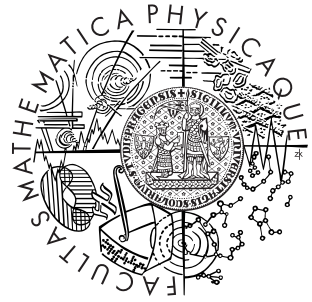
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

# BAKALÁŘSKÁ PRÁCE




David Mareček

## Novelizátor zákonů

Ústav formální a aplikované lingvistiky

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

2006

Chci poděkovat RNDr. Danovi Zemanovi Ph.D. za trpělivost, hodnotné rady a odborné vedení mé 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 24. 5. 2006 David Mareček

# Obsah

Název práce: Novelizátor zákonů
Autor: David Mareček
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí bakalářské práce: RNDr. Daniel Zeman, Ph.D.
e-mail vedoucího: zeman@ufal.mff.cuni.cz

Abstrakt: Práce se zabývá automatickou novelizací zákonů. Novely jsou pro normálního člověka nečitelné a neposkytují přehled o platném znění zákona. Úplné znění zákona se přitom vydává vždy až po několika novelách. Novely obsahují konstrukce typu „na konci písmena se tečka nahrazuje čárkou a doplňují se slova ...“, „za druhou větu odstavce se vkládá věta ...“, „paragraf 12 zní ...“ atd. Hlavním předmětem práce je vytvoření programu, který přečte zadané novely zákonů a aktualizuje znění těch předpisů, které jsou těmito novelami změněny. Dále bylo vybráno několik zákonů, na kterých byl program důkladně otestován. Při vyhodnocování jednotlivých vět se podařilo dosáhnout více jak 95% úspěšnosti.

Klíčová slova: novela zákona, úplné znění, zpracování přirozeného jazyka

Title: Automatic processing of law amendments
Author: David Mareček
Department: Institute of Formal and Applied Linguistics
Supervisor: RNDr. Daniel Zeman, Ph.D.
Supervisor's e-mail address: zeman@ufal.mff.cuni.cz

Abstract: Law amendments are unreadable for normal people and do not provide the actual wording of the law. The complete versions of laws are usually published after several amendments, if at all. The amendments often contain sentences like "the dot is substituted by comma at the end of the paragraph, and words ... are added", "after the second sentence, insert the sentence ...", "the new wording of the paragraph number 12 is ...". The main goal of this work is to create an application that will be able to read a given law together with its amendments and create the complete version of the law. A few laws were chosen to test the application, and an accuracy of 95% has been achieved.

Keywords: law amendment, actual wording of a law, natural language processing

# 1 Úvod

Novely zákonů jsou zákony, kterými se mění jiné, dříve vydané zákony. Zpravidla jsou děleny na části, kde každá odpovídá změně jednoho konkrétního zákona, ty jsou dále děleny na body. Bod obsahuje většinou jednu, někdy i více vět a přesně popisuje jednu změnu v příslušném zákoně. Novely je možné si prohlédnout ve Sbírce zákonů například na stránkách ministerstva vnitra (*http://www.mvcr.cz/sbirka*).

Cílem práce je analyzovat jazyk, který je v novelách používán, a vytvořit aplikaci, která mu porozumí a změny správně v příslušném předpise provede. Jde o přirozený jazyk, nikdy tedy nedokážeme dosáhnout úspěšnosti 100%. Budeme se ale snažit k ní co nejvíce přiblížit. Zdánlivě vypadá problém jednoduše: V novelách se vyskytuje pouze několik větných konstrukcí, které se stále opakují. Podrobnějším studováním ale zjistíme, že i na tak omezené množině slov, která je zde používána, se dají tvořit velmi rozmanité věty. Vyskytují se tady poměrně rozvětvená příslovečná určení místa, časté přívlastky a vícenásobné nebo naopak nevyjádřené větné členy.

Výsledný program poběží na příkazové řádce a bude pracovat s předpisy v pevně daném formátu. Neklade si za cíl být „user friendly“, spíše by mohl být použit jako jádro pro další okenní aplikace, které se budou novelizacemi zabývat.

# 2 Problematika zpracovávání předpisů

## 2.1 Získávání dat

K počítačovému zpracování textu je zapotřebí kvalitního a bezchybného vstupu. V našem případě nestačí pouze prostý textový soubor, potřebujeme ještě některé informace o formátování příslušného předpisu. Zákony České republiky se dělí na části, paragrafy, odstavce apod. Jsou v nich nadpisy, tabulky, obrázky a odkazy. Tohle všechno musíme zachytit. Skenování a rozpoznávání textu vydávaných částek tedy můžeme rovnou zamítnout. Jednotlivé částky jsou rovněž k dispozici na stránkách ministerstva vnitra (*http://www.mvcr.cz/sbirka*) ve formátu PDF. Ukázalo se ovšem, že tyto soubory na tom nejsou o moc lépe. Existují sice konvertory do čistě textového souboru nebo do HTML, žádná konverze těchto souborů však nepřinesla takové výsledky, se kterými by se dalo dále nějak reálně pracovat, pokud bychom předpis neprošli a ručně neopravili všechny chyby.

Je třeba se tedy poohlédnout na internetu po jiných podobách předpisů. Existuje několik portálů, které vystavují sbírku zákonů v elektronické podobě, některé pouze v TXT, některé v HTML. Ne všechny ji ovšem poskytují zdarma a ne všechny bez chyb. U některých je přímo vidět, že předpisy skenovali, jsou tam třeba zaměněna písmena „l“ s čísly „1“. To běžnému čtenáři nevadí, ale program takovou větu zkrátka nerozezná. Některé servery mají omezený počet přístupů a po stažení třeba dvou kompletních předpisů blokují IP adresu. Nejlépe z tohoto výběru vyšly dva servery, a to Portál veřejné správy (*http://portal.gov.cz*) a Zákony na webu (*http://www.sbcr.cz*). Oba poskytují zákony v HTML a jsou navíc pěkně formátované, ze zdrojových kódů lze snadno zjistit, kde začíná a končí odstavec, paragraf, co je nadpis apod. Na portálu veřejné správy se vyskytují aktuální znění předpisů. Od dubna 2006 se vyskytují pouze aktuální znění i na druhém zmiňovaném serveru, takže je tu opět problém, kde stahovat původní znění zákonů.

Nevýhodou u všech internetových portálů je jejich nedůvěryhodnost, zároveň se můžou kdykoli rozhodnout formát souborů změnit, nebo vůbec celou službu zrušit. A aby toho nebylo málo, jsou i v originálních zněních předpisů chyby, kterých si člověk třeba ani nevšimne, ale počítači nadělají mnoho problémů. Občas chybí například mezera nebo čárka, někdy chybějí uvozovky. Získávání přesných znění zákonů je tedy značně problematické a nedá se provést čistě automaticky bez drobných manuálních oprav.

## 2.2 Jednotný formát pro ukládání předpisů

Bude zapotřebí vytvořit formát předpisů, se kterým bude náš program pracovat. Měl by být jednoduchý, přehledný a zároveň jednoznačný a měl by zahrnovat všechny potřebné informace. Všechny předpisy z webu se vždy nejdříve jiným pomocným programem zkonvertují do tohoto formátu, stejně tak se budou konvertovat výstupy. U výstupů je to jednoduché, z námi vytvořeného formátu snadno vygenerujeme soubor HTML, PDF, nebo jiné, a to s nulovou chybovostí. U vstupů dostupných na webu záleží na jejich kvalitě. Potřebujeme mít úplnou informaci o tom, kde třeba začíná paragraf, kde končí, co přesně je jeho nadpis, co je jeho označení, co jsou poznámky pod čarou a co jsou odkazy na ně, stejně tak potřebujeme rozlišovat části, díly, oddíly, hlavy, články, odstavce, písmena, body, položky a další.

Portály zmiňované v předchozí kapitole se k tomuto účelu hodí, ve zdrojových kódech používají například značení `<div class="paragraf">` pro začátek paragrafu nebo `<div class="paragraf-nadpis">` pro jeho nadpis, všechny bloky jsou vždy ukončeny tagem `</div>`. Tímto způsobem je označeno téměř všechno, co potřebujeme, chybí pouze třeba označení odstavců nebo písmen, to lze ale snadno automaticky doplnit. Například označení odstavce (třeba „(5)“) je prostě první skupina znaků v odstavci, pokud je v závorkách.

Dalším problémem jsou uvozovky. Obzvláště v novelách zákonů se používají uvozovky poměrně často a někdy jsou i do sebe vnořené. Je tam třeba napsáno, že nějaká věta se nahrazuje větou jinou, přičemž obě dvě věty jsou v uvozovkách. Budeme potřebovat pro každé uvozovky zjistit, které jsou k nim párové. A to se s nepárovými ASCII uvozovkami zjišťuje těžko. Vzhledem k tomu, že počáteční i koncové uvozovky jsou tentýž znak, nedokážeme rozlišit například jedny uvozovky vnořené do druhých od dvou textů v uvozovkách vedle sebe. V našem formátu tedy budeme používat české horní a dolní uvozovky („ a “). Tím se vyřešil i problém kódování souboru. Kódování ISO 8859-2 (Latin 2) neobsahuje české uvozovky, budeme tedy používat kódování CP1250 (Windows-1250). Na webu se ale bohužel většinou používají nepárové uvozovky. Bude tedy třeba se rozhodovat, které odpovídají horním a které dolním. Před dolními obvykle bývá mezera a před horními ne, nelze se ale na to vždy spolehnout.

Každý předpis bude uložen v prostém textovém souboru, do kterého budou navíc přidány tzv. značky. Jak budou tyto značky vypadat? Budeme potřebovat označovat začátky a konce jednotlivých bloků (paragrafů, odstavců, jejich nadpisů a označení atd.) K tomu se hodí značkování podobné jako v jazycích HTML nebo XML. Jednotlivé značky (tagy) budou uzavřené do špičatých závorek (`<`, `>`) a ukončující tag bude mít vždy stejný název jako uvozující, až na to že bude před ním navíc lomítko.

