Mendelova univerzita v Brně
Provozně ekonomická fakulta

Detekcia a korekcia typografických javov v značkovanom texte

Diplomová práca

Bc. Dominik Makeš

Vedúci práce: RNDr. Tomáš Hála, Ph. D.

Brno 2015

NA MÍSTĚ TOHOTO LISTU
SE NACHÁZÍ ORIGINÁL
ZADÁNÍ PRÁCE

Chcel by som poďakovať RNDr. Tomášovi Hálovi, Ph. D., ktorý mi umožnil vypracovávať prácu, ktorá ma zaujala, a poskytol mi odborné vedenie, konzultácie a rady, ktoré mi prácu zjednodušili.

Čestné prehlásenie

Prehlasujem, že som prácu *Detekcia a korekcia typografických javov v značkovanom texte* vypracoval samostatne a všetky použité zdroje a informácie uvádzam v zozname použitej literatúry. Súhlasím, aby moja práca bola zverejnená v súlade § 47b zákona č. 111/1998 Sb., o vysokých školách, v znení neskorších predpisov a v súlade s platnou Směrnicí o zveřejňování vysokoškolských závěrečných prací. Som si vedomý, že sa na moju prácu vzťahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzatvorenie licenčnej zmluvy a použitie tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona. Ďalej sa zaväzujem, že pred spísaním licenčnej zmluvy o použití diela inou osobou (subjektom) si vyžiadam písomné stanovisko univerzity, že predmetná licenčná zmluva nie je v rozpore s oprávnenými záujmami univerzity a zaväzujem sa uhradiť prípadný príspevok na úhradu nákladov spojených so vznikom diela, a to až do ich skutočnej výšky.

V Brne dňa 17. mája 2015

.....

podpis

Abstract

Makeš, Dominik. *Detection and correction of typography in structured text.* Diploma thesis. Brno, 2015.

In this thesis, I am dealing with an design and implementation of finite deterministic automaton. Its role is to process the ConTEXt input file and make a decision about its typography correctness. In next step, it will offer the recommendation how to correct the mistake and will be able to make simple automatic correction. Thesis is also including the comparision with existing programs and the tests on real documents. Thesis is also including theoretical subscription of formal languages and finite automaton.

Key words

automatic correction, ConTEXt, detection, deterministic finite automaton, regular language, typography

Abstrakt

Makeš, Dominik. *Detekcia a korekcia typografických javov v značkovanom texte.* Diplomová práca. Brno, 2015.

V tejto práci sa zaoberám návrhom a implementáciou konečného deterministického automatu. Jeho úlohou je spracovať vstupný text nástroja ConTEXt a vyhodnotiť správnosť typografie. K nájdených chybám ďalej navrhne riešenie a umožní jednoduchú automatickú korekciu. Súčasťou je aj porovnanie s existujúcim programovým vybavením a testy na reálnych súboroch. Práca obsahuje aj teoretický popis danej problematiky z oblasti formálnych jazykov a konečných automatov.

Kľúčové slová

automatická korekcia, ConTEXt, detekcia, deterministický konečný automat, regulárny jazyk, typografia

Obsah

1 Úvod a cieľ práce

1.1 Úvod

Denne sa vyprodukuje enormné množstvo informácií, dát, textov a mnoho iného. Pri súčasnom trende môžeme s trochou nadhľadu povedať, že každým dňom prekonáme rekord v produkcii informácií z predošlého dňa. Aby sme však informácie ľahšie pochopili a spracovali, je potrebné ich zaznamenať v podobe jasnej pre všetkých. Jedným zo spôsobov je text v elektronickej podobe, s ktorou sa stretávame každodenne. Rovnako ako písaný text, ktorý sme sa naučili na pravopise, aj elektronický text má však svoje pravidlá. Tieto pravidlá sa pre jednotlivé národy líšia rovnako ako ich jazyk alebo písmo. Z tohto mora pravidiel, si v tejto práci vyberám tu našu malú kvapku. Český a slovenský jazyk sú si v mnohom podobné, dalo by sa ich označiť za súrodencov. Pre ich typografické pravidlá platí takmer presne to isté.

Práve pravidlá pre sadzbu elektronických textov, presnejšie správny zápis typografických javov, bude témou tejto práce. Každý z nás už v tejto digitálnej dobe určite písal nejaký článok, prácu, projekt alebo len jednoduchú správu kamarátovi prostredníctvom počítača. Keďže mýliť sa je ľudské, určite sa v týchto textoch objavili aj chyby. Veď koľkokrát sa stane, že aj po niekoľkom prečítaní, stále po sebe nachádzame nové a nové chyby, preklepy a podobne. Niektoré si však nevšimneme vôbec, prípadne až keď je neskoro. Takisto sa často stáva, že chybu ktorú ja prehliadnem desaťkrát, niekto iný nájde na prvý pohľad. Práve tento druhý pohľad v podobe pohľadu počítača prináša táto práca.

1.2 Cieľ práce

Cieľom tejto diplomovej práce je v prvej fáze vytvoriť nástroj, ktorý dokáže analyzovať vstupný štruktúrovaný text a vybrať z neho časti, vhodné pre syntaktickú analýzu. Analyzátor vstupného textu by teda mal vhodne rozlíšiť, ktoré časti súboru sú súčasťou značiek nástroja ConTEXt a ktoré naopak možno považovať za prostý text. Toto delenie je veľmi dôležité a je potreba podrobne analyzovať samotný nástroj, pretože na značky ConTEXt-u, by sa nemala vzťahovať detekcia typografických chýb a už vôbec nie jej korekcia. Dôvodom je, že napríklad parametre príkazov majú striktne predpísané pravidlá, ktoré sa výraznou mierou líšia od typografických pravi-

diel v prostom, českom texte. Takéto úpravy by v konečnom dôsledku mohli narušiť formátovanie textu, exportovaného nástrojom ConTEXt alebo v horšom prípade narušiť syntax súboru tak, že nebude vôbec preložiteľný.

Druhá fáza, do ktorej budú vstupovať útržky prostého textu, má za úlohu previesť detekciu typografických chýb a podať správu o chybách, vrátane návrhu riešenia opravy. V prípade, že užívateľ s návrhom súhlasí, program by mal byť schopný v prijateľnej miere aplikovať opravy na pôvodný text a jeho výstupom by mal byť druhý súbor s opravenou typografiou.

Javy, ktoré bude program schopný rozoznávať a hľadať v nich chyby zahŕňajú nasledovnú množinu interpunkcie:

- Bodka .
- Čiarka ,
- Bodkočiarka ;
- Výkričník !
- Otáznik ?
- Zátvorky ( )
- České úvodzovky „“ ‚‘
- Dvojité medzery (vrátane nezalomiteľnej)
- Spojovník a pomlčka - a –
- Odsuvník (apostrof) ’

Množstvo funkcií a prípadov chovania by však malo byť možné ovládať a vhodne prispôsobovať, preto bude nedeliteľnou súčasťou aj konfiguračný súbor, v ktorom bude napríklad možné dopĺňať vlastné makrá ConTEXtu, ako aj prispôsobovať, ktoré časti budú podliehať formátovaniu a ktoré nie, prípadne ako sa má prejavovať oprava v nerozhodných situáciách (napr. sekvenciu dvoch bodiek nahradiť jednou alebo výpustkom, či syntakticky analyzovať komentáre a podobne).

Dôležitá bude aj samotná dokumentácia programu, ktorá bude obsahovať popis fungovania samotného programu, ale hlavne možnosti, ktoré má užívateľ nastavovať (prevažne v konfiguračnom súbore).

2 Metodika práce

Práca bude pozostávať z dvoch základných častí – teoretickej a praktickej. Teoretická časť sa bude zaoberať teóriou formálnych jazykov, kde budú vysvetlené základné pojmy k tejto problematike, ale aj podrobnejšie rozpracovanie klasifikácie jazykov, konkrétne jazykov typu 3, ako aj následný postup pri návrhu automatu a jeho implementácii.

Teoretická časť bude ďalej obsahovať základné informácie o nástroji pre sadzbu textu ConTEXt, ktorý je rozšírením systému TEX. Okrem popisu základných vlastností, bude dôležité zistiť v akých situáciách sa môže v súbore nachádzať prostý text, ale aj množiny javov, ktoré môžu nastať po príkazoch ConTEXt-u (je dôležité poznať aký typ zátvoriek, prípadne koľko ich môže nasledovať za konkrétnym príkazom, aby sme vedeli rozhodnúť, či sa jedná o parametre príkazu alebo už program číta text, ktorý podlieha detekcii javov). Súčasťou preto bude aj výpis všetkých situácií, ktoré môžu nastať pri náleze príkazu nástroja ConTEXt.

Poslednou časťou teórie bude porovnanie podobných nástrojov, zameraných na detekciu a korekciu typografie. Keďže pre nástroj ConTEXt žiadny podobný nástroj neexistuje, bude toto porovnanie zamerané všeobecne na nástroje, ktoré umožňujú detekciu a korekciu. Téme tejto práce najbližším, je nástroj LaCheck, ktorý je určený pre inú nadstavbu nástroja TEX – je ním LATEX. Okrem LaChecku aj funkcie programu MS Word, Ispell, Ikor, ale aj programu Vlna (vlnka), ktorý je určený pre prostredie TEX-u.

V praktickej časti bude zdokumentovaný celý postup práce. Na základe analýzy ConTEXtu bude stanovená regulárna gramatika, ktorá bude realizovaná konečným automatom. Jeho výstupom budú úseky prostého textu zo vstupného súboru vo formáte „.tex“. Tieto úseky textu, budú ďalej vstupom pre funkciu na opravu typografických javov, ktorá bude vo vstupnom reťazci hľadať typografické chyby a opravovať ich na základe stanovených pravidiel. Táto funkcia opätovne vráti upravený reťazec automatu, ktorý ním nahradí pôvodný vstup a zapíše ho do výstupného súboru. Súčasťou funkcie bude aj matica so súradnicami jednotlivých chýb a ich popisom. Tá bude neskôr použitá pri návrhu možných opráv.

Automat bude implementovaný v jazyku Lua 5.3.0, čo je v súčasnosti aktuálna verzia tohto programovacieho jazyka z 6. 1. 2015. Jazyk Lua je procedurálny, imperatívny jazyk, ktorý poskytuje široké možnosti práce s reťazcami a od verzie 5.3 aj priamu podporu pre prácu s UTF-8, čo je oproti kódovaniu v ASCII veľkou výhodou pri práci s českými znakmi a odstraňuje tým nutnosť využívania knižníc pre UTF-8, ako tomu bolo v predošlých verziách.

Pri práci bude využité vývojové prostredie Zero Brane Studio 1.0, ktoré ponúka široké možnosti a podporu pri programovaní v jazyku Lua. Toto prostredie posky-

tuje automatické dopĺňanie kódu (resp. jeho návrhy), farebné zvýrazňovanie syntaxe, analyzátor kódu vrátane debuggeru, ale aj „*live coding*“, teda priebežnú kontrolu kódu a „*debugging*“. Z týchto dôvodov je najvhodnejším prostredím pre pohodlný a značne zjednodušený vývoj, čo boli hlavné kritériá pre výber programu.

Okrem spomenutých treba uviesť aj to, že pri tvorbe, kontrole, ale aj prevodoch a zapisovaní grafov jednotlivých gramatík, bol využitý nástroj Desátomat (Dusíková, 2012). Jedná sa o online nástroj slúžiaci práve k týmto účelom, preto bude efektívne pracovať s ním. Ušetrí to pri tvorbe práce nemálo času a zaručí správnosť vytvorených gramatík, prípadne jednoduchú opravu.

Okrem softwaru, ktorý bol uvedený, je vhodné v krátkosti uviesť aj prostriedky a metódy, ktoré budú využité na dosiahnutie cieľa. Bude sa jednať o tri základné bloky programu. Prvý, bude konečný automat, ktory prijíma vstupnú pásku (vstupný súbor). Jeho úlohou je rozdeliť vstupný súbor na značkované a neznačkované časti na základe riadiacich znakov. Časti, ktoré označí za neznačkované (Implementované pomocou string bufferu – každý znak reťazca má svoju bunku v poli, čím sa vyhneme postupnému reťazeniu. To by spôsobilo zvýšenú pamäťovú aj časovú náročnosť behu programu.), budú ďalej vstupom pre druhý blok. Jedná sa o skupinu samostatných deterministických konečných automatov, ktoré v bufferi hľadajú chyby v typografických javoch. Každý typografický jav má vlastnú funkciu implementovanú vo forme rozhodovacieho stromu, ktorý jednoznačne rozhodne, či vstup patri do jazyka daného javu alebo nie. O všetkých nájdených chybách si zaznamená ich polohu v súbore a popis. Až po dokončení všetkých funkcií, sa string buffer prevedie na reťazec a zapíše do výstupného súboru. Po skončení čítania vstupného suboru sa vygeneruje subor s návrhmi riešení nájdených chýb. Obidva spomenuté bloky budu vo viacerých bodoch podliehať konfiguračnému súboru (vo formáte .lua, čo umožní jehoobsah priamo vložiťdo kódu a zjednodušiť tak jeho načítanie), v ktorom bude možné prispôsobiť beh programu potrebám užívateľa. Tým sa dostávame k tretiemu bloku programu, ktorý je nutné spustiť akonáhle sa užívateľ rozhodne, že všetky návrhy opráv sú vhodné, prípadne ich vhodne upravil alebo nežiadúce zakázal. Tento tretí blok, je samostatný program, ktorý na základe súboru s návrhmi opráv, nahradí v pôvodnom vstupnom súbore chybné riadky opravenými. Výstupom bude nový súbor s opravenými chybami vo formáte rovnakom ako vstupný súbor.

3 Teória formálnych jazykov

Táto kapitola bude zameraná na popis pojmov, ale aj súvislostí medzi nimi, ktoré súvisia s témou formálnych jazykov. Formálne jazyky, majú na rozdiel od prirodzených jazykov jednu významnú vlastnosť – sú konečné. Je to ich základnou podmienkou, pretože v prípade, že by množina prípustných slov bola nekonečná, bola by jeho implementácia nerealizovateľná. V práci nás bude zaujímať taký jazyk, ktorý má konečnú a zároveň neprázdnu množinu prípustných symbolov (abecedu), ďalej má konečnú množinu pravidiel (gramatiku), podľa ktorých skladáme z jednotlivých symbolov abecedy celé vety. Základné pojmy, spomenuté na predošlých riadkoch budú definované v nasledujúcej podkapitole. (Kocur, 2001)

3.1 Základy formálnych jazykov

Abeceda Σ (niekedy nazývaná aj slovník) je ľubovoľná neprázdna množina symbolov, ktorá musí byť konečná. Príkladmi môžu byť latinka {a, b, c, …, z}, grécka abeceda {α, β, γ, …, ω}, ale aj binárna abeceda {0, 1}. (Kocur, 2001)

Veta (nazývaná aj slovo alebo reťazec) je ľubovoľná konečná postupnosť znakov danej abecedy. Pri zápise viet nepoužívame oddeľujúce čiarky medzi jednotlivými prvkami postupnosti. Vetu w zapíšeme ako:

$$w = \{a_i\}_{i=1}^{n}$$
$$w = a_1 \dots a_n$$

Dĺžku vety označujeme $|w|$. (Kocur, 2001)

$$| \epsilon | = 0$$
$$| 0\ 1\ 10 | = 3$$

Iterácia množiny Σ (označujeme ju Σ^*) je množina všetkých konečných neprázdnych postupností utvorených z prvkov abecedy (Σ^+), vrátane prázdneho reťazca (ϵ). (Kocur, 2001)

$$\Sigma^* = \Sigma^+ \cup \{ \epsilon \}$$

Prázdna veta ϵ (slovo, reťazec) je taktiež považovaná za vetu. Je jej zvláštnym prípadom kedy postupnosť neobsahuje žiadny symbol. (Kocur, 2001)

Jazyk je ľubovoľná podmnožina reťazcov z Σ^*. Jeho definícia je:

$$L \subseteq \Sigma^*$$

L nazveme **jazykom** nad abecedou Σ alebo jazykom v abecede Σ. V prípade, že má jazyk konečnú množinu slov, nazývame ho konečným. Keď je naopak táto množina nekonečná, označíme tento jazyk za nekonečný a v prípade, že je množina slov jazyka prázdna, nazveme aj daný jazyk ako prázdny. (Kocur, 2001)

Keďže jazyk je množinou viet, Kocur (2001) uvádza, že môžeme s jazykmi vykonávať aj množinové operácie – zjednotenie, prienik, rozdiel a doplnok. Okrem nich môžeme jazyk aj reťaziť. Formálny zápis potom vyzerá nasledovne:

- $L_1 \cup L_2$ (zjednotenie) – jazyk obsahujúci všetky slová patriace do L_1 alebo do L_2, prípadne do oboch;
- $L_1 \cap L_2$ (prienik) – jazyk obsahujúci len slová, ktoré patria zároveň do L_1 aj L_2;
- $L_1 - L_2$ (rozdiel) – jazyk obsahujúci všetky slová z L_1, nepatriace do jazyka L_2;
- $\Sigma^* - L_1$ (doplnok) – jazyk obsahujúci všetky slová z Σ^* (množina všetkých prípustných slov), ktoré ale nepatria do jazyka L_1.

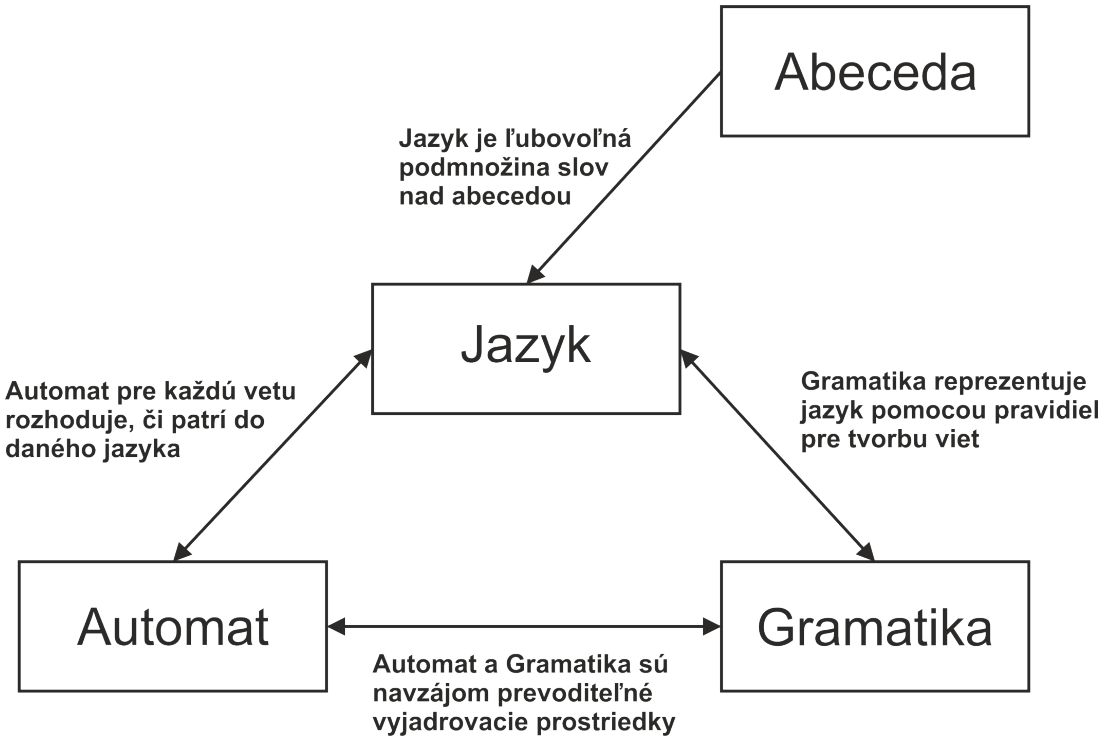
Jazyk môžeme podľa Kocura (2001) špecifikovať (definovať) niekoľkými základnými spôsobmi:

- spoločnou vlastnosťou (vymenovaním všetkých slov);
- automatom (akceptorom) – spôsob využiteľný u regulárnych jazykov, ktorý spracováva reťazec a na základe konečného počtu krokov stanoví, či slovo patrí alebo nepatrí do daného jazyka – toto slovo buď akceptuje, alebo zamieta;
- regulárnymi výrazmi – vhodné len pre regulárne jazyky;
- generatívnym popisom (gramatikou) – jedná sa o najčastejšie používanú metódu špecifikácie jazyka, kedy je systém schopný vygenerovať všetky slová daného jazyka. Považuje sa za univerzálny spôsob špecifikácie formálneho jazyka.

3.2 Formálne gramatiky

Ako uvádza Chytil (1984), gramatika je jeden zo spôsobov špecifikácie formálnych jazykov. Jedná sa o najrozšírenejší a univerzálny spôsob špecifikácie, pretože poskytuje možnosť vygenerovať všetky slová formálneho jazyka. Gramatiku jednoznačne definuje štvorica G = (N, Σ, P, S) kde:

- N udáva množinu neterminálnych symbolov, ktoré sa používajú k označe-



Obrázok 3.1 Vzťah medzi základnými provkami teŕie formálnych jazykov (Rybička, 1999)

niu syntaktických celkov. Označujeme ich obvykle veľkými latinskými písmenami.

