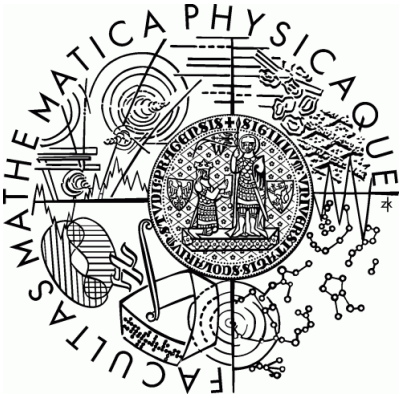
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Vladimír Rovenský

Určení smysluplnosti české věty na základě syntaktické informace

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Vidová Hladká Barbora, Ph.D.,
Studijní program: Informatika

2009

Děkuji Mgr. Barboře Vidové Hladké za pomoc při psaní této práce.

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

V Praze dne 2.5.2009 Vladimír Rovenský

Obsah

Seznam obrázků

Seznam tabulek

Název práce: Určení smysluplnosti české věty na základě syntaktické informace
Autor: Vladimír Rovenský
Vedoucí bakalářské práce: Mgr. Vidová Hladká Barbora, Ph.D., Ústav formální a aplikované lingvistiky
e-mail vedoucího: hladka@ufal.mff.cuni.cz

Abstrakt: Práce se zabývá navržením znalostního algoritmu pro rozpoznání smysluplnosti věty. Jedná se o velmi zajímavou úlohu v rámci zpracování přirozeného jazyka, například pro potřeby internetových vyhledávačů. "Smysluplnost" věty nelze definovat absolutně - v našem návrhu respektujeme systém rovin popisu přirozeného jazyka. Tento přístup vychází z koncepce tří vrstev, a sice vrstvy morfologické, syntaktické a sémantické - bakalářská práce bude pokrývat první dvě z těchto tří vrstev. Cílovým jazykem bude čeština.

Klíčová slova: smysluplnost

Title: Syntax-based classification of meaningful Czech sentences
Author: Vladimír Rovenský
Supervisor: Mgr. Vidová Hladká Barbora, Ph.D.,Institute of Formal and Applied Linguistics
Supervisor's e-mail address: hladka@ufal.mff.cuni.cz

Abstract: This thesis tries to formulate a knowledge-based algorithm for meaningful sentence classification. This is a very interesting task for the applications of natural language processing, such as the web search engines. "To-be-meaningful" is a feature that cannot be defined in an absolute way - we try to respect the natural language description layer system. In this approach, we pursue a layer system that goes from the morphological layer through the syntactical layer to the semantic layer - the bachelor thesis will cover the first two of three layers. Czech will be used as the object language.

Keywords: meaningful, meaningless

Kapitola 1

Úvod

Tato práce se zabývá problémem rozpoznání smysluplnosti české věty na základě morfologických a syntaktických informací. Jedná se tedy jednak o vlastní definici pojmu smysluplnost a následně navržení a implementaci procedury, která na svém vstupu očekává obecný shluk textových řetězců, o němž po určité analýze rozhodne, zda se jedná o smysluplnou větu ve smyslu zavedené definice.

Každý řetězec na vstupu je opatřen morfologickou (rod, číslo, pád...) a syntaktickou (větný rozbor) informací, kterou poskytne externí nástroj `tool_chain`, pokud dokáže zkoumaný řetězec rozpoznat. V opačném případě je k dispozici pouze informace o tom, že daný řetězec nebyl rozeznán.

Výstupem je výsledek analýzy (vstupem je/není smysluplná česká věta) a případně také podrobnější popis běhu programu.

Při řešení problému se nabízí více postupů, například statistický - rozhodovat o smysluplnosti věty na základě souhrnných informací extrahovaných z korpusů. Náš přístup se však snaží respektovat systém rovin jazyka. Jedná se o pravidlový systém, který pracuje na úrovni informací morfologické a syntaktické vrstvy. Tento přístup pomáhá sledovat, jaký přínos mají informace jednotlivých vrstev k úspěšnosti aplikace.

Předpokládáme také, že každému řetězci na vstupu bude přiřazen právě jeden základní tvar (lemma) a jemu odpovídající morfologické a syntaktické informace. To zajišťuje tzv. *tagger* - součást nástroje `tool_chain`. Ten vybere z množiny možných základních tvarů pro daný řetězec ten, který se v kontextu věty jeví jako nejpravděpodobnější (s úspěšností zhruba 96%).

Pokud se tedy například ve větě nachází řetězec *"ženu"* a *tagger* jej označí za podstatné jméno ve čtvrtém pádu, nezabýváme se dále možností, že by se mohlo jednat o sloveso v první osobě.

U nesmyslných vět může být tento postup zavádějící, neboť *tagger* pracuje na základě souhrnných dat získaných z množiny smysluplných vět. Nabízí se tedy coby alternativní postup zkoumat množinu všech možných základních tvarů (a jim odpovídajících morfologických a syntaktických informací) pro každý řetězec.

1.1 Pojem smysluplnost

Je vhodné si uvědomit, že pojem smysluplnost věty může mít pro různé lidi značně odlišné interpretace. Existuje spousta kritérií, podle kterých přirozeně hodnotíme, zda nám ta která věta dává smysl. U většiny vět se lze shodnout na tom, že jsou buď zjevně smysluplné, nebo zjevně nesmyslné. Můžeme ale také nalézt příklady vět, na jejichž smysluplnost mohou mít různí lidé různé názory:

Díky mě se nic hrozného nestalo.	Pokud věta obsahuje gramatickou chybu, budeme ji chápat jako smysluplnou?
Druhá světová válka se odehrála před rokem.	Chybí čtyřčíslí udávající rok, nicméně z hlediska morfologie (nebo člověka neznalého historie) se jedná o správně zkonstruovanou větu.
Chléb, mléko, máslo, sýr.	Pokud řetězec neobsahuje sloveso, prohlásíme jej za smysluplnou větu?

S ohledem na tyto skutečnosti je třeba při řešení podobného úkolu jasně definovat kritéria, podle kterých hodnotíme smysluplnost. Součástí práce je i tato definice.

Kapitola 2

Popis použitého algoritmu

Základní myšlenkou je zkoumat nejprve smysluplnost menších částí věty (dvojic / trojic slov). K tomu slouží předem definovaná sada podmínek týkajících se hodnot jednotlivých morfologických kategorií, syntaktických informací, slovosledu atp. Na základě informací o smysluplnosti těchto menších úseků pak rozhodneme o smysluplnosti celé věty.

2.1 Dělení souvětí na věty jednoduché

Jelikož vstupní věta může být prakticky libovolně dlouhá, je vhodné ji rozdělit na menší části a ty následně analyzovat zvlášť. Proto je prvním krokem algoritmu rozdělení vstupní věty na jednotlivé věty jednoduché. Celé souvětí je potom smysluplné, právě když jsou smysluplné všechny jeho věty jednoduché. Větou jednoduchou je myšlena věta, která obsahuje nejvýše jedno sloveso. Souvětí je věta obsahující alespoň dvě slovesa, skládající se z vět jednoduchých.

Algoritmus dělení souvětí na věty jednoduché

Program nejprve najde pozici prvního slovesa ve větě, tuto si zapamatuje. Následně postupuje od posledního slova věty směrem k prvnímu a hledá sloveso (ne v infinitivním tvaru). Když na nějaké narazí, pokračuje v načítání slov dokud nenarazí na slovní jednotku, která může oddělovat věty (např. spojka, čárka). V tom okamžiku byla nalezena věta jednoduchá. Program načte všechny další případné oddělovače (může jich být za sebou víc, např. *“, ale i”*) , uloží větu jednoduchou a opakuje.

V okamžiku, kdy narazí na první sloveso věty, automaticky přidá coby větu jednoduchou zbytek vstupu a končí.

Je zřejmé, že takto jednoduchý algoritmus nebude mít při dělení složitějších souvětí stoprocentní účinnost. To však není zásadní problém, účelem tohoto kroku algoritmu je zejména rozdělit vstupní větu na menší úseky před dalším zpracováním. Pokud některé z nich přesně neodpovídají větám jednoduchým, pravděpodobně to (ne)smysluplnost věty příliš neovlivní.

Příklad: *"Umění sahá hodně daleko zpátky , a čím dál se díváme , tím kontroverznější jsou důkazy."* Prvním slovesem je *"sahá"*. Algoritmus postupuje od konce věty. Najde sloveso *"jsou"*, dojde k čárce a přidá větu *", tím kontroverznější jsou důkazy."* Dále najde sloveso *"díváme"*, po přečtení *",a čím"* přidá další větu (*"a čím dál se díváme"*). Dále narazí na sloveso *"sahá"*, to je prvním slovesem věty a proto celý zbytek vstupu přidá coby poslední větu jednoduchou (*"Umění sahá hodně daleko zpátky"*).

2.2 Algoritmus zjišťování smysluplnosti jednoduché věty

Pakliže máme vstupní větu rozdělenou na jednotlivé věty jednoduché, je dalším logickým krokem zjišťování smysluplnosti věty jednoduché.

