MASARYKOVA UNIVERSITA
FAKULTA INFORMATIKY

Morfologický analyzátor češtiny

DIPLOMOVÁ PRÁCE

Radek Sedláček

Brno, 1999

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Poděkování

Za odborné vedení, cenné rady při řešení diplomové práce a poskytnutou literaturu děkuji Mgr. Pavlu Rychlému, za potřebné připomínky patří můj dík Mgr. Pavlu Smržovi, Dr. Rovněž bych chtěl poděkovat PhDr. Kláře Osolsobě, CSc. za přínosné a inspirující konzultace a Mgr. Marku Veberovi za pomoc při začleňování algoritmu pro automatické určování vzorů českých slov do programu `ced`.

Shrnutí

V teoretické části práce je popsána a diskutována metoda efektivního uložení jazykových jednotek – slov, resp. kmenů, pomocí vyhledávací struktury trie. Je podán výklad vztahu mezi trie a deterministickými konečnými automaty. Druhou část práce – praktickou – tvoří dokumentace popisující formát strojového slovníku češtiny, definičního souboru koncovkových množin a vzorů i samotnou implementaci morfologického analyzátoru a nástroje pro převod strojového slovníku do binárního tvaru.

Klíčová slova

morfologický analyzátor, formální morfologie, trie, deterministický konečný automat, strojový slovník češtiny, slovník kmenů

Obsah

Kapitola 1

Úvod

V současné době je téměř každý osobní počítač vybaven softwarem, který umožňuje zpracovávat texty. Mezi nejrozšířenější patří různé *textové editory* nebo *systémy pro počítačovou sazbu*. Tyto programy obvykle uživateli poskytují funkce umožňující řešit běžné problémy, které při psaní textů vznikají. Vedle standardních operací týkajících se vnější úpravy textu, kam patří mazání a vkládání částí textu, grafická úprava a podobně, existuje celá řada problémů spjatých s jazykovou stránkou takto vznikajících textů. I zde je snahou podobné procesy co nejvíce automatizovat. Hovoří se o tzv. *jazykové podpoře* např. pro textové editory, systémy pro počítačovou sazbu, případně pro další programy. Spadají sem programy pro automatické dělení slov na konci řádku, automatické korektory překlepů, tzv. *spell-checkery*, počítačové *thesaury*, což jsou v podstatě synonymické slovníky nabízející v interaktivním režimu uživateli výběr vhodného jazykového výrazu, *grammar-checkery*, tedy programy pro automatickou gramatickou korekturu textu, *style-checkery* pro stylistické úpravy atd. Programy tohoto druhu mohou být, a ve většině případů také bývají, alespoň z části navrženy na základě automatické morfologické analýzy a lemmatizace.

Další oblastí, kde lze s výhodou využít automatického zpracování gramatických informací, je *korpusová lingvistika*. *Korpus* – reprezentativní soubor jazykových dat – jakožto základ zkoumání jazykových zákonitostí a jejich popisu, se s rozvojem počítačové techniky dostává do popředí zájmu lingvistů. K základnímu softwarovému vybavení pro lexikologicko-lexikografické využití korpusu patří *morfosyntaktický analyzátor*, *lemmatizátor* a tzv. *tagger* [Čerm–95].

Automatický morfologický analyzátor je základním nástrojem pro *značkování* korpusu, tj. přiřazení příslušné gramatické značky (informace o slovním druhu a gramatickém významu) jednotlivým slovům uloženým v korpusu. Je nanejvýš žádoucí, aby automatizace tohoto procesu byla co možná nejvyšší a současně vykazovala malé procento chybovosti. Cílem předkládané práce je takový morfologický analyzátor češtiny navrhnout a implementovat.

V kapitole 2 uvádíme metodologická východiska a nejdůležitější pojmy morfologické paradigmatiky, s jejichž využitím předkládáme popis segmentace českých slov navržené v disertační práci PhDr. Kláry Osolsobě [Osol–96]. Z této segmentace vycházíme při návrhu vlastního algoritmu morfologické analýzy, jehož podrobný rozbor tvoří závěr kapitoly.

Hlavním faktorem ovlivňujícím efektivitu algoritmu morfologické analýzy se ukázalo vyhledávání kmenových základů ve slovníku. Volba pro tento účel nejvhodnější vyhledávací struktury, a sice struktury trie, výčet jejích vlastností a diskuse různých způsobů jejího uložení v paměti počítače jsou náplní kapitoly 3.

Nevýhodou trie je značná prostorová složitost. Při snaze o její snížení jsme využili analogie mezi strukturou trie a deterministickým konečným automatem. Algoritmus minimalizace počtu stavů konečného automatu jsme poté aplikovali na samotnou strukturu trie. Kapitola 4 obsahuje všechen potřebný formální aparát opravňující provedení korektní minimalizace konečného automatu. Čtenář zasvěcený do problematiky může proto s klidným svědomím tuto kapitolu přeskočit. Jeho pozornosti bychom snad pouze doporučili část 4.7 věnovanou formálnímu popisu vztahu mezi deterministickým konečným automatem a trie.

Teoretické výsledky předchozích kapitol jsme zúročili při vlastní implementaci morfologického analyzátoru, kterou dokumentujeme v kapitole 5. Jelikož jsme zvolili slovníkový přístup k řešení problematiky morfologické analýzy, uvádíme zde popis formátu definičního souboru koncovkových množin a vzorů, strojového slovníku češtiny, dokumentaci implementace programového nástroje `abin` pro jejich převod do binárního tvaru a samotného morfologického analyzátoru `ajka`. V závěru kapitoly se zmiňujeme o využití těchto datových souborů při řešení problému přiřazení českého slova k příslušnému vzoru.

Kapitola 2

Algoritmický popis české formální morfologie

V této kapitole objasníme metodologická východiska práce a zavedeme základní pojmy morfologické paradigmatiky nezbytné pro výklad segmentace českých slov. Na závěr popíšeme algoritmus morfologické analýzy, který se o takovou segmentaci slov opírá.

2.1 Metodologická východiska

Počítačová lingvistika je oborem jazykovědy, který se zabývá strojovým zpracováním přirozeného jazyka. Primárním cílem počítačové lingvistiky je automatizace procesu porozumění přirozenému jazyku, a to jak v mluvené, tak i v psané formě. Výzkumem této oblasti se zabývají různé vědní obory, počítačovou lingvistikou počínaje, informatikou a psychologií konče. Ačkoliv každý z těchto vědních oborů volí jiné postupy a klade si jiné cíle, základ jejich snažení je společný – zcela explicitní teorie zmíněných procesů. Taková teorie dosud nebyla ve své obecnosti vybudována, nicméně již nyní existuje řada fungujících systémů založených na alespoň částečném porozumění některému z těchto procesů.

Úkolem počítačové lingvistiky je, mimo jiné, vytvoření explicitního popisu jazyka, tj. přiřazení každé identifikovatelné textové jednotce příslušnou gramatickou informaci. Po stránce metodologické je třeba vyvinout metody pro rozlišení jednotek v textu, určení vztahu mezi jednotkami nižších a vyšších struktur a pro jejich popis. Proto je nejdříve nutné definovat jednotky, s nimiž se pracuje, které jsou vydělitelné z vyšších struktur a jejichž vzájemné vztahy se řídí systémem jistých pravidel.

Metoda segmentace textu na jednotky vychází z přesných definic jednotlivých segmentů a hranic mezi nimi. Přestože běžně vzdělaný rodilý mluvčí je s to bez problémů hláskovat či slabikovat slova, je schopen říci, ze kterých slov se skládá daná věta a kde jsou její hranice, je explicitní popis hlásek, slabik, morfémů, slov, ba i samotných vět složitou záležitostí.

V případě automatické morfologické analýzy češtiny jsme vycházeli z formální definice slova a jeho segmentů, které byly vytvořeny pro potřeby algoritmického popisu flexe přirozeného jazyka v disertační práci Dr. K. Osolsobě [Osol–96].

Stejně tak otázka lemmatizace, tedy přiřazení základního tvaru slova k libovolnému textovému výskytu, je řešena pouze na úrovni slova definovaného ve

zmíněné disertační práci, tj. slova jakožto řetězce znaků ohraničeného mezerami. Proto jsme také problém víceznačnosti v případě lemmatizace vyřešili obdobným způsobem. V mnoha případech totiž není možné na základě libovolného slovního tvaru jednoznačně přiřadit tvar základní. V textu, tedy psaném jazyce, se rozlišují dva typy víceznačnosti:

homografie – z jednoho slovního tvaru lze bez ohledu na kontext vytvořit dva tvary základní. Například tvaru *ubrus* lze jako základní tvar přiřadit nominativ substantiva *ubrus* nebo infinitiv slovesa *ubrousit*;

homonymie – dvě slova nelišící se v žádném z tvarů mají přitom dva různé významy. Kupříkladu slovo *zámek* označující *panské sídlo* nebo *součást dveřního kování*.

Protože rozhodnutí, který základní tvar danému slovnímu tvaru přiřadit, je v případě homografie kontextově závislé, jsou uváděny obě možnosti. Problém homonymie jsme ve své práci takřka neuvažovali, neboť jde v drtivé většině případů o záležitost sémantickou, nikoliv morfologickou. Přesto však některé případy homonymie nezanedbáváme. Tyto případy se ovšem výrazně liší od uvedeného příkladu. Jedná se o některá slova patřící k neohebným slovním tvarům, jako například *se* coby *zvratné zájmeno* a *se* coby *předložka*. Podobně jako u homografů jsou i zde uvedeny možnosti obě.

2.2 Základní pojmy morfologické paradigmatiky

V této části stručně připomeneme základní pojmy používané v klasických popisech formální morfologie. Definice předkládáme v podobě, v jaké je uvádí Mluvnice češtiny [MČII–86].

Základní jazykovou jednotkou v rámci našeho zkoumání je *slovo*. Slovo bývá pro různé roviny jazyka definováno různým způsobem. V [MČII–86] se rozlišují dvě stránky pojmu slovo, a to:

- *slovo* jako reálně vyčlenitelná jednotka jazykového projevu (textu), jako sled, řetězec morfů;
- *slovo* jako potenciální jednotka jazyka (jazykového systému), která je slovem v předchozím smyslu reprezentována.

Z této definice vyplývá též rozlišování slova *textového* a slova *systémového*, nebo také *slovoformy* a *lexému*. Textové slovo je realizací systémového slova. Pro náš účel zaujímají ústřední postavení textová slova, která jsou analyzována prostřednictvím segmentace na jednotky nižšího řádu. Textové slovo je vzhledem k zaměření morfologické analýzy na psaný text definováno jako posloupnost znaků (písmen) ohraničená po obou stranách mezerou. Textová slova jsou realizacemi systémových slov, která jsou definována ve slovníku kmenových základů.

V jazycích typu češtiny má zmíněné rozlišování systémového a textového slova velký význam. Vezmeme-li v úvahu skutečnost, že jednomu jedinému systémovému slovu odpovídá u substantiv 4–12 textových slov, u adjektiv 5–13 textových slov a u sloves až 27 (bereme-li v úvahu pouze jednoduché slovesné tvary) textových slov, je nutné předpokládat, že rodilý mluvčí má ve své paměti uloženy informace o systému české flexe, z nichž při realizaci jednotlivých textových slov při promluvovém aktu vychází. Algoritmus, který je základem programu pro automatickou morfologickou analýzu češtiny, se v oblasti morfologie snaží tento systém simulovat.

Definice 1: *Slovní tvar* je lineární segment promluvy, charakterizovaný celistvostí jak významově funkční, tak i zvukovou a grafickou, se zásadní samostatností, projevující se v jeho přemístitelnosti (omezené ovšem zákonitostmi pořádku slov ve větě).

Uvedená definice vymezuje jednoduché (syntetické) slovní tvary, které stojí v popředí našeho zájmu. Slovní tvary (textová slova) vyjadřují gramatické významy morfologických kategorií příslušných slovních druhů. Textové slovo lze dále segmentovat na jednotky nižšího řádu.

Definice 2: *Tvarotvorný základ* je lexikální složka slovního tvaru, která je všem tvarům ohebného slova společná.

Definice 3: *Tvarotvorný formant* je gramatická složka slovního tvaru, nese gramatické významy, mění se během ohýbání slova.

Definice 4: *Koncovka* je tvarotvorná přípona stojící v absolutním konci slova (pádová, osobní, infinitivní, ...).

Definice 5: *Kmen* je ta část slovního tvaru, která zbývá po odtržení koncovky.

Definice 6: *Morfémy* jsou elementární znaky jazyka. V praxi rozlišujeme různé typy morfémů:

1. *kořeny* – nesamostatné morfémy nesoucí elementární lexikální významy
2. *afixy*, které se dále dělí
   - podle funkce:
     (a) gramatické
     (b) slovotvorné
   - podle postavení vzhledem ke kořeni:
     (a) *prefixy* – morfémy stojící před kořenovým morfémem
     (b) *sufixy* – morfémy připojované za kořenové morfémy
     (c) *postfixy* – slovotvorné morfémy připojované až za gramatický sufix

Gramatický prefix je např. *po-* ve slově *po-jed-u*, slovotvorný prefix je např. *vy-* ve slově *vy-uč-i-t*. Sufixy mohou být finální, to jsou gramatické sufixy, například *-a* ve slově *ryb-a*, nebo nefinální, což jsou slovotvorné sufixy ohebných slov, např. *-tel-* ve slově *uči-tel-é*. Příkladem postfixu je *-koli* ve slově *kter-ý-koli*.

2.3 Segmentace slova pro potřeby strojového popisu

Při návrhu algoritmu pro morfologickou analýzu jsme vycházeli ze segmentace slov, která byla navržena v [Osol–96]. Náplní následujících řádků bude tedy přehledné shrnutí způsobů segmentace tak, jak byly pojaty v citované práci.

V první řadě je třeba odlišit dva základní přístupy k segmentaci textových slov. Podstatným je v tomto smyslu směr, v jakém se textové slovo segmentuje. Rozlišujeme tedy dvě alternativy:

- segmentace od začátku slova
- segmentace od konce slova

Nejdříve se budeme věnovat segmentaci od začátku slova. Rozlišujeme dva typy segmentů vyčleněné z počátku slova:

1. segmenty se snadno formalizovatelným výskytem vázaným gramaticky:
   - negativní prefix *ne-*
   - superlativní prefix *nej-*
   - futurální slovesný prefix *po-*
2. segmenty s nesnadno formalizovatelným výskytem vázáným sémanticky:
   - prefixy
   - první členy kompozit
   - prefixy *ni-*, *ně-* zájmen neurčitých a záporných

Zvláštní skupinu tvoří adjektivní kompozita číslovek a adjektiv označujících počet, množství a věk (*třicetičlenný, pětikilogramový, jedenáctiletý, ...*). Tato adjektiva jsou ve slovníku kmenových základů označena speciálním symbolem a mají automaticky přidělenu množinu všech číslovek v příslušném tvaru jakožto první členy kompozit.

Zatímco analýza prefixů prvního typu je vázána na informace uložené mimo slovník a je součástí popisu formální morfologie, je analýza prefixů druhého typu pouze záležitostí úspornějšího uložení dat ve slovníku kmenových základů.