- Σ udáva množinu terminálnych symbolov, ako už označenie napovedá, jedná sa vlastne o abecedu nad ktorou je jazyk definovaný. Terminálne symboly označujeme malými písmenami. Terminálne a neterminálne symboly spolu tvoria slovník gramatiky alebo celkovú abecedu gramatiky ($V = N \cup \Sigma$).
- P je konečná množina prepisovacích pravidiel, ktorej definícia znie:

$$P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^* = (V^* N V^*) \times V^*$$

Prepisovacie pravidlo sa skladá z dvojice slov, pričom prvé slovo (na ľavej strane) prepisovacieho pravidla musí obsahovať aspoň jeden neterminálny symbol. Slová zložené z terminálnych (malé písmená latinky) a neterminálnych symbolov (veľké písmená latinky) budeme označovať písmenami gréckej abecedy. Dvojicu slov $<\alpha, \beta>$ z P zapisujeme $\alpha \rightarrow \beta$ a nazývame prepisovacím pravidlom. Prepisovacími pravidlami nazývame také dvojice slov α, β, u ktorých musí platiť, že pravidlo α musí obsahovať aspoň jeden neterminálny symbol, teda $\alpha \in (N \cup \Sigma)^+$; $\beta \in (N \cup \Sigma)^*$. Ďalej platí, že pravidlá, ktoré majú

rovnakú ľavú stranu, v tomto prípade slovo α, zjednodušene zapisujeme ako: $\alpha \to \beta_1 \mid \beta_2 \mid \beta_3$ namiesto pôvodných troch pravidiel $\alpha \to \beta_1$; $\alpha \to \beta_2$; $\alpha \to \beta_3$ Zápis je v takomto prípade stručnejší a prehľadnejší.
- S je štartovací symbol gramatiky, pričom $S \in N$. Nazýva sa koreň gramatiky.

Názornejšie vysvetlenie fungovania gramatiky bude na príklade. Majme teda krátku gramatiku s pravidlami:

$S \to aB \mid bA$
$A \to a \mid aS \mid bAA$
$B \to b \mid bS \mid aBB$

Takáto gramatika potom môže generovať napríklad slovo bbaaab. Jeho generovanie by vyzeralo nasledovne:

$$S \to^1 bA \to^2 bbAA \to^3 bbaA \to^4 bbaaS \to^5 bbaaaB \to^6 bbaaab$$

V tomto prípade sa štartovací symbol S v prvom kroku prepisuje na slovo bA, z ktorého sa ďalej neterminálny symbol a prepisuje na slovo bAA, pričom sa postupne tvorí slovo jazyka zložené z terminálnych symbolov.

3.3 Chomského klasifikácia formálnych jazykov

Chytil (1984) uvádza, že niektoré jazyky môžeme generovať aj gramatikami, ktoré majú prepisovacie pravidlá špeciálneho typu, čím sa myslí, že nevyužívajú všetky možností ponúkaných všeobecnou definíciou gramatiky. Má preto význam, deliť jednotlivé gramatiky podľa tvarov pravidiel, ktoré obsahujú. Najstaršou a zároveň najznámejšou klasifikáciou gramatík založenou na princípe delenia gramatík podľa typov obsiahnutých pravidiel je tzv. Chomského klasifikácia. Je to teda spôsob klasifikácie gramatík, ktorý definuje pravidlá pre zaradenie gramatiky do jedného zo stupňov klasifikácie. Keďže sa jedná o hierarchickú štruktúru, platí, že každá gramatika patriaca na určitý stupeň hierarchie, musí zároveň spĺňať aj podmienky všetkých predošlých stupňov. V tomto prípade napríklad gramatika typu 2, je zároveň gramatikou typu 1, ktorá bezpodmienečne musí byť aj gramatikou typu 0.

$$L0 \supset L1 \supset L2 \supset L3$$

Stupne Chomského klasifikácie sú podľa Češku a Rábovej (1988) štyri, sú označené číslami od nula po tri. Zdrojom pre tvorbu príkladov je dielo od Rybičku (1999).

Gramatika typu 0 – nazýva sa aj neobmedzená gramatika, ktorá generuje jazyk typu 0. V prípade gramatiky typu 0 nie sú na prepisovacie pravidlá kladené žiadne podmienky, okrem pravidla, že slovo α musí obsahovať aspoň jeden neterminálny symbol. Definícia je tak zhodná s definíciou samotnej gramatiky z čoho vyplýva, že každá gramatika je gramatikou typu 0.

$$\alpha \rightarrow \beta$$

kde $\alpha \in (N \cup \Sigma)^* \,.\, N \,.\, (N \cup \Sigma)^*$; $\beta \in (N \cup \Sigma)^*$

Jazyky typu 0 generované touto gramatikou sú rozpoznateľné Turingovým strojom.

Príklad:
Gramatika G = ({A, B}, {a, b}, P, A) s pravidlami P:
$A \rightarrow AbB \mid a$
$AbB \rightarrow baB \mid BAbB$
$B \rightarrow b \mid \epsilon$

Gramatika typu 1 – nazývaná kontextová gramatika. Týmto termínom nazývame gramatiky typu 0 pre ktoré platí, že každé pravidlo gramatiky má tvar:

$$\alpha X \beta \rightarrow \alpha \gamma \beta$$

kde $X \in N$; $\alpha, \beta \in (N \cup \Sigma)^*$; $\gamma \in (N \cup \Sigma)^+$; $\mid \gamma \mid \leq 1$, teda $\gamma \neq \epsilon$

Slovne by sa toto pravidlo dalo interpretovať ako podmienka, že neterminálny symbol X môže byť nahradený slovom γ len v prípade, že zostal jeho ľavý (α) a pravý (β) kontext nezmenený. Toto pravidlo zaisťuje, že výsledné generované slovo sa nikdy neskracuje – buď sa jeho dĺžka nemení, alebo predlžuje. Pravidlá gramatiky typu 1 ďalej nepripúšťajú existenciu pravidla, ktoré by prepisovalo neterminálny symbol na prázdne slovo ϵ. Jedinou výnimkou kedy je možné generovať prázdne slovo je prípad $S \rightarrow \epsilon$. Toto pravidlo umožňuje v prípade, že sa neterminálny symbol S nevyskytuje v žiadnom slove na pravej strane, generovať prázdne slovo daného jazyka. Gramatika tohto typu generuje jazyk typu 1, ktorý je akceptovaný lineárne obmedzeným automatom.

Príklad:
Gramatika G = ({A, S}, {0, 1, c}, P, S) s pravidlami P:
$S \rightarrow 0A1$
$0A \rightarrow 00A1$
$A \rightarrow c$

Gramatika typu 2 – označovaná aj ako bezkontextová gramatika, musí okrem podmienky, že je aj gramatikou typu 1, spĺňať aj pravidlo:

$$X \rightarrow \gamma$$

kde $X \in N$; $\gamma \in (N \cup \Sigma)^*$; $|\gamma| \leq 1$, teda $\gamma \neq \epsilon$

Toto pravidlo prináša oproti predošlému typu gramatiku zmenu v tom, že gramatika typu 2 nevyžaduje pri prepise neterminálneho symbolu X z ľavej strany na slovo γ prítomnosť kontextov α a β ako tomu bolo pri predošlom type gramatiky. Gramatika typu 2 ďalej umožňuje prítomnosť prepisovacieho pravidla $X \rightarrow \epsilon$. Neterminálny symbol tak môžeme prepísať na prázdny symbol (nie len štartovací ako tomu bolo u gramatiky typu 1). Platí to ale opäť len v prípade, že sa štartovací symbol S nevyskytuje na pravej strane žiadneho pravidla. Jazyky typu 2 rozpoznávajú zásobníkové automaty.

Príklad:
Gramatika G = ({S, A, B}, {a, b}, P, S) s pravidlami P:
$S \rightarrow aB \mid bA$
$A \rightarrow a \mid aS \mid bAA$
$B \rightarrow b \mid bS \mid aBB$

Gramatika typu 3 – označovaná aj ako lineárna gramatika, je najvyšším stupňom Chomského klasifikácie. Obsahuje prepisovacie pravidlá, ktoré obsahujú na pravej strane vždy terminálny symbol, ktorý sa nachádza vždy úplne vpravo alebo úplne vľavo slova. Regulárne gramatiky sa podľa polohy terminálneho symbolu delia na pravé regulárne gramatiky a ľavé regulárne gramatiky. Prepisovacie pravidlá potom majú nasledovný tvar:

Pravá lineárna gramatika:

$$A \rightarrow \omega B \text{ alebo } a \rightarrow \omega$$

kde $A, B \in N$; $\omega \in \Sigma^*$

Ľavá lineárna gramatika:

$$A \rightarrow B\ \omega \text{ alebo } a \rightarrow \omega$$

kde $A, B \in N$; $\omega \in \Sigma^*$

Gramatiky typu 3, generujú jazyky typu 3, resp. regulárne jazyky. Tie budú podrobnejšie rozobraté v nasledujúcej kapitole.

Príklad:
Gramatika G = ({A, B}, {a, b, c}, P, A) s pravidlami P:
$A \rightarrow aaB \mid ccB$
$B \rightarrow bB \mid \epsilon$

Na základe uvedených pravidiel klasifikácie gramatík podľa Chomského klasifikácie môžeme povedať, že zložitosť tried na jednotlivých stupňoch postupne klesá. S rastúcim stupňom tried v klasifikácii sa naopak zjednodušuje algoritmické zisťovanie niektorých vlastností formálnych jazykov. (Kocur, 2001)
Rybička (1999) uvádza príklady:

- Príslušnosť slova do jazyka, kedy pre dané slovo ω a gramatiku G zisťujeme, či $\omega \in L_G$ (teda či slovo ω patrí do množiny slov, resp. jazyka generovaného gramatikou G). Tento problém je riešiteľný od jazyka typu 1 vyššie.
- Ekvivalencia gramatík – snažíme sa zistiť, či platí, že dve zadané gramatiky G_1 a G_2 sú schopné generovať na rovnakom jazyku rovnakú množinu slov, inak povedané, či generujú rovnaký jazyk. Formálne zapísané ako $L_{G_1} = L_{G_2}$. Algoritmicky je tento problém riešiteľný len pre jazyky typu 3.

Z praxe môžeme povedať, že jazyky typu 0 a 1 sa využívajú v teórii vyčísliteľnosti a jazyky typu 2 a 3 v aplikovanej informatike. (Rybička, 1999)

3.4 Jazyky typu 3

Jazyky typu 3 sú jazykmi najvyššej triedy Chomského klasifikácie, pričom postačujú pre popis všetkých základných prvkov napríklad programovacích jazykov (identifikátory, čísla, reťazce, …). Obsahujú podmnožinu všetkých konečných jazykov. Základné prostriedky na reprezentáciu jazykov typu 3 sú:

- lineárna gramatika (podrobnejšie popísaná v kapitole 3.3);
- regulárny výraz;
- konečný automat.

Všetky tri spomenuté prostriedky sú navzájom ekvivalentné a je zároveň algoritmicky možné previesť reprezentáciu jazyka z jedného prostriedku na iný. Dokážeme napríklad na základe lineárnej gramatiky zostaviť konečný automat, prípadne regulárny výraz a naopak. (Kocur, 2001)

3.4.1 Lineárna gramatika a jej prevod na regulárnu

Rybička (1999) tvrdí, že z ľubovoľnej lineárnej gramatiky, dokážeme zostrojiť príslušnú regulárnu gramatiku, ktorá bude generovať rovnaký jazyk. Zmena však opäť nastáva v zápise pravidiel, kde rozdiel nastáva v počte terminálnych symbolov na kraji generovaného slova. Pripomeňme si preto predpis pravidiel lineárnej gramatiky, pre ktoré platilo nasledovné:

$$A \rightarrow \omega B \mid \omega$$

kde A, B $\in$ N; $\omega \in \Sigma^*$ (pravá lineárna gramatika)

$$A \rightarrow B\omega \mid \omega$$

kde A, B $\in$ N; $\omega \in \Sigma^*$ (ľavá lineárna gramatika)

Významnou zmenou pre regulárnu gramatiku je podľa Rybičku (1999) podmienka, že v každom prepisovacom pravidle sa smie nachádzať práve jeden terminálny symbol. Znamená to, že slovo, ktoré bolo v prípade lineárnej gramatiky zložené z ľubovoľného počtu terminálnych symbolov abecedy Σ ($\omega \in \Sigma^*$), bude nahradené v každom pravidle len jediným terminálom x danej abecedy ($x \in \Sigma$). Pravidlá pre regulárnu gramatiku budú vyzerať nasledovne:

$$A \rightarrow xB \mid x$$

kde A, B $\in$ N; $x \in \Sigma$ (pravá regulárna gramatika)

$$A \rightarrow Bx \mid x$$

kde A, B $\in$ N; $x \in \Sigma$ (ľavá regulárna gramatika)

V prípade, že $\epsilon \in L_G$ (jazyk L generovaný gramatikou G obsahuje aj prázdne slovo), môže existovať aj pravidlo $S \rightarrow \epsilon$. Podobne ako aj v predošlých prípadoch, ani v tomto sa však neterminálny symbol S nesmie vyskytovať na pravej strane žiadneho prepisovacieho pravidla gramatiky G. Jazyk môžeme generovať množstvom rôznych spôsobov, znamená to, že pre gramatiku G, generujúcu jazyk L_G, existuje nekonečné množstvo ekvivalentných gramatík G', ktoré sú schopné generovať rovnaký jazyk L_G. Líšiť sa podľa Rybičku (1999) môžu v:

- množine neterminálnych symbolov N;
- množine terminálnych symbolov Σ;
- prepisovacích pravidlách P;
- štartovacom symbole S.

Podľa Rybičku (1999) spôsob, akým dokážeme získať z pôvodnej gramatiky jej ekvivalent, nazývame transformácia gramatiky. Cieľom je, aby nová gramatika mala určitú požadovanú vlastnosť. Medzi základné vlastnosti, ktoré môžu byť cieľom transformácie patria:

- Redukovaná gramatika – prípad, kedy gramatika neobsahuje zbytočné a nedostupné symboly.
- Nevypúšťajúca gramatika – gramatika, ktorá neobsahuje žiadne pravidlo $A \rightarrow \epsilon$, prípadne existuje jediné pravidlo $S \rightarrow \epsilon$ za podmienky, že S sa nevyskytuje na pravej strane žiadneho prepisovacieho pravidla.
- Vlastná gramatika – je gramatika, ktorá spĺňa predošlé dve vlastnosti a zároveň neobsahuje cykly (teda pravidlá typu $a \rightarrow +$ a pre žiadne $a \in N$) resp. jednoduché pravidlá. Príkladom jednoduchého pravidla je $a \rightarrow B$.

Na základe vyššie uvedených pravidiel teda prevod z lineárnej gramatiky na regulárnu prebieha na základe algoritmu, ktorý v prvom kroku odstráni nedostupné a zbytočné symboly. Potom terminálne symboly typu $a \rightarrow \omega B$ a následne odstráni ϵ-pravidlá typu $a \rightarrow \epsilon$. V poslednom kroku odstráni aj jednoduché pravidlá typu $A \rightarrow B$. (Rybička, 1999)

3.4.2 Regulárny výraz a množina

Regulárnu množinu môžeme označiť za množinu slov zložených zo symbolov konečnej abecedy Σ, zostrojenú na základe stanovených pravidiel. Jej formálna definícia, ako uvádza Rybička (1999), je:

- „$\emptyset$ je regulárna množina nad Σ.
- $\{\epsilon\}$ je regulárna množina nad Σ.
- $\{a\}$ pre každé $a \in \Sigma$ je regulárna množina nad Σ.
- Ak sú P, Q regulárne množiny nad Σ, tak aj $P \cup Q$, $P \,.\, Q$ a P^* sú regulárne množiny nad Σ.
- Regulárne množiny sú práve tie množiny, ktoré je možné získať aplikáciou pravidiel 1–4.“

Regulárny výraz je podľa Rybičku (1999) notácia, využívaná pre zápis pravidiel regulárnych množín. Obsahuje:

- operandy – reťazce nad abecedou Σ;
- operátory – množinové operácie, zreťazenie (P . Q), iterácia (P*).

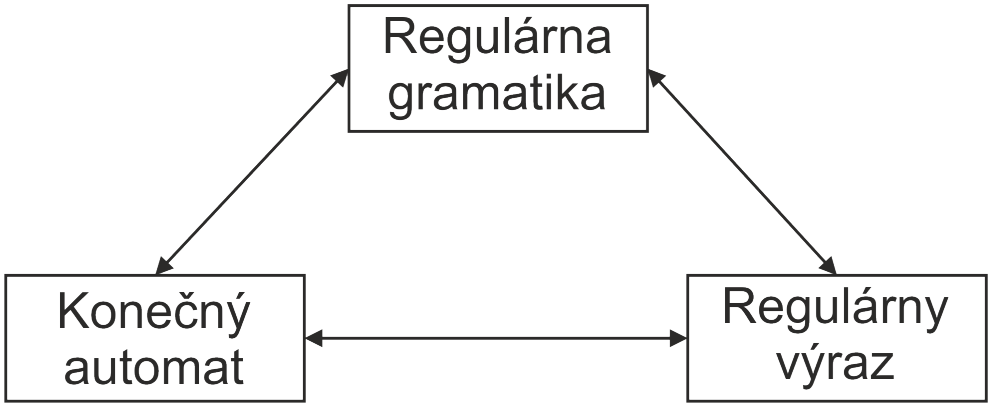
Rybička (1999) ďalej uvádza, že množina regulárnych výrazov nad abecedou Σ, označovaná RE(Σ) = $\Sigma \cup \{\emptyset, \epsilon, +, \cdot, *, (,)\}$, je definovaná induktívne:

- „ϵ, $\emptyset$, a pre každé a $\in \Sigma$ sú (základné) regulárne výrazy nad Σ.
- Ak sú E, F regulárne výrazy nad Σ, sú aj (E . F), (E + F) a (E)* regulárne výrazy nad Σ.
- Každý regulárny výraz vznikne po konečnom počtu aplikácií krokov 1–2.“

Regulárne výrazy definujú jediným výrazom celý jazyk. Slúžia hlavne na vymedzenie možných podôb textového reťazca. Ich využitie je potom efektívne napríklad pri vyhľadávaní na internete, hľadaní súboru v počítači (keď vieme napríklad len príponu súboru), ďalej na vyhľadávanie textu v súboroch, databázach, slovníkoch a tak podobne. (Rybička, 1999)

3.4.3 Konečný automat

Je poslednou z troch základných nástrojov na reprezentáciu jazykov typu 3. Keďže táto forma reprezentácie je pre túto prácu najdôležitejšia, bude konečný automat popísaný podrobne v nasledujúcej kapitole.



Obrázok 3.2 Prostriedky zobrazenia jazyku typu 3 (Rybička, 1999)

3.5 Konečný automat

Vzťah medzi regulárnou gramatikou a konečným automatom definuje Kleenova veta: „Ľubovoľný jazyk je regulárny práve vtedy, ak je rozpoznateľný konečným automatom.“ Z tejto vety ďalej vyplýva, že množina je regulárna nad Σ, vtedy a len vtedy, ak je akceptovaná vhodným konečným automatom nad Σ. (Kocur, 2001)

Konečný automat je abstraktný systém s konečným počtom vnútorných stavov (do ktorých sa môže dostať riadiaca jednotka), na ktorého vstup prichádza konečné množstvo symbolov vstupnej abecedy vo forme vstupnej pásky, ktorú číta čítacia hlava zľava doprava. Vstupom konečného automatu je veta ω a výstupom je odpoveď, či veta patrí alebo nepatrí do jazyka L. (Rybička, 1999)

Konečný automat je typicky deterministický stroj, preto je aj výpočet a implementácia deterministická – jednoznačná. Pri návrhu je však často jednoduchšie navrhnúť nedeterministický automat, ktorý následne prevedieme na jeho deterministický ekvivalent. (Rybička, 1999)

Nedeterministický automat je podľa Kocura (2001) jednoznačne definovaný päticou $M = (Q, \Sigma, \delta, q_0, F)$ kde:
Q – konečná, neprázdna množina stavov automatu
Σ – konečná, neprázdna množina vstupných symbolov (vstupná abeceda)
δ – prechodová funkcia definovaná ako zobrazenie $\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow P(Q)$, kde $P(Q)$ je potenčná množina
q_0 – počiatočný stav automatu ($q_0 \in Q$), niekedy sa uvádza aj množina počiatočných stavov S ($S \subseteq Q$), v prípade, že počiatočný stav automatu nie je jednoznačne určený
F – množina koncových stavov automatu ($F \subseteq Q$)

Či sa jedná o deterministický alebo nedeterministický automat, môžeme rozhodnúť na základe **prechodovej funkcie** δ. V prípade, že sa prechodová funkcia môže dostať do vždy do najviac jedného nasledujúceho stavu, hovoríme o deterministickom automate. Ak je nasledujúcich dostupných stavov viac, automat je nedeterministický. Špeciálnou formou deterministického automatu je totálny automat, ktorý je v každom stave schopný reagovať na všetky signály. Spomenutá prechodová funkcia, je dôležitou súčasťou každého automatu, pretože definuje prechod medzi konfiguráciami automatu. **Konfiguráciou** konečného automatu nazývame takú dvojicu, pre ktorú platí: $(q, \omega) \in Q \times \Sigma^*$. Konfiguráciu (q_0, ω), kde q_0 je počiatočný stav automatu, nazývame počiatočná konfigurácia. V opačnom prípade konfiguráciu (q_F, ω) kde q_F patrí do množiny konečných stavov F ($q_F \in F$), nazveme koncovou konfiguráciou. Konfigurácia automatu tak jasne definuje v akom stave sa momentálne automat nachádza a zvyšok vstupnej vety, ktorú ešte musí prečítať. V prípade, že sa automat dostane do stavu, ktorý patrí do množiny koncových stavov F, hovoríme, že veta je

automatom **prijatá** (**akceptovaná**). V takomto prípade platí, že automat je schopný nájsť takú postupnosť krokov (prechodov medzi stavmi, značime $\vdash$), kedy platí, že $(q_0, \omega) \vdash^* (q_F, \epsilon)$. Znamená to, že existuje taká postupnosť **krokov**, ktorá nám umožní dostať sa z počiatočného stavu do koncového. (Rybička, 1999)

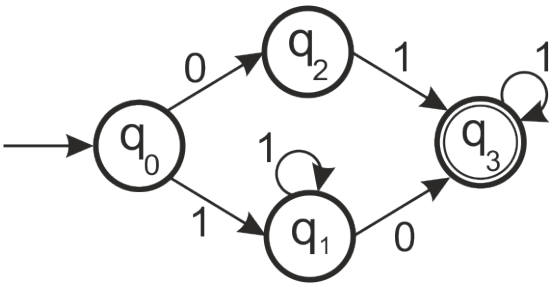
Kocur (2001) ďalej uvádza, že prechodová funkcia δ býva reprezentovaná tromi základnými spôsobmi:

- Tabuľka prechodov
  - = v záhlaví stĺpcov sú označenia stavov z Q,
  - = počiatočný stav označujeme symbolom $\rightarrow$ mieriacim na symbol stavu,
  - = koncový stav označujeme opačne orientovanou šípkou $\leftarrow$,
  - = v prípade, že sa koncový stav zhoduje s počiatočným použijeme symbol $\leftrightarrow$,
  - = v záhlaví riadkov sa nachádzajú označenia symbolov vstupnej abecedy,
  - = v poli tabuľky sa potom nachádzajú množiny stavov;

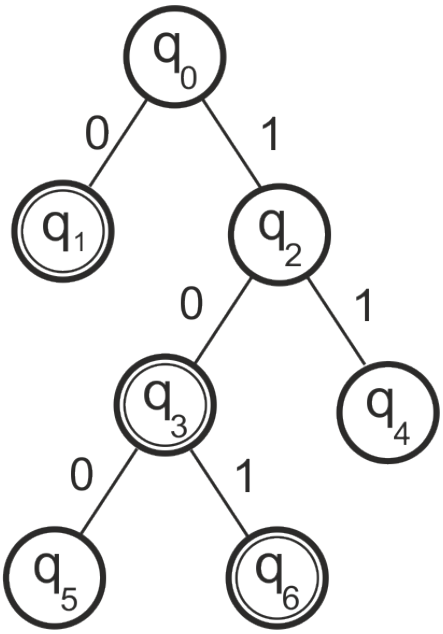
Tabuľka 3.1 Príklad prechodovej tabuľky

Δ	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
q_1	$\{q_3\}$	
q_2		$\{q_3\}$
$\leftarrow q_3$	$\{q_3\}$	$\{q_3\}$

- Stavový diagram – znázorňuje konečný automat ako konečný orientovaný graf, v ktorom:
  - = uzly grafu znázorňujú stavy z Q,
  - = hrany zobrazujú prechody,
  - = hrany sú označené príslušným terminálnym symbolom vstupnej abecedy,
  - = vstupné stavy sú označené symbolom $\rightarrow$
  - = koncové stavy sú označené dvojitým krúžkom okolo uzlu (stavu);
- Stavový strom – vhodný len na znázornenie deterministických konečných automatov, pričom je schopný zobraziť len stavy dostupné z počiatočného. Stavový strom je špeciálnym prípadom stavového diagramu, v ktorom platí:
  - = počiatočný symbol je koreňom stromu, preto už nie je potrebné ho inak vyznačovať,
  - = ohodnocovanie následníkov prebieha na základe prechodovej funkcie,
  - = koncové stavy sa označujú dohodnutým spôsobom (väčšinou dvojitým krúžkom rovnako ako u stavového diagramu).