2.2.1 Podmínky pro smysluplné dvojice slov

Analýza jednoduché věty probíhá tak, že se pro každou dvojici slovních jednotek pokusíme zjistit, zda je smysluplná. K tomu jsou pro každou dvojici slovních druhů definovány (morfologické a syntaktické) podmínky, za kterých je tato smysluplná. Může jich být pro jednu dvojici slovních druhů několik, jedna, nebo vůbec žádná, pokud neexistuje smysluplné spojení těchto slovních druhů. Program prochází všechny dvojice slovních jednotek ve větě a na základě jejich slovních druhů vybere příslušné podmínky, jejichž platnost následně ověřuje. Pokud testovaná dvojice slovních jednotek splňuje alespoň jednu z nich, potom tvoří smysluplnou dvojici slovních jednotek. V následujícím textu budou označovány jako *podmínky typu (1)*.

Příklad: Dvojice podstatné a přídavné jméno je smysluplná, pokud mají stejný rod, pád i číslo, případně můžeme požadovat, aby přídavné jméno bylo ve větě přívlastkem.

U dvojice podstatné jméno a předložka můžeme pro smysluplnost požadovat, aby předložka ve větě předcházela podstatnému jménu. O každé předložce navíc víme, se kterými pády podstatného jména se může vázat, můžeme tedy zároveň u podstatného jména vyžadovat správný pád vzhledem k předložce.

Pro ilustraci bude dále sloužit (jednoduchá) věta: *"Ve škole, v práci a v knihovně často trávím svůj volný čas."* Program při zkoumání této věty nalezne následující smysluplné dvojice slovních jednotek dle podmínek typu (1):

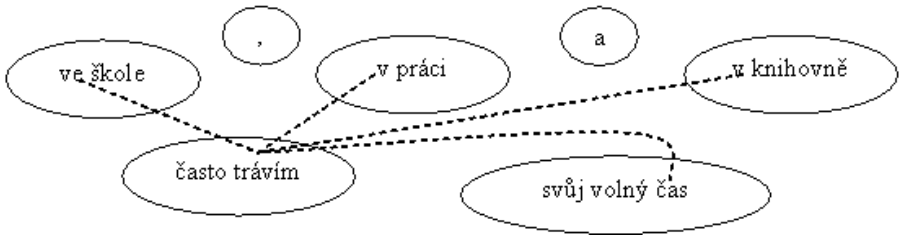
ve škole + v práci + v knihovně	Podmínky pro spojení předložky a podstatného jména lze definovat například tak, že předložka musí být ve větě před podstatným jménem, a to musí mít správný pád vzhledem k základnímu tvaru předložky.
často trávím	Dvojice příslovce a sloveso je většinou smysluplná bez dalších omezení, resp. omezení by ubrala na obecnosti.
svůj čas + volný čas	Dvojice podstatné jméno + slovní druh rozvíjející podstatné jméno (například přídavné jméno, zájmeno) lze rozpoznat podle shody v rodu, pádu, čísle atp.
trávím v + trávím čas	Pro pospojování logických částí věty (příslovečné určení místa, podmětná část, přísudková část atp.) jsou definovány (velmi obecné) podmínky spojující například sloveso s předložkou, nebo sloveso s podstatným jménem.

2.2.2 Definice pojmu smysluplnost

Jakmile máme takto definovány potřebné podmínky, můžeme větu přepracovat na graf $G = (V, E)$, kde V = slovní jednotky věty a $\forall t1, t2$ slovní jednotky: $t1, t2 \in E \Leftrightarrow \exists$ podmínka pro smysluplnou dvojici, které t1 a t2 vyhovují.

Smysluplnost věty je potom definována jako **souvislost takto vzniklého grafu smysluplnosti G**.

Takto vypadají komponenty konstruovaného grafu po aplikaci podmínek (1):



Obrázek 2.1: Komponenty grafu po aplikaci podmínek (1)

Je vidět, že graf zatím souvislý není, ačkoli se jedná o smysluplnou větu - je třeba ještě zapojit do věty spojky a čárky. Vzhledem k tomu, že vztah pro zapojení spojky je přirozeně ternární (spojka a dva výrazy, které spojuje), nemá smysl jej postihovat binárními podmínkami typu (1). Proto je za tímto účelem zaveden speciální typ podmínek (2) popsaný v následující části.

Při pohledu na graf si lze všimnout ještě jednoho problému. Jednotlivé komponenty (vyznačené plnou čarou) nyní představují menší logické úseky věty, například rozvité podstatné jméno (*"svůj volný čas"*), rozvité sloveso (*"často trávím"*) nebo příslovečná určení (*"ve škole", "v práci", "v knihovně"*). Tyto úseky lze ve větě poměrně přesně rozeznat na základě podmínek typu (1), neboť mají určitou přesně danou strukturu. Vztahy mezi nimi (čárkované hrany) jsou ale naopak velmi obecné. Jedná se například o spojení podstatného jména a slovesa pro spojení úseků *"často trávím"* a *"svůj volný čas"* nebo napojení příslovečného určení místa na sloveso (*"často trávím"* a *"ve škole"*).

Pokud definujeme podmínku typu (1) pro takový vztah, bude téměř nebo úplně prázdná a budou jí tedy vyhovovat jakékoli dvě slovní jednotky s da-

nými slovními druhy. Potom stačí, aby například v každém úseku věty bylo jedno podstatné jméno a v celé větě jediné sloveso a graf bude souvislý bez ohledu na smysluplnost věty. Proto je vhodné nějakým způsobem omezit vznikání hran odpovídajících těmto obecným vztahům. K tomu slouží podmínky typu (3) popsané dále.

2.2.3 Podmínky pro spojování částí věty

Výše definované podmínky typu (1) dokáží vcelku rozumně rozložit (implicitně) větu na několik úseků, například rozvité podstatné jméno, určení místa atp. Problémem je rozumné zapojení spojek a čárek do konstruovaného grafu. K tomuto účelu slouží podmínky typu (2) – definice ternárního vztahu slovní druh – spojka (čárka) – slovní druh, který říká, za jakých podmínek může spojka spojovat dvojici slovních jednotek s danými slovními druhy. Je možné přesně definovat morfologické vlastnosti slovních jednotek na každé straně spojovacího výrazu a tím zpřísnit spojování úseků věty.

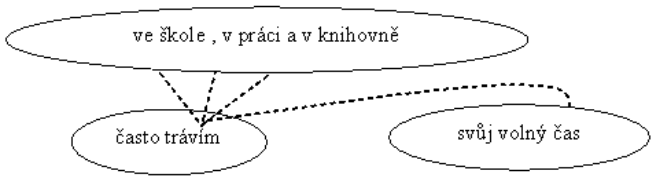
Příklad: Vraťme se k rozpracované větě *"Ve škole, v práci a v knihovně často trávím svůj volný čas."* Při hledání trojic splňujících některou podmínku typu (2) budou nalezena spojení: *"škole , práci"* a *"práci a knihovně"* – v obou případech se jedná o spojení podstatných jmen ve stejném pádě, jednou čárkou a jednou spojkou.

Obecně jsou pro každou dvojici shodných slovních druhů definovány podmínky (2), například dvě přídavná jména mohou být spojena spojkou (čárkou), pokud mají stejný pád a slovní poddruh, slovesa mohou být spojena spojkou, pokud jsou obě v infinitivním tvaru atp.

V této větě byly tedy podmínkami (2) spojeny tři určení místa do jednoho souvislého úseku a rovněž byly do věty zapojeny slovní jednotky *','* a *'a'* – i ty jsou brány v potaz při určování smysluplnosti.

Pro tento příklad by se zřejmě zdálo logičtější, kdyby byly podmínkami (2) spojeny předložky místo podstatných jmen, nicméně algoritmus se zastaví u první nalezené dvojice, jež může zkoumaná spojka / čárka spojovat – pro potřeby analýzy smysluplnosti stačí, že existuje nějaká taková dvojice, kdyby hledání pokračovalo dál, byla by nalezena i možnost spojení předložek.

Takto budou vypadat komponenty konstruovaného grafu po aplikaci podmínek (2):



Obrázek 2.2: Komponenty grafu po aplikaci podmínek (2)

2.2.4 Zakázané podmínky

Ne všechny jevy je možné postihnout pouze pozitivními podmínkami. Často se setkáváme se spojeními, která sama o sobě učiní větu nesmyslnou a je proto výhodné mít možnost formulovat takovou podmínku, při jejímž naplnění je věta okamžitě prohlášena za nesmyslnou.

Kteroukoli podmínku typu (1) nebo (2) je možné deklarovat jako zakázanou přidáním speciální klauzule (viz dále). Program při svém běhu zkoumá tyto podmínky přednostně a pokud je některá z nich splněna, okamžitě větu prohlásí za nesmyslnou bez další analýzy. Proto by zakázané podmínky měly postihovat výhradně taková spojení, která jsou za všech okolností nesmyslná. Algoritmus automaticky kontroluje, zda jsou všechny pozice morfologického tagu obou slovních jednotek, na které se odkazuje zakázaná podmínka definovány (tedy že nemají hodnotu 'X'). V opačném případě není zakázaná podmínka vyhodnocována.

Příklad: Mějme větu *"K jemu se tato zpráva nedonesla."* Spojení *"k jemu"* je vždy špatně, proto pro něj můžeme definovat zakázanou podmínku. Definice je stejná jako pro kteroukoli jinou podmínku typu (1), pouze přidáme speciální klauzuli *forbidden*. Jakmile dojde k analýze příslušné dvojice slovních jednotek, bude věta prohlášena za nesmyslnou.