Všechny kmenové základy, od nichž lze tvořit opozitum prefixem – záporkou *ne-* (zejména se jedná o kmeny slovesné), jsou ve slovníku speciálně označeny. U skupiny pěti sloves první třídy časovaných podle klasického vzoru *brát* a skupiny jednoslabičných sloves páté třídy, která se časují podle klasického vzoru *dělat*,

má libovolný prefix, a tedy i prefix *ne-*, vliv na vokalické alternace kmenotvorné přípony. Tato slovesa mají proto ve slovníku přiřazen speciální tvar hesla.

Problém nastává také u záporných prefigovaných sloves, kde záporka – prefix *ne-* stojí mezi prefixem a slovotvorným základem (*za-ne-db-a-t*), a kde tudíž nejde o pouhý gramatický význam negace, nýbrž o nově utvořené slovo. V tomto případě pracujeme s prefixem *ne-*, jako by se jednalo o prefix druhého typu.

Problém analýzy prefixů druhého typu je vyřešen velmi jednoduše. Každý slovotvorný základ uložený ve slovníku kmenových základů je opatřen seznamem prefixů, s nimiž se pojí. Toto řešení zejména zmenšilo rozsah slovníku. Navíc je takto organizovaný slovník jedinečným materiálovým zdrojem pro lingvistické výzkumy směřující k formálnímu popisu tvoření slov prefixací.

Základem segmentace slova pro potřeby strojového popisu morfologické analýzy bylo navrženo dvoufázové ternární členění slovního tvaru od jeho konce. Segmentace od konce slova probíhá tedy ve dvou etapách:

1. rozdělení slovního tvaru na *kmen* a *koncovku*

2. další segmentace kmene na *kmenový základ* a *intersegment*

Pod pojem *kmen* byly zahrnuty *slovotvorné základy*, které mohou být reprezentovány kupř. kořennými morfémy (*nes-0-u, vod-a*), nebo u slov odvozených a složených různě rozsáhlým komplexem morfémů a konektémů. Jako příklad uveďme slova *(((vy-uč-uj-)íc)-í)*, *(((samo)-(ob(-služ)-n))-ý)*.

Koncovkou se v případě jmen a určitých slovesných tvarů myslí koncovka, jak byla definována výše, tj. tvarotvorná přípona stojící v absolutním konci slova, která je nositelkou gramatických významů příslušných gramatických kategorií pro daný slovní druh. U neurčitých slovesných tvarů je za koncovku považován celý komplex tvořený derivačním sufixem příslušného participia a rodovou koncovkou. Rovněž derivační adverbizační sufix, jímž se paradigmaticky tvoří adverbia od adjektiv, je koncovkou. Speciálním případem je *nulová koncovka*, která zahrnuje jak případy, kdy kmen zůstává beze změny ve srovnání s tvary s nenulovou koncovkou, tak i případy, kdy v souvislosti s výskytem nulové koncovky dochází k alternaci uvnitř kmene, nebo na jeho konci.

Systémy flektivních koncovek v současné češtině se zásadně liší podle slovního druhu ohebných slov. Na jedné straně stojí systémy jmenných koncovek, na druhé straně rozvětvený systém koncovek slovesných. Obecný princip, který byl uplatněn pro všechny koncovkové systémy i podsystémy, spočívá ve vytvoření kritérií pro rozdělení koncovkových množin definovaných jakožto tvaroslovné charakteristiky (klasické vzory) do strukturovaného systému podmnožin. Každé klasické paradigma se tak rozpadne na dvě části:

- jádrová koncovková množina – koncovky, které nemají vliv na podobu kmene, vzhledem k podobě kmene závazné pro většinu koncovek příslušného paradigmatu

- periferní koncovkové množiny – koncovky ovlivňující podobu kmene, které se dále dělí podle různých kritérií daných zejména příslušností slovnědruhovou

Jako *intersegment* byla označena *finální skupina kmene*, která se během flexe mění. Intersegmenty byly dále rozděleny do dvou skupin:

- *intersegmenty 1. řádu* – části kmene izolované od konce, které se mění u substantiv, adjektiv, základních číslovek a zájmen během skloňování, u sloves během časování a tvoření participií a infinitivu. Pokud zůstává kmen během flexe neměnný, považuje se intersegment 1. řádu za *nulový*. Patří sem:
  - *kmenotvorná přípona* slovesných tvarů
  - *kmenová finála*, tj. poslední hláska kmene, u níž docházívá k alternaci
  - *kořenová finála*, tj. poslední písmeno kořenového morfu
  - *finální dvojice* samohlásky *-e-* a libovolného konsonantu *c*, u které dochází k alternaci *e+c* na *0+c*
  - *kmenové zakončení -o-k* nebo *-k-*, které je součástí adjektivního kmene 1. stupně a odpadá při tvoření 2. a 3. stupně
  - *stupňovací sufix* adjektiv a adverbií
- *intersegmenty 2. řádu* – příslušné slovotvorné sufixy stojící mezi intersegmentem 1. řádu a koncovkou derivovaného tvaru. Patří sem:
  - *přípona* pro tvoření posesivních adjektiv
  - *koncovkový derivační komplex*, tzn. původní koncovka pasivního participia, popř. přechodníku, která se stala slovotvorným sufixem pro derivaci různých typů deverbativ
  - *derivační sufix druhových číslovek*
  - *derivační sufix násobných číslovek*
  - *derivační sufix názvů zlomků*
  - *derivační sufix pojmenování číslic*
  - *derivační sufix názvů n-tic*

Cílem algoritmického popisu české formální morfologie bylo vytvořit dynamický popis skloňování jmen a časování sloves. V průběhu práce na přípravě strojového popisu však autoři narazili na některé mezní případy stojící na pomezí mezi formální morfologií a tvořením slov. Do popisu formální morfologie substantiv tudíž zahrnuli popis automatické derivace posesivních adjektiv od životních maskulin a feminin. Popis formální morfologie adjektiv rozšířili o stupňování adjektiv a automatické odvozování adverbií paradigmaticky tvořených od příslušných adjektiv a jejich stupňování. A konečně do slovesné flexe zahrnuli nejen konjugaci

a tvoření participií, ale rovněž tvoření slovesných substantiv a adjektivizovaných trpných participií a přechodníků. Zvláštní formu kombinovaného popisu zvolili u číslovek a zájmen.

Definice jednotlivých vzorů je hlavní součástí algoritmického popisu české formální morfologie a odvozování některých slovních tvarů. Při popisu tvaroslovného systému je základním termínem *paradigma*.

Definice 7: *Morfologické (tvaroslovné) paradigma* je soubor tvarů ohebného slova vyjadřující systém jeho mluvnických kategorií.

U substantiv a adjektiv se tedy jedná o soubor tvarů jednotlivých pádů v jednotném a množném čísle (paradigma singuláru a plurálu). U sloves se rozlišuje několik dílčích paradigmat (prézentní, imperativní, atd.).

Definice 8: *Vzor* je reprezentace tvaroslovného paradigmatu paradigmatem určitého konkrétního slova [MČII–86].

Pod pojmem vzor se rozumí konkrétní slovo reprezentující množinu všech slov, která tvoří ohebné tvary pomocí identického inventáře koncovek a jejichž společným rysem dále je, že tvoří paradigmaticky odvoditelné tvary podle příslušného slovního druhu, jak o tom byla řeč výše, a u kterých dochází ke stejným změnám finální skupiny kmene.

Algoritmický popis zahrnuje na prvním místě definice koncovkových množin. Vzory jsou pak definovány prostřednictvím vzorových slov, která se rozpadají na tyto části:

- neměnná část vzorového slova – *kmenový základ*
- proměnlivé části vzorového slova – *intersegmenty*
- *koncovkové množiny* obsahující utříděné seznamy všech přípustných koncovek vzorového slova spolu s jejich gramatickými významy.

Popis vzoru je vlastně formálním pravidlem přípustné kombinace právě uvedených komponent (segmentů) ohebného slova.

2.4 Algoritmus morfologické analýzy a syntézy

Segmentace slov od jejich konce se stala podstatou algoritmu pro morfologickou analýzu navrženého v práci [Osol–96]. Při pokusu o implementaci tohoto algoritmu se však zdá, že bychom narazili na řadu úskalí. Největší nevýhodou algoritmu je podle našeho názoru jeho neefektivita, která by se při prostém přepsání algoritmu do některého z programovacích jazyků jistě neblahým způsobem projevila.

Postup analýzy založené na slovníku a morfologickém popisu zahrnujícím postup segmentace a identifikace segmentovaných prvků tak, jak byl předveden v disertační práci [Osol–96], je popsán schématem, které znázorňuje obrázek 2.1.

1. *Řetězec nalezen ve slovníku neohebných slov?*
2. **Ano:** Úspěšný konec analýzy.
3. **Ne:** *Řetězec nalezen ve slovníku kmenů?*
4. **Ano:** Úspěšný konec analýzy
5. **Ne:** Odtrhni poslední znak od konce řetězce znaků a hledej jej v popisu českých koncovek.
   Řetězec nalezen v popisu českých koncovek?
6. **Ano:** Hledej zbytek řetězce ve slovníku kmenů.
   Zbytek řetězce nalezen ve slovníku kmenů?
7. **Ano:** Úspěšný konec analýzy
8. **Ne:** Odtrhni další znak od konce řetězce a hledej jej v tabulce intersegmentů, jdi na bod 14.
9. **Ne:** Odtrhni další znak od konce řetězce znaků a hledej jej v popisu koncovek.
   Řetězec nalezen v popisu koncovek?
10. **Ano:** Jdi k bodu 6.
11. **Ne:** Odtrhni další znak od konce řetězce znaků a hledej jej v popisu koncovek.
    Řetězec nalezen v popisu koncovek?
12. **Ano:** Jdi k bodu 6.
13. **Ne:** Vezmi celý řetězec z bodu 3., odtrhni poslední znak od konce a hledej jej v tabulce intersegmentů
14. *Řetězec nalezen v tabulce intersegmentů?*
15. **Ano:** Hledej zbytek řetězce ve slovníku kmenů.
    Zbytek řetězce nalezen ve slovníku kmenů?
16. **Ano:** Úspěšný konec analýzy.
17. **Ne:** Přejdi k bodu 18.
18. **Ne:** Odtrhni další znak od konce řetězce a hledej jej v tabulce intersegmentů.
    Řetězec nalezen v tabulce intersegmentů?
19. **Ano:** Přejdi k bodu 15.
20. **Ne:** Vrať se k bodu 18. (maximálně 4x), pak selhání analýzy

Obrázek 2.1: Schéma algoritmu morfologické analýzy

Již první krok algoritmu, tedy hledání slova ve slovníku neohebných slov, jsme implementovali odlišně. Ve slovníku prakticky slova ohebná a neohebná nerozlišujeme, neboť každé slovo je charakterizováno vzorem. K neohebným slovním tvarům tudíž přistupujeme naprosto stejně jako k ohebným. Rozdíl spočívá pouze v tom, že slova neohebná mají pokaždé přiřazen vzor, který připouští jedinou správnou kombinaci segmentů slova, tedy kmenotvorný základ, nulový intersegment a nulovou koncovku.

Ze zbylé části algoritmu je patrné, že při analýze dochází k postupné segmentaci slova od jeho konce, a to znak po znaku. Při každém odtržení znaku se opakovaně provádí hledání řetězce v příslušné tabulce. Často je hledání neúspěšné, a proto analýza pokračuje buď odtržením dalšího znaku od konce slova a opětovným hledáním v téže tabulce, nebo hledáním téhož řetězce v jiné tabulce. Tato opakovaná hledání neúměrně zvyšují časovou složitost analýzy, což nás dovedlo nakonec k tomu, že jsme k řešení problému morfologické analýzy a posléze též syntézy přistoupili vlastním způsobem. Přitom nezastíráme, že hlavní zdroj inspirace jsme z citované práce čerpali.

Segmentaci slov popsanou v [Osol–96] považujeme za nedotknutelnou a při morfologické analýze z ní též vycházíme. Samotnou analýzu však pojímáme spíše opačným způsobem. Pro náš způsob analýzy je rozhodující identifikace kmenového základu S slova W. Slovo W se v nejobecnějším případě segmentuje na čtyři části; od začátku slova to je po řadě prefix P, kmenotvorný základ S, intersegment I a koncovka T. První, třetí a čtvrtý segment mohou být samozřejmě nulové. Nyní dostáváme jakousi „rovnici“ $W = P + S + I + T$.

Prvním krokem analýzy směřujícím k identifikaci kmenového základu S je tudíž odtržení prefixu. Jedná se samozřejmě o prefix 1. typu (viz str. 7). V této fázi odstraňujeme pouze superlativní prefix *nej-* a záporku *ne-* (v tomto pořadí), pokud to samozřejmě lze. Zbytek slova $W_1 = S + I + T$ nyní obsahuje hledaný kmenový základ na svém počátku. Navíc pro pevně daný kmenový základ S máme ve slovníku uveden seznam vzorů, které určují jediné správné kombinace $I + T$.

Od této chvíle se identifikace kmenového základu S děje znak po znaku. Předpokládejme, že slovo W_1 lze napsat jako posloupnost písmen $a_1a_2 \ldots a_n$, kde je $n \geq 0$. Pokud jde o správně utvořené české slovo a prefix byl odtržen korektně, je hledaným kmenovým základem S některý z řetězců $S_i = a_1a_2 \ldots a_i, 1 \leq i \leq n$. Skutečně, pokud odtržení prefixu bylo korektní, musí kmenový základ obsahovat alespoň jeden znak. Z důvodu možné víceznačnosti je potřeba prověřit všechny tyto kandidáty na kmenový základ.

Ověření, zda kandidát S_i je skutečně kmenovým základem slova W_1, probíhá ve třech krocích. Nejdříve se S_i hledá ve slovníku kmenových základů. Samozřejmě, že v případě, kdy je S_i prázdný řetězec, nemá šanci být nalezen. Pokud se kandidát S_i ve slovníku najde, prověřuje se, zda zbytek $a_{i+1}a_{i+2} \ldots a_n$ slova W_1 je korektní kombinací $I + T$ povolenou v definici některého ze vzorů, k nimž daný kmenový základ S_i patří. Spolu se správnou koncovkou získáme též jí příslušnou gramatickou informaci. Poněvadž na výstupu požadujeme všechny možné hodnoty gramatických kategorií, je nutné tímto způsobem prověřit všechny vzory

a v nich všechny možné kombinace $I + T$. Je ovšem zjevné, že již při neshodě prvního písmene, tedy a_{i+1}, s prvním písmenem intersegmentu nebo koncovky není třeba další písmena $a_{i+2}, \ldots, a_n$ porovnávat. V případě neshody u intersegmentu tak ušetříme porovnání všech koncovek vyskytujících se v koncovkových množinách, které patří danému intersegmentu. K menšímu zpoždění tedy dochází pouze v případě nulových intersegmentů. Nakonec se provede ověření, zda byl prefix P korektně odtržen (pokud odtržen skutečně byl), tj. v případě odtržení záporky *ne-*, zda slovní tvar může tvořit negativní formu, a v případě odtržení superlativního prefixu, zda se jedná o adjektivum, resp. adverbium druhého stupně.

Nedojde-li ke kolizi v žádném z uvedených tří kroků prověřování kandidáta na kmenový základ, je slovo W přijato a analyzováno jako korektní. Navíc je díky definici koncovkových množin k dispozici veškerá gramatická informace o tomto slově.

