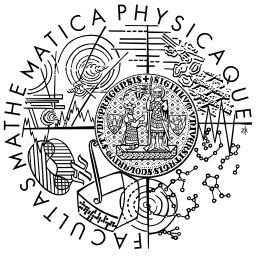


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Lenka Krippnerová

Identifikace složených gramatických tvarů

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: RNDr. Daniel Zeman, Ph.D.
Studijní program: Informatika

Praha 2024

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Děkuji vedoucímu této práce, RNDr. Danielu Zemanovi, Ph.D., za veškeré rady a pomoc. Dále děkuji své rodině a přátelům za trpělivost a psychickou podporu.

Název práce: Identifikace složených gramatických tvarů

Autor: Lenka Krippnerová

Ústav: Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: RNDr. Daniel Zeman, Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt: Cílem této práce je identifikovat a vhodně označit složené gramatické tvary v datech z projektu Universal Dependencies. Data z projektu Universal Dependencies jsou anotována na morfologické i syntaktické rovině, nicméně tyto anotace se vztahují pouze k jednotlivým slovům, složené tvary tedy nelze snadno vyhledat. Výstupem této práce je program, který na základě nastudovaných gramatických pravidel slovanských jazyků tyto složené gramatické tvary v datech slovanských jazyků objeví a označí. Dále program, jenž načte pravidla z konfiguračního souboru a na základě těchto pravidel v datech označí složené tvary. Konfigurační soubor je navržen ve formátu YAML, lze ho snadno upravovat v klasickém textovém editoru a je dostatečně jednoduchý na to, aby s ním dokázal bez větších obtíží pracovat i uživatel bez zkušeností s programováním.

Klíčová slova: anotovaný korpus, značkování, morfologie, syntax, zpracování přirozeného jazyka, universal dependencies

Title: Identification of periphrastic grammatical forms

Author: Lenka Krippnerová

Institute: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Daniel Zeman, Ph.D., Institute of Formal and Applied Linguistics

Abstract: The goal of this work is to identify and appropriately mark periphrastic grammatical forms in the data from the Universal Dependencies project. The data from the Universal Dependencies project are annotated on the morphological and syntactic level, however, these annotations relate only to individual words, so periphrastic forms cannot be easily searched. The output of this work is a program which, based on the studied grammatical rules of Slavic languages, discovers and marks these periphrastic grammatical forms in the data of Slavic languages. Furthermore, a program that reads the rules from the configuration file and, based on these rules, marks the periphrastic forms in the data. The configuration file is designed in YAML format, can be easily edited in a classic text editor, and is simple enough that even a user without programming experience can work with it without much difficulty.

Keywords: annotated corpus, tagging, morphology, syntax, natural language processing, universal dependencies

Obsah

Úvod

Tato práce pracuje s jazykovými daty z projektu Universal Dependicies. Data z tohoto projektu jsou oanotovaná na morfologické i syntaktické rovině — tedy například u určitého tvaru slovesa jsou uvedeny informace o osobě, čísle, způsobu, čase, slovesném rodu, vidu, ale i další morfologické rysy, pokud je toto sloveso vyjadřuje. Jelikož se ale tyto informace vztahují typicky k jednomu slovu, je poměrně obtížné vyhledat v datech složené gramatické tvary. Například ve větě *Byl bych to udělal už včera.* je podmiňovací způsob vyznačen jen u slova *bych*, nikde není zmíněno, že podmiňovací způsob je v této větě tvořen třemi slovy, a sice *byl*, *bych* a *udělal*.

Jedním z cílů práce je identifikovat složené slovesné tvary v oanotovaných datech z projektu Universal Dependicies pro slovanské jazyky. To obnáší nastudovat gramatická pravidla tvoření slovesných časů a způsobů v těchto jazycích a dále použít tato pravidla pro naprogramování několika bloků v jazyce Python s využitím rozhraní `Udapi`. Tyto bloky do dat doplní anotaci identifikovaných složených tvarů. Dalším cílem této práce je navrhnout obecná pravidla pro odhalení složených gramatických tvarů. Výstupem bude program, jenž načte konfigurační soubor s pravidly tvoření složených gramatických tvarů v daném jazyce ve stanoveném formátu a na základě těchto pravidel nalezne a oanotuje nalezené složené tvary. Tento konfigurační soubor by měl mít jednoduchý formát, aby i uživatel bez programátorských zkušeností dokázal bez větších obtíží správný konfigurační soubor napsat. Dále je cílem sjednotit anotaci v těch případech, kdy to dává smysl.

Tato práce je rozdělena do čtyř kapitol. V první kapitole si vysvětlíme, co je to jazykový korpus a k čemu ho lze v oblasti zpracování přirozeného jazyka využívat, dále si představíme projekt Universal Dependencies a formát CoNLL-U, což je formát, v němž jsou uložena data projektu Universal Dependencies. Ve druhé kapitole se seznámíme s gramatickými pravidly slovanských jazyků, v čem se odlišují a v čem se naopak shodují. Tyto poznatky jsem využila při programování jednotlivých bloků, které ve vstupních datech hledají a označují složené gramatické tvary. Ve třetí kapitole je popsána uživatelská dokumentace, tedy mimo jiné se zde dozvíme, jak získat vstupní data, jak jednotlivé bloky spustit a jak nastavit konfigurační soubor s pravidly pro hledání složených gramatických tvarů. Čtvrtá kapitola obsahuje programátorskou dokumentaci. Zde se tedy dočteme, jak jsou jednotlivé bloky naprogramovány, ale také jak obtížné je nasadit program na nový jazyk, jestli lze pro blízce příbuzné jazyky využívat stejnou sadu pravidel a nakonec jak je program úspěšný.

1 Universal Dependencies a formát CoNLL-U

1.1 Základní informace o Universal Dependencies

Nedílnou součástí oblasti zpracování přirozeného jazyka jsou jazykové korpusy. Jazykový korpus je *rozsáhlý soubor autentických textů (psaných nebo mluvených) převedený do elektronické podoby v jednom formátu tak, aby v něm bylo možné jednoduše vyhledávat jazykové jevy (zejména slova a slovní spojení)*[1].

Tyto korpusy jsou využívány jak v lingvistickém výzkumu, tak i v oblasti zpracování přirozeného jazyka. Zde se například využívají jako zdroje dat ve strojovém překladu, kontrolách pravopisu, vývoji chatbotů a v mnoha dalších oblastech. Texty v korpusech jsou často anotované. Tedy k jednotlivým jevům jsou doplňovány dodatečné lingvistické či strukturní informace jako například lemmata či gramatické vlastnosti[2]. Korpusy obsahující syntakticky anotované struktury vět v podobě závislostních stromů se nazývají treebanky[3].

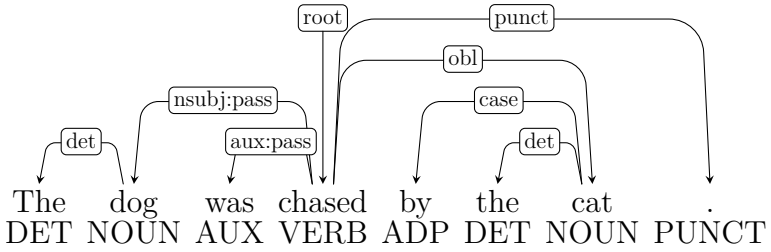
Universal Dependencies (UD) je projekt, který vyvíjí mezijazykově konzistentní anotaci treebanků pro mnoho jazyků. Cílem tohoto projektu je usnadnit vývoj vícejazyčných syntaktických analyzátorů, mezijazykové učení a výzkum parsování z pohledu jazykové typologie[4]. To ilustrují následující příklady syntakticky aonotovaných vět v angličtině (viz Obrázek 1.1), češtině (viz Obrázek 1.2) a švédštině (viz Obrázek 1.3), tyto příklady jsou převzaty ze stránek projektu [4].

V současné době obsahuje projekt Universal Dependencies (ve verzi 2.14 z května 2024) 283 treebanků pro 161 jazyků ze 31 jazykových rodin. Oficiální stránky projektu jsou dostupné na url adrese `https://universaldependencies.org/`.

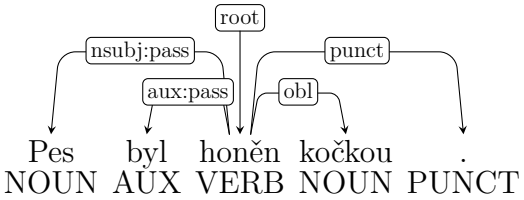
1.1.1 Morfologická a syntaktická anotace

Každé slovo z korpusů projektu Universal Dependencies je morfologicky i syntakticky oanotováno. Morfologická anotace slova (viz Tabuka 1.1) sestává z lemmatu, `UPOS` značky (viz Tabulka 1.2) a morfologických rysů (viz Tabulka 1.3). Lemma (tj. slovníkový tvar slova) nemusí být obsaženo povinně, lze místo něj uvést znak podtržítko („_“). `UPOS` značka je na rozdíl od lemmatu povinná, tedy nelze jako `UPOS` značku uvést znak podtržítko. Pokud se v seznamu `UPOS` značek vhodná značka nenachází, je třeba použít značku `X`. Seznam `UPOS` značek je fixní, nelze ho rozšiřovat tak, aby pokrýval jazykově specifické rysy, nicméně je možné, že se v některých jazycích některé `UPOS` značky nepoužijí. Morfologické rysy jsou stejně jako lemmata nepovinné, jedná se o doplňkové informace o slově. Každý morfologický rys je tvaru `Name=Value`. Každé slovo může obsahovat libovolný počet těchto rysů, jednotlivé rysy jsou odděleny znakem svislítko („|“)[5].

Syntaktická anotace (viz Obrázek 1.4) sestává z typovaných závislostních vztahů mezi slovy (přehled možných typů závislostních vztahů je dostupný na url adrese `https://universaldependencies.org/u/dep/index.html`). Závislosti jsou reprezentovány pomocí stromu, kde právě jedno slovo je hlavou věty



Obrázek 1.1 Syntaktická anotace anglické věty.



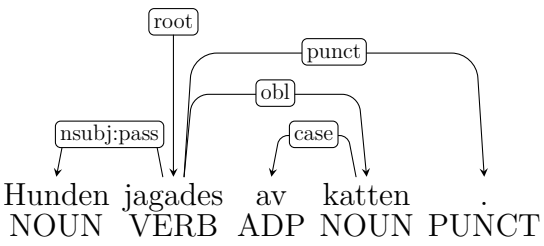
Obrázek 1.2 Syntaktická anotace české věty.

(`HEAD`), závislé na pomyslném kořeni (`ROOT`), všechna ostatní slova jsou závislá na jiném slově ve větě[6].

Tabulka 1.3 Přehled morfologických rysů. Přehled hodnot, kterých jednotlivé rysy nabývají, je dostupný na stránkách UD, na url adrese `https://universaldependencies.org/u/feat/all.html`.

FEAT	**definice**
`Abbr`	informace, zda se jedná o zkratku
`AdpType`	druh adpozice
`AdvType`	druh příslovce
`Animacy`	životnost
`Aspect`	slovesný vid; rys, který určuje dobu trvání akce v čase, zda byla akce dokončena atd.
`Case`	pád; obvykle flektivní rys podstatných jmen a v závislosti na jazyce dalších slovních druhů (zájmena, přídavná jména, …), které označují shodu s podstatnými jmény

Pokračování na další straně



Obrázek 1.3 Syntaktická anotace švédské věty.

Tabulka 1.3 – *Pokračování z předchozí strany*

FEAT	**definice**
`Clusivity`	rys osobních zájmen v první osobě v množném čísle; vyjadřuje, zda je posluchač zahrnut do výpovědi, nebo není
`ConjType`	druh spojky
`Definite`	určitost; jeho hodnota rozlišuje, zda mluvíme o něčem známém a konkrétním, nebo o něčem obecném či neznámém; může být označena na určitých a neurčitých členech nebo přímo na podstatných jménech, přídavných jménech atd.
`Degree`	stupeň; typicky flektivní rys některých přídavných jmen a příslovcí
`Deixis`	rys typicky ukazovacích zájmen, determinantů a příslovcí; jeho hodnota klasifikuje umístění odkazované entity s ohledem na umístění mluvčího nebo posluchače
`DeixisRef`	osoba, ke které je Deixis relativní; Deixis zakóduje pozici entity vzhledem k mluvčímu nebo posluchači, pokud je nutné rozlišit osobu, jejíž poloha je referenčním bodem, použije se DeixisRef
`Evident`	morfologické označení zdroje informací mluvčího
`Foreign`	informace, zda se jedná o cizí slovo
`Gender`	jmenný rod
`Hyph`	informace, zda se jedná o část složeniny se spojovníkem; v závislosti na tokenizaci může být složenina jedním tokenem nebo může být rozdělena na několik tokenů, pak tokeny potřebují značky
`Mood`	slovesný způsob
`NameType`	druh pojmenované entity
`NounClass`	třída jmen
`NumForm`	informace, zda je číslo vyjádřené číslicemi nebo slovem
`NumType`	druh číslovky
`Number`	číslo (např. jednotné, množné, duál, …)
`PartType`	druh částice
`Person`	osoba (např. 1, 2, 3, …)
`Polarity`	informace, zda se jedná o zápor
`Polite`	zdvořilost; různé jazyky mají různé prostředky k vyjádření zdvořilosti nebo respektu, některé prostředky jsou morfologické
`Poss`	informace, zda je slovo přivlastňovací
`PrepCase`	osobní zájmena mají v některých jazycích různé tvary v závislosti na tom, zda se před nimi nachází předložka (např. v češtině zájmeno *jemu* x *němu*)
`PronType`	druh zájmena
`PunctSide`	rys, který rozlišuje mezi počáteční a konečnou formou párové interpunkce
`PunctType`	druh interpunkce
`Reflex`	informace, zda je slovo zvratné
`Style`	styl nebo podjazyk, ke kterému tvar slova patří
`Tense`	čas; typický rys sloves; může se také vyskytovat u jiných slovních druhů (podstatná jména, přídavná jména, příslovce)

Pokračování na další straně

Tabulka 1.3 – *Pokračování z předchozí strany*

FEAT	**definice**
`Typo`	informace, zda se jedná o chybný nebo typograficky neočekávaný tvar slova
`VerbForm`	tvar slovesa nebo deverbativa
`VerbType`	druh slovesa
`Voice`	slovesný rod

1.2 Základní popis formátu CoNLL-U

Formát CoNLL-U je textový formát dat, v němž jsou uložena data z Universal Dependencies. Tento formát předpokládá kódování UTF-8 a obsahuje následující typy řádků:

- řádek s anotacemi tokenu
- řádek pro tzv. „víceslovné tokeny“ a pro abstraktní uzly, jedná se o podtyp řádku s anotacemi
- prázdný řádek ukončující větu
- řádek s komentáři na začátku věty

Každý řádek s anotacemi tokenu je rozdělen na deset polí. V následujícím seznamu jsou vypsána pole se svými názvy a krátkým popisem v tomtéž pořadí, jako požaduje formát CoNLL-U.

