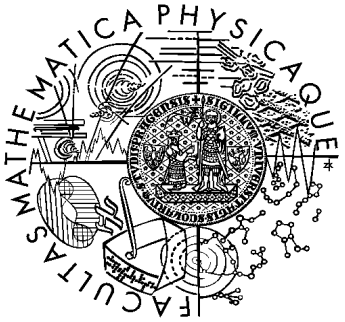
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jan Procházka

Segmentační analýza českých vět

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: RNDr. Vladislav Kuboň, Ph.D.
Studijní program: Informatika, obor Programování

2008

Chtěl bych poděkovat vedoucímu své práce, RNDr. Vladislovu Kuboňovi, Ph. D. , za odborné i věcné rady, čas, který mi ochotně věnoval, i poskytnutí materiálů pro vylepšení práce. Dále bych chtěl velmi poděkovat RNDr. Markétě Lopatkové, Ph. D. za vytvoření anotovaného korpusu pro segmentační analýzu i za cenné připomínky k anotačnímu programu.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne Jan Procházka

Obsah

Název práce: Segmentační analýza českých vět

Autor: Jan Procházka

Katedra (ústav): Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: RNDr. Vladislav Kuboň, Ph.D.

e-mail vedoucího: vk@ufal.mff.cuni.cz

Abstrakt: Cílem této práce je implementace segmentační analýzy českého jazyka včetně vytvoření seznamu separátorů. Kromě toho je zde navržena a implementována metoda rozdělení do klauzí. Implementace využívá český morfologický analyzátor prof. Hajiče. Samotný program je napsán v Pythonu. Metoda byla odladěna na korpusu 62 vět a otestována na korpusu velikosti 80 vět.

Klíčová slova: segmentační analýza, souvětí, syntaktická analýza, čeština, klauze, Python

Title: Segmentation analysis of Czech sentences

Author: Jan Procházka

Department: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Vladislav Kuboň, Ph.D.

Supervisor's e-mail address: vk@ufal.mff.cuni.cz

Abstract: Objective of this work is implementing of segmentation analysis method for Czech language including creating list of separators. Also method, how to divide long sentences into clauses, is proposed and implemented. Implementation uses Czech „Free“ Morfology by Jan Hajič. Program is written in Python. Method was debugged on 62-sentences and tested on 80-sentences corpus.

Keywords: segmentation analysis, complex sentences, syntactic analysis, Czech language, Python

1. Úvod

Člověk se od zvířete liší mimo jiné tím, že dokáže používat řeč. Poměrně přirozenou se jeví jeho snaha obdařit touto schopností i počítače, které v mnohých oblastech už dnes lidský mozek zdaleka překonají. Ač existuje sposta počítačových programů pracujících s přirozeným jazykem (od slovníků a automatických překladačů, přes „kecací programy“, které dovedou vést jistým velmi omezeným způsobem rozhovor s člověkem, vyhledávací služby, které dokáží najít všechny tvary hledaného slova, až po expertní systémy umožňující kladení otázek v přirozeném jazyce), zatím se jazykové schopnosti počítače člověku zdaleka neblíží.

Jedním z důležitých a netriviálních kroků nutných k tomu, aby byl počítač schopen porozumět přirozenému jazyku, je větný rozbor, jinak řečeno syntaktická analýza. Úloha syntaktické analýzy zatím pro češtinu není uspokojivě zvládnuta. Obecně se dá říci, že syntaktická analýza pro kratší a jednodušší věty má celkem slušnou úspěšnost, nicméně delší věty a souvětí způsobují nemalé problémy.

Ideou segmentační analýzy, jež je tématem této práce, je rozdělit vstupní souvětí na jednotlivé klauze, které se pak budou lépe zpracovávat dostupnými syntaktickými analyzátory.

Tato bakalářská práce vychází z [1], kde je navržena a podrobně popsána metoda segmentační analýzy, přesněji řečeno způsob, jakým rozdělit dlouhé souvětí na kratší části nazývané segmenty. V této práci je navržen jednoduchý způsob, jakým spojovat segmenty do klauzí. Hlavní náplní je implementace algoritmů segmentační analýzy a otestování na reálných datech. Pro češtinu jsem vytvořil seznam separátorů a korpus 62 anotovaných vět. Software je po drobných úpravách použitelný i pro jiné jazyky, pro které je k dispozici morfologická analýza, ale kromě ní je nutné pro příslušný jazyk vytvořit seznam separátorů.

V druhé části jsou stručně sepsány poznatky a definovány pojmy z lingvistiky, pouze v minimální míře nezbytné pro potřeby segmentační analýzy. V třetí části je rozebrána samotná metoda segmentační analyzy (podrobnosti lze najít v [1]). V části čtvrté je rozebrána úspěšnost metody na zkoumaném vzorku. Pátá část je uživatelskou dokumentací k implementaci, poslední šestá část plní roli technické a programátorské dokumentace.

2. Pár slov o lingvistice[1]

Předmětem studia lingvistiky je přirozený jazyk. Lingvistika má mnoho podoborů, mezi ty najdůležitější (s ohledem na téme této práce) řadíme:

- fonetika a fonologie – zkoumá zvukovou stránku jazyka
- morfologie – nauka o tvoření tvarů a jejich významech
- syntax – studuje vztahy mezí slovy ve větě, vztahy klauzí (jednoduchých vět) v souvětí
- sémantika – nauka o významu slov, případně vět a jejich vztahu ke sktutečnosti, kterou označují
- lexikologie – nauka o slovní zásobě

Počítačová lingvistika je pak oborem jazykovědy, který se zabývá strojovým zpracováním přirozeného jazyka. V této práci budu využívat výsledky známé z morfologie, těžiště této práce by patřilo do nauky o větné stavbě (syntax).

2.1. Syntaktická analýza

Syntaktická analýza je algoritmus, který umí provést větný rozbor (tj. určení větných členů a závislosti mezi nimi). Metody syntaktické analýzy lze rozdělit na metody s ručně psanými pravidly a metody pravděpodobnostní.

Máme-li program provádějící syntaktickou analýzu pravděpodobnostním způsobem, je potřeba nejdříve sehnat dostatečně velký soubor jazykových trénovacích dat (korpus), ze kterých se program naučí pravděpodobnosti syntaktických vztahů v různých situacích. Na základě takto naučených dat pak provádí syntaktickou analýzu vstupních vět. Výsledkem syntaktické analýzy pak bývá jeden syntaktický strom, který je vyhodnocen jako nejpravděpodobnější.

Syntaktická analýza založená na ručně psaných pravidlech nepotřebuje žádná trénovací data, všechny možné syntaktické vztahy jsou popsané souborem pravidel. Výsledkem je většinou množina všech syntaktických stromů, které lze z daných pravidel a vstupu postavit. Konkrétní tvar pravidel je závislý na použitém programu, jako příklad uvedu pravidlo zachycující shodu podmětu s přísudkem zapsaný v redukční gramatice[2]:

Noun[case=1, num=$1, gender=$2] + Verb[num=$1, gender=$2] => Verb[num=$1, gender=$2];

Aplikace tohoto pravidla spojí dva sousední uzly (levé je podstatné jméno, pravé sloveso) shodující se v číslo a rodě do jednoho řídícího uzlu (tímto řídícím uzlem bude sloveso).

Syntaktická analýza většinou probíhá v několika krocích:

- **Tokenizace** – rozdělení vstupního textu na tokeny (slova)
- **Morfologická analýza** – provádí se pro každý token zvlášť, výsledkem morfologické analýzy pro jeden token je
  - **lemma** – základní slovní tvar, např. u podstatného jméno je to tvar v prvním pádě jednotného čísla
  - **morfologický tag** – je značka nesoucí v sobě informace o slovním druhu, a

1 Tato kapitola čerpá z [2] a [3]

2 Redukční gramatika je gramatika, která popisuje, jak z dané věty postavit syntaktický strom spojováním více sousedních slov. Spojením těchto sousedních slov vznikne podstrom výsledného stromu, v další analýze se tato skupina slov chová jako kořen zmiňovaného podstromu.

morfologických kategoriích závislých na slovním druhu (osoba, číslo, čas, pád, rod, atd.)

- výsledek morfologické analýzy není jednoznačný, např. token **mezi** může být předložka, ale i podstatné jméno **mez** v 3., 5. nebo 6. pádě

- **Tagging** – úkolem taggingu je zjednoznačnění výsledků morfologické analýzy – na základě pravděpodobnostních dat se pro každý token vybere jedna varianta z výsledku morfologické analýzy. Tato fáze nemá smysl, pokud mají být výsledkem všechny možnosti.

Syntaktická analýza relativně dobře zvládá kratší věty, ale analýza delších vět je časově náročná, kromě toho je více náchylná k chybám. Tento problém by měla řešit segmentační analýza.

2.2. Morfologické tagy[3]

Morfologický tag je 15-znakový řetězec. Morfologické kategorie jsou kódovány do znaků na pevné pozici, jak ukazuje následující tabulka.

Pozice	**Jméno**	**Popis**
1	POS	Slovní druh
2	SubPOS	Slovní poddruh
3	Gender	Rod
4	Number	Číslo
5	Case	Pád
6	PossGender	Rod vlastníka
7	PossNumber	Číslo vlastníka
8	Person	Osoba
9	Tense	Čas
10	Grade	Stupeň
11	Negation	Negace
12	Voice	Slovesný rod (aktivní/pasivní)
13	Reserve1	Rezerva
14	Reserve2	Rezerva
15	Var	Varianta

Uvádím pár příkladů morfologických tagů:

- hraniční: `AAIS4----1A----` přídavné jméno, rod mužský neživotný, jednotné číslo, 4. pád, pozitivní
- potok: `NNIS4-----A----` podstatné jméno, rod mužský neživotný, jednotné číslo, 4. pád, pozitivní
- podle: `RR--2----------` předložka (nevokalizovaná) pojící se s 2. pádem

V segmentační analýze si vystačíme s prvními dvěma pozicemi.

3 Tato kapitola byla převzata z dokumentace k [4], kde lze také nalézt více podrobností

2.2.1. Pozice 1 – slovní druh

První pozice určuje slovní druh. Znak na první pozici je redundantní, neboť je jednoznačně určen znakem na druhé pozici, nicméně pomáhá rychlé orientaci, pokud si člověk nepamatuje všech 67 kategorií, které jsou vyjádřeny druhým znakem..

Hodnota	**Popis**
A	Přídavné jméno
C	Číslovka
D	Příslovce
I	Citoslovce
J	Spojka
N	Podstatné jméno
P	Zájmeno
V	Sloveso
R	Předložka
T	Částice
X	Neznámý
Z	Interpunkce

Na pozici druhé je upřesnění slovního druhu. Tabulku vzhledem k rozsahu uvádím v příloze, kapitola 8.1.

3. Segmentační analýza

Jak již bylo řečeno, úkolem segmentační analýzy je rozdělit vstupní větu (souvětí) na menší celky zvané segmenty. Jednotlivé segmenty jsou od sebe odděleny separátory. Zde uvedený algoritmus segmentační analýzy je optimalizován pro češtinu, ale metoda na češtinu strikně vázána není.

Segmentační analýza se skládá z několika kroků:

- rozdělení věty na segmenty
- vytvoření segmentačního diagramu (tj. určení závislostí mezi segmenty, provede se přiřazením hloubky zanoření každému segmentu)
- spojování segmentů do klauzí (klauze je v této práci definována tak, že obsahuje právě jedno sloveso v určitém tvaru)

3.1. Rozdělení na segmenty[4]

Neformálně řečeno, segmentem nazýváme souvislý úsek věty. Segmentací pak nazýváme rozdělení věty na segmenty.

> *Formálně buď $S= t_1,t_2,...,t_n$ věta, t_i jsou jednotlivé tokeny (slova nebo intepunkční znaménka) věty S. Segmentací věty S rozumíme posloupnost segmentů $w_1w_2w_3,...,w_k$, kde posloupnost tokenů w_k nazýváme segmentem. Zřetězením všech segmentů za sebe dostaneme původní větu. Každý segment může být uvozen separátorem, což je poslouponost tokenů na začátku segmentu. Zbytek segmentu nazýváme tělo segmentu.*
>
> *Segment w_k je tedy posloupnost*
>
> *$w_k= t_B, t_{B+1},..., t_{B+S-1}, t_{B+S}, t_{B+S+1}, ..., t_{b+L-1}$, kde B je index tokenu, kterým začíná segmentu w_k (tedy správně bychom měli psát B_k, jelikož je B na k závislé), S je počet tokenů v separátoru segmentu w_k a L je délka segmentu w_k (analogicky bychom měli psát S_k a L_k). Separátor segmentu w_k, je posloupnost $sep(w_k)= t_B, t_{B+1},..., t_{B+S-1}$,*
> *tělo segmentu $body(w_k)= t_{B+S}, t_{B+S+1}, ..., t_{b+L-1}$.*

Všechny segmenty kromě prvního segmentu w_1 musí být uvozeny neprázdným separátorem, u prvního segmentu je separátor nepovinný. Poslední segment mívá prázdné tělo (většinou je to tečka na konci věty, což je separátor, a za ní už nic není).