Efektivita námi navrženého algoritmu morfologické analýzy je ovlivněna zejména rychlostí hledání jednotlivých kandidátů S_i na kmenové základy. Jakmile je jisté, že daný kandidát opravdu může být kmenovým základem (byl nalezen ve slovníku), musí se provést následné ověření konce slova, tj. kombinace $I + T$. Toto ověření, jak jsme již naznačili, musí v každém případě vyčerpat všechny možnosti. Zdá se tedy, že v tomto kroku analýzy nelze příliš zvýšit efektivitu tím, že by se jistá porovnání neprováděla. Jedná se navíc o jednoduchá porovnávání řetězců, která při vhodné implementaci efektivitu nijak nezhoršují.

Proto nezbývá, než úsilí směřující k efektivitě analýzy věnovat vyhledávání kandidátů na kmenové základy v jejich slovníku. Pro uložení kmenových základů jsme zvolili vyhledávací strukturu trie. Hlavním důvodem pro tento výběr byl fakt, že mezi jednotlivými kandidáty existuje určitý vztah – předchozí kandidát S_i je *vždy prefixem* následujícího S_{i+1}. Při hledání S_{i+1} je tedy nanejvýš výhodné využít znalosti, že S_i buď jistě ve slovníku je, nebo tam jistě není.

Použijeme-li vyhledávací strukturu trie, která je založena na principu sdílení společných prefixů hledaných klíčů pomocí opakované indexace, stane se proces vyhledávání velmi efektivním zvláště v situaci, kdy trie implementujeme a vhodným způsobem uložíme jako speciální případ stromu.

Za tohoto předpokladu pak v případě, kdy předchozí kandidát nebyl nalezen, nemá smysl pokračovat v hledání následujícího kandidáta S_{i+1}, neboť ten se ve slovníku s jistotou nevyskytuje. Jestliže S_i ve slovníku je, můžeme využít této znalosti a pokračovat v hledání řetězce $a_{i+1}a_{i+2}\ldots a_n$ z místa, kde jsme skončili hledání předchozího kandidáta S_i. Podrobněji se k této výhodné vlastnosti i dalším přednostem trie vyjádříme v následující kapitole.

Automatická syntéza ohebných, popř. odvozených tvarů, je vlastně obráceným postupem využití algoritmického popisu české morfologie. K libovolnému kmenovému základu uloženému ve slovníku kmenových základů je k dispozici seznam vzorů, pod které příslušný kmenový základ spadá. Aplikací jednotlivých vzorů z tohoto seznamu jakožto pravidel určujících správné zakončení slova kombinací $I + T$ lze vygenerovat množinu všech správných tvarů (textových slov) a jejich potenciálních gramatických významů.

Automatická morfologická analýza a syntéza jsou východiskem automatické lemmatizace. Analyzovanému tvaru je přiřazen nejen možný gramatický význam, ale i vzor. Aplikací vzorů lze vygenerovat všechny tvary spolu s jejich možným gramatickým významem. Základní tvar – lemma – je pak např. tvar nesoucí gramatický význam nominativu singuláru (plurálu) jména a infinitivu slovesa.

Kapitola 3

Vyhledávací struktura trie

Cílem této kapitoly je snaha vysvětlit důvody, které nás přiměly zvolit vyhledávací strukturu trie pro uložení kmenových základů českých slov. Po úvodních definicích stromů a srovnání dvou způsobů uložení stromů v paměti počítače následuje samotná definice struktury trie a shrnutí jejích vlastností na základě porovnání s optimálním binárním vyhledávacím stromem.

3.1 Stromy a lesy

Na úvod této podkapitoly připomeňme, že námi uvedené definice nejsou jediné možné. Konkrétní pojmy lze samozřejmě definovat i jinými, možná formálnějšími způsoby. Příliš formální definice jsou v tomto případě, domníváme se, spíše na závadu. A to z prostého důvodu. Stromy jsou matematickými abstrakcemi jisté struktury dat, v počítačových programech snad jednou z nejrozšířenějších nelineárních struktur. Své jméno dostaly právě proto, že lze vypozorovat jisté analogie mezi nimi a skutečnými stromy v přírodě. Notace používaná v [Knu1–73] a [Knu3–73], odkud jsme některé definice a většinu výsledků přejali, z této analogie rovněž vychází. V žádném případě však neplatí, že by níže uvedené definice byly neobvyklé nebo dokonce nekorektní. Pro názornost je text doplněn obrázky, které použití méně formálních pojmů podpoří a ospravedlní.

Definice 9: *Strom* je konečná neprázdná množina T (její prvky nazýváme *uzly*) taková, že:

(i) obsahuje právě jeden speciálně vyznačený uzel, tzv. *kořen* stromu, $root(T)$

(ii) zbývající uzly (vyjma kořene) jsou rozděleny do $m \geq 1$ disjunktních množin $T_1, T_2, \ldots, T_m$ takových, že každá z těchto množin je opět stromem. Stromy $T_1, T_2, \ldots, T_m$ nazýváme *podstromy* kořene.

Jak je patrné, definice stromu je rekurzívní, tj. definuje stromy pomocí termínu strom. Samozřejmě, že definice je korektní, neboť stromy pouze s jedním uzlem se musí skládat pouze z kořene a stromy s $n > 1$ uzly jsou definovány pomocí stromů s méně než n uzly. Existují též nerekurzívní definice stromů, ale rekurzívní verze

se zdají být vhodnějšími, poněvadž rekurzivita je charakteristickou vlastností stromové struktury.

Z definice dále vyplývá, že každý uzel stromu je kořenem nějakého podstromu obsaženého v rámci celého stromu.

Definice 10: Nechť n je uzlem stromu T. *Stupeň* uzlu n, $deg(n)$, definujeme jako počet podstromů uzlu n. Uzly stupně 0 nazýváme *koncové* uzly nebo častěji *listy*. Nekoncové uzly pojmenujme jako *vnitřní* uzly.

Definice 11: Nechť T je strom a m nejmenší přirozené číslo takové, že pro libovolný uzel n stromu T platí $deg(n) \leq m$. Pak strom T nazýváme *m-ární*.

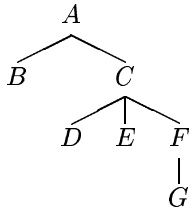
Uvědomme si, že libovolný strom je m-ární pro jisté přirozené číslo m. Toto číslo m je rovno maximálnímu stupni uzlu v daném stromu. Maximum vždy existuje, neboť jsme definovali strom jako konečnou množinu.

Definice 12: Nechť opět n je uzel stromu T. *Úroveň (patro)* uzlu n ve stromě T definujeme induktivně:

(a) úroveň kořene v T je 0

(b) ostatní uzly mají úroveň v T o jedna vyšší než je jejich úroveň v podstromu T_j kořene, který je obsahuje.

Pojetí stromů v souladu s našimi definicemi ilustruje obrázek 3.1, který znázorňuje ternární strom T se sedmi uzly. Kořenem stromu je uzel A, který má dva podstromy $\{B\}$ a $\{C, D, E, F, G\}$. Strom $\{C, D, E, F, G\}$ má kořen C. Uzel C má úroveň 1 v T a má tři podstromy $\{D\}$, $\{E\}$ a $\{F, G\}$, tedy C má stupeň 3. Listy ve stromu T jsou právě uzly B, D, E a G. Uzel F je jediným uzlem stupně 1, G jediným uzlem s úrovní 3. Obrázek 3.1 naznačuje též, jakým způsobem budeme stromy znázorňovat graficky. Kořen bude vždy nejvýše položeným uzlem, listy nejníže položenými uzly.



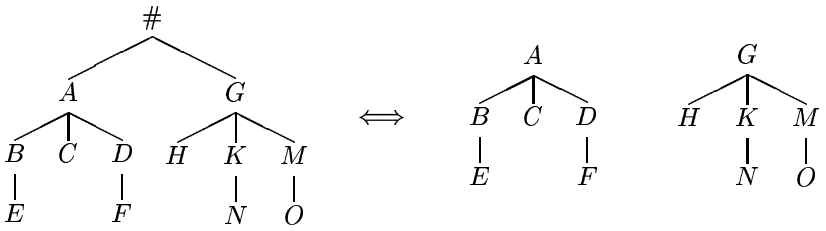
Obrázek 3.1: Příklad stromu

Definice 13: Záleží-li na pořadí podstromů $T_1, T_2, \ldots, T_m$ v části (ii) definice stromu, mluvíme o *uspořádaném* stromu. Pokud $m \geq 2$, má smysl mluvit o T_1 jako o prvním, o T_2 jako o druhém podstromu kořene atd.

V dalším textu, budeme-li mluvit o stromech, máme na mysli stromy uspořádané, pokud explicitně neuvedeme jinak.

Definice 14: *Les* je (obvykle uspořádaná) množina disjunktních stromů.

Mezi lesy a stromy je pouze nepatrný rozdíl. Odstraníme-li kořen stromu, dostáváme les, naopak přidáním jediného uzlu k lesu obdržíme strom. Tyto transformace názorně ukazuje obrázek 3.2.



Obrázek 3.2: Transformace stromu na les a opačně

V následujících definicích zavedeme pojmy, které budeme dále často používat. Jsou převzaty z terminologie rodokmenů, takže jejich chápání je opět v souladu s našimi intuicemi.

Definice 15: O každém kořeni říkáme, že je *otcem* kořenů svých podstromů. Kořeny podstromů, které mají společného otce, jsou navzájem *bratři*, hovoříme o nich jako o *synech* nebo též *následnících* jejich společného otce.

Je zjevné, že kořen celého stromu nemá otce, listy pak nemají syny, nemá tudíž smysl o nich mluvit jako o otcích. V případě uspořádaných stromů, má-li uzel m podstromů, má smysl hovořit o prvním(*nejstarším*), druhém,..., m-tém(posledním, *nejmladším*) synovi. Analogicky budeme o hovořit o *starších(levých)* a *mladších (pravých)* bratrech. Terminologii lze rozšířit i na další pokolení (např. *děd, praděd, bratranec* apod.), ale tyto termíny nebudeme v práci dále potřebovat.

Pro ilustraci opět vezměme obrázek 3.1. Uzel C má tři syny. Nejstarším synem je D, druhým synem v pořadí je E a nejmladší syn je F. F je otcem G. Uzly B a C jsou bratři, přičemž C je mladším bratrem B.

Pro manipulaci se stromovými strukturami existuje mnoho algoritmů. V těchto algoritmech se neustále opakuje proces *procházení stromem*. Jedná se o metodu systematického prozkoumávání uzlů stromu tak, že každý uzel je navštíven právě jednou. Kompletní průchod stromem nám poskytne lineární uspořádání uzlů, takže v mnoha algoritmech lze potom hovořit o „následujícím“ nebo „předchozím“ uzlu některého z uzlů v takovéto posloupnosti.

Pro průchod stromem lze principiálně použít dva základní způsoby. Uzly lze navštěvovat v pořadí odpovídajícím procházení stromu *do hloubky* nebo *do šířky*.

Tyto dvě metody jsou definovány za použití pomocných struktur, které nám slouží k uchovávání uzlů. Jedná se o *zásobník* jako představitele *last-in-first-out* struktury pro procházení do hloubky a *frontu* – zástupce *first-in-first-out* struktury pro procházení do šířky. Procházení stromu T do hloubky je možné zapsat v podobě následujícího algoritmu.

1. Ulož $root(T)$ na vrchol zásobníku.
2. Dokud není zásobník prázdný, opakuj:
   (a) Odeber uzel N z vrcholu zásobníku a projdi jím.
   (b) Jsou-li $T_1, T_2, \ldots, T_m$ podstromy uzlu N, ulož na vrchol zásobníku uzly $root(T_m), root(T_{m-1}), \ldots, root(T_1)$ (v tomto pořadí).

Algoritmus procházení stromu T do šířky je velmi podobný algoritmu pro průchod stromem do hloubky. Rozdíl spočívá pouze ve využití fronty na místě zásobníku.

1. Ulož $root(T)$ na konec fronty.
2. Dokud není fronta prázdná, opakuj:
   (a) Odeber uzel N ze začátku fronty a projdi jím.
   (b) Jsou-li $T_1, T_2, \ldots, T_m$ podstromy uzlu N, ulož na konec fronty uzly $root(T_1), root(T_2), \ldots, root(T_m)$ (v tomto pořadí).

Aplikací uvedených algoritmů v daném pořadí na les, jak jej znázorňuje obrázek 3.2, dostaneme například následující posloupnosti uzlů:

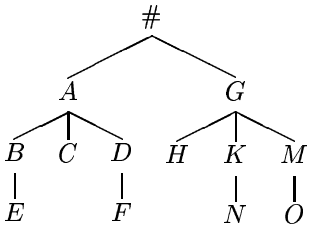
$$A\ B\ E\ C\ D\ F\ G\ H\ K\ N\ M\ O$$
$$A\ G\ B\ C\ D\ H\ K\ M\ E\ F\ N\ O$$

3.2 Možnosti uložení stromů

Existuje mnoho způsobů, jak uložit stromovou strukturu v paměti počítače. Výběr příslušné metody závisí zejména na tom, jaké operace budeme chtít se stromem provádět. Je samozřejmé, že lze dosáhnout velmi úsporného uložení, ale na úkor rychlosti procházení stromem a naopak, uložení stromu preferující rychlost průchodu vyžaduje vyšší paměťové nároky.

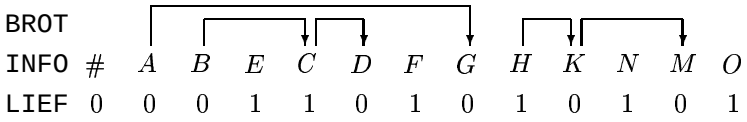
Hlavním problémem při ukládání stromu do paměti počítače je fakt, že strom reprezentuje hierarchickou strukturu, zatímco paměť počítače bývá obvykle charakteru sekvenčního. Proto je nutné jistým systematickým způsobem strom „linearizovat“. V předchozí části jsme uvedli dvě metody, kterými lze získat jednoznačnou posloupnost uzlů stromu reprodukující jeho hierarchickou stavbu. Šlo o procházení stromu do hloubky a do šířky. Postupným zapisováním jednotlivých navštívených uzlů získáváme kýžené posloupnosti. Následující text podrobněji rozebere obě diskutované možnosti.

Nejprve se věnujme uložení stromu odpovídajícímu procházení do hloubky. Podívejme se proto na obrázek 3.3.



Obrázek 3.3: Strom

Strukturu uzlů a jejich odpovídající sekvenční uspořádání ukazuje následující obrázek 3.4.



Obrázek 3.4: Průchod stromem do hloubky

Každý uzel zde má tři položky. `BROT` je ukazatel na nejstaršího z mladších bratrů. Pokud uzel má takového bratra, je ukazatel na něj znázorněn šipkou k němu vedoucí. Neexistuje-li takový bratr, je položka `BROT` prázdná. Druhou položkou je `INFO`, ve které je uložena informace o uzlu, jeho identifikace. Obsah položky `INFO` se liší v závislosti na použití stromu. V případě, že strom reprezentuje uspořádání například této diplomové práce, bude v položce `INFO` vždy název příslušné kapitoly nebo části textu. Reprezentuje-li strom algebraický výraz, může `INFO` obsahovat operace nebo operandy. V našem případě, jak je vidět z obrázku, obsahuje políčko `INFO` kód písmene. Konečně třetí položka, `LIEF`, je jednobitový indikátor určující, zda uzel je listem (`LIEF=1`) nebo vnitřním uzlem (`LIEF=0`).