- **ID**: číslo označující pořadí slova/tokenu ve větě, přičemž číslování začíná od jedničky, může se jednat i o interval (v případě víceslovných tokenů) nebo o desetinné číslo (v případě prázdných uzlů)
- **FORM**: tvar slova nebo interpunkční znaménko
- **LEMMA**: základní (slovníkový) tvar slova nebo kmen slova
- **UPOS**: slovní druh (jednotná sada značek pro všechny jazyky)
- **XPOS**: slovní druh (nepovinná značka ze sady specifické pro daný treebank)
- **FEATS**: seznam morfologických vlastností
- **HEAD**: rodič slova ve stromové reprezentaci věty, hodnotou je buď hodnota ID rodiče, nebo 0
- **DEPREL**: závislostní vztah vzhledem k HEAD
- **DEPS**: rozšířený závislostní graf ve formě seznamu párů HEAD-DEPREL
- **MISC**: jakákoli další anotace

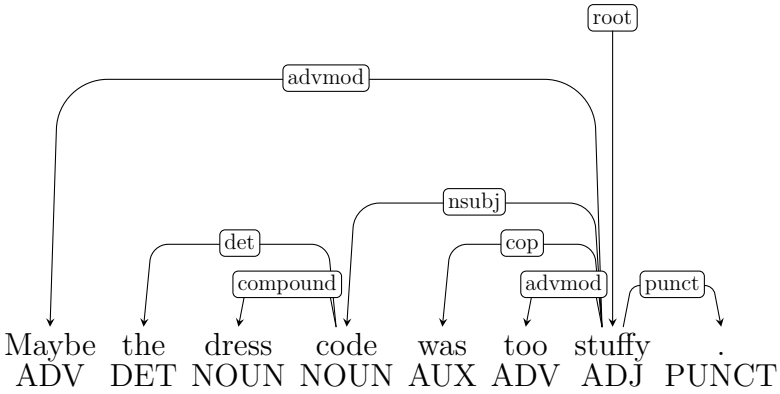
ID	FORM	LEMMA	UPOS	FEATS
1	Maybe	maybe	ADV	_
2	the	the	DET	Definite=Def|PronType=Art
3	dress	dress	NOUN	Number=Sing
4	code	code	NOUN	Number=Sing
5	was	be	AUX	Mood=Ind|Number=Sing|Person=3|...
6	too	too	ADV	_
7	stuffy	stuffy	ADJ	Degree=Pos
8	.	.	PUNCT	_

Tabulka 1.1 Příklad morfologické anotace.

UPOS	anglický název	český název
`ADJ`	adjective	přídavné jméno
`ADP`	adposition	adpozice[a]
`ADV`	adverb	příslovce
`AUX`	auxiliary	pomocné slovo
`CCONJ`	coordinating conjunction	souřadicí spojka
`DET`	determiner	determinující výraz v platnosti členu
`INTJ`	interjection	citoslovce
`NOUN`	noun	podstatné jméno
`NUM`	numeral	číslovka
`PART`	particle	částice
`PRON`	pronoun	zájmeno
`PROPN`	proper noun	vlastní jméno
`PUNCT`	punctuation	interpunkce
`SCONJ`	subordinating conjunction	podřadicí spojka
`SYM`	symbol	symbol
`VERB`	verb	sloveso
`X`	other	jiné

Pozn: [a] Adpozice je souhrnný název pro prepozice (předložky) a postopozice (adpozice, které jsou kladeny za jméno), postpozice se v češtině nevyskytují, má je ale např. němčina (*des Vaters* ***wegen***)[7].

Tabulka 1.2 Přehled hodnot atributu `UPOS`.



Obrázek 1.4 Příklad syntaktická anotace.

Řádky začínající znakem # jsou považovány za řádky obsahující komentáře nebo metadata relevantní pro následující větu, tyto řádky jsou zakázány uvnitř vět. Každá věta musí nutně obsahovat atributy `sent_id` (unikátní id v rámci celého souboru) a `text` (reprezentující text samotné věty). Za každou větou (tedy za posledním řádkem s anotacemi tokenu z dané věty) se musí vyskytovat právě jeden prázdný řádek, včetně poslední věty v souboru.

Detailnější popis formátu, který ale není stěžejní pro tuto práci, je dostupný na oficiálních stránkách projektu Universal Dependicies na url adrese `https://universaldependencies.org/format.html`. Z tohoto zdroje také vychází tato kapitola[8].

2 Slovanské jazyky

Jelikož se morfologická anotace v Universal Dependencies vztahuje vždy ke konkrétnímu výskytu jednoho slova, nejsou zde vůbec zaznamenány rysy, které by náležely složeným gramatickým tvarům. V některých případech je dokonce u slova, jež je součástí složeného gramatického tvaru, uveden jiný rys, než který konkrétní složený gramatický tvar vyjadřuje. Například u pomocného slovesa *jsem* je uveden přítomný čas, i když se podílí na složeném minulém čase *našel jsem* (viz Tabulka 2.1). V původních UD datech je tedy poměrně obtížné složené slovesné tvary vyhledat.

Jedním z cílů této práce je vymyslet pravidla a naprogramovat bloky pro hledání víceslovných slovesných tvarů ve slovanských jazycích. Pro různé slovesné časy/způsoby/infinitiv/přechodník jsou k dispozici bloky, které daný slovesný tvar objeví a označí. Jako vstupní data těchto bloků slouží data z Universal Dependencies (viz Tabulka 2.3). Tyto bloky označí nalezené složené gramatické tvary tak, že vypíšou příslušné atributy a jejich hodnoty do desátého sloupce dat, tzv. `MISC` sloupce, a sice k řídícímu členu celého složeného tvaru. Jedná se o následující atributy, které jsou detailněji popsány v kapitole 3.2.3.

- `Phrase`: seznam ordinál slov (tj. čísel reprezentujících pozici slova ve větě), která jsou součástí slovesného tvaru, tato čísla odpovídají hodnotě sloupce ID ve formátu CoNLL-U
- `PhraseAnimacy`: životnost slovesného tvaru
- `PhraseAspect`: vid slovesného tvaru
- `PhraseForm`: tvar slovesného tvaru
- `PhraseGender`: jmenný rod slovesného tvaru
- `PhraseMood`: způsob slovesného tvaru
- `PhraseNumber`: číslo slovesného tvaru
- `PhrasePerson`: osoba slovesného tvaru
- `PhrasePolarity`: informace, zda je slovesný tvar záporný či nikoli
- `PhraseReflex`: nabývá pouze hodnoty `Yes` a je přítomen pouze, pokud je slovesná tvar zvratný
- `PhraseTense`: čas slovesného tvaru
- `PhraseVoice`: slovesný rod slovesného tvaru

Po spuštění bloků na vstupních datech se do výstupního souboru označí všechny nalezené složené slovesné tvary a jejich morfologické rysy pomocí těchto atributů. Vyhledat složené slovesné tvary v takto oanotovaných datech je již triviální (viz Tabulka 2.2). Dokumentace, jak bloky spustit a jak stáhnout vstupní data, je k dispozici v kapitole 3.

ID	FORM	POS	FEATS
1	Několik	DET	Case=Acc|NumType=Card|PronType=Ind
2	jsem	AUX	…Polarity=Pos|**Tense=Pres**|VerbForm=Fin…
3	jich	PRON	Case=Gen|Number=Plur|Person=3|PronType=Prs
4	našel	VERB	…Polarity=Pos|Tense=Past|VerbForm=Part…
5	.	PUNCT	_

Tabulka 2.1 U slovesa *jsem* je zaznamenán přítomný čas i přes to, že se podílí na složeném minulém čase *našel jsem*. Pro přehlednost jsou některé sloupce a některé atributy ve `FEATS` sloupci vynechány.

ID	FORM	POS	MISC
1	Několik	DET	_
2	jsem	AUX	_
3	jich	PRON	LId=on-1
4	našel	VERB	Phrase=[2,4]…PhrasePerson=1|PhraseTense=Past…
5	.	PUNCT	_

Tabulka 2.2 Vyhledat složené slovesné tvary v bloky oanotovaných datech je již triviální. Pro přehlednost jsou některé sloupce a některé atributy v `MISC` sloupci vynechány.

2.1 Přítomný čas

Ve slovanských jazycích je přítomný čas tvořen pomocí osobních koncovek, které se přidávají ke kmeni slovesa (viz Tabulka 2.4)[9].

2.2 Budoucí čas

Na pomezí přítomného a budoucího času stojí vyjadřování budoucnosti pomocí dokonavých sloves. V UD datech se napříč různými slovanskými jazyky liší hodnoty atributu `Tense`. Zatímco např. v českých PUD datech je tato hodnota `Pres`, tedy přítomný čas, např. v ruštině je tato hodnota `Fut`, tedy budoucí čas. Rozhodnout, jestli je vhodnější značit čas těchto sloves jako přítomný, nebo budoucí je téměř nemožné, pro obě možnosti existují smysluplné důvody. Přítomný čas dává smysl z hlediska morfologie, tvar slovesa totiž odpovídá prézentní formě. Z hlediska sémantického nicméně slovesný tvar vyjadřuje budoucnost, je tedy zcela validní označit čas tohoto tvaru jako budoucí.

Budoucí čas nedokonavých sloves je ve slovanských jazycích tvořen pomocným slovesem a infinitivem či participiem plnovýznamového slovesa (viz Tabulka 2.5). Jako pomocné sloveso ve většině slovanských jazyků je využito sloveso být v budoucím čase, nicméně chorvatština a srbština používají pomocné sloveso *htjeti*, resp. *hteti* „chtít“ v čase přítomném. Bulharština a makedonština tvoří budoucí čas odlišně než většina slovanských jazyků, a sice pomocí pomocného slovesa *ще*, resp. *ќе* a plnovýznamového slovesa v přítomném čase[10]. Na tento způsob tvoření budoucího času je třeba dávat pozor při programování bloků hledajících přítomný čas. Nechceme za přítomný čas označit ty slovesné tvary v přítomném čase, na nichž visí nějaké toto pomocné sloveso.

Samostatnou kapitolou je budoucí čas ve staroslověnštině. Vyjadřování budoucnosti totiž ve staroslověnštině nebylo plně gramatikalizováno. Zatímco budoucí čas

dokonavých sloves se tvořil prézentním tvarem slovesa (tedy stejně jako např. dnes v češtině), budoucí čas nedokonavých sloves se tvořil spojením pomocného slovesa a infinitivu, jako pomocná slovesa byla využívána slovesa modální a fázová, méně často pak i sloveso být. Nicméně modální a fázová pomocná slovesa si zpravidla alespoň částečně ponechávala svůj význam i ve vyjadřování budoucnosti, a hranice mezi přítomným a budoucím časem proto byla velmi tenká. Ve staroslověnských datech z Universal Dependencies není v těchto případech rozlišeno, jestli se jedná o přítomný čas modálního či fázového sloveso a infinitiv, nebo jestli se jedná o pomocné sloveso a infinitiv vyjadřující budoucí čas. Například ve větě *ли бо единого вѣзненавидитъ а другаго възлюбитъ ли единого дрѣжитъ ся а о другыѣмь неродити начьнетъ.* „Buď totiž bude jednoho nenávidět a druhého milovat, nebo se bude toho jednoho držet a tím druhým pohrdne.“ není nikde

Jazyk	**Název treebanku**	**Počet vět**	**Počet tokenů**
běloruština	HSE	25 231	305 109
bulharština	BTB	11 138	156 149
čeština	CAC	24 709	493 306
	PDT	19 416	332 542
	FicTree	12 760	166 432
	CLTT	1 121	35 997
	PUD	1 000	18 565
	Poetry	297	6 273
hornolužická srbština	UFAL	646	11 196
chorvatština	SET	9 010	199 409
makedonština	MTB	155	1 360
polština	PDB	22 152	347 319
	LFG	17 246	130 967
	PUD	1 000	18 333
pomačtina	Philotis	2 250	34 348
ruština	Taiga	17 872	197 001
	Poetry	5 086	64 112
	SynTagRus	87 336	1 515 683
	GSD	5 030	97 994
	PUD	1 000	19 355
slovenština	SNK	10 604	106 043
slovinština	SSJ	13 435	267 097
	SST	6 104	76 341
srbština	SET	4 384	97 673
staroruština	RNC	3 528	95 551
	Ruthenian	3 523	96 803
	TOROT	26 496	246 850
	Birchbark	3 114	27 269
staroslověnština	PROIEL	22 628	198 843
ukrajinština	IU	7 092	122 701

Tabulka 2.3 Dostupné treebanky pro jednotlivé slovanské jazyky s jejich velikostmi.