## 2.3 Analýza textu

Nyní se dostáváme k jádru problému, a tím je analýza jazyka, který je použit v novelách zákonů. Pro lepší přiblížení si ukážeme několik příkladů vět, které jsou pro tento styl typické:

*V § 5 odst. 3 písm. b) se body 3, 4 a 7 zrušují.*
*V § 6 odst. 1 písmenech d) a e) a v § 7 odst. 3 se slovo „musí“ nahrazuje slovem „mohou“.*
*Odstavec 2 se zrušuje, dosavadní odstavce 3 až 8 se označují jako odstavce 2 až 7.*
*Poznámka pod čarou č. 4e) zní: „4e) § 6 zákona č. 96/1993 Sb.“.*
*V § 22 odst. 2 se tečka na konci písmene z) nahrazuje čárkou a vkládá se nové písmeno za), které zní: „za) hodnota závazků, jejichž úhrada by byla výdajem.“.*
*V § 5 odst. 10 písm. a) větě první se za slovo „započtením“ vkládají slova „, splynutím práva s povinností u jedné osoby,“.*
*V § 6 odst. 4 zákona č. 440/2003 Sb., o nakládání se surovými diamanty, o podmínkách jejich dovozu, vývozu a tranzitu a o změně některých zákonů, ve znění zákona č. 60/2005 Sb., se na konci textu písmene c) doplňují slova „o boji proti legalizaci výnosů z trestné činnosti“.*

Je vidět, že slovník tohoto jazyka je značně omezený, vynecháme-li všechna čísla, názvy písmen, texty v uvozovkách a názvy zákonů. Podrobnějším studováním novel zjistíme, že se zde například vyskytuje pouze několik sloves: nahrazovat, doplňovat, vkládat, zrušovat, označovat a znít. Podobné je to s ostatními druhy slov. Věty však vůbec nejsou jednoduché po syntaktické stránce. Často jsou tu dlouhá členitá souvětí, vyskytují se zde dlouhé přívlastky, rozvinutá jsou zejména příslovečná určení místa. Mnohdy je to na úkor jednoznačnosti. Například u příslovečného určení *v § 5 v odst. 4, v nadpisu a bodech 1, 2 a 4 přílohy č. 2* není jasné, jestli se slovo *nadpis* vztahuje k *paragrafu* nebo k *příloze*. Čtenář se podívá na oba dva nadpisy a usoudí o který nadpis asi jde. Počítačový program ale v tomto případě nemá šanci. Může zkusit jeden nadpis, může zkusit oba, každopádně ale už není zaručeno, že provede danou změnu v pořádku.

Je potřeba nějak zapsat gramatická pravidla, podle kterých se tyto věty tvoří. Například: Souvětí se skládá z vět oddělených čárkou a mezerou, nebo spojkou „a“. Věta může znít třeba tak, že se někde něco něčím nahrazuje. Za zkratkou „odst.“ se vždy vyskytuje číslo tohoto odstavce. K takovému popisu se hodí bezkontextová gramatika. Co je bezkontextová gramatika? Je to soubor pravidel, která se skládají z terminálů a neterminálů. Terminálem označujeme jakékoli slovo našeho jazyka, například „odstavec“, „vkládá“ nebo „125/2005 Sb.“. Neterminál slouží jako proměnná, která se pak pomocí pravidel nahrazuje dalšími terminály a neterminály. Pravidla se skládají z jednoho neterminálu nalevo, který se přepisuje do skupiny terminálů a neterminálů napravo. Objasníme si to na následujícím příkladě. Máme tato pravidla:

*VĚTA → KDE se CO nahrazuje ČÍM*
*VĚTA → KDE se CO zrušuje*
*KDE → v ČEM*
*ČEM → odstavci 2*
*ČEM → paragrafu 5*
*CO → slovo „nákladní“*
*CO → písmeno c)*
*ČÍM → slovem „osobní“*

Velkými písmeny byly v pravidlech označovány neterminály, malými terminály. Jaké typy vět jsme touto gramatikou popsali? Vycházíme z neterminálu *VĚTA* a dokud se nám ve větě vyskytuje nějaký neterminál, nahrazujeme ho použitím libovolného pravidla vycházejícího z tohoto neterminálu dalšími terminály a neterminály. Na konci nám zbudou ve větě pouze terminály tvořící větu vygenerovanou touto gramatikou. Tak například věta *v odstavci 2 se písmeno c) zrušuje* je větou tohoto jazyka. Dosáhneme jí tímto způsobem: *VĚTA → KDE se CO zrušuje → v ČEM se CO zrušuje → v odstavci 2 se CO zrušuje → v odstavci 2 se písmeno c) zrušuje*. Stejně tak gramatika generuje větu *v paragrafu 5 se slovo „nákladní“ nahrazuje slovem „osobní“*. Negeneruje ovšem třeba větu *v paragrafu 5 odstavci 2 se slovo „nákladní“ zrušuje*. To bychom tam museli přidat ještě pravidlo *KDE → v ČEM ČEM*. Více o gramatikách se dozvíte v [1].

Je zřejmé, že jazyk novel předpisů půjde nějak takovýmto způsobem popsat. Ještě zbývá jedna drobnost. Co s čísly, písmeny, texty v uvozovkách a názvy zákonů, které dokonce ani v uvozovkách nejsou? Takových by mohlo být nekonečně mnoho a není možné je do gramatiky napsat. Řešení je jednoduché, pro tyto věci si vyhradíme zvláštní neterminály, ze kterých nebudou smět žádná pravidla gramatiky vycházet a budou prostě znamenat: tento neterminál označuje jakékoliv číslo, tento neterminál obsahuje libovolný text, který začíná a končí uvozovkami. V naší vzorové gramatice tedy definujeme dva zvláštní neterminály *ČÍSLO* a *TEXTVUVOZOVKÁCH* a přidáme pravidla

| | |
|---|---|
| *ČEM → odstavci ČÍSLO* | *CO → slovo TEXTVUVOZOVKÁCH* |
| *ČEM → paragrafu ČÍSLO* | *ČÍM → slovem TEXTVUVOZOVKÁCH* |

Nyní nám už gramatika generuje třeba větu *v paragrafu 258 se slovo „benzínový“ nahrazuje slovem „naftový“* a nekonečně mnoho dalších. Tímto způsobem se dají ovšem ošetřit pouze neterminály, které se dají nějak jinak algoritmicky popsat. Číslo je skupina alfanumerických znaků, kde prvním znakem je číslice (musíme k nim zahrnout například i číslo *32zb*). Text v uvozovkách je libovolná skupina znaků, která začíná dolními uvozovkami a končí k nim párovými horními. V předpisech se ale bohužel vyskytují i další slova, která nejsou v uvozovkách a která se týkají spíše tématu konkrétního předpisu. Jako příklad uvedeme věty:

*V příloze č. 1 odpisové skupině 4 položce (4-2) se slova „budov“ a inženýrských staveb zrušují.*
*V příloze č. 3 části 2 v kolonce g se pro referenční čísla 1 až 60 slova „30.9.2004“ nahrazují slovy „31.12.2005“.*

Slovo *odpisová skupina* má význam jen v zákoně o daních z příjmů, podobně slovo *referenční číslo* se týká jen několika málo předpisů. Takováto slova samozřejmě nemůžeme všechna do gramatiky přidat, s novým předpisem by se vždy objevila nějaká nová, a to už odporuje našemu zadání – automaticky zákony novelizovat. Budeme se tedy muset spokojit s tím, že program zkrátka všechny věty novely nebude umět číst. Drtivou většinu vět ovšem naše gramatika generovat bude.

Pomocí bezkontextové gramatiky tedy budeme moci u každé věty určit, zda ji gramatika generuje nebo negeneruje, v kladném případě pak nalezneme její strom (postup, jakým byla tato věta pomocí pravidel vytvořena). Z tohoto stromu potřebujeme nějak zjistit, co je třeba v předpise vlastně opravit. To už půjde relativně jednoduše. Použili jsme například pravidlo *VĚTA → KDE se CO zrušuje* – budeme tedy mazat. Co budeme mazat, to se dozvíme z toho, jaká pravidla byla použita pro rozvinutí neterminálu *CO*. Kde to máme hledat se zase dozvíme z neterminálu *KDE*.

Není to ale vždycky takto jednoduché. Potíže bude dělat právě třeba neterminál typu *KDE*, který bývá často velmi rozvitý a odkazuje na víc míst v jednom předpisu. Vezměme například text *v § 8 odst. 2, v písmenech b) a d) odstavce 3 a v § 9 v nadpise a odstavci 1*. Kolik míst nám toto velmi rozvinuté příslovečné určení místa vlastně určuje? Je jich 5. Jde o odstavec, dvě písmena, nadpis a ještě jeden odstavec. Zde se musíme rozhodovat, která informace o umístění nám pouze více specifikuje předchozí informaci a která nám říká, že jde o jiné místo. Částečně to jde rozlišit podle oddělovačů jednotlivých jmen, jsou buď oddělená pouze mezerou, čárkou nebo spojkou „a“. Někdy se před dalším jménem opakuje předložka, někdy ne. Bohužel se způsob tohoto členění příslovečných určení v různých předpisech liší. Píšou je různí lidé a každý z nich to cítí trochu jinak.

Některá pravidla však přesto můžeme vyvodit: Pokud za jménem následuje přívlastek ve druhém pádě, jméno tento přívlastek určitě konkretizuje (např. slova *v písmenu b) odstavce 3*). Stejně tak pokud mezi dvěma jmény je pouze mezera, bude taky nejspíše to druhé konkretizovat to první (např. *v § 8 odst. 2*). V ostatních případech už nemůžeme nic s určitostí tvrdit. Přichází na řadu tedy sémantika jmen. Dvě stejná jména musí rozlišovat dvě různá místa, slovo *nadpis* už nelze dál konkretizovat, a podobně. To, aby se k sobě jednotlivá jména správně přiřadila, musí zařídit gramatika. Dlouhé příslovečné určení se tedy nejdřív rozdělí podle čárek a spojek „a“, pak podle předložek a až nakonec se rozdělí jména oddělená pouze mezerou. Při procházení stromu věty pak budeme muset zajistit správné slučování jmen.

Závěrem tedy můžeme říci, že tento úkol není lehký a z výše uvedených důvodů nemůžeme nikdy dosáhnout stoprocentní úspěšnosti. Jediným spolehlivým řešením by bylo přimět tvůrce předpisů, aby se při psaní novel vyvarovali nejednoznačných vět, omezili se pouze na určitý slovník a nedělali chyby. Toto omezení není až tak veliké, protože naprostá většina vět nyní těmto požadavkům vyhovuje. V každé obsáhlejší novele se ovšem vždy najde nějaká, se kterou si automat zkrátka neporadí.

# 3 Popis programu

## 3.1 Ovládání programu

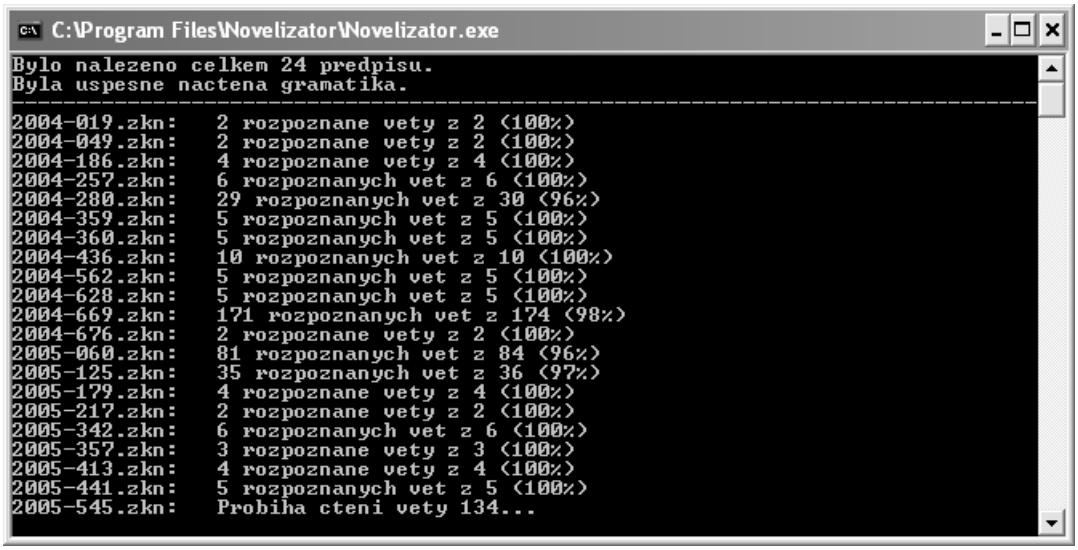
Základní funkcí programu `novelizator.exe` je načíst všechny požadované novely, načíst příslušné předpisy, které tyto novely mění, a předpisy aktualizovat. Není samozřejmě zaručena stoprocentní úspěšnost aktualizace, proto program ještě generuje do zvláštního adresáře chybové soubory. Jde o kopie vstupních novel, ve kterých jsou už ale pouze ty věty, které nebyly programem rozpoznány nebo se je nepodařilo vykonat. Některé věty by samy o sobě nedávaly smysl, protože navazují na předchozí, např. věta „Písmeno a) se zrušuje.“ nám nedává žádné informace o čísle předpisu a paragrafu, tyto jsou zřejmě definovány v předchozích větách. Proto i předchozí věty, které jsou součástí stejného bloku nebo nadřazených bloků se v souboru ponechávají. Chybový soubor by měl tedy mít stejný tvar jako obyčejná novela. V ideálním případě bude prázdný.