Segmentaci názorně ukážu na větě „ Ačkoliv uměl několik cizích řečí, česky neuměl v době, kdy se stal českým králem. “, vypadá následovně:

4 Značení a terminologie pro tuto kapitolu, stejně jako znázornění diagramu v kapitole následující, bylo po mírné upravě převzato z [1], tam lze také nalézt podrobnosti

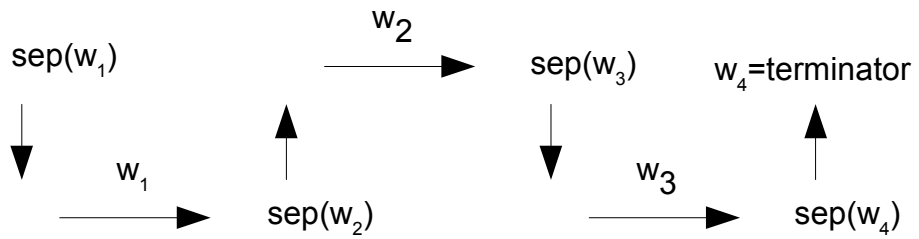
Segment	Část segmentu	Tokeny
w_1	separátor	Ačkoliv
	tělo	uměl několik cizích řečí
w_2	separátor	, (čárka)
	tělo	česky neuměl v době
w_3	separátor	, kdy
	tělo	se stal českým králem
w_4	separátor	. (tečka)
	tělo	

Segmentace věty se pro účely této práce považuje za jednoznačnou.

3.2. Segmentační diagram

Segmentační diagram přiřazuje jednotlivým segmentům ze segmentace hloubku zanoření, čímž jsou částečně určeny závislosti mezi segmenty. Z diagramu jsou vidět vztahy souřadnosti a podřadnosti, nicméně není v něm explicitně uvedeno, jak jsou segmenty na sobě závislé.

Diagram pro větu „ Ačkoliv uměl několik cizích řečí, česky neuměl v době, kdy se stal českým králem. “ je na následujícím obrázku:

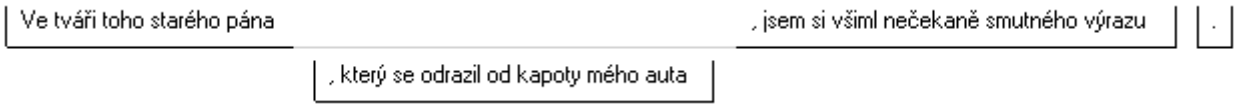


Z obrázku je na první pohled vidět, že w_2 je věta hlavní, w_1 a w_3 jsou o úroveň níže, zde je tedy jasné, že w_1 a w_3 závisí na w_2 . Segmentační diagram (na rozdíl od segmentace) není určen jednoznačně. Hloubku zanoření pro jednotlivé segmenty určujeme podle typu separátoru, který bezprostředně předchází danému segmentu. Například podřadící spojka způsobí, že segment vpravo od ní bude mít úroveň o jedničku větší, než segment vlevo od ní. Více v kapitole věnované separátorům.

3.3. Spojování segmentů do klauzí

Klauze je definována jako souvislá skupina segmentů se stejnou hloubkou zanoření, která obsahuje právě jedno sloveso. Segmenty v jedné klauzi spolu musí sousedit, ovšem jsou povolené vložené segmenty s vyšší hloubkou zanoření.

Příklad:



Tato věta s skláda z těchto dvou standardních klauzí:

- Ve tváři toho starého pána, jsem si všiml nečekaně smutného výrazu.
- , který se odrazil od kapoty mého auta
- třetí, speciální klauzí je tečka na konci věty

V dalším textu budeme většinou využívat trošku méně názorné ASCII obrázky segmentace, které mají tu výhodu, že se jimi dá znázornit libovolně dlouhá věta. ASCII obrázek předchozí věty vypadá takto:

```
L0 |Ve tváři toho starého pána|                                        %, jsem si všiml
L1                            |, který se odrazil od kapoty mého auta%
-------------------------------------------------------------------------------------------
L0  nečekaně smutného výrazu|.
L1                          |
===========================================================================================
```

Hloubka zanoření segmentu je zde dána řádkou, ve které je segment umístěn (L0 – věta hlavní …), „|“ slouží jako oddělovač klauzí, „%“ slouží jako oddělovač segmentů, které tvoří dohromady jednu klauzi. Přesněji řečeno, segmentem, který je uvozen znakem %, pokračuje klauze, jejíž předchozí segment je nejbližší vlevo na stejné úrovni.

Definice klauze, kterou používáme, je vhodná pro strojové zpracování a v naprosté většině případů odpovídá výsledkům, které by člověk intuitivně očekával, nicméně ve výjimečných případech dá trošičku jiný výsledek, jako v následujícím příkladě:

```
L0 |Malé částečky%, menší než tisícina průměru lidského vlasu%, rozptylují|a odráží modré
L1
-------------------------------------------------------------------------------------------
L0  světlo%, stejně jako malé molekuly ve vzduchu|
L1                                               |, které tvoří modrou oblohu .
===========================================================================================
```

Zde by člověk pravděpodobně dvojici sloves „rozptylují a odráží“ považoval za vícenásobný přísudek, nicméně pro účely této práce je považován za dvě klauze.

3.4. Formální popis diagramu

Pro účely jednoznačného popisu algoritmů zavedu formální popis segmentačních diagramů.

> *Buď t_1, t_2, t_3, …, t_n věta, t_i jsou jednotlivé tokeny (t_n je terminátor věty, většinou tečka), w_1, w_2, w_3, … ,w_m je segmentace věty.*
>
> *Úroveň segmentu označíme $level_D(w_i)$, příznak, zda segmentem začíná klauze, označíme $clausebeg_D(w_i)$, kde D je diagram, kterého se úrovně týkají. Pokud je to z kontextu jasné, můžeme D vynechat, tj. Pak se píše např. $level(w_i)$*
>
> *Samotný diagram zavádíme jako zobrazení D: $W->N_0 x \{0,1\}$, kde W je množina segmentů, N_0 přirozená čísla s nulou, tedy $D(w_i)=(level_D(w_i), clausebeg_D(w_i))$*
>
> *Klauze je pak definována takto[5]:*
>
> *w_k_1, w_k_2, w_k_3, … w_k_n je klauze, ($k_1..k_n$ jsou indexy segmentů) právě když platí: $clausebeg(w\_k_1)=true$, $clausebeg(w\_k_i)=false$ pro $i >= 2$*

5 Značení w_k_1 je použito, aby nemusel být používán špatně čitelný dvojitý dolní index. Tedy k_1 je index w

$level(w\_k_1)=level(w\_k_2)=...=level(w\_k_n)$
pro každé i,j: $k_i<j<k_i+1 \Rightarrow level(w\_j)>level(w\_k_i)$
uspořádaná množina $k_1<k_2..<k_n$ je maximální vzhledem k výše uvedeným podmínkám

3.5. Separátory

Separátorem rozumíme poslounost tokenů, která od sebe odděluje segmenty (resp. kterým začíná segment vpravo od separátoru). Pro analyzovaný jazyk je třeba shromáždit kompletní seznam separátorů. Separátory mohou být zadány pomocí regulárních výrazů a morfologických tagů (tj. např. určením slovního druhu), proto tento seznam nemusí až tak obsáhlý.

Každý separátor má následující vlastnosti:

- Hodnotu (příp. více hodnot) úrovně zanoření následujícího segmentu, a to pro případ, kdy následující segment uvozuje klauzi (atribut newlevel), i kdy ji neuvozuje (atribut newlevel_noclause). Hodnota se zadává pomocí výrazu, který může obsahovat proměnnou **current** představující úroveň zanoření aktuáního segmentu. Uvádím příklady této hodnoty pro vybrané základní separátory:
  - Podřadící spojka: **current+1**
  - Čárka: **0..current** (znakem vodorovné dvojtečky **..** se zadává interval)
  - Souřadící spojka: **current**
- Posloupnost tokenů, která tvoří separátor; token může být zadán
  - regulárním výrazem ověřovaným na tokenu
  - regulárním výrazem ověřovaným na morfologickém tagu
- Prioritu – pokud posloupnost tokenů odpovídá více separátorům, je potřeba deterministicky vybrat jeden – ten s nejvyšší prioritou, který se použije (priorita se určuje pořadím separátorů v definici – vybere se první separátor, který odpovídá zadaným kritériím)
- Kontextová podmínka (nepovinné) – pomínka, která podmiňuje použití separátoru v závislosti na předchozích separátorech (např. uzavírací závorka se musí vyskytovat po otevírací závorce)

V části věnované technické dokumentaci je přesná specifikace formátu souboru separátorů.

Úplný seznam separátorů pro češtinu uvádím v příloze 2, kapitola 8.2.

Vzhledem k nejednoznačnosti morfologické analýzy se může stát, že o nějaké posloupnosti tokenů nevíme jistě, zda je to separátor, nebo ne. V takovém případě ji za separátor považujeme, což sice zvětší počet segmentů, na druhou stranu se vyhneme přílišnému nárůstu možností.

3.6. Pravidla pro diagram

Diagram může být zadán ve formátech XML/SEG nebo XML/ANOT (viz kapitola 6). Pokud je segmentace ověřená nebo vytvořená člověkem (tím pádem pro jednu větu je jen jeden segmentační diagram), preferovaný formát je XML/ANOT.

- Věta je rozdělená do segmentů, přičemž každý segment začíná separátorem (kromě prvního segmentu, ten separátorem začínat nemusí, ale může).
- Každý segment má přiřazenou úroveň (atribut **level**), která může nabývat nezáporných hodnot. Věta vedlejší má úrveň o jedničku větší, než věta, na které závisí.

- Souřadně spojené věty mají stejnou úroveň. Jedna klauze se skládá ze segmentů na stejné úrovni, přičemž první segment klauze má nastaven atribut **clausebeg** na „1“.
- Uvozovky se pro účely segmentační analýzy ignorují v tom smyslu, že neovlivňují segmentační diagram, ale jsou zahrnuty do segmentů (pokud jsou uvozovky bezprostředně vedle separátoru, připojují se k pravému segmentu.
- Závorky
  - Otevírací závorka tvoří separátor, který uvozuje segment o jedničku vyšší než jeho levý soused
  - Uzavírací závorka uvozuje segment, který je na úrovni o jedničku nižší než příslušející otevírací závorka.
- Klauze obsahuje právě jedno sloveso v určitém tvaru (tedy vícenásobný přísudek je rozdělen do více klauzí)
  - Výjimkou je segment mezi závorkami, tento segment ja klauzí, i pokud neobsahuje sloveso
  - Pokud je segment typu **terminator**, je považován za klauzi.
- Ukončení věty – tečka, vykřičník nebo otazník – tvoří zvláštní segment (i klauzi) na úrovni 0, typ segmentu je **terminator**.

3.7. Ukázky anotací jednotlivých jevů

3.7.1. Souvětí s podřadící spojkou na začátku

```
L0 |                        |, vjel jsem omylem do jiného dvora|a naboural do stromu|
L1 |Když jsem prijížděl domů|                                  |                    |,
-------------------------------------------------------------------------------------------
L0                                    |.
L1  protože ho doma na tomto místě nemám|
===========================================================================================
```

3.7.2. Vložená věta vedlejší, vícenásobné větné členy

```
L0 |Pašijový rituál|                                                              %, se koná
L1                |, při němž je symbolicky přivázán na dřevěný kříž vybraný muž%
-------------------------------------------------------------------------------------------
L0  na místě|                                        %       %       |.
L1          |, na němž kdysi lidé uctívali bohy země%, větru%a deště|
===========================================================================================
```

3.7.3. Závorky

```
L0 |" Bílý dům je přesvědčen|                    |                           |
L1                          |, že jedinou možností|                           |
L2                                               |, jak vyřešit tento problém|
L3                                                                           |( íránských
-------------------------------------------------------------------------------------------
L0                  % %                                  |                   |, " řekl
L1                  % %, je změnit mocenskou strukturu Íránu|, a to znamená válku|
L2                  %)%                                  |                   |
L3  jaderných zbraní% %                                  |                   |
-------------------------------------------------------------------------------------------
L0  nejmenovaný poradce amerického ministerstva obrany|.
L1                                                    |
L2                                                    |
L3                                                    |
===========================================================================================
```

3.7.4. Uvozovky

```
L0 |Polská prokuratura již vyšetřuje|                                          %
L1                                  |, kam se podělo nejenom oněch " pět v českých%" , ale
L2
-------------------------------------------------------------------------------------------
L0                                  |
L1  také dalších 1,3 miliardy korun|
L2                                  |, které měly být polskými lobbisty údajně rozděleny na "
-------------------------------------------------------------------------------------------
L0           |" .
L1           |
L2  provizích|
===========================================================================================
```

Zvláště zajímavá je poslední uvozovka, která by intuitivně patřila na úroveň L2, nicméně podle definice je součástí dalšího segmentu.