Jazyk	Příklad přítomného času
běloruština	Я прачытаў кнігу.
bulharština	Четете книгата.
čeština	Čtu knihu.
chorvatština	Čitam knjigu.
makedonština	Читам книга.
polština	Czytam książkę.
ruština	Я читаю книгу.
slovenština	Čítam knihu.
slovinština	Bral sem knjigo.
srbština	Čitam knjigu.
ukrajinština	Читаю книгу.

Tabulka 2.4 Příklad přítomného času ve slovanských jazycích.

Jazyk	Příklad budoucího času
běloruština	Я буду чытаць кнігу.
bulharština	Ще чета книга.
čeština	Budu číst knihu.
chorvatština	Pročitat ću knjigu.
makedonština	Ќе ја прочитам книгата.
polština	Będę czytał książkę.
ruština	Я буду читать книгу.
slovenština	Budem čítať knihu.
slovinština	Bom prebral knjigo.
srbština	Pročitaću knjigu.
ukrajinština	Я буду читати книгу.

Tabulka 2.5 Příklad budoucího času ve slovanských jazycích.

v anotacích uvedeno, že se jedná o budoucí čas, a ze samotných dat to nelze nijak poznat. Rozhodla jsem se tedy všechny takové případy neoznačovat za budoucí čas, jelikož bych pak za budoucí čas chybně označila i tvary času přítomného. Proto mnou naprogramovaný blok, který hledá a označuje víceslovné i jednoslovné slovesné tvary vyjadřující budoucí čas, po spuštění na staroslověnských datech neodhalí budoucí čas tvořený pomocí modálního či fázového pomocného slovesa a infinitivu[11].

2.3 Minulý čas

Minulý čas je ve slovanských jazycích tvořen různě, mezi nejčastější způsob patří vyjádření pomocí pomocného slovesa a l-ového příčestí plnovýznamového slovesa (viz Tabulka 2.6), přičemž v západoslovanských jazycích dochází k vynechání pomocného slovesa u třetí osoby, zatímco v jihoslovanských k tomuto vynechávání nedochází. Takto je minulý čas tvořen například v češtině a slovenštině (kde k vynechávání pomocného slovesa u třetí osoby dochází: *Spala dlouho.*), dále ve

Jazyk	Příklad minulého času
běloruština	Я прачытаў кнігу.
bulharština	Аз прочетох книгата.
čeština	Četl jsem knihu.
chorvatština	Pročitao sam knjigu.
makedonština	Јас ја прочитав книгата.
polština	Ja czytałem książkę.
ruština	Я читал книгу.
slovenština	Čítal som knihu.
slovinština	Prebral sem knjigo.
srbština	Čitao sam knjigu.
ukrajinština	Я читав книгу.

Tabulka 2.6 Příklad minulého času ve slovanských jazycích.

slovinštině, srbštině či chorvatštině (kde k vynechání pomocného slovesa u třetí osoby nedochází: *Spala je dolgo.* „Spala dlouho.“). V ruštině, běloruštině a v některých dalších slovanských jazycích je minulý čas jednoduchý, jedná se o participiální část složeného minulého času výše zmíněných jazyků: *Она долго спала.* „Spala dlouho.“ . V některých slovanských jazycích (v bulharštině a v makedonštině) se objevují i aorist a imperfektum. Aorist i imperfektum jsou jednoduché minulé časy, které se používaly ve starých slovanských jazycích, dnes jsou již ve většině těchto jazyků zaniklé[10].

2.4 Pasivum

Trpný rod je ve slovanských jazycích tvořen pomocí pomocného slovesa být a participia plnovýznamového slovesa. Kromě toho ještě některé jazyky vyjadřují trpný rod pomocí zvratného zájmena se (např. v češtině *Knihy se prodávají.*)[9].

2.5 Imperativ

Imperativ neboli rozkazovací způsob je určitý tvar slovesa, který vyjadřuje osobu a číslo, ale nikoliv čas. Imperativ nelze vyjádřit v první osobě jednotného čísla a v osobě třetí, typicky ho také nelze utvořit od modálních sloves. Ve slovanských jazycích se imperativ tvoří pomocí koncovek přidávaných ke kmeni slovesa (viz Tabulka 2.7).

2.6 Kondicionál

Kondicionál neboli podmiňovací způsob se ve slovanských jazycích tvoří pomocí l-ového příčestí a slovesa být ve speciálním tvaru, a to jak podmiňovací způsob přítomný, tak i podmiňovací způsob minulý. Podmiňovací způsob minulý se tvoří obdobně jako podmiňovací způsob přítomný, jen zpravidla navíc obsahuje ještě l-ové příčestí pomocného slovesa být. V některých případech se může pomocné

Jazyk	Příklad imperativu
běloruština	Чытай кнігу!
bulharština	Прочетете книгата!
čeština	Čtěte knihu!
chorvatština	Pročitajte knjigu!
makedonština	Прочитајте ја книгата!
polština	Czytaj książkę!
ruština	Читайте книгу!
slovenština	Čítajte knihu!
slovinština	Preberite knjigo!
srbština	Pročitajte knjigu!
ukrajinština	Читайте книгу!

Tabulka 2.7 Příklad imperativu ve slovanských jazycích.

sloveso spojit s podřadicí spojkou: např. české *kdybychom* či polské *gdybyśmy* vzniklo spojením spojky *když*, resp. *gdyby* a pomocného slovesa *bychom*, resp. *śmy*.

Pomocné sloveso podmiňovacího způsobu v některých slovanských jazycích vyjadřuje osobu a číslo (např. v srbštině: *Oni bi otišli* „Oni by odešli“, *Mi bismo otišli* „My bychom odešli“), v jiných slovanských jazycích je používán jeden tvar pro všechny osoby i čísla (např. v ruštině: *Они бы ушли.* „Oni by odešli“, *Мы бы ушли* „My bychom odešli“). Ve slovenštině je podmiňovací způsob tvořen pomocným slovem *by*, dále přítomným časem pomocného slovesa *byť* „být“ a l-ovým příčestím plnovýznamového slovesa (např. *My by sme odišli* „My bychom odešli“)[9].

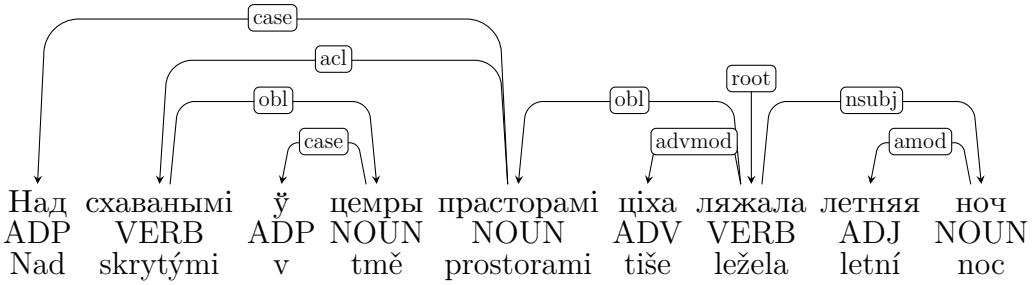
2.7 Přechodník

Přechodník je jeden z neurčitých tvarů slovesa, který vyjadřuje děj doprovázející nebo dolňující jiný děj. Rozlišuje se přechodník přítomný, jenž slouží k vyjádření současnosti dvou dějů (*Četla si knihu, pobrukujíc si chytlavou melodii.*), a přechodník minulý, který slouží k vyjádření předčasnosti jednoho děje před druhým (*Spatřiv na něho se ženoucí stádo krav, dal se na útěk.*). V některých slovanských jazycích přechodníky vyjadřují jmenný rod a číslo (např. v ruštině či češtině), v některých nikoliv (ve slovenštině nebo v polštině). Dnes jsou ve většině slovanských jazyků přechodníky považovány za archaické či knižní[10].

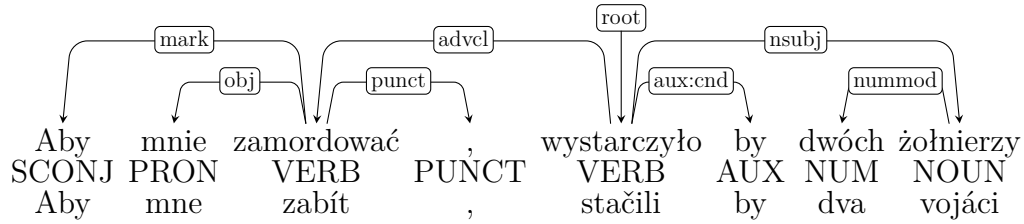
2.8 Infinitiv

Infinitiv je neurčitý slovesný tvar, tedy nevyjadřuje číslo, osobu, čas ani způsob. V některých slovanských jazycích je využíván například při tvoření budoucího času. V bulharštině a makedonštině se infinitiv nevyskytuje, vyjadřuje se opisně pomocí částice *да*, např. v makendonštině ve větě *Мора да работи.* „Musí pracovat.“ sloveso *работи* „pracuje“ vyjadřuje osobu i číslo[9].

V češtině infinitivy končící koncovkou *-t* mají ještě svou archaickou podobu končící koncovkou *-ti* (*mluvit* a *mluviti*). Ve slovinštině je jako infinitiv používán



Obrázek 2.1 V běloruských datech jsou některá adjektiva utvořená od sloves značená jako slovesa.



Obrázek 2.2 V polských datech je kondicionál značen pomocí podtypu závislosti `aux:cnd`.

jen tvar končící koncovkou *-ti* (*govoriti* „mluvit“), druhý tvar se nazývá supinum a je používán po pohybových slovesech[10].

2.9 Preprocessing dat

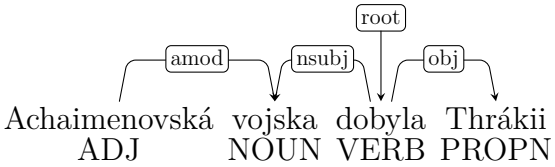
Anotace napříč různými treebanky se mírně odlišují, pro přehlednost a snadnější orientaci v datech dává smysl některé anotace sjednotit. V této kapitole jsou popsány případy, na které jsem narazila během programování jednotlivých bloků pro vyhledávání složených gramatických tvarů ve slovanských jazycích.

V běloruských datech jsou některá adjektiva utvořená od sloves značená jako slovesa (viz Obrázek 2.1). Zároveň je ale u těchto slov nastaven `feats` atribut `Case`. Blok `Preprocessor` přenastaví hodnotu atributu `upos` na `ADJ`. V jiných slovanských jazycích jsou tato slova značena jako `ADJ`.

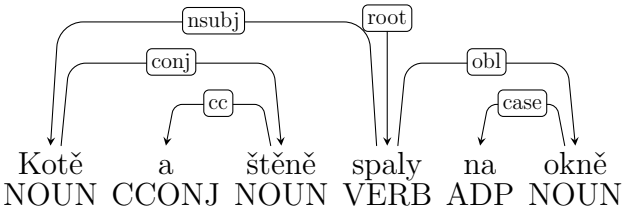
V polských datech je odlišně značen podmiňovací způsob. Zatímco v datech ostatních slovanských jazyků je podmiňovací způsob u pomocného slovesa označen ve `feats` atributu jako `Mood=Cnd`, v polských datech je podmiňovací způsob označen v `deprel` atributu jako `aux:cnd` (viz Obrázek 2.2). Blok `Preprocessor` přidá ke slovům s hodnotou `deprel` atributu `aux:cnd` ještě `feats` atribut `Mood=Cnd`.

Další atribut, který by bylo vhodné sjednotit, je `feats` atribut `Polarity`. Zatímco v datech některých jazyků (např. v ruských datech) je tento atribut vyznačen jen, pokud je jeho hodnota `Neg`, v datech jiných jazyků (např. v českých datech) tento atribut nabývá hodnot `Pos` i `Neg`. Blok `Preprocessor` tedy přenastaví veškeré výskyty `Polarity=Pos` na `Polarity=""`.

V českých a staroslověnských datech je u participií vyznačeno více jmenných rodů (např. `Gender=Fem,Neut`). Z jednoho izolovaného slova nelze rozpoznat,



Obrázek 2.3 Jmenný rod l-ového příčestí lze převzít z vyjádřeného podmětu.