Vyhodnocování zakázaných podmínek je možné snadno vypnout zadáním volby *-f 0* při startu programu, případně nastavením příslušné možnosti v konfiguračním souboru.

2.2.5 Vyžádané podmínky

Opakem zakázaných podmínek jsou vyžádané podmínky. Jedná se o takové podmínky, které musí některá dvojice slovních jednotek ve větě splnit, aby byla věta smysluplná. Narozdíl od zakázaných podmínek zde není možné určit obecnou sadu vyžádaných podmínek pro všechny situace, záleží na konkrétní představě uživatele o tom, co přesně pro něj znamená smysluplná věta. Program s vyžádanými podmínkami pracuje na základě skupin - je možné definovat libovolný počet skupin vyžádaných podmínek a při analýze věty musí být splněny všechny podmínky z alespoň jedné skupiny.

Formálněji: definujeme-li n skupin vyžádaných podmínek $S_1..S_n$ a označíme-li k_i počet podmínek ve skupině S_i a $\{S_{ij}\}_{j=1}^{k_i}$, podmínky v i-té skupině, pak v analyzované větě musí pro každé $j \in \{1..k_i\}$ existovat nějaká dvojice slovních jednotek, která splňuje podmínku S_{ij}, aby věta vyhověla skupině S_i. Pokud existuje alespoň jedna skupina, které věta vyhoví, pokračuje program standardní analýzou smysluplnosti, jinak větu zamítne coby nesmyslnou.

Při definici podmínky k tomu slouží klauzule `req #`, kde # je číslo skupiny, do které má vyžádaná podmínka patřit.

Příklad: Mohu definovat následující tři skupiny vyžádaných podmínek:

- Skupina 1: podstatné jméno coby podmět navázané na sloveso (přísudek), přídavné jméno rozvíjející podstatné jméno.
- Skupina 2: podstatné jméno coby podmět navázané na sloveso (přísudek), předložka navázaná na podstatné jméno.
- Skupina 3: spojení dvou podstatných jmen spojkou nebo čárkou

Těmto požadavkům vyhoví například tyto věty:
"Auto projelo kolem vysokou rychlostí." - splňuje skupinu 1.
"Děti chodí do školy." - splňuje skupinu 2.
"V košíku byl chléb, maso a čerstvé mléko." - splňuje skupinu 1 a 3.

Naopak jako nesmyslné budou zamítnuty například následující věty, neboť nesplňují kompletní požadavky žádné skupiny:
"Městský úřad Litomyšl"
"V Praze dne 15.4.1999"

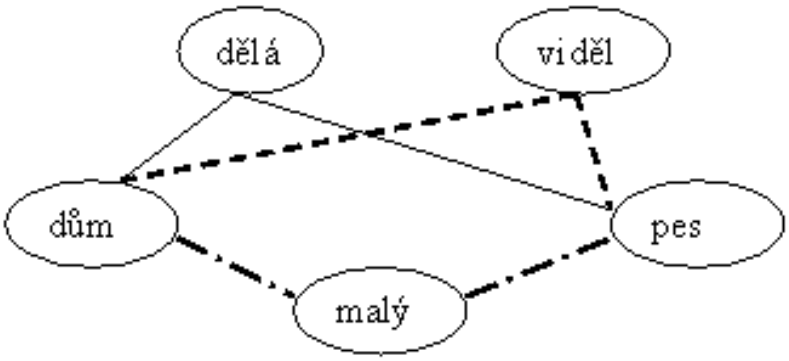
2.2.6 Podmínky pro maximální počet vztahů

Dalším prostředkem pro zpřísnění analýzy jsou podmínky pro omezení počtu vztahů (podmínky typu (3)). Některé vztahy ve větě jsou natolik obecné, že jim odpovídající podmínka typu (1) je prakticky prázdná a vyhovují jí tedy jakékoli dvě slovní jednotky daných slovních druhů. Například podmínka spojující podstatné jméno / částici / citoslovce se slovesem nemá prakticky žádné morfologické informace, které by se daly kontrolovat, tedy jakékoli citoslovce se naváže na všechna slovesa věty bez omezení.

Podmínky typu (3) poskytují prostředek k vyjádření omezení typu "jedna částice se může navázat nejvýše na jedno sloveso" atp. To vede k omezení počtu hran v konstruovaném grafu smysluplnosti a tedy k přesnější analýze zejména nesmyslných vět.

Příklad: Do třetice všeho dobrého se podívejme na větu *"Ve škole, v práci a v knihovně často trávím svůj volný čas."* Tato věta je smysluplná, proto zde podmínky typu (3) nehrají zásadní roli. Zde by se projevila pouze jediná podmínka (3) říkající, že podstatné jméno se naváže nejvýše na jednu předložku. Tím pádem by například nedošlo ke vzniku dvojic *"ve práci"* nebo *'v škole"*. Na výsledku to však nic nezmění, věta by byla vyhodnocena jako smysluplná i bez podmínek typu (3).

Nyní uvažme zjevně nesmyslnou větu *"Dům dělá malý viděl pes."* Tato "věta" neobsahuje spojku ani čárku, takže dělením na věty jednoduché projde beze změny a bude analyzována coby věta jednoduchá. Takto by vypadal zkonstruovaný graf bez podmínek (3):



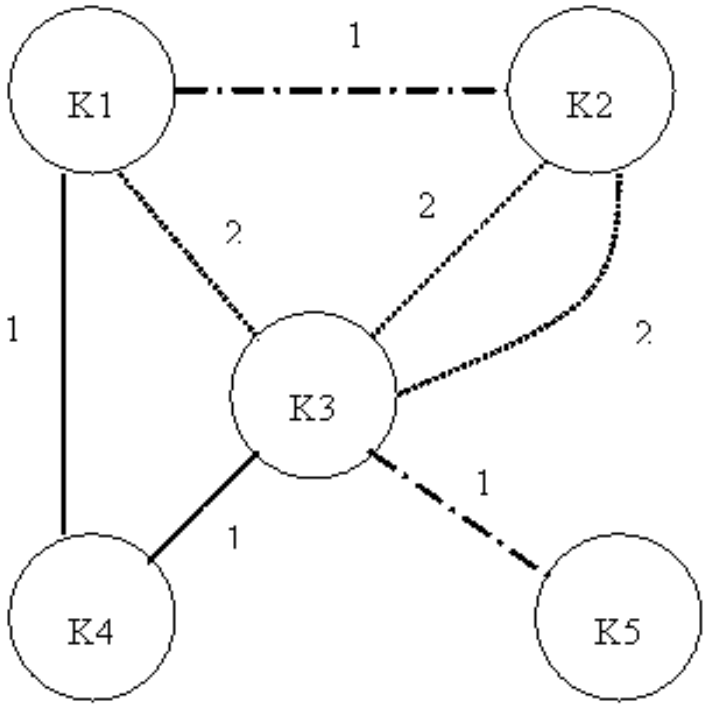
Obrázek 2.3: Nesmyslná věta a použití podmínek (3)

Všechny vyznačené hrany (bez ohledu na styl čáry) odpovídají podmínkám (1). Graf je zjevně souvislý, navzdory naprosté nesmyslnosti věty. Zde poskytuje zavedení podmínek (3) větší přesnost. Máme například definovány podmínky (3) určující, že přídavné jméno se smí navázat nejvýše na jedno podstatné jméno a podstatné jméno nejvýše na jedno sloveso. Hrany nyní svým stylem (plná / přerušovaná / čerchovaná čára) odpovídají těmto omezením: z každé množiny hran stejného stylu můžeme použít pouze (v tomto případě) jednu.

Nyní už graf díky menšímu počtu hran souvislý nebude. Nejvýše lze vybrat tři hrany, ale kostra grafu na pěti vrcholech má hrany čtyři.

Kontrola těchto podmínek je realizována tak, že pokud ze všech smysluplných vztahů lze vybrat alespoň jednu jejich podmnožinu vyhovující podmínkám (3), stačí to pro smysluplnost věty. Při kontrole spojitosti grafu smysluplnosti jsou vždy nejprve všechny hrany omezené některou podmínkou typu (3) opomenuty. Pokud je takto vzniklý graf spojitý, je zřejmé, že je spojitý i původní graf a věta je smysluplná.

Pokud nový graf spojitý není, pak se skládá z několika komponent, mezi kterými vedou pouze hrany omezené podmínkami typu (3). Program následně hledá způsob, jakým vybrat podmnožinu těchto hran, která všechny komponenty spojí do jedné a zároveň neporuší žádnou z podmínek (3). Pokud takovou podmnožinu najde, věta je prohlášena za smysluplnou, v opačném případě je prohlášena za nesmyslnou. Situaci popisuje obrázek.



Obrázek 2.4: Aplikace podmínek (3)

K1 – K4 jsou komponenty vzniklé DFS[1] průchodem po hranách grafu neomezených podmínkami (3), mezi nimi vedou hrany omezené podmínkami (3). Z hran vyznačených jedním stylem lze vybrat pouze k hran, kde k je číslo uvedené u každé hrany z této skupiny. Situaci na obrázku by např. vyhovovala množina hran K3K5, K1K4,K1K3 a K2K3.