Obrázok 3.3 Príklad stavového diagramu



Obrázok 3.4 Príklad stavového stromu

Rybička (1999) tvrdí, že jednoduchšie ako hneď navrhnúť deterministický automat, je najprv vytvoriť jeho nedeterministickú verziu, ktorú následne algoritmicky prevedieme na deterministický automat. Tento algoritmus prebieha v niekoľkých krokoch:

- Rozšírime pôvodnú množinu stavov Q tak, že niekoľko pôvodných stavov $\{q_0, q_1\}$ vytvorí nový stav $\{q_{01}\}$. Tým vznikne mnoho nových stavov, z ktorých mnoho bude nedostupných a nadbytočných. Tie bude potreba následne odstrániť.
- Novovzniknutý kompozitný stav $\{q_{01}\}$ preberie väzby a vlastnosti všetkých pôvodných stavov, z ktorých vznikol.

- Kompozitný stav vzniká tam, kde výsledok prechodovej funkcie je množinou stavov a nie jediným stavom, teda $|\delta(q_i, a)| > 1$.
- Následne už nie je nutné písať v tabuľke prechodov množinové zátvorky (v každom prípade nasleduje už len jeden stav).

4 Nástroj ConTEXt

V nasledujúcej kapitole je popísaný nástroj pre sadzbu textu ConTEXt, ktorého súbor bude použitý ako vstup pre konečný automat vytvorený v rámci tejto práce. Okrem základných informácií o ňom, bude popísaná aj jeho analýza potrebná pre správne spracovanie vstupného súboru a následnú detekciu typografických javov.

4.1 Základné informácie

Laakso (2010) uvádza, že: „ConTEXt patrí do rodiny nástroja TEX. TEX je jazyk navrhnutý pre sadzbu textu, eventuálne sadzbu matematiky, prípadne iných technických a vedeckých materiálov. Jeho história sa začala písať koncom 70. rokov minulého storočia, kedy Donald Knuth robil revíziu druhého vydania svojej knihy Umenie počítačového programovania, no pri pohľade na sadzbu reagoval len ‚fuj'. Táto skúsenosť ho viedla k nápadu vytvoriť úplne nový nástroj, ktorý by dokázal vysadiť text, s ktorým by bol spokojný."

Spomenúť môžeme napríklad LATEX, vyvinutý začiatkom 80. rokov, ktorý je založený na myšlienke zjednodušenia použitia TEX-u. Dôvodom je, že formátovacie príkazy čistého TEX-u sa dajú označiť za „low-level", čím sú pre obyčajného užívateľa pomerne náročné na použitie. Je tak pre neho jednoduchšie použiť LATEX, ktorý využíva silu TEX-u (jeho makrá) a zároveň program LATEX-u na spracovanie súboru. Jeho hlavnou myšlienkou je nechať vizuálny návrh súboru na dizajnérov a autorov nechať pokračovať v písaní samotného textu. (Hagen a Hoekwater, 2013)

Existuje nespočetné množstvo ďalších balíčkov a rozšírení, ktoré sú však určené hlavne pre LATEX. Môžeme ešte spomenúť napríklad XETEX – pre podporu Unicode, bibTEX – špeciálne vytvorený pre sadzbu citácií, pdfTEX a iné. (Laakso, 2010)

Nástroj ConTEXt, ktorý je pre túto prácu najvýznamnejší, je značkovací jazyk, ktorého systém je takisto ako u LATEX-u založený na sádzacom nástroji TEX. ConTEXt podobne ako LATEX, má za cieľ poskytovať jednoduché používateľské rozhranie pre kvalitnú sadzbu systémom TEX. Rozdielom však je, že zatiaľ čo LATEXizoluje užívateľa od typografických a formátovacích detailov, ConTEXt umožňuje prístup, široké možnosti a rozhranie pre manipuláciu s typografiou, farbami, pozadím, hypertextovými odkazmi, textom, podmienené kompilácie a množstvo iných možností. To dáva užívateľovi rozsiahlu kontrolu nad formátovaním súboru, pričom môže ľahko vytvárať a spravovať rozloženie a štýly bez učenia makier systému TEX. Jednotný dizajn ConTEXtu tiež zabraňuje stretom medzi ďalšími balíčkami, ako tomu niekedy býva

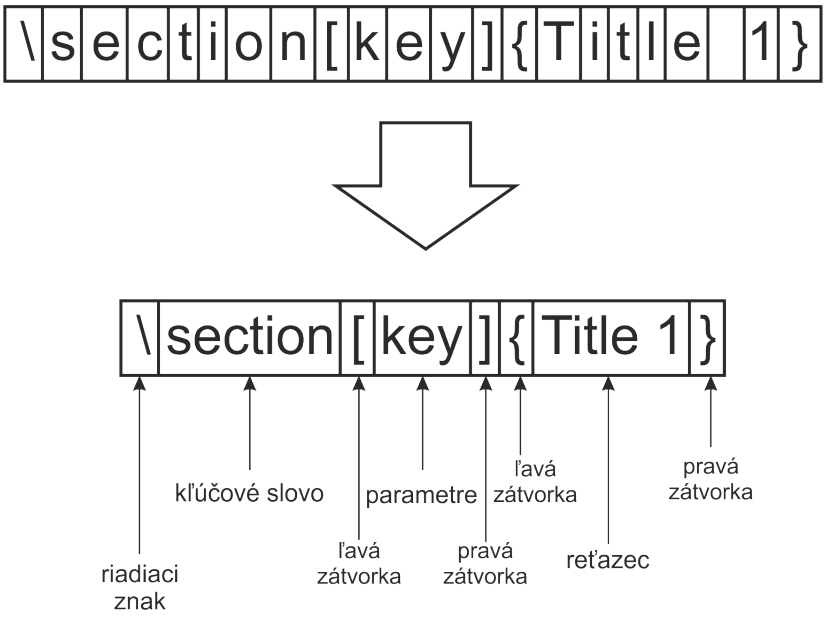
pri LATEX-e. ConTEXt sa tiež snaží viac rešpektovať syntaktický štýl čistého TEX-u. (Hagen a Hoekwater, 2013)

4.2 Analýza nástroja ConTEXt

Ako už bolo spomenuté, analýza jazyka ConTEXt je dôležitá pre správne fungovanie programu. Môžeme tvrdiť, že jazyk ConTEXt je postupnosť viet, skladajúcich sa zo slov vytvorených z jednotlivých symbolov abecedy. Je preto dôležité symboly vstupného súboru zoskupiť do slov a tie následne pospájať do celých viet. Jedine takto dokážeme pochopiť význam viet a v konečnom dôsledku aj celého súboru, na základe čoho môžeme ďalej postupnosť vstupných viet analyzovať a ďalej podľa potreby spracovávať. Túto analýzu môžeme rozdeliť do dvoch základných častí. Prvou je lexikálna analýza, ktorej cieľom je zo znakov na vstupe, vytvoriť postupnosť „tokenov“, ktoré dalej využíva druhá časť analýzy – syntaktická analýza. Token reprezentuje množinu reťazcov so spoločným významom. Sú výstupom lexikálnej analýzy a zároveň vstupom syntaktickej analýzy. Zvyčajne sa jedná o rezervované slová, identifikátory, operátory, konštanty a iné. (Dvořáková, 1999)

4.2.1 Lexikálna analýza

Princíp lexikálnej analýzy je znázornený na obrázku 4.1. Okrem spájania jednotlivých znakov do tokenov, má aj niekoľko ďalších funkcií. Medzi ne patrí napríklad vymazanie prázdnych znakov, ktoré však musíme v tejto práci vynechať, pretože výsledkom tohto automatu nie je len rozhodnutie, či veta patrí do daného jazyka, ale aj výstupný text, v ktorom nie je možné odstrániť prázdne znaky – súbor by v tomto prípade stratil význam (hlavne jeho časti obsahujúce text v prirodzenom jazyku). Rovnaké tvrdenie platí aj v prípade komentárov – musia zostať zachované. Takisto nemá v prípade korektoru typografie zmysel striktne kontrolovať syntax, je vhodné vychádzať z toho, že vstupný súbor je skompilovaný a bez chýb, teda preložiteľný prekladačom nástroja ConTEXt. Podobné vlastnosti budú podrobne rozobraté v praktickej časti, kde budú popísaný princíp fungovania konečného automatu a jeho vlastnosti. (Dvořáková, 1999)



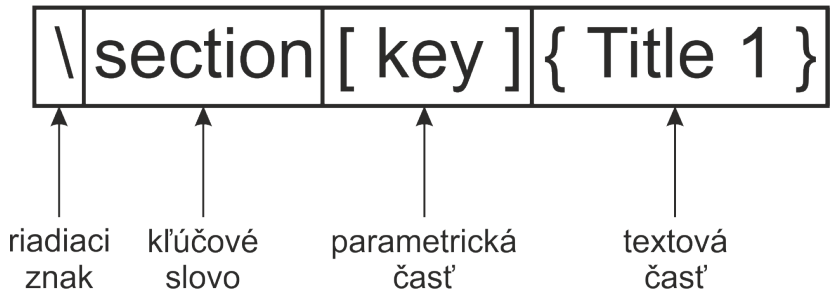
Obrázok 4.1 Princíp lexikálnej analýzy

4.2.2 Syntaktická analýza

Beneš a Kolář (1999) uvádzajú, že syntaktická analýza má za úlohu spracovať postupnosť tokenov a určiť, ktoré symboly zo vstupu patria k sebe a tvoria celok. Na základe nich rozpoznáva syntaktickú konštrukciu. Za základné syntaktické konštrukcie môžeme považovať:

- Postupnosť – v prípade obrázku 4.1, ak by po kľúčovom slove section musela nasledovať hranatá zátvorka (bola by daná postupnosť section → []).
- Alternatívu – príkladom by mohol byť prípad z obrázku 4.1, kedy by hranatá zátvorka bola voliteľná. Nastala by situácia kedy by po kľúčovom slove section nasledovala buď alternatíva hranatej zátvorky, alebo alternatíva zloženej zátvorky, prípadne iné alternatívy.
- Opakovanie – nastať by mohlo v prípade, že by po kľúčovom slove section nasledovala pasáž viacerých hranatých zátvoriek s parametrami (section → [] [] []).
- Hierarchiu.

Príklad z obrázku 4.2 je veta zo súboru vo formáte ConTEXt, kedy príkaz section (v nástroji ConTEXt sa príkazy píšu za znak „backslash \“) vysadí nadpis novej časti textu. Za týmto príkazom nasledujú voliteľné hranaté zátvorky, ktorých obsahom sú parametre (v ktorých nemáme záujem hľadať typografické chyby), nasledované zlo-



Obrázok 4.2 Princíp syntaktickej analýzy

ženými zátvorkami. V nich sa môže nachádzať text podliehajúci kontrole typografie. Zloženými zátvorkami časť section končí a čítanie pokračuje ďalej. Takto rozčlenený vstup však môžeme ďalej spracovávať, čo sa týka hlavne obsahu zloženej zátvorky a čo je zároveň cieľom syntaktickej analýzy v tejto práci – dostať zo súboru časti obsahujúce neštruktúrovaný text. Podrobnejšie budú pravidlá a postup delenia vstupného súboru na štruktúrované časti a neštruktúrované popísané v praktickej časti práce.

5 Analýza podobných nástrojov

Programy, ktoré umožňujú automatické korekcie, sa nazývajú korektory. Ich forma môže byť rôzna. Môžu byť implementované ako súčasť textového editora alebo ako samostatný program, ktorý korektúru vykonáva na vstupnom súbore a podobne. Ich dôležitou vlastnosťou je, či sú schopné rozlišovať značkovaný text alebo nie. Ak do korektoru, určeného pre neznačkované texty, vložíme značkovaný vstupný súbor, môžeme tým spôsobiť nevhodné zásahy do štruktúry súboru. V nasledujúcej časti budú uvedené korektory pre značkované texty (LaCheck, Aspell, Vlna), ale aj korektory pre neznačkované texty, ktoré sú svojou funkcionalitou príbuzné tejto práci (Korektor, MS Word, Grammaticon a iné).

5.1 LaCheck

Keďže sa jedná o korektor určený priamo pre prostredie TEX-u, je vhodné uviesť ho ako prvý. Jedná sa o nástroj (určený pre štruktúru LATEX-u), ktorý číta vstupný súbor a v prípade, že nájde nevhodné sekvencie znakov, zobrazuje varovania aj s umiestnením chyby v súbore. Ako to už však pri pravidlách písania textov býva, pravidlá nie sú vždy jednoznačné, a teda nevhodné sekvencie znakov sa dajú vnímať subjektívne. (Thorup, 1998a)

Manuál k programu LaCheck (Thorup, 1998a) uvádza tieto sledované javy:

- nezrovnalosti v párovaní skupín a zátvoriek, prípadne oddeľovačov matematického prostredia;
- nesprávne použitie medzier v skratkách, vrátane chýbajúcich akcentov;
- nesprávne umiestnenie medzier;
- použitie ... (sekvencie troch bodiek) namiesto príkazu \ldots alebo použitie príkazu \ldots na mieste kde by malo byť použité \cdots;
- chýbajúce ~ pred príkazmi \cite alebo \ref;
- dvojité medzery (vrátane nezalomiteľných);
- medzera pred poznámkami;
- rôzne opravy kurzívy v texte;
- nesprávna interpunkcia v okolí oddeľovačov matematického prostredia, vrátane sekvencií prázdnych znakov v okolí interpunkcie;
- nesprávne použitie úvodzoviek.

LaCheck však dokáže rozpoznať pomerne malé spektrum chýb, väčšinou typických pre LATEX. Kvôli odlišnej syntaxi sa tak stáva nepoužiteľným pre ConTEXt.

5.2 Vlna

Vlna (Olšák, 2010a) je jednoúčelový program, ktorý v súbore zadanom v parametre programu (používa sa pre súbory, kde je nezalomiteľná medzera reprezentovaná znakom „~“, typicky teda v prostredí TEX-u), nahradí znak medzery za neslabičnou predložkou alebo spojkou znakom vlnky. To zabráni, aby zlom riadku nastal na nevhodnom mieste. Užívateľ si tiež môže podľa potreby upravovať množinu objektov, ktoré sú programom sledované. (Hála a Urbanová, 2012)

Podobne ako v tejto práci, takisto aj program Vlna (Olšák, 2010a) obsahuje značky pre reguláciu behu programu. Ich zápis je %~- pre vypnutie a %~+ pre zapnutie nahradzovania medzier vlnkami. Práve značky tohto programu boli inšpiráciou pre ich zavedenie aj v tejto práci. (Olšák, 2010b)

```
Toto je príklad chyby
v~zalomení textu.

Toto je opravený príklad chyby
v~zalomení textu.
```

5.3 Aspell

Korektor Aspell pracuje pod OS Unix, je to samostatný program, ktorého zameraním sú preklepy vo vstupnom texte. Jeho súčasťou je mód určený pre spracovanie vstupného súboru vo formáte TEX. Interpunkciou sa však korektor Aspell nezaoberá. (Hála a Urbanová, 2012)

5.4 MS Word

Keďže program MS Word nepracuje so značkovaným textom, spomenieme len v krátkosti niektoré funkcie a nástroje zamerané na hľadanie a opravy chýb v texte. Korektor programu MS Word je rozdelený do niekoľkých samostatných častí: automatické opravy textu, pravopisná kontrola a nástroj Tezaurus, ktorý slúži ako synonymický slovník, prípadne pomáha pri výbere vhodnejších slov. (Mansfield, 1998)

Opravami interpunkcie sa korektor priamo nezaoberá. Obsahuje však len niekoľko

funkcií, ako napríklad prepis malého písmena na veľké v prípade, že sa nachádza za bodkou (čo môže často vplývať negatívne v prípadoch, kedy bodka neznamená koniec vety, napríklad pri radových číslovkách), eventuálne na začiatku odseku. Ďalším takýmto javom zabudovaným do opráv MS Word je sledovanie českých, resp. slovenských úvodzoviek („“), pre ktoré platí, že najprv automaticky vkladá spodné a následne horné. Ak ale dávame do úvodzoviek už napísaný text, úvodzovky vkladá na základe toho, z ktorej strany sa text nachádza (ak už je napísané slovo príklad a pridáme nakoniec úvodzovky, korektor automaticky pridá horné úvodzovky aj bez prítomnosti spodných – príklad“). Posledným príkladom môže byť náhrada spojovníka za pomlčku na základe toho, či sa okolo znaku nachádzajú medzery alebo nie, táto implementácia je však len základným riešením a v písaní týchto znakov existuje veľmi veľa výnimiek. Nevýhodou je aj komerčnosť programu MS Word. (Hála a Urbanová, 2012)

5.5 Ikor

Starším, no svojím zameraním pomerne blízkym korektorom je program Ikor (Rybička, 1997). Jedná sa o interaktívny program, ktorý na základe lexikálnej analýzy vybraných atómov vstupu stanovil ich poradie, ktoré bolo následne podrobené analýze vhodnosti. V tejto analýze hľadá nezrovnalosti v medzerovaní a nesprávnom poradí interpunkcie. Nájdené nezrovnalosti program zapísal do výstupného súboru, z ktorého sa v druhej fáze čerpali informácie pre výslednú korekciu. Program Ikor bol vytvorený pre OS DOS a nie je verejne dostupný. (Hála a Urbanová, 2012)

5.6 Libre Office Writer

Je súčasťou kancelárskeho balíku Libre Office, ktorý je na rozdiel od balíku MS Office voľne dostupný. Tento textový editor obsahuje korektor, ktorý odstraňuje výskyt dvojitých medzier, náhradu spojovníku (vrátane náhrady dvoch spojovníkov za poľštvorčekovú pomlčku) podobne ako v MS Word. Korektor ďalej dokáže odstraňovať nevhodné medzery a tabulátory na začiatkoch a koncoch riadkov, prípadne celých odsekov. Automaticky prebieha aj náhrada sekvencie troch bodiek za znak výpustku. Ďalšími javmi súvisiacimi s interpunkciou sa však korektor nezaoberá. Aj tu však treba zdôrazniť, že korektor nie je vhodný na korekcie značkovaného textu. (Hála a Urbanová, 2012)

5.7 Korektor

Korektor (Fiala, 2013a) je jediný zástupca online korekcie, ktorý umožňuje zadať text priamo do textového poľa na internetových stránkach a vykonať korekciu. Opravený text sa vysadí do vedľajšieho okna a je teda jednoducho porovnateľný s pôvodným (chyby, zmeny a upozornenia sú farebne vyznačené a po kliknutí ponúkajú možnosť voľby medzi navrhovanými opravami). Ako však uvádza sám autor, tento nástroj nie je vhodný pre detekciu chýb v odborných textoch a informatických článkoch. Určite pre to nie je vhodný ani pre korekciu v značkovaných textoch. Jedná sa však o moderný nástroj, ktorý je prispôsobenýna opravu typografie v českých textoch a výpis všetkých javov je uvedený na internetových stránkach nástroja Korektor. Výhodou je samozrejme používanie zdarma a veľmi jednoduché uzívateľské rozhranie. (Fiala, 2013b)

5.8 Grammaticon

Posledným príkladom je program Grammaticon (Lingea, 2003), ktorý je spomedzi spomenutých najkomplexnejší a poskytuje interaktívne riešenie pri detekcii a korekcii gramatických javov a interpunkcie, prípadne detekcii chýbajúcich alebo nadbytočných čiarok. Tento korektor je však komerčný. (Hála a Urbanová, 2012)

5.9 Vyhodnotenie dostupných možností

Na základe vyššie spomenutého programového vybavenia sme dospeli k záveru, že neexistuje žiadny voľne dostupný a voľne šíriteľný produkt, ktorý by bol schopný detekcie typografických chýb a následnej, do vhodnej miery automatickej, korekcie. Pre ConTEXt platí toto tvrdenie dvojnásobne, pretože v súčasnej dobe neexistuje vôbec žiadny korektor typografie. (Hála a Urbanová, 2012)

6 Návrh a implementácia

Návrh programu a jeho následná implementácia vychádza z niekoľkých základných predpokladov. Prvým je, že program predpokladá syntaktickú správnosť vstupného súboru, pretože typografický analyzátor nemá za cieľ hľadať syntaktické chyby vo vstupnom súbore. Cieľom programu je však hľadať typografické chyby aj v prípade, že súbor nie je syntaktický správny. Príkladom môže byť, že program ignoruje absenciu zloženej zátvorky, ktorá je za normálnych okolností povinná. V prípade detektora typografie toto nie je problémom a preto ak nájde iný text, jednoducho pokračuje a správa sa, ako keby bol príkaz zapísaný korektne. Ďalším prípadom môže byť nekorektné ukončenie súboru (neuzavreté zátvorky, neukončený príkaz a pod.), ktoré by za normálnych okolností pri preklade vrátilo chybu, detektor sa však snaží pracovať so vstupom, ktorý dostane a hľadať chyby aj v takomto prípade. Ak nájde nečakane koniec súboru, čítanie ďalej nepokračuje a snaží sa dostať z programu von a korektne ho ukončiť (ak je to možné). Vyhneme sa tým prerušeniam behu programu, a je tak možné zamerať sa na hlavný cieľ, ku ktorému je tento program určený. Chybové hlásenia program používa len v nevyhnutných prípadoch, kedy chyba vo vstupnom súbore neumožňuje korektný priebeh detekcie. Detekcia párovosti príkazov, zátvoriek, správnosti syntaxe a podobných javov je skôr úlohou syntaktického analyzátora. Toto je základný rozdiel v princípe fungovania konečného automatu, ktorý má za úlohu stanoviť, či daná veta patrí do jazyka alebo nie a vrátiť chybové hlásenie v prípade, že sa dostane do stavu neprípustného pre konečný automat. Automat implementovaný v rámci tejto práce bude modifikovaný tak, aby lepšie vyhovoval požiadavkám stanoveným na začiatku – detekovať a korigovať typografické chyby. Vo väčšine prípadov preto môže nasledovať prostý text, na základe čoho program pokračuje v čítaní vstupného súboru.