3.8. Algoritmus segmentační analýzy

Algoritmus se skládá z několika na sebe navazujících kroků:

- rozdělení vstupního textu na věty
- morfologická analýza
- rozdělení věty na segmenty
- generování všech možných segmentačních diagramů včetně rozdělení do klauzí (tyto možnosti jsou generovány na základě typu separátoru)
- vyloučení nesmyslných možností – např. se vyloučí diagramy, které nemají ani jednu hlavní větu, nebo kde existuje klauze, která nemá právě jedno určité sloveso

Výsledkem algoritmu je množina všech nevyloučitelných diagramů. Celý program je navržen tak, aby správná možnost byla mezi výslednými možnostmi (tedy aby algoritmus měl 100% pokrytí).

3.8.1. Rozdělení věty na segmenty[6]

Označme větu $t_1, t_2, t_3, \ldots t_n$. Máme seznam separátorů s_1, s_2, s_3.. Každý separátor se skládá z několika slov, označme i-té slovo j-tého separátoru $s_{j\,i}$

Algoritmus probíhá větu zleva doprava, pro i-té slovo vstupní věty vypadá takto:

- *najdi nejmenší j (první separátor) tak, že:*
  - *t_i „=“ $s_{j\,1}$, t_{i+1} „=“ $s_{j\,2,}$, … , t_{i+k-1} „=“ $s_{j\,k}$, kde k je délka separátoru j*
    - *relace a „=“ $s_{j\,k}$ odpovídá na otázku: je token a vstupní věty přípustný pro k-tý token separátoru s_j?*
  - *pokud takové j existuje, dál pokračuj na slově t_{i+k} , mezi slova t_{i-1} a t_i vlož hranici segmentů, segment bude uvozen separátorem s_j*
  - *pokud ne, pokračuj na slově t_{i+1}*

Výsledek tohoto kroku je seznam segmentů, každý segment je uvozen separátorem (u prvního segmentu je uvozující separátor nepovinný).

6 Implementace je provedena v souboru preseg.py, funkce process_sentence

3.8.2. Generování segmentačních diagramů[7]

Na vstupu máme seznam segmentů $w_1, w_2, \ldots, w_n$, segment w_i uvozuje separátor w_i (separátor je jeho součástí). Výstupem této fáze je množina všech možných segmentačních diagramů včetně spojení do klauzí (tato množina obecně obsahuje víc diagramů, než je ve výsledku – možné diagramy jsou omezeny pouze typem separátorů, v další fázi se pak tato množina dále filtruje).

> *Formálně je tedy výstupem množina {D_k; k je index výsledného diagramu}, přičemž D_k je ohodnocení segmentů D_k: w_i -> (level, clausebeg) , kde level je úroveň segmentu w_i a clausebeg je příznak, zda tímto segmentem začíná klauze.*

Klíčové pro tento algoritmus jsou atributy newlevel a newlevel_noclause v definici separátoru. Úrovně segmentů se počítají zleva doprava (tedy pokud počítáme úroveň segmentu w_i, známe úrovně všech segmentů w_j pro všechna j<i).

Krok pro separátor w_i tedy vypadá takto:

```
funkce krok(i, částačná definicie diagramu DP):
  if i > počet segmentů:
    pridej DP do výsledku
    return

  newlevels := zjisti množinu z atributů newlevel separátoru wi
  newlevels_noclause := zjisti množinu z atributů newlevel_noclause separátoru wi

  pro každý prvek x z množiny newlevels:
    krok (i+1, DP + (wi->(newlevel=x, clausebeg=True)))

  pro každý prvek x z množiny newlevels_noclause:
    krok (i+1, DP + (wi->(newlevel=x, clausebeg=False)))
```

Jeden krok spočítá úroveň segmentu a příznak clausebeg, a rekurzivně se spustí pro další segment. Možná trochu matoucí může být zápis *DP + (w_i->(newlevel=x, clausebeg=True))* , DP je částečná definice diagramu, tedy zobrazení, zobrazení *DP + (w_i->(newlevel=x, clausebeg=True))* vrací dvojici (x, True), pokud je argument w_i, jinak vrací stejné hodnoty jako zobrazení DP.

První krok algorimu vypadá takto:

```
krok(0, prázdné zobrazení)
```

3.8.3. Filtrování diagramů[8]

Nejprve si připomeňme značení. Buď D diagram, w_k segment, pak $level_D(w_k)$ je úroveň segmentu w_k v diagramu D, pak $clausebeg_D(w_k)$ je příznak, zda segment w_k představuje začátek klauze.

Aby byl diagram ponechán ve výsledné množině, musí splňovat několik kritérií:

- Musí mít hlavní větu (tj. musí existovat segment w_k tž. $level(w_k)=0$)
- Pokud pro segment w_k platí $clausebeg(w_k)=0$ (tj. segment není začátkem klauzí), musí ve stejném diagramu existovat segment w_i ($i<k$) tž. $level(w_i)=level(w_k)$ & $clausebeg(w_i)=1$, navíc pro všechny segmenty w_m pro $i<m<k$ musí platit $level(w_m) > level(w_k)$
- Další podmínky jsou dány konfigurací uloženou v souboru checkers.xml, standardně se používají tyto kritéria (stačí, když pro každou klauzi platí alespoň jedna z níže uvedených podmínek):
  - Klauze obsahuje právě jedno určité sloveso
  - Klauze obsahuje právě jedno sloveso v minulém čase s právě jedním pomocným tvarem slovesa být (např. plaval jsem)

7 Implementace je provedena v souboru generator.py, funkce generate_sentence

8 Implementace je provedena v souboru sentfilter.py, funkce sentences_of_variant

- Klauze je text v závorkách (tedy text v závorkách je klauzí, i když neobsahuje sloveso)
- Klauze je terminátor věty (tedy např. tečka na konci věty)

4. Použití na reálných datech

V práci byly použity tři korpusy:

- korpus stažený z portálu zpravy.idnes.cz o délce 54 vět + 8 ručně vybíraných ukázkových vět
- korpus 35 vět, na kterém byla testována metoda v práci [1]
- korpus 80 vět z elektronické verze Respektu

První jmenovaný korpus byl použit pro vývoj, takže při jeho zpracování vykazuje metoda 100 % úspěšnost[9]. U tohoto korpusu podrobněji rozebíráme nejednoznačnost, kolik segmentačních diagramů se vytvoří pro jednu větu.

Druhé dva korpusy používám pro otestování metody (tzn. úspěšnost se počítá i s chybami objevenými na základě dat z tohoto korpusu). Zde se bohužel 100 % pokrytí nedosáhlo, dále tedy uvádím úspěšnost a nejednoznačnosti se nevěnuji.

4.1. Nejednoznačnost v grafech a tabulkách

K analýze z idnes.cz byly vybírány pouze věty s alespoň čtyřmi čárkami. Tato podmínka byla zvolena proto, abych algoritmus odzkoušel na dostatečném počtu netriviáních vět. Na druhou stranu, věty s alespoň 4 čárkami jsou nadprůměrně složité, proto nejednoznačnost na běžném českém textu bude pravděpodobně menší.

Tato tabulka ukazuje základní statistiku korpusu z iDnes:

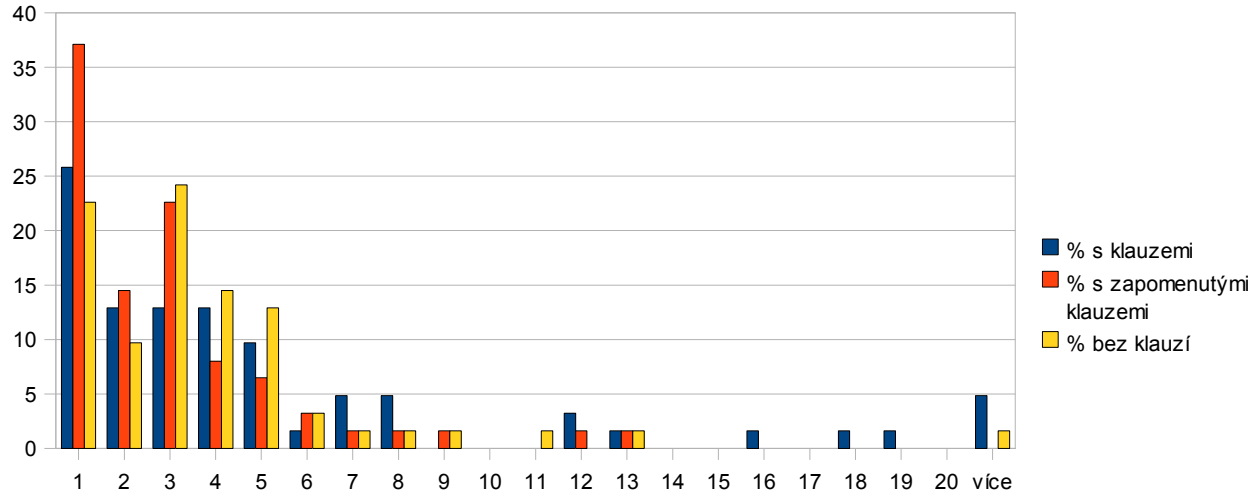
Vlastnost	Hodnota	Průměr na větu
Počet vět	62	
Klauze	286	4.61
Segmenty	460	7.42
Počet diagramů	370	5.97, medián 3.0
Počet diagramů se zapomenutými klauzemi	184	2.97, medián 2.0
Počet diagramů bez ohledu na klauze	262	4.22, medián 3.0

- Počet diagramů – počet různých diagramů včetně rozdělení do klauzí. Diagramy se považují za různé, pokud mají různé úrovně nebo rozdělení do klauzí – formálně D_1, D_2 jsou různé, pokud existuje segment w_k tž. $level_{D1}(w_k) <> level_{D2}(w_k)$ nebo $clausebeg_{D1}(w_k) <> clausebeg_{D2}(w_k)$
- Počet diagramů se zapomenutými klauzemi – počet diagramů, které se liší úrovněmi. Formálně – diagramy D_1, D_2 se liší úrovněmi, pokud existuje segment w_k tž. $level_{D1}(w_k) <> level_{D2}(w_k)$
- Počet diagramů bez ohledu na klauze – počet různých diagramů, pokud zpracování probíhá

9 Přesněji řečeno 100% pokrytí, což znamená, že metoda vrátí více výsledků, z nichž některé jsou špatné, ale ten správný se mezi nimi musí vždy nacházet

bez použití klauzí. V takovém případě je příznak clausebeg vždy true, tudíž se diagramy liší jen úrovněmi

Následující graf znázornňuje četnost výskytu počtu diagramů (X-ová osa znázoňuje počet diagramů ve výsledku segmentační analýzy, na Y-ose je nanesena procentuální četnost).



Pro větší přesnost uvádím ještě tabulku s těmi samými údaji – nejdříve pro počty diagramů s klauzemi:

D	1	2	3	4	5	6	7	8	12	13	16	18	19	33	46	42					
#	16	8	8	8	6	1	3	3	2	1	1	1	1	1	1	1					

V prvním řádku je počet diagramů, v druhém je jejich četnost.

Ta samá tabulka, pokud se zapomenou klauze (tj. za různé považujeme jen ty diagramy, ve kterých se liší úroveň segmenů), vypadá takto:

D	1	2	3	4	5	6	7	8	9	12	13										
#	23	9	14	5	4	2	1	1	1	1	1										

Je vidět, že rozdělení do klauzí je dost nejednoznačné – pokud ho vynecháme, možností bude výrazně méně. Na druhou stranu klauze pomáhají vyloučit spoustu možností, takže minimálně jako vnitřní součást algoritmu segmentační analýzy je vhodné klauze použít. Důkaz tohoto tvrzení vidíme v následující tabulce (nebo v grafu s označením % bez klauzí), která ukazuje případ, pokud během algoritmu vůbec nebereme na klauze ohled:

D	1	2	3	4	5	6	7	8	9	11	13	15	25								
#	14	6	15	9	8	2	1	1	1	1	1	2	1								

Výsledek má stejnou informační hodnotu jako v tabulce se zapomenutými klauzemi (tj. jen úrovně segmentů), nicméně má znatelně více nerozhodnutelných výsledků

4.1.1. Nejednoznačnost separátorů

V následující tabulce uvádím statistiku jednotlivých separátorů. Sloupeček „nejednoznačnost“ uvádí průměrný počet různých interpretací separátorů. Tento pojem zadefinujeme trochu přesněji, použijeme značení z kapitoly 3.7, „Algoritmus segmentační analýzy“.