Obrázek 2.4 U několikanásobného podmětu s jednotlivými členy v jednotném čísle nemusí být zcela jasné, který jmenný rod zvolit.

o jaký rod se jedná, protože jsou pro různé rody používány stejné koncovky. Konkrétně se tato problematika týká koncovky *-a*, která se uplatňuje v ženském rodě v jednotném čísle a středním rodě v množném čísle, a koncovky *-y*, která se uplatňuje v množném čísle v ženském rodě a mužském neživotném rodě. Tyto nejednoznačné hodnoty atributů lze snadno odstranit, pokud je ve větě vyjádřený podmět, stačí převzít jmenný rod z tohoto podmětu (viz Obrázek 2.3). Problém ovšem nastane u několikanásobného podmětu s jednotlivými členy v jednotném čísle. Například u typu *Kotě a štěně spaly na okně.* není zcela jasné, který jmenný rod by měl být zvolen (viz Obrázek 2.4). Takových vět je ale v datech velmi málo a `Preprocessor` je ignoruje. V případě, že je na participiu s více rody závislé nějaké vztažné zájmeno (který nebo jenž), lze jmenný rod převzít z těchto zájmen. Největší skupinu těchto dat ale tvoří věty s nevyjádřeným podmětem. Zde již je určení jmenného rodu obtížnější, a jelikož není cílem této práce odstranit víceznačné hodnoty atributu `Gender`, `Preprocessor` tato data nechává beze změny.

3 Uživatelská dokumentace

Tento program slouží k rozpoznání (složených i jednoduchých) slovesných tvarů. Skládá se ze dvou částí, a sice z několika menších bloků, jež jsou zaměřeny na slovanské jazyky, a z obecného bloku, který na základě informací, jež získá z konfiguračního souboru, označí víceslovné tvary v jazykových datech v CoNLL-U formátu.

3.1 Instalace Udapi

Pro práci s daty z UD je potřeba mít nainstalovaný framework `Udapi`. Návod, jak `Udapi` nainstalovat, je dostupný na url adrese `https://github.com/udapi/udapi-python`. Zde je v nabídce buď instalace uživatelské verze `Udapi`, nebo verze pro vývojáře, jež oproti uživatelské verzi umožňuje mít jednotlivé bloky uloženy v jiném adresáři než v tom, kde je nainstalované `Udapi`. Po instalaci `Udapi` lze používat příkaz `udapy`[12].

3.2 Bloky pro slovanské jazyky

První část tohoto programu se zaměřuje na hledání slovesných tvarů ve slovanských jazycích. Skládá se z několika menších programů napsaných v Pythonu, které se nazývají bloky. Každému slovesnému času a způsobu je věnován určitý blok. Přehled jednotlivých bloků je popsán níže.

- **Slavic_cond**: označení podmiňovacího způsobu
- **Slavic_future**: označení budoucího času
- **Slavic_imperative**: označení rozkazovacího způsobu
- **Slavic_inf**: označení infinitivních tvarů
- **Slavic_past**: označení minulého času
- **Slavic_pres**: označení přítomného času
- **Slavic_transgressive**: označení přechodníků

3.2.1 Získání vstupních dat

Program očekává data v CoNLL-U formátu. Jedná se o tabulkový formát dat, více informací o tomto formátu je popsáno v kapitole 1.2. Taková data lze stáhnout z Universal Dependicies na url adrese `https://universaldependencies.org/`, ze seznamu jazyků si lze vybrat libovolný (slovanský) jazyk, na němž bude program spuštěn. Po rozkliknutí řádky vybraného jazyka se objeví nabídka treebanků. Po rozkliknutí řádky vybraného treebanku je třeba rozkliknout odkaz na master repozitář. Tento GitHub repozitář je možné si naklonovat nebo stáhnout jako ZIP. Další možností, jak získat vstupní data, je stáhnout si je z repozitáře LINDAT/CLARIAH-CZ. Na url adrese `http://hdl.handle.net/11234/1-5502` jsou dostupné všechny jazyky UD jako jeden velký balík.

3.2.2 Spuštění programu

Spuštění všech bloků

Ke spuštění všech bloků slouží program `main.py`. Tento program přijímá následující parametry:

- `--help` — vytiskne popis přijímaných parametrů a následně program skončí
- `--lang` — jazyk, v němž jsou data, na kterých je program spuštěn
- `--input` — cesta k souboru se vstupními daty, defaultně je hodnota tohoto parametru nastavena na `stdin`, což značí čtení ze standardního vstupu
- `--output` — cesta k souboru, kam mají být uložena výstupní data, defaultně je hodnota tohoto parametru nastavena na `stdout`, což značí vypisování na standardní výstup
- `--conf` — cesta ke konfiguračnímu souboru, defaultně je hodnota tohoto parametru nastavena na `conf.json`
- `--module_prefix` — cesta k blokům, které budou spuštěny, jednotlivé adresáře jsou odděleny tečkami, defaultně je hodnota tohoto parametru nastavena na blocks.
- `--writer_prefix` — cesta k souboru `writer.py`, který je potřeba ke spuštění bloků, jednotlivé adresáře jsou odděleny tečkami, defaultně je hodnota tohoto parametru nastavena na prázdný řetězec

Konfigurační soubor, který je potřeba ke spuštění programu `main.py`, se nazývá `conf.json` a je uložen ve stejném adresáři jako bloky slovanských jazyků. Popis formátu tohoto konfiguračního souboru a toho, jak do něj přidávat další bloky a jazyky, je dostupný v kapitole 4.2.3.

Spuštění jednoho bloku

V situaci, kdy by nás zajímala jen jedna oblast slovesných tvarů, by stačil spustit jen jeden konkrétní blok, a bylo by tedy zbytečné pouštět bloky všechny. V takovém případě lze spustit jen jeden tento konkrétní blok pomocí příkazu `udapy` z příkazové řádky. To se provede následovně:

```
cat in.conllu | udapy -s .slavic_blocks.Block_name > out.conllu
```

Soubor `in.conllu` je soubor se vstupními daty, soubor `out.conllu` je soubor s výstupními daty. Parametr `-s` zajistí uložení dat. Spustí se blok s názvem `Block_name` uložený v adresáři s názvem `slavic_blocks`. Tečka před názvem adresáře zajistí, že `Udapi` před název bloku nepřidá `udapi/block/`. Každý slovanský blok potřebuje ke spuštění také soubor `writer.py`. Pokud nepouštíme program ze složky, kde se tento soubor nachází, je třeba přidat cestu k tomuto souboru, kde jsou jednotlivé adresáře odděleny tečkami.

```
cat in.conllu | udapy -s .slavic_blocks.Block_name
writer_prefix=slavic_languages.blocks > out.conllu
```

3.2.3 Výstup programu

Výstupem programu je CoNLL-U soubor s označenými slovesnými tvary a gramatickými informacemi o těchto tvarech. Tyto informace jsou zaznamenány v desátém sloupci dat (který se nazývá MISC a který slouží pro doplňkové informace), a sice u řídícího členu celého slovesného tvaru. Tedy u složených slovesných tvarů jsou tyto informace uloženy u plnovýznamového slovesa, u sponových tvarů jsou informace uloženy u jména.

Přehled vygenerovaných značek

- `Phrase`: seznam ordinál slov (tj. čísel reprezentujících pozici slova ve větě), která jsou součástí slovesného tvaru, tato čísla odpovídají hodnotě sloupce ID ve formátu CoNLL-U
- `PhraseAnimacy`: životnost slovesného tvaru
- `PhraseAspect`: vid slovesného tvaru
- `PhraseForm`: tvar slovesného tvaru
- `PhraseGender`: jmenný rod slovesného tvaru
- `PhraseMood`: způsob slovesného tvaru
- `PhraseNumber`: číslo slovesného tvaru
- `PhrasePerson`: osoba slovesného tvaru
- `PhrasePolarity`: informace, zda je slovesný tvar záporný či nikoli
- `PhraseReflex`: nabývá pouze hodnoty `Yes` a je přítomen pouze, pokud je slovesná tvar zvratný
- `PhraseTense`: čas slovesného tvaru
- `PhraseVoice`: slovesný rod slovesného tvaru

Jednotlivé rysy mohou nabývat stejných hodnot jako odpovídající morfologické rysy jednotlivých slov. Např. hodnoty atributu `PhraseGender` odpovídají hodnotám definovaným pro featuru `Gender`, jedná se o hodnoty `Com`, `Fem`, `Masc`, `Neut`. Přehled možných hodnot, kterých mohou jednotlivé atributu nabývat, je dostupný na stránkách UD, konkrétně na url adrese `https://universaldependencies.org/u/feat/`. Hodnoty jednotlivých featur, jež se mohou objevit v daném jazyce, jsou uloženy v souboru v JSON formátu na GitHubu Universal Dependencies, konkrétně na url adrese `https://github.com/UniversalDependencies/tools/blob/master/data/feats.json`.

Může se ale stát, že phrase featura nabyde hodnoty, která v odpovídající feature pro tento jazyk neexistuje. Např. ve slovinštině se nevyskytuje jako hodnota featury `Tense` hodnota `Past`. Slovinština tvoří minulý čas obdobně jako čeština, tedy pomocí pomocného slovesa *biti* „být“ a l-ového příčestí (na rozdíl od češtiny ale nevynechává pomocné sloveso ve třetí osobě). Ve slovinských datech ale u tohoto

příčestí není featura `Tense` vůbec uvedena. Po odhalení složeného minulého času bude do `MISC` sloupce vygenerován mimo jiné i atribut `PhraseTense=Past`.

Vygenerované značky se mohou objevit ve výstupním souboru, ale nutně se nemusí objevit všechny. U řídícího slova daného slovesného tvaru jsou zaznemanány jenom ty gramatické kategorie, které daný tvar vyjadřuje. Tedy například u slovesného tvaru *nebudu si číst* se objeví značky `Phrase`, `PhraseAspect`, `PhraseForm`, `PhraseMood`, `PhraseNumber`, `PhrasePerson`, `PhrasePolarity` a `PhraseReflex`, zbylé značky zde zmíněny nebudou.

3.3 Konfigurační soubor

Další částí tohoto programu je obecný blok, jenž na základě informací z konfiguračního souboru označí víceslovné (i jednoslovné) tvary ve vstupních datech v CoNLL-U formátu. Hlavním cílem tohoto obecného bloku a konfiguračního souboru je usnadnit uživatelům doplnění pravidel pro další složené tvary nebo další jazyky. Bloky pro slovanské jazyky, zmíněné v kapitole výše, se zaměřují na slovesné tvary, nicméně pomocí konfiguračního souboru je možné označit i jiné složené tvary (např. jmenné skupiny s předložkovou vazbou atd.). Uživatelé, kteří mají zkušenosti s programováním, si samozřejmě mohou sami v Pythonu doprogramovat další bloky pro `Udapi`, poměrně jednoduchý formát konfiguračního souboru ale umožňuje doplnit pravidla i uživatelům, kteří s programováním zkušenosti nemají. Kód jednotlivých bloků je také velmi podobný, díky konfiguračnímu souboru a obecnému bloku nemusí docházet k opakování kódu, který je již napsán.

3.3.1 Formát YAML

YAML Ain't Markup Language je formát určený k serializaci dat. Jedná se o uživatelsky přívětivý datový formát, který lze bez větších komplikací upravovat v klasickém textovém editoru. Struktura YAMLu je dána odsazeními.

Pro naše použití YAMLu je nezbytné znát reprezentace dat pomocí slovníků, seznamů a sklalárů (textových řetězců a čísel). Seznam je v YAMLu reprezentován následovně:

```
- 1
- 2
- 3
```

Před každou položkou jednotlivou položkou seznamu je napsána pomlčka a mezera, až poté následuje konkrétní hodnota. Další důležitou strukturou je slovník. Ten je v YAMLu reprezentován jako klíč, mezera, dvojtečka, hodnota, jak lze pozorovat z následujícího příkladu:

```
author: James Joyce
nationality: Irish
century: 20th
```

Lze tvořit i zanořené struktury pomocí odsazení dvěma mezerami:

```
author_1:
  name: James Joyce
  nationality: Irish
  century: 20th
  notable_works:
    - Ulysses
    - Dubliners
    - A Portrait of the Artist as a Young Man
```

Celá tato podkapitola vychází z oficiální dokumentace YAMLu, kde je možné získat detailnější informace o tomto formátu. Tato dokumentace je dostupná na url adrese `https://yaml.org/spec/1.2.2/`[13].

3.3.2 Formát konfiguračního souboru

Konfigurační soubor slouží k nastavení pravidel pro vyhledání složených gramatických tvarů v daném jazyce. Soubor je ve formátu YAML, který je popsán v kapitole výše.

Konfigurační soubor má následující strukturu:

```
pravidlo_1:
  slovo_1.1:
    nastavení_podmínek
  slovo_1.2:
    nastavení_podmínek
  phrase:
    nastavení_hodnot

pravidlo_2:
  slovo_2.1:
    nastavení_podmínek
  slovo_2.2:
    nastavení_podmínek
  slovo_2.3:
    nastavení_podmínek
  phrase:
    nastavení_hodnot
```

Tedy každé pravidlo obsahuje slova důležitá pro daný víceslovný tvar, každé takové slovo pak obsahuje podmínky, které toto slovo musí v daném víceslovném tvaru splňovat. V následujících kapitolách je popsáno, jak takové pravidlo vytvořit.