Z vyššie uvedených dôvodov je vhodné a odporúčané, previesť kontrolu syntaktickej správnosti vstupného súboru pred použitím detektora typografie. Takto sa vyhneme možným nepresnostiam pri rozhodovaní programu o tom, ktorú časť je vhodné formátovať, a ktorú nie. Ďalším predpokladom je použitie kódovania UTF-8, ktoré umožňuje pracovať jazyku Lua s národnými znakmi a program takisto využíva niektoré funkcie pracujúce s týmto kódovaním, takže v prípade vstupného súboru v inom kódovaní, prípadne použitia inej národnej sady, nemusí program pracovať korektne napríklad v prípade práce s českými úvodzovkami.

Nedeliteľnou súčasťou programu bude konfiguračný súbor. Tento súbor bude slúžiť pre prispôsobenie funkcionality užívateľom a bude okrem iného obsahovať možnosť doplniť vlastné makrá do niektorej zo skupín príkazov, nastaviť rôzne chovanie detekcie (napr. v komentároch), prednastaviť, ktoré typografické javy sa budú sledovať a iné. Na jeho konci bude kompletný výpis množín príkazov.

Ako už bolo viackrát spomenuté, samotný program bude fungovať ako deterministický konečný automat. Vstupom, resp. vstupnou páskou, bude v tomto prípade súbor, zadaný ako parameter pri spustení skriptu v jazyku Lua. Chovanie automatu bude popísané formálnym jazykom typu 3, teda regulárnym. Automat a jeho chovanie bude implementované vo forme podprogramov (procedúr a funkcií). Súbor vo formáte ConTEXt, sa skladá z niekoľkých typov riadiacich znakov.

Na najvrchnejšej úrovni to môže byť znak backslash „\“, po ktorom nasleduje makro nástroja ConTEXt. Makrá sa takisto dajú ďalej rozdeliť na makrá typu: reťazec, jednopísmenové a iné znaky (napr. \, %). Reťazce sa ďalej delia na niekoľko špeciálnych štruktúr, ktoré budú zadelené do množín príkazov so spoločnými prvkami, nasledujúcimi za príkazom. Všetky skupiny budú bližšie popísané formálnym jazykom a aj vysvetlené v kapitole venovanej tejto problematike. Na teraz uvediem len príklad niekoľkých možností spoločných prvkov, ktoré môžu nasledovať za štruktúrou typu „\command“. Séria hranatých zátvoriek (často voliteľných), sekvencia zložených zátvoriek – tie bude nutné ďalej rozlišovať na tie, ktoré môžu obsahovať text podliehajúci formátovaniu a tie, ktoré bude potrebné vynechať. Ďalšími možnosťami sú kombinácie hranatých zátvoriek so zloženými typu [][]{} alebo {}[][] v rôznom počte zátvoriek a poradí. Okrem týchto základných skupín môže súbor obsahovať aj bloky, ktorých celý obsah nebude podliehať detekcii typografie. Nasledovať môže aj názov súboru prípadne niektoré ďalšie zvláštne skupiny, ktorým budú venované vlastné časti tejto práce.

Druhým riadiacim znakom najvyššej úrovne je znak percenta „%“. Jeho funkcia je jasná, pokiaľ nenasleduje hneď za znakom „backslash“, kedy sa jedná o znak, ktorý sa vysadí do textu, značí začiatok komentáru v ConTEXt-ovom súbore. Komentár je vždy jednoriadkový a jeho čítanie ukončuje znak konca riadku. To, že za týmto znakom môže nasledovať čokoľvek (komentár môže obsahovať ľubovoľné znaky), je využité k lepšej ovládateľnosti detekcie typografie. Pretože v prípade, že za percentom nasleduje špeciálna postupnosť znakov, definovaná výhradne pre tento program, môžeme vypínať a zapínať sledovanie jednotlivých typografických javov. Okrem možnosti vypnúť alebo zapnúť kontroly (bodky, čiarky, úvodzoviek atď.), je tu možnosť vypnúť všetky jediným príkazom. Táto funkcia umožňuje pomerne jednoducho vynechať časti súboru, o ktorých vopred vieme, že majú zvláštne formátovanie, prípadne formátovanie, o ktorom vieme, že sa líši od formátovania definovaného typografickými pravidlami. Často je vhodné takisto zapnúť detekciu typografie až v momente výskytu prvého textu. Je preto jednoducho možné vypnúť detekciu na časť, ktorá obsahuje len definície, nastavenia nástroja ConTEXt a podobné časti súboru, často obsiahnuté v hlavičke. Ďalšou časťou, ktorú môže chcieť užívateľ často vynechať, je časť zdrojov a literatúry. Tu môže byť interpunkcia používaná inak, ako definujú pravidlá. Okrem toho, že sa jednoducho vyhneme niektorým neželaným detekciám, urýchlime aj beh celého programu, ktorý nebude zbytočne analyzovať obsah textu, pri vypnutom formátovaní, ale text z vstupnej pásky len jednoducho prečíta. Presný popis chovania algoritmu na spracovávanie oblasti za znakom percenta, vrátane for-

málneho jazyku a automatu, bude rozobratý podrobne neskôr.

Za posledný riadiaci znak najvyššej úrovne, sa dá okrem dvoch vyššie uvedených znakov, považovať aj množina všetkých znakov (percenta a spätného lomítka) a prázdneho reťazca. Ten symbolizuje koniec súboru a teda aj koniec vstupu. Tieto časti obsahujú prostý text, ktorý je nutné vyčleniť pre následnú detekciu typografických javov. Vo formáte súboru nástroja ConTEXt, nemusí byť takýto text nijak zvláštne vyčlenený. Niekedy sa vyčleňuje ako skupina a je preto ohraničený zloženými zátvorkami. Všeobecne sa však dá povedať, že pravidlá pre jeho písanie v jazyku ConTEXt sú pomerne voľné. V nasledujúcom príklade je krátky úsek textu vo formáte ConTEXt, kde sú veľmi jednoducho uvedené základné štruktúry:

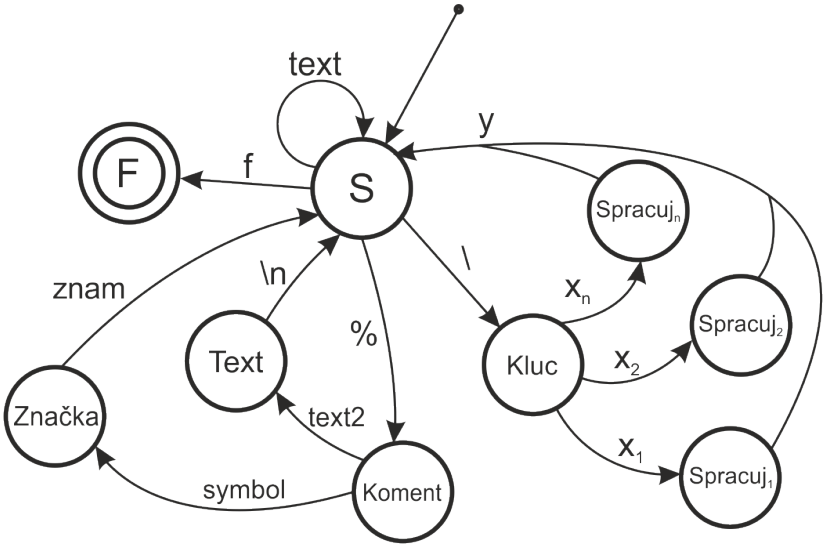
```
\príkaz [parametre]
\príkaz2 [parametre] {text s~možnými chybami} %komentár
Prostý text podliehajúci detekcii typografických javov.
\príkaz3
```

Takto vyčlenený text bude vstupom pre ďalšie spracovanie. Ďalším spracovaním myslíme procedúry pre detekciu typografických chýb v tomto prostom texte. Každý sledovaný znak má svoju vlastnú procedúru pre spracovávanie a preto bude aj samostatne popísaný vlastným formálnym jazykom. Výstupom týchto detekcií bude pole obsahujúce stručný popis chyby a jej polohu vo vstupnom súbore – riadok a číslo znaku. Na základe tohto poľa chýb bude generovaný súbor s návrhmi opráv, na základe ktorého bude prebiehať korekcia vstupného súboru. Popis tohto súboru a jeho štruktúra budú uvedené v dokumentácii.

7 Popis formálnym jazykom

Pre zjednodušenie návrhu formálneho jazyka budem popisovať jednotlivé gramatiky postupne od najvyššej úrovne a pre prehľadnosť často aj v bezkontextovom jazyku, ktorý bude obsahovať podstatne menej symbolov, čo značne zvýši jeho prehľadnosť. Každý z týchto jazykov je však možné previesť na regulárny jazyk a tým ho aj popísať konečným automatom. Jazyky budú v deterministickej forme.

Ako už bolo spomenuté, na najvyššej úrovni sa nachádzajú tri základné riadiace znaky. Zjednodušene by sa princíp fungovania na najvyššej úrovni dal zakresliť do grafu (obr. 7.1):



Obrázok 7.1 Grafické znázornenie fungovania automatu na najvyššej úrovni

Slovne by sa tento princíp dal opísať tak, že automat začne čítanie vstupu v neterminálnom symbole S, ktoré funguje ako základné „rázcestie“ najvyššej úrovne. Ak je súbor prázdny, pôjde po vetve konca súboru *f* a korektne skončí. V inom prípade nájde jednu zo štruktúr uvedených skôr – % Koment, \Kluc (príkaz jazyka ConTEXt) alebo prostý text, ktorého čítanie končí až nálezom symbolu percenta (ak nie je escapovaný), spätného lomítka alebo konca súboru. Vetva komentáru môže obsahovať komentár alebo značku (v prípade, že nasleduje symbol označujúci sledované typografické javy), ktorá mení nastavenia detekcie typografických javov. Príkazová vetva Kluc po náleze spätného lomítka prečíta príkaz x_i a zaradí ho do jednej zo skupín

príkazov, ktoré sú definované vlastnými podgramatikami, ktoré sú v tomto prípade označené ako Spracuj$_i$. Symbolom *y* je označený výsledok gramatík v časti Spracuj$_i$. Formálny zápis gramatiky pre zjednodušený pohľad na najvyššiu úroveň spracovania by vyzeral nasledovne:

G = ({S, F, KLUC, KOMENT, TEXT, SPRACUJ$_i$, ZNACKA}, { \n, \, %, f, symbol, text, text2, x$_i$, y, znam}, P, S)

Množiny terminálnych symbolov:
f = {nil = znak konca súboru}
text = Σ – { \, %, f}
znam = {+, -}
symbol = {@ ~. , ! ? ; - * (’}
text2 = Σ – {eol, c}
x1, x2, ... = množiny príkazov nástroja ConTEXt napr. { section, subsection }
y = výstup gramatík jednotlivých množín príkazov

S → f | \KLUC | % KOMENT | text S
KLUC → x$_1$ SPRACUJ$_1$* | x$_2$ SPRACUJ$_2$ | x$_n$ SPRACUJ$_n$
SPRACUJ$_i$ → yS
KOMENT → text2 TEXT | znam ZNACKA
TEXT → \n S
ZNACKA → znam S

* Neterminály SPRACUJ$_1$, SPRACUJ$_2$ atď. označujú vnorené gramatiky, ktoré budú definované samostatne v ďalších kapitolách. Tieto podgramatiky sa na svojom konci vždy vrátia na „rázcestie“, ktoré je reprezentované symbolom S. Terminálna časť spracovaná týmito gramatikami je označená ako y.

Keďže táto gramatika vysvetľuje len základný princíp fungovania, bude nutné popísať jednotlivé časti samostatne. Úplná gramatika by z dôvodu pomerne veľkého počtu skupín príkazov bola veľmi rozsiahla a značne neprehľadná. Z tohto dôvodu budú všetky skupiny rozobraté samostatne.

Už z vyššie uvedenej zjednodušenej gramatiky je jasné, že jeden zo základných riadiacich znakov – prostý text – už nie je potrebné ďalej rozoberať. Jeho funkcia je jasná – čítaj text až kým nenarazíš na iný riadiaci znak alebo koniec súboru. Ostatné dva typy však bude potrebné popísať podrobnejšie a budú k nim uvedené ich vlastné formálne jazyky. Spojením všetkých sa potom následne bude dať zostrojiť jeden veľký celok, ktorý bude vyjadrovať princíp chovania programu pri rozdeľovaní súboru na štruktúrované časti a prostý text.

7.1 Komentár

Prvou a zároveň oveľa kratšou vetvou je komentár. Táto vetva sa dá rozdeliť na spomínané dve časti: značky detekcie alebo text komentáru. Zhrnúť by sa to dalo tak, že komentár začína znakom % a končí koncom riadku (\n). K lepšej možnosti ovládania programu, je to zároveň ideálne miesto pre vkladanie značiek, ktoré nijak neovplyvnia normálny beh a preklad vstupného súboru. Ničím tak do výsledku nezasiahneme. Tieto značky sa skladajú z troch častí:

- % – ako prvý symbol, začiatok komentáru v nástroji ConTEXt.
- Symbol definujúci, ktorý z typografických javov budeme nastavovať. Množina symbolov je: {@ ~. , ! ? ; - * (’}, kde hviezdička označuje formátovanie úvodzoviek, vlnka formátovanie medzier a znak zavináča nastavuje formátovanie ako celok. Znamená to, že všetky funkcie zapne alebo vypne. Ostatné javy sú formátované príslušným symbolom.
- Posledným znakom značky sú znamienka – a +. Tie udávajú, či zvolenú funkciu zapíname alebo vypíname.

V prípade, že sa o značku nejedná, teda nespĺňa stanovenú štruktúru, jedná sa o textový komentár, ktorého obsah sa prečíta a na základe voľby v konfiguračnom súbore prebieha alebo neprebieha detekcia typografie. Poslednou možnosťou je okamžitý nález konca riadku. V tomto prípade komentár hneď za znakom percenta končí, dá sa považovať za prázdny komentár. Komentár je popísaný nasledovným jazykom:

G_{kom} = ({S_{kom}, A, B}, {\, sym, text, znam}, P, S_{kom})

$S_{kom} \rightarrow$ sym a | text B | \n
A $\rightarrow$ znam
B $\rightarrow$ \n

Znak percenta číta už gramatika na najvyššej úrovni ako riadiaci znak, preto už nie je obsiahnutý v gramatike popisujúcej komentár. Gramatika je tým zostručnená a zobrazuje vyššie spomenuté tri možnosti, ktoré môžu nastať v časti komentárov. Ak by sme chceli túto gramatiku pripojiť k pôvodnej, o úroveň vyššie, jednoducho by sme doplnili za všetky terminálne symboly ukončujúce túto gramatiku symbol S z hlavnej gramatiky spomenutej v predošlej podkapitole (pravidlo a by v takom prípade vyzeralo: a $\rightarrow$ znam S). Tým by sme sa dostali opäť na úroveň hľadania jedného z trojice riadiacich znakov.

7.2 Príkazy jazyka ConTEXt

Nasledujú vždy sa znakom spätného lomítka. Keďže jazyky typu TEX sa v podstate nedajú popísať jednou gramatikou popisujúcou celý jazyk, je časť príkazov riešená pomocou množín príkazov, ktoré sú rozdelené podľa toho, čo za nimi nasleduje. Okrem týchto množín sa však za spätným lomítkom nachádzajú aj niektoré špeciálne znaky, prípadne jednopísmenové príkazy.

Spomenutý už bol špeciálny znak %, ktorý zapísaný ako % nie je možné považovať za komentár. Jedná sa o symbol, ktorý do výsledného súboru zapíše znak percenta. Ďalšie znaky sú napríklad:

- \\
- \#
- \$
- \~
- { }
- iné

Takto zapísané znaky väčšinou znamenajú špeciálne znaky ConTEXt-u. Tento zápis znamená, že sa vysadia priamo do textu a nie sú súčasťou štruktúrovaného zápisu. Je ich pomerne dosť, no väčšina z nich len slúži na sadzbu špeciálnych znakov. Nemajú preto vplyv na rozhodovanie o tom, či sa jedná o prostý text, alebo štruktúrovanú časť jazyka ConTEXt. Z tohto dôvodu nie je ich výskyt ničím zvlášť podstatný. Až na niekoľko výnimiek im preto program nevenuje zvláštnu pozornosť, a preto pri ich náleze za spätným lomítkom sa vracia na najvyššiu úroveň rozhodovania – hľadanie základného riadiaceho znaku.

Zaujímavejšou skupinou, ktorú je dôležité sledovať, aby sme sa vyhli nesprávnym zásahom do súboru, sú príkazy zložené z písmen. Tie budeme deliť do skupín podľa toho, čo po nich nasleduje. Obsahom zložených zátvoriek často býva text, ktorý podlieha detekcii typografie. Nie je tomu tak vždy a preto je nutné rozdeliť príkazy na tie, kde sa text môže vyskytovať, a tie, kde výskyt textu môžeme vylúčiť. Ich formálny jazyk bude identický, no v prípade, že sa jedná o príkaz, kde by sa mohol vyskytnúť hľadaný text, bude nutné obsah zátvoriek ďalej spracovať. V opačnom prípade automat ich obsah len prečíta bez ďalšieho spracovania.

Program rozpoznáva niekoľko skupín príkazov:

- sekvencia zložených zátvoriek (jedna až štyri – pre každý počet vlastná množina príkazov);
- sekvencia hranatých zátvoriek (jedna až štyri – pre každý počet vlastná množina príkazov);
- najprv hranaté zátvorky a potom zložené (rozdelené na základe počtov jednotlivých zátvoriek, prípadne vhodne spojené);
- najprv zložené zátvorky a potom hranaté (rozdelené na základe počtov jed-

notlivých zátvoriek, prípadne vhodne spojené);
- iné kombinácie zložených a hranatých zátvoriek (napr. [] {} []);
- okrúhle a potom zložené zátvorky (jediný výskyt pri príkaze \position);
- nasleduje názov súboru;
- oblasť, ktorá nepodlieha detekcii;
- bezparametrické.

Značnou komplikáciou, ktorá platí pre všetky skupiny príkazov je, že v ktorejkoľvek časti sa môže nachádzať komentár. Tento fakt, hlavne pri regulárnych gramatikách veľmi komplikuje a rozširuje potrebný zápis, postačujúci k pochopeniu fungovania formálneho jazyku. Z tohto dôvodu, budú gramatiky zapísané v jazyku typu 2 Chomského klasifikácie – bezkontextovým jazykom. Takto bude možné v každom bode zapísať pravidlo pre komentár ako:

S → % KOM S
A → % KOM a
KOM → text B
B → \n

Takéto pravidlo umožní jednoduchý návrat naspäť do časti v ktorej sa komentár nachádza a nie je preto nutné vytvárať pre každé miesto v gramatike, kde sa môže komentár nachádzať, vytvárať samostatnú vetvu. To by znamenalo, že všetky pravidlá, ktorú majú možnosť komentáru by potrebovali niekoľko ďalších pravidiel:

S → % KOM1
A → % KOM2
KOM1 → text B1
KOM2 → text B2
B1 → \n S
B2 → \n A

Tento príklad uvádza komentár len na dvoch miestach a nie je ani uvedená kompletná gramatika pre komentár, no už tu je jasne vidieť rozširovanie celej gramatiky. Náhrada za bezkontextový zápis preto značne sprehľadní zápis a na pochopenie jednotlivých gramatík pre každú skupinu kľúčových slov bude postačujúca. Bezkontextové gramatiky budú však vždy uvedené v podobe, ktorá je algoritmicky prevoditeľná na jej regulárnu ekvivalentnú formu.