Tato reprezentace stromu má několik zajímavých vlastností, u kterých bychom se rádi zastavili. Za prvé, všechny podstromy daného uzlu vždy okamžitě tento uzel následují, takže všechny podstromy v příslušném lese následují v sekvenčních blocích. Za druhé, všimněme si, že ukazatele `BROT` se nikdy „nekříží“. To platí obecně pro jakýkoliv strom (les) uložený tímto způsobem, neboť všechny uzly mezi daným uzlem a uzlem určeným ukazatelem `BROT` leží v levém podstromu daného uzlu, takže z této části stromu nemůže žádný ukazatel `BROT` „ukazovat ven“. A konečně lze nahlédnout, že políčko `LIEF` indikující list stromu je redundantní, protože má hodnotu 1 v případě, kdy se jedná o konec lesa, a v případě, kdy uzel bezprostředně předchází každý „dolů směřující“ ukazatel `BROT`.

Uvedená pozorování nás tedy vedou k tomu, že i ukazatel BROT je zbytečný. Vše, co potřebujeme v takovém případě, je (místo ukazatele BROT) opět jednobitový indikátor LAST určující, zda uzel je nejmladším synem (tedy nemá mladších bratrů), či nikoliv. Cíl ukazatele BROT lze pak určit z mnohem menšího objemu dat. Na vysvětlenou shlédněme obrázek 3.5.

LAST	1	0	0	1	0	1	1	1	0	0	1	1	1
INFO	#	*A*	*B*	*E*	*C*	*D*	*F*	*G*	*H*	*K*	*N*	*M*	*O*
LIEF	0	0	0	1	1	0	1	0	1	0	1	0	1

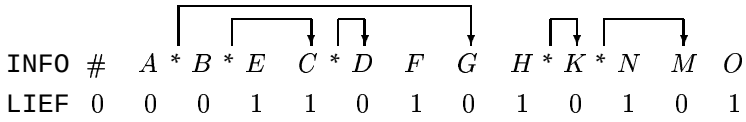
Obrázek 3.5: Průchod stromem do hloubky – úsporná varianta

Popišme, jak lze spočítat hodnotu ukazatele na nejstaršího z mladších bratrů daného uzlu. Při čtení posloupnosti uzlů zleva doprava indikuje hodnota 0 políčka LAST existenci mladšího bratra. Přesněji řečeno indikuje, že uzel není posledním (nejmladším) synem svého otce a tedy má alespoň jednoho mladšího bratra. Ukazatel na tohoto bratra má být v budoucnu doplněn odpovídající hodnotou. Pokaždé, když přejdeme přes uzel s hodnotou 1 políčka LIEF, doplníme poslední dosud nevyplněnou instanci ukazatele BROT. To znamená, že takovéto instance ukazatelů BROT mohou být uloženy na zásobníku. Poslední uzel stromu (lesa) má hodnotu políčka LIEF vždy nastavenu na 1. V tomto případě ovšem po přejití nedoplňujeme instanci ukazatele BROT, neboť zásobník je v tuto chvíli již vyprázdněn.

Pozornému čtenáři jistě neuniklo, že v případě, kdy si nemůžeme dovolit sekvenčně projít celým stromem (lesem), je výše popsaná redukce dat nemožná. Proto je ve většině případů zapotřebí uchovávat všechna relevantní data, jak ukazuje obrázek 3.4. Nicméně pro uložení stromové struktury například na pevném disku je tato redukce často žádoucí, zvláště tehdy, počítáme-li s tím, že strom bude stejně celý načten do paměti. Tehdy, poněvadž čtení z disku je nutně sekvenční, lze při přenosu dat do paměti odpovídající ukazatele dopočítat. I zde ovšem záleží na implementaci a je třeba zvážit, zda ušetřené místo na disku vyrovná ztrátu času nutného pro přídavný výpočet.

I přes pozitivní vlastnosti uvedené výše je ale nasnadě, že prosté uložení do hloubky poněkud plýtvá s pamětí. Při podrobnějším pohledu na obrázek 3.4 zjistíme, že v tomto konkrétním případě je více než polovina ukazatelů BROT nulových. To nás vede k myšlence zmenšení velikosti každého uzlu odstraněním políčka BROT a zavedením speciálních $*$-uzlů bezprostředně za ty uzly, které by jinak měly tento ukazatel nenulový. Ideu ilustruje obrázek 3.6.

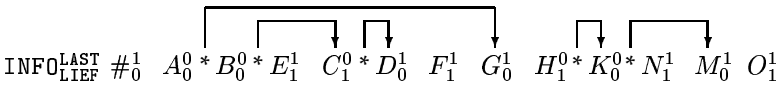
Ve speciálních uzlech označených „$*$“ položka INFO jistým způsobem charakterizuje uzel jako ukazatel směřující na místo označené šipkou. Pokud políčka INFO a BROT mají zhruba stejnou velikost, celkovým efektem této změny je mnohem menší spotřeba paměti, protože počet $*$-uzlů je *vždy* menší než počet ostatních uzlů.



Obrázek 3.6: Odstranění položky BROT

V ideálním případě lze oba indikátory LAST a LIEF uložit v rámci položky INFO. Zvláště tehdy, má-li položka INFO velikost, která se v počítači problematicky ukládá, například zabírá počet bitů, který není dělitelný 8. Potom je často třeba tuto položku zarovnat na velikost násobku celé slabiky (8 bitů). Bity sloužící k doplnění velikosti lze ale využít právě pro indikátory LAST a LIEF. Z tohoto pohledu je proto „ideální", když položka INFO zabírá 6 bitů a zbývající dva bity (do 1 slabiky) vyhradíme pro LAST a LIEF.

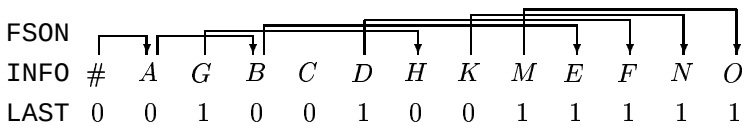
V našem případě položka INFO obsahuje kód českého písmene, těch je 41. Bohatě tedy s 6 bity ($2^6 = 64$ různých kódů) vystačíme. Zbývající 2 bity použijeme tak, jak je popsáno v předchozím odstavci. Dostáváme tak uzel velikosti jedné slabiky (8 bitů). Uložení stromu pak přechází na tvar zachycený na obrázku 3.7.



Obrázek 3.7: Uložení indikátorů LAST a LIEF v rámci položky INFO

Ve skutečnosti i listy mají speciální strukturu, resp. informace v nich uložená je odlišného charakteru než informace uložená v položce INFO vnitřních uzlů. Podrobněji strukturu uzlů včetně listů stromu implementovaného jako součást morfologického analyzátoru popíšeme v kapitole týkající se samotné implementace.

Nyní popíšeme druhý způsob uložení stromů (lesů), tj. uložení odpovídající průchodu stromem do šířky. Například uložení stromu na obrázku 3.3 do šířky ukazuje obrázek 3.8.



Obrázek 3.8: Průchod stromem do šířky

Položka FSON znamená ukazatele na prvního (nejstaršího) syna, zbylé dvě položky INFO a LAST mají tentýž význam jako v případě uložení do hloubky.

Co se týče uložení do šířky, za zmínku stojí jedna velmi významná vlastnost. Při pohledu na obrázek 3.8 zjistíme, že políčka LAST s hodnotou 1 znamenají hranici jedné „rodiny“ (sourozenců). To znamená, že všichni synové následují bezprostředně za sebou, nejstarším počínaje a nejmladším konče (ten má právě LAST=1). Při vyhledávání určitého syna tedy není třeba provádět skoky na často velmi vzdálená místa v paměti určená v případě uložení do hloubky ukazatelem BROT zvláště na nižších úrovních stromu. Při uložení do šířky mladší bratr vždy bezprostředně následuje svého staršího sourozence.

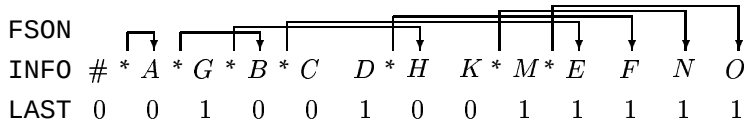
Podobně jako v případě uložení do hloubky, lze i nyní nahlédnout, že cíle, kam směřují ukazatele FSON, lze vypočíst z mnohem méně dat. Úspornější variantu uložení do šířky ukazuje obrázek 3.9.

LIEF	0	0	0	0	1	0	1	0	0	1	1	1	1
INFO	#	A	G	B	C	D	H	K	M	E	F	N	O
LAST	0	0	1	0	0	1	0	0	1	1	1	1	1

Obrázek 3.9: Průchod stromem do šířky – úsporná varianta

Jak nyní spočítat hodnotu ukazatele na prvního syna daného uzlu? Obdobně jako v případě uložení do hloubky. Při průchodu posloupností uzlů zleva doprava indikuje hodnota 0 políčka LIEF existenci syna. Přesněji řečeno indikuje, že uzel není listem, tedy má alespoň jednoho syna. Ukazatel na tohoto syna by měl být v budoucnu doplněn odpovídající hodnotou. Pokaždé, když přejdeme přes uzel s hodnotou 1 políčka LAST, doplníme první dosud nevyplněnou instanci ukazatele FSON. To znamená, že takovéto instance ukazatelů FSON mohou být uloženy ve frontě. Poslední uzel stromu (lesa) má hodnotu políčka LAST vždy nastavenu na 1. V tomto případě ovšem po přejití nedoplňujeme instanci ukazatele FSON, neboť fronta je v tuto chvíli již prázdná.

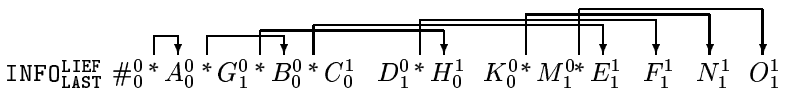
Platné zůstávají též úvahy o efektivnosti úsporného uložení v případě nemožnosti sekvenčního průchodu celým stromem i to, že v daném případě téměř polovina ukazatelů FSON je nulových. Proto lze opět snížit velikost uzlu odstraněním této položky a zavedením speciálních $*$-uzlů. Stále platí, že jich je vždy méně než ostatních uzlů (viz obrázek 3.10).



Obrázek 3.10: Odstranění položky FSON

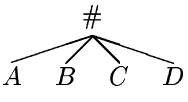
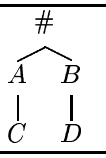
Podobně jako v případě uložení stromu odpovídajícím průchodu do hloubky, lze i nyní využít velikosti políčka INFO. Pokud totiž obsahuje dva volné bity (např.

z důvodu zarovnání na celou slabiku), lze je využít pro uložení indikátorů LIEF a LAST. Tuto situaci postihuje obrázek 3.11.



Obrázek 3.11: Uložení indikátorů LIEF a LAST v položce INFO

Až dosud se jeví uložení do šířky a do hloubky jako zcela „ekvivalentní". Skutečně tomu tak je, zvláště srovnáme-li obrázek 3.5 a obrázek 3.9. První je opravdu „permutací" druhého. Rozdíl je však zřejmý při pohledu na obrázek 3.6 a obrázek 3.10, pokud jde o velikost místa potřebného k uložení stromu. Podstatným je v tomto případě počet ∗-uzlů. Snad ještě markantnější rozdíl ve velikosti stromu při různých způsobech uložení ilustruje tabulka 3.1.

Strom		
Uložení do hloubky	INFO # A * B * C * D LIEF 0 1 1 1 1	INFO # A * C B D LIEF 0 0 1 0 1
Uložení do šířky	INFO # * A B C D LAST 1 0 0 0 1	INFO # * A * B * C D LAST 1 0 1 1 1

Tabulka 3.1: Srovnání metod uložení stromu do hloubky a do šířky

V tabulce jsou znázorněny dva zcela odlišné stromy. Pod nimi je pak uvedena reprezentace při uložení do hloubky a do šířky. Vidíme, že například v případě levého stromu je k uložení do hloubky zapotřebí tří ∗-uzlů, zatímco při uložení do šířky je třeba pouze jeden. Pro druhý strom je tento poměr právě opačný. Lze tedy vyslovit závěr, že k uložení do hloubky jsou vhodné stromy „spíše hlubší než širší", v případě uložení do šířky je tomu naopak.

3.3 Vyhledávací struktura trie

Připomeňme pro osvěžení, co je naším cílem. Je to nalezení struktury pro uložení kmenových základů. Klademe přitom požadavky na co možná nejmenší pamě-

ťovou náročnost a dostatečně rychlé vyhledávání. Repertoár takovýchto struktur je poměrně bohatý. Za všechny jmenujme například binární vyhledávací stromy, B-stromy, B^*-stromy a jiné.

Všechny tyto struktury jsou založeny na vhodném *uspořádání* hledaných klíčů, mezi kterými obvykle neexistuje žádný vztah. Jazyková data, řekněme slova, však mají poněkud odlišný charakter. Jednak uložení celých slov je velmi neefektivní, vzhledem k jejich počtu i velikosti a dále, slova patřící k témuž přirozenému jazyku, například češtině, vykazují jisté pravidelnosti dané morfologicky, gramaticky a foneticky. Bylo by proto nevhodné těchto závislostí nevyužít.

Místo toho, abychom založili vyhledávací metodu na porovnávání mezi klíči, využijeme jejich reprezentace jako posloupnosti písmen. Představme si například klasický knižní rejstřík. Z prvního písmene jistého slova jsme schopni při pohledu do rejstříku okamžitě lokalizovat stránky obsahující všechna slova začínající právě tímto písmenem. Aplikujeme-li tuto metodu opakovaně (na druhé, třetí, ... písmeno), dostaneme vyhledávací strukturu založenou na opakované indexaci, obdobnou té, kterou ilustruje tabulka 3.2.

	␣	a	e	i	o	p	r	s	v	z
0	-	-	-	-	-	pro	-	1	v	za
1	s	-	se	si	-	-	-	-	-	-

Tabulka 3.2: Vyhledávací struktura trie

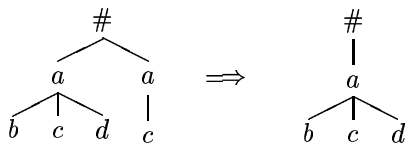
Zmíněná struktura se nazývá *trie*. Poprvé byla uveřejněna E. Fredkinem v roce 1960, který její pojmenování vysvětluje jako část sousloví „information re*trie*val“. Pro pořádek uvedeme formálnější definici.

Definice 16: *Trie* je m-ární strom, jehož uzly jsou m-ární vektory s komponentami odpovídajícími vzájemně různým písmenům. Každý uzel na úrovni l reprezentuje množinu všech klíčů, které začínají jistou posloupností l písmen danou odpovídající větví z kořene do daného uzlu. Uzel tak určuje m-ární větvení závisející na $(l + 1)$-ním písmeni.

V definici je důležitá podmínka, že komponenty vektorů v uzlech reprezentují vzájemně různá písmena. Jinými slovy, klíče uložené v trie sdílejí společný prefix, tj. žádný společný prefix není uložen více jak jednou. Kompresi společného prefixu se snaží ukázat obrázek 3.12.

