# Uživatelská dokumentace k ročníkovému projektu

Vladimír Rovenský

15. února 2009

| | |
|---|---|
| Vedoucí projektu: | Barbora Vidová-Hladká |
| Téma: | Rozeznání smysluplnosti české věty |
| Platformy: | Unix, Windows |
| Programovací jazyk: | C++ (IDE MS Visual Studio) |
| HW požadavky: | žádné speciální |
| SW požadavky: | OS Unix / Linux / Windows, tool_chain |

# 1 Úvod

## 1.1 Anotace

Cílem ročníkového projektu je implementace automatické procedury, která v reálném čase rozezná, je-li shluk textových řetězců českou větou. Automatická procedura bude využívat výstupy externích automatických modulů, které se týkají tvarosloví (slovní druhy a jejich kategorie - rod, číslo, pád, osoba aj.). Vyřešení tohoto úkolu je využitelné ve fulltextovém vyhledávání, kde je mj. důležité rozeznat shluk klíčových slov určených pro zmatení vyhledávače od věty určené uživateli.

## 1.2 Pojem smusluplnost

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 ("Děti chodí do školy."), nebo zjevně nesmyslné ("Ráno zcela napříč našel i."). Můžeme ale také nalézt příklady vět, na jejichž smysluplnost mají různí lidé různé názory:

| | |
|---|---|
| Tatínek šli do školy. | Může se jednat o překlep, nebo o odpověď na otázku "Kdo co kam nese?" |
| 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.

## 1.3 Stručný popis programu

Jedná se o nástroj ke zjišťování smysluplnosti české věty na základě tvaroslovných (morfologických) informací. Vstupem programu je řetězec českých slov zpracovaný nástrojem tool_chain, který poskytne potřebné tvaroslovné informace. Výstupem programu je informace o tom, je-li či není vstupní řetězec smysluplnou českou větou. Větou může být i otázka.

## 1.4 Instalace programu pod OS Windows

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

## 1.5 instalace programu pod OS Unix

Nejprve do libovolného adresáře rozbalte soubory v adresáři source/sense z instalačního balíčku *Sense*. Dále je třeba standardním způsobem tyto zdrojové kódy zkompilovat (například spuštěním příkazu `"g++ ‘ls~./*.cpp‘ -w -o sense"` v adresáři, kam byly rozbaleny).

## 1.6 Spuštění programu

Samotné spuštění je stejné pod Windows i Unixem. V adresáři, kam byl program nainstalován, najdete dva soubory: *conditions.txt* a *sense.exe* (respektive *sense* v OS Unix / Linux). V prvním jsou uložené vztahy, 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:

| | |
|---|---|
| *-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 a tagováním nástroje tool_chain. Výchozí hodnotou je standardní vstup. |
| *-o soubor* | Specifikace souboru, do nějž má být ukládán výstup programu. Výchozí hodnotou je standardní výstup. |
| *-h* | Zobrazí krátkou nápovědu. |
| *-v* | Zapnutí podrobného výstupu, program vypíše nalezené vztahy (viz dále). |
| *-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 říká, kolik klauzulí podmínky musí být splněno, aby byly ostatní - nesplněné - řádky vypsány coby kolize. Za číslem lze napsat symbol '%', potom bude zadaná hodnota chápána jako procentuální podíl splněných klauzulí na celkovém počtu klauzulí. |
| *-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 prakticky 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). |
| *-co soubor* | Určení souboru, do kterého mají být ukládány nalezené kolize (viz volba *-c*. Výchozí hodnotou je standardní výstup. |

## 1.7 Soubor s nastavením

Po prvním spuštění programu je automaticky vygenerován 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.

## 1.8 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 analýzu českých textů, pro potřeby dokumentovaného programu jsou důležité pouze služby tokenizace, morfologická analýza a tagování, ze kterých analýza smysluplnosti vychází. Pro analýzu vstupní věty nástrojem tool_chain je třeba spustit jej následovně: `tool_chain -tAT -i soubor_s_větou -o soubor_s_výstupem`
Volba *–tAT* zajistí, že se provedou všechny tři 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.

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`.
Program *sense* očekává na vstupu soubor právě v tomto formátu.