7.2.1 Príkazy nasledované zloženými zátvorkami

Zložené zátvorky sú spomedzi delenia príkazov najdôležitejšie, pretože môžu obsahovať textové časti, ktoré bude treba skontrolovať. Spomenuté už bolo, že zložené zátvorky budú mať dva varianty. Prvá, kedy sa bude jednať o množinu príkazov s následnými zloženými zátvorkami s textom, a druhá, kedy nasleduje zložená zátvorka, ktorej obsah netreba kontrolovať. Formálny jazyk pre popis oboch možností je identický, ale chovanie programu sa bude líšiť. V jazyku je spomínaný text v oboch prípadoch označený ako t2 – text v zložených zátvorkách. Zatiaľ čo jedna skupina tento obsah len prečíta, druhá ho bude považovať za vstup pre ďalšie spracovanie. Nie je to však stále také jednoduché. Zložené zátvorky môžu obsahovať nie len prostý text, ale aj ďalšie príkazy. Trochu nadsadene by sa dalo povedať, že sa môže jednať o súbor ConTEXt-u v súbore ConTEXt-u. Minimálne, čo sa pre spracovanie týmto programom týka. Obsah týchto hranatých zátvoriek preto bude rekurzívne spracovávaný rovnakou procedúrou ako samotný vstupný súbor. Týmto spôsobom bude možné v obsahu zátvoriek rozhodnúť, čo je naozaj text, a čo je len ďalšia vnorená štruktúra, do ktorej nie je vhodné zasahovať.

Gramatika príkazu, po ktorom nasleduje maximálne jedna zložená zátvorka vyzerá nasledovne:

G = ({K, O, S_z}, {t, t2, {, }, eol, kom, sym, znam}, P, S_z)
t = Σ − {%, {, ϵ}
t2 = Σ − {%, }, ϵ}
kom = Σ − eol, sym
eol = {\n}
znam = {+, -}
sym = {@ ~. , ! ? ; - * (’}

$S_z \rightarrow$ % K S_z | { O | t
K $\longrightarrow$ kom eol | eol | sym znam
O $\longrightarrow$ % K O | t2 O | }

Tejto gramatike predchádza terminálny symbol vo forme reťazca, ktorý je súčasťou množiny slov, po ktorých nasleduje maximálne jedna zložená zátvorka. Po tomto reťazci môže nasledovať slovo generované uvedenou gramatikou. Integrácia tejto gramatiky do celku by opäť prebehla pripojením symbolu S z najvyššej gramatiky nakoniec všetkých terminálnych symbolov ukončujúcich gramatiku pre jednu zloženú zátvorku.

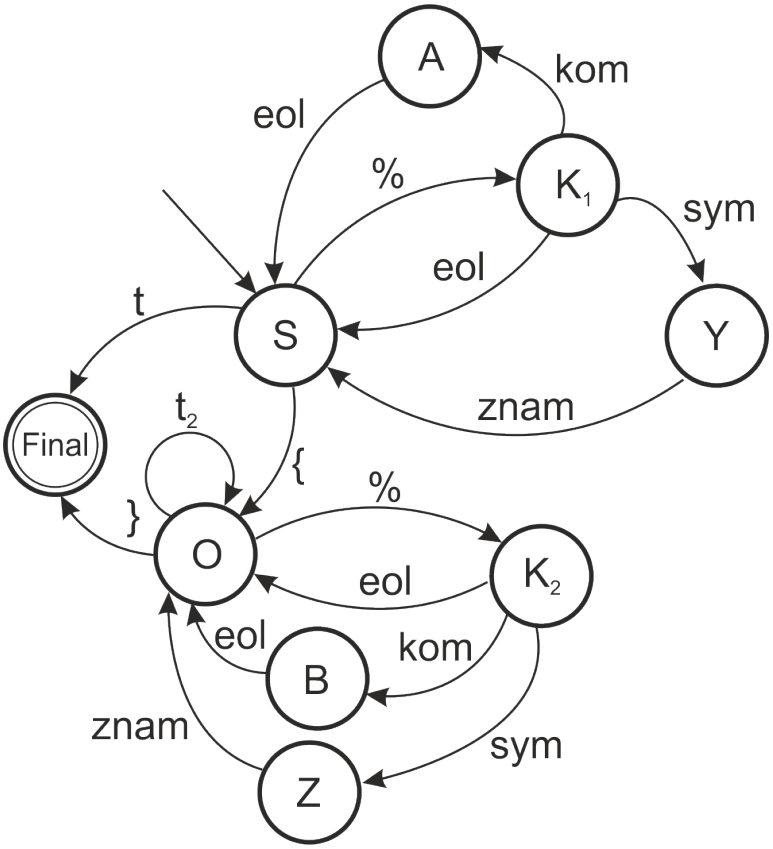
Keďže sa jedná o pomerne jednoduchú gramatiku, uvedieme aj jej prepis na regulárnu, vrátane prechodovej tabuľky automatu a jeho grafického znázornenia.

G = ({A, B, K1, K2, O, S_z, Y, Z}, {t, t2, {, }, eol, kom, sym, znam}, P, S_z)
t = Σ − {%, {, ϵ}
t2 = Σ − {%, }, ϵ}
kom = Σ − eol, sym
eol = {\n}
znam = {+, -}
sym = {@ ~. , ! ? ; - * (’}

$S_z \longrightarrow$ % K1 | { O | t
A $\longrightarrow$ eol S
B $\longrightarrow$ eol O
K1 $\longrightarrow$ kom a | eol S_z | sym Y
K2 $\longrightarrow$ kom B | l O | sym Z
O $\longrightarrow$ t2 O | % K2 | }
Y $\longrightarrow$ znam O
Z $\longrightarrow$ znam S_z

Tabuľka 7.1 Prechodová tabuľka pre popis jazyka jednej zloženej zátvorky

Δ	%	t2	kom	eol	sym	t	znam	{	}
$\rightarrow q_s$	q_{k1}					q_{Final}		q_o	
q_o	q_{k2}	q_o							q_{Final}
q_a				q_s					
$\leftarrow q_{Final}$									
q_b				q_o					
q_{k1}			q_a	q_s	q_y				
q_{k2}			q_b	q_o	q_z				
q_y							q_s		
q_z							q_o		



Obrázok 7.2 Grafické znázornenie konečného automatu spracujúceho príkaz so zloženou zátvorkou

Z tohto základu by sa dalo vychádzať pri tvorbe gramatiky pre viacnásobné zložené zátvorky nasledujúce za príkazom. Nasledujúca gramatika pripúšťa, že po prečítaní prvej zloženej zátvorky, nemusí nastať koniec, ale môže prísť ešte druhá zátvorka. Takéto reťazenie gramatík generujúcich hranaté zátvorky prebieha jednoducho: za koncový symbol prvej uzatváracej zloženej zátvorky vložím možnosť, že môže nasledovať opäť celá postupnosť s tým rozdielom, že po ukončení druhej zloženej zátvorky gramatika končí. Gramatika potom bude vyzerať takto:

G = (K, N, O, O2, $_{z2}$, t, t2, , , eol, kom, sym, znam, P, S_{z2})
t = Σ – {%, {, ϵ}
t2 = Σ – {%, }, ϵ}
kom = Σ – eol, sym
eol = {\n}
znam = {+, -}
sym = {@ ~. , ! ? ; - * (’}

$S_{z2} \longrightarrow$ % K S_{z2} | { O | t
K $\longrightarrow$ kom eol | eol | sym znam
O $\longrightarrow$ % K O | t2 O | }N
N $\longrightarrow$ % K N | { O2 | t
O2 $\longrightarrow$ % K O2 | t2 O2 | }

Na rovnakom princípe by sa postupovalo aj v prípade troch a štyroch zložených zátvoriek, nie je preto potrebné uvádzať aj tieto gramatiky.

7.2.2 Príkazy nasledované hranatými zátvorkami

Vo väčšine prípadov sa jedná o príkaz, ktoré vo svojich hranatých zátvorkách niečo definujú, nastavujú, štartujú alebo menia formát, farbu, typ čiar a podobne. Nie je preto vhodné do týchto častí zasahovať. Všeobecne platí, že v hranatých zátvorkách sa nachádzajú štruktúrované časti súboru. Výnimkou môžu byť oblasti, ktoré sú v úvodzovkách – tie budú považované za text podliehajúci detekcii typografie. Samotný jazyk popisujúci hranaté zátvorky bude však značne podobný tomu, ktorý spracováva zložené zátvorky. Rozšírený bude len o možnosť nálezu textu v úvodzovkách, v ktorom bude prebiehať detekcia chýb.

G = ({K, N, O, O2, S_h}, {t, t2, t3, {, }, eol, kom, sym, znam}, P, S_h)
t = Σ – {%, [, ϵ}
t2 = Σ – {%,], ϵ}
t3 = Σ – {%, ", ϵ}
kom = Σ – {eol, sym}
eol = {\n}
znam = {+, -}
sym = {@ ~. , ! ? ; - * (’}

$S_h \longrightarrow$ % K S_h | [O | t
K $\longrightarrow$ kom eol | eol | sym znam
O $\longrightarrow$ " U O | t2 O | % K O |]
U $\longrightarrow$ % K U | t3 U | "

Rovnako ako u zložených zátvoriek, v prípade viacerých prípustných hranatých zátvoriek za príkazom, môžeme za terminálny symbol] umiestniť neterminálny symbol, ktorý bude ďalej možné rozviesť na druhú zátvorku (prípadne ďalšie).

7.2.3 Príkazy nasledované kombináciou zátvoriek

Podobne ako nadviazať sekvenciu rovnakých zátvoriek, môžeme prepájať medzi sebou aj rôzne typy zátvoriek. Program obsahuje viacero skupín, ktoré obsahujú rôzne poradie a počet zátvoriek, je preto zbytočné uvádzať jazyky popisujúce všetky kombinácie. Nižšie budú opäť uvedené len základné „stavebné kamene“ ďalších množín príkazov. Ich kombináciami a pripájaním ďalších zátvoriek, môžeme popísať všetky skupiny slov. Nasledujúce gramatiky popisujú výskyt jednej hranatej zátvorky a jednej zloženej zátvorky za príkazom, respektíve naopak. V skutočnosti však množiny obsahujú aj kombinácie:

- [] {};
- [] {} {};
- {} {};
- {} [] [];
- {} [] {};
- a iné.

Nasledujúca gramatika popisuje jedny hranaté zátvorky, nasledované jednými zloženými. Kdekoľvek okolo nich, ale aj priamo v nich sa môže vyskytovať komentár. Obe zátvorky sú voliteľné a ich čítanie preto môže byť prerušené výskytom prostého textu.

G_{hz} = ({K, N, O, O2, S_{hz}, U}, {t, t2, t3, t4, t5, [,], {, }, eol, kom, sym, znam}, P, S_{hz})
t = Σ − {%, [, {, ϵ}
t2 = Σ − {%,], ϵ}
t3 = Σ − {%, ", ϵ}
t4 = Σ − {%, }, ϵ}
t5 = Σ − {%, {, ϵ}
kom = Σ − {eol, sym}
eol = {\n}
znam = {+, -}
sym = {@ ~. , ! ? ; - * (’}

S_{hz} ⟶ % K S_z | [O1 | t | { O2
K ⟶ kom eol | eol | sym znam
N ⟶ % K N | O2 | t5
O1 ⟶ " U O1 | t2 O1 | % K O1 |] N
O2 ⟶ % K O2 | t4 O2 | }
U ⟶ % K U | t3 U | "

Druhá gramatika pripúšťa taktiež jedny hranaté a jedny zložené zátvorky. Ich poradie je však narozdiel od prvej uvedenej gramatiky v opačnom poradí. Slúži ako porovnanie pre lepšie pochopenie závislostí medzi poradím a počtami zátvoriek za príkazom. Gramatiky sú takmer identické, no je dôležité si uvedomiť, po ktorej zátvorke môžeme ešte čítať ďalšiu, a po ktorej už nie. Je nutné týmto závislostiam prispôsobiť pravidlá tak, aby sa program v každom prípade dokázal jednoznačne rozhodnúť a posúdiť, či sa ešte jedná o súčasť príkazu alebo už sme narazili na prostý text.

G_{zh} = ({K, N, O, O2, S, U}, {t, t2, t3, t4, t5, [,], {, }, eol, kom, sym, znam}, P, S)
t = Σ − {%, [, {, ϵ}
t2 = Σ − {%,], ϵ}
t3 = Σ − {%, ", ϵ}
t4 = Σ − {%, }, ϵ}
t5 = Σ − {%, {, ϵ}
kom = Σ − {eol, sym}
eol = {\n}
znam = {+, -}
sym = {@ ~. , ! ? ; - * (’}

S_{zh} ⟶ % K S_{zh} | { O1 | t | [O2
K ⟶ kom eol | eol | sym znam
N ⟶ % K N | [O2 | t5
O1 ⟶ t4 O1 | % K O1 | } N
O2 ⟶ " U O2 | % K O2 | t2 O2 |]
U ⟶ % K U | t3 U | "

Rovnakým spôsobom je potom možné zátvorky ďalej reťaziť a kombinovať, čím by sme sa jednoducho dopracovali k jazykom ostatných skupín. Pre účely tejto práce je však zbytočné ich všetky popisovať.

7.2.4 Príkazy nasledované názvom súboru

Asi najjednoduchšou gramatikou popisujúcou chovanie pri náleze príkazu je množina slov nasledovaná reťazcom, ktorý definuje názov súboru. Od ostatných častí je oddelený medzerou. Môže nastať len situácia, kedy po príkaze nasleduje názov súboru alebo komentár(e) a potom názov súboru. Jazyk popisujúci túto skupinu je preto veľmi krátky a vyzerá nasledovne.

G_f = ({S_f, K}, {file*, eol, kom, sym, znam, %}, P, S)

$S_f \rightarrow$ % K S_f | file
K $\rightarrow$ kom eol | eol | sym znam

file* = množina všetkých znakov prípustných pre označenie názvu súboru, nepripúšťa medzery
ostatné terminálne symboly sú rovnaké ako pri gramatikách spomenutých v predošlých kapitolách

7.2.5 Príkaz \position

Tento príkaz je samostatnou kategóriou, pretože ako jediný obsahuje okrúhle zátvorky. Jeho štruktúra je \position () {}. Je teda obdobou príkazov, kde sú jedny hranaté zátvorky nasledované jednými zloženými zátvorkami. Obsah okrúhlych zátvoriek, rovnako ako v prípade hranatých zátvoriek, nepodlieha formátovaniu (nie je v nich sledovaný ani výskyt úvodzoviek). Naopak v prípade zložených zátvoriek ich obsah sledujeme, pretože je v nich pravdepodobný výskyt prostého textu. Formálny jazyk pre spracovanie príkazu \position vyzerá nasledovne:

G_{pos} = ({K, N, O, O2, S_{pos}}, {t, t2, t3, t4, t5, (,), {, }, eol, kom, sym, znam}, P, S_{pos})
t = Σ – {%, (, {, ϵ}
t2 = Σ – {%,), ϵ}
t3 = Σ – {%, “, ϵ}
t4 = Σ – {%, }, ϵ}
t5 = Σ – {%, {, ϵ}
kom = Σ – eol, sym
eol = {\n}
znam = {+, -}
sym = {@ ~. , ! ? ; - * (’}

$S_{pos} \rightarrow$ % K S_{pos} | (O1 | t | { O2
K $\rightarrow$ kom eol | eol | sym znam
N $\rightarrow$ % K N | { O2 | t5
O1 $\rightarrow$ t2 O1 | % K O1 |) N
O2 $\rightarrow$ % K O2 | t4 O2 | }

7.2.6 Bloky nepodliehajúce detekcii

Táto množina obsahuje príkazy, ktoré začínajú určitý blok, ktorý nie je vhodné podrobiť detekcii chýb. Patria sem napríklad chemické, matematické, prípadne programové bloky. Tie sú spúšťané príkazmi \startchemical a \startmath. Toto sú typické príklady, kedy je vhodné vynechať celé bloky. Špeciálnou kategóriou bloku je párový príkaz \starttyping, ktorý v prostredí ConTEXt-u zachováva vo výstupnom súbore formátovanie tak, ako je vysadené v zdrojovom súbore ConTEXt-u. Je preto otázne, či je alebo nie je vhodné detekovať typografické javy v tejto oblasti. Z tohto dôvodu je v konfiguračnom súbore možnosť voľby. V skutočnosti však k tejto množine spúšťacích príkazov náleží aj druhá množina, ktorá každému príkazu typu \startblok priraďuje jeho párový príkaz typu \stopblok, ktorý ho ukončuje. Kompletný zoznam blokov je uvedený v konfiguračnom súbore, ale aj v prílohe rovnako ako aj množiny všetkých ostatných príkazov. V princípe gramatika popisujúca správanie automatu pri čítaní blokov funguje tak, že ak nájde za spätným lomítkom slovo spúšťajúce niektorý z blokov, ktorý nepodlieha detekcii, číta vstup bez ohľadu na to, čo je jeho obsahom. Výnimkou sú len komentáre a spätné lomítka. V prípade komentáru sa správa rovnako ako všade inde. V prípade spätného lomítka prečíta príkaz a v prípade, že sa jedná o príkaz ukončujúci daný blok, táto gramatika korektne skončí a program sa vráti do stavu, kedy hľadá riadiaci znak na najvyššej úrovni. Formálny zápis tejto gramatiky vyzerá takto:

G_{blok} = ({S_{blok}, K, W}, {%, \, t, stop, word, eol, kom, sym, znam, P, S_{blok})
t = Σ – {%, \, ϵ}
word = príkaz neukončujúci blok (word = {príkazy} – {stop})
stop = slovo typu \stopblok ukončujúce blok
kom = Σ – {eol, sym}
eol = {\n}
znam = {+, -}
sym = {@ ~. , ! ? ; - * (’}

$S_{blok} \longrightarrow$ \W | % K S_{blok} | t
K $\longrightarrow$ kom eol | eol | sym znam
W $\longrightarrow$ stop | word S_{blok}

7.2.7 Bezparametrické príkazy

Je samostatná skupina, ktorá má vlastnú množinu príkazov, no správa sa rovnako ako v prípade nálezu neznámeho príkazu. V prípade, že automat nájde príkaz z množiny bezparametrických, jednoducho sa vráti do stavu hľadania riadiaceho znaku najvyššej úrovne.

8 Detekcia a korekcia

V momente nálezu prostého textu prvým automatom, vstupujeme do druhej časti programu. Jedná sa o procedurálne hľadanie typografických chýb na vstupnom reťazci. Tento reťazec už neobsahuje značkovaný text nástroja ConTEXt. Táto druhá časť bude takisto popísaná formálnym jazykom a spracovávaná automatom, ktorý na vstupnej páske hľadá sledovaný symbol a na základe množiny dostupných stavov je ďalej schopný rozhodnúť, či sa v danom momente jedná o chybu v typografii alebo nie. V prípade nálezu chyby pridá záznam do zoznamu chýb, kde je zaznamenaný riadok chyby, číslo znaku na riadku a popis chyby. Tento zoznam neskôr použije pre spracovanie súboru s výpisom nájdených chýb a návrhmi opráv.

Z dôvodu, že niektoré javy typografie sa sledujú aj spätne, bude tento automat v niektorých bodoch modifikovaný oproti klasickému automatu, ktorý rozhoduje o príslušnosti vety do jazyka. Uvediem príklad, kedy pri náleze bodky vo vete, nestačí skontrolovať, či za ňou nasleduje medzera. Je potrebné, skontrolovať, či jej medzera nepredchádza, čo by bolo rovnako považované za chybu. Tu sa dostávame k jadru problému, čím je sledovanie znakov spätne. Automat v jeho definícii číta vstupnú pásku zľava doprava a k prečítaným znakom sa nevracia. Bolo by však veľmi neefektívne v tomto prípade sledovať každú jednu medzeru a v prípade, že by po nej nasledovala bodka, zapísať chybu, resp. ďalej sledovať postupnosť znakov po nej, kde by sa mohla vyskytnúť ďalšia chyba (medzier sa v súbore nachádza veľmi veľa a spúšťať procedúru pri každom jej náleze by určite nebolo optimálnym riešením). Automat si preto bude sledovať aj predchádzajúce znaky, aby v prípade nálezu sledovaného znaku mohol posúdiť správnosť aj spätne. Všetky procedúry obsahujú „string buffer“, do ktorého jednotlivé znaky vkladajú. Nie je preto problém vybrať z neho posledný znak (v niektorých prípadoch aj viac znakov). Po ukončení procedúry detekcie a korekcie chýb sa tento buffer pomocou príkazu „concat“ spojí do jedného reťazca, ktorý je výstupom zapísaným do výstupného súboru. V procedúre automaticky prebieha aj korekcia, takže v spomenutom prípade, kedy by chýbala pred bodkou medzera, by odstránil posledný znak bufferu a pokračoval ďalej v čítaní. Takto upravené reťazce sú neskôr použité pri návrhoch riešení chýb.

Jednotlivé javy sú vyhodnocované samostatnými funkciami. Riešenie je tak prehľadnejšie a je možné vypínať detekciu jednotlivo pre každý jav. Detekcia a korekcia prebiehajú pre tieto javy (v tomto poradí):

1. viacnásobné medzery;
2. bodka;
3. čiarka;
4. výkričník;
5. otáznik;

6. bodkočiarka;
7. úvodzovky;
8. zátvorky;
9. spojovník a pomlčka;
10. odsuvník (apostrof).