Struktura trie, kterou ukazuje tabulka 3.2, má pouze dva uzly. Uzel `0` je kořenem a zde také hledáme první písmeno slova. Pokud je jím, řekněme, p, tabulka nám říká, že naše slovo buď musí být `pro`, nebo v tabulce není. Na druhé straně, pokud prvním písmenem bude s, v uzlu `0` najdeme pod `s` odkaz na uzel `1`, přemístíme se proto do uzlu `1`. Uzel `1` říká, že druhým písmenem slova musí být buď znak ␣ (označení konce slova), `e`, nebo `i`.

Předchozí vyhledávací postup přepíšeme do přesnější podoby algoritmu. Předpokládejme, že je dána tabulka m-árního trie. Následující algoritmus provede hle-



Obrázek 3.12: Komprese společného prefixu

dání daného klíče K. Uzly trie jsou m-ární vektory, jejichž indexy jsou čísla od 0 do $m - 1$. Každou komponentou těchto vektorů je buď klíč, nebo ukazatel (NULL nevyjímaje).

1. Nastav proměnnou P tak, aby ukazovala na kořen trie.
2. Nastav k na další písmeno klíče K (zleva doprava). Pokud byl klíč již přečten celý, nastav k na symbol $\sqcup$. Písmena jsou reprezentována čísly v rozsahu $0 \leq k < m$. Nechť X je políčko číslo k uzlu, na nějž ukazuje P v tabulce trie. Pokud X je ukazatel, jdi na bod 3, jinak jdi na bod 4.
3. Pokud $X \neq$ NULL, nastav P=X a vrať se k bodu 2; jinak algoritmus končí neúspěchem.
4. Je-li $K = X$, algoritmus úspěšně končí, jinak je algoritmus ukončen neúspěchem.

Poznamenejme, že v případě neúspěšného hledání je nalezen nejdelší shodný prefix klíče K. Tato vlastnost je pro náš případ velmi účelná, poněvadž je-li slovo ohebné a má neprázdnou koncovku, popř. intersegment, jedná se vždy o hledání neúspěšné. V trie totiž ukládáme pouze kmenové základy českých slov. Tímto způsobem nalezneme nejdelší prefix, tedy nejdelší řetězec, který je kandidátem na kmenový základ slova. Vzhledem k segmentaci od konce slova tento kandidát bývá většinou tím pravým kmenovým základem daného slova. V mnohem menším počtu případů jím je některý z jeho prefixů. Nicméně tento lze detekovat při hledání klíče K jakožto kmenového základu. Při „cestě“ po trie pomocí ukazatele P stačí testovat obsah políčka tabulky, jehož číslo odpovídá znaku $\sqcup$ (konec slova), vůči nulovému ukazateli (obsahem takového políčka totiž nikdy není nenulový ukazatel, nýbrž vždy je jím buď klíč, nebo NULL). V případě, že obsah políčka je nenulový (rozumí se různý od NULL), je dosud přečtená část klíče K jedním z možných kmenových základů.

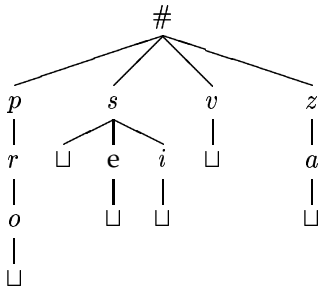
Profesor D. Knuth v [Knu3–73] provedl porovnání efektivity vyhledávání pomocí struktury trie a optimálního binárního vyhledávacího stromu. Experiment byl proveden na množině 31 nejfrekventovanějších anglických slov. Závěr lze shrnout do následujících tří bodů.

- Trie zabírá mnohem více paměti. K reprezentaci 31 různých klíčů (anglických slov) je třeba 360 slov, zatímco optimální binární vyhledávací strom vystačí

s 62 slovy. Nicméně lze jistými technikami dosáhnout toho, že trie nebude větší než 55 slov. Samozřejmě to znamená zvýšení času hledání.

- Úspěšné vyhledání trvá téměř shodně v obou případech. Ovšem neúspěšné hledání proběhne v trie mnohem rychleji, než v optimálním binárním vyhledávacím stromu. Pro naše data, resp. data tohoto druhu je zřejmé, že hledání bude neúspěšné v drtivé většině případů. Z hlediska času vyhledání tedy nezbývá, než preferovat trie.
- V některých případech ztrácí trie svoji výhodu. Uvažme například slova *program* a *programátor*. Trie vyžaduje 8 iterací na jejich rozlišení, proto by zde bylo lepší vybudovat trie tak, jako by slova byla čtena pozpátku.

Podíváme-li se pozorněji na strukturu trie (viz tabulka 3.2), zjistíme, že většina ukazatelů v uzlech je `NULL`. Proto se zdá, že je možno ušetřit nemalý objem paměti, budeme-li ukládat jen skutečně „významné“ ukazatele, nikoliv ty, které mají hodnotu `NULL`. Pro nenulové ukazatele lze pak použít například zřetězené seznamy v každém uzlu. Jinými slovy, trie v podobě tabulky přejde do podoby stromu. Z tohoto pohledu obrázek 3.13 zachycuje ekvivalentní strukturu trie jako tabulka 3.2.



Obrázek 3.13: Trie v podobě stromu

Vyhledávání v takovémto stromu začíná v kořeni. V každém uzlu hledáme jeho syna, který odpovídá dalšímu písmeni hledaného slova. Jednoduše toto hledání probíhá od nejstaršího po nejmladšího syna. Pokud existuje takový syn – najdeme shodu písmene v něm uloženého s patřičným písmenem hledaného slova, hledání syna zastavíme a pokračujeme přečtením dalšího písmene slova a hledáním shody stejným způsobem. V opačném případě končí hledání neúspěšně v nejmladším synovi. Opět konec slova označujeme přidáním speciálního symbolu ␣ na konec slova.

Hledáme-li kupříkladu slovo *si*, přečteme první znak *s* a hledáme syna kořene s písmenem *s*. Nejstarším synem kořene je uzel *p*, zde shoda nenastane, pokračujeme tedy jeho mladším bratrem, uzlem *s*. Nyní ke shodě dojde, přečteme proto další písmeno *i* hledaného slova. Následuje hledání odpovídajícího syna uzlu *s*.

Prvním jeho synem je uzel ⊔, shoda nenastává, pokračujeme ke druhému synovi e, ke shodě opět nedochází, postoupíme tedy k nejmladšímu synovi, uzlu i. Zde ke shodě dojde, proto přečteme další písmeno slova, znak jeho konce ⊔. Tentokráte je shody dosaženo v jediném synovi uzlu i označeném ⊔. Hledání slova proto končí jeho úspěšným nalezením. Hledali bychom například slovo *to*, prošli bychom postupně všechny syny kořene. Ani v jednom z nich by nenastala shoda, došli bychom tedy až k nejmladšímu synovi, uzlu z. Zde také ke shodě nedojde, a proto hledání slova končí neúspěšně.

Uložení trie v podobě stromu se od tabulkové reprezentace v několika rysech liší. Pro srovnání použijme tabulku 3.2 a obrázek 3.13. V tabulce, jakmile je jisté, že jde o jistý klíč, např. je-li první písmeno *p*, je v uzlu přímo tento klíč uložen a nemusíme tedy dále procházet další uzly. Ve stromu ovšem při prvním písmeni *p* přejdeme v tomto případě na nejstaršího syna kořene. Zde ovšem zjišťujeme, že slovo musí pokračovat jediným správným písmenem, a to *r*, podobně též v uzlu pro písmeno *r* nelze jinak, než pokračovat písmenem *o* a teprve až v uzlu odpovídajícím písmeni *o* je nutným následujícím symbolem znak ⊔, tedy konec slova.

Z uvedeného vyplývá, že uzlům v tabulce, v nichž je jednoznačně určen klíč, odpovídají právě ty uzly ve stromu, z nichž vede cesta (posloupnost synů) až k listu, na níž nelze „odbočit", tj. žádný uzel v této posloupnosti nemá více jak jednoho syna. Ve skutečnosti má každý uzel této cesty syna právě jednoho. Nevýhodou tabulkového zápisu je, že políčko tabulky musí být alespoň tak velké jako nejdelší z klíčů, proto u krátkých klíčů dochází ke značnému plýtvání s pamětí. Stromová reprezentace naopak umožňuje efektivnější uložení klíčů s různou délkou.

Zvýšení časové složitosti u stromového reprezentace trie je diskutabilní. Je sice pravdou, že je třeba testovat navíc uzly na cestě, vzpomeneme-li si však na čtvrtý krok algoritmu pro vyhledávání v tabulce pro trie (viz strana 25), zjistíme, že i zde je zapotřebí testovat rovnost mezi klíčem uloženým v tabulce a hledaným slovem. Pokud je tento test proveden písmeno po písmeni, je tabulková reprezentace méně výhodná, neboť se testují i počáteční, v tuto chvíli již zcela jistě správná, písmena slova. V konkrétním případě například začíná-li slovo písmenem *p*, je třeba v uzlu 1 otestovat, zda slovo není rovno klíči `pro`, tedy opět (zbytečně) testovat první písmeno vůči *p*, dále druhé, je-li *r*, a konečně poslední, zda se nejedná o písmeno *o*. Ve stromové reprezentaci první písmeno není třeba testovat, zbylá dvě se otestují stejným způsobem jako při použití tabulky a navíc se provede test na konec slova. Takže z hlediska počtu porovnání v tomto konkrétním případě jsou obě metody ekvivalentní. Pokud bychom tedy v tabulce pro trie uchovávali místo celých klíčů pouze jejich dosud neotestované části, například pro písmeno *p* bychom v uzlu 0 uložili pouze `ro`, nikoliv `pro`, byla by metoda uložení trie do stromu horší jen o počet porovnání konce slova, tj. pro každé *úspěšně* nalezené slovo jedno porovnání navíc.

Preciznější porovnání obou metod lze nalézt v [Knu3–73]. Uvedeme pouze pro nás důležitá zjištění. Při hledání slova ve stromu trie způsobem výše popsaným děláme více méně podobné testy jako při vyhledávání v optimálním binárním vy-

hledávacím stromu. Pouze s tím rozdílem, že nyní provádíme test na rovnost, zatímco v binárním vyhledávacím stromu test na velikost (menší-větší). Elementární teorie tedy říká, že musíme v průměru provést přinejmenším $\log_2 n$ porovnání, abychom rozlišili mezi n klíči. Proto průměrný počet porovnání při vyhledávání ve stromu reprezentujícím trie musí být přinejmenším stejně tak velký jako počet porovnání provedených při vyhledávání v binárním vyhledávacím stromu.

Na druhé straně, trie v podobě tabulky umožňuje m-ární větvení najednou. Vidíme, že průměrný čas potřebný k vyhledání je pro velká n pouze $\log_m n$ iterací, pokud jsou vstupní data náhodná. Dále lze nahlédnout, že prostá tabulka pro trie (viz např. tabulka 3.2) obsahuje přibližně $n/\ln m$ uzlů, má-li rozlišit mezi n náhodnými klíči. Tudíž celkové množství potřebné paměti je úměrné $mn/\ln m$.

Závěrem zopakujme důvody, které nás vedly ke zvolení vyhledávací struktury trie pro uložení kmenových základů. Je to komprese společných prefixů (mezi něž samozřejmě patří i prefixy v morfologickém slova smyslu), velmi rychlé neúspěšné hledání spjaté s nalezením nejdelšího prefixu a schopnost vícenásobného větvení najednou. Zvolili jsme stromovou reprezentaci trie, která ovšem zachovává obě uvedené výhody, zejména schopnost vícenásobného větvení najednou. Z tohoto důvodu jsme se rozhodli pro uložení odpovídající průchodu stromem do šířky. Jediným problémem byla paměťová náročnost trie. Tu jsme se pokusili snížit na minimum. Posloužil nám k tomu postřeh, že trie v podstatě odpovídá deterministickému konečnému automatu. Jeho minimalizaci i zmíněnou vzájemnou korespondenci formálně vyložíme v následující kapitole.

Kapitola 4

Základy teorie automatů

Obsahem této kapitoly je veškerý formální aparát nutný ke korektnímu popisu minimalizace deterministického konečného automatu. Definice, tvrzení a jejich důkazy v jednotlivých částech kapitoly (vyjma 4.4 a 4.7) byly převzaty z [Koze–97]. Na závěr kapitoly je popsána korespondence mezi deterministickým konečným automatem a vyhledávací strukturou trie.

4.1 Řetězce, množiny a operace nad nimi

V této části zavedeme základní definice týkající se řetězců a jejich vztahu k množinám, které budou potřebné při výkladu v dalších částech textu. V rámci celé kapitoly považujeme za přirozená čísla množinu $\{0, 1, 2, \ldots\}$.

Definice 17: *Abeceda* je konečná množina, značíme ji Σ, její prvky nazýváme *písmena* nebo též *symboly*.

Například mluvíme-li o desítkových číslech, používáme abecedu $\{0, 1, \ldots, 9\}$, pokud mluvíme o anglickém textu, máme např. na mysli množinu všech ASCII znaků atd. Symboly budeme obvykle značit malými písmeny ze začátku abecedy, tj. $a, b, c, \ldots$ Neklademe na ně žádná omezení kromě toho, že jich musí být konečně mnoho.

Definice 18: *Řetězec* nad abecedou Σ je konečná posloupnost symbolů.

Je-li $\Sigma = \{a, b\}$, pak $aabba$ je řetězec nad Σ. Pro označení řetězců budeme používat malá písmena z konce abecedy, tedy $x, y, z, \ldots$

Definice 19: *Délka* řetězce x je počet symbolů, které obsahuje. Délku řetězce x označujeme $|x|$.

Definice 20: *Prázdný řetězec* je řetězec délky 0, označuje se ϵ.

Definice 21: Nechť $n \geq 0$ je přirozené číslo. Řetězec a^n složený z právě n symbolů a induktivně definujeme takto:

$$a^0 \stackrel{def}{=} \epsilon,$$
$$a^{n+1} \stackrel{def}{=} a^n a.$$

Definice 22: Množinu všech řetězců nad abecedou Σ značíme Σ^*. Pro úplnost definujeme $\emptyset^* \stackrel{def}{=} \{\epsilon\}$.

Například pro $\Sigma = \{a, b\}$ je $\Sigma^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$. Pokud Σ je neprázdná, pak Σ^* je nekonečná množina řetězců konečné délky. Ve skutečnosti je možné uvažovat i o řetězcích nekonečné délky, ty ale pro náš účel nemají smysl.

Mezi množinami a řetězci je několik podstatných rozdílů, uveďme alespoň dva z nich:

- $\{a, b\} = \{b, a\}$, ale $ab \neq ba$
- $\{a, a, b\} = \{a, b\}$, ale $aab \neq ab$

Navíc je třeba rozlišovat mezi $\emptyset$, $\{\epsilon\}$ a ϵ. V prvním případě jde o prázdnou množinu, ve druhém o množinu s právě jedním prvkem – prázdným řetězcem, a ve třetím o řetězec, nikoliv o množinu.

Definice 23: *Konkatenace* je operace, která ze dvou řetězců x a y v tomto pořadí vytvoří jediný řetězec xy připojením řetězce y na konec řetězce x.