- Pro připomenutí, D: w_i -> (level, clausebeg) je ohodnocení segmentů, neboli formální popis diagramu
- Obrazem separátoru pro diagram D nazveme dvojici ($level_D(w_{i-1})$ - $level_D(w_i)$, $clausebeg_D(w_i)$)
- Počtem interpretací separátoru myslíme počet různých obrazů pro všechny diagramy pro uvažovanou větu

Jednoduše řečeno, interpretace separátoru jsou různé, pokud se liší rozdíl úrovní levého a pravého segmentu sousedícího se separátorem, nebo příznak clausebeg.

Separátor[10]	Počet výskytů	Průměrná nejednozn.	Min nejednozn.	Max nejednozn.	Medián nejednozn.
comma	129	2.08	1	10	1
sent_close	62	2.08	1	5	2
coord	60	1.33	1	2	2
comma_sub	45	1	1	1	1
comma_pronoun	35	1	1	1	1
comma_coord	26	2.58	1	6	4.5
prep_pronoun	10	1	1	1	1
comma_sub_multi	8	2.125	2	3	2
colon	5	1	1	1	1
comma_adv	5	1	1	1	1
dash	4	1.75	1	2	2
sub_start	4	1	1	1	1
comma_nez	3	2	2	2	2
comma_jak	3	2	2	2	2
par_close	2	1	1	1	1
par_open	2	1	1	1	1

4.2. Úspěšnost na testovacích korpusech

První test byl proveden na větách už anotovaných v práci [1].

Tyto věty jsou anotovány trochu jiným způsobem:

- souvětí se rozděluje na segmenty na základě hranic
- hranice segmentů jsou definovány takto:
  - čárka, dvojtečka, středník, otazník, pomlčka (všech délek), závorka levá i pravá, svislítko, uvozovky
  - souřadné spojky
  - těsně sousedící hranice jsou brány jako jedna složená hranice

10 Seznam separátorů je uveden k kapitole 3.3, „Seznam separátorů“

- u segmentů je dána pouze úroveň, klauze nejsou nijak řešeny

Poměr segmentů se správnou úrovní v tomto testu vyšel **74.48 %** .

Druhým testovacím korpusem byl soubor 80 vět z Respektu, který byl anotován primárně pro potřeby této práce, tudíž definice segmentů byly stejné, navíc je tento korpus anotovaný i s klauzemi. Výsledky uvádí následující tabulka:

Objekt	Celkový počet	Počet/poměr správně	Počet/poměr špatně
Věty	81	49 / 60.49 %	32 / 39.51 %
Věty bez klauzí	81	66 / 81.48 %	15 / 18.52 %
Úrovně	354	342 / 96.61 %	12 / 3.39 %
Začátky klauzí	354	313 / 88.42 %	41 / 11/.8 %

V řádku „Věty“ je počet vět, které byly kompletně správně, v řádku „Věty bez klauzí“ jsou věty, které mají správně určené úrovně segmentů (klauze nás nezajímají). V řádku „Úrovně“ je počet segmentů se správně určenými úrovněmi, v řádku „Začátky klauzí“ počet segmentů se správně určeným příznakem clausebeg i správnou úrovní. (Pokud je příznak clausebeg na špatné úrovni, jeho hodnota je irelevantní).

Vidíme, že **60 %** vět bylo určeno kompletně správně, pokud nebereme v úvahu klauze, výší se toto číslo na u **82** %. Výsledek **96.61 %** správně určených úrovní segmentů je výrazně lepší než analogický výsledek (74.48 %) u minulého testu. Důvodem je to, že věty v prvním testu jsou výrazně složitější než věty z Respektu – věty z prvního korpusu byly vybírány ručně za účelem otestování metody, věty z Respektu byly vybírány automaticky.

Namátkou jsem vybral nadprůměrně složité věty z obou korpusů, takto vypadá věta z prvního korpusu:

> *Lidé, kteří uzavřou smlouvu se zahraniční pojišťovnou, která nemá povolení u nás působit , se vystavují nebezpečí, že pojišťovna nedodrží podmínky smlouvy a že v případě jejího vypršení či takzvané pojistné události, kterou může být například úraz, nedostanou žádnou pojistnou náhradu.*

A takto z Respektu:

> *Třeba Hillary Clintonová totiž svůj plán neformuluje jako oběť , ale naopak jako výzvu – vydělají ti , kteří do nového světa přijdou s nejlepšími řešeními , s novými technologiemi , s novou vizí.*

4.3. Speciální případy

4.3.1. Souřadné spojení vedlejších vět

Tento případ nastává, pokud nejsou vedlejší věty spojeny souřadící spojkou nebo čárkou, ale jsou všechny uvozeny podřadící spojkou:

> *"Drazí přátelé, modlete se za mne, **abych** se naučil milovat Pána stále víc, **abych** stále více miloval vás, svatou církev i každého jednotlivě a všechny společně.*

Tento případ je ošetřen speciálně tak, že při vícenásobném výskytu téže podřadící spojky jsou všechny segmenty touto spojkou uvozené na stejné úrovni zanoření. (Tato interpretace je přidávána k ostatním, je možné mít dvě stejné podřadící spojky, které nejsou spojeny souřadně).

Problém může být i komplikovanější, jako v následujícím případě, avšak tento případ už speciálně řešen není.

> *Tisíckrát účinnější náprava vznikne, **když** neplatit alimenty bude pro krkavce nebezpečné, **když** s neplatiči stát tvrdě zatočí, **než když** je trénuje v neodpovědnosti: kašlete na děti, stát vás zastoupí.*

4.3.2. Vztažná příslovce

Vztažná příslovce jsou morfologickou analýzou označeny tagem Db.., tedy stejně, jako onstatní příslovce. Proto je v seznamu separátorů explicitně vyjmenován seznam vztažných příslovcí:

kde
kam
kudy
kdy
jak
proč

Pokud bych to neudělal, musel by se separátor „čárka příslovce“ chovat navíc jako obyčejná čárka, což by vzhledem k velké nejednoznačnosti čárky znamenalo výrazné zhoršení přesnosti.

4.3.3. Spojka „než“

Spojka „než“ je morfologickou analýzou označena jako podřadící, tj. měla by zvyšovat zanoření segmentu o jedničku, jako v následující větě:

> *České podniky loni do ovzduší vypustily méně emisí oxidu uhličitého, než měly povoleno.*

Ovšem bez čárky se dá použít i jako souřadící spojka:

> *Vždy je elegantnější politik, který alespoň předstírá, že se svými ideály i spojenci vyhraje či padne, **než** politik, který s nimi přímočaře kramaří.*

V seznamu separátorů je proto tato spojka uvedená explicitně, přičemž kromě významu podřadící spojky je jí také přisouzeno spojování neklauzí.

4.3.4. Spojka „jak“

Tato spojka slouží jednak jako podřadící spojka:

> ***Jak** uvedla londýnská firma Barclays Capital, "zdá se, že problémy Itálie vězí tak hluboko, že příští vláda, ať už bude její složení jakékoli, sotva udělá něco více, než že je začne řešit".*

Nebo může sloužit k připojení neshodného přívlastku:

> *"Bílý dům je přesvědčen, že jedinou možností, **jak** vyřešit tento problém, je změnit mocenskou strukturu Íránu, a to znamená válku," řekl nejmenovaný poradce amerického ministerstva obrany.*

Kromě toho může být jak i podstatné jméno[11]. Stejně jako „než“, ja i spojka „jak“ ošetřena speciálně v seznamu separátorů.

4.3.5. Podřadící spojka na začátku souvětí

Podřadící spojka nemusí být bezprostředně na začátku souvětí:

11 Označení jak (nebo také yak) se používá pro dlouhosrstý, hrbatý domácí skot pocházející z Tibetu

__A tak__ když George Bush označí zprávy o vojenském řešení za "divoké dohady", má to zhruba stejnou hodnotu jako jeho někdejší ujištění, že Irák má ještě šanci vyhnout se válce, které pronesl poté, co o válce už rozhodl.

Obecně je to v seznamu separátorů podchyceno tak, že před podřadící spojkou na začátku souvětí mohou být maximálně dvě souřadící spojky.

4.3.6. Dvě podřadící spojky za sebou

Segment může začínat i dvěma podřadícími spojkami za sebou:

NATO nám vzkázalo, __že když__ nás přijme, budeme muset bez výjimky dostát všem závazkům, které platí pro ostatní členy.

V seznamu separátorů je tato možnost ošetřena tak, že segment vpravo od separátoru má úroveň o 2 větší než levý segment. V [1] je tato situace řešena jinak. Separátor „(čárka) že když“ se považuje za tzv. sekvenci separátorů, skládající se ze dvou separátorů.

```
L0 |NATO nám vzkázalo|                  |                                                    |
L1                   |                  |, budeme muset bez výjimky dostát všem závazkům|
L2                   |, že když nás přijme|                                                  |,
-------------------------------------------------------------------------------------------
L0                                |.
L1                                |
L2  které platí pro ostatní členy|
```

4.4. Zdroje nejednoznačnosti

Vlastností metody pospané v této práci je to, že vždy vrací správný výsledek (nebo alespoň v drtivé většině případů). To na druhou stranu vede k velké množině možných výsledků, ze kterých je potřeba vybrat ten správný. Bohužel se poměrně často stává, že kvůli ošetření jednoho speciálního případu – kvůli nějaké málo časté větné konstrukci – se musí do zpracování přidat pravidlo, které sice tento speciální případ ošetří, ale na druhou stranu se díky tomuto ošetření v běžných konstrukcích objeví ve výsledku velké množství špatných segmentačních diagramů.

V této kapitole rozeberu na příkladech nejčastější větné konstrukce, které způsobují nejednoznačnosti.

4.4.1. Souřadné spojení vět hlavních s čárkou

Jedná se o separátory **comma_coord** a **comma**. Program ví, že další segment není na vyšší (více zanořené) úrovni než aktuální, nicméně může být na libovolné nižší úrovni, protože čárka může ukončovat vloženou větu vedlejší.

```
L0 |Zdravotníky pobouřilo%i samotné Zumovo přiznání|                    |              |
L1                                                  |, že nepoužil kondom|              |
L2                                                                       |, ačkoli věděl|
L3                                                                                      |, že
L4
-------------------------------------------------------------------------------------------
L0      |               %                |.
L1      |               %                |
L2      |               %                |
L3  žena|               %                |
L4      |, se kterou spí%, je nakažená HIV|
```

U této věty je diagram celkem jednoznačný[12], problém dělá až poslední segment, *je nakažená HIV.*

12 Toto tvrzení není tak úplně pravda, pro tuto chvíli jsem zanedbal pravidlo, které korektně zpracuje souřadně spojené vedlejší věty uvozené stejnou podřadící spojkou – zde spojkou že, aplikace tohoto pravidla zanesla do tohoto příkladu další nejednoznačnosti, jak bude vidět hned v další kapitolce

Všechny segmenty předtím jsou uvozeny podřadící spojkou, takže je jasné, že musí být o úroveň níže, než segment vlevo od nich. Poslední segment je však uvozen separátorem „čárka“ (comma), tudíž může být na úrovni menší nebo rovné než jeho levý soused. Tedy tento segment může být na libovolné z úrovní L0, L1, L2, L3, L4.

4.4.2. Souřadné spojení vět vedlejších

Jedná se o separátor **comma_sub_multi**. Aplikuje se při vícenásobném výskytu téže podřadící spojky v jedné větě. Nejednoznačnost vzniká tím, že segment vpravo od separátoru může jednak pokračovat o úroveň níž, než levý segment, ale také může pokračovat na stejné úrovni, jako minulý segment uvozený stejnou podřadící spojkou.

```
L0 |Zdravotníky pobouřilo|i samotné Zumovo přiznání|                    |             |
L1                                                |, že nepoužil kondom|             |, že
L2                                                                       |, ačkoli věděl|
-------------------------------------------------------------------------------------------
L0      |               %, je nakažená HIV|.
L1  žena|               %                 |
L2      |, se kterou spí%                 |
```

Segment *že žena* by měl být o úroveň níže, než předchozí segment *ačkoli věděl*, bohužel programově se nedá rozhodnout, zda se náhodou nejedná o souřadné spojení vět vedlejších, proto je vygenerovaná i tato chybná varianta.