Funkcia korekcie dvojitých medzier je nutne prvá, pretože ostatné funkcie nerátajú s ich prítomnosťou a preto napríklad po náleze znaku čiarky, kontrolujú len jeden prechádzajúci znak, ktorý by spravidla mal byť odlišný od medzery. Ak by sa však pred čiarkou nachádzali dve medzery, program by odstránil len jednu a chyba by v takomto prípade v súbore ostala. Je preto vhodné mať túto funkciu vždy zapnutú (ak je to možné a vhodné pre užívateľa). Vyhneme sa tým neúplným opravám, ktoré by mohli viacnásobné medzery spôsobiť. Na poradí zvyšných funkcií až tak nezáleží, pretože sú navrhnuté tak, aby sa nedostali do vzájomných kolízií. Ich poradie je stanovené na základe úvahy autora, s prihliadnutím na sledované pravidlá, prioritu pravidiel, prípadne zložitosť funkcií.

Jednotlivé funkcie detekcie budú ďalej popísané vo vlastných podkapitolách. Budú v nich opísané všetky variácie a výnimky v sadzbe spomenutých typografických javov, ktoré sú schopné funkcie rozlíšiť. Opísané budú slovne spolu s vysvetlením, ale aj formálnym jazykom, popisujúcim jednotlivé typografické javy.

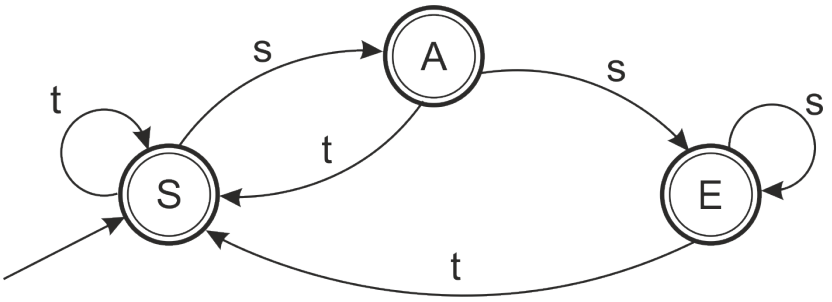
8.1 Viacnásobné medzery

Funkcia, ktorej názov hovorí sám za seba. Za medzeru sa v tomto prípade považuje znak medzery (ASCII kód 32) a znak vlnky (ASCII kód 126) v súbore nástroja ConTEXt reprezentujúci nezalomiteľnú medzeru (ak nebol označený spätným lomítkom – to by znamenalo, že znak sa vysadí do textu a neznamená medzeru). Funkcia vo vstupnom reťazci hľadá miesta, kedy sa medzery, vlnky, prípadne ich kombinácia, nachádzajú vedľa seba. Takéto sekvencie funkcia považuje za chybné a celú sekvenciu nahradí jediným znakom medzery. Keďže táto funkcia má len toto jediné pravidlo, aj jej formálny jazyk je veľmi jednoduchý. Neterminálnym symbolom E bude označený stav, ktorý zapisuje chybu. Po chybe čítanie samozrejme pokračuje a nie je prerušené, takže je možné po prečítaní celej sekvencie medzier vrátiť sa na začiatok. Keďže reťazec môže skončiť v akomkoľvek bode, môžeme zároveň všetky stavy považovať za výstupné (označené terminálnym symbolom e, ktorý znamená nález koncu reťazca). Znamená to, že čítanie môže skončiť sekvenciou medzier, textom, prípadne medzerou. Reťazec môže končiť medzerou, pretože sa môže vo vstupnom súbore vyskytovať časť: Nástroj \CONTEXT\ slúži… V takomto prípade, by sa podľa automatu vyhľadávajúceho textové časti, považovali za reťazec časti „Nástroj “ a „ slúži“. Medzera je teda prípustná na konci ako v prípade prvého spomenutého reťazca.

Formálny jazyk detekcie viacnásobných medzier je navrhnutý tak, aby rozlišoval prípustný stav A (jedna medzera) a chybný stav E (viac medzier). Toto je dôvodom, že zápis nie je zjednodušený len na stavy A a S – čítanie medzier a čítanie zvyšných znakov. V prípade zjednodušenia by nebolo možné rozlíšiť dĺžku sekvencie medzier – stav E (ktorý by v prípade zápisu regulárnej gramatiky podľa stanovených pravidiel bol zbytočný) je dostupný len v prípade dvoch a viac medzier. Formálny jazyk pre funkciu detekcie viacnásobných medzier vyzerá nasledovne:

G = ({S, A, E}, {s, t, e}, P, S)
s = {~, ␣}
t = Σ – {s}
e = znak nil – koniec reťazca

S → tS | sA | e
A → tS | sE | e – jedna medzera, ak bude nasledovať text, je to v poriadku
E → tS | sE | e – viac medzier, číta medzery až kým nenajde text a zapíše chybu



Obrázok 8.1 Konečný automat znázorňujúci detekciu viacnásobných medzier

8.2 Bodka

Zdrojom pravidiel písania bodky je norma ČSN 01 6910. V nej sa uvádza, že: „Všetky interpunkčné znamienka (okrem pomlčky a výpustku) sa pripájajú k predchádzajúcemu slovu, skratke, značke alebo číslu bez medzery. Za každé interpunkčné znamienko náleží medzera. Výnimkou sú zápisy, v ktorých sa interpunkčné znamienka považujú za súčasť ucelenej značky (napr. bodka vo webových adresách alebo e-mailových adresách). Ak nasleduje viac interpunkčných znamienok za sebou, me-

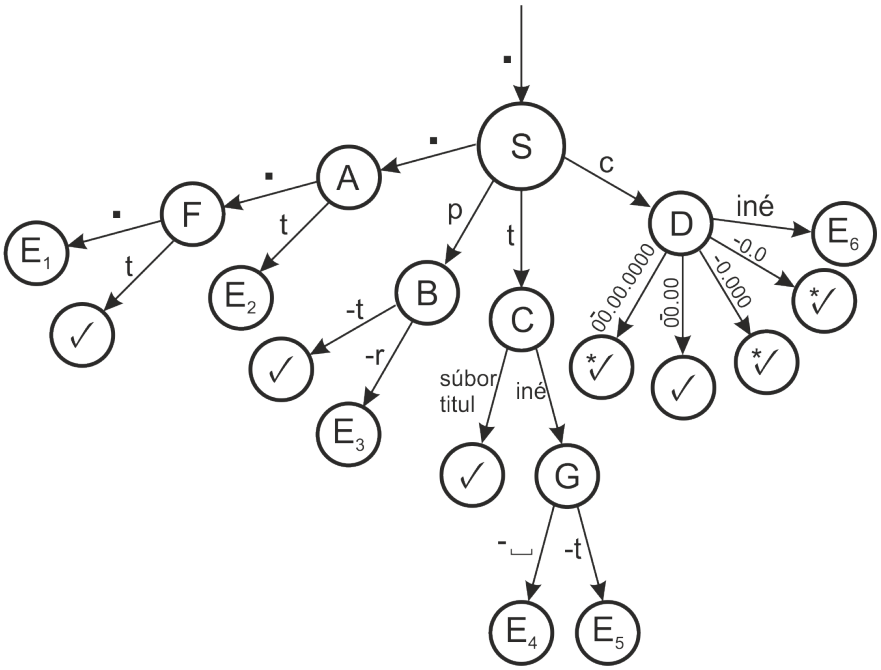
dzera patrí za posledné z nich. Bodka za skratkou na konci vety platí aj ako bodka za vetou.“ (Český normalizační institut, 2014)

Okrem vymenovaných základných pravidiel existujú samozrejme výnimky. Detektor typografie implementovaný v tejto práci, sleduje tieto ďalšie javy a výnimky:

- nevhodná sekvencia bodiek – v prípade nálezu dvoch bodiek je možnosť voľby v konfiguračnom súbore (nahradí jednou alebo výpustkom). V prípade nálezu troch bodiek nahradí výpustkom (v nástroji ConTEXt definovaným príkazom \dots). Ak nájde dlhšiu sekvenciu bodiek, nahradí ju takisto výpustkom;
- strojový dátum vo formáte 01.05.2015 – jedná sa o akceptovateľný zápis. V celom súbore by sa však mal vyskytovať len jeden typ zápisu dátumu. Strojový alebo častejšie zaužívaný vo formáte 1. 5. 2015. Je preto opäť možné nastaviť v konfiguračnom súbore, či je strojový zápis dátumu akceptovateľný. Ak áno, automat tento zápis vyhodnotí za správny a nepovažuje ho za chybu, v opačnom prípade tento zápis vyhodnotí ako chybu. V tomto konkrétnom prípade, by našiel hneď dve chyby. Prvú za zápisom dňa a druhú za zápisom mesiaca;
- zápis niektorých titulov, kde sa bodka vyskytuje medzi znakmi a nie je sprava oddelená medzerou. Typickým príkladom je titul Ph.D.;
- zápis najčastejšie používaných typov súborov. Množina prípon súborov, obsahuje sto najčastejšie používaných prípon. Automat tak v prípade, že bodka nie je oddelená medzerou, pripúšťa možnosť, že môže nasledovať reťazec patriaci do množiny prípon. Súbor automat.zip, by z tohto dôvodu nebol v texte považovaný za chybu. Rovnako tak, by za chybu nebol považovaný zápis .zip;
- zápis času vo formáte 10.15;
- zápis kapitol súboru vo formáte: Kapitola 5.5 alebo Kapitola 5.5.;
- posledným javom je oddeľovač tisícov, ktorý sa v normálnom texte nepoužíva, ale existujú prípady, kedy by užívateľ mohol takýto zápis použiť. Táto možnosť je preto opäť nastaviteľná z konfiguračného súboru – prednastavene je však vypnutá.

Pre spracovanie znaku bodky je podľa uvedených pravidiel a javov navrhnutá funkcia, ktorá pri jej náleze analyzuje okolité znaky. Správanie tejto funkcie je znázornené v nasledujúcom rozkladovom strome. Rozkladový strom na svojom konci dospeje do stavu, ktorý buď prijme, alebo odmietne daný výskyt bodky. Znaky kontrolované spätne sú označené znakom mínus (–t teda znamená výskyt text pred bodkou).

Automat teda funguje tak, že v prípade nálezu znaku bodky, vstupuje do tohto rozhodovacieho stromu. Začína v stave S a ďalej sa rozhoduje na základe nasledujúceho znaku. Má štyri možnosti – ďalšia bodka, množina prípustných znakov po bodke p, číslica c alebo text t (zahŕňa písmená, ale aj ostatné znaky, ktoré nepatria do zvyšných troch kategórií). Množiny titulov a prípustných súborov budú uvedené v prílohách a konfiguračnom súbore. Vetvy iné označujú fakt, kedy sa daný typ nehodí do inej vetvy – v prvom prípade keď sa slovo za bodkou nenachádza ani v množine titulov ani v množine súborov a v druhom prípade keď číselný formát identifikovaný programom nezapadá do žiadneho formátu z prípustných. Označenie v listoch je vždy buď



Obrázok 8.2 Rozhodovací strom funkcie pre detekciu chýb v bodkách

E_i, alebo znak ✓. Hviezdička pri fajke znamená, že ak je v konfiguračnom súbore táto možnosť vypnutá, výsledkom bude chyba. Popis jednotlivých chýb:

- E_1 – sekvencia viac ako troch bodiek;
- E_2 – sekvencia dvoch bodiek;
- E_3 – pred bodkou je nevhodný znak z množiny r, ktorá obsahuje medzeru alebo zlom riadku;
- E_4 – za bodkou nie je medzera a pred bodkou je;
- E_5 – za bodkou ani pred bodkou nie je medzera;
- E_6 – za bodkou nie je medzera (nespadá do žiadnej z možností čísel okolo bodky).

Po ukončení stromu a stanovení verdiktu nad výskytom bodky, prechádza program naspäť do stavu čítania vstupného reťazca a číta vstup až do ďalšieho nálezu bodky, prípadne do konca vstupného reťazca.

```
Príklad:
Prikladný text obsahujúci chyby .Ich prítomnosť je nájdená . Následne
opravená.. Dňa 1.5.2015 o~15.15 v~súbore file.zip.

Oprava:
Prikladný text obsahujúci chyby. Ich prítomnosť je nájdená. Následne
opravená. Dňa 1. 5. 2015 o~15.15 v~súbore file.zip.
```

8.3 Čiarka

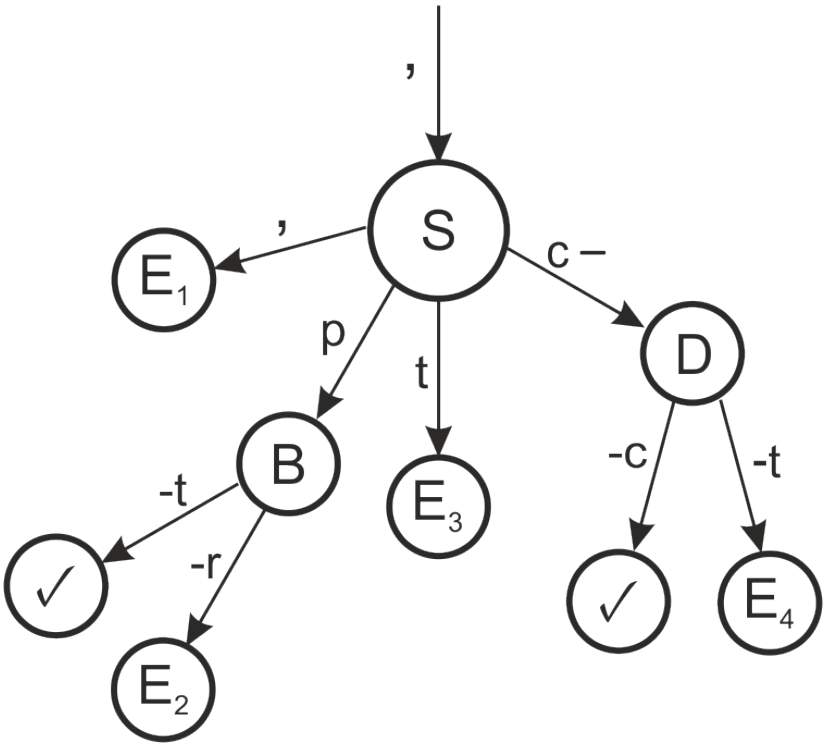
Pravidlá v písaní čiarky sú väčšinou rovnaké ako pri písaní bodky. Pred čiarkou, rovnako ako pred bodkou, sa nevyskytuje medzera. Za ňou sa naopak môže vyskytovať len medzera (prípadne zlom riadku). Od bodky sa teda líši tým, že za čiarkou sa nevyskytuje iná interpunkcia. Keďže písanie čiarky má oproti bodke menej výnimiek, je množstvo sledovaných javov menšie. Výnimky sledované pri výskyte čiarky sú:

- Vyjadrenie peňažnej sumy, kedy po čiarke pripúšťame možnosť výskytu pomlčky. Napr. Kč 50,–.
- Sekvencia čiarok – možný je výskyt len jednej čiarky. Viac čiarok po sebe je označených za chybu a nahradený jednou.
- Vyjadrenie desatinnej čiarky – v prípade, že sa okolo čiarky z oboch strán nachádza číslica, je tento zápis akceptovaný a považovaný za desatinnú čiarku. Ak je číslica len z jednej strany, zápis je chybný (neplatí ak je z ľavej strany a z pravej je medzera).

Popis rozhodovacieho stromu:

- *c* označuje číslicu (v tomto prípade je spojená s vetvou pomlčky);
- *p* je množina prípustných znakov za čiarkou (medzera, vlnka, koniec riadku alebo reťazca);
- *r* je množina neprípustných znakov pred čiarkou (medzera, vlnka, koniec riadku);
- *t* označuje všetky znaky, ktoré nie sú prípustné v druhej na rovnakej úrovni (doplnok druhej vetvy na množine sigma).

Spracovanie prebieha rovnako ako u bodky, pri náleze čiarky sa spúšťa stromová štruktúra, ktorá rozhodne o správnosti zápisu. Ak je znak rôzny od čiarky, jednoducho ho vloží do bufferu a pokračuje v čítaní vstupu. Pri všetkých procedúrach ďalej platí aj to, že kontrola predošlého znaku prebieha vždy. V prípade, že by pred sekvenciou čiarok bola medzera, dokáže ju detekovať a opraviť v rámci jednej chyby.



Obrázok 8.3 Rozhodovací strom funkcie pre detekciu chýb v čiarkách

```
Príklad:
Tu je chybná veta ,,opravím ju. Ak chýba medzera,doplním ju , poznám aj
desatinné čísla 5,5.

Oprava:
Tu je chybná veta, opravím ju. Ak chýba medzera, doplním ju, poznám aj
desatinné čísla 5,5.
```

Strom je zobrazený bez týchto vetiev, ktoré by sa nachádzali vždy na konci chybných vetiev a obsahovali by ďalšie dva chybné listy – chyby by sa líšili len v tom, že jeden by opravil len sekvenciu čiarok, no druhý by okrem sekvencie čiarok opravil aj nadbytočnú medzeru. Toto však nie je potrebné zobrazovať pre všetky vetvy stromu a stačí skončiť v bode, kedy je jasné, že daný zápis je chybný. Rovnako by sa v tomto konkrétnom prípade kontroloval aj znak po sekvencii čiarok – tam by sa mala nachádzať medzera (prípadne zlom riadku). V prípade, že by medzera chýbala, program ju automaticky doplní. Od tohto v strome takisto abstrahujeme, no program aj tento prípad opravuje. Stavy E_i z obrázku opäť označujú nález chyby, ich popis je nasledovný:

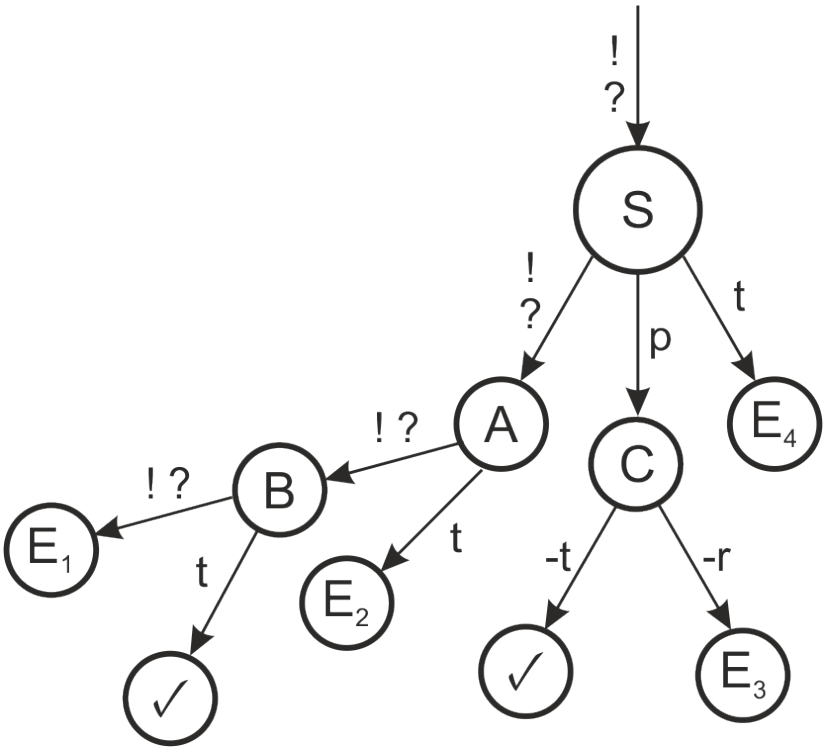
- E_1 sekvencia čiarok
- E_2 medzera pred čiarkou
- E_3 chýba medzera za čiarkou
- E_4 za čiarkou je číslica alebo pomlčka, no pred ňou nie je – nejedná sa o výnimku a teda chýba medzera za čiarkou

8.4 Výkričník a otáznik

Pravidlá pre písanie výkričníku a otázniku sú podobné ako u ostatných interpunkčných znamienok a zároveň takmer identické medzi sebou. V programe majú tieto dva javy každý samostatnú funkciu, aby bolo možné ich ovládať samostatne. No pre oba platí, že zľava sú pritlačené k slovu bez medzery a sprava sa naopak medzera nachádza. Výnimkami, kedy sa po týchto dvoch znamienkach nenachádza medzera je výskyt uzatváracích zátvoriek alebo úvodzoviek hneď za nimi, prípadne prítomnosť dvojíc (?! alebo !?). Okrem sledovania prítomnosti medzier pred a za znakom, sa sleduje ešte jeden typografický jav. Tým je sekvencia rovnakých znakov (výkričník alebo otáznik). Platí, že na sekvenciu dvoch program len upozorní, sekvenciu troch považuje za správnu, a sekvencie dlhšie ako tri označí za chybu, pričom ich nahradí sekvenciou troch znakov výkričníku alebo otázniku. Chovanie programu v prípade nálezu jedného z týchto dvoch znakov môžeme vidieť na prechodovom strome, uvedenom na obrázku 8.4.

Samozrejme platí, že v celej vetve sekvencie sa vždy musí jednať o ten istý znak. Označenie platí rovnaké ako pri strome čiarky a zároveň platí aj to, že po náleze chyby sa kontrola nekončí. Príkladom môže byť vetva, ktorá po náleze výkričníku nenájde ďalej prípustný znak p, ani ďalší výkričník a skončí tak chybou – nenájdený prípustný znak po výkričníku. Medzeru tam samozrejme vloží no ešte pred ukončením celého rozhodovania skontroluje, či sa náhodou nenachádza medzera pred výkričníkom. Ak áno, zmaže ju a až v tomto momente je ukončená celá detekcia daného výkričníku. Toto opäť platí aj pre sekvenciu rovnakých znakov. Chyby v tomto type kontroly sú:

- E_1 sekvencia viac ako troch znakov (? alebo !);
- E_2 upozornenie na výskyt dvoch – neopravuje;
- E_3 neprípustný znak pred jedným zo sledovaných znakov;
- E_4 za sledovaným znakom chýba medzera.