Konkatenace má některé užitečné vlastnosti:

- *asociativita*: $(xy)z = x(yz)$
- prázdný řetězec ϵ je *nulovým prvkem* pro konkatenaci: $\epsilon x = x\epsilon = x$
- $|xy| = |x| + |y|$

Definice 24: Nechť $n \geq 0$ je přirozené číslo. Řetězec x^n vzniklý konkatenací právě n řetězců x induktivně definujeme takto:

$$x^0 \stackrel{def}{=} \epsilon,$$
$$x^{n+1} \stackrel{def}{=} x^n x.$$

Definice 25: *Prefix* řetězce x je takový řetězec y, pro nějž existuje řetězec z tak, že platí $x = yz$.

Například řetězec aba je prefixem řetězce $ababba$. Nulový řetězec je prefixem kteréhokoliv řetězce, jakýkoliv řetězec je sám sobě prefixem.

Nyní se budeme zabývat množinami. Množiny řetězců budeme značit velkými písmeny ze začátku abecedy, tedy A, B, C, ... *Mohutnost* (počet prvků) množiny A budeme značit $|A|$. Prázdná množina $\emptyset$ je jedinou s mohutností 0. Dále uvedeme některé z operací na množinách, které budeme v dalším textu používat. Popisu vlastností těchto operací se však věnovat nebudeme, pro vyjasnění vztahu mezi řetězci a množinami nám postačí jejich definice.

Definice 26: *Sjednocení*: $A \cup B \stackrel{def}{=} \{x \mid x \in A \vee x \in B\}$.

Definice 27: *Komplement v* Σ^*: $\bar{A} \stackrel{def}{=} \{x \in \Sigma^* \mid x \notin A\}$.

Definice 28: *Konkatenace množin*: $AB \stackrel{def}{=} \{xy \mid x \in A \wedge y \in B\}$.

Jinými slovy $z \in AB$ právě tehdy, když lze z napsat jako konkatenaci dvou řetězců x a y, kde $x \in A$ a $y \in B$. Například $\{a, ab\}\{b, ba\} = \{ab, aba, abb, abba\}$. Při vytváření množinové konkatenace je třeba zahrnout *všechny* řetězce, které lze takto obdržet. Je třeba si uvědomit, že AB a BA jsou obecně různé množiny, např. $\{b, ba\}\{a, ab\} = \{ba, bab, baa, baab\}$.

Definice 29: Nechť $n \geq 0$ je přirozené číslo. *Mocninu* A^n *množiny* A definujeme induktivně:

$$A^0 \stackrel{def}{=} \{\epsilon\},$$
$$A^{n+1} \stackrel{def}{=} AA^n.$$

Definice 30: *Asterace* A^* množiny A je sjednocení všech konečných mocnin A:

$$A^* \stackrel{def}{=} A^0 \cup A^1 \cup A^2 \cup \ldots = \{x_1 x_2 \ldots x_n \mid n \geq 0 \wedge x_i \in A, 1 \leq i \leq n\}.$$

Uvědomme si, že n může být 0, proto je $\epsilon \in A^*$ pro jakoukoliv množinu A. Výše jsme definovali Σ^* jako množinu všech řetězců nad Σ konečné délky. To je právě asterace množiny Σ, takže námi zavedená notace je konzistentní.

4.2 Deterministické konečné automaty a regulární množiny

Deterministické konečné automaty jsou matematickým modelem tzv. *přechodových systémů*. Intuitivně si lze pod *stavem* systému představit celkový obraz systému v daném časovém okamžiku. Stav tedy poskytuje všechnu relevantní informaci o tom, jak se bude systém nadále vyvíjet. *Přechody* jsou pak změny stavů, které se mohou odehrávat buď spontánně, nebo jako odpovědi na vnější podnět. V reálném životě lze nalézt řadu příkladů takovýchto systémů, za všechny jmenujme alespoň elektronické obvody, výtahy nebo digitální hodinky. Přechodový systém, který má pouze konečně mnoho stavů, bývá označován jako tzv. *konečně-stavový přechodový systém*. Jeho abstrakcí je *konečný automat*.

Definice 31: *Deterministický konečný automat* (DKA) je uspořádaná pětice $M = (Q, \Sigma, \delta, s, F)$, kde

- Q je konečná množina, její prvky nazýváme *stavy*
- Σ je konečná množina, *vstupní abeceda*
- $\delta : Q \times \Sigma \to Q$ je *přechodová funkce*
- $s \in Q$ je *počáteční stav*
- $F \subseteq Q$, prvky F se označují jako *akceptující* nebo *koncové stavy*

Výpočet deterministického konečného automatu lze neformálně ilustrovat pomocí následující představy. Na začátku výpočtu umístíme pomyslný oblázek do počátečního stavu. Vstupem DKA může být libovolný řetězec $x \in \Sigma^*$. Postupně čteme vstupní řetězec zleva doprava, v jednom časovém okamžiku vždy jeden symbol. Přitom přemisťujeme oblázek v souladu s δ: je-li následujícím symbolem vstupního řetězce písmeno b a oblázek je ve stavu q, přemístíme jej do stavu $\delta(q, b)$. Po dosažení konce vstupního řetězce se oblázek nachází v nějakém stavu, řekněme p. Pokud $p \in F$, říkáme, že řetězec byl *akceptován* strojem M, jinak ($p \notin F$) mluvíme o tom, že řetězec x byl *odmítnut* strojem M. Nyní se pokusíme intuitivní představu definovat formálně.

Definice 32: Definujeme funkci $\hat{\delta} : Q \times \Sigma^* \to Q$ pomocí funkce δ induktivně vzhledem k délce řetězce jakožto jejího druhého argumentu:

$$\hat{\delta}(q, \epsilon) \stackrel{def}{=} q,$$
$$\hat{\delta}(q, xa) \stackrel{def}{=} \delta(\hat{\delta}(q, x), a).$$

Definice 33: Řekneme, že řetězec x je DKA $M = (Q, \Sigma, \delta, s, F)$ *akceptován*, jestliže $\hat{\delta}(s, x) \in F$ a *odmítnut*, jestliže $\hat{\delta}(s, x) \notin F$.

Definice 34: *Množinu* nebo *jazyk akceptovaný deterministickým konečným automatem* $M = (Q, \Sigma, \delta, s, F)$ označujeme $L(M)$ a definujeme:

$$L(M) \stackrel{def}{=} \{x \in \Sigma^* \mid \hat{\delta}(s, x) \in F\}.$$

Definice 35: Podmnožina $A \subseteq \Sigma^*$ se nazývá *regulární*, jestliže $A = L(M)$ pro nějaký deterministický konečný automat M.

4.3 Minimalizace počtu stavů DKA

Definice 36: Nechť $M = (Q, \Sigma, \delta, s, F)$ je DKA. Řekneme, že stav $q \in Q$ je *nedosažitelný*, jestliže neexistuje žádný řetězec $x \in \Sigma^*$ takový, že $\hat{\delta}(s, x) = q$.

Nechť je dán DKA $M = (Q, \Sigma, \delta, s, F)$ akceptující regulární množinu A. Minimalizační (co se počtu stavů týče) proces sestává ze dvou kroků:

1. eliminace nedosažitelných stavů,
2. sjednocení ekvivalentních stavů.

Při minimalizačním procesu jde především o to, aby minimalizovaný konečný automat přijímal právě stejný jazyk, jako jeho neminimalizovaný protějšek. Z definice nedosažitelného stavu plyne, že eliminací těchto stavů se množina akceptovaná automatem nezmění. Tyto stavy lze eliminovat prostým procházením přechodového grafu automatu do hloubky. Pro náš případ ani nemá smysl uvažovat

o nedosažitelných stavech vzhledem ke způsobu, jakým budeme konečný automat konstruovat.

Ve druhém kroku je nejpodstatnější znalost oné „ekvivalence“ na stavech automatu. Ve zbytku této části takovou ekvivalenci formálně definujeme a v části následující předvedeme jeden z možných konkrétních algoritmů pro její výpočet.

Definice 37: Nechť $M = (Q, \Sigma, \delta, s, F)$ je DKA. Definujeme relaci ekvivalence $\approx$ na množině stavů Q takto:

$$p \approx q \overset{def}{\iff} \forall x \in \Sigma^* \ (\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F).$$

Skutečně není těžké nahlédnout, že právě definovaná relace je opravdu ekvivalencí, neboť je definována pomocí ekvivalence logické. Stejně jako všechny ostatní ekvivalence, i $\approx$ rozděluje množinu, na níž je definována, na disjunktní třídy ekvivalence:

$$[p] \overset{def}{=} \{q \in Q \mid q \approx p\}.$$

Každý prvek $q \in Q$ je obsažen v právě jedné třídě ekvivalence $[q]$ a platí:

$$p \approx q \iff [p] = [q].$$

Dále se pokusíme definovat *podílový konečný automat*, jehož stavy korespondují s třídami ekvivalence $\approx$. Pro každou třídu ekvivalence obsahuje právě jeden stav.

Definice 38: Nechť $M = (Q, \Sigma, \delta, s, F)$ je DKA. *Podílový DKA* k DKA M je pětice $M/\approx \overset{def}{=} (Q', \Sigma, \delta', s', F')$, kde

- $Q' \overset{def}{=} \{[q] \mid q \in Q\}$,
- $\delta'([q], a) \overset{def}{=} [\delta(q, a)]$,
- $s' \overset{def}{=} [s]$,
- $F' \overset{def}{=} \{[p] \mid p \in F\}$.

Je třeba ukázat, že definice δ' je korektní. Funkce δ' je na třídě ekvivalence $[q]$ definována pomocí jejího reprezentanta q. Volba jiného reprezentanta by mohla vést k jiné hodnotě funkce δ'. Následující lemma tuto situaci formálně vylučuje.

Lemma 1: *Jestliže $p \approx q$, pak $\delta(p, a) \approx \delta(q, a)$. Ekvivalentně, pokud $[p] = [q]$, je $[\delta(p, a)] = [\delta(q, a)]$.*

Důkaz: Předpokládejme $p \approx q$ a nechť $a \in \Sigma$ a $y \in \Sigma^*$ jsou libovolné. Pak

$$\begin{aligned} \hat{\delta}(\delta(p, a), y) \in F &\iff \hat{\delta}(p, ay) \in F \\ &\iff \hat{\delta}(q, ay) \in F \text{ (protože } p \approx q) \\ &\iff \hat{\delta}(\delta(q, a), y) \in F. \end{aligned}$$

Protože y bylo zvoleno libovolně, je (podle definice $\approx$) $\delta(p, a) \approx \delta(q, a)$.

Lemma 2: $p \in F \iff [p] \in F'$.

Důkaz: Směr $\Rightarrow$ okamžitě plyne z definice množiny koncových stavů F' podílového automatu. K důkazu opačné implikace $\Leftarrow$ stačí ukázat, že je-li $p \approx q$ a $p \in F$, pak $q \in F$. Jinými slovy, každá třída ekvivalence $\approx$ je buď podmnožinou F, nebo je s F disjunktní, což ovšem okamžitě vyplývá z definice ekvivalence $\approx$, vezmeme-li $x = \epsilon$.

Lemma 3: $\forall x \in \Sigma^* \ (\hat{\delta'}([p], x) = [\hat{\delta}(p, x)])$.

Důkaz: Indukcí vhledem k $|x|$. Pro $x = \epsilon$ máme

$$\begin{aligned}\hat{\delta'}([p], \epsilon) &= [p] \text{ (z definice } \hat{\delta'}) \\ &= [\hat{\delta}(p, \epsilon)] \text{ (z definice } \hat{\delta}).\end{aligned}$$

Za účelem indukčního kroku předpokládejme $\hat{\delta'}([p], x) = [\hat{\delta}(p, x)]$. Nechť dále $a \in \Sigma$. Dostáváme

$$\begin{aligned}\hat{\delta'}([p], xa) &= \delta'(\hat{\delta'}([p], x), a) \text{ (z definice } \hat{\delta'}) \\ &= \delta'([\hat{\delta}(p, x)], a) \text{ (podle indukčního předpokladu)} \\ &= [\delta(\hat{\delta}(p, x), a)] \text{ (z definice } \delta') \\ &= [\hat{\delta}(p, xa)] \text{ (z definice } \hat{\delta}).\end{aligned}$$

Po předchozích nezbytných pomocných tvrzení jsme nyní připraveni formulovat a zejména dokázat nejdůležitější výsledek této kapitoly, a sice, že deterministický konečný automat M a k němu zkonstruovaný podílový automat $M/\approx$ jsou ekvivalentní. To znamená, že jazyky, které oba přijímají, jsou totožné. Formálně zapsáno dostáváme následující větu.

Věta 1: $L(M/\approx) = L(M)$.

Důkaz: Pro $x \in \Sigma^*$ platí

$$\begin{aligned}x \in L(M/\approx) &\iff \hat{\delta'}(s', x) \in F' \text{ (z definice akceptování)} \\ &\iff \hat{\delta'}([s], x) \in F' \text{ (z definice } s') \\ &\iff [\hat{\delta}(s, x)] \in F' \text{ (z lemma 3)} \\ &\iff \hat{\delta}(s, x) \in F \text{ (z lemma 2)} \\ &\iff x \in L(M) \text{ (z definice akceptování).}\end{aligned}$$

Přirozeně po konstrukci podílového automatu vyvstává otázka, zda v tomto podílovém automatu nelze opět některé stavy sjednotit. Jednoduše tak, že zopakujeme konstrukci podílového automatu tentokráte aplikovanou na podílový automat. Ukazuje se, že tomu tak není, tedy konstrukci podílového automatu stačí provést jen jednou. Abychom se o tom přesvědčili, proveďme konstrukci podílového automatu podruhé.

Definujme relaci ekvivalence na stavech podílového automatu:

$$[p] \sim [q] \stackrel{def}{\iff} \forall x \in \Sigma^* \ (\hat{\delta}'([p], x) \in F' \iff \hat{\delta}'([q], x) \in F').$$

Toto je stejná definice jako dříve definice relace $\approx$, pouze aplikovaná na podílový automat $M/\approx$. Ekvivalenci na Q' značíme $\sim$, abychom ji odlišili od ekvivalence $\approx$ na Q. Nyní dostáváme následující řetěz implikací:

$$\begin{aligned}
[p] \sim [q] &\implies \forall x \in \Sigma^* \ (\hat{\delta}'([p], x) \in F' \iff \hat{\delta}'([q], x) \in F') \text{ (z definice } \sim) \\
&\implies \forall x \in \Sigma^* \ ([\hat{\delta}(p, x)] \in F' \iff [\hat{\delta}(q, x)]) \in F' \text{ (z lemma 3)} \\
&\implies \forall x \in \Sigma^* \ (\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F) \text{ (z lemma 2)} \\
&\implies p \approx q \text{ (z definice } \approx) \\
&\implies [p] = [q].
\end{aligned}$$

Tedy libovolné dva ekvivalentní stavy podílového automatu $M/\approx$ jsou ve skutečnosti totožné, a proto relace ekvivalence $\sim$ na Q' není ničím jiným, než identickou relací. Opakovat konstrukci podílového automatu podruhé tudíž nemá smysl.

4.4 Minimalizační algoritmus

Zde předvedeme jeden z možných algoritmů pro výpočet relace $\approx$ definované v předchozí části. Tento konkrétní algoritmus uvádíme proto, že jsme si jej vybrali pro implementaci.