# 2 Popis algoritmu

## 2.1 Načtení vstupu

Na vstupu máme nástrojem tool_chain analyzovanou větu ve formátu CSTS. Takto může vypadat příklad vstupu programu po analýze nástrojem tool_chain (tokenizace, morfologická analýza, tagování, tedy volba *-tAT*). Vstupní větou byl řetězec “Před domem stojí auto a v domě štěká pes.”

```
<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="in:001-p1s1">
<f cap>před<MDl src="m">před<MDt src="m">R--------------
<f>domem<MDl src="m">dům<MDt src="m">NNIS7-----A----
<f>stojí<MDl src="m">stát-4<MDt src="m">VB-S---3P-AA---
<f>auto<MDl src="m">auto<MDt src="m">NNNS4-----A----
<f>a<MDl src="m">a-1<MDt src="m">J^-------------
<f>v<MDl src="m">v-1<MDt src="m">RR--6----------
<f>domě<MDl src="m">dům<MDt src="m">NNIS6-----A----
<f>štěká<MDl src="m">štěkat<MDt src="m">VB-S---3P-AA---
<f>pes<MDl src="m">pes<MDt src="m">NNMS1-----A----
<D>
<d>.<MDl src="m">.<MDt src="m">Z:-------------
</c>
</doc>
</csts>
```

Takovýto vstup je rozdělen na jednotlivé slovní jednotky (neboli tokeny – nejčastěji jedno slovo nebo interpunkční znaménko) a ty jsou uloženy do příslušných struktur spolu se svou morfologickou informací, která je reprezentována jako patnáctipoziční značka (tag) pro každou slovní jednotku. 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`.
Z CSTS výstupu nástroje tool_chain jsou pro aplikaci *Sense* relevantní především značky
`<f>` (vlastní slovní jednotka), `<d>` (interpunkce), `<MDl>` (základní tvar) a `<MDt>` (morfologický tag).

## 2.2 Dělení souvětí na věty jednoduché

Pokud je na vstupu souvětí, je nejprve rozděleno na věty jednoduché. Celé souvětí je 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.

### 2.2.1 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.

**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").

## 2.3 Algoritmus zjišťování smysluplnosti jednoduché věty

### 2.3.1 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 podmínky, za kterých je tato smysluplná. Těchto podmínek může 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 tyto podmínky označovány jako *podmínky typu (1)*.

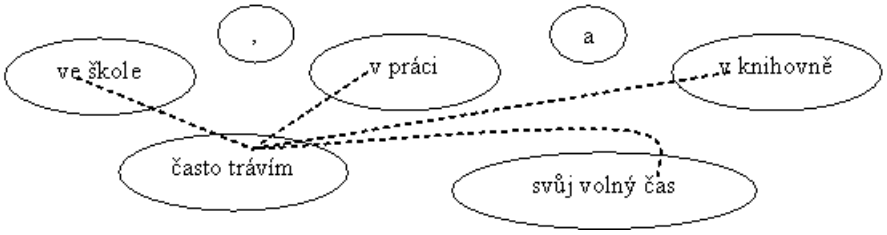
**Příklad:** Dvojice podstatné a přídavné jméno je smysluplná, pokud mají stejný rod, pád i číslo. U dvojice podstatné jméno a předložka můžeme pro smysluplnost požadovat, aby předložka ve větě předcházela podstatné jméno. O každé předložce navíc víme, se kterými pády podstatného jména se může vázat, 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. |

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):




Čárkovaně jsou vyznačeny hrany odpovídající spojením, která jsou natolik obecná, že budou omezena dalšími typy podmínek (výše zmíněná spojení předložky a slovesa nebo slovesa a podstatného jména). Základem jsou tedy podmínky a jejich definice. K tomuto účelu slouží soubor *conditions.txt* v adresáři s programem. V něm jsou veškeré tyto podmínky definovány a do něj je možné další doplňovat, jeho formát bude popsán dále.

Takto definované podmínky ale neposkytují dostatečně silný prostředek k rozpoznání nesmysluplnosti věty, neboť například spojení slovesa a podstatného jména je natolik obecné, že je podmínka prakticky prázdná. Potom stačí, aby v každé komponentě grafu bylo jedno podstatné jméno a v celé větě jediné sloveso a graf bude souvislý, bez ohledu na smysluplnost věty. Proto lze definovat další typy podmínek, které smysluplná věta musí splňovat.