Program pracuje se třemi typy souborů, každý soubor odpovídá jednomu předpisu a pro každý typ je vytvořen zvláštní adresář. Jména těchto adresářů včetně cest lze měnit v konfiguračním souboru `novelizator.cfg`. Pro zjednodušení je v tomto textu budeme nazývat jejich originálními názvy, jde tedy o adresáře `predpisy`, `uplna_zneni` a `nezpracovane_vety`. Adresář `predpisy` je určen pro vstup. Odtud se načítají novely zákonů, které bude program číst, stejně tak se z něj načítají původní znění předpisů, které se budou měnit. Do adresáře `uplna_zneni` se pak ukládají výstupy – již aktualiovované předpisy. Do posledního adresáře `nezpracovane_vety` se ukládají již výše zmíněná zredukovaná znění novel, ve kterých je to, co se nepodařilo programu změnit. V konfiguračním souboru se dá generování těchto souborů zakázat, nahradí-li se jméno tohoto adresáře slovem „NE“. Přesný formát a možnosti souboru `novelizator.cfg` jsou popsány v kapitole 3.3. Pokud neexistují adresáře `uplna_zneni` nebo `nezpracovane_vety`, program je na začátku sám vytvoří. Jména souborů se skládají z ročníku, pomlčky a čísla předpisu doplněného vepředu nulami tak, aby bylo trojciferné. Přípona souboru je `.zkn` (od slova zákon). Jde tedy o jména typu `1992-586.zkn`, `2006-080.zkn` apod. Přesný formát těchto souborů je popsán v kapitole 3.2.

Program nemá žádné grafické prostředí a spouští se z příkazového řádku. Pokud nezadáme žádné parametry, přečte všechny novely z adresáře `predpisy`. Konfigurační soubor má ještě jeden parametr, který se v tomto případě využije. Lze definovat číslo předpisu, od kterého se mají novely číst. Všechny starší novely se pak tedy ignorují. Jako parametry můžeme zadat konkrétní soubory (včetně cest a wildcardových znaků `*` a `?`), které chceme číst. Program v tomto případě čte pouze tyto soubory a obsah adresáře `predpisy` na to tedy nemá vliv. Žádné jiné parametry kromě jmen souborů nejsou povoleny.

Gramatika, kterou program používá, je uložena v souboru `gramatika.cfg`. Lze ji tedy přidáváním dalších pravidel rozšiřovat bez nutnosti zásahu do zdrojového kódu programu. Provádět změny v tomto souboru se však doporučuje pouze zkušenějším uživatelům, kteří mají představu o tom, na jakém principu celý program funguje. V každém případě je vhodné si původní soubor zálohovat. Formát souboru `gramatika.cfg` je popsán v kapitole 3.4.

Po spuštění programu se načítají a kontrolují oba konfigurační soubory `novelizator.cfg` a `gramatika.cfg`. Pokud se nalezne chyba, vypíše se na obrazovku a program se ukončí. Jinak se vypíše počet nalezených novel a hlášení o úspěšném načtení gramatiky. Poté začne program procházet jednotlivé novely, číst jejich věty a novelizovat předpisy. Na obrazovku se vždy vypíše jméno čteného souboru a číslo věty, kterou právě program čte. Většina vět se přečte rychle, občas se ale vyskytne nějaká komplikovanější věta, jejíž rozpoznávání může trvat i několik sekund. Po skončení program vypíše statistiku – kolik vět četl a kolik vět z toho rozpoznal a vykonal.




*Obrázek 1: Běh programu Novelizator.exe*

Formát `*.zkn` není pro uživatele přehledný, obsahuje množství značek a špatně se v něm orientuje. Proto byl vytvořen ještě pomocný program `zkn2html.exe`, který převádí předpisy do jazyka HTML a lze si je tedy pak ve kterémkoli prohlížeči prohlédnout. Jako parametry uvedeme jména souborů (včetně cest a wildcardových znaků `*` a `?`), které chceme zkonvertovat, přepínač `-o` slouží k definování výstupního adresáře. Volání tedy může vypadat například takto:

```
zkn2html.exe -o uplna_zneni_v_html uplna_zneni\*.zkn
```

Program je určen pro operační systém Windows 98 a vyšší, minimální konfigurace počítače je procesor 466MHz, 64MB RAM. Na pomalejších strojích nebyl testován, ale fungoval by nejspíš také, samozřejmě ale na úkor rychlosti zpracovávání.

V příloze této práce je CD obsahující jak hlavní program `novelizator.exe`, tak i konvertor `zkn2html.exe` a to včetně všech zdrojových kódů a testovacích dat. Dále je tam několik skriptů pro účely konvertování předpisů stáhnutých z internetu do formátu `*.zkn` a pro testování programu. Pozor, tyto skripty jsou pouze pracovní a nekonvertují předpisy dokonale. Po jejich použití je třeba ještě spoustu chyb v předpisech opravit. Na neupravených předpisech se chybovost novelizátoru prudce zvýší, stačí třeba jen jedny chybné uvozovky nebo chybějící tag na začátku souboru a program nebude schopen přečíst ani jednu větu.

## 3.2 Formát souborů *.zkn

Struktura souboru je podobná například jazyku HTML. Pokud bychom odstranili všechny tagy, které jsou uzavřeny do ostrých závorek, zbude nám prostý text, naprosto shodný se zněním zákona, až na mezery a odřádkování. Program bere (stejně jako u HTML) všechny shluky bílých znaků jako jednu mezeru. K formátování a zarovnávání textu tady slouží tagy. Hlavním úkolem tagů je ale dávat programu informace o názvu a označení jednotlivých částí textu. Všechny tagy kromě `<tab>` a `<br>` jsou povinně párové. Zde je přehled používaných tagů:

| | | | |
|---|---|---|---|
| `<zkn>...</zkn>` | zákon | `<ods>...</ods>` | odstavec |
| `<cst>...</cst>` | část | `<psm>...</psm>` | písmeno |
| `<cln>...</cln>` | článek | `<bod>...</bod>` | bod |
| `<dil>...</dil>` | díl | `<pol>...</pol>` | položka |
| `<odd>...</odd>` | oddíl | `<ppc>...</ppc>` | poznámka pod čarou |
| `<prg>...</prg>` | paragraf | `<opp>...</opp>` | odkaz na poznámku |
| `<tit>...</tit>` | titulek předpisu | `<prl>...</prl>` | příloha |
| `<zdn>...</zdn>` | datum schválení | `<ozn>...</ozn>` | označení bloku |
| `<vzn>...</vzn>` | ve znění zákona | `<ndp>...</ndp>` | nadpis bloku |
| `<tab>` | tabulátor | `<tbl>...</tbl>` | tabulka |
| `<br>` | konec řádky | | |

Pro lepší pochopení významu jednotlivých tagů bude nejvhodnější podívat se přímo do některých vzorových předpisů v adresáři `predpisy`. Každý soubor musí začínat a končit tagy `<zkn>` resp. `</zkn>`. Na začátku každého bloku (části, paragrafu, písmena) je mezi tagy `<ozn>` jeho označení (např. „ČÁST PÁTÁ“, „§ 12“, „c)“). Pak následuje nepovinný nadpis (`<ndp>`). Ještě je nutné zmínit několik pravidel, která je třeba dodržovat. Uvozovky v souboru musí být české párové, tj. „a “. Označení paragrafů se píše s mezerou, tedy například „§ 26“, nikoli „§26“. U odkazu na poznámku pod čarou se píše závorka mezi tagy, tedy `<opp>15)</opp>`, nikoli `<opp>15</opp>)`. U tabulek je oddělovačem buněk `<tab>`, oddělovačem řádků je `<br>`. Na začátku a konci řádku se `<tab>` vynechává, stejně tak na začátku a konci tabulky se vynechává `<br>`. Zde je uveden příklad tabulky:

```
<tbl>
  první<tab>druhá buňka<tab>třetí buňka<br>
  dvě spojené buňky ve druhém řádku<tab><tab>buňka<br>
  buňka<tab>buňka<tab>buňka
</tbl>
```

## 3.3 Formát souboru novelizator.cfg

V tomto souboru jsou uloženy některé parametry předávané programu. Prázdné řádky a řádky začínající znakem pro komentář (#) program ignoruje. Ostatní musí mít tvar `promenna='hodnota'`.

Význam jednotlivých proměnných:

- `pre` – Vstupní adresář, kde jsou uloženy všechny předpisy určené ke zpracování.
- `uzn` – Do tohoto adresáře se ukládají vygenerovaná úplná znění.
- `nzp` – Zde se ukládají vygenerované novely s nerozpoznanými větami. Je v nich tedy jenom to, čemu program neporozuměl. Pokud se místo cesty v apostrofech uvede slovo „NE“, nebudou se vůbec tyto soubory generovat.
- `odp` – Číslo předpisu, od kterého se začne číst. Starší předpisy budou ignorovány. Program tedy zpracuje pouze předpisy s vyšším číslem včetně tohoto.

## 3.4 Formát souboru gramatika.cfg

V tomto souboru jsou uložena všechna gramatická pravidla, zároveň slouží i jako slovník. Je v textové podobě, aby se dala lépe přidávat nová pravidla, nová slova a nové obraty podle potřeby, bez nutnosti zasahovat do zdrojového kódu programu. Jazyk, který je používán v novelách zákonů, je zde popsán bezkontextovou gramatikou (viz kapitola 2.3). Každé pravidlo je na zvláštním řádku, za ním jsou v jednoduchém jazyce napsány příkazy, které je třeba po použití tohoto pravidla provést. Pravidlo má následující tvar:

{*neterminal*} [*terminaly a neterminaly*] *jednotlivé příkazy oddělené středníkem*

Pravá část pravidla je uzavřena do hranatých závorek. Názvy neterminálů uzavíráme do složených závorek, například neterminál označující větu je `{veta}`. Pravidla tedy mohou vypadat například takto:

```
{veta} [parlament se usnesl na tomto zákoně České republiky]
{veta} [{kde} se {co} nahrazuje {cim}]
{jmeno1p} [poznámka pod čarou č. {cisla}]
{jmeno1p} [věta {kolikata}]
```

Mezi závorkami může být libovolný počet mezer. Mezery uvnitř závorek jsou však již součástí pravidel! Existuje několik neterminálů (budeme jim říkat „speciální“), které jsou automaticky zabudovány v programu, a tudíž nesmí být použity v levé části žádného pravidla:

- `{cislo}` označuje číslo. Není to ale pouze libovolný shluk cifer, je to jakákoli část textu bez mezer, která začíná číslicí a za ní mohou být další alfanumerické znaky, povolena je navíc ještě tečka a pomlčka. Patří sem tedy třeba čísla 124, 8a, 21bc, 2.2, 58-16 a podobně.