Obrázok 8.4 Prechodový strom funkcií výkričníku a otáznika

8.5 Úvodzovky

Úvodzovky sa používajú hlavne pri zápise priamej reči, citácii, zápise názvov alebo nadsadených slov, prípadne pri použití slov hovorových a podobne. Pri zápise úvodzoviek platí viacero pravidiel. Prvým, ktorým je vhodné začať je, že v českých a slovenských textoch sa používajú úvodzovky v tvare čísel 6 (otváracie) a 9 (uzatváracie). Otváracie úvodzovky sa píšu dole a uzatváracie hore. Hovoríme o znakoch „ , ’ “. Zároveň platí, že uvedené jednoduché úvodzovky sa používajú pri vnorení do dvojitých. Ostatné úvodzovky, ktoré je niekedy možné vidieť, prípadne ich zápis sa označuje za anglický – teda použitie otváracích úvodzoviek hore “ (znak rovnaký ako české uzatváracie úvodzovky) a uzatváracích úvodzoviek taktiež hore, ktoré sú ale otočené opačne ako české úvodzovky ”. Ďalšou možnosťou je použitie strojových úvodzoviek, ktoré sú rovné a oboje sa píšu hore. Ich znak vyzerá ".

Program kontroluje výskyt úvodzoviek počas celého čítania vstupného súboru a na konci vyhodnotí, či je počet párny. Ak nie je, program na to vo výsledku upozorní. Program takisto upozorňuje aj na výskyt anglických úvodzoviek v texte.

Čo sa ďalších pravidiel týka, tak pri písaní úvodzoviek platí, že otváracie úvodzovky sú zľava oddelené medzerou a sprava sú pritlačené k textu bez medzery. Výnimkou pre otváracie úvodzovky sú len otváracie zátvorky, ktoré sa môže vyskytnúť pred nimi a pre vnorené úvodzovky, môžu byť predchodcom aj dvojité úvodzovky. Uzatváracie sú presne naopak, teda zľava je text bez medzery a sprava je medzera. Výnimkou je prítomnosť interpunkcie za uzatváracími úvodzovkami, uzatváracích zátvoriek alebo pre jednoduché úvodzovky aj prítomnosť uzatváracích dvojitých.

Program ďalej nepripúšťa možnosť výskytu otváracích úvodzoviek na konci riadku a uzatváracích na začiatku riadku. Takúto situáciu vyhodnotí ako chybu a úvodzovky presunie – otváracie na začiatok ďalšieho riadku a uzatváracie na koniec predošlého. Posledným javom, ktorý program sleduje pri písaní úvodzoviek, sú ich sekvencie rovnakých znakov – dvoje otváracie dvojité po sebe a pod.

8.6 Bodkočiarka

Sadzba bodkočiarok nemá v podstate žiadne výnimky. Platí preto, že za ňou nasleduje medzera, uzatváracie zátvorky alebo úvodzovky, prípadne zlom riadku. Nijak sa teda nevymyká pravidlám. Jediná vec, ktorou sa líši jej kontrola oproti ostatným, je špeciálny prípad voľby pre užívateľa, ktorý má možnosť zapnúť sekvencie bodkočiarok. Táto voľba je podľa východiskového nastavenia vypnutá, no zmysel má vtedy, ak by sa v súbore nachádzali úseky textu typické pre formát .csv. V tom sa spravidla používa bodkočiarka ako oddeľovač a prípade, že je pole prázdne, je možné použiť zápis pole;;;pole2;;pole3. Užívateľ preto môže vypnúť opravu v takýchto častiach a okrem toho, že program nevyhodnotí sekvenciu ako chybu, tak ani chýbajúca medzera po poslednom znaku sekvencie nie je vyhodnotená za chybu.

8.7 Zátvorky

Zátvorky majú identické pravidlá ako úvodzovky, nie je preto potrebné ich nejako zvlášť rozpisovať. Oproti úvodzovkám, zátvorkám nevadia sekvencie rovnakých znakov. Pre pripomenutie len v krátkosti:

- otváracie zátvorky
  - pred nimi je medzera, otváracie zátvorky alebo otváracie úvodzovky, začiatok riadku,
  - za nimi nikdy nie je medzera alebo zlom riadku;
- uzatváracie zátvorky

= pred nimi nikdy nie je medzera alebo zlom riadku,
= za nimi je medzera, interpunkcia, uzatváracie zátvorky alebo uzatváracie úvodzovky, koniec riadku.

8.8 Spojovník a pomlčka

Sú špeciálna kategória, ktorá však kvôli nespočetnému množstvu výnimiek je skôr predmetom samostatnej práce. Všeobecne však platí, že spojovník je z oboch strán pritlačený k textu bez medzier. Pomlčka presne naopak, teda oddelená medzerami (výnimkou je napríklad písanie intervalov 8–10 %). Táto interpretácia je kontrolovaná aj automatom v tejto práci. Z dôvodu veľkého počtu výnimiek, ale program opravuje len prípady, kedy sa z jednej strany pomlčky alebo spojovníku nachádza medzera, pričom z druhej nie. Prípustné sú však aj prípady, kedy sa medzera vyskytuje z jednej strany spojovníku alebo pomlčky. Príkladom môžu byť zápisy: modro- alebo červenožltý, -ný, natý, itý a iné. Ostatné prípady – teda absenciu medzier okolo pomlčky a prítomnosť medzier okolo spojovníku rieši varovaním.

Pre spojovník a pomlčku platia pravidlá:

– spojovník
  = varovanie ak sa okolo spojovníku z oboch strán nachádzajú medzery,
  = ak sa z jednej strany nachádza medzera, zmaže ju a označí za chybu;
– pomlčka
  = varovanie ak sa okolo pomlčky nenachádzajú medzery,
  = ak je medzera len z jednej strany pomlčky, program označí tento výskyt za chybu a doplní medzeru aj na druhú stranu (výnimkou je zápis peňažnej čiastky 50,–).

8.9 Odsuvník

Znak odsuvníku, označovaný aj ako apostrof, sa používa v prípadoch, kedy chcem naznačiť „prehltnutie“ nejakej časti slova alebo vety. Dalo by sa povedať, že apostrof je v pozícii písmena v slove a podľa toho aj píšeme medzery okolo neho. V českých a slovenských textoch sa používa spravidla na konci slova (napr. Upaď na zem.). V takomto prípade sa zľava nachádza text bez medzier a nasleduje medzera, uzatváracie úvodzovky alebo zátvorky, interpunkcia alebo koniec riadku. Niektoré jazyky, však používajú apostrof aj v strede slov – príkladom môžu byť skrátené anglické frázy I'm, didn't, we'll a iné, prípadne aj francúzske slová ako c'est alebo l'amour. Táto situácia

je opäť možnosťou voľby v konfiguračnom súbore – primárne je však vypnutá. Pre výnimočné prípady je v konfiguračnom súbore aj možnosť zapnúť apostrof na začiatku slova, táto voľba je však tiež vypnutá. Príkladom takéhoto použitia môže byť zápis: rok ’97.

9 Testy súborov

Testom funkcií bolo podrobených 26 testovacích akademických súborov z oblasti IT v českom a slovenskom jazyku. Rozsah jednotlivých súborov je štyri až šesť strán. Spolu s medzerami to znamená rozsah zhruba 12 000 až 15 000 znakov. Sledované boli všetky vyššie spomenuté javy, ktoré boli následne všetky ručne kontrolované a vyhodnocované. Cieľom bolo, aby javy, ktoré program označí za chyby boli naozaj nesprávnym zápisom, čím by sa v budúcnosti predišlo nesprávnym korekciám v správne zapísanom texte.

9.1 Výsledky testov

Výsledkom testovania je tabuľka 9.1, kde sú zapísané počty chýb v jednotlivých súboroch a ich celkové súčty v každom súbore, ale aj súčty pre každý jav zvlášť. Tabuľka 9.2 uvádza celkový počet výskytov jednotlivých javov, ktorá slúži k presnejšej predstave o tom, koľkokrát sa napríklad bodka vyskytuje v danom súbore, prípadne v súčte vo všetkých súboroch dohromady. Keďže väčšinu sledovaných javov tvoria medzery, sú v tabuľkách uvedené aj počty chýb a celkový počet výskytov javov bez medzier.

Tabuľka 9.1 Počty chybných zápisov pre jednotlivé javy

Súbor	.	,	!	?	;	- —	( )	␣	„“	’	Σ s ␣	Σ bez ␣
1	5	1	0	0	0	0	0	2	0	0	8	6
2	6	1	0	0	0	6	0	10	0	0	23	13
3	1	0	0	0	0	0	0	8	1	0	10	2
4	2	1	0	0	0	0	0	4	0	0	7	3
5	5	0	0	0	0	12	0	4	0	0	21	17
6	1	0	0	0	0	0	1	1	0	0	3	2
7	11	1	0	0	0	0	0	18	0	0	30	12
8	6	1	0	1	0	6	0	3	0	0	17	14
9	1	3	0	0	0	1	1	24	2	0	32	8
10	7	2	0	0	0	5	0	23	0	0	37	14
11	2	4	0	0	0	0	0	0	0	0	6	6
12	0	0	0	0	0	0	0	16	2	0	18	2
13	2	0	0	0	0	2	0	3	0	0	7	4
14	0	0	0	0	0	0	0	3	0	0	3	0
15	0	0	0	0	0	0	0	0	0	0	0	0
16	1	1	0	0	0	0	0	10	0	0	12	2
17	0	0	0	0	0	4	1	11	0	0	16	5
18	0	2	0	0	0	8	1	4	2	0	17	13
19	1	0	0	0	0	0	1	14	2	0	18	4
20	3	0	0	0	0	3	0	6	0	0	12	6
21	0	0	0	0	0	7	0	51	0	0	58	7
22	3	0	0	0	0	0	1	1	1	0	6	5
23	0	0	0	0	0	2	0	9	0	0	11	2
24	1	0	0	0	0	0	0	37	0	0	38	1
25	6	1	0	1	0	4	1	9	2	0	24	15
26	1	4	0	1	0	0	0	9	0	0	15	6
Spolu	**65**	**22**	**0**	**3**	**0**	**60**	**7**	**280**	**12**	**0**	**449**	**169**

Tabuľka 9.2 Celkové počty výskytov jednotlivých javov

Súbor	.	,	!	?	;	- —	( )	␣	„“	’	Σ s ␣	Σ bez ␣
1	95	123	0	0	1	27	7	992	1	0	1 246	254
2	106	84	0	0	0	19	27	1 364	10	0	1 610	246
3	135	131	0	1	0	24	6	1 599	4	0	1 900	301
4	70	56	0	0	0	12	7	930	0	0	1 075	145
5	146	62	0	0	0	22	14	1 092	3	0	1 340	248
6	186	153	0	4	1	13	44	2 185	8	1	2 594	409
7	89	108	0	0	0	13	6	1 297	5	0	1 518	221
8	134	173	1	1	0	48	9	2 195	1	0	2 562	367
9	127	145	0	1	0	24	15	1 909	4	0	2 225	316
10	138	202	1	0	0	37	17	2 063	6	0	2 464	401
11	168	179	0	0	11	18	27	1 834	0	1	2 238	404
12	123	153	0	1	0	4	13	2 333	2	0	2 629	296
13	185	187	0	7	4	21	28	2 421	4	2	2 859	438
14	142	131	0	0	1	10	21	1 878	6	0	2 189	311
15	70	68	0	0	0	7	11	1 160	2	0	1 318	158
16	110	101	0	0	0	14	2	1 751	2	0	1 980	229
17	82	120	0	0	0	14	17	1 479	1	0	1 713	234
18	115	151	0	2	0	21	9	1 892	4	0	2 194	302
19	134	88	0	0	1	21	10	1 256	6	0	1 516	260
20	135	158	0	3	0	36	12	1 556	0	0	1 900	344
21	93	102	0	0	0	21	15	1 294	0	0	1 525	231
22	89	138	0	0	0	11	8	1 082	4	0	1 332	250
23	62	78	0	0	6	3	4	1 174	9	0	1 336	162
24	143	155	0	4	0	83	1	2 227	2	0	2 615	388
25	161	110	0	2	0	34	10	1 589	2	0	1 908	319
26	139	99	0	2	0	34	14	1 566	0	0	1 854	288
Spolu	**3 177**	**3 255**	**2**	**28**	**25**	**591**	**354**	**42 118**	**86**	**4**	**49 640**	**7 522**

Väčšina súborov obsahuje niečo medzi 1 300 až 2 500 sledovaných typografických javov. Ak by sme abstrahovali od medzier, ktoré tvoria väčšinu chýb, ale aj sledovaných javov, pretože sa v textoch objavujú jednoznačne najčastejšie, je toto číslo výrazne nižšie. Dostalo by sa na hodnoty od 150 do 350 sledovaných znakov pre väčšinu súborov. Pre lepšiu predstavu však porovnáme celkové počty chýb v pomere k ich celkovému počtu výskytov. Tieto údaje sú uvedené v tabuľke 9.3.

Tabuľka 9.3 Chybovosti jednotlivých javov

Jav	Počet chýb	Počet výskytov	Chybovosť javu
Bodka	65	3 177	2,05 %
Čiarka	22	3 255	0,68 %
Výkričník	0	1	0,00 %
Otáznik	1	28	3,57 %
Bodkočiarka	0	25	0,00 %
Spojovník	60	591	10,15 %
Zátvorky	7	354	1,98 %
Medzery	280	42 118	0,66 %
Úvodzovky	12	86	13,95 %
Odsuvník	0	4	0,00 %
Spolu	**447**	**49 640**	**0,90 %**

V tabuľke 9.4 opäť uvádzam aj prehľad bez zaznamenaných medzier, kde je percentuálna chybovosť značne vyššia. Spôsobené je to hlavne tým, že samotné medzery tvoria v porovnaní s ostatnými znakmi zhruba 85 % zo všetkých výskytov sledovaných javov. Percentuálna chybovosť je tak vo väčšej miere závislá na chybovosti v medzerách.

Tabuľka 9.4 Chybovosti jednotlivých javov (bez medzier)

Jav	Počet chýb	Počet výskytov	Chybovosť javu
Bodka	65	3 177	2,05 %
Čiarka	22	3 255	0,68 %
Výkričník	0	1	0,00 %
Otáznik	1	28	3,57 %
Bodkočiarka	0	25	0,00 %
Spojovník	60	591	10,15 %
Zátvorky	7	354	1,98 %
Úvodzovky	12	86	13,95 %
Odsuvník	0	4	0,00 %
Spolu	**167**	**7 521**	**2,22 %**

V tabuľke 9.4 je podľa očakávaní chybovosť vyššia, pretože v medzerách nie je veľa možností ako spraviť chybu. Väčšinou sa jedná o preklep alebo nepozornosť, kedy vložíme do textu omylom dve medzery za seba. Nejedná sa o nejakú veľkú chybu, no pri častejších výskytoch môžu viacnásobné medzery pôsobiť rušivo a neesteticky.

Zameriame sa však hlavne na ostatné javy, ktoré vykázali vyššiu chybovosť. Začneme bodkou, ktorá má niečo cez dve percentá výskytov zapísaných chybne. Po sledovaní nájdených chýb môžem tvrdiť, že väčšinu chýb pri písaní bodky tvorila chýbajúca medzera za ňou (najčastejšie chýbajúca medzera za skratkou). Opakovali sa však aj iné výskyty, ako napríklad omylom zapísané dve bodky alebo sekvencia troch bodiek namiesto výpustku. Len zriedkavo alebo skoro vôbec sa vyskytovali chyby, kedy by bola medzera z oboch strán bodky, prípadne medzera zapísaná omylom pred bodkou a nie za ňou. Vyššia chybovosť v písaní bodiek však nie je žiadnym prekvapením, pretože ich výskytov a výnimiek existuje pomerne veľa – či už v skratkách, dátumoch, čase a podobne.

O niečo jednoduchší je zápis čiarky, ktorý už pri uvedení sledovaných pravidiel napovedal, že má len dve sledované výnimky, ktoré program sleduje. Tými sú desatinná čiarka a zápis peňažnej sumy, kedy musí nasledovať pomlčka a pred čiarkou sa musí vyskytovať číslica. Na základe tohto možno tvrdiť, že vysoká väčšina výskytov bodiek bude klasicky na vyčlenenie časti vety. To znamená, že pred ňou sa nachádza text, zatiaľ čo za ňou medzera. Tento zápis je však takmer každému známy a chybovosť je tým pádom výrazne nižšia – len 0,68 %.

Ďalšou sledovanou skupinou je výkričník, ktorý sa však vyskytol len raz – bez chyby. Dôvodom je hlavne to, že sa jednalo o akademické odborné práce, kde sa výkričník vyskytuje len veľmi málo. Môžeme ho však spoločne vyhodnotiť spolu s otáznikom, ktorý ma rovnaké pravidlá v zápise. Otáznik sa však vyskytol 28-krát, hlavne

v pozícii rečníckej otázky. Jeho zápis bol však zväčša správny, keďže pravidlá pre jeho sadzbu sú pomerne jednoduché – pred ním sa medzera nenachádza, no za ním áno (až na výnimky spomenuté v časti, kde boli pravidlá rozobraté dopodrobna).

Nulovú chybovosť má zápis bodkočiarok, ktorý sa vyskytoval hlavne vo forme oddeľovača vetných častí, prípadne odsekov v odrážkach namiesto čiarky. Niekoľkokrát však aj v zápise intervalov, prípadne iných matematických zápisoch, kde je nevhodné použiť čiarku.

Podobne ako bodkočiarka má nulový počet chýb aj odsuvník, ktorý bol však použitý len štyrikrát. Všetky použitia boli v zápisoch anglických skrátených slov a boli zapísané korektne. Prípady, kedy boli v týchto zápisoch použité horné úvodzovky, prípadne iné nevhodné znamienka, sú považované za chyby v úvodzovkách.

Zátvorky obsahovali chyby len minimálne. Jednalo sa o chýbajúce, resp. nadbytočné medzery okolo nich, ktoré boli však spravidla dielom nepozornosti, pretože v dokumente sa už na iných miestach nevyskytovali.

Posledné dva javy som si nechal nakoniec, pretože ich chybovosť je vyššia ako 10 %, čo už znamená, že každý desiaty zápis (alebo dokonca častejšie) je chybný. Jedná sa o zápis pomlčky a spojovníku, ktoré sú vyhodnocované jednou funkciou a zápis úvodzoviek, kde je taktiež vyhodnocovanie viacerých čiastkových kontrol zahrnuté v jednej funkcii. Čiastkovými kontrolami sa myslí okrem klasických medzier okolo úvodzoviek aj kontrola anglických úvodzoviek, prípadne rovných, strojových úvodzoviek, ktoré môžu byť na základe voľby považované za nevhodné. Chybou takisto môže byť aj použitie niektorého zo znaku úvodzoviek na mieste apostrofu, ako už bolo spomenuté vyššie.

Z dvojice spojovník a pomlčka, tvorí takmer úplnú väčšinu nesprávny zápis spojovníku. Jedná sa o zápisy, kedy je spojovník zapísaný na mieste, kam patrí pomlčka. V týchto prípadoch bol spojovník z oboch strán oddelený od textu medzerami. V prípade náhrady spojovníku znakom pomlčky, by bol zápis korektný a program preto upozorňuje na tieto výskyty.

U úvodzoviek boli typy chýb pomerne rôznorodé. Vyskytli sa chyby ako zabudnuté medzery za uzatváracími úvodzovkami, chýbajúce medzery pred otváracími, ale aj nesprávny zápis, kedy boli použité dvoje úvodzovky hore (otváracie aj uzatváracie), použité anglické úvodzovky, prípadne bola jednoduchá horná úvodzovka použitá ako apostrof a podobne.

Program teda dokázal rozpoznať viacero javov a v každom z nich viac, či menej typov chýb. Je to dôležité hlavne z pohľadu toho, že nielenže identifikuje umiestnenie chyby, ale aj navrhne opravu a dokáže identifikovať, čo je problémom pri tej ktorej chybe. Užívateľ si tak okrem opráv môže aj pozrieť, čo je v jeho zápise nesprávne a nabudúce sa rovnakej chybe vyvarovať.

9.2 Porovnanie výsledkov

Zaujímavé porovnanie výsledkov prináša však aj porovnanie chybovosti, ktoré je uvedené vyššie s chybovosťou, ktoré vo svojej práci uvádza Urbanová (2003). Jej práca bola zameraná takisto na detekciu typografických chýb, no pri ich detekcii využila regulárne výrazy, ktoré boli zamerané na hladanie chýb v neznačkovaných textoch. Výsledky chybovosti javov uvedené v práci Urbanovej (2003), sú uvedené v tabuľke 9.5 (uvedené sú len javy spoločné pre obe práce).

Tabuľka 9.5 Chybovosť sledovaných javov podľa Urbanovej (2003)

Jav	Počet chýb	Počet výskytov	Chybovosť javu
Bodka	22	1 728	1,27 %
Čiarka	12	1 862	0,64 %
Výkričník	0	0	0,00 %
Otáznik	1	13	5,26 %
Bodkočiarka	0	18	0,00 %
Pomlčka	1	284	0,35 %
Zátvorky	3	588	0,51 %
Úvodzovky	2	154	1,30 %
Odsuvník	0	0	0,00 %
Spolu	**47**	**4 888**	**0,96 %**

Keďže sa jednalo o takmer identické súbory, dalo sa očakávať, že výsledky budú nápadne podobné. Podobnosť medzi nimi určite je, no výsledky v tejto práci vykazujú mierne vyššiu chybovosť v najpočetnejších skupinách. Znak bodky má početnosť chýb vyššiu o 0,8 %, čo môže byť spôsobené širším záberom sledovaných javov a výnimiek. Čiarka vykazuje takmer identickú chybovosť, no veľký skok opäť nastáva u úvodzoviek. Chybovosť vykazovaná automatom je niekoľkonásobne vyššia. Spôsobené je to nepochybne tým, že zatiaľ čo regulárny výraz sleduje len prítomnosť, prípadne absenciu medzier, automat kontroluje viacero možností považovaných za chybu. Tie už boli viackrát spomenuté skôr. Ďalším porovnávaným znakom je pomlčka, ktorá bola podľa výsledkov Urbanovej zapísaná chybne len jedenkrát, čo približne zodpovedá výsledkom tejto práce, pretože ako bolo spomenuté, výraznú väčšinu chýb tvoria spojovníky. Tie však nie sú sledované regulárnymi výrazmi a bolo by preto skresľujúce porovnávať tieto výsledky. Podobné výsledky boli dosiahnuté aj pri sledovaní otáznikov, ktorých zápis bol chybný v každej práci raz. Zátvorky vy-

kazujú taktiež podobnú, oproti ostatným javom pomerne nízku chybovosť, čo určite súvisí hlavne s ich pomerne jasne stanoveným zápisom bez výnimiek. Nulovú chybovosť zhodne v oboch prácach vykázali funkcie na detekciu bodkočiarok, výkričníkov a odsuvníkov. U výkričníkov a odsuvníkov je to spôsobené hlavne nízkym počtom výskytov – Urbanová (2003) neuvádza žiadny výskyt v textoch z oblasti IT a v tejto práci je výskyt výkričníku len jeden a odsuvníky boli nájdené štyri.