### 2.3.2 Podmínky pro spojování částí věty

Výše definované podmínky dokáží vcelku rozumně rozložit (implicitně) větu na několik úseků, například rozvité podstatné jméno, určení místa atp. Problémem je rozumné spojení těchto úseků,

viz výše zmíněný problém se slovesem a podstatným jménem a rozbor ilustrační věty. K tomuto účelu je slouží druhý druh podmínek – 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.

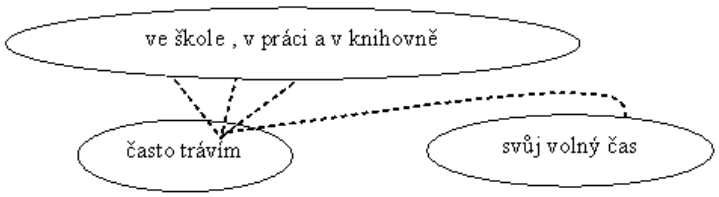
**Příklad:** Vraťme se k rozpracované větě "Ve škole, v práci a v knihovně často trávím svůj volný čas." Při hledání trojic splňujících některou podmínku typu (2) budou nalezena spojení: "škole , práci" + "práci a knihovně" – v obou případech se jedná o spojení podstatných jmen ve stejném pádě, jednou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

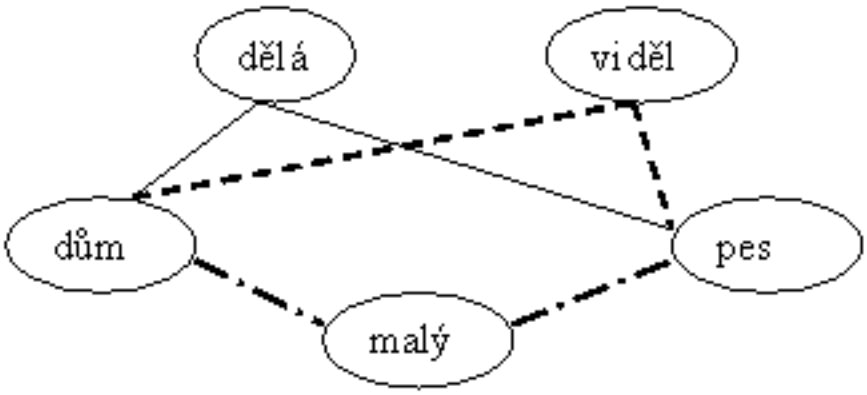



### 2.3.3 Podmínky pro maximální počet vztahů

Dalším prostředkem pro zpřísnění analýzy jsou podmínky pro omezení počtu vztahů. Pokud není počet vztahů nijak omezen, nastává problém s rozpoznáváním nesmyslných vět, například podmínka spojující částici / citoslovce se slovesem opět nemá prakticky žádné morfologické informace, které by se daly kontrolovat, tedy jakékoli citoslovce se naváže na všechna slovesa věty bez omezení. Tato podmínka poskytuje prostředek k vyjádření podmínek typu "jedna částice se může navázat nejvýše na jedno sloveso" atp. V tomto případě se jedná o nesymetrickou relaci.

**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):




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 (bez ohledu na to, jaké hrany vybereme - nejvýše lze vybrat tři, ale kostra grafu na pěti vrcholech má čtyři hrany).

### 2.3.4 Popis formátu souboru podmínek

Soubor s podmínkami má speciální formát, který je třeba dodržet při jeho případném rozšiřování. Ve finální verzi bude nejspíše s programem dodávána i k tomu určená utilita. V souboru je možné psát jednořádkové komentáře uvozené //. Nezáleží na pořadí definovaných podmínek.

**Definice podmínek pro smysluplné dvojice (1):**

| *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]* | hodnota | |
| *[ord]* | 1,2 | [1,2] |
| *end* | | |

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:

**Identifikátory a čísla slovních druhů:**

| 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 |

Následují podmínky pro hodnoty 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). *Hodnota* také nemusí být vyplněna vůbec, v tom případě je význam stejný jako +.

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*. Vzdálenosti slov se rozumí absolutní hodnota rozdílu pozic slov ve větě. Pokud je tato větší než hodnota uvedená v klauzuli *dist*, podmínka neuspěje.