4.4.3. Několikanásobné větné členy

Problém s několikanásobnými větnými členy nastává, pokud člen je mezi dvěma klauzemi, a není jasné, jestli patří do levé, nebo pravé klauze. Názorně je to vidět na následující větě. Takto vypadá správný diagram:

```
L0 |" Vrazili do kanceláří%, do kuchyně%, noclehárny%i do skladu potravin|, rozbili všechno|
L1                                                                                        |,
-------------------------------------------------------------------------------------------
L0                 %, taky klimatizaci%a vlastně všechno|               |.
L1  co bylo ze skla%                   %                |, co rozbít šlo|
===========================================================================================
```

Následující diagram je skoro stejný, jen jsou jinak seskupené klauze

```
L0 |" Vrazili do kanceláří|, do kuchyně%, noclehárny%i do skladu potravin%, rozbili všechno|
L1                                                                                        |,
-------------------------------------------------------------------------------------------
L0                 |, taky klimatizaci%a vlastně všechno|               |.
L1  co bylo ze skla|                   %                |, co rozbít šlo|
===========================================================================================
```

Při správném čtení věty máme první 2 klauze

- klauze 1: Vrazili do kanceláří, do kuchyně, noclehárny i do skladu potravin
- klauze 2: rozbili všechno

Na spodním diagramu vypadají první dvě klauze takto:

- klauze 1: Vrazili do kanceláří
- klauze 2: do kuchyně, noclehárny i do skladu potravin , rozbili všechno

Bohužel opět nelze strojově rozeznat, které čtení je správné. Intuitivně by se dalo předpokládat, že čárkou oddělené členy budou patřit spíš k levé klauzi, bohužel vždycky to pravda není, jak je vidět na segmentu *taky klimatizaci*.

Ještě přikládám jeden, možná o trochu názornější příklad:

```
L0 |Není náhodou|
L1              |, že taková forma reakce trhu v podobě růstu cen na tuto strategickou
-------------------------------------------------------------------------------------------
L0                                                 %            %      %
L1  komoditu je většinou spojená s jinými událostmi%- pro občana%, stát%i svět tragického
-------------------------------------------------------------------------------------------
L0      |                        %         %       %      |.
L1  rázu|- jako je zemětřesení%, povodně%, válka%a jiné|
===========================================================================================
```

Zde je vidět, že kterýkoliv segment od segmentu – *pro občana* doprava může být na úrovni L0. Proto je rovněž vygenerován například tento diagram:

```
L0 |Není náhodou|
L1              |, že taková forma reakce trhu v podobě růstu cen na tuto strategickou
-------------------------------------------------------------------------------------------
L0                                                 %            %, stát%i svět tragického
L1  komoditu je většinou spojená s jinými událostmi%- pro občana%      %
-------------------------------------------------------------------------------------------
L0  rázu%- jako je zemětřesení%, povodně%, válka%a jiné|.
L1      %                      %         %       %      |
===========================================================================================
```

Hodně chybných diagramů je rovněž způsobeno slovesem *je*, které může být zároveň zájmenem.

4.4.4. Nejednoznačnost morfologické analýzy

Nejednoznačnost morfologické analýzy se v menší míře projeví už ve fázi tvorby segmentace, pokud nějaký slovní tvar může být považován za separátor, je za něj považován a většina separátorů způsobuje nejednoznačnosti výsledku.

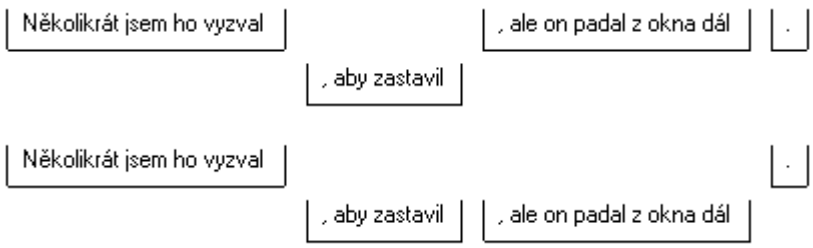
Ve významější míře se pak projeví při testování klauzí na přítomnost právě jednoho slovesa v určitém tvaru. Například slovesný tvar *je* (3. os. jedn. č. slovesa být) může být také zájmenem, (4. p. zájmena oni), takže z výskytu dvou tvarů slova *je* v jedné klauzi nelze vyvodit nekorektnost této klauze.

4.4.5. Kombinace nejednoznačností

Bohužel nejednoznačnosti se mezi sebou kombinují, proto se u delších vět dá očekávat exploze možných segmentačních diagramů. Věty výše jsem vybral z korpusu jako věty s velkým počtem možností, u většiny těchto vět se kombinuje více příčin nárůstu počtu výsledků.

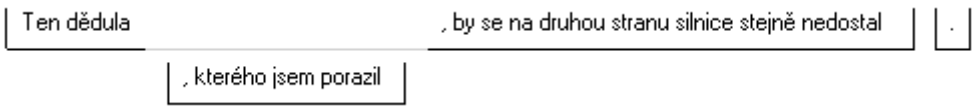
4.4.6. Jednodušší věty

Aby tato kapitola nevyzněla tak negativně, schoval jsem si na závěr pár příkladů, které se zanalyzují (skoro) jednoznačně. Pokud to délka věty dovolí, používám pravé obrázky, ne jen znakové napodobeniny. V tomto případě jsou ukázány všechny výsledky, které analýza vrátila.

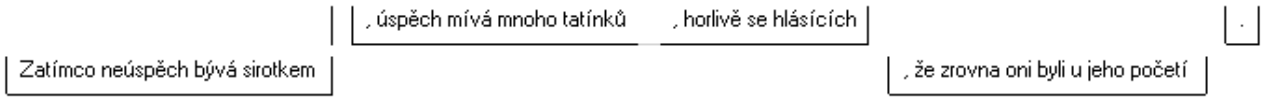


První věta ještě jednoznačná není, a to kvůli souřadící spojce za čárkou na úrvovni větší než nula (zde spojka „ale“). Která z uvedených variant je správná, je zjevné[13].

13 Ano, je to ta první



Ovšem tato věta se už zanalyzuje jednoznačně, přestože je použit nejednoznačný separátor čárka – díky podmínce na klauze, jež musí obsahovat sloveso. Nakonec ukážu 4-segmentovou větu převzatou z [1], která se také zanalyzuje jednoznačně.



5. Uživatelská dokumentace

Segmentační analýza je implementována sadou skriptů spouštěných z příkazového řádku. Skripty jsou napsané v Pythonu, ke svému běhu vyžadují instalaci Pythonu (http://www.python.org) + knihovny lxml (http://pypi.python.org/pypi/lxml/) pro zpracování XML. Morfologický analyzátor je napsán v Perlu, jehož instalace je tedy nutná. Na operačním systému je tento software nezávislý.

5.1. Hromadné zpracování

Hromadné zpracování vyžaduje následující strukturu adresářů:

- adresář data
  - podadresář 0-input – obsahuje věty určené k analýze v kódování UTF-8
  - podadresář 1-morf – obsahuje věty po morfologické analýze ve formátu XML/MORF
  - podadresáš 2-seg – obsahuje věty po segmentační analýze ve formátu XML/SEG
  - podadresáš 3-pres – obsahuje ASCII obrázky vět ve formátu TEXT/SEG
  - podadresáš 2-gold – obsahuje pouze správné varianty segmentační analýzy formátu XML/ANOT (tyto varianty jsou ověřeny člověkem, používají se k ověření správnosti algoritmu během vývoje). Souboru mají příponu anx.
  - podadresáš 3-gold – obsahuje pouze správné varianty segmentační analýzy formátu TEXT/SEG

5.2. Utilita make

Příkazem **makeall.py** se vygeneruje obsah adresářů 1-morf, 2-seg a 3-pres. Z jednoho souboru v adresáři 0-input vznikne stejnojmenný soubor v ostatních adresářích (lišící se příponou). Program makeall.py bere v úvahu datum poslední změny souboru, proto pokud změníme vstupní soubor v adresáři 0-input a ostatní soubory necháme beze změny, přenegerují se jen soubory vzniklého z tohoto vstupu. Popis parametrů lze získat pomocí přepínače --help.

```
usage: makeall.py [options]

options:
  -h, --help      show this help message and exit
  --omitclauses   Omits clause processing
  -s, --forceseg  Forces performing segmentation analysis for all files
```

5.3. Morfologická analýza

Morfologická analýza se provádí skriptem **runmorf.py**, který jako vstup dostane text v libvolném kódování a jako výstup vrátí XML/MORF soubor s provedenou morfologickou anaýzou. Tento skript interně pouští morfologický analyzátor prof. Hajiče [5]. Popis parametrů lze získat pomocí přepínače --help.

```
usage: runmorf.py [options]

options:
  -h, --help            show this help message and exit
  -i INFILE, --infile=INFILE
                        Input file in raw text format
  -o OUTFILE, --outfile=OUTFILE
                        Output file in raw XML format
  -e ENCODING, --encoding=ENCODING
                        Encoding of input file
```

5.4. Segmentační analýza

Segmentační analýza se provádí skriptem **seganal.py**, který jako vstup dostane soubor ve formátu XML/MORF (výstup z **runmorf.py**), XML soubor se seznamem separátorů (formát XML/SEP) a XML soubor se seznamem podmínek na kluze (ve formátu XML/CHECK). Výstupem je soubor formátu XML/SEG obsahující všechny segmentační diagramy. Popis parametrů lze získat pomocí přepínače –help.

```
usage: seganal.py [options]

options:
  -h, --help            show this help message and exit
  -i INFILE, --infile=INFILE
                        Input file in XML format (after mofrological analysis)
  -o OUTFILE, --outfile=OUTFILE
                        Ouput file in XML
  -s SEPFILE, --sepfile=SEPFILE
                        File with separator list in XML
  -c CHECKFILE, --checkfile=CHECKFILE
                        File with checkers list in XML
  -l LOGFILE, --logfile=LOGFILE
                        Log file, implicitly stdout
  -m MAXRESULTS, --maxresults=MAXRESULTS
                        Maximum number of temporary results; if number of
                        results is bigger
  --segsonly            Makes only dividing into segments, no joining into
                        clauses and level guessing
  --omitclauses         Omits clause processing
  --loglevel=LOGLEVEL   Log level, implicitly 5 (bigger log level=>more
                        vorbose logs)
```

5.5. Malování diagramů

Vzhledem k nízké názornosti segmentačních diagramů uložených ve formátu XML/SEG je možné použít utilitku **niceprint.py**. Tato utilitka vytvoří z jednoho XML/SEG nebo XML/ANOT souboru „obrázek“ (textový soubor znázorňující zanoření segmentů) ve formátu TEXT/SEG. Tento obrázek dál není možné nijak využít, slouží pouze pro názornější zobrazení segmentačního diagramu. Popis parametrů lze získat pomocí přepínače –help.

```
usage: niceprint.py [options]

options:
  -h, --help            show this help message and exit
  -i INFILE, --infile=INFILE
                        Input file in XML format, segmented text
  -t TOUTFILE, --toutfile=TOUTFILE
                        Ouput file in text format
  -w MAXWIDTH, --maxwidth=MAXWIDTH
                        Maximal width in characters
```

5.6. Statistika korpusu

Prográmek **stats.py** počítá statistiky korpusu uloženého ve standardní adresářové struktuřu popsané výše. Konktrétně prochází souboru z adresáře data/2-seg, pokud je zadán parametr **-g**, zpracovávají

se jen ty soubory z adresáře data/2-seg, které jsou zároveň v adresáři data/2-gold (s příponou anx). Neboli při zadání parametru -g se zpracovávají pouze věty, které jsou v korpusu v adresáři 2-gold. Popis parametrů lze získat pomocí přepínače --help.

```
usage: stats.py [options]

options:
  -h, --help      show this help message and exit
  -g, --goldonly  Process only sentences, which are in gold standard
```

Program využívá STDOUT pro výpis statistiky a STDERR pro výpis informací o průběhu zpracování, proto přesměrování standardního výstupu do souboru prostředky operačního systému vytvoří rozumný výpis.

5.7. Kontrola správnosti

Příkazem **goldcheck.py** nebo **goldcompare.py** se dá zkontrolovat, zda ve výsledku segmentační analýzy je správný výsledek, který je uložen ve formátu XML/ANOT. **goldcheck.py** vrací pouze výsledek ANO/NE, **goldcompare.py** vybere nejpodobnější segmentační diagram z souboru **testfile** a u toho testuje, kolik segmentů má stejnou úroveň, jako segmenty v souboru **goldfile**. Obě utilitky předpokládají, že všechny segmentační diagramy mají stejný počet segmentů.
Popis parametrů lze získat pomocí přepínače –help.

```
usage: goldcheck.py [options]

options:
  -h, --help            show this help message and exit
  -g GOLDFILE, --goldfile=GOLDFILE
                        File with correct segmentation
  -t TESTFILE, --testfile=TESTFILE
                        Tested file, corrent segmentation should be contained
```

5.8. Dávková kontrola správnosti

Příkazem **goldcheckall.py** se zkontrolují všechny soubory z adresáře 2-seg, pro které existuje stejnojmenný soubor v adresáři 2-gold. Kontroluje se správnost segmentace pomocí programu **goldcheck.py**.