Konkrétní příklad konfiguračního souboru

V následujících podkapitolách si detailně vysvětlíme formát konfiguračního souboru, nyní si jen pro ilustraci ukážeme, jak takový konfigurační soubor může vypadat.

```
all:
  refl:
    feats[Reflex]: Yes
```

```
    oblig: facult

pres_simple:
  node:
    feats[Tense]: Pres
    feats[Aspect]: Imp
    feats[VerbForm]: Fin
  phrase:
    phraseTense: const Pres
    phrasePerson: node feats[Person]
    phraseNumber: node feats[Number]

pres_pass:
  node:
    feats[Voice]: Pass
  aux_pass:
    udeprel: aux
    feats[Tense]: Pres
    oblig: oblig
  aux_forb:
    udeprel: aux
    feats[Tense]: not Pres
    oblig: forb
  phrase:
    phraseTense: const Pres
    phrasePerson: aux_pass feats[Person]
    phraseNumber: aux_pass feats[Number]
```

Tento jednoduchý konfigurační soubor vyhledá a označí jednoduchý přítomný čas a trpný rod v přítomném čase v češtině. Obsahuje celkem tři pravidla, a sice pravidla s názvy `all`, `pres_simple` a `pres_pass`. Pravidlo `all` je speciální pravidlo, slova v něm definovaná se přidají do definic ostatních pravidel. V pravidle `all` je definováno slovo `refl`, toto slovo musí splňovat dvě podmínky: `feats[Reflex]=Yes` a druhá podmínka `oblig: facult` značí, že se dané slovo v nalezeném složeném slovním tvaru vyskytovat může, ale nemusí.

Pravidlo `pres_simple` vyhledává jednoduchý přítomný čas v češtině, tedy nedokonavá slovesa v přítomném čase. Jedná se o všechna slova, která splňují podmínky vypsané u slova `node`. Dále se sem doplní definice slova `refl` z pravidla `all`, tedy v dětech slov, která splňují pravidla definovaná ve slově `node`, se zkusí vyhledat zvratná zájmena. Ve slově `phrase` je definováno, jaké anotace se mají zapsat k nalezenému tvaru. Klíčové slovo `const` znamená, že jako hodnota atributu se zapíše slovo následující po tomto klíčovém slově. Místo slova `const` je ještě možné napsat název slova definovaného v daném pravidle, v tom případě se jako hodnota atributu zapíše hodnota vybraného atributu z tohoto slova. Tedy v případě `phrasePerson: node feats[Person]` se jako hodnota atributu `phrasePerson` nastaví hodnota atributu `feats[Person]` řídícího členu nalezeného tvaru.

Pravidlo `pres_pass` vyhledá trpný rod v přítomném čase. Toto pravidlo funguje velmi obdobně jako pravidlo `pres_simple`. Obsahuje navíc definice slov `aux_pass` (toto slovo má atribut `oblig` nastaven na `oblig`, tedy musí se v nalezeném složeném tvaru vyskytovat) a `aux_forb` (toto slovo má atribut `oblig` nastaven

na `forb`, tedy v nalezeném složeném tvaru se vyskytovat nesmí, čímž zakážeme složené tvary jako *byli jsme zachráněni*).

3.3.3 Nastavení pravidla pro hledání víceslovných tvarů

Každé pravidlo má následující strukturu:

```
pravidlo_1:
  slovo_1.1:
    nastavení_podmínek
  slovo_1.2:
    nastavení_podmínek
  phrase:
    nastavení_hodnot
```

Každé pravidlo je tedy tvaru `název_pravidla: tělo_pravidla`. Samotné tělo pravidla je popsáno v následujících kapitolách. Jednotlivé názvy pravidel mohou být zvoleny libovolně, jen je třeba dodržovat, aby byly názvy různé. A to i přes to, že samotný formát YAML nevyžaduje unikátní klíče. V případě, že by více pravidel mělo identický název, budou všechna tato pravidla s identickým názvem přepsána posledním definovaným pravidlem s tímto názvem. Speciálním případem je pravidlo s názvem `all`, které je popsáno níže.

Klíčové slovo all

Pravidlo s názvem `all` je určeno pro definování slov, která se objevují ve všech ostatních pravidlech. Jedná se například o zvratná zájmena nebo o částice pro vyjádření záporu. Abychom tato slova nemuseli definovat v těle každého pravidla znovu, nadefinujeme je jen jednou v těle pravidla s názvem `all`. Definice těchto slov se pak automaticky přidají do všech ostatních pravidel. Pokud nějaké pravidlo obsahuje definici slova se stejným názvem jako slovo definované v pravidle `all`, pak je původní definice slova z pravidla `all` přepsána definicí tohoto slova z tohoto pravidla.

Tělo pravidla

Tělo každého pravidla sestává ze slovníku, jehož klíče jsou tvořeny názvy slov důležitých pro hledaný víceslovný tvar a jehož hodnoty jsou podmínky, které tato slova musejí splňovat. Tyto podmínky jsou popsány v následujících kapitolách, nyní se zaměříme na názvy jednotlivých slov. Tyto názvy si uživatel opět může zvolit až na několik výjimek libovolně, jen je opět třeba dbát na to, aby byly tyto názvy v rámci jednoho pravidla unikátní. Zároveň nelze jako název použít slovo `const`, jedná se o klíčové slovo, jež je popsáno v kapitole 3.3.6.

Jedno ze slov, které nemůže mít libovolný název, je slovo `node`. Jedná se o hlavní slovo z celého víceslovného spojení. Právě k tomuto slovu budou do MISC sloupce připsány informace o celém nalezeném víceslovném spojení. Node ale nemusí být v těle pravidla definován. Pokud tato situace nastane, pak pro toto slovo neexistují žádné omezující podmínky mimo to, že se mezi na něm závislými slovy musejí vyskytovat ostaní definovaná slova v tomto pravidle kromě slova s názvem `phrase`.

`Phrase` je další slovo, jehož název nemůže být libovolný. Jedná se o reprezentaci informací, které se následně zapíšou do `MISC` sloupce k hlavnímu slovu celého víceslovného spojení. Slovo `phrase` musí být obsaženo v těle každého pravidla s výjimkou pravidla s názvem `all`.

3.3.4 Nastavení podmínek pro slovo z hledaného víceslovného tvaru

Reprezentace každého slova důležitého pro daný víceslovný tvar má následující strukturu:

```
název_slova:
  atribut_1: hodnota_1
  ...
  atribut_n: hodnota_n
```

Reprezentace slova `phrase` je popsána v kapitole 3.3.6. Tato reprezentace se totiž sémanticky mírně odlišuje od reprezentace ostatních slov, na kterou se zaměříme nyní.

Tělo každého slova sestává ze slovníku, jehož klíče jsou tvořeny názvy atributů a jehož hodnoty jsou tvořeny samotnými hodnotami atributů, jejichž struktura je popsána v následujících kapitolách. Nyní se zaměříme na názvy jednotlivých atributů.

Názvy atributů téměř zcela odpovídají názvům atributů Universal Dependencies. Jedná se o následující atributy:

- **form**: tvar slova ve větě
- **lemma**: základní (slovníkový) tvar slova
- **upos**: slovní druh (jednotná sada značek pro všechny jazyky)
- **xpos**: slovní druh (nepovinná značka ze sady specifické pro daný treebank)
- **feats**: seznam morfologických vlastností, název konkrétního atributu se v konfiguračním souboru nastaví jako feats[feature_value], tedy např. feats[Person]
- **deprel**: závislostní vztah
- **udeprel**: část závislostního vztahu
- **misc**: dodatečné informace o slovu, nastavení jednotlivých kategorií je obdobný jako u feats, tedy misc[misc_value], např. misc[SpaceAfter]
- **ord**: pořadí slova ve větě, přičemž prvnímu slovu ve větě je přiřazeno číslo 1
- **oblig**: tento atribut narozdíl od atributů uvedených výše není definován v UD a je popsán v následující podkapitole

Atribut oblig

Konfigurační soubor navíc definuje atribut s názvem `oblig`. Tento atribut nese informaci o tom, jestli je dané slovo ve vícelovném spojení obsaženo povinně (`oblig`), nepovinně (`facult`), nebo jestli dané slovo ve víceslovném spojení nesmí být obsaženo vůbec (`forb`). Tento atribut je jako jediný z atributů povinný, a musí ho tedy definovat všechna slova v pravidle, s výjimkou slov `node` a `phrase`.

3.3.5 Nastavení podmínky

Jednoduchá podmínka

Chceme-li nastavit podmínku, aby určitý atribut nabýval určité hodnoty, pak postačuje napsat název atributu, dvojtečku, mezeru a konkrétní hodnotu, již požadujeme. Je potřeba dodržet stejný formát, jak jsou atributy a jejich hodnoty reprezentovány v datech z Universal Dependencies. Tedy hodnoty atributu `upos` jsou psány velkými písmeny, hodnoty featur malými písmeny s velkým počátečním písmenem atd. Přehled veškerých atributů je popsán v kapitole 1.1.1. Tedy například pokud bychom chtěli, aby atribut `feat[Tense]` nabýval hodnoty `Past`, pak bychom tuto podmínku mohli napsat následovně:

```
feats[Tense]: Past
```

Negace

Konfigurační soubor umožňuje definovat negaci. Pokud si uživatel přeje, aby určitý atribut nenabýval nějaké hodnoty, stačí před tuto hodnotu napsat klíčové slovo `not`. Tedy například pokud bychom chtěli, aby atribut `feat[Tense]` nenabýval hodnoty `Past`, pak bychom tuto podmínku mohli napsat následovně:

```
feats[Tense]: not Past
```

Seznam

Někdy ovšem nestačí jen to, že daný atribut musí nebo nesmí nabývat jedné konkrétní hodnoty. V některých případech bychom mohli požadovat, aby daný atribut nabýval alespoň jedné z několika hodnot, nebo naopak aby nenabýval ani jedné z několika hodnot. Takového chování lze docílit použitím datové struktury seznam. Pokud požadujeme, aby daný atribut nabýval alespoň jedné z několika různých hodnot, jednoduše jako hodnotu atributu napíšeme seznam obsahující všechny povolené hodnnoty. Tedy například pokud požadujeme, aby atribut `upos` nabýval hodnot `VERB`, `ADJ` nebo `NOUN`, pak bychom tuto podmínku mohli napsat následovně:

```
upos:
  - VERB
  - ADJ
  - NOUN
```

Můžeme také požadovat, aby daný atribut nenabýval ani jedné z určitých hodnot. To lze vyjádřit obdobně opět pomocí seznamu, jednotlivé položky seznamu

musejí být ovšem znegované, tj. před každou položkou se musí vyskytovat klíčové slovo `not`. Tedy například pokud požadujeme, aby atribut `upos` nenabýval ani jedné z hodnot `VERB`, `ADJ` nebo `NOUN`, pak bychom tuto podmínku mohli napsat následovně:

```
upos:
  - not VERB
  - not ADJ
  - not NOUN
```

V případě, že seznam obsahuje jak negované, tak nenegované hodnoty, je podmínka vyhodnocena jako konjunkce jednotlivých dílčích podmínek obsažených v seznamu. Nicméně je třeba si uvědomit, že míchání negovaných podmínek s nenegovanými nedává moc smysl. Pokud jsou zapnuté automatické kontroly konfiguračního souboru, program na takovéto případy upozorní a doporučí uživateli, aby podmínky upravil tak, aby seznam obsahoval buď jen negované hodnoty, nebo jen nenegované hodnoty.

3.3.6 Nastavení phrase

Trochu odlišně než atributy výše se nastavují atributy nalezeného slovního spojení. Zde nenastavujeme podmínky, které musejí být splněny, nýbrž hodnoty, které se mají uložit jakožto informace o nalezeném víceslovném tvaru. Atributy lze pojmenovat libovolně, jen by se v rámci jednoho těla `phrase` neměly jmenovat zcela identicky. V takovém případě dojde k přepsání poslední hodnotou atributu s tímto názvem.

Podle toho, jakou hodnotu chceme nastavit konkrétnímu atributu, může nastat několik odlišných situací, jež jsou popsány v následujících podkapitolách.

Klíčové slovo const

V případě, že požadujeme, aby daný atribut nabýval vždy konstantní hodnoty nehledě na hodnoty atributů jednotlivých komponentů slovního spojení, se hodnota nastaví jako dvouslovné spojení, kde prvním slovem je klíčové slovo `const` a druhým slovem je pak samotná požadovná hodnota atributu. Tato hodnota musí být jednoslovná. V případě, že bychom chtěli nastavit víceslovnou hodnotu, je třeba např. mezery mezi jednotlivými slovy nahradit podtržítky. Tedy například pokud bychom chtěli, aby atribut `phraseTense` nabýval hodnoty `Fut`, pak bychom tento požadavek mohli napsat následovně:

```
phraseTense: const Fut
```

Přejímání hodnot atributů ze slov obsažených ve víceslovném spojení

Dále bychom mohli požadovat, aby nějaký atribut nabýval hodnoty převzaté z nějakého slova ze slovního spojení. Například bychom chtěli, aby číslo celého slovního spojení bylo totožné s číslem pomocného slovesa. V takovémto případě se hodnota příslušného atributu opět skládá ze dvou slov, přičemž prvním slovem je námi definovaný název slova, z něhož chceme hodnotu atributu převzít, a druhým slovem je pak název tohoto přejímaného atributu. Tedy například pokud

bychom chtěli, aby atribut `phraseNumber` nabýval stejné hodnoty jako atribut `feats[Number]` pomocného slovesa, jež je pojmenované jako `aux`, pak bychom tento požadavek mohli napsat následovně:

```
phraseNumber: aux feats[Number]
```

Reprezentace hodnoty atributu pomocí seznamu

Při přejímání hodnot atributů je ale potřeba dávat pozor na to, aby se slovo, z něhož tyto hodnoty přejímáme, vždy vyskytovalo v konkrétním slovním spojení. Může ale nastat situace, kdy bychom chtěli převzít hodnotu určitého atributu z konkrétního slova, ale jen v případě, že se toto slovo v konkrétním slovním spojení vyskytuje. Pokud by se toto slovo v konkrétním slovním spojení nevyskytovalo, mohli bychom chtít například nastavit nějakou konstantní hodnotu atributu nebo převzít hodnotu atributu ze slova jiného.

