{agr} & \left\{ \begin{array}{l} \begin{bmatrix} \textsf{case} & \left\{\textsf{nom, vok}\right\} \\ \textsf{gend} & \textsf{fem} \\ \textsf{num} & \textsf{sg} \\ \textsf{pers} & \textsf{first} \end{bmatrix} \\ \begin{bmatrix} \textsf{case} & \left\{\textsf{nom, aku, vok}\right\} \\ \textsf{gend} & \textsf{neut} \\ \textsf{num} & \textsf{pl} \\ \textsf{pers} & \textsf{first} \end{bmatrix} \end{array} \right\} \end{bmatrix} \right\}$$

Obrázek 2: Sestava rysů pro reprezentaci slovního tvaru *má*.

```
[
  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 ]
]
```

Obrázek 3: Zápis variantové sestavy reprezentující slovní tvar *má* v jazyce AX. Odsazení i zalomení řádků bylo přidáno jen pro přehlednost.

Stejnou sestavu rysů zapsanou v syntaxi jazyka AX uvádíme v příkladu 3. Setkáváme se tak se všemi významnými prvky syntaxe variantových sestav rysů:

- 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 zatím nutno uvádět v apostrofech a ve výrazu 'da(<hodnota>)'. Tento nedostatek bude v nejbližší uveřejněné verzi systému odstraněn.

Pro často užívané sestavy rysů je vhodné zavést zkratky pomocí direktivy shortcut nebo shortcuts. Ve filtrech či pravidlech je pak možné užít místo celé sestavy prostě jméno zavedené zkratky. Systém AX rovněž podporuje zadání sestavy „instantní morfologickou analýzou" – zadáme slovní tvar ohraničený zpětnými apostrofy, v době kompilace je na toto místo kódu pak vložena úplná sestava popisující daný slovní tvar:

```
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

shortcut jsem = `jsem`
```

4 Filtry

Filtr v jazyce AX má vždy tvar *regulárního výrazu nad sestavami rysů.* Pojem regulárního výrazu jistě není třeba čtenáři představovat. 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. Sestava přitom může být zadána explicitně, pomocí zkratky nebo pomocí „instantní morfologické analýzy".
- 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.

Detaily syntaxe regulárních výrazů nad sestavami rysů jsou opět uvedeny v práci Bojar (2002), zde uveďme pouze stručný příklad dvou různých filtrů:

```
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
```

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 :.

5 Pravidla

Pravidla jazyka AX slouží k provedení úprav vstupního čtení věty a generují nová čtení věty. Pravidla 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 požadavky dodatečných <omezení>.
- Úspěšně nalezený úsek je ve větě nahrazen řetězcem sestav rysů <náhrada>.

V základní variantě jsou ve vstupní posloupnosti nalezeny všechny možné úseky vyhovující pravidlu, a pravidlo generuje tedy více různých čtení věty. Tento nedeterministický postup lze omezit, pokud do šipky v pravidle vepíšeme klíčové slovo `detstart` (deterministický začátek: po prvním nalezení vstupního úseku nezkoušej již hledat úseky začínající dále vpravo) nebo `detend` (deterministický konec: po prvním úspěšném nalezení vstupního úseku nezkoušej již úsek jiné délky, delší či kratší, podle typu opakování (operátor `*`) v regulárním výrazu).

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.

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. 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
```

Výstupní <náhrada> může též obsahovat pokyn ocitovat část nalezeného úseku do výstupu. To umožňuje pohodlnou formulaci pravidel, která syntakticky zpracují i vzdálená vstupní slova. Regulární výraz přitom dovoluje vyslovit omezení na to, co se mezi spojovanými slovy vyskytnout smí a co ne. Jako velmi výstižný příklad uveďme pravidlo zpracovávající složené slovesné tvary typu *zalil jsem*, *zalili jste*:

```
# složený minulý čas
rule complex_past_tense:
  complex \gap trace
  ---->
     zalil {gap:!{verb,comma,conj}*} jsem
   | jsem {gap:!{verb, comma, conj}*} zalil
  ::
  zalil <- morfcat, voice -> ‘zalil‘,
  zalil = [cat-verb],
  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-"XstopaX"]
end
```

Pravidlo najde takovou posloupnost sestav rysů, která začíná určitým slovesem a končí pomocným slovesem *být* (nebo naopak) a to ve správně shodě. Úsek mezi těmito sestavami pak podle regulárního výrazu nesmí obsahovat ani jiné sloveso, ani čárku a ani spojku. Navíc je úsek mezi částmi složeného slovesa označen názvem `gap` (konstrukce {`gap:`...} v regulárním výrazu). Výstupem pravidla je sestava, která reprezentuje celý složený slovesný tvar, následovaná sestavami, které tvořily mezeru `gap`, a nakonec „stopou“ po druhém členu složeného slovesného tvaru. Stopu samozřejmě není nutné uvádět, pokud není potřeba v dalším zpracování věty.

6 Výsledky použití systému AX

Systém AX byl použit pro výběr vět vhodných k extrakci typických doplnění sloves, slovesných rámců. V jazyce AX byla implementována posloupnost 15 filtrů a 21 pravidel s cílem získat věty, v nichž jsou doplnění sloves podměrně dobře pozorovatelná, a které jsou současně dostatečně jednoduché pro zpracování současnými parsery. Postup zpracování je přibližně tento (pro úplný výpis použitých pravidel viz Bojar (2002)):

- 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 více plnovýznamovými slovesy v jedné klauzi (doplnění sloves mohou být libovolně promíchána),
- Redukce jmenných a předložkových skupin včetně jednoduché koordinace,
- Zamítnutí vět s příliš složitou strukturou klauzí (ponechána nejvýše dvojčlenná souvětí) nebo nevyřešenou koordinací,
- Vyšetření rizika syntaktické homonymie jmenných a předložkových skupin[2], případné zamítnutí vět s „podezřelými slovoslednými vzorci“ (WOP).

[2]Podrobně viz Straňáková-Lopatková (2001).

Přibližně 15 až 20 % vět z korpusu projde touto filtrací. Pro přehlednost je dále označujme jako „velmi jednoduché věty".

Vybrané věty pak byly syntakticky anotovány pomocí tří parserů dostupných pro češtinu: Parser Michaela Collinse (Collins et al. (1999)), parser Dana Zemana (Zeman (1997) a nověji též Zeman (2002)) a parser Zdeňka Žabokrtského (nepublikováno).

Výsledky zlepšení úspěšnosti parserů na velmi jednoduchých větách ve srovnání se všemi dostupnými větami v evaluační části Pražského závislostního korpusu (PDT[3], Böhmová et al. (2001)) jsou uvedeny v tabulce 1.

		Statistické		Pravidlový
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 %
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 %

Tabulka 1: Srovnání parserů na „velmi jednoduchých větách".

Z hlediska extrakce slovesných rámců je nejvýznamnější počet sloves, u nichž parser správně identifikuje všechna doplnění, tj. bezprostředně podřízená slova („pozorovaný slovesný rámec"). Ukazuje se, že nejlepší z dostupných parserů, Collinsův parser, podle tohoto kritéria dosahuje přibližně 55 % správně pozorovaných rámců. Použijeme-li tento parser pouze na velmi jednoduché věty, jak byly identifikovány předchozí filtrací, dosáhneme zlepšení o přibližně 10 %.

Zajímavé je zlepšení úspěšnosti parserů i podle tradičního kritéria, počtu všech správně zapojených uzlů: sledované parsery dosahují na velmi jednoduchých větách úspěšnosti o 5 až 10 % vyšší. Nejvíce je význam výběru velmi jednoduchých vět patrný podle posledního kritéria: počtu vět, které parser analyzuje zcela bez chyby. Parsery Dana Zemana i Zdeňka Žabokrtského dosáhly více než dvojnásobku původní úspěšnosti, nejlepší Collinsův parser ja pak schopen zcela správně syntakticky rozebrat více než polovinu velmi jednoduchých vět, což je o 20 % více ve srovnání s větami bez omezení složitosti.

7 Shrnutí

Představili jsme systém určený pro částečnou syntaktickou analýzu morfologicky anotovaných vět přirozeného jazyka a filtraci vět na základě lingvistických kritérií. Ukázali jsme též konkrétní příklad filtrace s cílem získat příklady vět vhodné k extrakci slovesných rámců. Význam filtrace byl pak potvrzen podstatným zlepšením úspěšnosti tří testovaných parserů.

8 Poděkování

Výsledky uvedené v tomto článku vycházejí především z diplomové práce Bojar (2002). Rád bych na tomto místě proto poděkoval vedoucímu diplomové práce, Dr. Vladislavu Kuboňovi, a rovněž pracovníkům Ústavu formální a aplikované lingvistiky MFF UK, bez jejichž technické pomoci by nemohla práce vzniknout. Práce na tomto tématu byla rovněž podpořena grantem GAČR č. 201/02/1456 a grantem GAUK č. 300/2002/A INF-MFF.

[3] `http://ufal.mff.cuni.cz/pdt/`

Literatura

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.

Bojar, Ondřej. 2002. Automatická extrakce lexikálně-syntaktických údajů z korpusu. in Czech.

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.

Penn, Gerald. 2000. *The Algebraic Structure of Attributed Type Signatures*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University.

Straňáková-Lopatková, Markéta. 2001. Homonymie předložkových skupin v češtině a možnost jejich automatického zpracování. Technical Report TR-2001-11, ÚFAL/CKL, Prague, Czech Republic. in Czech.

Zeman, Daniel. 1997. A Statistical Parser of Czech. in Czech.

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).