5.9. Testování anotovaného korpusu

V případě, že máme anotovaný korpus (soubory ve formátu XML/ANOT), příkazem **testanot.py** lze otestovat, nakolik dávají výsledky segmentační analýzy stejné výsledky, jako jsou v tomto korpusu. Výsledkem je nejen počet správně analyzovaných vět, ale i počet segmentů, u kterých byla úroveň určená segmentační analýzou stejná jako úroveň určená člověkem. Skript **testanot.py** pro svou funkcionalitu používá pouští skripty **xml2text.py**, **runmorf.py**, **seganal.py** a **goldcompare.py**.
Popis parametrů lze získat pomocí přepínače –help.

```
usage: testanot.py [options]

options:
  -h, --help            show this help message and exit
  -i INFILE, --infile=INFILE
                        Input file in XML/ANOT format
  -d INDIR, --indir=INDIR
                        Input directory with files in XML/ANOT format
```

5.10. Extrakce vět z XML souborů

Máme-li soubor ve formátu XML/ANOT nebo XML/SEG, původní text z něho můžeme dostat pomocí skriptu xml2text.py.
Popis parametrů lze získat pomocí přepínače –help.

```
usage: xml2text.py [options]

options:
  -h, --help            show this help message and exit
  -i INFILE, --infile=INFILE
                        Input file in XML/ANOT or XML/SEG format
  -o OUTFILE, --outfile=OUTFILE
                        Output file in plain text file
  -e ENCODING, --encoding=ENCODING
                        Encoding of output file, default UTF-8
```

5.11. Převod formátu XML/SEG do XML/ANOT

O tento převod se stará skript **seg2anot.py**. Do formátu XML/SEG se vejde více variant segmentačních diagramů, zatímco formát XML/ANOT unese pouze jednu variantu, takže jako vstupní parametr je třeba kromě vstupního souboru ve formátu XML/SEG předat také variantu, která bude uložena do výsledného souboru. Varianta je identifikována atributem **pyid** v formátu XML/SEG, tatáž varianta je vypisována i ve fromátu TEXT/SEG.
Popis parametrů lze získat pomocí přepínače –help.

```
usage: seg2anot.py [options]

options:
  -h, --help            show this help message and exit
  -i INFILE, --infile=INFILE
                        Input file in XML/SEG format
  -o OUTFILE, --outfile=OUTFILE
                        Output file in XML/ANOT format
  -v VARIANT, --variant=VARIANT
                        Variant to select, if not given, infile must contain
                        one or zero variant per sentence
  -e, --extinfo         Include morfological tags and separator names in
                        result
  --indentoutput        Indent output XML
```

5.12. Import textu do formátu XML/ANOT

Tento import provádí utilita **anotimport.py**. Na vstupu dostane množinu textových souborů, na které pustí morfologickou analýzu, a provede segmentaci (rozdělení do segmentů), už ale neprovádí další fáze segmentační analýzy (generování diagramů). Ve výsledných souborech XML/ANOT je místa atributů **level** a **clausebeg** zástupný řetězec **???** naznačující, že tato data je třeba doplnit jinak (vetšinou anotátorem - člověkem, buď utilitou **segview**, nebo ručně přímo editací XML souboru). Tuto utilitu volá program segview.exe.
Popis parametrů lze získat pomocí přepínače –help.

```
usage: anotimport.py [options]

options:
  -h, --help            show this help message and exit
  -i INFILES, --infile=INFILES
                        Input file, can be specified more times
  -o OUTPREFIX, --outprefix=OUTPREFIX
                        Output directory and file prefix, result file names
                        are created by appending sentence number to this
                        prefix
  -e ENCODING, --encoding=ENCODING
                        Input file encoding
  -c COMMACOUNT, --commacount=COMMACOUNT
                        Minimal comma count
```

5.13. Přidání věty do „gold“ korpusu

Toto přidání provede velmi jednoduchý skript **setasgold.py**. Předpokládá existenci zanalyzované věty v adresáří data/2-seg, jejíž jedna varianta je správnou. Skriptu se předá jméno souboru v adresáři data/2-seg (bez cesty a bez přípony) a číslo správné varianty, výsledkem je stejnojmenný soubor v adresáři data/2-gold s přiíponou anx.
Popis parametrů lze získat pomocí přepínače –help.

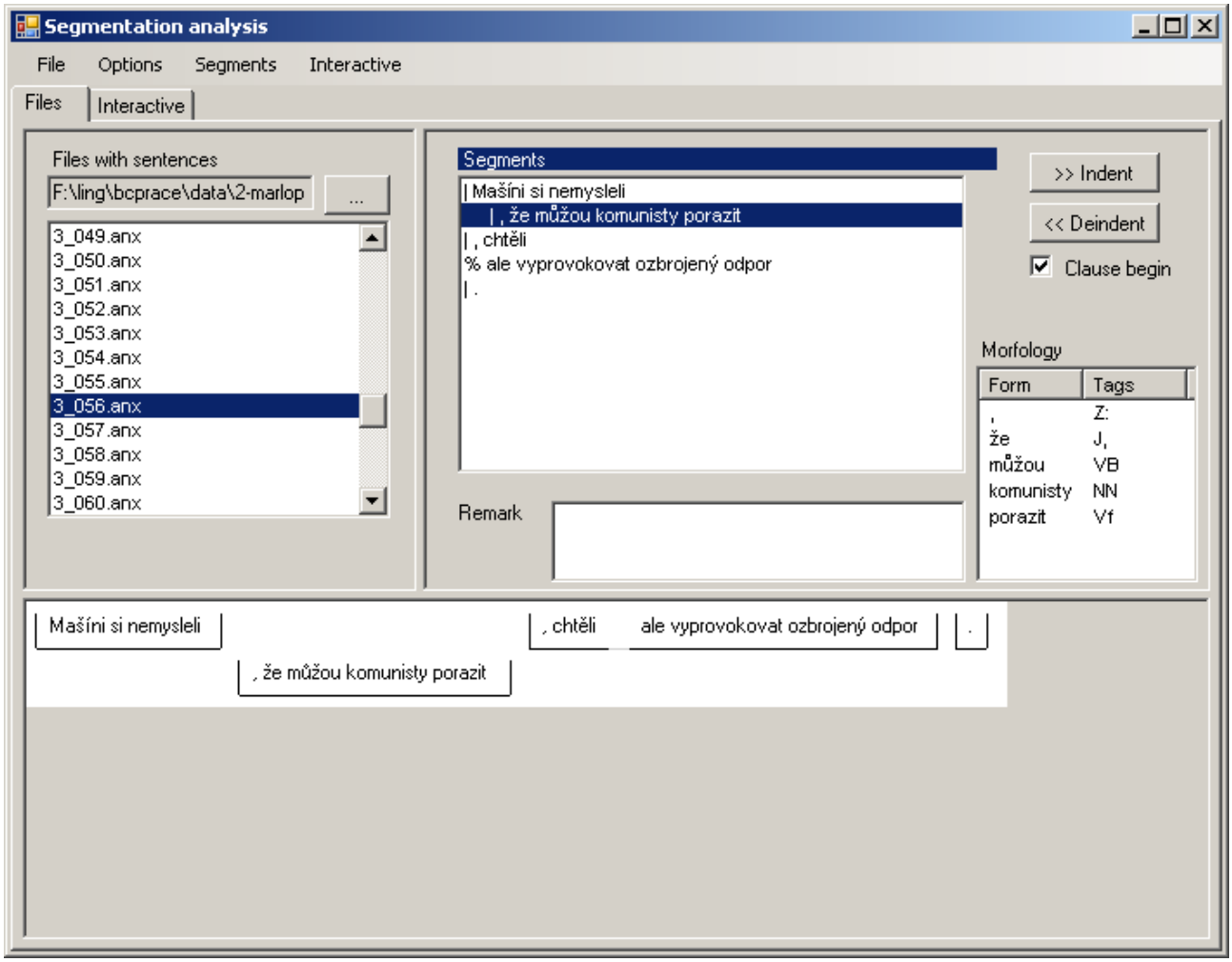
```
usage: setasgold.py [options]

options:
  -h, --help            show this help message and exit
  -g GOLDFILE, --goldfile=GOLDFILE
                        File with correct segmentation to be moved in 2-gold
  -v VARID, --varid=VARID
                        Correct variant ID, other variants will be deleted
```

5.14. Prohlížeč a editor segmentačních diagramů

Obsahem práce je také GUI program na prohlížení a úpravu segmentačních diagramů, **segview.exe**. Tento program využijí například anotátoři, připravující testovací data pro segmentační analýzu.

Program potřebuje pro svůj běh nainstalovanou knihovnu .NET 2.0[14]. Celý program je velmi jednoduchý, jak je vidět na následujícím obrázku:

14 .NET 2.0 je volně stažitelné běhové prostředí od Microsoftu, na Linuxu lze využít open-source ekvivalent Mono (http://www.**mono**-project.com/)



V levé části okna je seznam souborů uložených ve pracovním adresáři (ten se dá změnit tlačítkem označeným „...“).

V pravé části je seznam segmentů (rozdělení na segmenty se nedá měnit, dají se měnit jen vlastnosti jednotlivých segmentů). V dolní části je obrázková reprezentace segmentačního diagramu.

Program je uzpůsoben pro ovládání klávesnicí, klávesové zkratky jsou napsány v menu.

Stručně popíšu jednotlivé příkazy menu:

- File
  - Import sentences – do pracovního adresáře provede import vět z souboru, které uživatel vybere v následujícím dialogu. Předpokládá se, že soubory jsou kódovány v UTF-8. Tento příkaz vyžaduje přítomnost ostatních skriptů z projektu, kromě toho instalaci Pythonu, Perlu a lxml (viz kapitola „Softwarové požadavky“)
  - Save current sentence – uloží aktuální větu do souboru, ze kterého byla načtena
  - Save all – uloží všechny změněné věty (přepíše jejich původní umístění)
  - Export as image – uloží obrázkovou reprezentaci segmentačního diagramu aktuální věty
  - Previous – vybere předchozí soubor v pracovním adresáři
  - Next – vybere další soubor v pracovním adresáři
  - Exit – ukončí aplikaci

- Options
  - General – spustí konfigurační dialog
- Segments – provede operaci na vybraném segmentu
  - Indent – zvětší zanoření vybraného segmentu (tlačítko „>> Indent“)
  - Deindent – zmenší zanoření vybraného segmentu (tlačítko „<< Deindent“)
  - Toggle clause begin – změní příznak clausebeg (zda vybraným segmentem začíná klauze) (přepínač „Clause begin“)
- Interactive – slouží k interaktivnímu pouštění segmentační analýzi, je určen pro záložku „Interactive“
  - Run analysis – pustí analýzu na větu napsanou v editoru
  - Next result – přepne na další segmentační diagram
  - Previous result – přepne na předchozí segmentační diagram
  - Save image – uloží aktuální segmentační diagram
  - Add to working directory – přidá aktuální diagram do pracovního adresáře
  - Set current sentence as interactive – pustí segmentační analýzu na aktuální větu a přepne ji do okna „Interactive“

5.15. Softwarové požadavky

Celý projekt byl vyvíjen pod OS Windows, ale vzhledem k tomu, že je implementován v Pythonu, není důvod, proč by neměl fungovat pod Linuxem, případně po jinými OS.

Pro správnou funkčnost je potřeba mít nainstalováno:

- Python: http://www.python.org/download/releases/2.5.2/
- lxml: http://pypi.python.org/pypi/lxml/2.0.3
- Perl: http://www.activestate.com/Products/activeperl/features.plex

Python je moderní skriptovací programovací programovací jazyk, lxml je knihovna pro Python pro práci s XML, je potřeba pro pouštění všech skriptů. Perl je potřeba pro spuštění morfologické analýze prof. Hajiče, která je automaticky volána z příkazů popsaných výše.

Pro spuštění GUI prohlížeče segmentačníám diagramů **segview.exe** je potřeba mít nainstalovaný .NET 2.0, http://www.microsoft.com/downloads/details.aspx?FamilyID=0856EACB-4362-4B0D-8EDD-AAB15C5E04F5&displaylang=en

6. Technická dokumentace

6.1. Formáty souborů

Většina formátů souborů použitých v práci je nějaká specializace formátu XML. Pro popis formátu XML používám komentované Relax NG schéma[15], kompatkní syntaxi.

6.1.1. XML/MORF

Tento formát souboru slouží k uložení výsledků morfologické analýzy.

Relax NG schéma

```
root = element morfotext {
    element sentence {
        element word {
            ## použitý tvar slova
            attribute form {text},
            ## pořadí slova číslované od 0
            attribute index {text},
            ## možný výsledek morfologické analýzy
            element variant {
                ## základní tvar slova
                attribute lemma {text},
                ## morfologický tag
                attribute tag {text}
            } *
        } *
    } *
}

start = root
```

6.1.2. XML/SEG

Tento formát souboru slouží k uložení výsledků segmentační analýzy včetně všech možných segmentačních diagramů.