Taková situace nastává například u českého minulého času, který je tvořen pomocným slovesem být v přítomném čase a l-ovým příčestím plnovýznamového slovesa. U třetí osoby se ale pomocné sloveso vynechává, a minulý čas je tak tvořen pouze l-ovým příčestím plnovýznamového slovesa. V tomto případě bychom chtěli nastavit hodnotu atributu `phrasePerson` na `aux feats[Person]`, pokud se v nalezeném slovním spojení pomocné sloveso vyskytuje, jinak bychom chtěli nastavit hodnotu `phrasePerson` na `const 3`.

V takových případech lze jako hodnotu atributu nastavit seznam, jehož každá položka obsahuje nastavení hodnoty. Jako hodnota atributu bude nastavena první položka seznamu, která obsahuje slova přítomná v konkrétním nalezeném slovním spojení. Je tedy potřeba dbát na správné pořadí položek seznamu. Situaci s českým minulým časem popsanou v předchozím odstavci bychom tedy vyjádřili následovně:

```
phrasePerson:
  - aux feats[Person]
  - const 3
```

Jelikož hodnota `const 3` není závislá na žádném konkrétním slově slovního spojení, je až poslední položkou seznamu, protože může být nastavena vždy nehledě na to, jestli se v nalezeném slovním spojení pomocné sloveso vyskytuje, či nikoliv.

Může ale nastat i situace, kdy hodnota určitého atributu závisí na přítomnosti nějakého slova v konkrétním slovním spojení. Můžeme požadovat, aby atribut nabýval hodnoty, která se ale nevyskytuje u žádného slova přítomného ve slovním spojení. Příkladem takové situace může být zvratné pasivum v češtině. Například ve větě *Knihy se prodávají.* je víceslovný tvar *se prodávají* v trpném rodě, tato informace ale vyplývá z kontextu na základě přítomnosti zvratného *se*, označeného v UD datech jako `expl:pass`. U slovního tvaru *prodávají* je dokonce zaznamenáno, že je v činném rodě. Chtěli bychom tedy nastavit `phraseVoice` na `Pass`, pokud se ve slovním spojení vyskytuje zvratné se určující trpný rod, jinak bychom chtěli `phraseVoice` nastavit na `node feats[Voice]`. Toho lze opět dosáhnout pomocí seznamu, nýbrž v tomto případě může seznam obsahovat i trojslovné definice hodnoty. Prvním slovem je název slova, jehož přítomnost ve slovním spojení chceme otestovat, poté následují dvě slova splňující formát klasické dvojslovné definice hodnoty. Tedy buď klíčové slovo `const` následované konkrétní hodnotou,

nebo uživatelem definovaný název slova, z něhož se má převzít hodnota daného atributu, následované názvem tohoto atributu.

Situaci s českým zvratným pasivem popsanou v předchozím odstavci bychom tedy vyjádřili následovně (za předpokladu, že zvratné se je pojmenované jako refl_pass):

```
phraseVoice:
  - refl_pass const Pass
  - node feats[Voice]
```

3.3.7 Spuštění obecného bloku s konfiguračním souborem

Kontrola konfiguračního souboru

Než bude blok s daty z konfiguračního souboru spuštěn, lze požádat o různé kontroly konfiguračního souboru. Základní kontrola upozorňuje na špatný formát konfiguračního souboru — například na chybějící slovo `phrase` v těle pravidla, na pojmenování slova v těle pravidla jako `const` či na chybějící atribut `oblig` v definicích těch slov, která ho vyžadují. Dále je kontrolováno, zda jsou vhodně definovány hodnoty `phrase` atributů, které se následně vepisují do dat. Zde je mimo správného formátu hodnoty také kontrolováno, jestli definice hodnot obsahují jen slova popsaná v těle pravidla. Dále také jestli není požadováno, aby byla převzata hodnota atributu z nějakého slova, které je v těle pravidla definováno, nicméně má hodnotu atributu `oblig` nastavenou na `forb`. Z takového slova nelze přejímat hodnoty atributů, jelikož se v hledaném víceslovném spojení nemůže vyskytovat. Také je kontrolováno, jestli jednotlivé definice slov obsahují správné názvy atributů.

Lze ale požádat i o pokročilejší kontroly, a sice o kontroly hodnot atributů `feats`, `deprel` a `udeprel`. K této kontrole jsou třeba validační data, jež jsou uložena na GitHubu projektu Universal Dependencies. Tato data obsahují informace o tom, kterých hodnot mohou výše zmíněné atributy nabývat v jednotlivých jazycích. Jak tato data získat, je popsáno v následující podkapitole.

Pokud program při kontrole narazí na chybu, vypíše tuto skutečnost na standardní chybový výstup. Pokud se nejedná o chybu, kvůli níž nelze v kontrolách pokračovat (např. o chybějící validační soubor při kontrole hodnot `feats`, `deprel` a `udeprel` atributů), program v kontrolách pokračuje a každou další nalezenou chybu vypisuje na standardní chybový výstup. Chybová hláška má následující formát:

```
Error found in path: popis chyby.
```

`Path` označuje cestu k nalezené chybě, která sestává z názvu pravidla, v němž se chyba nachází, dále případně z názvu slova a atributu v závislosti na tom, v jakém zanoření se chyba vyskytuje.

Získání validačních dat

K tomu, aby mohla být provedena kontrola hodnot `feats`, `deprel` a `udeprel` atributů, je potřeba mít stažená validační data. Tato data jsou uložena na GitHubu projektu Universal Dependencies, který je dostupný na url adrese `https://`

github.com/UniversalDependencies/tools/tree/master. Tento repozitář si lze naklonovat nebo stáhnout jako ZIP. Toho lze docílit kliknutím na zelené tlačítko s nápisem Code, nacházející se v pravém horním rohu. Po kliknutí na toto tlačítko se objeví nabídka, kde lze zvolit, aby byl stažen ZIP archiv nebo lze zkopírovat odkaz pro klonování.

Soubory, jež jsou využívány pro kontroly, se nacházejí v adresáři s názvem data. Jedná se o soubor `deprels.json` pro kontrolu hodnot deprel a udeprel atributů a o soubor `feats.json` pro kontrolu feats atributů.

Spuštění bloku pomocí příkazu udapy

Obecný blok se spouští obdobně jako jednotlivé bloky zmíněné v kapitole 3.2.2, a sice pomocí příkazu `udapy` z příkazové řádky. Pokud máme stažené `Udapi`, můžeme používat příkaz `udapy`. Návod, jak `Udapi` nainstalovat, je popsán v kapitole 3.1. Program lze spustit následovně:

```
cat in.conllu | udapy -s .general_block.General_block conf=config.yaml > out.conllu
```

Program očekává parametr `conf`, kde dostane cestu ke konfiguračnímu souboru, z něhož použije pravidla pro hledání víceslovných spojení. Parametr `-s` zajistí, že se výstup programu uloží. Soubor `in.conllu` je soubor obsahující vstupní data a `out.conllu` je soubor s týmiž daty, jen navíc obsahuje informace o nalezených víceslovných spojeních. Tyto informace jsou uloženy v posledním sloupci dat vždy u hlavního slova celého souvětí. Kromě informací definovaných v konfiguračním souboru je ještě dodán atribut s názvem `Phrase`, jehož hodnotou je seznam obsahující hodnoty atributu `ord` každého slova, které se v nalezeném spojení objevuje. Pomocí tohoto atributu `Phrase` lze tedy určit jednotlivé komponenty daného víceslovného spojení.

Tento obecný blok lze také spustit s výše zmíněnými kontrolami. K tomuto účelu slouží následující parametry:

- `check` — pokud je hodnota tohoto parametru nastavena na `True`, provedou se před samotným spuštěním bloku kontroly, blok je následně spuštěn jen v případě, že v konfiguračním souboru nebyly nalezeny žádné chyby, defaultně nastavena na `False`
- `lang` — povinný parametr, pokud je vyžadována kontrola hodnot feats nebo deprel/udeprel atributů, zkratka jazyka, v němž jsou vstupní data
- `feats` — cesta k validačnímu souboru s hodnotami feats atributů, defaultně `feats.json`, povinný parametr, pokud se provádí kontrola hodnot feats atributů
- `deprels` — cesta k validačnímu souboru s hodnotami deprel atributů, defaultně `deprels.json`, povinný parametr, pokud se provádí kontrola deprel a udeprel atributů
- `check_feats` — pokud je hodnota tohoto parametru nastavena na `True` a zároveň je hodnota parametru `check` nastavena na `True`, provede se kontrola feats atributů, defaultně nastavena na `True`

- `check_deprels` — pokud je hodnota tohoto parametru nastavena na `True` a zároveň je hodnota parametru `check` nastavena na `True`, provede se kontrola deprel a udeprel atributů, defaultně nastavena na `True`

Tedy například pokud bychom chtěli spustit obecný blok s konfiguračním souborem `conf.yaml` na datech v českém jazyce a požadovali bychom kontroly hodnot feats atributů, ale nikoli hodnot deprel a udeprel atributů, spustili bychom blok následovně (předpokládejme, že validační data ke kontrole hodnot feats atributů jsou uložena v souboru `feats.json`):

```
cat in.conllu | udapy -s .general_block.General_block conf=config.yaml check=True
lang=cs feats=validation_data/feats.json check_deprels=False > out.conllu
```

3.4 Preprocessing dat

Anotace napříč různými treebanky se mírně odlišují, jak je popsáno v kapitole 2.9. `Preprocesor` je další blok, který ovšem nevyhledává složené slovesné tvary, nýbrž částečně sjednocuje rozdílnou anotaci tam, kde to dává smysl. Pokud chceme vstupní data před samotným spuštěním bloků pro slovanské jazyky či obecného bloku s konfiguračním souborem sjednotit, postačuje spustit blok `Preprocessor` obdobným způsobem, jako se spouští jednotlivé bloky slovanských jazyků, a sice:

```
cat in.conllu | udapy -s .slavic_blocks.Preprocessor > out.conllu
```

Soubor `in.conllu` je soubor se vstupními daty, soubor `out.conllu` je soubor s výstupními daty se sjednocenou anotací. Parametr `-s` zajistí uložení dat. Spustí se blok s názvem `Preprocessor` uložený v adresáři s názvem `slavic_blocks`. Tečka před názvem adresáře zajistí, že `Udapi` před název bloku nepřidá `udapi/block/`.

4 Programátorská dokumentace

4.1 Udapi

Tento program pracuje s daty z projektu Universal Dependencies. Jedná se o data oanotovaná jak na morfologické, tak i na syntaktické úrovni. Pro přístup k těmto anotacím program využívá rozhraní `Udapi`, jež je implementováno v Pythonu. Návod, jak Udapi nainstalovat, je popsán v kapitole 3.1[12].

V rámci tohoto programu je rozhraní `Udapi` využito pro zpracování jednotlivých tokenů v datech, pro přistupování k jednotlivým atributům těchto tokenů i pro zapisování samotných informací o nalezených slovesných tvarech do dat. Návod, jak s Udapi pracovat, je dostupný na url adrese `https://udapi.github.io/tutorial/`.

4.2 Bloky pro slovanské jazyky

Naprogramování jednotlivých bloků pro vyhledání a označení (víceslovných i jednoslovných) slovesných tvarů ve slovanských jazycích nebylo nijak programátorsky obtížné. Stěžejní část úkolů této práce tkvěla v tom zjistit, jak jednotlivé slovanské jazyky tvoří slovesné časy a způsoby, jaké jsou rozdíly mezi těmito slovesnými tvary a co mají naopak společného. Další důležitou součástí práce bylo zjistit, jak jsou tyto skutečnosti označeny v samotných datech z Universal Dependencies, jelikož i přes to, že zde existují anotační pravidla, různé treebanky se v anotaci mírně odlišují. Řešením těchto úkolů se věnuje kapitola 2. Návod, jak tento program správně spustit a kde získat data, je popsán v kapitole 3.

Program pro vyhledávání a označování (víceslovných i jednoslovných) slovesných tvarů sestává z několika menších programů napsaných v Pythonu, nazývaných bloky. Níže je uveden seznam těchto bloků:

- **Slavic_cond**: označení podmiňovacího způsobu
- **Slavic_future**: označení budoucího času
- **Slavic_imperative**: označení rozkazovacího způsobu
- **Slavic_inf**: označení infinitivních tvarů
- **Slavic_past**: označení minulého času
- **Slavic_pres**: označení přítomného času
- **Slavic_transgressive**: označení přechodníků

Zdrojový kód každého bloku je umístěn v souboru s příponou `.py`, který je pojmenován stejně jako daný blok, jen jeho název začíná malým písmenem. Každý blok je potomkem třídy `Block` z modulu `udapi.core.block`.