V závere testovania však môžeme povedať, že výsledky chybovosti interpunkcie vo vybraných testoch nie je zanedbateľná. Výsledkom je, že 2,22 % zápisov interpunkcie je chybných. Ak zoberieme do úvahy, že sa jedná o akademické práce, kde by sa už chyby mali vyskytovať len minimálne, môžeme povedať, že je to pomerne vysoké číslo. Pripočítať môžeme ešte výskyt viacnásobných medzier, ktorých chybovosť udáva hodnotu 0,66 %. Pre lepšiu predstavu môžem uviesť, že testované dokumenty mali spolu 125 strán. Znamená to, že:

- Chyba v zápise bodky bola na každej druhej strane.
- Chyba v zápise čiarky bola na každej šiestej strane.
- Chyba v spojovníku alebo pomlčke na každej druhej strane.
- Chyba v úvodzovkách na každej desiatej strane.
- Dvojité medzery sa v priemere vyskytli dvakrát na jednej strane (občas sa však jednalo aj o nesprávne formátovanie textu – náhrada za tabulátory a podobne – každý takýto výskyt sa však považuje len za jednu chybu).

Vo výsledku môžeme povedať, že na každých piatich stranách bolo v týchto dokumentoch šesť chýb v interpunkcii a desať v medzerovaní. Pre objektivitu však možno povedať, že jeden text bol úplne bez chýb a v jednom z nich sa vyskytli len dve chyby v medzerách.

10 Konfiguračný súbor

Konfiguračný súbor obsahuje časti programu nastaviteľné užívateľom. Aby bolo tento modul jednoduché spracovať programom, je vo formáte .lua, čo znamená, že jeho celý obsah sa považuje za kód. Má preto špecifickú štruktúru a nevhodný zásah do tejto štruktúry by mohol narušiť funkčnosť programu. Jeho štruktúra by sa dala rozdeliť do dvoch častí. Prvá obsahuje prednastavené hodnoty pro sledovanie jednotlivých typografických javov vo forme poľa hodnôt true/false. Toto pole udáva, v ktorých javoch sa budú chyby hľadať (nastavenia je možné meniť pomocou značiek v komentári). Pole vyzerá nasledovne:

```
--check   space  .    ,    !    ?   „"    ;   ()   -    '
settings={true,true,true,true,true,true,true,true,true,true}
```

Okrem tohto poľa, sa však jednotlivé javy a chovanie automatu dajú nastavovať v špecifickejších voľbách, ktoré už boli v práci spomenuté. Ich nastavenia (bez popisu, ktorý je inak v súbore uvedený vo forme poznámok) vyzerajú takto:

```
context_mode=true           -- true value means that … description
brackets_equotation=true    -- true value enables … description
check_comments=false        -- description
check_typing_block=false    -- etc.
two_periods_ellipsis=false
use_dots=false
mistake_description=true
thousands_separator=false
apostrophe_begin=false
apostrophe_middle=true
semicolon_sequence=false
machine_date_format=false
chapters_number=true
english_quotation=true
start_stop_text=false
do_offered_corrections=true
lines_to_output=1
```

Druhou časťou konfiguračného súboru sú množiny príkazov nástroja ConTEXt. Tie sa kvôli prehľadnosti skladajú opäť z dvoch častí. Jednou sú množiny vyhradené užívateľovi pre jednoduché pridanie svojich makier do kontroly. Sú preto prichystané prázdne množiny príkazov s popismi, kde si užívateľ môže jednoducho pridať svoje,

ktoré si pre nástroj ConTEXt nadefinoval. Tieto množiny sa na začiatku behu programu spoja s ostatnými množinami príkazov s rovnakými vlastnosťami. Sú uvedené na spodku konfiguračného súboru tak aby svojim rozsahom neprekážali, ale zároveň tak, aby sa k nim užívateľ mohol jednoducho dostať, upravovať ich, prípadne doplňať ďalšie príkazy. Jednoduchý prístup (bez nutnosti otvárať hlavný program) k týmto množinám je dôležitý, pretože pri výbere príkazov do množín, boli síce zahrnuté všetky príkazy z manuálu a ďalšie pomerne veľké množstvo príkazov často používaných príkazov, ale určite nie všetky. Obsiahnuť všetky príkazy ConTEXt-u ani nie je v ľudských silách a preto je dôležité mať možnosť ich v prípade nálezu chýbajúceho príkazu jednoducho doplniť. Konfiguračný súbor je uvedený v prílohách.

```
--add your own commands:
--put the command to the braces with no backslash in format{"command"}
--bracesA means the {}, where the mistakes are being checked
--bracesB means the {}, where the mistakes are not being checked
--in brackets [] or parentheses () is typography never checked

add_three_brackets={"MyCommand"}    -- \command[][][]
add_two_bracesA={}                  -- \command{A}{A}
add_two_bracesB={"MyCommand2"}      -- \command{B}{B}
…
```

Okrem konfiguračného súboru, kde je možné nastavovať všetky voliteľné možnosti, obsahuje práca aj súbor README.txt, v ktorom je návod na použitie. Obsahuje doplňujúci popis, ktorý je potrebný k správnemu použitiu programu. Priložený je rovnako ako všetky ostatné časti v prílohách.

11 Súbor návrhov opráv

Všetky nezrovnalosti na vstupnej páske, ktoré automat identifikuje ako chyby, sú zaznamenané do matice chýb. Táto matica obsahuje na riadkoch vždy jednu chybu, pričom v prvom stĺpci obsahuje riadok chyby v súbore, v druhom je číslo znaku na tomto riadku a v poslednom stĺpci je krátky popis chyby. Táto matica je výstupom automatu a zároveň vstupom pre procedúru, ktorá generuje súbor s návrhmi opráv (vo formáte .tex alebo .txt). Tento súbor ma slúžiť užívateľovi na jednoduché prezeranie nájdených chýb. Jednotlivé návrhy sú rozdelené do „buniek“. Každá bunka obsahuje súradnice danej chyby, jej popis (voliteľne v konfiguračnom súbore), originálny riadok zo vstupného súboru, kde bola chyba nájdená (vrátane okolia – riadky pred, respektíve za chybným, pričom ich počet je opäť nastaviteľný v konfiguračnom súbore), a samozrejme navrhovaný opravený riadok. V prípade, že sa na jednom riadku nachádza viacero chýb, je ich návrh spojený do jedného.

Pre každý návrh môže užívateľ jednoducho rozhodnúť o tom, či sa má aplikovať alebo nie. Stačí pred slovo „WARNING“, ktoré zahajuje každý blok vložiť znak spojovníku. V takom prípade bude tento navrhovaný riadok z korekcie vynechaný. Ak by však užívateľ súhlasil s tým, že sa jedná o chybu, ale nesúhlasil s návrhom opravy, má možnosť navrhovaný riadok opraviť podľa seba. Program potom jednoducho vloží tento riadok a užívateľ tak nemusí otvárať celý vstupný súbor a hľadať, respektíve opravovať chybu ručne. Dôležité však je zachovať pomerne striktnú štruktúru tohto súboru a to hlavne v dvoch bodoch. Prvým je, že každý blok musí obsahovať slovo WARNING a pred ním môže byť maximálne jeden znak (spojovník alebo plus), ale aj nemusí v takom prípade je voľba závislána nastavení v konfiguračnom súbore. Druhá časť, je úvod riadku NEW, kedy musí ostať nezmenené toto slovo. Za ním je možné vykonávať ľubovoľné zmeny. Tie budú použité pri korekcii (bude teda aplikovaný celý riadok okrem slova NEW a medzery za ním). Príklad jednej bunky súboru s opravami je uvedený nižšie:

```
WARNING! Line: 7 Character(s) number: 8, 19, 38
OLD 6. Riadok pred chybným riadkom.
OLD 7. Riadok ,obsahujúci  chyby v~typografii!Opravíme
OLD 8. Riadok za chybným riadkom.
NEW Riadok, obsahujúci chyby v~typografii! Opravíme
```

12 Diskusia

V tejto časti by sa dalo rozobrať pomerne veľké množstvo vecí, ktoré sa vyskytli buď ako problém, alebo ako príležitosť a priestor pre dodatočné zlepšenie práce. Vyberám však niekoľko z nich, ktoré považujem za najdôležitejšie a najzaujímavejšie.

Prvou spomenutou si jednoznačne zaslúži byť spomenutý problém „pomlčka vs. spojovník“. V rámci tejto práce je detekcia riešená základným pravidlom, ktoré znie: Okolo pomlčky by sa mali vyskytovať medzery a u spojovníku naopak nie. Chybou takisto je výskyt medzery len z jednej strany oboch znakov. Oprava je však väčšinou na užívateľovi, pretože takéto jednoduché pravidlo sa môže často mýliť. Správny zápis týchto dvoch znakov je však pomerne častým problémom, ktorý mnoho ľudí ignoruje. Z toho však potom vyplýva pomerne vysoká chybovosť pri detekcii. Spôsobené to však môže byť aj pomerne vysokým počtom možných výskytov, pravidiel a tým pádom aj výnimiek, ktoré sa pri zápise týchto dvoch znakov vyskytujú. Nie vždy je preto isté, že pomlčka je vyčlenená medzerami a podobne. Časté nezrovnalosti v zápise vznikajú v zemepisných názvoch, prípadne mestských častiach zapísaných v kombinácii s mestom a iných prípadoch.

Ideálnym riešením by bola komplexná analýza, ktorá by identifikovala všetky výnimky a pravidlá zápisu. Pre všetky výnimky by v takom prípade mohol byť vytvorený slovník, na základe ktorého by sa program dokázal v každom prípade exaktne rozhodnúť, či sa jedná o chybu alebo nie. Takéto riešenie by však mohlo byť cieľom samostatnej menšej práce a jej výsledok doplnený do nášho automatu (dokázali by sme tak s oveľa väčšou presnosťou určiť, či je zápis chybný).

Druhým okruhom, ktorým sa diskusia zaoberá, je vynechanie niektorých častí v súboroch ConTEXt-u. Jedná sa o celé bloky, v ktorých je vypnutá detekcia chýb. Typickým príkladom nevhodným pre detekciu sú napríklad bloky začínajúce \startlua, \startMPcode, \startLuacode a iné. Tieto časti súboru obsahujú kód v úplne inej štruktúre, ako je štruktúra nástroja ConTEXt. Bolo by preto nutné vytvoriť ďalší automat podobný tomuto, ktorý by rozdeľoval takto vložený kód v jazyku Lua na štruktúrované a neštruktúrované časti. Podobne tak aj v iných blokoch vynechaných detekciou. Trochu jednoduchšími blokmi, ktoré sú však tiež vynechané, sú napríklad matematické prostredie a chemické prostredie. Tie na rozdiel od vyššie spomenutých prostredí v jazyku Lua neprechádzajú do úplne iného prostredia a ostávajú v ConTEXt-e, no platia tu iné pravidlá. Samotného prostého textu sa tu však väčšinou vyskytuje pomerne málo a častejšie by mohlo prísť k nevhodnému zásahu do matematických alebo chemických vzorcov, prípadne rovníc a podobne. To je však nežiadúce. V prípade záujmu, by bolo vhodnejšie vytvoriť týmto prostrediam vlastné stavy, zaoberajúce sa jednotlivými prostrediami. Pre účely tejto práce to však nie je nevyhnutné.

Posledným bodom diskusie je samotné riešenie opráv, ktoré poskytuje pomerne

široké možnosti úprav a zásahov pre užívateľa, no má však jednu slabú stránku. Ide o riešenie, ktoré rieši opravy tak, že nahradí celý riadok. Toto môže byť nevýhodou v prípade, že sa na jednom riadku nachádza veľa rôznych chýb, z ktorých užívateľ niektoré chce a niektoré nechce aplikovať. V takom prípade potrebuje úpravy dokončiť sám a prepísať ich tak, ako sám potrebuje, aby mohol byť vložený celý riadok. Spomedzi riešení sa toto však stále zdalo byť najlepšie, pretože síce v niektorých prípadoch môže spôsobiť trochu práce navyše, no väčšinou skôr urýchli prístup k opravám. Taktiež dáva užívateľovi možnosť jednoducho si prispôsobiť zvolený spôsob opravy.

Ďalšími uvažovanými riešeniami, ktoré by možno stáli za zváženie, pretože by mohli byť v niektorých prípadoch efektívnejšie sú: návrh a oprava každej chyby zvlášť, prípadne druhá možnosť, kde by program umožňoval okrem jednotlivého spravovania chýb, zakázať aj všetky opravy rovnakého typu. Rovnako ako implementované riešenia, aj tieto majú však svoje nevýhody. Prvé spomenuté riešenie by bolo lepšie v prípadoch, kedy sa na jednom dlhom riadku vyskytuje veľa chýb, pretože by pre každú chybu malo vlastný výpis a návrh. Oprava by potom prebiehala pre každú chybu samostatne. Aplikovať by sa však dalo len navrhované riešenie, pretože v prípade zásahu užívateľa by mohli v programe vznikať kolízie. Druhé riešenie má takisto svoje pre a proti. Výhodou by jednoznačne bola možnosť, ktorá by zakázala všetky opravy s rovnakým popisom. Príkladom môže byť vloženie medzery za bodku v čísle kapitoly (5. 5), ak by sme zabudli pred spustením programu zapnúť možnosť výskytu takéhoto zápisu v konfiguračnom súbore. Nevýhodou by však opäť bol možný vznik kolízií.

Problémy, ktoré sa pri práci vyskytli, boli spôsobené hlavne kódovaním vstupným súborov. Súčasná podoba programu, je optimalizovaná pre čitanie vstupných súborov vo formáte UTF-8. Problémy však vznikali hlavne pri snahe pracovať s národnými znakmi (hlavne s úvodzovkami), ktoré program pri vkladaní do bufferu rozdeľoval a nebol schopný rozpoznávať. Bolo preto nutné všetky národné znaky, uložené na viacerých bytoch, vhodne rozpoznať a spojiť do jednej bunky v bufferi. V takomto prípade je jazyk Lua schopný znaky rozpoznať, a teda aj vhodne aplikovať stanovené pravidlá.

Okrem vecí, ktoré by sa dali ešte ďalej zlepšovať však treba povedať, že niekoľko vecí sa naopak podarilo spraviť nad rámec stanovených cieľov. Za zmienku určite stojí mód optimalizovaný pre vstupný súbor vo formáte .txt. Tento mód je možné jednoducho zapnúť tým, že vypneme v konfiguračnom súbore položku context_mode. V takomto pripade, sa celý vstupný súbor považuje za prostý text a program v ňom nehľadá základné riadiace znaky formátu ConTEXt.

13 Záver

V úvode som spomínal pohľad počítača, ktorý má hneď vidieť to, čo človek prehliadne. Práve takýto pohľad v podobe konečného automatu sa podarilo vytvoriť v rámci tejto práce. Okrem toho sa však podarilo vytvoriť nástroj schopný vhodne analyzovať značkovaný text vo formáte nástroja ConTEXt. Tento nástroj je schopný si najprv zo vstupného súboru vyčleniť text, ktorý je vhodný na podrobenie typografickej detekcie chýb a následne v ňom hľadá nezrovnalosti v porovnaní s pravidlami pre sadzbu textu v elektronickej podobe.

Samotná detekcia by však pri súčasnom programovom vybavení nebola ničím zaujímavá. Preto program poskytuje širokú škálu možností, pomocou ktorých sa dá detekcia prispôsobiť na mieru každému súboru. Takisto sa dá do behu programu zasahovať aj prostredníctvom vstupného súboru. Do neho môžeme na ľubovoľné miesto vkladať značky nastavujúce formátovanie. Môžeme tak sami jednoducho zabrániť zbytočnej detekcii v nevhodných častiach súboru. Užívateľ takisto môže jednoducho spravovať, prispôsobovať, prípadne úplne zakázať alebo prepísať návrhy opráv, ktoré potom program použije ako novú opravu.

Pred testovaním reálnych súborov som bol spočiatku skeptický, pretože som čakal, že podobné práce budú obsahovať oveľa menej chýb. Chýb však bolo pomerne dosť a boli rôznorodé. Toto bol pre mňa podnetom pre test na mojich vlastných prácach. Zo zvedavosti som preto otestoval aj svoju bakalársku prácu. Takisto som našiel viacero rôznych chýb. Tento program som preto použil na detekciu chýb aj v texte tejto práce, kde sa mi ich takisto podarilo niekoľko opraviť (13).

Táto práca bola pre mňa výzvou, pretože som nepoznal prostredie ConTEXt, ani programovací jazyk Lua. Teraz však môžem povedať, že som túto výzvu splnil, no bolo k tomu potreba znalosti z najrôznejších oblastí. Patria sem určite typografia, o ktorej som sa dozvedel veľmi veľa, ďalej sadzba v nástroji ConTEXt (ale aj iných TEX-ových nástrojoch), oblasť teoretickej informatiky, ktorá bola nevyhnutná pre správne aplikovanie do praxe. Hlavne pri návrhu, ale aj realizácii gramatík a automatov, ktoré sú základom tejto práce.

Úplne na záver môžem konštatovať, že cieľ stanovený na začiatku sa mi podarilo naplniť. Nástroj na detekciu a korekciu typografických javov v značkovanom texte prostredia ConTEXt-u bol úspešne navrhnutý a naimplementovaný do podoby, v ktorej je schopný chyby nájsť, navrhnúť vhodnú možnosť ich opravy a následne aj opraviť. Môže tak výrazne zefektívniť kontrolu správnosti textu v prostredí ConTEXt-u ale aj pre obyčajné neznačkované texty.

Literatúra

Beneš, Miroslav; Kolář, Dušan. *Syntaktická analýza* [on-line]. 2005. [cit. -05-16]. Dostupné na: http://www.cs.vsb.cz/behalek/vyuka/udp/prednasky/05-parser-4.pdf.

Češka, Milan; Rábová, Zdena. *Gramatiky a jazyky.* 3. vyd. Brno : Vysoké učení technické, 1988. 133 s.

Dusíková, Hana. *DESÁTOMAT* [on-line]. 2011. [cit. 2015-05-16]. Dostupné na: http://www.desatomat.cz/?lang=desatomat.

Dvořáková, Jana. *Kompilátory: Lexikálna analýza* [on-line]. 2007. [cit. 2015-05-16]. Dostupné na: http://foja.dcs.fmph.uniba.sk/kompilatory/docs/prednasky/komp02-lexan.pdf.

Fiala, Jakub. *Korektor* [on-line] [software]. 2013a. [cit. 2015-05-16].*. Dostupné na: http://www.liteera.cz/korektor/.

Fiala, Jakub. *Korektor - Návod k použití* [on-line]. 2013b. [cit. 2015-05-16]. Dostupné na: http://www.liteera.cz/korektor/navod.

Hagen, Hans; Hoekwater, Taco. *ConTeXt reference manual* [on-line]. 2013. [cit. 2015-05-16]. Dostupné na: http://pmrb.free.fr/contextref.pdf.

Hála, Tomáš; Urbanová, Gita. *Softwarová podpora korektur interpunkce* [on-line]. 2012. [cit. 2015-05-16]. Dostupné na: http://bulletin.cstug.cz/pdf/-2.pdf.

Chytil, Michal. *Automaty a gramatiky.* 1. vyd. Praha : Státní nakladatelství technické literatury , 1984. 331 s. ISBN 02-012-84.

Kocur, Pavel. *Úvod do teorie konečných automatů a formálních jazyků.* 1. vyd. Plzeň : Západočeská univerzita, 2001. 104 s. ISBN 80-7082-813-7.

Laakso, Robin. *Just what is TeX?* [on-line]. 2010. [cit. 2015-05-16]. Dostupné na: https://tug.org/whatis.html.

Libre Office. *Libre Office* [software]. 2010. 3.3.

Lingea. *Grammaticon* [software]. 2003. 1.0.

Mansfield, Ron. *Word 97.* 1. vyd. Praha : Grada Publishing, 1998. 887 s. ISBN 80-7169-517-3.

Olšák, Petr. *Vlna* [software]. 2010a. 1.5.

Olšák, Petr. *vlna.man* [on-line]. 2010b. [cit. 2015-05-16]. Dostupné na: http://labmaster.mi.infn.it/wwwasdoc.web.cern.ch/wwwasdoc/TL8/texmf/doc/html/manpages/vlna.html.

Rybička, Jiří. *ikor* [software]. 1997. [1.0].

Rybička, Jiří. *Úvod do teorie formálních jazyků.* 1. vyd. Brno : Konvoj, 1999. 39 s. ISBN 80-85615-25-8.

Thorup, Kresten Krab. *lacheck.man* [on-line]. 1998a. [cit. 2015-05-16]. Dostupné na: `https://www.slac.stanford.edu/comp/unix/package/tex/man-pages/lacheck.html`.

Thorup, Kresten Krab. *lacheck* [software]. 1998b. 1.26.

Český normalizační institut. *Úprava písemností zpracovaných textovými editory : česká technická norma ČSN 01 6910.* Praha : ÚNMZ, 2014. 45 s.

Urbanová, Gita. *Implementace korektoru české interpunkce. (Bakalářská práce.).* Brno : Mendelova zemědělská a lesnická univerzita, 2003. 35 s.

Zoznam obrázkov

Zoznam tabuliek

Prílohy:

Všetky prílohy sa nachádzajú na priloženom CD.

Zoznam príloh:
- ctx_checker.lua
- ctx_fixer.lua
- ctx_config.lua
- ctx_typography.lua
- README.txt