- `{pismeno}` označuje písmeno. Tvar je stejný jako u čísla, musí tedy začínat písmenem, pokračovat může dalšími alfanumerickými znaky, tečka a pomlčka ale nejsou v tomto případě povoleny. Například sem tedy patří písmena p, Zx8, Ca, ale nikoli třeba h-a.

- `{text}` označuje část textu uzavřenou mezi dolní a horní uvozovky, a to s těmito uvozovkami včetně.

- `{cislozakona}` označení zákona, tedy číslo, lomítko, ročník a „Sb.“. Například 586/1992 Sb., 90/2004 Sb., 545/2005 Sb. apod.

- `{ocem}` méně důležitý neterminál, který se používá za číslem zákona a označuje, o čem tento zákon pojednává. Definovaný je tak, že je to jakákoli část textu začínající písmenem „o“, po kterém následuje mezera. Na tento neterminál je třeba dávat pozor, může se na něj chytit i to, co nechceme.

Pravidla mohou být v souboru uspořádána libovolně, nezáleží tedy, že v pravé části pravidla se vyskytuje neterminál, který je definován až níže. Pokud se někde vyskytne neterminál, který není nikde definován, program na to upozorní a hned skončí. Stejně tak upozorní na chybné řádky. Přeskakuje pouze prázdné řádky, složené jen z bílých znaků, a řádky začínající znakem #. Ten je vyhrazen pro komentář.

Mějme však na paměti, že program se bude pokoušet aplikovat pravidla postupně od začátku tak, jak jsou v souboru zapsány. Z hlediska větší časové optimalizace je třeba tedy ta složitější pravidla dávat spíše před ta jednodušší. U jednoduchých vět se ta složitější okamžitě zamítnou, naopak na složité věty by šla ta kratší a jednodušší pravidla aplikovat velmi dobře a velmi dlouho by pak trvalo, než by se vyzkoušením všech možností dospělo k závěru, že toto opravdu nelze použít.

Pro příkazy za pravidly byl vytvořen speciální jednoduchý jazyk. Jeho proměnné mohou nabývat několika typů:

**retezec** – klasický řetězec jak ho známe, tedy nějaká uspořádaná množina znaků.

**vycet** – je to jakési pole hodnot typu retezec, které je určeno k výčtu označení bloků v předpisech. V zákoně se například objeví slovní spojení „odstavce 5, 6, 8a a 10“ a k označení těchto odstavců se použije právě proměnná typu `vycet`, která bude mít hodnotu `{5,6,8a,10}`. Nesmíme zapomenout na případy, kde se definuje označení i rozsahem hodnot, třeba „písmeno b) až e)“. V tomto případě se do výčtu vkládá speciální znak „-“, který určuje interval. Tento znak se samozřejmě nesmí nacházet ani na začátku, ani na konci výčtu. Čísla „11, 12, a 18 až 22a“ tedy budou v proměnné typu výčet vypadat takto: `{11,12,18,-,22a}`.

**vec** – označuje podstatné jméno ve větě, tedy třeba slova paragraf, odstavec, písmeno, poznámku pod čarou atd. Kromě názvu (typu `retezec`) a označení (typu `vycet`) má spoustu dalších vlastností, které jsou popsány níže.

**skupina** – pole proměnných typu `vec`. Označuje jeden konkrétní blok, tedy např. zákon 586/1992 Sb., § 22a, odstavec 1, písmeno p).

**misto** – je také pole proměnných typu `vec`, ale může být dvourozměrné. Ukládají se do něj informace o umístění hledané věci v zákoně. Takových míst může být samozřejmě i více, proto tedy dvourozměrné. Například: zákon 120/2002 Sb., § 2, nadpis; zákon 120/2002 Sb., § 3, odstavec 12, věta druhá.

**veta** – jedna věta obsahující jedno určité sloveso, také má spoustu vlastností a metod.

**souveti** – pole proměnných typu `veta`

Proměnná se deklaruje klíčovým slovem označujícím typ, do závorky se pak napíše její jméno, které smí sestávat pouze z písmen. Tedy například: `souveti(s);`, `misto(mst);` nebo `retezec(r);`. Kromě s vámi nadeklarovanými proměnnými se dá také pracovat s návratovými hodnotami z ostatních pravidel. Pro příklad uvedeme pravidlo

```
{veta} [{co} zní: {text}]  veta(v); v.co({1}); v.zneni({2});
v.akce('nahradit'); navrat(v);
```

Vycházíme z toho, že části věty označené neterminály `{co}` a `{text}` již byly zpracovány a vrátily nějaké návratové hodnoty (konkrétně neterminál `{co}` vrací hodnotu typu `skupina` a `{text}` hodnotu typu `retezec`). Jména těchto návratových hodnot jsou `{1}` a `{2}`, tedy čísla určující pořadí v pravidle uzavřená do složených závorek. S tímto pravidlem se tedy vytvoří proměnná `v` typu `veta`, provedou se na ní přiřazovací funkce `co` s parametrem `{1}`, `zneni` s parametrem `{2}` a `akce` s parametrem `'nahradit'`. Návratová hodnota tohoto pravidla bude `v` a je typu `veta`. Všimněme si, že proměnnou typu `retezec` není třeba explicitně definovat, pokud se použije pouze jednou jako parametr nějaké funkce. Pokud je tedy parametrem něco v apostrofech, automaticky se vytváří proměnná typu `retezec` a předává se rovnou funkci (např. `v.akce('nahradit');`).

Každá řádka musí končit voláním funkce `navrat()`. Klasicky má jeden parametr, ale může být i bez parametrů, pokud se nevrací nic. Návratová hodnota pak ale samozřejmě nesmí být použita. Pozor, program neprovádí typovou kontrolu! Pokud narazí na nějakou nesrovnalost v typech, se vším končí, prohlásí právě analyzovanou větu za nerozpoznanou a pokračuje následující větou.

Jednotlivé typy proměnných mají následující metody (v závorkách jsou uvedeny typy parametrů):

- retezec:
  `prirad(retezec)`
- vycet:
  `pridejretezec(retezec)`
  `pridejvycet(vycet)`
  `pridejinterval(retezec, retezec)`
- vec:
  `nazev(retezec)`
  `vlastnost(retezec)`
  `oznaceni(vycet)`
  `ceho(vec)`
  `text(retezec)`
  `vcetne(vec)`
  `umisteni(retezec)`

- skupina:
  `pridejvec(vec)` `pridejskupinu(skupina)`
- misto:
  `pridejvec(vec)` `pridejmisto(misto)`
- veta:
  `kde(misto)` `jakoco(skupina)`
  `kam(skupina)` `akce(retezec)`
  `co(skupina)` `zneni(retezec)`
  `cim(skupina)`
- souveti:
  `pridejvetu(veta)` `pridejsouveti(souveti)`

Zvláštní postavení zde mají parametry typu `retezec`, jejichž hodnoty jsou většinou zadávány uživatelem. U některých metod mohou parametry typu `retezec` nabývat pouze některých hodnot, kde každá má svůj specifický význam. Všechny ostatní hodnoty jsou programem ignorovány. Jde o tyto metody:

**veta.akce(retezec)** – parametr označuje typ změny: `'nahradit'`, `'vlozit'`, `'smazat'`, `'preznacit'` nebo `'nastavit'`.

**vec.vlastnost(retezec)** – parametr zde může nabývat hodnot `'novy'`, `'cely'`, `'dosavadni'`, `'vsechny'` nebo `'ostatni'`.

**vec.umisteni(retezec)** – zde přicházejí v úvahu hodnoty `'v'` (uvnitř bloku), `'nz'` (na začátku), `'nk'` (na konci), `'pr'` (před) a `'za'`. Pokud se umístění neuvede, bere se automaticky jako `'v'`.

**vec.nazev(retezec)** – tady může parametr nabývat těchto hodnot:

| | |
|---|---|
| `'bod'` – bod | `'ppc'` – poznámka pod čarou |
| `'crk'` – čárka | `'prl'` – příloha |
| `'cst'` – část | `'prg'` – paragraf |
| `'dil'` – díl | `'psm'` – písmeno |
| `'dvt'` – dvojtečka | `'slv'` – slovo |
| `'hlv'` – hlava | `'str'` – středník |
| `'ndp'` – nadpis | `'tck'` – tečka |
| `'odd'` – oddíl | `'ucu'` – úvodní část ustanovení |
| `'ods'` – odstavec | `'vet'` – věta |
| `'ozn'` – označení | `'zcu'` – závěrečná část ustanovení |
| `'pol'` – položka | `'zkn'` – zákon |

Všechny metody buďto přiřazují nějaké své proměnné nějakou hodnotu, přidávají prvek na konec nějakého pole nebo nějaká dvě pole slučují. Nejsložitější je zřejmě metoda `misto.pridejmisto()`, která slučuje dvě dvojrozměrná pole typu místo. Musí je sloučit tak, aby se v jedné části neopakovaly věci se stejným názvem, zároveň se někdy musí při přidávání do pole některé věci kopírovat do více částí.