Jednotlivé bloky mají velmi podobnou strukturu kódu. Hlavní metodou každého bloku je metoda `process_node(self, node)`. Kód této metody je aplikován postupně na každý token ze vstupních dat. Tělo této metody sestává z několika

podmínek, jež jsou odvozeny z teoretické části této práce, tedy zohledňují tvoření jednotlivých slovesných časů a způsobů různých slovanských jazyků a také anotaci dat jednotlivých jazyků. Pomocí několika if větví je hledán samotný požadovaný slovesný tvar v činném rodě. Jsou zde kontrolovány podmínky vztahující se k atributům tohoto tokenu, tedy například jestli je daný slovní tvar v požadovaném čase či jestli vyjadřuje požadovaný slovesný způsob atd. Dále je kontrolováno, jestli se v dětech právě prohledáváného tokenu nacházejí všechna potřebná pomocná slovesa či další požadované tokeny a jestli se zde naopak nenacházejí nějaké tokeny, které jsou pro daný čas či způsob zakázané, a tedy značí, že se jedná o jiný slovesný čas či způsob než ten, který program momentálně hledá. Kromě vyhledávání slovesných tvarů v činném rodě jsou také hledány slovesné tvary v rodě trpném a spony, a sice podobným způsobem, který byl popsán pro hledání slovesných tvarů v činném rodě. Při vyhledávání trpného rodu je kontrolováno, jestli se mezi dětmi příčestí plnovýznamového slovesa s feats atributem `Voice=Pass` (které je v některých datech značeno jako sloveso, v jiných jako přídavné jméno) nacházejí všechna potřebná pomocná slovesa či jiné tokeny a nenacházejí se zde žádné tokeny zakázané. Při hledání spon je řídícím slovem jméno, mezi dětmi tohoto jména je hledána spona (tento rys je zaznamenán jako `udeprel=cop`) s požadovanými morfologickými kategoriemi, případně další pomocná slovesa či jiné tokeny.

4.2.1 Blok Preprocessor

Anotace napříč různými treebanky se mírně odlišují, jak je popsáno v kapitole 2.9. `Preprocesor` je další blok, který ovšem nevyhledává složené slovesné tvary, nýbrž částečně sjednocuje rozdílnou anotaci tam, kde to dává smysl. Jeho zdrojový kód je uložen v souboru `preprocessor.py`. Tento blok postupně prochází vstupní data a podle nastavených podmínek přepisuje anotaci. Přehled těchto podmínek je taktéž popsán v kapitole 2.9.

4.2.2 Třída Writer

Třída `Writer`, uložená v souboru `writer.py` obsahuje několik metod, které jsou využívány všemi výše zmíněnými bloky. Tato třída obsahuje metody pro zjišťování hodnot některých atributů celého slovesného tvaru, jako například jestli je tento slovesný tvar zvratný (metoda `get_is_reflex(self, node, refl)` či jaký je slovesný rod tohoto tvaru (metoda `get_voice(self, node, refl)`. Dále se tato třída stará o zapisování informací do `MISC` sloupce. To má na starosti metoda s názvem `write_node_info`. Tato metoda přijímá poměrně velké množství nepovinných parametrů, jejichž hodnota je defaultně nastavena na hodnotu `None`. Každý takový parametr odpovídá jedné z `Phrase` značek, jejichž přehled je popsán v kapitole 3.2.3. Díky takovému způsobu zapisování do `MISC` sloupce se v názvech vygenerovaných značek neobjevují překlepy a je jednodušší tyto názvy měnit a přidávat.

4.2.3 Konfigurační soubor a main.py

Pro spuštění všech bloků na vstupní data slouží program `main.py`, který mimo jiné požaduje konfigurační soubor, v němž je nastaveno, jaké bloky se mají v jakém

pořadí spustit pro daný jazyk. Tento konfigurační soubor je ve formátu JSON a obsahuje slovník. Klíče tohoto slovníku jsou ve formě zkratky jazyka a hodnoty jsou seznamu názvů všech bloků v tom pořadí, ve kterém budou následně na datech spuštěny. Je důležité, aby se bloky jmenovaly totožně (až na velikost písmen) jako soubory, v nichž jsou uloženy jejich zdrojové kódy. Očekává se, že název bloku začíná velkým písmenem, poté následují jen písmena malá a jednotlivá slova v tomto názvu jsou oddělena znakem „_“, název souboru se zdrojovým kódem je stejný jako název bloku jen s jedním rozdílem, a sice že začíná malým písmenem. Tedy například konfigurační soubor obsahující data pro český a slovenský jazyk by vypadal následovně:

```
{
    "cs": [
        "Slavic_cond",
        "Slavic_future",
        "Slavic_imperative",
        "Slavic_inf",
        "Slavic_past",
        "Slavic_pres",
        "Slavic_transgressive"
    ],
    "sk": [
        "Slavic_cond",
        "Slavic_future",
        "Slavic_imperative",
        "Slavic_inf",
        "Slavic_past",
        "Slavic_pres",
        "Slavic_transgressive"
    ]
}
```

Program `main.py` přijímá následující parametry:

- `--help` — vytiskne popis přijímaných parametrů a následně program skončí
- `--lang` — jazyk, v němž jsou data, na kterých je program spuštěn
- `--input` — cesta k souboru se vstupními daty, defaultně je hodnota tohoto parametru nastavena na `stdin`, což značí čtení ze standardního vstupu
- `--output` — cesta k souboru, kam mají být uložena výstupní data, defaultně je hodnota tohoto parametru nastavena na `stdout`, což značí vypisování na standardní výstup
- `--conf` — cesta ke konfiguračnímu souboru, defaultně je hodnota tohoto parametru nastavena na `conf.json`
- `--module_prefix` — cesta k blokům, které budou spuštěny, jednotlivé adresáře jsou odděleny tečkami, defaultně je hodnota tohoto parametru nastavena na blocks.

- `--writer_prefix` — cesta k souboru `writer.py`, který je potřeba ke spuštění bloků, jednotlivé adresáře jsou odděleny tečkami, defaultně je hodnota tohoto parametru nastavena na prázdný řetězec

Tento program načítá dynamicky moduly podle toho, jaké bloky je potřeba spustit pro daný jazyk. Tuto informaci se program dozví až za běhu z konfiguračního souboru. K těmto dynamickéým importům program využívá knihovnu `importlib`. Pomocí metody `import_module(name, package=None)` je naimportován modul s názvem `name`[14]. Po importu prvního bloku je tento blok spuštěn na vstupních datech. Anotace, které blok doplnil, jsou uloženy do výstupního souboru. Další bloky jsou spouštěny na výstupních datech toho bloku, který byl spuštěn před nimi. Proto je třeba pečlivě zvolit pořadí, v němž budou jednotlivé bloky spuštěny, mohlo by totiž dojít k přepsání dat jednoho bloku blokem jiným.

4.3 Obecný blok

Dalším cílem této práce bylo navrhnout obecná pravidla pro odhalení složených gramatických tvarů a napsat program, který načte pravidla z konfiguračního souboru a nalezne a označí požadované složené gramatické tvary ve vstupních datech v CoNLL-U formátu. V této kapitole se budeme věnovat tomu, jak tento obecný blok funguje. Jak tento program spustit a jak napsat příslušný konfigurační soubor, je popsáno v kapitole 3.3.

4.3.1 Třída General_block

Třída `General_block` dědí od třídy `Block` z modulu `udapi.core.block`. Konstruktor třídy `General_block` příjímá kromě parametru `self` ještě následující parametry:

- `check` — pokud je hodnota tohoto parametru nastavena na `True`, provedou se před samotným spuštěním bloku kontroly, blok je následně spuštěn jen v případě, že v konfiguračním souboru nebyly nalezeny žádné chyby, defaultně nastavena na `False`
- `lang` — povinný parametr, pokud je vyžadována kontrola hodnot feats nebo deprel/udeprel atributů, zkratka jazyka, v němž jsou vstupní data
- `feats` — cesta k validačnímu souboru s hodnotami feats atributů, defaultně feats.json, povinný parametr, pokud se provádí kontrola hodnot feats atributů
- `deprels` — cesta k validačnímu souboru s hodnotami deprel atributů, defaultně deprels.json, povinný parametr, pokud se provádí kontrola deprel a udeprel atributů
- `check_feats` — pokud je hodnota tohoto parametru nastavena na `True` a zároveň je hodnota parametru `check` nastavena na `True`, provede se kontrola feats atributů, defaultně nastavena na `True`

- `check_deprels` — pokud je hodnota tohoto parametru nastavena na `True` a zároveň je hodnota parametru `check` nastavena na `True`, provede se kontrola deprel a udeprel atributů, defaultně nastavena na `True`

Data z konfiguračního souboru jsou načtena pomocí knihovny `PyYAML` a jsou uložena do slovníku (do proměnné `self.config`). V konstruktoru je také volána metoda `retype_dict(self, dictionary, path)`, která zajistí, že se všechny hodnoty v načteném slovníku přetypují na `string`. Pro jednodušší nastavování konfiguračního souboru jsou totiž povoleny i hodnoty např. typu `int`, ale i libovolných jiných skalárních typů. To usnadní uživateli, který nemusí mít vůbec žádné zkušenosti s programováním, např. zapisování hodnot atributu `feats[Person]`, kdy není potřeba, aby kolem hodnoty tohoto atributu psal uvozovky.

Hlavní metodou této třídy je opět metoda `process_node(self, node)`. Na rozdíl od bloků pro slovanské jazyky nejsou v těle této metody sepsány podmínky pro hledaný složený gramatický tvar, tyto podmínky jsou načítány z konfiguračního souboru. Ve chvíli, kdy se nalezne složený gramatický tvar, který splňuje všechny podmínky pravidla z konfiguračního souboru, se informace o tomto složeném gramatickém tvaru zapíšou do dat a další pravidla už pro daný token nejsou zpracovávána. Tedy ve výstupních datech budou označeny složené gramatické tvary informacemi prvního pravidla z konfiguračního souboru, jehož podmínky daný token splňuje.

Kontroly jednotlivých podmínek z konfiguračního souboru jsou prováděny pomocí metody `get_attrs(self, attrs)`, která je definována ve třídě `Node` v modulu `udapi.core.node`. Tato metoda vrací hodnoty atributů, jejichž seznam dostane jako vstupní parametr. Hodnoty, jež tato metoda vrátí, jsou následně porovnávány s hodnotami z konfiguračního souboru, přičemž je brána v potaz i negace nebo nastavní hodnot v konfiguračním souboru pomocí seznamu. Pokud program při vyhodnocování podmínek narazí na chybu v konfiguračním souboru (např. na chybějící definici slova `phrase`, na neexistující název atributu či na jakoukoli jinou chybu), na standardní chybový výstup vypíše chybovou hlášku s popisem této chyby a skončí.

4.3.2 Třída Checker

Pro kontroly konfiguračního souboru slouží třída `Checker`. Pokud je obecný blok spuštěn s parametrem `check=True`, pak je vytvořena instance této třídy a před samotným spuštěním bloku na vstupních datech se provedou kontroly konfiguračního souboru.

Hlavní metodou této třídy je metoda `check_config_file(self, feats, deprels, lang, check_feats, check_deprels)`. Tato metoda prochází celý slovník načtený z konfiguračního souboru. Postupně kontroluje, jestli všechna pravidla (kromě pravidla s názvem `all`) obsahují definici slova `phrase`, jestli není název některého definovaného slova v těle jakéhokoli pravidla `const`, jestli definice slov (mimo slova `phrase` a `node`) obsahují povinný atribut `oblig` a jestli je hodnota tohoto atributu `oblig`, `facult` nebo `forb`. Dále je kontrolováno, jestli se v definicích neobjevují názvy atributů, které v datech z Universal Dependencies neexistují, jestli má atribut `feats[feat_value]` správný formát (tedy jestli začíná slovem `feat` a dále obsahuje nějaký textový řetězec v hranatých závorkách), jsou také kontrolovány hodnoty atributu `upos`. Jelikož se názvy těchto

hodnot nemění, jsou uloženy v seznamu ve statické proměnné s názvem `UPOS` přímo ve třídě `Checker`. Také je kontrolován formát hodnot atributů v definicích slov `phrase` – zda se tato hodnota skládá ze dvou nebo tří slov, zda se odkazuje k definovaným slovům v rámci pravidla, zda se neodkazuje k zakázaným slovům či zda se nepřejímá z nějakého slova neexistující atribut.

Pokud je nastaven parametr `check_feats=True`, pak jsou kontrolovány i hodnoty feats atributů, pokud je nastaven parametr `check_deprels=True`, pak jsou kontrolovány i hodnoty deprel a udeprel atributů. K těmto kontrololám jsou třeba validační data, v nichž je uvedeno, jaké hodnoty těchto atributů se mohou objevit v jakých jazycích. Jak získat tato data je popsáno v kapitole 3.2.3. Tato data jsou uložena ve formátu JSON, k jejich načtení program používá vestavěný balíček `json`.

Pokud se při kontrolách narazí na chybu v konfiguračním souboru, je tato chyba i s popisem vypsána na standardní chybový výstup. Pokud se nejedná o chybu, kvůli níž nelze v kontrolách pokračovat, program neskončí a v kontrolách pokračuje. Obecný blok se po dokončení kontrol spustí jen v tom případě, že v konfiguračním souboru nebyly nalezeny žádné chyby, v opačném případě program skončí a blok spuštěn není.

4.3.3 Konfigurační soubor

V této kapitole se budeme věnovat samotnému návrhu konfiguračního souboru a jeho nevýhodám. To, jak konfigurační soubor správně napsat, a popis jeho formátu se nachází v kapitole 3.3.

Tento konfigurační soubor je navržen tak, aby ho zvládl pochopit i nastavit uživatel, který nemá s programováním žádné zkušenosti. Proto je formát konfiguračního souboru co nejjednodušší a nejpřímočařejší. To s sebou ale nese i jistá omezení.