Definice 39: Nechť $M = (Q, \Sigma, \delta, s, F)$ je deterministický konečný automat. Definujeme posloupnost relací na množině Q takto:

$$\begin{aligned}
p \stackrel{0}{\sim} q &\stackrel{def}{\iff} p \in F \iff q \in F, \\
p \stackrel{i}{\sim} q &\stackrel{def}{\iff} p \stackrel{i-1}{\sim} q \wedge \forall a \in \Sigma \ (\delta(p, a) \stackrel{i-1}{\sim} \delta(q, a)).
\end{aligned}$$

Rozklad $Q/\stackrel{i}{\sim}$ množiny Q podle $\stackrel{i}{\sim}$ budeme značit R_i.

Lemma 4: *Nechť $M = (Q, \Sigma, \delta, s, F)$ je DKA. Pak platí následující tvrzení.*

(i) *Každá z relací $\stackrel{i}{\sim}$ je ekvivalencí na množině Q.*

(ii) *Pro každé přirozené číslo i relace $\stackrel{i+1}{\sim}$ zjemňuje relaci $\stackrel{i}{\sim}$, tj. $\forall p, q \in Q \ (p \stackrel{i+1}{\sim} q \implies p \stackrel{i}{\sim} q)$.*

(iii) *Pro každé přirozené číslo i platí: je-li $R_i = R_{i+1}$, pak pro každé přirozené číslo $t \geq 1$ je $R_i = R_{i+t}$.*

(iv) *Nechť $n = |Q|$ je počet stavů automatu M. Pak existuje přirozené číslo $k \leq n - 1$ takové, že $R_k = R_{k+1}$.*

(v) *Pro každé přirozené číslo k takové, že $R_k = R_{k+1}$, platí $\stackrel{k}{\sim} = \approx$.*

Důkaz: Tvrzení *(i)* a *(ii)* okamžitě plynou z definice $\overset{i}{\sim}$.

(iii) Víme, že $R_i = R_{i+1}$, tedy $\overset{i}{\sim} = \overset{i+1}{\sim}$. Z definice $\overset{i+1}{\sim}$ plyne, že $p \overset{i+1}{\sim} q \iff p \overset{i}{\sim} q \wedge \forall a \in \Sigma\, (\delta(p,a) \overset{i}{\sim} \delta(q,a))$. Protože $\overset{i}{\sim} = \overset{i+1}{\sim}$, je nyní $p \overset{i+1}{\sim} q$ a současně $\forall a \in \Sigma\, (\delta(p,a) \overset{i+1}{\sim} \delta(q,a))$. To však podle definice znamená, že $p \overset{i+2}{\sim} q$. Je tedy $p \overset{i+1}{\sim} q \iff p \overset{i+2}{\sim} q$, tedy $\overset{i+1}{\sim} = \overset{i+2}{\sim}$. Předchozí poznatek můžeme induktivně aplikovat na jakékoliv $i + t$, kde $t \geq 1$.

(iv) Označme c_i index ekvivalence $\overset{i}{\sim}$. Předpokládejme, že $R_0, R_1, \ldots, R_k$ jsou vzájemně různé a $R_k = R_{k+1}$ je první výskyt rovnosti sousedů v řadě R_i. Protože $\overset{i+1}{\sim}$ zjemňuje $\overset{i}{\sim}$, platí vztah $1 < c_0 < c_1 < \ldots < c_k \leq n$. Proto $k + 1 \leq n$, tedy $k \leq n - 1$.

(v) Definice relace $\overset{i}{\sim}$ se dá interpretovat takto: $p \overset{i}{\sim} q$, jestliže pro každé slovo x o délce nejvýše i platí, že $\hat{\delta}(p,x) \in F \iff \hat{\delta}(q,x) \in F$. Nechť $R_k = R_{k+1}$. Podle bodu *(iii)* je $R_k = R_{k+t}$ pro každé přirozené $t \geq 1$. Vezmeme-li tedy jakoukoliv horní hranici d délky slova, potom $p \overset{i}{\sim} q$ je ekvivalentní s tím, že pro všechna slova $x, |x| \leq d$ platí $\hat{\delta}(p,x) \in F \iff \hat{\delta}(q,x) \in F$. Tato vlastnost je tedy splněna pro všechna slova x, což je ekvivalentní s definicí relace $\approx$. Je tedy $p \overset{k}{\sim} q \iff p \approx q$, tedy $\overset{k}{\sim} = \approx$.

Právě dokázané lemma odpovídá na otázku, jak lze vypočítat relaci $\approx$, respektive jí indukovaný rozklad na stavech automatu. Budeme konstruovat posloupnost relací $\overset{i}{\sim}$ (resp. odpovídající rozklady) tak dlouho, dokud bude následující relace $\overset{i}{\sim}$ různá od předchozí $\overset{i-1}{\sim}$. Výsledkem pak bude relace $\approx$, resp. jí indukovaný rozklad. Formálněji lze tento postup zapsat ve formě algoritmu.

1. Nastav $i = 0$.
2. Nastav $R_0 = \{Q - F, F\}$.
3. Opakuj:
   (a) vypočítej R_{i+1} z R_i podle definice,
   (b) nastav $i = i + 1$,
4. dokud není $R_i = R_{i-1}$.

Korektnost algoritmu je dána platností tvrzení *(v)* předchozího lemmatu, algoritmus skončí díky tvrzení *(iv)* tamtéž. Praktický výpočet předvedeme na konkrétním příkladě. Nechť $M = (\{1, 2, \ldots, 9\}, \{a, b\}, \delta, 1, \{3, 5, 6\})$ je DKA, jehož přechodová funkce je definována následující tabulkou.

δ	a	b
1	2	3
2	2	4
3	3	5
4	2	7
5	6	3
6	6	6
7	7	4
8	2	3
9	9	4

Podle kroku 2 je $R_0 = \{\text{I, II}\}$, kde $\text{I} = \{1, 2, 4, 7, 8, 9\}$ a $\text{II} = \{3, 5, 6\}$. Provedeme konstrukci R_1 následovně:

I	a	b
1	I	II
2	I	I
4	I	I
7	I	I
8	I	II
9	I	I

II	a	b
3	II	II
5	II	II
6	II	II

Máme tedy $R_1 = \{\text{III, IV, V}\}$, kde $\text{III} = \{1, 8\}$, $\text{IV} = \{2, 4, 7, 9\}$ a $\text{V} = \{3, 5, 6\}$. Obdobným způsobem zkonstruujeme R_2:

III	a	b
1	IV	V
8	IV	V

IV	a	b
2	IV	IV
4	IV	IV
7	IV	IV
9	IV	IV

V	a	b
3	V	V
5	V	V
6	V	V

Dostáváme $R_2 = \{\text{VI, VII, VIII}\}$, kde $\text{VI} = \{1, 8\}$, $\text{VII} = \{2, 4, 7, 9\}$ a $\text{VIII} = \{3, 5, 6\}$. Je tedy $R_2 = R_1$, tj. $\overset{2}{\sim}=\overset{1}{\sim}=\approx$ a podle podmínky v kroku 4 algoritmus končí. Příslušný podílový automat je $M/\approx = (\{\text{III, IV, V}\}, \{a, b\}, \delta', \text{III}, \{\text{V}\})$, kde δ' je dána tabulkou:

δ'	a	b
III	IV	V
IV	IV	IV
V	V	V

4.5 Myhill-Nerodovy relace

Cílem této části je ukázat, že pokud M a N jsou DKA bez nedosažitelných stavů akceptující tutéž množinu, pak podílové automaty $M/\approx$ a $N/\approx$ zkonstruované

minimalizačním algoritmem z minulé části jsou isomorfní. Tedy DKA zkonstruovaný tímto algoritmem je minimálním DKA pro množinu, kterou akceptuje a je až na isomorfismus dán jednoznačně.

Poslouží nám k tomu výklad korespondence mezi DKA se vstupní abecedou Σ a jistou ekvivalencí na Σ^*. Ukážeme, že minimální DKA pro regulární množinu R může být přirozeně definován *přímo z* R, a že libovolný minimální DKA pro R je s ním isomorfní.

Definice 40: Dva deterministické konečné automaty $M = (Q_M, \Sigma, \delta_M, s_M, F_M)$, $N = (Q_N, \Sigma, \delta_N, s_N, F_N)$ jsou *isomorfní*, jestliže existuje bijekce $f : Q_M \to Q_N$ taková, že

- $f(s_M) = s_N$
- $\forall p \in Q_M, a \in \Sigma\ (f(\delta_M(p, a)) = \delta_N(f(p), a))$
- $p \in F_M \iff f(p) \in F_N$.

To znamená, že M a N jsou v podstatě stejné DKA až na pojmenování stavů. Je samozřejmé, že isomorfní DKA akceptují tutéž množinu.

Definice 41: Nechť $R \subseteq \Sigma^*$ je regulární množina a nechť $M = (Q, \Sigma, \delta, s, F)$ je DKA. Automat M indukuje relaci ekvivalence $\equiv_M$ na množině Σ^* definovanou takto:

$$x \equiv_M y \overset{def}{\iff} \hat{\delta}(s, x) = \hat{\delta}(s, y).$$

Nezaměňujme relaci $\equiv_M$ s relací $\approx$. Relace $\approx$ byla definována na množině Q, zatímco $\equiv_M$ je definována na množině Σ^*. Opět je snadné se přesvědčit o tom, že jde skutečně o relaci ekvivalence (je definována pomocí rovnosti). Navíc $\equiv_M$ splňuje další užitečné vlastnosti:

(i) Je to *pravá kongruence*, tj. pro libovolné $a \in \Sigma$ a $x, y \in \Sigma^*$ platí

$$x \equiv_M y \implies xa \equiv_M ya.$$

Skutečně tomu tak je. Předpokládejme $x \equiv_M y$. Pak

$$\begin{aligned}\hat{\delta}(s, xa) &= \delta(\hat{\delta}(s, x), a) \\ &= \delta(\hat{\delta}(s, y), a) \text{ (podle předpokladu)} \\ &= \hat{\delta}(s, ya).\end{aligned}$$

(ii) *Zjemňuje* R: pro libovolné $x, y \in \Sigma^*$ platí

$$x \equiv_M y \implies (x \in R \iff y \in R).$$

Poněvadž $\hat{\delta}(s, x) = \hat{\delta}(s, y)$ a to je buď akceptující, nebo zamítající stav, jsou řetězce x a y buď oba akceptovány, nebo oba odmítnuty. Jinými slovy, každá třída ekvivalence $\equiv_M$ má buď všechny prvky v R, nebo je s R disjunktní, tedy R je sjednocením tříd ekvivalence $\equiv_M$.

(iii) Má *konečný index*, tj. má jen konečně mnoho tříd ekvivalence, poněvadž pro každý stav $q \in Q$ stroje M existuje právě jedna třída ekvivalence

$$\{x \in \Sigma^* \mid \hat{\delta}(s, x) = q\}.$$

Definice 42: Libovolná relace ekvivalence $\equiv$ na množině Σ^*, která je pravou kongruencí, má konečný index a zjemňuje $R \subseteq \Sigma^*$ se nazývá *Myhill-Nerodova relace pro* R.

Zajímavý na této definici je fakt, že charakterizuje právě relace na Σ^*, které jsou $\equiv_M$ pro nějaký DKA M. Můžeme tedy zkonstruovat konečný automat M z relace $\equiv_M$ jen díky skutečnosti, že $\equiv_M$ je Myhill-Nerodovou relací. Proto dále ukážeme, jak zkonstruovat automat $M_\equiv$ pro množinu R z dané Myhill-Nerodovy relace $\equiv$ pro R. Navíc uvidíme, že konstrukce $M \mapsto \equiv_M$ a $\equiv \mapsto M_\equiv$ jsou inverzní až na isomorfismus automatů.

Definice 43: Nechť $R \subseteq \Sigma^*$ a nechť $\equiv$ je nějakou Myhill-Nerodovou relací na R. $\equiv$*-třída* řetězce $x \in \Sigma^*$ je

$$[x] \stackrel{def}{=} \{y \mid y \equiv x\}.$$

Ačkoliv existuje nekonečně mnoho řetězců v Σ^*, díky vlastnosti (iii) je počet $\equiv$-tříd konečný.

Definice 44: Definujeme DKA $M_\equiv = (Q, \Sigma, \delta, s, F)$, kde

- $Q \stackrel{def}{=} \{[x] \mid x \in \Sigma^*\}$,
- $s \stackrel{def}{=} [\epsilon]$,
- $F \stackrel{def}{=} \{[x] \mid x \in R\}$,
- $\delta([x], a) \stackrel{def}{=} [xa]$.

Opět je nutné ověřit, že přechodová funkce δ je korektně definována. To je zaručeno vlastností (i) Myhill-Nerodových relací. Vlastnosti pravé kongruence totiž při výběru jiného reprezentanta třídy $[x]$, řekněme y, znemožňuje situaci, ve které $[xa] \neq [ya]$.

Jsme připraveni dokázat, že $L(M_\equiv) = R$. Nejprve několik pomocných tvrzení.

Lemma 5: $x \in R \iff [x] \in F$.

Důkaz: Implikace $\Rightarrow$ plyne z definice F a $\Leftarrow$ plyne z definice F a vlastnosti (ii) Myhill-Nerodových relací.

Lemma 6: $\hat{\delta}([x], y) = [xy]$.

Důkaz: Indukcí vzhledem k $|y|$. $\hat{\delta}([x], \epsilon) = [x] = [x\epsilon]$.

Za účelem indukčního kroku předpokládejme, že $\hat{\delta}([x], y) = [xy]$. Nechť dále $a \in \Sigma$. Máme

$$\begin{aligned}\hat{\delta}([x], ya) &= \delta(\hat{\delta}([x], y), a) \text{ (z definice } \hat{\delta}) \\ &= \delta([xy], a) \text{ (indukční předpoklad)} \\ &= [xya] \text{ (z definice } \delta).\end{aligned}$$

Věta 2: $L(M_{\equiv}) = R$.

Důkaz: Máme

$$\begin{aligned}x \in L(M_{\equiv}) &\iff \hat{\delta}([\epsilon], x) \in F \text{ (z definice akceptování)} \\ &\iff [x] \in F \text{ (z lemma 6)} \\ &\iff x \in R \text{ (z lemma 5)}\end{aligned}$$

Popsali jsme dvě přirozené konstrukce $M \mapsto \equiv_M$ (k danému DKA M bez nedosažitelných stavů sestrojí odpovídající Myhill-Nerodovu relaci $\equiv_M$ pro R) a $\equiv \mapsto M_{\equiv}$ (rekonstruující DKA $M_{\equiv}$ pro R z Myhill-Nerodovy relace $\equiv$ pro R). V tuto chvíli bychom rádi dokázali, že tyto operace jsou až na isomorfismus navzájem inverzní.