Pro lepší pochopení významu jednotlivých metod je nejlepší se podívat přímo do zdrojového souboru `vyhodnocovani.cpp`. Uvedeme ještě jeden příklad. Máme pravidlo

```
{veta} [doplňuje se {co}] veta(v); v.co({1}); v.akce('vlozit');
skupina(s); vec(vv); vv.umisteni('nk'); s.pridejvec(vv);
v.kam(s); navrat(v);
```

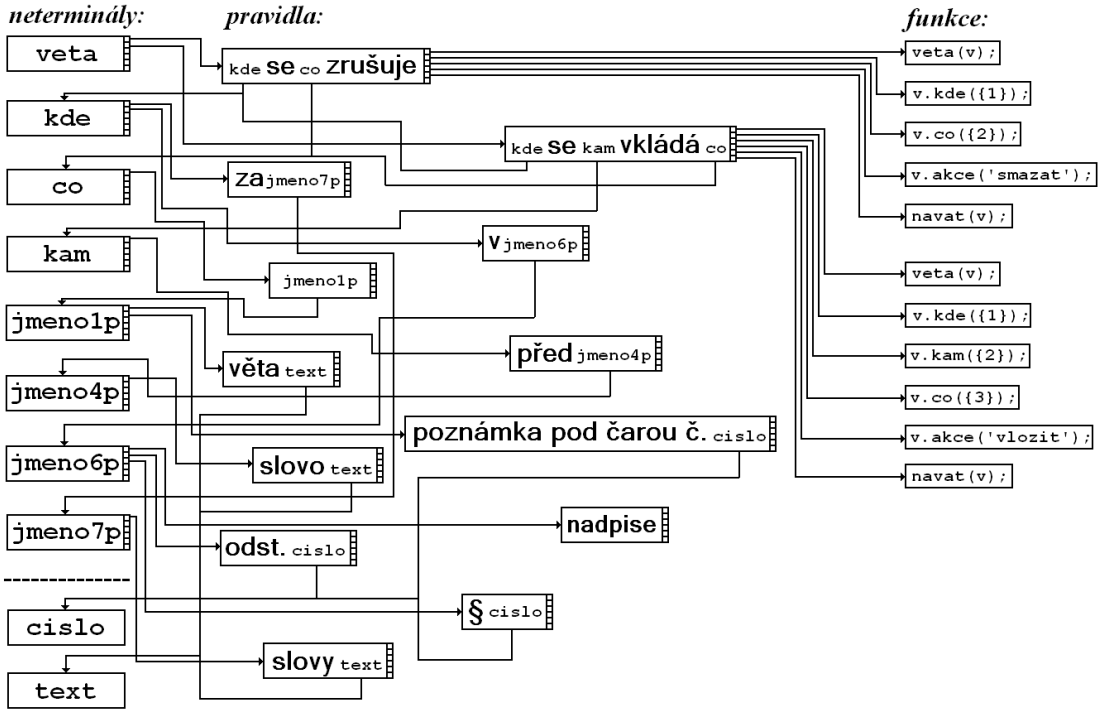
Vytvoří se tedy věta `v`, ve které se do proměnné `co` přiřadí návratová hodnota z neterminálu `{co}`. Slovo „doplňuje“ znamená, že se bude vkládat, akce se tedy nastaví na `'vlozit'`. Při vkládání však vždycky musíme vědět kam vkládat. V této větě o tom není zmínka. Použilo se ale sloveso „doplňovat“, předpokládáme tedy, že se bude vkládat na konec. Toto musíme programu nějak oznámit. Proměnná `kam` ve větě je typu `skupina`. Proto vytvoříme novou proměnnou typu `skupina`, do které vložíme proměnnou typu `vec`, která bude mít hodnotu `umisteni` nastavenou na `'nk'`. Nakonec se celý objekt `v` typu `veta` vrátí.

## 3.5 Implementace

Ze všeho nejdřív se načtou data z konfiguračního souboru `novelizator.cfg`. K tomu je určena funkce `nacti_udaje()` souboru `inicializace.cpp`. Ta nám naplní globální objekt `udaje` typu `Udaje`, který obsahuje jména předpisů určených ke zpracování a jména pracovních adresářů. Pokud neexistuje vstupní adresář (`predpisy`), který je zadán v konfiguračním souboru, program ohlásí chybu a skončí. V případě neexistence výstupních adresářů (`uplna_zneni` a `nezpracovane_vety`) tyto adresáře vytvoří. Pokud jsou zadány programu nějaké parametry, naplní pole `predpisy` cestami ke všem souborům odpovídajícím zadaným parametrům (zadány mohou být i s použitím wildcardových znaků `*` a `?`). V opačném případě naplní toto pole cestami ke všem souborům požadovaného typu (`*.zkn`) adresáře `predpisy`. Popis formátu konfiguračního souboru najdete v kapitole 3.3.

Následuje načtení gramatiky ze souboru `gramatika.cfg` pomocí funkce `nacti_gramatiku()` souboru `inicializace.cpp`. Gramatika se načítá do poměrně složité datové struktury tvořené objekty tříd `Neterminal`, `Pravidlo` a `Funkce` (viz hlavičkový soubor `pravidla.h`). Gramatika je v souboru zaznamenána pravidly bezkontextové gramatiky. Za každým pravidlem jsou ještě jakési příkazy (funkce), které se při použití tohoto pravidla vykonávají. Podrobněji o formátu těchto dat se dozvíte v kapitole 3.4. Prozatím bude stačit, když všechna tato pravidla a funkce načteme do paměti. Objekty třídy `Neterminal` obsahují seznam odkazů na pravidla, která z tohoto neterminálu vycházejí. V objektech třídy `Pravidlo` jsou uložena pravidla s odkazy na všechny neterminály, které jsou tam použity. Konkrétně mají proměnnou `retezec` typu `string`, který obsahuje všechny terminální znaky pravidla, místo neterminálů je vkládán vždy jeden speciální znak. Dále je tam pole odkazů na tyto neterminály. Odkazy jsou tam právě v tom pořadí, jak jdou za sebou v pravidle, pravidlo je tedy určeno jednoznačně. Poslední proměnnou je pole odkazů na objekty třídy `Funkce`. Každý tento objekt reprezentuje jeden příkaz (funkci) uvedený v souboru za pravidlem. Obsahuje tedy název proměnné (pokud tam nějaká je), jméno metody, která se na ní volá, a seznam parametrů.

Jak tedy tato funkce funguje? Je zřejmé, že se nemůžou rovnou vytvářet objekty třídy `Pravidlo`, protože ještě neexistují objekty typu `Neterminál` a naopak. Pravidla se tedy dočasně ukládají do pole typu `Predzpracovane_pravidlo`, stejně tak každý nový neterminál, ať už se vyskytuje na levé straně pravidla nebo na pravé, se uloží do objektu `Predzpracovany_neterminal`. Funkce tedy celý soubor projde a naplní pole dočasných objektů. Na konci už má plnou informaci o všech vyskytujících se neterminálech a může je tedy vytvořit, tím dostane jejich adresy a může vytvořit pravidla, která se na tyto neterminály odkazují. Podobně se vytvoří i objekty třídy `Funkce`. Vytvoří se i „speciální“ neterminály (reprezentující čísla, písmena, texty v uvozovkách apod.), které se už žádnými pravidly dál nepřepisují a na které také některá pravidla ukazují. Konstruktor objektu třídy `Neterminal` má jako parametr typ neterminálu (znak `n` pro obyčejný neterminál a znaky `c`, `p`, `t`, `z` a `o` pro speciální neterminály `{cislo}`, `{pismeno}`, `{text}`, `{zakon}` a `{ocem}`). Dále je definován jeden neterminál jako startovací, v našem případě je to neterminál `{souveti}`. Ten se musí vždy vyskytovat na levé straně nějakého pravidla a naopak nesmí být nikde na pravé straně. V opačném případě program ohlásí chybu a skončí. Funkce vrátí právě odkaz na tento startovací neterminál. Je to zároveň jediné spojení s celou touto strukturou.




*Obrázek 2: Struktura načtené gramatiky*

Nyní se už můžou začít číst jednotlivé předpisy, všechny potřebné funkce jsou v souboru `cteni.cpp`. Zavolá se hlavní funkce `zpracuj_predpisy()`, které se předají ukazatele na údaje a na startovací neterminál. Funkce projde všechny předpisy určené k přečtení, každý nejprve načte do paměti (`nacti_blok()`) a pokud se to podaří, zpracuje ho funkcí `precti_blok()`. Tato funkce postupně z předpisu umazává to, co se správně přečetlo. Nakonci se pak zbytek, tedy ty části novely, na kterých program selhal, uloží do chybového souboru do adresáře `nezpracovane_vety`. Pro přečtení všech předpisů se funkcí `uloz_uplna_zneni()` uloží nově vytvořená znění předpisů.

Funkce `nacti_predpis()` má parametry `nazev` a `predpis`. Z adresáře `predpisy` načte soubor se jménem `nazev` do stringu `predpis`. V něm ovšem nahradí všechny shluky bílých znaků jednou mezerou. Tím odstraní veškeré formátování souboru včetně konců řádků.

Funkce `nacti_blok()` je rekurzivní. Její parametry jsou reference na zpracovávaný předpis a pozice, od které se má daný blok načítat. Blokem rozumíme jakoukoli část předpisu, která je oddělena navzájem si odpovídajícími párovými tagy, je to tedy například zákon, článek, paragraf, odstavec, bod a podobně. Mezi bloky se v případě čtení novel nepočítají bloky, které jsou v uvozovkách a jsou součástí nějaké věty. Obecně dostane funkce `nacti_blok()` parametrem `pozice` zadán začátek vnitřku bloku. Začne tedy od této pozice číst znak po znaku a rozhoduje se následovně: Pokud narazí na nějaký tag uvozující nějaký podblok, zavolá se sama rekurzivně, přičemž předá pozici za tímto uvozujícím tagem. Pak testuje, následuje-li tag, který tento podblok ukončuje. Pokud ne, vrátí `false`, což znamená, že se tuto část nepodařilo zpracovat. Pokud ano a návratová hodnota z podbloku je `true`, považuje se tento blok za úspěšně vyřízený a může se z předpisu vymazat. Pokud nenásleduje tag, následuje nějaký text, který se načte funkcí `nacti_text()` a následně se zpracuje funkcí `zpracuj_text()`. Funkce `nacti_text()` je důležitá hlavně v tom, že přeskakuje formátovací tagy `<br>` a `<tab>` a ošetřuje uvozovky. Jakmile narazí na dolní uvozovky, čte tak dlouho, dokud nenajde k nim párové horní.

Nyní jsme se již dostali k analýze textu. Funkce `zpracuj_text()` nejprve zadanou část předpisu rozdělí na jednotlivé věty. Konec věty je definován tak, že se tam nachází znak tečka nebo dvojtečka, za kterým následuje mezera a velké písmeno. Nepředpokládáme, že se budou v novelách vyskytovat spojení typu „sv. Jan“ apod. Stejně jako v předchozí funkci jsou ošetřeny texty v uvozovkách. Na jednotlivé věty se nyní aplikují gramatická pravidla. Funkce `test` zjistí, zdali námi zadaná gramatika tuto větu generuje, současně se vytváří strom věty. Pokud ano, vytvoří se objekt `souveti`, který obsahuje už systematicky setříděné informace o významu věty a funkce `prepis_predpis()` pak provede příslušné změny v předpisech. Zároveň se postupně zvyšují hodnoty proměnných `pocet_nactenych_vet` a `pocet_provedenych_vet`, které nám pak na konci dávají informaci o celkové úspěšnosti programu.

Rekurzivní funkce `Neterminal::test()` má jako parametry 2 ukazatele na `char` a určuje, zda lze daný neterminál přepsat nějakými pravidly naší gramatiky na text umístěný mezi těmito adresami. Podobně funkce `Pravidlo::test()` říká, jde-li právě použitím tohoto pravidla dosáhnout požadované části textu. Tyto dvě funkce se volají navzájem. Právě určení těchto pravidel je v rámci celého programu časově nejnáročnější, jeho složitost je exponenciální. Je tedy třeba se v těchto funkcích omezit na co nejméně operací. Proto se během testování generuje strom věty. Lze si ho představit jako seznam odkazů na pravidla, která byla pro vygenerování věty použita. S tímto stromem se pak pracuje při další analýze věty, která už není časově náročná.

Test neterminálu se provádí podle jeho typu. Je-li to obyčejný neterminál (`n`), jednoduše se projdou všechna pravidla, která z tohoto neterminálu vychází a pokusí se použít. Pokud žádné neuspěje, vrátí funkce `false`. Pokud jde o speciální neterminál

`{cislo}`, `{pismeno}` nebo `{zakon}`, zkoumají se znaky v textu, jestli vyhovují příslušným požadavkům, u neterminálu `{text}` musí být prvním a posledním znakem uvozovky, jinak neuspěje, u neterminálu `{ocem}` stačí, když příslušná část textu začíná písmenem `o` a za ním je mezera. Místo odkazu na použité pravidlo se u těchto speciálních neterminálů ukládají do stromu odkazy na tuto část textu.