15 Více informací o Relax NG, včetně tutoriálů a validátorů, se dá nalézt na webu, http://relaxng.org/

Relax NG schéma

```
start = element segtext {
    ## pokud je clauses, segmentace byla dělána s podporou klauzí
    attribute mode {"clauses" | "noclauses"}?,
    element sentence {
        ## větný segment
        element segment {
            ## pořadí segmentu číslované od 1
            attribute segid {text},
            ## jméno separátoru, kterým začíná segment
            attribute separator {text}?,
            ## typ segmentu
            attribute segtype {text}?,
            ## slova, ze kterých se skládá separátor na začátku segmentu
            element sepwords {
                element_word *
            },
            ## slova, ze kterých se skládá segment (bez separátoru)
            element words {
                element_word *
            }
        } *,
        ## možný segmentační diagram
        element variant {
            ## ID diagramu (varianty)
            attribute pyid {text},
            ## ohodnocení segmentu v rámci diagramu
            element seg {
                ## ID segmentu (atribut segid v elementu segment)
                attribute id {text},
                ## úroveň segment (0: věta hlavní)
                attribute level {text},
                ## 0 | 1 , příznak, zda segmentem začíná klauze
                attribute clausebeg {text}
            } *
        } *
    } *
}

## význam stejný jako u elementu word ve formátu XML/MORF
element_word = element word {
    attribute form {text},
    attribute index {text},
    element variant {
        attribute lemma {text},
        attribute tag {text}
    } *
}
```

6.1.3. XML/ANOT

Tento formát slouží k uložení jednoho segmentačního diagramu jedné věty. Teoreticky je možné do tohoto formátu uložit více vět, ale v tuto možnost nevyužívám a ne všechny nástroje s ní počítají. Je používán pro uchování tzv. gold-kolekce vět (analýza provedená/zkontrolované člověkem). Prohlížení a editaci vět v tomto formátu je možné provádět pomocí GUI nástroje SegView .

Příklad věty uložené v tomto formátu vypadá následovně:

```
<root>
    <sentence>
        <segment level="0" clausebeg="1">
            <word form="Ale" sep="1"/>
            <word form="z"/>
            <word form="pohledu"/>
            <word form="ostatních"/>
        </segment>
        <segment level="1" clausebeg="1">
            <word form="," sep="1"/>
            <word form="kteří" sep="1"/>
            <word form="nespekulují"/>
        </segment>
        <segment level="0" clausebeg="0">
            <word form="," sep="1"/>
            <word form="se"/>
            <word form="to"/>
            <word form="často"/>
            <word form="považuje"/>
            <word form="za"/>
            <word form="jev"/>
            <word form="nedobrý"/>
        </segment>
        <segment level="0" clausebeg="0">
            <word form="," sep="1"/>
            <word form="protispolečenský"/>
        </segment>
        <segment level="0" clausebeg="0">
            <word form="," sep="1"/>
            <word form="někdy" sep="1"/>
        </segment>
        <segment level="0" clausebeg="0">
            <word form="až" sep="1"/>
            <word form="amorální"/>
        </segment>
        <segment level="0" type="terminator" clausebeg="1">
            <word form="."/>
        </segment>
    </sentence>
</root>
```

Relax NG schéma

```
start = element root {
    element sentence {
        element segment {
            attribute type {"terminator" | "parentheses"}?,
            ##príznak, zda timto segmentem zacina klauze
            attribute clausebeg {"0" | "1"},
            ## uroven zanorení segmentu
            attribute level {xsd:integer},
            element word {
                ## slovni tvar
                attribute form {text},
                ## priznak, zda je slovo soucast separatoru
                attribute sep {"0" | "1"}?
            } *
        } *
    } *
}
```

6.1.4. TEXT/SEG

Formát TEXT/SEG je textovou reprezentací diagramů z formátu XML/SEG. Je určen pro pohodlné čtení člověkem, není vstupem žádného skriptu. Tento formát tedy udržuje všechny segmentační diagramy pro vstupní větu.

```
VARIANT 11423960
L0 |Asketický generál Čamlong Srimuang|
L1                                    |, který už v roce 1992 vedl lidové povstání proti
-------------------------------------------------------------------------------------------
L0                |                                        %, tvrdí|
L1  vládnoucí moci|, při němž bylo zabito kolem 50 lidí%       |, že tentokrát se nic
-------------------------------------------------------------------------------------------
L0                            |.
L1  podobného nebude opakovat|
===========================================================================================
VARIANT 11424240
L0 |Asketický generál Čamlong Srimuang|
L1                                    |, který už v roce 1992 vedl lidové povstání proti
L2
-------------------------------------------------------------------------------------------
L0                |                                        %, tvrdí|
L1  vládnoucí moci|                                        %       |, že tentokrát se nic
L2                |, při němž bylo zabito kolem 50 lidí%       |
-------------------------------------------------------------------------------------------
L0                            |.
L1  podobného nebude opakovat|
L2                            |
===========================================================================================
```

Hloubka zanoření segmentu je zde dána řádkou, ve které je segment umístěn (L0 – věta hlavní …), „|“ slouží jako oddělovač klauzí, „%“ slouží jako oddělovač segmentů, které tvoří dohromady jednu klauzi.

6.1.5. XML/SEP

Tento formát slouží pro uložení definice separátorů[16].

16 Typicky se jedná o soubor separators.xml

Relax NG schéma

```
start = element separators {
  (
    element separator {
        (attribute t1 {text} | attribute f1 {text}),
        (attribute t2 {text} | attribute f2 {text})?,
        (attribute t3 {text} | attribute f3 {text})?,
        (attribute t4 {text} | attribute f4 {text})?,
        attribute newlevel {text}?,
        attribute newlevel_noclause {text}?,
        attribute name {text},
        attribute segtype { "parentheses" | "terminator" }?,
        (
        element condition {
          attribute type {"eq_prev_separator"},
          attribute levelname {text}
        }
         |
        element condition {
          attribute type {"find_prev_separator"},
          attribute levelname {text},
          attribute sepname {text}
        }
        )?
    }?,
    element start {
        (attribute t1 {text} | attribute f1 {text}),
        (attribute t2 {text} | attribute f2 {text})?,
        (attribute t3 {text} | attribute f3 {text})?,
        (attribute t4 {text} | attribute f4 {text})?,
        attribute level {text},
        attribute name {text}
    }?
  )*
}
```

- elementem separator se definuje separátor, který odděluje dva segmenty nebo ukončuje celou větu
- elementem start se definuje separátor, který uvozuje první segment
- atributy ti obsahují regulární výraz, který se pouští na tagu, fi obsahuje regulární výraz, který je použit na slovní tvar (i=1,2,3,4); pro jedno i vždy se zadává jen jeden z atributů ti, fi. Délka separátoru je dána počtem těchto atributů (čísluje se od jedničky)
- atribut newlevel obsahuje výraz, jehož vyhodnocením se spočítá množina úrovní dalšího segmentu, kterým začíná klauze
  - za proměnnou current se dosadí úroveň aktuálního segmentu
  - výraz může obsahovat
    - buď výčet hodnot oddělených čárkou
    - nebo operátor intervalu .. (low..hi)
    - nebo jednu hodnotu
  - podporovány jsou aritmetické operátory a vestavěné funkce Pythonu
- atribut newlevel_noclause obsahuje výraz, jehož vyhodnocením se spočítá množina úrovní dalšího segmentu, kterým nezačíná klauze
- atribut name obsahuje jméno separátoru
- nepovinný element condition obsahuje podmínku, která musí být splněna, aby mohl být separátor použit, typ podmínky se určí atributem type
  - podmínka typu eq_prev_separator je splněna, pokud existuje vlevo od testovaného

separátoru stejný separátor (stejným separárem se myslí separátor skladající se ze stejných tokenů), do proměnné, jejíž název je uveden v atributu levelname, se uloží úroveň tohoto separátoru, tato hodnota se pak dá použít ve výrazech v atributech newlevel a newlevel_noclause

- podmínka typu find_prev_separator je splněna, pokud existuje vlevo od testovaného separátoru separátor, jehož jméno je uvedeno v atributu sepname. Atribut levelname se využívá stejně jako u předchozí podmínky.

6.1.6. XML/CHECK

Tento formát slouží k uložení definice tzv. checkerů, tedy podmínek, jaké musí splňovat výstupní diagram.[17] Existují dva základní typy checkerů, jeden pro kontrolu klauzí a druhý pro kontrolu celých souvětí.

Pro kontrolu klauzí je možné užít následující podmínky:

- or – složená podmínka, musí být splněna aspoň jedna podmínka uvedena jako dětský element této podmínky
- and – složená podmínka, musí být splněna všechny podmínky uvedené jako dětské elementy této podmínky
- testone – podmínka je splněna, pokud tokeny, které jsou uvedeny jako dětské elementy elementu testone, mohou být[18] obsaženy v klauzi právě jednou
  - element token může obsahovat atributy tag a lemma, což jsou regulární výrazy testované proti tagu/lemmatu
- segtype – obsahuje atribut segtype; podmínka je splněna, pokud klauze obsahuje aspoň jeden segment, který má typ zadaný v atributu segtype

Pro kontrolu celých souvětí je možné užít následující podmínky:

- parentheses – kontroluje, zda je ve větě korektná uzávorkování (tj. zda například segmenty uvnitř závorek nejsou výše než závorky)

Takto vypadá soubor checkers.xml používaný v tomto projektu:

17 Typicky se jedná o soubor checkers.xml

18 Vzhledem k nejednoznačnosti morfologické analýzy se nedá zjistit, zda tokeny jsou v klauzi obsaženy právě jednou, proto podmínka je splněná, kdykoliv se to nedá vyloučit

```
<checkers>
    <clause>
        <!-- zde jsou uvedeny vsechny checkery vztahujici se na klauzi -->
        <or>
            <!-- veta obsahuje prave jedno urcite sloveso -->
            <testone>
                <token tag='V[^fcs]'/>
            </testone>

            <!-- veta obsahuje sloveso v minulem case s pomocnym tvarem slovesa
byt  -->
            <testone>
                <token tag='V[^fcs]'/>

                <!--
                <token lemma='být' tag='VB'/>
                aby to fungovalo i bez lemmatu z morfologicke analyzi
                -->
                <token form='jsem|jsi|jsme|jste'/>

            </testone>

            <!-- typ segmentu je uzavorkovany text -->
            <segtype segtype='parentheses'/>

            <!-- typ segmentu je uzavorkovany konec vety -->
            <segtype segtype='terminator'/>
        </or>
    </clause>

    <sentence>
        <!-- zde jsou uvedeny vsechny chekery vztahujici se na cele souveti -->
        <parentheses open='par_open' close='par_close'/>
    </sentence>
</checkers>
```

6.2. Zdrojový kód – základní přehled

6.2.1. Morfologická analýza

Morfologická analýza pouští externí analyzýtor prof. Hajiče, který se je uložen v perlovém skriptu **FMAnawin.pl**. Tento skript využívá slovník uložený v souboru **CZE-a.il2** . Pythonovské rozhraní pro morfologickou analýzu je ve skriptu **runmorf.py**, pro parsování CSTS souboru se využívají třídy definované v modulu **cstsfile.py**

6.2.2. Segmentační analýza

Hlavní skript je uložen v souboru **seganal.py** . Používají se tyto moduly:

- **preseg.py**[19]
    - algoritmus rozdělení na segmenty (funkce process_sentence)
    - třídy pro definici separátoru (Separator, Start (=separátor před prvním segmentem), End)
    - bázová třída SeparatorCondition slouží pro definici podmínky, za jaké smí být separátor

19 Název preseg.py je použit z historických důvodů, kdy jsem segmentům říkal presegmenty a klauzím segmenty. Bohužel uvnitř zdrojových kódů ještě někde tato chybná terminologie zůstala zachována

použit
    - bázová třída Match reprezentuje výraz testující jeden token separátoru
- **morfotext.py** – obsahuje třídy sloužící k načtení souboru s výsledky morfologické analýzy
- **gen.py** – obsahuje generátor segmentačních diagramů
- **sentfilter.py** – obsahuje funkci sentences_of_variant, která ze segmentačního diagramu vytvoří seznam klauzí; funkce acceptable pak pomocí checkeru testuje všechny klazue
- **checkers.py** – obsahuje definice checkerů, pomocí funkce load_checkers se načítá soubor checkers.xml

6.2.3. Ostatní skripty

Ostatní skripty jsou poměrně jednoduché, jsou implementovány v jednom souboru., případně pouštějí jiné skripty projektu. Podrobný návod k použití je v části věnující se uživatelské dokumentaci. Jedná se o tyto skripty:

- **anotimport.py** – import textových souborů do formátu XML/ANOT (pouští morfologickou analýzu a část segmentační analýzy)
- **goldcheck.py** – kontrola, zda soubor formátu XML/SEG obsahuje segmentační diagram uložený ve formátu XML/ANOT
- **goldcompare.py** – test anotovaného diagramu oproti diagramům vytvořeným segmentační analýzou
- **goldcheckall.py** – dávková kontrola všech souborů
- **makeall.py** – zpracování všech zdrojových souborů, které nově přibily, případně změněných souborů
- **niceprint.py** – převede soubor ve formátu XML/SEG nebo XML/ANOT do snadno čitelného textového formátu
- **seg2anot.py** – převede soubor ve formátu XML/SEG do formátu XML/ANOT
- **setasgold.py** – zkopírování segmentačního diagramu do „gold“ korpusu
- **stats.py** – statistiky segmentačních diagramů
- **xml2text.py** – extrakce původního textu z XML souborů
- **testanot.py** – test anotovaného korpusu

6.2.4. GUI editor diagramů

Tento editor je napsán v jazyce C# 2.0, projekt pro Visual Studio 2005 je uložen v adresáři segview.