Jedním takovým významnějším omezením je nastavování hodnot nalezeného složeného gramatického tvaru. Lze nastavit jakoukoli konstantní hodnotu, lze přejímat hodnoty atributů z jiných komponentů složeného tvaru a lze nastavovat různé konstantní hodnoty a přejímat hodnoty atributů s ohledem na to, zda se ve složeném tvaru nachází určitý komponent. Co ale nelze, je nastavovat konstantní hodnoty či přejímat hodnoty atributů podle toho, jestli nějaký atribut nějakého slova, které je součástí složeného tvaru, nabývá nebo nenabývá konkrétní hodnoty. Přesněji, neexistuje pro to vyjádření v rámci definice slova `phrase`, lze toho ale dosáhnout jinak. A sice rozdělením pravidla. Lze definovat více různých pravidel, v jejichž tělech budou nastaveny hodnoty tohoto atritubutu a každé toto pravidlo bude mít mírně odlišné nastovávání hodnot atributů `phrase`. Tento způsob ale není ideální hlavně z toho důvodu, že vede ke kopírování pravidel, přičemž v tělech těchto pravidel dochází jen k minimálním změnám. Konfigurační soubor se tak stává nepřehlednějším a složitějším pro pozdější úpravy.

Podobným nedostatkem je nemožnost nastavit složenou podmínku, kdy by měla hodonota nějakého atributu nabývat určité hodnoty nebo hodnota zcela jiného atributu nabývat jiné určité hodnoty. Čili nelze nastavovat podmínky různých atributů spojené pomocí logické spojky `or`. Pomocí seznamů lze nastavovat takové podmínky v rámci téhož atributu, jinak ale `or` použít nelze. Tento problém má stejné neelegantní řešení jako problém zmíněný výše.

Důvodem, proč je konfigurační soubor navržen takto i s ohledem na zřejmé nedostatky, je snaha o co nejjednodušší formát. Pokud by konfigurační soubor řešil výše zmíněné problémy nějak chytřeji, pomocí dalších klíčových slov či složitějších konstrukcí, pak by bylo namísto studia formátu konfiguračního souboru jednodušší napsat si několik bloků v Pythonu, a to možná dokonce i přes to, že se uživatel do té doby s programováním vůbec nesetkal. S využitím rozhraní `Udapi` totiž není programování bloků natolik složité.

4.4 Nasazení obecného bloku na nový jazyk

Ukázalo se, že nasazení obecného bloku a konfiguračního souboru na nový jazyk není obtížné, pokud tomuto novému jazyku rozumíme či alespoň známe základní gramatická pravidla potřebná k tvoření složených časů nebo způsobů. V takovém případě totiž postačuje seznámit se s anotacemi vstupních dat a tato pravidla ve správném formátu zapsat do konfiguračního souboru, což jsem si vyzkoušela na češtině.

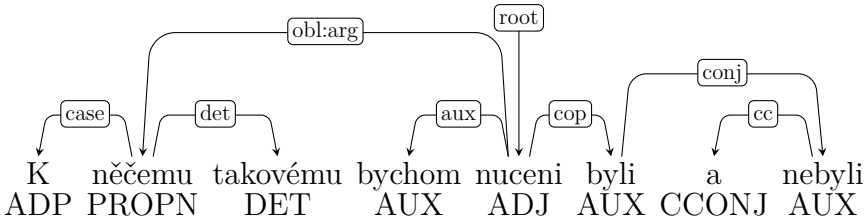
Výhodou dat z Universal Dependencies je mezijazyková konzistentní anotace. Pokud již máme konfigurační soubor pro jeden jazyk, je možné tento konfigurační soubor použít i pro jazyky jiné. V případě blízce příbuzných jazyků není ani třeba konfigurační soubor nijak výrazně upravovat, pokud jsme k definování pravidel nepoužili nějaké specifické rysy původního jazyka, jako jsou například lemmata pomocných sloves. Tedy nasazení obecného bloku a konfiguračního na nový, blízce příbuzný jazyk taktéž obtížné není. Například pokud již máme konfigurační soubor s pravidly pro češtinu, na slovenštině tento blok funguje taktéž.

Pokud ale tento nový jazyk vůbec neznáme a netušíme, jak zde fungují slovesné časy a způsoby, není tento úkol takto jednoduchý. K odhalení a označení složených tvarů je totiž potřeba správně definovat pravidla daného jazyka. Do takové situace jsem se dostala se staroslověnštinou či bulharštinou. Tyto jazyky nepoužívají latinku, což stěžuje orientaci v korpusech těchto jazyků, a i když se jedná o slovanské jazyky, některá gramatická pravidla se od češtiny poměrně odlišují. Tedy k nasazení obecného bloku a konfiguračního souboru na neznámý nový jazyk je potřeba nejprve nastudovat gramatická pravidla pro tvoření složených slovesných časů a způsobů, dále zorientovat se v anotaci vstupních dat a nakonec tato pravidla sepsat do konfiguračního souboru.

4.5 Úspěšnost programu

Úspěšnost obecného bloku závisí především na správné definici pravidel v konfiguračním souboru. Obecný blok totiž označí nalezený složený tvar těmi anotacemi, jež jsou popsány v prvním pravidle konfiguračního souboru, které tento složený tvar splňuje. Při psaní pravidel je tedy potřeba definovat nejprve ta nejspecifičtější pravidla a až poté ta obecnější. Nicméně při větším množství pravidel toto může být velmi obtížné a uživatel si nemusí všimnout, že nějaký složený tvar splňuje i pravidlo definované dříve než to pravidlo, které bylo pro tento složený tvar zamýšlené. Snadno tak může dojít k označení složeného tvaru nevhodnými anotacemi.

Dalším faktorem, který ovlivňuje úspěšnost, je anotace vstupních dat. Někdy



Obrázek 4.1 Ve větě *K něčemu takovému bychom nuceni byli a nebyli.* program odhalí pouze složený tvar *bychom nuceni byli*.

může být konfigurační soubor napsán bezchybně, ale i přes to zůstanou nějaké složené slovesné tvary neobjeveny. V mém případě se nepodařilo odhalit některé tvary budoucího a přítomného času v češtině, jelikož u některých sloves nejsou uvedeny vidy, na které se ale jednotlivá pravidla odkazují. V datech hornolužické srbštiny nejsou vidy uvedeny vůbec, tedy úspěšnost nalezení tvarů vyjadřujících budoucí čas dokonavých sloves je nulová. Ve staroslověnských datech není zaznamenán rozdíl mezi přítomným časem modálního či fázového slovesa a infinitivu a pomocným modálním nebo fázovým slovesem a infinitivu vyjadřujícím budoucí čas. Jelikož má znalost staroslověnštiny není dostačující k tomu, abych dokázala vymyslet nějaké pravidlo, které by rozhodlo, o jaký čas se v případě jakého složeného tvaru jedná, rozhodla jsem se všude ponechat přítomný čas. Tedy úspěšnost programu na staroslověnských datech při označování budoucího času je tatktéž velmi nízká. Zároveň nejsem schopna určit, jak moc velkých nepřesností jsem se v tomto případě dopustila, jelikož neumím poznat, v kolika případech jsem tento budoucí čas neodhalila a opravdu se jednalo o budoucí čas a v kolika případech se jednalo o správně označený čas přítomný.

Bohužel se mi z časových důvodů nepodařilo ve vyhledávání složených gramatických tvarů pokrýt případy s koordinačně spojenými pomocnými slovesy (tj. věty typu *Není a nikdy nebyl dobrým podnikatelem.*). Ačkoli zastoupení takových vět v jednotlivých korpusech je v řádu jednotek, jedná se o poměrně nepříjemný nedostatek. Tedy například ve větě *K něčemu takovému bychom nuceni byli a nebyli.* program odhalí pouze složený tvar *bychom nuceni byli* (viz Obrázek 4.1). To je z toho důvodu, že hledá pomocná slovesa pouze v dětech řídícího členu složeného gramatického tvaru, nicméně v tomto případě se nenalezené slovo *nebyli* nachází v dětech pomocného slovesa *byli*. I kdyby ale program toto pomocné sloveso odhalil, bylo by ještě třeba vyřešit problém, jak zapsat anotace o složeném tvaru. Vyskytují se zde totiž složené tvary dva, a sice *bychom nuceni byli* a *bychom nuceni nebyli*.

Závěr

Prvním cílem práce bylo najít a označit složené gramatické tvary v oanotovaných datech z projektu Universal Dependencies pro slovanské jazyky. Prvním krokem ke splnění tohoto cíle bylo nastudovat, jak jednotlivé slovanské jazyky tvoří různé časy a způsoby a jak jsou tyto skutečnosti označeny v UD datech. Ukázalo se, že ačkoli se projekt Universal Dependencicies zaměřuje na mezijazykově konzistentní anotaci treebanků, stále se zde nacházejí případy, kdy anotace zcela konzistentní není, i když by po jednoduchých úpravách být konzistentní mohla (jako například u značení podmiňovacího způsobu, kdy v polských datech není tento morfologický rys uveden ve `FEATS` sloupci u pomocného slovesa). Dalším krokem pak bylo v jazyce Python naprogramovat bloky pomocí rozhraní `Udapi`, které pomocí těchto nastudovaných pravidel v datech označí nalezené složené slovesné tvary, a sice tak, že do `MISC` sloupce k řídícímu členu složeného tvaru připíšou další anotace.

Druhým cílem této práce bylo navrhnout obecná pravidla pro odhalení složených gramatických tvarů. To obnášelo navrhnout formát konfiguračního souboru, v němž lze popsat pravidla pro jazyk zvolený uživatelem, a naprogramovat obecný blok, který na základě těchto dat nalezne v daném jazyce požadované složené gramatické tvary. Formát konfiguračního souboru je poměrně jednoduchý, tedy by ho bez větších problémů měl zvládnout vytvořit i uživatel, který nemá žádné zkušenosti s programováním. To s sebou ale nese i nevýhody, konfigurační soubor není schopen elegantně pokrýt všechna možná pravidla a v některých případech je nezbytné některá pravidla rozkopírovat a jen minimálně změnit, což vede k nepřehlednosti a špatné udržitelnosti celého konfiguračního souboru. Nicméně hlavním cílem návrhu konfiguračního souboru bylo, aby byl jednoduchý a dostupný i pro programátorsky nezkušené uživatele, což tento návrh splňuje.

Ukázalo se, že nasazení obecného bloku společně s konfiguračním souborem na nový jazyk není obtížné, pokud je tento jazyk blízce příbuzný jazyku, pro nějž již máme konfigurační soubor vytvořený, nebo pokud jsou nám gramatická pravidla tohoto jazyka známa. Pokud bychom ovšem chtěli nasadit obecný blok s konfiguračním souborem na jazyk, který je nám zcela neznámý, není tento proces zcela snadný. Nejprve je totiž potřeba nastudovat tato pravidla, což může být u některých jazyků poměrně složitý úkol.

Literatura

1. Čermák, František. *Nový encyklopedický slovník češtiny, Korpus* [online]. [cit. 2024-06-23]. Dostupné z: https://www.czechency.org/slovnik/KORPUS.
2. *Český národní korpus* [online]. [cit. 2024-06-23]. Dostupné z: https://wiki.korpus.cz/doku.php/pojmy:korpus.
3. Petkevič, Vladimír. *Nový encyklopedický slovník češtiny, Treebank* [online]. [cit. 2024-06-23]. Dostupné z: https://www.czechency.org/slovnik/TREEBANK.
4. *Universal Dependencies, úvod* [online]. [cit. 2024-06-23]. Dostupné z: https://universaldependencies.org/introduction.html.
5. *Universal Dependencies, Morfologie* [online]. [cit. 2024-07-17]. Dostupné z: https://universaldependencies.org/u/overview/morphology.html.
6. *Universal Dependencies, Syntax* [online]. [cit. 2024-07-17]. Dostupné z: https://universaldependencies.org/u/overview/syntax.html.
7. Biskup, Petr. *Nový encyklopedický slovník češtiny, Předložka* [online]. [cit. 2024-07-16]. Dostupné z: https://www.czechency.org/slovnik/P%C5%98EDLO%C5%BDKA.
8. *Universal Dependencies, CoNLL-U formát* [online]. [cit. 2024-06-23]. Dostupné z: https://universaldependencies.org/format.html.
9. Corbett, Greville; Comrie, Bernard. *The Slavonic Languages*. Routledge, 1993. 1st Edition. ISBN 9780415047555.
10. Zeman, Daniel. Universal Annotation of Slavic Verb Forms. *The Prague Bulletin of Mathematical Linguistics*. 2016, roč. 105, s. 143–193. Dostupné také z: https://ufal.mff.cuni.cz/pbml/105/art-zeman.pdf.
11. Fidlerová, Alena Andrlová; Kučera, Karel; Rejzek, Jiří. *Základy staroslověnštiny*. Filozofická fakulta Univerzity Karlovy, 2004. ISBN 80-7308-078-8.
12. Popel, Martin; Žabokrtský, Zdeněk; Vojtek, Martin. Udapi: Universal API for Universal Dependencies. In: Marneffe, Marie-Catherine de; Nivre, Joakim; Schuster, Sebastian (ed.). *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*. Gothenburg, Sweden: Association for Computational Linguistics, 2017, s. 96–101. Dostupné také z: https://aclanthology.org/W17-0412.
13. *YAML Ain't Markup Language version 1.2* [online]. [cit. 2024-06-18]. Dostupné z: https://yaml.org/spec/1.2.2/.
14. *importlib documentation* [online]. [cit. 2024-06-27]. Dostupné z: https://docs.python.org/3/library/importlib.html.

Seznam obrázků

Seznam tabulek

A Přílohy

A.1 Zdrojový kód

Elektronické příloha s názvem *periphrastic_grammatical_forms.zip* obsahuje zdrojové kódy.