[1]Depth First Search, tedy průchod grafem do hloubky – algoritmus, který dokáže projít postupně všechny vrcholy (souvislého) grafu. Podrobný popis lze nalézt na `http://en.wikipedia.org/wiki/Depth-first_search` nebo v češtině na `http://cs.wikipedia.org/wiki/DFS`

2.3 Časová složitost algoritmu

Označme n = počet slovních jednotek ve větě. Načtení věty proběhne zjevně v $O(n)$, načtení souboru s podmínkami $O(s)$, kde s je délka souboru s podmínkami.

Rozdělení souvětí na věty jednoduché proběhne v čase $O(n)$, následné hledání dvojic podle podmínek (2) rovněž $O(n)$, neboť v obou částech je nejhorším případem průchod celé věty.

Při hledání podmínek (1) jsou zkoumány všechny dvojice slovních jednotek věty a pro každou by měly být v konstantním čase (asociativní pole) vyhledány a ověřeny odpovídající podmínky. Pokud budeme předpokládat, že počet definovaných podmínek pro jednu dvojici slovních jednotek bude konstantní, vychází časová složitost této fáze $O(n^2)$.

Dále je třeba projít větu a zanést do konstruovaného grafu informaci o podmínkách (3), pokud opět budeme uvažovat konstantní počet těchto podmínek, vychází čas $O(n)$.

Předposlední fází je kontrola souvislosti grafu, ta sestává jednak z DFS průchodu grafu. Zde je třeba uvažovat složitost $O(V + E)$, kde V je počet vrcholů grafu a E je počet hran. Vzhledem k tomu, že vrcholy grafu tvoří slovní jednotky, vychází časová složitost $O(n^2)$.

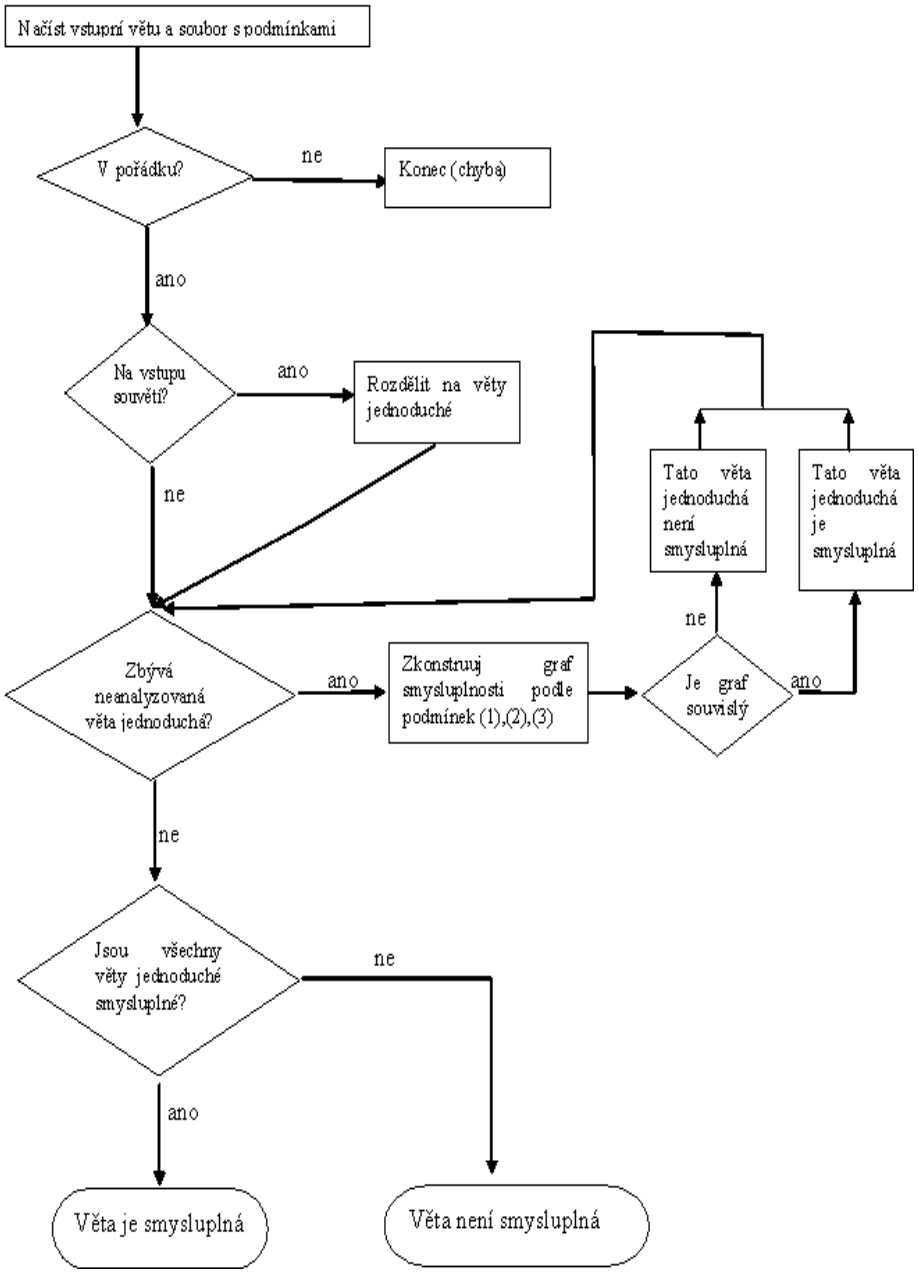
Konečně nejnáročnější fází je hledání řešení, které splňuje všechny definované podmínky (3). Označíme-li p počet hran omezených podmínkami (3), potom všech podmnožin této množiny je až 2^p a kontrola jedné z nich trvá až $O(n)$, takže tato fáze trvá až $O(n * 2^p)$, což je řádově déle, než zbytek algoritmu. Proto je v aplikaci zabudována pevná mez pro počet iterací procedury vykonávající tuto fázi, po jejímž překročení se namísto generování všech možností použije lineární heuristika – vyberou se vždy ty vazby, jejichž slovní jednotky k sobě mají ve větě nejblíže. Věta je vyhodnocena jako smysluplná, pokud je graf po přidání nalezených hran spojitý.

Celková složitost tedy vychází v nejhorším případě $O(s) + O(n^2)$ kde s je délka souboru s podmínkami a n je počet slovních jednotek věty.

2.4 Shrnutí algoritmu

Následuje stručné shrnutí celého použitého algoritmu

1. Načíst vstupní větu a soubor s podmínkami.
2. Pokud je na vstupu souvětí, je rozděleno na věty jednoduché. Celá věta je smysluplná právě když jsou smysluplné všechny její věty jednoduché.
3. Pro každou větu jednoduchou zkonstruovat "graf smysluplnosti", vrcholy = slovní jednotky věty a hrany přidávány na základě definovaných podmínek, tedy:
   (a) Najít všechny spojky a čárky ve větě a aplikovat podmínky (2)
   (b) Projít všechny dvojice slovních jednotek ve větě a aplikovat pro ně podmínky (1). Pokud je nalezena zakázaná podmínka, analýza končí a věta je prohlášena za nesmyslnou.
   (c) Ověřit splnění vyžádaných podmínek, pokud jsou definovány. Pokud věta nevyhovuje definovaným vyžádaným podmínkám, je prohlášena za nesmyslnou.
4. Zkontroluje se souvislost vzniklého grafu. Věta jednoduchá je smysluplná, právě když je graf souvislý. Zároveň je třeba kontrolovat platnost podmínek (3).
   (a) Provést DFS na sestavený graf, bez použití jakékoli hrany, která je omezená některou z podmínek (3). Pokud jsou takto dosažitelné všechny vrcholy, věta je smysluplná.
   (b) Pokud vznikne několik komponent, mezi nimi budou hrany omezené podmínkami (3) a je třeba hledat řešení problému popsaného v kapitole o podmínkách typu (3). Pokud nějaké řešení existuje, je věta jednoduchá vyhodnocena jako smysluplná, v opačném případě jako nesmyslná.



Obrázek 2.5: Vývojový diagram

Kapitola 3

Popis aplikace Sense

3.1 Struktura CD

S prací je dodáváno CD s následující adresářovou strukturou:

- Windows - zkompilovaná verze aplikace `Sense` pro OS Windows
- Bakalářská práce - tato práce
- ConEdit - Pomocná utilita `ConEdit` pro správu souboru podmínek
- src - Zdrojové kódy aplikace.

3.2 Instalace programu pod OS Windows

Stačí do libovolného adresáře rozbalit verzi aplikace pro OS Windows, tj. adresář `Windows` z instalačního balíčku `Sense`.

3.3 Instalace programu pod OS Unix

Nejprve do libovolného adresáře rozbalte soubory v adresáři `src`. Dále je třeba standardním způsobem tyto zdrojové kódy zkompilovat - například spuštěním příkazu
"`g++ ` ``ls ./*.cpp`` `  -o sense`" v adresáři, kam byly rozbaleny.

3.4 Použití nástroje tool_chain

Jelikož program k použití vyžaduje na vstupu text analyzovaný nástrojem `tool_chain`, zde je krátký popis jeho použití. Podrobnější informace o nástroji `tool_chain` lze nalézt na adrese `http://ufal.mff.cuni.cz/rest/CAC/doc-cac20/cac-guide/cz/html/ch3.html#nastroje-zprac`.