Uvádím seznam souborů s jejich stručnou charakteristikou:

- **MainForm.cs** – hlavní okno programu
- **Options.cs** – třída obsahující vlastnosti, které se upravují v menu Nastavení. V tomto menu se automaticky zobrazí všechny public vlastnosti této třídy.
- **OptionsForm.cs** – okno pro nastavení konfigurace programu
- **Program.cs** – hlavní program
- **Sentence.cs** – datová struktura pro uložení věty (načítá se z souboru XML/ANOT)

- **SentencePanel.cs** – panel pro zobrazení segmentačního diagramu věty
- **Toolkit.cs** – pomocné funkce

7. Závěr

Cílem této práce bylo implementovat rozdělení na segmenty, jak je popsáno v [1]. Vytvořil jsem sadu utilitek v Pythonu, které tento problém řeší. Kromě toho zde byla navržena a implementována metoda spojení segmentů do klauzí na základě velmi jednoduché definice, že klauze musí obsahovat právě jedno sloveso v určitém tvaru. Též byl vytvořen seznam separátorů pro češtinu, který plně pokryl soubor 62 vět, na základě testování s dalšími korpusy bylo ověřeno, že pro víceméně 100% pokrytí bude nutné tento seznam ještě rozšířit.

Otestování bylo provedeno na dvou korpusech o velikosti 35 vět (s úspěšností měřenou počtem správně ohodnocených segmentů 74.48 %) a 80 vět (zde bylo dosaženo úspěšnosti 96.61 % pro segmenty, 81.48 % pro celé věty).

Při implementaci byl kladen důraz na to, aby každý krok, který má smysl pouštět samostatně, bylo možné pustit příkazem z příkazové řádky. Celý projekt je víceméně systém utilitek, které se volají navzájem. Co se týče formátu dat, kde to bylo možné, bylo použito XML. Všechny používané formáty XML jsou popsány v části věnující se technické dokumentaci.

Poslední programátorskou částí práce je GUI prográmek, který dovoluje editovat segmentační diagramy. Využít se dá jak k prohlížení výsledků segmentační analýzy, tak k ruční anotaci jazykového korpusu.

V budoucnu by mohla být tato práce rozšířena o následující prvky:

- rozšíření anotovaného korpusu, potažmo doplnění seznamu separátorů, aby tento korpus byl úspěšně zpracován
- rozpoznávání typu vztahů mezi klauzemi, například typ věty vedlejší, poměr mezi větami hlavními, případně klasifikace segmentů (zda segment je několikanásobný větný člen, přístavek, neshodný přívlastek, oslovení...)
- navázání na syntaktickou analýzu, jako vstup do syntaktické analýzy by měly sloužit jednotlivé klauze
- více podmínek, které musí splňovat klauze, například pokud obsahuje zvratné sloveso, musí klauze též obsahovat příslušné zvratné zájmeno (je otázkou, zda toto určení nenechat do následujícího kroku – do syntaktické analýzy)
- dodání informací požadovaných segmentační analýzou do korpusu PDT (Prague Dependency Treebank)
- vylepšení GUI nástroje segedit.exe, například umožněním přímé editace nakreslení segmentačního diagramu v dolní části obrazovky

8. Přílohy

8.1. Morfologický tag - pozice 2 – slovní poddruh[20]

Na pozici duhé je upřesnění slovního druhu. V tabulce pod názvem POS+SubPOS uvádím první dvě hodnoty z morfologického tagu – slovní druh a poddruh.

POS+SubPOS	Popis	Příklady
	Spojky	
J,	Podřadící spojka	že, kdyby
J^	Souřadící spojka	a, nebo
	Zájmena	
P4	Vztažné zájmeno	jaký, který, čí
P5	Zájmeno za předložkou	něj, něho
P6	Vztažné zájmeno v dlouhém tvaru	sebe, sobě, sebou
P7	Vztažné zájmeno se, si	se, si, ses, sis
P8	Vztažné zájmeno svůj	svůj
P9	Vztažné zájmeno jenž, již, za předložkou	něhož, niž
PD	Ukazovací zájmeno	ten, onen
PE	Vztažné zájmeno „což“	což
PH	Osobní zájmeno, krátký tvar	mě, mi, ti mu
PJ	Vztažné zájmeno jenž, již, **ne** za předložkou	jenž, již
PK	Vztažné zájmeno „kdo“	kdo, kdož
PL	Zájmeno „všechen“, „sám“	všechen, sám
PM	Přídavné jméno odvozené od *verbal past transgressive form*	???
PQ	Vztažné zájmeno „co“	co, cožpak, copak
PS	Přivlastňovací zájmeno	můj, tvůj, jeho
PW	Záporné zájmeno	nic, nikdo, nijaký, žádný
PY	Vztažné zájmeno „co“ jako příklonka po předložce	oč, nač, zač
PZ	Neurčité zájmeno	nějaký, některý, čímkoli, cosi
	Číslovky	
C?	Číslovka „kolik“	kolik
C*	Číslovka „krát“	krát
C}	Číslo napsané římskými číslicemi	XIV
C=	Číslo napsané číslicemi	1234

20 Tato kapitola byla převzata z dokumentace k [4], kde lze také nalézt více podrobností

Ca	Neurčitá číslovka	mnoho, málo, tolik, několik
Cd	Číslovka s jmenným skloňováním	dvojí, desaterý
Ch	Číslovky „jedny“, „nejedny“	jedny, nejedny
Cj	Obecné číslovky větší nebo rovno 4, užívané jako podstatné jméno	čtvero, desatero
Ck	Obecné číslovky větší nebo rovno 4, užívané jako přídavné jméno	čtvery, desatery
Cl	Číslovka, kardinalita jeden, dva, tři, čtyři, půl, pokud není používáno jmenné skloňování	jeden, dva, tři, čtyři, půl
Cn	Číslovka, kardinalita větší nebo rovno 5	
Co	Číslovka, neurčitá násobná	mnohokrát, tolikrát
Cr	Číslovka, řadová	
Cu	Číslovka „kolikrát“	kolikrát
Cv	Číslovka násobná určitá	pětkrát
Cw	Číslovka, neurčitá, skloňování jako přídavné jméno	nejeden, tolikátý
Cy	Číslovka, zlomek končící -ina	sedmina
Cz	Číslovka „kolikátý“	kolikátý
Slovesa		
VB	Sloveso, přítomný nebo budoucí čas	pracuji
Vc	Podmíněný tvar slovesa být	by, bych, bys
Ve	Sloveso, přechodník přítomný (koncovky -e/-ě, -íc, -íce)	sedíce
Vf	Sloveso, infinitiv	pracovat
Vi	Sloveso, rozkazovací tvar	uklízej
Vm	Sloveso, přechodník minulý	udělav, udělaje
Vp	Sloveso, pčíčestí minulé, s příklonkou s	uslyšels
Vq	Sloveso, pčíčestí minulé, s příklonkou ť	
Vs	Sloveso, příčestí minulé, pasivum	rozdělený
Vt	Sloveso, přítomný nebo budoucí čas, s příklonkou -ť	
Interpunkce		
Z#	Ohraničení věty	. (tečka), ! (vykřičník)
Z:	Interpunkce kromě ohraničení věty	: (dvojtečka)
Podstatná jména		
N%	Signatura autora	haš-99_:B_;S
NN	Podstatné jméno – obecný tvar	bublina, hradu, kuřaty

Neznámý		
X@	Nerozpoznaný slovní tvar	edggew
Přídavné jméno		
AA	Obecné přídavné jméno	slunečný
AC	Jmenný tvar	rád, schopen
AG	Slovesný tvar přídavného jména odvozený od přechodníku přítomného	slyšící
AU	Přivlastňovací (s koncovkou -ův, -in)	otcův, matčin
Předložky		
RF	Část předložky nevyskytující se samostatně	nehledě (na), vzhledem (k)
RR	Předložka, bez vokalizace	s, z,v, pod
RV	Předložka s vokalizací	ve, pode, ku
Citoslovce		
II	Citoslovce	
Částice		
TT	Částice	
Příslovce		
Db	Příslovce, které se nadá negovat ani stupňovat	pozadu, naplocho
Dg ???	Příslovce, které se dá negovat nebo stupňovat	velký, zajímavý

8.2. Seznam separátorů

V následující tabulce je uveden úplný seznam separátorů (tento seznam je taktéž uveden v souboru separators.xml). Význam sloupečků je vysvětlen pod tabulkou.

Separátor	Jméno	NL	NL_nc	Popis
Souřadící separátory				
, *J\^*	comma_coord	0..current	0..current	souřadící spojka – s čárkou
*J\^*	coord	current	current	souřadící spojka – bez čárky
Podřadící separátory – s podřadící spojkou				
, *J\,*	comma_sub_multi	prevlev, current+1		„čárka“ podřadící spojka, vícenásobný vyskyt ve větě (prevlev je úroveň minulého výskytu)
, než	comma_nez	current+1	0..current-1	Podřadící nebo srovnávací spojka; pokud je na stejné úrovni, píše se bez čárky, proto chybí current
, jak	comma_jak	current+1	0..current	„jak“ je buď podřadící spojkou, pokud je jiným členem, neuvozuje klauzi
, *J\,*	comma_sub	current+1		„čárka“ podřadící spojka
, *J\, J\,*	comma_sub_sub	current+2		„čárka“ 2x podřadící spojka (např. „že když“)
Podřadící separátory – se vztažným zájmenem				
, *R[RV] P[49]*	prep_pronoun	current, current+1	current	„čárka“ + předložka + vztažné zájmeno
, *P[4EJKQY]*	comma_pronoun			„čárka“ + vztažné zájmeno
Podřadící separátory – s příslovcem				
, **kde|kam| kudy|kdy| jak|proč**	comma_adv	current+1		„čárka“ + vztažné příslovce
Podřadící separátory – s číslovkou				
, *C[\?uz]*	comma_num	current+1	current	„čárka“ + číslovka
Speciální interpunkce				
(	par_open	current+1		otevírací závorka
)	par_close		prevlev-1	Uzavírací závorka; prevlev je úroveň segmentu vpravo od příslušející otevírací závorky
:	colon	current		dvojtečka
;	semicolon	current		středník

\-|\–	dash	current	current	pomlčka (2 druhy), na rozdíl od středníku a dvojtečky nemusí oddělovat klauze
\.|\!|\?	sent_close	0		ukončení věty
Na začátku věty				
J\^ J\,	coord_sub_start	1		Podřadící spojka na začátku věty za souřadící spojkou
J\^ J\^ J\,	coord_sub_start	1		Podřadící spojka na začátku věty za dvěma souřadícími spojkami
J\,	sub_start	1		Podřadící spojka na začátku věty
Čárka				
,	comma	0..current	0..current	„čárka“

Popis sloupečků tabulky:

- Separátor – mezerou oddělené tokeny separátoru; pokud je token uveden **tučně**, jedná se o regulární výraz testovaný proti slovnímu tvaru, zatímco *kurzívou* je uveden regulární výraz, kterým se testuje morfologický tag
- Jméno – název separátoru užívaný v ostatních tabulkách
- NL – výraz, jehož vyhodnocením se spočítá množina úrovní následujícího segmentu, v případě, že následujícím segmentem začíná klauze
- NL_nc – výraz, jehož vyhodnocením se spočítá množina úrovní následujícího segmentu, v případě, že následujícím segmentem nezačíná klauze
- Popis – popis separátoru + poznámky

Seznam použité literatury

1: Kuboň, V., Lopatková, M., Plátek, M., Pognan, P, A Linguistically-Based Segmentation of Complex Sentences, 2007
2: Eva Hajičová, Jarmila Panevová, Petr Sgall, Úvod do teoretické a počítačové lingvistiky, 2003
3: WIKI, Wikipedie, 2002, http://cs.wikipedia.org
4: UFAL, Prague dependency treebank, 2006, http://ufal.mff.cuni.cz/pdt2.0/
5: Jan Hajič, Czech "Free" Morphology, 2000-2001, http://ufal.mff.cuni.cz/pdt/Morphology_and_Tagging/Morphology/index.html