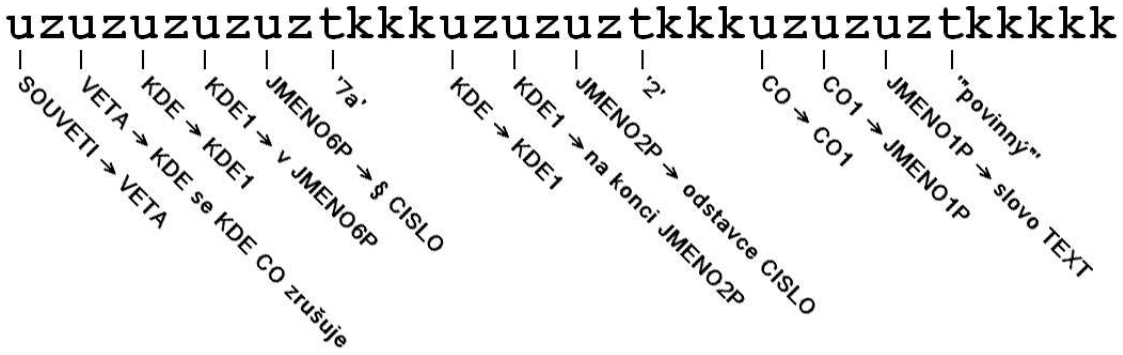
Test pravidla je složitější. Musí se totiž správně určit, které úseky zkoumaného textu odpovídají daným neterminálům v pravidle. Toto umístění hledá pomocná funkce `najdi_neterminaly()`. Jak se tedy postupuje? Od začátku se postupně testuje shodnost znaků pravidla a textu. Pokud se v pravidle narazí na neterminál, postupuje se dál pouze v textu. Pokud se tam objeví znak, který je v pravidle za neterminálem, je naděje, že neterminál skončil a pokračuje se v testování shodnosti znaků. Liší-li se znaky, je několik možností: Pokud se do této doby ještě nevyskytnul v pravidle neterminál, nelze toto pravidlo použít a funkce vrátí `false`. Jinak je možno předchozí neterminál prodloužit až do tohoto místa, respektive do nejbližšího místa, kde by se následující terminální znaky shodovaly. Pokud dojdeme na konec ve čtení pravidla, které končí neterminálem, roztáhneme neterminál až do konce textu. Pokud nekončí neterminálem, je třeba roztáhnout některé předchozí neterminály. Pokud dojdeme až na konec textu a ještě jsme nedošli na konec pravidla, toto pravidlo nelze použít.

Princip parsování gramatiky ukážeme na následujícím příkladě. Mámě třeba větu „V § 7a se na konci odstavce 2 slovo „povinný“ zrušuje.“ Předpokládejme, že až do teď všechna pravidla vycházející z neterminálu `{veta}` neuspěla a nyní se pokoušíme aplikovat pravidlo

```
{veta} [{kde} se {kde} {co} zrušuje].
```

Postupujeme od začátku. Za prvním neterminálem v pravidle se vyskytuje mezera. Hledáme první mezeru a jako neterminál `{kde}` označíme slovo „V“. Za mezerou se ovšem ve větě nevyskytuje dále znak „s“, jako je tomu v pravidle. Proto hledáme další mezeru a neterminál `{kde}` rozšíříme na slova „V §“. Ze stejných důvodů ale ani toto neobstojí. Aby se za tímto neterminálem vyskytly znaky „ se “, musíme ho rozšířit až na „V § 7a“. Za těmito znaky začíná další neterminál, za kterým je opět mezera. Proto hledáme ve větě další nejbližší mezeru a druhému neterminálu `{kde}` přiřadíme řetězec „na“. Tímto postupem pokračujeme dále a vyjde nám, že neterminál `{co}` odpovídá řetězci „konci odstavce 2 slovo „povinný““. Tím jsme našli rozvržení neterminálů v tomto pravidle a na každý zavoláme rekurzivně funkci `Neterminal::test()`, kterou zjistíme zda tento neterminál danou část textu generuje. První, u kterého se to nepodaří je druhý neterminál `{kde}`, který by měl generovat řetězec „na“. Pokusíme se ho tedy prodloužit. Doleva prodloužit nemůžeme, protože neterminály před ním mají minimální možnou velikost. Rozšíříme ho tedy doprava až k další mezeře, tedy na „na konci“, na neterminál `{co}` pak zbude řetězec „odstavce 2 slovo „povinný““. Opět všechny neterminály otestujeme. Nyní nám i druhý netereminál `{kde}` bude souhlasit, neterminál `{co}` ovšem slova „odstavce 2 slovo „povinný““ negeneruje. Prodloužit ho již nemůžeme, ale můžeme prodloužit ještě předchozí neterminál. Tímto způsobem postupujeme dál, až dostaneme takové rozvržení neterminálů, kde všechny jejich příslušné části textu generují. Funkce bude moci vrátit hodnotu `true`.

Strom věty je uložen v poli s prvky typu `PrvekStromu`. Každý prvek má svůj typ, buď je to neterminál (uzel stromu, někdy list), speciální neterminál (výhradně list), začátek uzlu nebo konev uzlu (pomocné prvky). Prvek typu neterminál obsahuje ukazatel na pravidlo, které bylo na tento neterminál použito. Za ním následuje výčet jeho synů, který je uzavřen mezi pomocné prvky začátek a konec uzlu. Synové jsou opět nějaké prvky typu speciální neterminál nebo neterminál (ten je stejným způsobem dále rozvíjen). Prvek typu speciální neterminál je listem, neodkazuje na žádné pravidlo, ale uchovává v sobě přímo řetězec, který v tomto případě reprezentuje. Strom naší vzorové věty z předchozího příkladu obsahuje právě tři tyto prvky: čísla „7a“ a „2“ a text v uvozovkách „povinný“. Strom se plní při parsování gramatiky. Pokud program zjistí, že zkoumané pravidlo nejde použít, celý odpovídající konec stromu smaže a pokračuje dalším pravidlem. Na následujícím obrázku je příklad stromu vzorové věty „V § 7a se na konci odstavce 2 slovo „povinný“ zrušuje“. Odkazy na pravidla jsou označeny písmenem `u`, speciální neterminály písmenem `t` a začátky a konce neterminálů písmeny `z` a `k`.




*Obrázek 3: Ukázka vygenerovaného stromu věty*

Předpokládejme tedy, že gramatika naší větu generuje a máme vygenerovaný strom s odkazy na použitá pravidla. Nyní potřebujeme zjistit, co ta věta říká. K tomu slouží funkce `projdi_strom()` a `vyhodnot_funkce()`. Každé pravidlo vždy vykoná nějaké funkce a vrátí nějakou hodnotu. Funkce `vyhodnot_funkce()` je rekurzivní, sama se spustí na další pravidla rozvíjející jednotlivé neterminály. Pokud se jedná o speciální neterminál a místo odkazu na pravidlo je tedy ve stromu uložen odkaz na řetězec, vytvoří se nová proměnná typu `Typ_Retezec`.

Nyní předpokládejme, že máme rekurzivně zjištěny návratové hodnoty ze všech neterminálů a můžeme tedy začít vykonávat příkazy. Proměnné se ukládají do pole `promenne`. Na každou funkci zavoláme metodu `vyhodnot()`, která podle jména funkce a parametrů změní příslušnou proměnnou, a pokud se jedná o příkaz návrat z pravidla, nastaví návratovou proměnnou. Po vykonání všech příkazů se můžou všechny proměnné, kromě té, která se bude vracet, smazat. První použité pravidlo nám pak nakonec po projití celého stromu věty vrátí hodnotu typu `Typ_Souveti`, která obsahuje veškeré důležité informace o větě. Příklad je na obrázku 4, kde je znázorněna struktura vzorové věty „V § 7a se na konci odstavce 2 slovo „povinný“ zrušuje“.

Funkce `vyhodnot()` rozhodne, zda-li se jedná o konstruktor nové proměnné, o návrat z pravidla nebo zda-li jde o volání metody nějaké proměnné. V případě konstruktoru se jednoduše vytvoří nová proměnná požadovaného jména a typu, v případě návratu se pouze označí jedna proměnná jako návratová a zakáže se vykonávání případných dalších příkazů. Pokud jde o metodu, tak se na příslušné proměnné zavolá funkce `proved()`. Tato funkce je virtuální a pro každý typ proměnné má své vlastní tělo. Je ve zvláštním souboru `vyhodnocovani.cpp`. Ve `vyhodnocovani.h` jsou deklarovány všechny typy proměnných a jaké mají vlastnosti. Pomocí funkce `najdi_promenne()` se najdou proměnné odpovídající parametrům metody. Parametry se následně otestují, jestli je jich správný počet a jestli jsou správného typu. Pak se příslušná metoda provede. Seznam všech metod jednotlivých typů naleznete v kapitole 3.4.

```
souveti[veta1[kde[misto1[vec1[nazev = 'prg'
                             oznaceni[polozka1[retezec = '7a']]]
                        vec2[nazev = 'ods'
                             oznaceni[polozka1[retezec = '2']]
                             umisteni = 'nk']]]]
        co[vec1[nazev = 'slv'
                oznaceni = 0
                retezec = 'povinný']]
        akce = 'smazat']]
```

*Obrázek 4: Struktura objektu Souveti u vzorové věty*

Teď už zbývá pouze provést příslušné změny v předpisech. K tomu slouží soubor `generovani.cpp` a jeho hlavní funkce `prepis_predpis()`, která jako parametr dostane právě objekt typu `Typ_Souveti`. Chceme najít daný předpis a v něm příslušné místo, ve kterém se bude něco měnit, pak najdeme objekt v předpisu odpovídající podmětu věty a ten buď nahradíme něčím jiným, nebo smažeme, případně tam tento objekt nově vložíme, nebo přeznačíme a podobně. Vstupující souvětí si rozdělíme na věty a budeme brát jednu po druhé. Věta obsahuje nějaké informace o umístění hledaného objektu, buď úplné (včetně čísla předpisu, paragrafu atd.), nebo jenom částečné (pouze odstavec nebo písmeno, někdy vůbec nic). Předpokládá se, že zbytek potřebných údajů se již objevil v předchozích větách. Je tu tedy jakási dědičnost informací o umístění. Toto zařizuje globální proměnná `aktualni_blok`. Je to pole hodnot typu `Typ_Vec` obsahující setříděné informace o umístění, tedy třeba číslo předpisu – paragraf – odstavec – písmeno – bod a podobně. Načteme-li další větu, aktualizujeme tuto globální proměnnou tak, aby z toho starého zůstalo jenom to, co je obecnější. Věta nám třeba dá informaci pouze o odstavci, původní odstavec – písmeno – bod tedy smažeme a nahradíme je hodnotou nového odstavce. Číslo předpisu a paragraf zůstanou stejné.