Jedná se o nástroj zajišťující tvaroslovnou a syntaktickou analýzu českých textů. Pro potřeby dokumentovaného programu jsou důležité následující služby:

- Tokenizace - rozdělení obecného vstupního řetězce na jednotlivé slovní jednotky.
- Morfologická analýza - existují slova, která mají více možných základních tvarů - například slovo *"ženu"* může být jak tvarem slovesa *"hnát"*, tak podstatného jména *"žena"*. Morfologická analýza pro každou slovní jednotku poskytne množinu možných dvojic základní tvar - odpovídající morfologická informace.
- Tagování - z množiny poskytnuté morfologickou analýzou vybere tu dvojici, která se v kontextu věty jeví jako nejpravděpodobnější (s úspěšností cca 96%)
- Parsování - přidání informace o větném rozboru.

Pro analýzu vstupní věty nástrojem `tool_chain` je třeba spustit jej s následujícími parametry:

```
tool_chain -tATP -i soubor_s_větou -o soubor_s_výstupem
```

Volba `-tATP` zajistí, že budou provedeny všechny čtyři výše uvedené služby, `soubor_s_větou` je textový soubor (prostý text) obsahující vstupní větu, `soubor_s_výstupem` je jméno souboru, kam má být uložen výsledek zpracování.

Rovněž je možné zadat vstup bez větného rozboru, věta potom bude analyzována pouze na základě morfologických informací. To znamená, že syntaktické klauzule v definovaných podmínkách budou ignorovány. Při spuštění bez parametru `-P` neposkytne `tool_chain` k dané větě syntaktickou informaci:

```
tool_chain -tAT -i soubor_s_větou -o soubor_s_výstupem
```

Výstupním formátem bude v tomto případě formát CSTS. Jedná se o značkový formát na bázi SGML. Jeho podrobný popis je k nahlédnutí na adrese `http://ufal.mff.cuni.cz/pdt2.0/doc/data-formats/csts/html/DTD-HOME.html`.

3.5 Příklad vstupu

Takto může vypadat příklad vstupu programu ve formátu CSTS, který odpovídá spuštění nástroje `tool_chain` s parametry `-tATP` (tedy služby tokenizace, morfologická analýza, tagování a větný rozbor). Vstupní větou byl řetězec "Váš boj je i naším bojem."

```
<csts lang=cs>
<doc file="in" id=1>
<a>
<mod>?
<txtype>?
<genre>?
<med>?
<temp>?
<authname>?
<opus>in
<id>001
</a>
<c>
<p n=1>
<s id=n01w-s14>
<f id=n01w-s14W1>Váš<l>tvůj_^(přivlast.)<t>PSYS1-P2-------<r>1<g>2<A>Atr
<f id=n01w-s14W2>boj<l>boj<t>NNIS1-----A----<r>2<g>3<A>Sb
<f id=n01w-s14W3>je<l>být<t>VB-S---3P-AA---<r>3<g>0<A>Pred
<f id=n01w-s14W4>i<l>i<t>J^--------------<r>4<g>5<A>AuxZ
<f id=n01w-s14W5>naším<l>můj_^(přivlast.)<t>PSZS7-P1-------<r>5<g>6<A>Atr
<f id=n01w-s14W6>bojem<l>boj<t>NNIS7-----A----<r>6<g>3<A>Pnom
<D>
<d id=n01w-s14W7>.<l>.<t>Z:--------------<r>7<g>0<A>AuxK </c>
</doc>
</csts>
```

Tabulka 3.1: Příklad vstupu

Z CSTS výstupu nástroje `tool_chain` jsou pro aplikaci `Sense` relevantní především značky <f> (vlastní slovní jednotka), <d> (interpunkce), <MDl> (základní tvar) a <MDt>, za kterou následuje potřebná morfologická informace reprezentovaná patnáctipoziční značkou (tagem). Každá pozice této značky jednoznačně určuje jednu informaci, například slovní druh, rod, osobu, číslo nebo pád slovní jednotky. Jejich přesný popis je k nahlédnutí na `http://ufal.mff.cuni.cz/~hladka/rp200809/cz-appendix-D.pdf`.

Pokud jsou k dispozici, jsou rovněž uloženy značky <r>, <g> a <a>. Jedná se o syntaktickou informaci - značka <r> obsahuje identifikátor slovní jednotky v rámci věty (odpovídá pořadí). Značka <g> udává nadřazený vrchol jednotky, tedy identifikátor slovní jednotky o úroveň výše v syntaktickém stromě (viz dále). Konečně značka <a> říká o jaký typ větného členu se jedná. Možné hodnoty jsou popsány na `http://ufal.mff.cuni.cz/pdt2.0/doc/data-formats/csts/html/A.html`.

Příklad: Zvažme zpracování slova *"boj"* v příkladu výše:

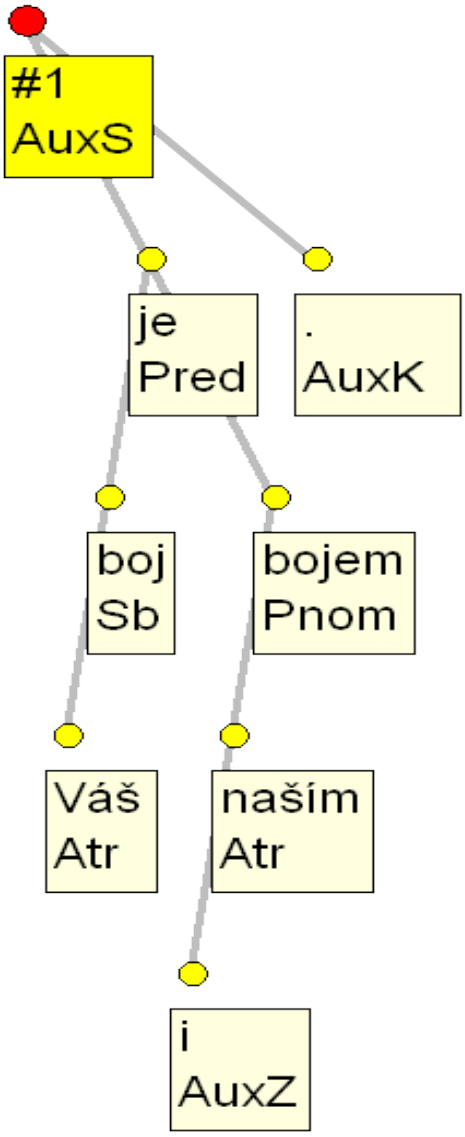
```
<f id=n01w-s14W2>boj<l>boj<t>NNIS1-----A----<r>2<g>3<A>Sb
```

Z této řádky CSTS výstupu nástroje `tool_chain` vidíme, že základním tvarem této slovní jednotky je *boj*. Následuje morfologická informace:

```
NNIS1-----A----
```

První pozice udává informaci o slovním druhu, v tomto případě se jedná o podstatné jméno (symbol 'N'). Na druhém místě se nachází informace o slovním poddruhu, zde se jedná o obecné podstatné jméno ('N'). Jako další následují informace o rodu ('I' - rod mužský neživotný), čísle ('S' - jednotné číslo) a pádu (zde přímo číslo - první pád). Na jedenácté pozici je ještě informace o negaci - zde se jedná o afirmativ ('A'). Po morfologické značce následují syntaktické informace. Značka <r> udává identifikátor slovní jednotky v rámci této konkrétní věty (odpovídá pořadí slovní jednotky - zde 2). Ve značce <g> je identifikátor nadřazeného uzlu v syntaktickém stromu, v tomto případě je nadřazeným uzlem přísudek *"je"*. Nakonec ve značce <A> je uvedeno, o jaký typ větného členu se jedná - tato slovní jednotka je ve zkoumané větě podmětem ('Sb').

Na následujícím obrázku je syntaktický strom výše uvedené věty. Vrcholy odpovídají slovním jednotkám (až na kořenový vrchol), hrany vedou vždy od slovní jednotky k jejímu nadřazenému vrcholu (značka <g>). U každého vrcholu je uveden větný člen (značka <A>).



Obrázek 3.1: Syntaktický strom poskytovaný nástrojem tool_chain

Nástroj `tool_chain` umožňuje provést dávkové zpracování většího množství vět najednou, ne jen jedné. K tomu stačí dodat mu na vstup soubor se všemi větami oddělenými znakem nového řádku. Výstup tohoto zpracování je možné dodat na vstup aplikaci `Sense`, která rovněž provede dávkovou analýzu smysluplnosti.

3.6 Spuštění programu

Samotné spuštění je stejné pod Windows i Unixem. V adresáři, kam byl program nainstalován, se nalézají dva soubory: `conditions.txt` a `sense.exe` (respektive `sense` v OS Unix / Linux). V prvním jsou uložené podmínky, podle kterých program větu analyzuje (jejich formát bude popsán dále). Při dodržení tohoto formátu je možné tento soubor libovolně rozšiřovat a upravovat.

Druhý z nich slouží ke spuštění aplikace a přijímá tyto volby:

-h Zobrazí krátkou nápovědu.

-i soubor Specifikace souboru se vstupem, jímž je věta zpracovaná programem `tool_chain` ve formátu CSTS. Program očekává na vstupu větu prošlou tokenizací, morfologickou analýzou, tagováním a parsováním nástroje `tool_chain`. Výchozí hodnotou je standardní vstup.

-o soubor Specifikace souboru, do nějž má být ukládán výstup programu. Výchozí hodnotou je standardní výstup.