Prakticky všechny tyto podmínky 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*.

**Definice podmínek pro spojování částí věty (2):**

| *rel* | *Slovní druh 1* | *Slovní druh 2* |
|---|---|---|
| *[!][1]* | [hodnota] | [hodnota] |
| *[!][2]* | [hodnota] | [hodnota] |
| *[!][3]* | [hodnota] | [hodnota] |
| *[!][4]* | [hodnota] | [hodnota] |
| ... | | |
| *[!][15]* | [hodnota] | [hodnota] |
| *[!][tok]* | [hodnota] | |
| *end* | | |

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. Hodnoty tagů jednotlivých slovních jednotek se definují stejně. Za klauzulí tok 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.

**Definice podmínek pro omezení počtu relací (3):**

*max* *Slovní druh 1* *Slovní druh 2* *maximum*

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.

Vynucování této podmínky je realizováno tak, že pokud ze všech smysluplných vztahů lze vybrat alespoň jednu podmnožinu splňující podmínku (3), stačí to pro smysluplnost věty. Proto je její ověřování náročnější než u ostatních podmínek, v současné verzi program s jistou optimalizací možnosti generuje.

## 2.4 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 $O(n)$.

Poslední 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.

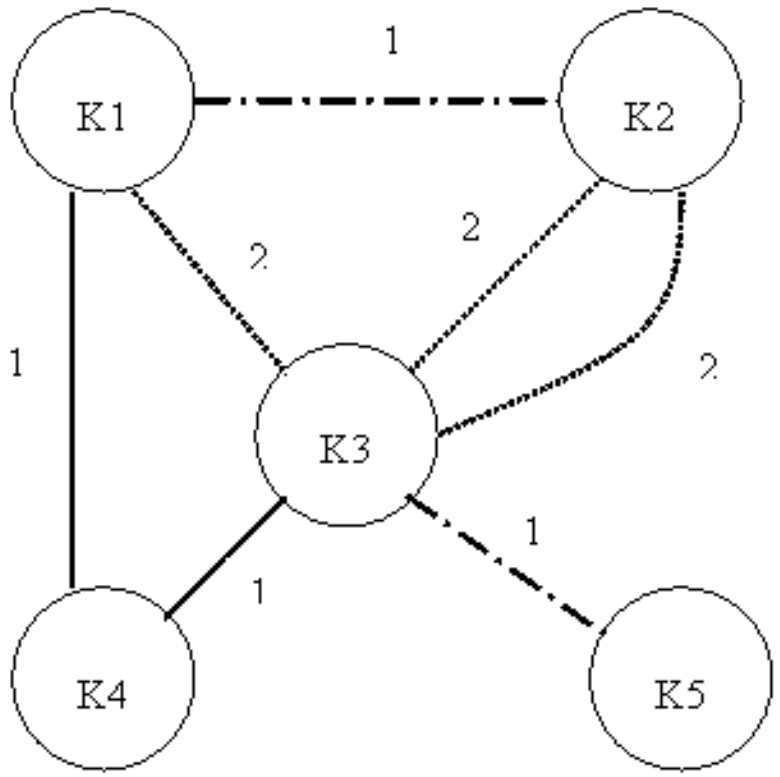
Celková složitost tedy vychází v nejhorším případě $O(s) + O(n^2)$ kde $s$ je délka souboru s podmínkami a $n$ je počet slovních jednotek věty.