Teď tedy už víme, kde hledat, je třeba tedy najít, které pozice tomuto místu odpovídají ve skutečném předpisu. To se dělá funkcí `najdi_blok()`, které se zadá místo, na kterém se má hledat, co se má hledat a ona vrátí seznam míst odpovídajících zadání. Typ `Misto` označuje nějakou část textu, obsahuje ukazatel na předpis, počáteční a koncovou pozici v předpisu a umístění, které nám říká, zda-li hledat na začátku tohoto bloku nebo na konci, před ním, nebo za ním, nebo jestli najít všechny výskyty.

Tímto způsobem najdeme místo změny, podle hodnoty `akce` u věty se rohodneme co dál. Pokud jde o vložení, najdeme ještě funkcí `najdi_blok` objekt `kam` a na toto místo vložíme text podmětu `co`. Pokud jde o mazání nebo nahrazení, najdeme v předpise ještě objekt `co`, který následně smažeme. V případě nahrazení navíc vložíme objekt `cim`. K přeznačování bloků slouží zvláštní funkce `zmen_oznaceni()`. Na něco ještě nesmíme zapomenout. Po změnách v předpise se znehodnotí všechny proměnné typu `Misto` ukazující do tohoto předpisu za příslušnou změnu. Funkcí `posun_pozice()` se jejich hodnoty aktualizují – změní o příslušný rozdíl.

Princip si opět ukážeme na příkladu. Máme větu „V § 5 odst. 10 se na konci písmene c) tečka nahrazuje čárkou a doplňuje se nové písmeno e), které zní:…“. Po vyhodnocení všech funkcí dostaneme výsledný objekt typu `Typ_Souveti`, který obsahuje 2 věty s těmito hodnotami:

**věta 1:**
akce = nahradit
kde = § 5, odst. 10, písm. c), na konci
kam = 0
co = tečka
cim = čárka
zneni = 0
jakoco = 0

**věta 2:**
akce = vložit
kde = 0
kam = na konec
co = písmeno e)
cim = 0
zneni = …
jakoco = 0

Hodnoty byly uvedeny pouze symbolicky, ve skutečnosti má třeba objekt `kde` strukturu složitější (viz např. obrázek 4). Ukážeme si, jak se podle něj provedou změny v předpisu. Věty v souvětí se vyhodnocují zvlášť. Nejprve najdeme místo, zpracujeme tedy objekt `kde`. Vidíme, že nemáme informaci o čísle předpisu, použijeme tedy číslo z předchozí věty. Nyní najdeme příslušný předpis a v něm funkcí `najdi_blok()` nalezneme § 5. Tím dostaneme hodnotu, typu `Misto`, která nám určuje pozici tohoto paragrafu. Na toto místo aplikujeme znovu funkci `najdi_blok()`, nyní ovšem hledáme odstavec 10, znovu dostaneme hodnotu typu `Misto`. Takto postupujeme dále a najdeme příslušné písmeno. Nyní se podíváme na hodnotu proměnné `akce`, ta je `'nahradit'`, proto ignorujeme objekt `kam`. Objekt `co` je „tečka“, hledáme tedy na místě, které jsme nalezli pomocí objektu `kde`, všechny tečky. Protože ale toto místo má vlastnost `'nk'` (na konci), vezmeme pouze tu poslední tečku. Nyní se už stačí jen podívat, že objekt `cim` je „čárka“, provedeme příslušnou změnu a jsme hotovi.

Druhá věta má prázdný objekt `kde`. Proto budeme informace o umístění dědit z věty předchozí. Co se ale všechno zdědí? Podíváme se na podmět věty `co`, ten je „písmeno“, proto zdědíme pouze hodnoty vyšších bloků, tj. číslo předpisu, paragraf a odstavec. Odstavec v příslušném předpisu vyhledáme stejně jako u předchozí věty. Hodnota `akce` je teď `'vlozit'`, podíváme se na objekt `kam`, ten říká „na konec“. Co budeme na konec vkládat určuje podmět `co` a pokud tento objekt neobsahuje znění, podíváme se ještě do proměnné `zneni`. Teď již můžeme vložení provést.

# 4 Zhodnocení dosažených výsledků

## 4.1 Způsoby testování programu

Jak můžeme testovat správnou funkčnost tohoto programu? Způsobů je několik. Tím nejintuitivnějším je to jednoduše opticky zkontrolovat: Otevřít si původní znění, novelu a vygenerované aktuální znění a větu po větě kontrolovat, jestli se změny zapsané v novele provedly. V tomto případě srovnáváme činnost programu subjektivně, tedy jak bychom my příslušný předpis aktualizovali. Je zřejmé, že tento způsob je velmi pracný a nemůžeme jím kontrolovat rozsáhlejší data. Nevýhodou je také to, že bychom správně měli kontrolovat aktuální znění předpisu po každé provedené změně. Často se stává, že se třeba něco v nějakém odstavci změní a hned v následující větě je pokyn, aby se všechny odstavce přečíslovaly. Toto pak ztěžuje následnou kontrolu. Velmi obtížná je pak kontrola správného znění předpisu až po několika novelizacích. Tento test je ale z hlediska určení správné funkce programu nejúčinnější. Odhalí totiž chyby v programu, nikoli chyby ve vstupních datech.

Po přečtení každé novely program vypíše svoji vlastní statistiku. Zobrazí se, kolik vět bylo programem čteno a kolika větám z toho program porozuměl a provedl je. Přirozeně, to, že program ohodnotí nějakou větu jako úspěšně provedenou, neznamená, že opravdu udělal to, co měl. Je to tedy horní odhad úspěšnosti programu. Ve většině případů neúspěšná věta znamená, že ji negeneruje gramatika. Může to ale být i tím, že se mu dané místo v předpise vůbec nepodařilo najít. Buď určil místo nesprávně, nebo byla v předpise chyba. Často se například přečíslovávají odstavce. Pokud toto nebylo někdy dříve správně provedeno, program hledá samozřejmě jinde, než by měl, a příslušnou věc nenajde.

Dalším způsobem testování je porovnání aktuálního znění vygenerovaného programem s úplným zněním uveřejněným někde na webu. S tím ale souvisí opět několik zásadních problémů. Na konci každého zákona jsou informace o účinnostech jeho jednotlivých částí a různá přechodná ustanovení, která často používají slovník vztahující se k tématu daného zákona a program je nedokáže přečíst. Jsou ve zvláštních paragrafech a často tvoří celou závěrečnou část novely. V úplných zněních vydaných sněmovnou jsou pak tyto informace zcela odlišné, vztahují se totiž už k jinému datu. V žádné novele nebylo nikdy napsáno, jak budou tyto paragrafy znít, program je tedy nemůže správně vygenerovat. Řešením tohoto problému je jednoduše všechny tyto paragrafy a dodatky smazat a porovnávat pouze ty části předpisu, které by se měly opravdu shodovat. Stejně tak se vymažou hlavičky předpisů a podpisy.

Dalším problémem je to, že servery nabízející aktuální znění předpisů už nenabízejí původní znění a naopak. Různé webové servery používají různý formát předpisů a často odlišné značení. Je tedy třeba vytvořit nový skript pro převádění dalšího formátu do `*.zkn` a znovu ručně opravit všechny chyby. Trochu pohodlnější bude vůbec

všechny tagy z obou souborů vymazat a porovnávat pouze prostý text. Ani tady se ale nevyhneme upravování uvozovek, tabulek a některých značení, jako je třeba mezera u označování paragrafů, např. „§5“ místo „§ 5“, nebo psaní čísla paragrafu před každým odstavcem, např. „§ 5 (2)“ místo pouhého „(2)“.

## 4.2 Výběr předpisů a výsledky testů

Pro účely testování vybereme několik předpisů z posledních let a aktualizujeme je všemi novelami, které do dneška vyšly. Program budeme pouštět postupně po jednotlivých novelách, a pokaždé zhodnotíme výsledek. Budeme se zajímat o počet rozpoznaných vět a vždy všechny změny opticky zkontrolujeme a určíme počet správně provedených. Je zřejmé, že úspěšnost bude se zvyšujícím se počtem novelizací klesat, protože se budou v předpisu postupně kumulovat chyby. Budeme rozeznávat několik typů chyb:

1. Program nerozumí větě novely, protože tato není v gramatice. Většinou jde o neobvyklá slovní spojení nebo o slova vztahující se spíše k tématu předpisu.

2. Program větě novely rozumí (nebo si to myslí), ale nemůže najít relevantní místo v zákoně. Buď neexistuje příslušně označený blok (paragraf, odstavec, písmeno), nebo existuje, ale není v něm text, který se má najít. Nejčastějším důvodem této chyby jsou chybná vstupní data, například text, který hledáme, neodpovídá přesně textu, který najít chceme (chybí nebo přebývá čárka, mezera, středník apod.). Dalším důvodem jsou chyby vyplývající již z předchozí špatné novelizace. Méně často se stává, že program špatně analyzuje větu a hledá někde jinde, než má.

3. Program rozumí větě novely, najde příslušné místo v zákoně, ale opraví ho špatně. Opět můžou být příčinou chybná vstupní data. Často se také stává, že se nesprávně vyhodnotí slova typu „na konec“, „na začátku“, „za“ apod. Program například neví, zda něco vložit před tečku nebo za tečku, před odkaz na poznámku nebo za něj, kam konkrétně do odstavce něco vložit, pokud to není nijak blíže specifikováno, atd.

Nejprve budeme testovat zákon č. 586/1992 Sb., o daních z příjmů. Byl vybrán především kvůli jeho poměrně rozsáhlé členitosti a množství novel. Základem bude jeho úplné znění z roku 2004 a budeme ho aktualizovat všemi jeho novelami, které vyšly do dubna 2006. Tento zákon byl upraven podle našich potřeb. Byly z něj vyjmuty změny dalších souvisejících zákonů, protože tyto stejně nemáme k dispozici. Podobně ve všech novelách, měnících i jiné zákony, byly tyto změny vymazány. Program by to samozřejmě zvládl, ale vypisoval by množství chyb, že příslušné předpisy nenašel. V následující tabulce je vždy za číslem novely uveden celkový počet vět, počet vět, kterým program porozuměl a správně příslušný příkaz vykonal, v následujících třech sloupcích pak počty chybně přečtených vět, rozdělené podle jednotlivých typů chyb. Úspěšnost je pak podíl správně vykonaných vět ku počtu všech, vyjádřený v procentech.