-pc Pokud je zadána tato volba, je pro každou větu před vlastní analýzou proveden průnik[1] konstruovaného grafu smysluplnosti se syntaktickým stromem, který poskytuje nástroj `tool_chain`. Implicitně vypnuto.

[1] V některých případech (například koordinace) je struktura grafu smysluplnosti jiná než u syntaktického grafu a průnik by vedl k přílišnému zpřísnění analýzy. Proto je v takových případech místo existence odpovídajících hran kontrolována pouze existence cesty délky nejvýše 2 mezi odpovídajícími vrcholy.

-v Zapnutí podrobného výstupu, program vypíše nalezené vztahy odpovídající definovaným podmínkám.

-g #[%] Často se stává, že program `tool_chain` nerozezná některé ze slov na vstupu, například kvůli překlepu, pravopisné chybě nebo prostě proto, že dané slovo nemá v databázi. Takovým slovům přiřadí na místě slovního druhu symbol 'X' a neposkytuje žádnou další morfologickou informaci, následkem čehož tato slova většinou nevyhoví žádné z definovaných podmínek a způsobí, že je věta vyhodnocena jako nesmyslná. Toto chování programu je mnohdy příliš přísné, proto lze za pomoci volby *-g* určit, kolik nerozpoznaných slovních jednotek může být z věty vypuštěno. Pokud je volba použita s parametrem #, bude z věty ještě před vlastní analýzou odstraněno # nerozpoznaných slovních jednotek. Po přidání symbolu '%' bude zadaná hodnota chápána jako procentuální podíl nerozpoznaných slovních jednotek na celkové délce věty. Odstraňování probíhá vždy od začátku věty (zleva).

-c # Zapnutí vypisování kolizí. Kolizí se rozumí takový vztah mezi slovními jednotkami, který téměř vyhověl některé z definovaných podmínek, avšak na některé části selhal. Parametr je následován číslem, které udává procentuální podíl splněných klauzulí na celkovém počtu klauzulí v podmínce, při jehož překročení kterém nastane vznik kolize. Například parametr `-c 75` znamená, že všechny vztahy, které splňují alespoň 75% klauzulí nějaké podmínky, budou vypsány coby kolize.

-co soubor Určení souboru, do kterého mají být ukládány nalezené kolize (viz volba *-c*). Výchozí hodnotou je standardní výstup.

-p 1/0 Kontrola napojení předložek. Pokud je zadána tato volba, je vyžadováno, aby každá předložka byla smysluplně navázána na některý větný člen, jinak je věta vyhodnocena jako nesmyslná. Implicitně zapnuto.

-f 1/0 Zapne / vypne vyhodnocování zakázaných podmínek. Pokud je vyhodnocování zakázaných podmínek vypnuto, jsou všechny tyto podmínky ignorovány. Implicitně zapnuto.

-m Vynutí použití pouze morfologických informací, pokud jsou na vstupu k dispozici informace syntaktické, jsou ignorovány. Implicitně vypnuto.

3.7 Soubor s nastavením

Po prvním spuštění programu je automaticky vygenerován konfigurační soubor `sense.ini`, který obsahuje veškerá výše uvedená nastavení programu. Program vždy nejprve načte nastavení z tohoto souboru, až poté nastavení z příkazové řádky takže není třeba často používané volby vždy předávat v parametrech programu.

Pokud je pro dané nastavení hodnota jak v souboru, tak na příkazové řádce, použije se hodnota z příkazové řádky.

3.8 Příklad výstupu

Mějme vstupní větu "Malé děti chodí do školy" analyzovanou nástrojem `tool_chain` a uloženou v souboru `input.txt`. Pokud chceme pouze základní informaci o výsledku analýzy, stačí spustit program následovně:

```
sense.exe -i input.txt -o output.txt
```

Výstup v souboru `output.txt` bude následující:

```
Checking sentence:
Malé děti chodí do školy

Sentence makes sense.
-----------------------
```

Pokud budeme chtít podrobnější výstup programu, přidáme volbu `-v`:

```
sense.exe -i input.txt -o output.txt -v
```

V souboru `output.txt` poté nalezneme následující výstup:

```
Checking sentence:
Malé děti chodí do školy

Checking bare sentence:
Malé děti chodí do školy

Setting conjunction conditions:

Setting relation conditions:
Relation: Malé --- děti
Relation: Malé --- chodí
Relation: děti --- chodí
Relation: děti --- školy
Relation: chodí --- do
Relation: chodí --- školy
Relation: do --- školy

Setting max number of relations conditions:
Marked edges:

Checking connectivity:

Components after dfs not using marked edges:
Malé --- 0
děti --- 0
chodí --- 0
do --- 0
školy --- 0

Marked edges used:
Bare sentence makes sense

Sentence makes sense.
-----------------------
```

Tabulka 3.2: Příklad výstupu

Vedle informace o smysluplnosti věty máme nyní k dispozici i podrobnější popis běhu programu. Nejprve je vypsána celá věta, poté jsou kontrolovány jednotlivé věty jednoduché. Jako první je zobrazen seznam nalezených podmínek typu 2 (Setting conjunction conditions:...). Dále jsou vypsány nalezené podmínky typu 1 (Setting relation conditions:...) a podmínky typu 3 (Setting max number of relations conditions:...). Poté je provedena kontrola

spojitosti grafu smysluplnosti - nejprve bez použití hran omezených podmínkami typu 3 (Components after dfs not using marked edges:...) a následně vybrané hrany omezené podmínkami typu 3 (viz kapitola o podmínkách typu 3).

Při následujícím spuštění programu budou do souboru `collisions.txt` uloženy všechny nalezené kolize, u kterých bylo splněno alespoň 75% klauzulí podmínky:

```
sense.exe -i input.txt -c 75% -co collisions.txt
```

V souboru `collisions.txt` pak nalezneme následující výpis kolizí. Pro každou nalezenou kolizi jsou vypsány příslušné slovní jednotky a podmínka a nakonec klauzule, na které (kterých) podmínka selhala.

```
------------------------------------------------
Collisions found for: Malé (A) --- chodí (V)
Condition:
Tag position 2:     A   ---   B i p
Tag position 4:     -   ---   -
Tag position 12:    +   ---   A
Lemma values:       +   ---   být
Token distance      <=   7

Collisions:
Token lemmas:       malý     chodit
------------------------------------------------
------------------------------------------------
Collisions found for: školy (N) --- Malé (A)
Condition:
Tag position 2:       N   ---   A U G
Tag position 2 NOT:   +   ---   C
Tag position 3:       -   ---   -
Tag position 4:       -   ---   -
Tag position 5:       -   ---   -

Collisions:
Tag position #4:        S --- P
------------------------------------------------
```

3.9 Popis formátu souboru podmínek

Pro uchování definovaných podmínek slouží soubor `conditions.txt` v adresáři s programem. Tento soubor má speciální formát, který je třeba dodržet při jeho případném rozšiřování. S aplikací je dodávána grafická utilita ConEdit, která umožňuje snadnou editaci souboru i bez znalosti jeho vnitřního formátu.

V souboru je možné psát jednořádkové komentáře uvozené //. Nezáleží na pořadí definovaných podmínek ani klauzulí v nich.

```
rel           Slovní druh 1  Slovní druh 2
[!]1             hodnota        hodnota
[!]2             hodnota        hodnota
[!]3             hodnota        hodnota
[!]4             hodnota        hodnota
...
[!]15            hodnota        hodnota
[!]lemma         hodnota        hodnota
[!]val           hodnota        hodnota
dist                #
ord               1,2            [1,2]
req                 #
[forbidden]
[!]a             hodnota        hodnota
[!]suffix        hodnota        hodnota
end
```

Tabulka 3.3: Formát podmínek (1)

Definice podmínek pro smysluplné dvojice (1):

Deklarace podmínky je uvozena klauzulí *rel*, za ní následují identifikátory *Slovní druh 1* a *Slovní druh 2* určující dvojici slovních druhů, pro které je podmínka smysluplnosti definována.

Ačkoli sémanticky na pořadí nezáleží, v souboru musí být uveden jako první slovní druh s nižším číslem. Oddělovačem je mezera nebo tabulátor. Identifikátory a čísla slovních druhů zachycuje následující tabulka:

Slovní druh	Identifikátor	Číslo	Slovní druh	Identifikátor	Číslo
Podstatné jméno	**N**	1	Příslovce	**D**	6
Přídavné jméno	**A**	2	Předložka	**R**	7
Zájmeno	**P**	3	Spojka	**J**	8
Číslovka	**C**	4	Částice	**T**	9
Sloveso	**V**	5	Citoslovce	**I**	10

Tabulka 3.4: Identifikátory a čísla slovních druhů

Morfologické klauzule

Následují podmínky pro hodnoty morfologických značek (tagů) obou slovních jednotek, na jednom řádku jedna podmínka, uvozena je číslem pozice dané morfologické kategorie (1-15), poté pro každou slovní jednotku požadovaná *hodnota*.

Hodnota může být buď jedna konkrétní hodnota dané kategorie, více možných hodnot ve tvaru (hod1 hod2 hod3) – tedy v závorkách oddělené mezerami, nebo znak + který zastupuje libovolnou hodnotu, nebo znak -, který znamená rovnost hodnotě stejné morfologické kategorie druhé slovní jednotky (pokud obě slovní jednotky mají coby hodnotu jedné kategorie -, mohou být hodnoty libovolné, ale musejí se rovnat).