## 2.5 Shrnutí algoritmu

Následuje stručné shrnutí celého použitého algoritmu

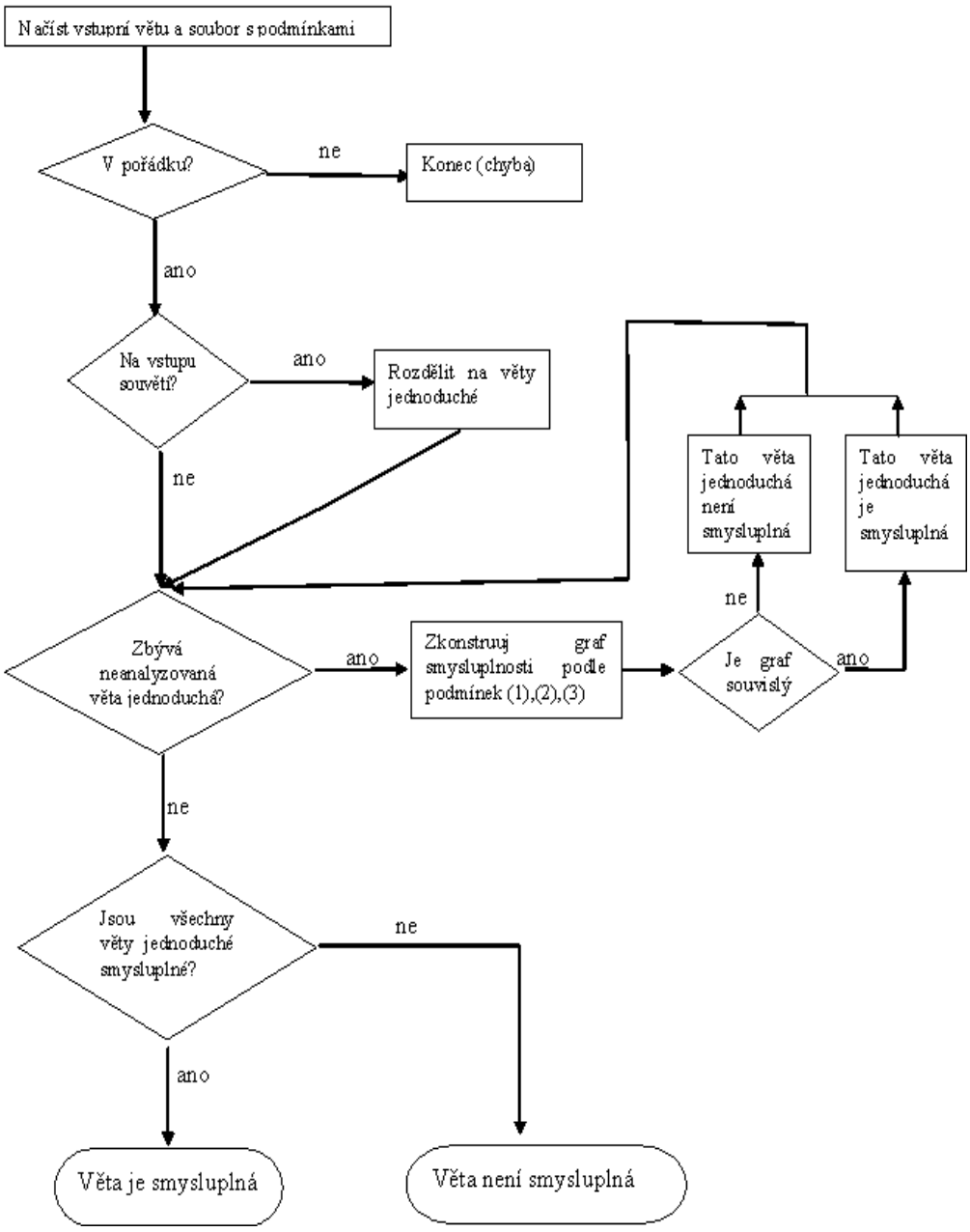
1. Načíst vstupní větu a soubor s podmínkami.
2. Pokud je na vstupu souvětí, je rozděleno na věty jednoduché. Celá věta je smysluplná právě když jsou smysluplné všechny její věty jednoduché.
3. Pro každou větu jednoduchou zkonstruovat "graf smysluplnosti", vrcholy = slovní jednotky věty a hrany přidávány 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)
4. Zkontroluje se souvislost vzniklého grafu, věta je smysluplná, právě když je graf souvislý. Zároveň je třeba kontrolovat platnost podmínek (3).
   (a) Provést DFS [1] 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). To znamená, že nemusí být možné je použít všechny, proto program hledá způsob, jakým vybrat podmnožinu těchto hran, která všechny komponenty spojí do jedné a zároveň neporuší žádnou z podmínek (3). Pokud takovou najde, věta (jednoduchá, ne celé souvětí) je prohlášena za smysluplnou. Situaci popisuje obrázek.

[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




K1 – K4 jsou komponenty vzniklé DFS průchodem po hranách 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.