| novela | počet vět | | | | | úspěšnost |
|---|---|---|---|---|---|---|
| | celkem | správně | chyba 1 | chyba 2 | chyba 3 | |
| 19/2004 | 2 | 2 | 0 | 0 | 0 | 100% |
| 47/2004 | 10 | 10 | 0 | 0 | 0 | 100% |
| 49/2004 | 2 | 2 | 0 | 0 | 0 | 100% |
| 257/2004 | 6 | 6 | 0 | 0 | 0 | 100% |
| 280/2004 | 30 | 29 | 0 | 0 | 1 | 97% |
| 359/2004 | 5 | 5 | 0 | 0 | 0 | 100% |
| 360/2004 | 5 | 5 | 0 | 0 | 0 | 100% |
| 436/2004 | 10 | 10 | 0 | 0 | 0 | 100% |
| 562/2004 | 5 | 5 | 0 | 0 | 0 | 100% |
| 628/2004 | 5 | 5 | 0 | 0 | 0 | 100% |
| 669/2004 | 174 | 167 | 3 | 2 | 2 | 96% |
| 676/2004 | 2 | 2 | 0 | 0 | 0 | 100% |
| 179/2005 | 4 | 4 | 0 | 0 | 0 | 100% |
| 217/2005 | 2 | 2 | 0 | 0 | 0 | 100% |
| 342/2005 | 6 | 6 | 0 | 0 | 0 | 100% |
| 357/2005 | 3 | 3 | 0 | 0 | 0 | 100% |
| 441/2005 | 5 | 5 | 0 | 0 | 0 | 100% |
| 530/2005 | 2 | 2 | 0 | 0 | 0 | 100% |
| 545/2005 | 201 | 192 | 4 | 4 | 1 | 96% |
| 56/2006 | 15 | 15 | 0 | 0 | 0 | 100% |
| 57/2006 | 4 | 4 | 0 | 0 | 0 | 100% |
| celkem | 498 | 481 | 7 | 6 | 4 | 97% |

*Tabulka 1: Testování programu zákonem č. 586/1992 Sb.*

Dále provedeme test na dvou dalších předpisech s poměrně malým množstvím novelizací. Konkrétně jde o zákon 120/2002 Sb., o podmínkách uvádění biocidních přípravků a účinných látek na trh a o změně některých souvisejících zákonů, a zákon 440/2003 Sb., o nakládání se surovými diamanty, o podmínkách jejich dovozu, vývozu a tranzitu a o změně některých zákonů.

| novela | počet vět | | | | | úspěšnost |
|---|---|---|---|---|---|---|
| | celkem | správně | chyba 1 | chyba 2 | chyba 3 | |
| 186/2004 | 4 | 4 | 0 | 0 | 0 | 100% |
| 125/2005 | 36 | 35 | 1 | 0 | 0 | 97% |
| celkem | 40 | 39 | 1 | 0 | 0 | 98% |

*Tabulka 2: Testování programu zákonem č. 120/2002 Sb.*

| novela | počet vět | | | | | úspěšnost |
|---|---|---|---|---|---|---|
| | celkem | správně | chyba 1 | chyba 2 | chyba 3 | |
| 60/2005 | 84 | 81 | 3 | 0 | 0 | 96% |
| 413/2005 | 4 | 4 | 0 | 0 | 0 | 100% |
| 70/2006 | 3 | 2 | 1 | 0 | 0 | 67% |
| celkem | 91 | 87 | 4 | 0 | 0 | 96% |

*Tabulka 3: Testování programu zákonem č. 440/2003 Sb.*

Nyní se pokusíme porovnat námi vygenerovaná úplná znění se skutečnými. Porovnávat budeme pouze texty, které si mají odpovídat. Vymažeme tedy u obou souborů úvodní a závěrečné části (např. účinnosti, přechodná ustanovení, podpisy). Pro lepší srovnávání pak dáme každé slovo na zvláštní řádek a prázdné řádky vymažeme. K porovnání obsahů použijeme nástroj Diff, jako první parametr zadáme soubor vygenerovaný programem (označme ho G), jako druhý pak skutečné znění (označíme S). Diff vypisuje na standardní výstup řádky, které se v obou souborech liší. Znakem < uvozuje řádky, které jsou v souboru G navíc oproti S, znakem > pak řádky, které v souboru G chybí. Vydělíme-li počet řádek, které jsou navíc, počtem řádek souboru G a odečteme od jedné, dostaneme číslo, které odpovídá přesnosti novelizace, označíme ho P (precision). Pokud vydělíme počet chybějících řádek počtem řádek souboru S a odečteme od jedné, dostaneme číslo odpovídající úplnosti novelizace, označme ho R (recall). Dále spočteme harmonický průměr obou těchto čísel:

$$H = \frac{2PR}{P+R}$$

Ten se počítá proto, abychom měli jediné číslo a mohli mezi sebou srovnávat jednotlivé novelizace. V následující tabulce jsou výsledky testu pro naše 3 zkušební předpisy.

| číslo předpisu | počet novel | P | R | H |
|---|---|---|---|---|
| 586/1992 | 21 | 0,9850 | 0,9796 | 0,9798 |
| 120/2002 | 2 | 0,9931 | 0,9967 | 0,9949 |
| 440/2003 | 3 | 0,9375 | 0,9658 | 0,9515 |

*Tabulka 4: Porovnávání vygenerovaných předpisů se skutečným zněním*

Je vidět, že zákon č. 440/2003 Sb. si vedl v tomto testu poměrně špatně. Hlavní příčinou je, že se při novelizaci nerozpoznaly 2 zásadní věty, které zrušují několik paragrafů. Naopak zákon č. 120/2002 Sb. se téměř shoduje. Jedinou chybou todiž bylo nepřeznačení nějakých odkazů na poznámky. Ve skutečnosti by měla být úspěšnost tohoto testu o něco vyšší. Vstupní soubory totiž opět nebyly bez chyb.

## 4.3 Nejčastěji se vyskytující chyby

Nejprve se zaměříme na chyby, na kterých nese vinu výhradně program, nikoli chyby ve vstupu. Tyto chyby by šly vhodným rozšířením programu postupně všechny ošetřit:

- Problém správné analýzy příslovečných určení místa, rozdíl mezi odpověďmi na otázky *kde?* a *kam?*. Například větu „Na konec odstavce 2 se doplňuje věta…“ program vyhodnotí správně. Naopak s větou „Na konci odstavce 2 se doplňuje věta…“ má problémy, protože nenašel přesnou pozici v předpisu, kam příslušnou větu vložit. Slova „na konci“ značí, že se bude něco hledat na konci, nikoli že se něco vloží na konec. Naproti tomu se „na konci“ nemůže vyhodnocovat stejně jako „na konec“, protože třeba „slovo na konci“ nemusí být nutně na konci, je za ním třeba ještě tečka, odkaz na poznámku pod čarou apod. Pokud není zřejmé, kam se má daná věc přesně vložit, vkládá se automaticky na konec příslušného bloku. Příkladem je věta „V § 12 se doplňuje věta…“.

- Vezměme větu „Slovo „a“ se nahrazuje středníkem a doplňují se slova…“. Zde programu není jasné, kam ta slova má doplnit. Doplní je tedy nakonec, ačkoliv se předpokládá že budou vloženy za zmiňovaný středník. Před středníkem navíc pouhým nahrazením vznikne nežádoucí mezera, která pak může následujícímu vyhledávání v textu vadit.

- Ne vždycky se podaří správně vyhodnotit komplikovanější rozvitá příslovečná určení typu „v § 8 odst. 2, v písmenech b) a d) odstavce 3 a v § 9 v nadpise a odstavci 1“. Často jsou nejednoznačná a je tedy nutné pracovat i s významem slov, někdy i s příslušným předpisem, aby bylo zřejmé, která místa tento popis vlastně označuje.

Druhým typem, jsou chyby, které vznikly příčinou chybných vstupů nebo špatnou formulací vět:

- Při nahrazování části textu jiným nebo při rušení textu se často nepodaří tento text najít, protože úplně nesouhlasí s hledaným textem. Většinou jde o chybějící nebo přebývající čárky nebo mezery. Například „a nebo“ a „anebo“, „(§ 35)“ a „( § 35 )“, někdy se hledá před čárkou mezera, která tam samozřejmě být nemá, někdy je odkaz na poznámku pod čarou před čárkou, jindy za čárkou, apod.

- Často se ruší paragrafy včetně nadpisů. Někdy se ovšem stává že nadpisy nejsou součástí paragrafů a tyto tam pak zůstávají.

- Slova za čárkou se nahrazují slovy, která začínají čárkou. Pak jsou tam čárky dvě. Stejně tak se někde vyskytnou dvě tečky na konci věty.

- Máme například větu: „Na konec odstavce 1 se vkládá věta…“. Tento odstavec se skládá z úvodní části ustanovení a písmen a) až d). Program tedy tuto větu vezme a vloží ji na konec odstavce těsně před ukončující tag `</ods>`. V úplném znění je ale tato věta kupodivu součástí posledního písmena d). Přitom jsou tyto věty v závěru odstavců po nějakém výčtu naprosto běžné.

# 5 Závěr

Ačkoli se na první pohled zdá, že se v novelách zákonů vyskytuje pouze pár typů vět, které půjdou snadno ošetřit, je problém automatické novelizace poměrně složitý. I přesto ale bylo dosaženo úspěšnosti vyšší než 95%, a to jak z hlediska počtu správně zpracovaných vět, tak i z hlediska odlišností vygenerovaného předpisu od jeho skutečného znění. Je třeba podotknout, že tento problém nejvíce ztěžovalo získávání korektních vstupních dat. Bez přístupu k bezchybným a správně formátovaným předpisům nelze tento program v praxi používat, protože samotná úprava vstupních dat a kontrola správného provedení zabere téměř tolik času, za který by se dal příslušný zákon změnit ručně. Nejvhodnější by bylo získat přístup přímo ke zdrojovým souborům, ze kterých se následně generují originální PDF soubory. Bohužel ani ty ale nejsou zcela bez chyb.

Tento projekt by šel dále rozšiřovat a zdokonalovat. Jednalo by se zejména o uživatelsky příjemnější grafické prostředí, kde by například uživatel viděl ve třech oknech původní znění, novelu a nově generované znění, mohl by do chodu programu zasahovat, pozastavit a rychleji ho tak kontrolovat, případné špatně provedené změny vracet a editovat je třeba ručně. Jednoduché grafické prostředí by se dalo vyrobit i pomocí dynamicky generovaného HTML na webovém serveru.

Dále by šlo použít některých efektivnějších algoritmů pro syntaktickou analýzu vět. Zřídka se totiž objevují v novelách dlouhé členité věty, třeba i s více než stovkou čárek, které použitý algoritmus nezvládá.

Samozřejmostí by bylo i rozšíření gramatiky, robustnější analýza a zotavení se z některých chyb v datech. Například pokud někde chybí mezera za čárkou nebo naopak přebývá mezera před čárkou, program by se s tím měl umět vyrovnat. Měl by zamezit možnosti výskytu dvou teček nebo dvou čárek za sebou apod.

Dále připadá v úvahu možnost získat znění zákona platné k určitému datu. To vyžaduje nejen filtrování novel, ale hlavně porozumění paragrafům o nabytí účinnosti. V některých případech to nepůjde dotáhnout do konce, protože paragrafy o účinnosti budou používat slovník vztahující se k tématu daného zákona (např. „bod 4 se poprvé uplatní ve zdaňovacím období 2005“), v tom případě by však program mohl být schopen alespoň upozornit u každého bodu na všechna ustanovení, která se na něj odkazují a nějak ovlivňují jeho účinnost. Totéž by šlo ještě rozšířit na návaznost celých zákonů, nejen novel. Např. zákon o daních z příjmu existuje od roku 1992, ale před ním tuto problematiku upravoval jiný zákon, který půjde vystopovat, protože zákon z roku 1992 ten předchozí zákon pravděpodobně zrušuje.

# Literatura

[1] Chytil M.: *Automaty a gramatiky*, SNTL, Praha, 1984.

[2] Kolektiv autorů: *Sbírka zákonů České republiky*, Praha, 1993-2006.