Syntaktické klauzule

Je možné se odkazovat i na syntaktické informace, tedy na větné členy a na tzv. funkční suffix, který upřesňuje roli větného členu ve větě. K specifikaci požadovaných větných členů slouží klauzule *a*, která má stejný zápis jako výše uvedené klauzule pro hodnoty morfologických značek. Je tedy možné pro obě slovní jednotky požadovat konkrétní větné členy, případně některý z množiny či rovnost. Totéž platí i pro klauzuli *suffix*, která přijímá hodnoty *Co* - koordinace,*Ap* - přístavek a *Pa* - výraz v závorkách. Bližší popis hodnot v obou klauzulích je k nahlédnutí na `http://ufal.mff.cuni.cz/pdt2.0/doc/data-formats/csts/html/A.html`.

Obecné klauzule

Dále se lze odkázat na základní tvary (lemmata) obou slovních jednotek, v tomto případě je *hodnota* příslušný základní tvar (seznam základních tvarů / + / - jako v prvním případě)

Na slovosled se lze odkázat pomocí klauzule *ord* a pořadového čísla pro každou slovní jednotku (1 – slovní jednotka je první z dvojice, 2 – slovní jednotka je druhá z dvojice).

Před řádkem je možné uvést nepovinně symbol ! označující negaci. Pokud je řádek negovaný, znamená to, že pokud je podmínka určená tímto řádkem splněna, celá podmínka splněna není. Tedy například "!lemma matka +" říká, že základní tvar první slovní jednotky nesmí být "matka".

Podobně jako na základní tvary se lze odkázat i na konkrétní tvary obou slovních jednotek tak, jak jsou ve větě. K tomu slouží klauzule *val*, jejíž zápis je stejný jako klauzule *lemma*.

Maximální vzdálenost slovních jednotek ve větě lze omezit pomocí klauzule *dist*. Vzdáleností slov se rozumí absolutní hodnota rozdílu pozic slov ve větě. Pokud je tato větší než hodnota uvedená v klauzuli *dist*, podmínka neuspěje.

Pro přidání definované podmínky do některé skupiny vyžádaných podmínek (viz sekce Vyžádané podmínky) slouží klauzule *req*. Jediným parametrem je v tomto případě číslo skupiny, do které má podmínka patřit. Toto číslo může být v podstatě libovolné, pouze musí být unikátní pro každou skupinu (tedy každé dvě podmínky v téže skupině musejí mít toto číslo stejné a každé dvě podmínky v různých skupinách jej musí mít odlišné).

Pokud má být daná podmínka zakázaná (viz sekce Zakázané podmínky), stačí přidat klauzuli *forbidden* bez parametrů.

Všechny tyto klauzule jsou nepovinné, lze vytvořit i úplně prázdnou podmínku, která bude splněna pro každou dvojici slovních jednotek s odpovídajícími slovními druhy.

Definice vztahu končí řádkem uvozeným klauzulí *end*.

Příklady: Podmínku pro smysluplné spojení podstatného jména a přídavného jména, které jej rozvíjí, lze zapsat například takto:

```
rel       N           A
2         N           ( A U G )
3         -           -
4         -           -
5         -           -
a         +           atr
end
```

V této podmínce požadujeme, aby jako podstatné, tak přídavné jméno měly správný slovní poddruh (řádek 2) a měly stejný rod, číslo a pád (řádky 3,4,5). Rovněž požadujeme, aby přídavné jméno bylo přívlastkem (řádek 6).
Podmínky pro napojení předložek na podstatná jména vypadají takto:

```
rel       N           R
2         N           ( R V )
5         7           +
lemma     +           ( s za před pod nad mezi )
ord       2           1
end
```

Tato podmínka se týká předložek, které se pojí se sedmým pádem. Opět vyžadujeme správný slovní poddruh obou slovních jednotek (řádek 2), dále pak aby podstatné jméno bylo v sedmém pádu (řádek 3), aby se jednalo o jednu z předložek, které se pojí se sedmým pádem (řádek 4 - specifikujeme možné základní tvary předložky) a nakonec aby ve větě předložka předcházela podstatnému jménu (řádek 5).

Příklad zakázané podmínky, postihující nesprávný výraz “k jemu” vypadá takto:

```
rel        P            R
val        jemu         k
ord        2            1
dist       1
forbidden
end
```

Zde požadujeme konkrétní hodnoty obou slovních jednotek ve větě (řádek 2), dále aby předložka byla ve větě před zájmenem (řádek 3) a aby obě slovní jednotky byly bezprostředně u sebe (řádek 4 - specifikujeme maximální vzdálenost = 1). Nakonec klauzulí `forbidden` specifikujeme, že se jedná o zakázanou podmínku.

```
con          Slovní druh 1  Slovní druh 2
[!]1         hodnota        hodnota
[!]2         hodnota        hodnota
[!]3         hodnota        hodnota
[!]4         hodnota        hodnota
...
[!]15        hodnota        hodnota
[!]lemma     hodnota        hodnota
[!]val       hodnota        hodnota
dist         #
ord          1,2            [1,2]
req          #
[forbidden]
[!]a         hodnota        hodnota
[!]suffix    hodnota        hodnota
[!]tok       hodnota
end
```

Tabulka 3.5: Formát podmínek (2)

Definice podmínek pro spojování částí věty (2):
Podmínka říká, kdy můžou být slovní jednotky se slovními druhy *Slovní druh 1* a *Slovní druh 2* spojeny spojkou nebo čárkou, tedy vztah je ternární, ale syntax má téměř stejnou jako předchozí binární. Všechny klauzule mají stejný význam jako u podmínek typu (1), navíc je jen klauzule *tok*, za níž následuje základní tvar spojky, pokud musí být nějaký konkrétní.

Podmínka je aplikována tak, že pokud je ve větě nalezena spojka nebo čárka, najde se k ní nejbližší pár slovních jednotek (jeden zleva, jeden zprava), který vyhovuje některé z takto definovaných podmínek. Další slovní jednotky již tato konkrétní spojka (čárka) neváže.

Příklady: Takto může vypadat podmínka typu (2) pro spojení dvou podstatných jmen spojkou nebo čárkou:

```
con       N           N
5         -           -
suffix    co          co
end
```

Požadujeme tedy, aby podstatná jména měla stejný pád a (pokud je tato informace k dispozici) aby byly součástí koordinace.

Další příklad popisuje podmínku pro zapojení spojky "jako" spojující sloveso a podstatné jméno:

```
con       N           V
ord       2           1
tok       jako
end
```

Zde chceme, aby sloveso ve větě předcházelo podstatné jméno (řádek 2) a aby se jednalo o spojku "jako" (řádek 3). Takto definované podmínce vyhoví například spojení "jede jako blesk".

Definice podmínek pro omezení počtu relací (3):

Tato podmínka je na jediný řádek. Uvozena klauzulí `max`, u následující dvojice identifikátorů slovních druhů záleží na pořadí (sémanticky i syntakticky), `maximum` je číslo větší nebo rovné 1. Takováto podmínka má význam "Jedna konkrétní slovní jednotka se slovním druhem `slovní druh 1` se může navázat nejvýše na `maximum` různých slovních jednotek se slovním druhem `slovní druh 2`." Navázat znamená vytvořit hranu v konstruovaném grafu.

```
max  Slovní druh 1  Slovní druh 2  maximum
```

Tabulka 3.6: Formát podmínek (3)

Příklady: Následující podmínka říká, že každé přídavné jméno se může navázat nejvýše na jedno podstatné jméno. To například způsobí, že při zkoumání věty "Rodinný dům a zelený trávník" vznikne o dvě relace méně ("rodinný trávník" a "zelený dům") a analýza bude tedy přesnější,

```
max      A      N      1
```

Podobně je možné požadovat, aby se každé příslovce navázalo nejvýše na jedno sloveso:

```
max      D      V      1
```

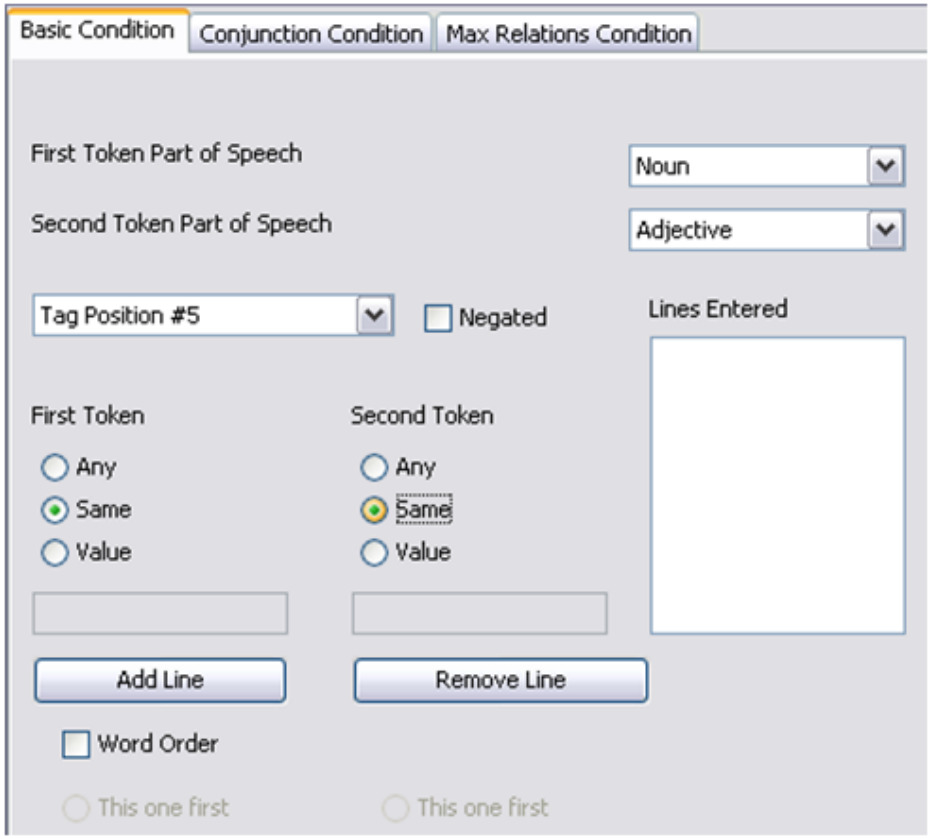
Při definici podmínek (3) je třeba dbát na pořadí slovních druhů, poslední podmínka například neříká nic o tom, na kolik příslovcí se může navázat jedno sloveso.