**Vývojový diagram**




## 2.6 Evaluace

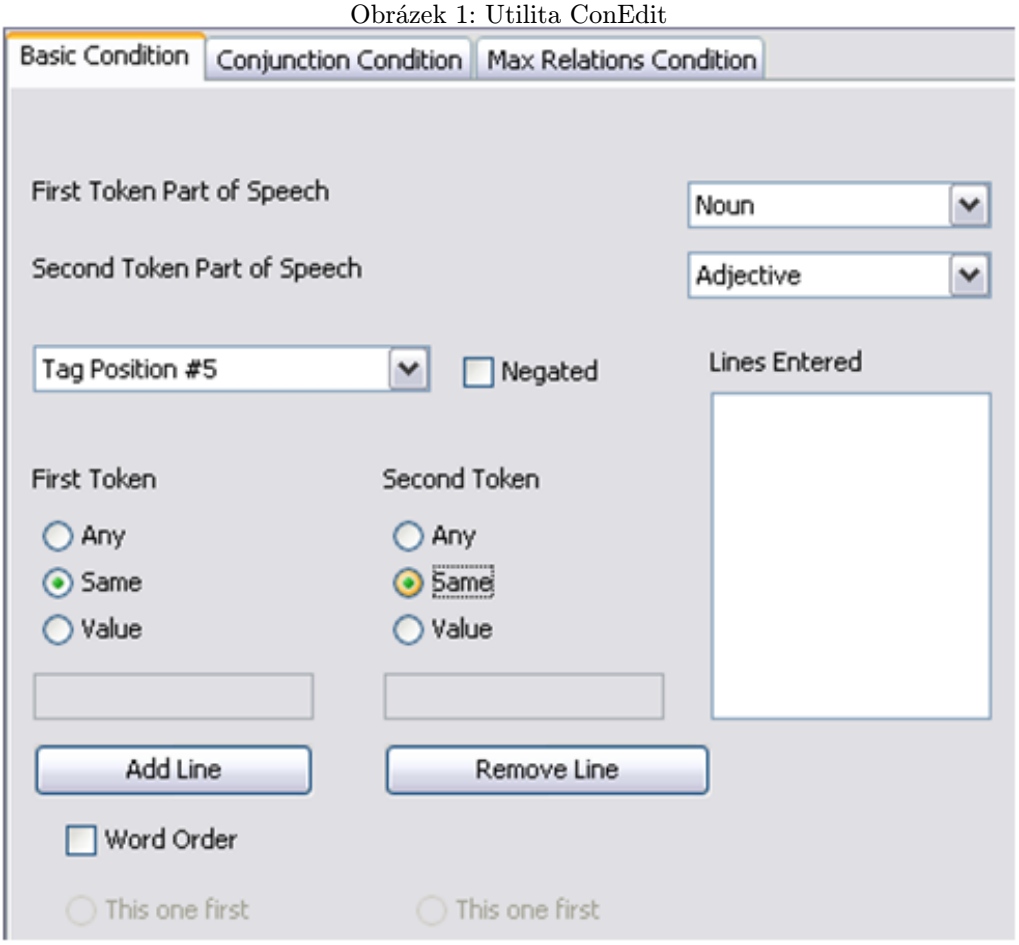
Program byl testován na třech sadách dat. První byla data poskytnutá portálem Seznam.cz, na smysluplných větách byla úspěšnost programu zhruba 85%, na nesmyslných 90%. Dále byl program testován na datech získaných z Českého akademického korpusu 1.0 – na smysluplných větách zde byla úspěšnost programu zhruba 80%, na nesmyslných potom 40%. Nižší úspěšnost je zčásti dána tím, že nesmyslné věty byly v tomto případě získány ze smysluplných pouhým vypuštěním některých slovních jednotek.
Třetí byla testovací sada dvaceti smysluplných a zhruba 100 nesmyslných vět. U smysluplných byla úspěšnost programu 85 %, u nesmyslných 8%.

## 2.7 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.

Obrázek 1: Utilita ConEdit



# Obsah



# Literatura

[1] *Barbora Vidová Hladká a kol., Český akademický korpus 2.0, v tisku*
**Popis nástroje tool_chain**
`http://ufal.mff.cuni.cz/rest/CAC/cac_20.html`

[2] **Popis morfologických značek:**
`http://ufal.mff.cuni.cz/rest/CAC/doc-cac20/cac-guide/cz/html/ch13.html`