3.10 Utilita pro úpravu podmínek

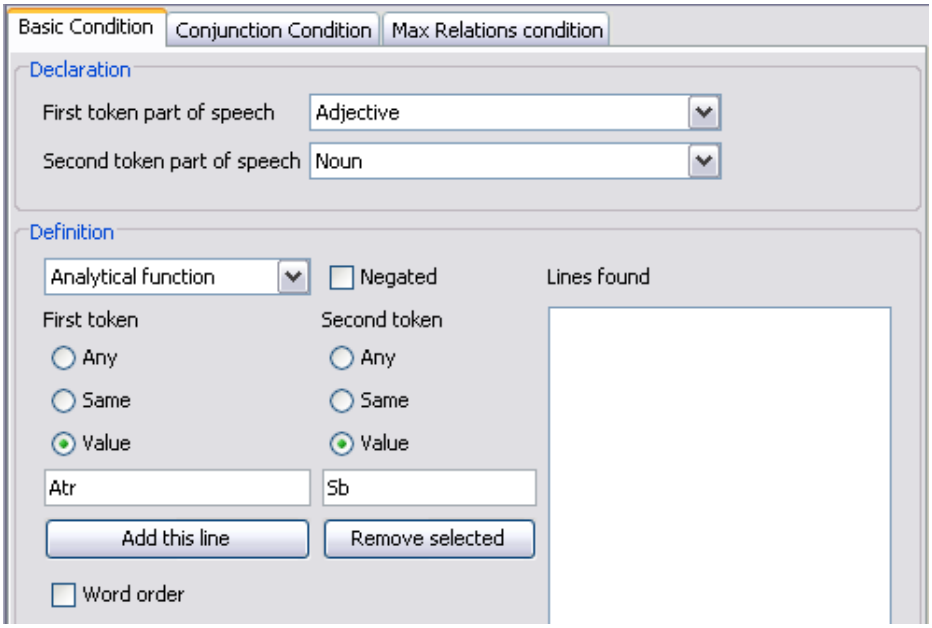
Pro usnadnění editace souboru podmínek slouží utilita *ConEdit* přiložená v instalačním balíčku. Poskytuje jednoduché grafické rozhraní, které umožňuje správu souboru s podmínkami bez jakékoli znalosti jeho vnitřního formátu. Utilita je napsána v jazyku Java, je tedy multiplatformní a její instalace i spouštění je stejné na OS Windows i Unix – stačí rozbalit obsah adresáře ConEdit do libovolného umístění a spustit soubor *ConEdit.jar*. Jediným softwarovým požadavkem je nainstalovaná Java Virtual Machine.

Po spuštění je třeba nejprve otevřít soubor s podmínkami (volba *file− >open*), následně budou načteny všechny uložené podmínky. Po kliknutí na kteroukoli z nich je možné ji editovat nebo smazat, případně je možné přidat novou podmínku.

Program rovněž umožňuje filtrování a třídění podmínek podle typu a aktualizaci souboru podmínek po internetu.



Obrázek 3.2: Utilita ConEdit - morfologická klauzule



Obrázek 3.3: Utilita ConEdit - syntaktická klauzule

3.11 Evaluace

Program byl testován na čtyřech sadách dat. První byla data poskytnutá portálem Seznam.cz, dále byl program testován na datech získaných z Českého akademického korpusu 1.0, jako třetí byla testovací sada vět pro účely evaluace ročníkového projektu (který pracoval pouze s morfologickými informacemi) a konečně čtvrtá testovací sada vět byla určena k evaluaci konečného stavu programu (za použití syntaktických klauzulí).

Každá sada obsahovala dva soubory - soubor smysluplných vět a soubor nesmyslných vět. Výsledky evaluace zachycuje následující tabulka. V levé části jsou výsledky dosažené ve fázi ročníkového projektu (pouze morfologické informace), v pravé pak výsledky dosažené finální verzí programu (morfologické a syntaktické informace). Pro každou sadu vět je v ní dále uvedena procentuální úspěšnost na smysluplných a nesmyslných větách.

	Ročníkový projekt		Bakalářská práce	
Data	smysluplné	nesmyslné	smysluplné	nesmyslné
Seznam.cz	80%	90%	78%	90%
ČAK 1.0	80%	40%	80%	50%
testovací věty 1	85%	8%	80%	60%
testovací věty 2	68%	62%	17%	40%

Tabulka 3.7: Evaluace

Nejlepších výsledků bylo dosaženo na datech poskytnutých portálem Seznam.cz. Jednalo se o typické řetězce, o jejichž smysluplnosti je třeba rozhodovat při prohledávání internetových stránek. Nesmyslné věty bylo možné poměrně snadno rozpoznat - jednalo se o různé seznamy, formuláře, úseky HTML kódu a podobně. Příklad nesmyslné věty z této sady: *Úvod Aktuality Preview Kino DVD Trailery.* Zde postačuje k rozeznání nesmyslnosti morfologická informace.

V případě dat z ČAK a testovacích vět došlo k poklesu úspěšnosti u testování nesmyslných vět. Ty byly v prvním případě vytvořeny z vět smysluplných vypuštěním určitých slovních jednotek (nejčastěji čísel). Proto bylo obtížnější je odlišit od vět smysluplných, informace morfologické a syntaktické vrstvy často nepostačují k tomu, aby podobné věty byly rozeznány coby nesmyslné.

Situaci vystihuje typická věta "Druhá světová válka se odehrála před rokem." Ve větě chybí číslo udávající rok, bude ale programem vyhodnocena jako smysluplná. Na morfologické a syntaktické úrovni je velmi obtížné zjistit, že ve větě chybí letopočet.

V první testovací sadě vět byly zejména věty používané pro evaluaci systému pro kontrolu gramatiky. Nižší úspěšnost byla způsobena tím, že v době testování pracoval systém pouze s morfologickou informací, nikoli se syntaktickou. V současné době za použití syntaktické informace a zakázaných podmínek je úspěšnost na tomto souboru nesmyslných vět zhruba 60%.

Příklad nesmyslné věty z tohoto souboru: *"K tě se tato informace asi nedostala."* Spojení "k tě" je typický příklad jevu, který je vždy nesmyslný, lze jej proto postihnout zakázanou podmínkou.

Druhá testovací sada byla podobná první, s tím rozdílem, že některé věty ve druhé testovací sadě by k rozpoznání smysluplnosti vyžadovaly informace sémantické vrstvy. Příklad: *"Pojedu-li do Londýna, byl bych býval jedl brambory."* Tyto věty je obtížné klasifikovat na úrovní syntaktické vrstvy, neboť jsou syntakticky správně, nesmyslnost je ve významu. V mnoha případech také nebyla nesmyslnost rozpoznána z důvodu valence (se kterou algoritmus nepracuje) - příklad: *"Bál se důsledkům."* Algoritmus nemá k dispozici informaci o tom, že sloveso *"bát se"* se pojí s druhým, nikoli se třetím pádem.

Obecným problémem se ukázala být úspěšnost *taggeru* - pokud je některému slovu chybně přiřazena morfologická informace, snadno to vyústí v chybnou klasifikaci věty. Příklad: ve (smysluplné) větě *"Nikdy se nemůžete vrátit k drogám na stálo"* je slovo *"stálo"* označeno za sloveso, proto se nenaváže na předložku *"na"* a věta bude vyhodnocena jako nesmyslná.

U některých vět také narážíme na problém subjektivní definice pojmu smysluplnost, jedna z nesmyslných vět byla například: *"Nová silnice točí se z jedné strany na druhou."* Různí lidé mohou mít různé názory na její smysluplnost, algoritmem byla tato věta vyhodnocena jako smysluplná.

Literatura

[1] Barbora Vidová Hladká a kol.: *The Czech Academic Corpus 2.0 - Guide.*, Prague Bulletin of Mathematical Linguistics 89, pp. 41–96, 2008.

[2] *Czech Academic Corpus 2.0* , `http://ufal.mff.cuni.cz/rest/CAC/cac_20.html`.