Lemma 7: *Nechť $\equiv$ je Myhill-Nerodova relace pro R. Aplikujeme-li konstrukci $\equiv \mapsto M_{\equiv}$ na relaci $\equiv$ a na výsledek konstrukci $M \mapsto \equiv_M$, pak výsledná relace $\equiv_{M_{\equiv}}$ je identická s původní relací $\equiv$.*

Důkaz: Nechť $M_{\equiv} = (Q, \Sigma, \delta, s, F)$ je DKA zkonstruovaný aplikací konstrukce $\equiv \mapsto M_{\equiv}$ na relaci $\equiv$. Pak pro libovolné $x, y \in \Sigma^*$ dostáváme:

$$\begin{aligned}x \equiv_{M_{\equiv}} y &\iff \hat{\delta}(s, x) = \hat{\delta}(s, y) \text{ (z definice } \equiv_{M_{\equiv}}) \\ &\iff \hat{\delta}([\epsilon], x) = \hat{\delta}([\epsilon], y) \text{ (z definice } s) \\ &\iff [x] = [y] \text{ (z Lemma 6)} \\ &\iff x \equiv y.\end{aligned}$$

Lemma 8: *Nechť M je DKA pro R bez nedosažitelných stavů. Aplikujeme-li konstrukci $M \mapsto \equiv_M$ na stroj M a na výsledek konstrukci $\equiv \mapsto M_{\equiv}$, pak výsledný DKA $M_{\equiv_M}$ je isomorfní s původním DKA M*

.

Důkaz: Nechť $M = (Q, \Sigma, \delta, s, F)$ a $M_{\equiv_M} = (Q', \Sigma, \delta', s', F')$. Z konstrukce připomeňme, že

- $[x] = \{y \mid y \equiv_M x\} = \{y \mid \hat{\delta}(s, y) = \hat{\delta}(s, x)\}$,
- $Q' = \{[x] \mid x \in \Sigma^*\}$,

- $s' = [\epsilon]$,
- $F' = \{[x] \mid x \in R\}$,
- $\delta'([x], a) = [xa]$.

Definujme zobrazení f takto:

$$f : Q' \to Q$$
$$f([x]) = \hat{\delta}(s, x).$$

Ukážeme, že f je isomorfismem DKA $M_{\equiv_M}$ na M. Z definice $\equiv_M$ plyne, že $[x] = [y] \iff \hat{\delta}(s, x) = \hat{\delta}(s, y)$, takže zobrazení f je na $\equiv_M$-třídách dobře definováno a je injektivní. Protože DKA M nemá nedosažitelné stavy, je f surjektivní. Celkem je f bijekce.

K důkazu, že f je též isomorfismus automatů, stačí ukázat, že f zachovává strukturu automatu, tedy počáteční stav, přechodovou funkci a koncové stavy. To znamená ukázat:

- $f(s') = s$,
- $f(\delta'([x], a)) = \delta(f([x]), a)$,
- $[x] \in F' \iff f([x]) \in F$.

Ovšem to je splněno následovně:

$$\begin{aligned} f(s') &= f([\epsilon]) \text{ (z definice } s') \\ &= \hat{\delta}(s, \epsilon) \text{ (z definice } f) \\ &= s \text{ (z definice } \hat{\delta}); \end{aligned}$$

$$\begin{aligned} f(\delta'([x], a)) &= f([xa]) \text{ (z definice } \delta') \\ &= \hat{\delta}(s, xa) \text{ (z definice } f) \\ &= \delta(\hat{\delta}(s, x), a) \text{ (z definice } \hat{\delta}) \\ &= \delta(f([x]), a) \text{ (z definice } f); \end{aligned}$$

$$\begin{aligned} [x] \in F' &\iff x \in R \text{ (z definice } F \text{ a vlastnosti (ii))} \\ &\iff \hat{\delta}(s, x) \in F \text{ (protože } L(M) = R) \\ &\iff f([x]) \in F \text{ (z definice } f). \end{aligned}$$

Celkem jsme dokázali následující větu.

Věta 3: *Nechť Σ je konečná abeceda. Až na isomorfismus existuje injektivní korespondence mezi deterministickými konečnými automaty nad abecedou Σ bez nedosažitelných stavů akceptujících množinu R a Myhill-Nerodovými relacemi pro R na množině Σ^*.*

4.6 Myhill-Nerodova věta

Na základě výsledku dosaženého v minulé části dokážeme, že existuje *nejhrubší* Myhill-Nerodova relace $\equiv_R$ pro libovolnou, ale pevně danou regulární množinu R. To znamená, že jakákoliv jiná Myhill-Nerodova relace pro R rozklad na množině R zjemňuje. Relace $\equiv_R$ přitom koresponduje s jediným minimálním DKA pro R (až na isomorfismus).

Definice 45: Říkáme, že relace $\equiv_1$ *zjemňuje* relaci $\equiv_2$, jestliže $\equiv_1 \subseteq \equiv_2$.

Jinými slovy, $\equiv_1$ zjemňuje $\equiv_2$, jestliže pro všechna x a y skutečnost $x \equiv_1 y$ implikuje fakt, že $x \equiv_2 y$. Pro relace ekvivalence to znamená tolik, že pro každé x je $\equiv_1$-třída pro x obsažena v $\equiv_2$-třídě pro x.

Například relace ekvivalence $x \equiv y \bmod 6$ na celých číslech zjemňuje relaci ekvivalence $x \equiv y \bmod 3$. Také kupříkladu vlastnost (ii) Myhill-Nerodových relací říká, že Myhill-Nerodova relace $\equiv$ pro R zjemňuje relaci ekvivalence na R s třídami ekvivalence R a $\Sigma^* - R$. Relace *zjemnění* je částečné uspořádání, neboť je reflexivní (každá relace zjemňuje sama sebe), antisymetrická (jestliže $\equiv_1$ zjemňuje $\equiv_2$ a současně $\equiv_2$ zjemňuje $\equiv_1$, pak $\equiv_1$ a $\equiv_2$ jsou tytéž relace) a transitivní (pokud $\equiv_1$ zjemňuje $\equiv_2$ a současně $\equiv_2$ zjemňuje $\equiv_3$, pak $\equiv_1$ zjemňuje $\equiv_3$). Navíc na libovolné množině U vždy existuje *nejjemnější* a *nejhrubší* relace ekvivalence, tzv. *identická relace* $\{(x,x) \mid x \in U\}$, respektive *universální relace* $\{(x,y) \mid x,y \in U\}$.

Definice 46: Nechť $R \subseteq \Sigma^*$ je libovolná (ne nutně regulární) podmnožina. Definujeme relaci $\equiv_R$ na množině Σ^* následovně

$$x \equiv_R y \overset{def}{\iff} \forall z \in \Sigma^* \; (xz \in R \iff yz \in R).$$

Neformálně řečeno, dva řetězce x a y jsou v $\equiv_R$ ekvivalentní, jestliže připojením téhož řetězce na konec obou vzniknou řetězce, které buď oba patří do R, nebo oba nepatří do R. Není těžké ukázat, že pro libovolnou množinu R je relace $\equiv_R$ ekvivalencí.

Dále ukážeme, že pro libovolnou množinu R splňuje relace $\equiv_R$ vlastnosti (i) a (ii) Myhill-Nerodových relací a je na množině Σ^* nejhrubší takovou relací. V případě, že R je regulární, má $\equiv_R$ též konečný index, a tedy je Myhill-Nerodovou relací pro R. Ve skutečnosti jde o nejhrubší možnou Myhill-Nerodovu relaci pro R a odpovídá tak jedinému minimálnímu DKA pro R.

Lemma 9: *Nechť $R \subseteq \Sigma^*$ je libovolná podmnožina, ne nutně regulární. Relace $\equiv_R$ je pravou kongruencí zjemňující R a je nejhrubší možnou takovou relací na Σ^*.*

Důkaz: K důkazu, že $\equiv_R$ je pravou kongruencí, vezměme v její definici $z = aw$, dostáváme:

$$\begin{aligned} x \equiv_R y &\implies \forall a \in \Sigma, \forall w \in \Sigma^* \; (xaw \in R \iff yaw \in R) \\ &\implies \forall a \in \Sigma \; (xa \equiv_R ya) \end{aligned}$$

Pro důkaz toho, že $\equiv_R$ zjemňuje R, vezmeme $z = \epsilon$ v definici $\equiv_R$:

$$x \equiv_R y \Longrightarrow (x \in R \Longleftrightarrow y \in R).$$

Navíc $\equiv_R$ je nejhrubší takovou relací, protože libovolná jiná relace ekvivalence splňující vlastnosti (i) a (ii) Myhill-Nerodových relací zjemňuje $\equiv_R$:

$$\begin{aligned} x \equiv y &\Longrightarrow \forall z \in \Sigma^* \ (xz \equiv yz) \text{ (indukcí vzhledem k } |z| \text{ a z vlastnosti (i))} \\ &\Longrightarrow \forall z \in \Sigma^* \ (xz \in R \Longleftrightarrow yz \in R) \text{ (vlastnost (ii))} \\ &\Longrightarrow x \equiv_R y \text{ (z definice } \equiv_R). \end{aligned}$$

V tomto bodě máme připravena všechna tvrzení a definice potřebné k důkazu vlastní Myhill-Nerodovy věty.

Věta 4: *Nechť $R \subseteq \Sigma^*$. Následující tři tvrzení jsou ekvivalentní.*

(a) *R je regulární.*

(b) *Existuje Myhill-Nerodova relace pro R.*

(c) *Relace $\equiv_R$ má konečný index.*

Důkaz: (a) $\Rightarrow$ (b) : Nechť je dán DKA M pro R, pak konstrukce $M \mapsto \equiv_M$ produkuje Myhill-Nerodovu relaci pro R.

(b) $\Rightarrow$ (c) : Podle lemma 9 má jakákoliv Myhill-Nerodova relace pro R konečný index a zjemňuje $\equiv_R$, tudíž $\equiv_R$ má konečný index.

(c) $\Rightarrow$ (a) : Má-li $\equiv_R$ konečný index, pak je Myhill-Nerodovou relací pro R a konstrukce $\equiv \mapsto M_{\equiv}$ produkuje DKA pro R.

Protože $\equiv_R$ je jednoznačnou nejhrubší Myhill-Nerodovou relací pro regulární množinu R, koresponduje s DKA, který má nejmenší počet stavů mezi všemi DKA pro R. Minimalizační algoritmus právě tento DKA konstruuje. Předpokládejme, že $M = (Q, \Sigma, \delta, s, F)$ je DKA pro R, který je již minimalizován minimalizačním algoritmem. To znamená, že neobsahuje nedosažitelné stavy a relace $\approx$ dříve definovaná jako:

$$p \approx q \overset{def}{\Longleftrightarrow} \forall x \in \Sigma^* \ (\hat{\delta}(p, x) \in F \Longleftrightarrow \hat{\delta}(q, x) \in F)$$

je identickou relací na Q. Pak Myhill-Nerodova relace $\equiv_M$ odpovídající deterministickému konečnému automatu M je právě $\equiv_R$:

$$\begin{aligned} x \equiv_R y &\Longleftrightarrow \forall z \in \Sigma^* \ (xz \in R \Longleftrightarrow yz \in R) \text{ (z definice } \equiv_R) \\ &\Longleftrightarrow \forall z \in \Sigma^* \ (\hat{\delta}(s, xz) \in F \Longleftrightarrow \hat{\delta}(s, yz) \in F) \text{ (z definice akceptování)} \\ &\Longleftrightarrow \forall z \in \Sigma^* \ (\hat{\delta}(\hat{\delta}(s, x), z) \in F \Longleftrightarrow \hat{\delta}(\hat{\delta}(s, y), z) \in F) \text{ (indukcí na } |z|) \\ &\Longleftrightarrow \hat{\delta}(s, x) \approx \hat{\delta}(s, y) \text{ (z definice } \approx) \\ &\Longleftrightarrow \hat{\delta}(s, x) = \hat{\delta}(s, y) \text{ (protože } M \text{ je minimalizovaný)} \\ &\Longleftrightarrow x \equiv_M y \text{ (z definice } \equiv_M). \end{aligned}$$

4.7 Vztah trie a konečných automatů

Velikost struktury trie v podobě stromu je dána počtem uzlů. Jistě pozornému čtenáři neuniklo, že při ukládání stromů, a trie je speciálním případem stromu, naprosto zanedbáváme hrany spojující jednotlivé uzly. Hrany prostě ukládány nejsou, neboť hierarchická struktura stromu je, jak již víme z předchozí kapitoly, plně rekonstruovatelná z pořadí uložených uzlů. Z tohoto hlediska je počet uzlů nejvýznamnějším faktorem ovlivňujícím celkovou velikost stromu.

Při snaze minimalizovat velikost trie vyvstává tedy okamžitě otázka minimalizace počtu uzlů ve struktuře. Přitom minimalizace nesmí porušit informaci, kterou bychom rádi v trie vyhledávali. Tyto požadavky se shodují s požadavky, které klademe na minimalizaci deterministických konečných automatů. I zde se snažíme co nejvíce snížit počet stavů a přitom zachovat množinu akceptovanou automatem.

Analogie je ještě markantnější, jestliže si uvědomíme, že i deterministické konečné automaty lze znázornit graficky pomocí grafů, které opět ve speciálním případě tvoří strom, a že tabulka pro trie vlastně vyjadřuje jakousi přechodovou funkci. Stačí tedy provést o něco formálnější popis korespondence mezi deterministickým konečným automatem a strukturou trie.

Viděli jsme, že k trie v podobě tabulky lze zkonstruovat odpovídající strom, podobně i naopak lze přirozeným způsobem ze stromové reprezentace trie získat jemu odpovídající tabulku. Jednoduše se provede očíslování uzlů stromu (například průchodem do šířky nebo do hloubky) tak, že kořen má vždy číslo 0.

Například vezměme obrázek 3.13. Při očíslování uzlů tohoto stromu v pořadí odpovídajícím průchodu stromem do šířky od 0 dostáváme tabulku 4.1.

t	␣	a	e	i	o	p	r	s	v	z
0	-	-	-	-	-	1	-	2	3	4
1	-	-	-	-	-	-	5	-	-	-
2	+	-	6	7	-	-	-	-	-	-
3	+	-	-	-	-	-	-	-	-	-
4	-	8	-	-	-	-	-	-	-	-
5	-	-	-	-	9	-	-	-	-	-
6	+	-	-	-	-	-	-	-	-	-
7	+	-	-	-	-	-	-	-	-	-
8	+	-	-	-	-	-	-	-	-	-
9	+	-	-	-	-	-	-	-	-	-

Tabulka 4.1: Trie

Každému uzlu stromu odpovídá jeden řádek tabulky. Sloupce jsou označeny písmeny, jimiž byly původně pojmenovány uzly stromu. Každému sloupci odpovídá právě jedno písmeno. Každé políčko tabulky, které je určeno řádkem odpovídajícím uzlu `n` a sloupcem označeným písmenem `c`, obsahuje:

- symbol +, je-li c = ⊔ a uzel n je listem,
- symbol -, neexistuje-li syn uzlu n v původním stromě označený písmenem c,
- číslo syna uzlu n v původním stromě označeného písmenem c.

Označíme-li nyní množinu uzlů (jejich číselných označení) jako N a množinu všech symbolů vyskytujících se v trie jako C, pak předchozí tabulka odpovídá funkci $t : N \times C \to N \cup \{+,-\}$. Konečně položme $L = \{n \in N \mid t(n, ⊔) = +\}$. Při uvedeném značení definujme:

- $Q \stackrel{def}{=} N$,
- $\Sigma \stackrel{def}{=} C$,
- $\delta : Q \times \Sigma \to Q$ tak, že $\delta(q, a) \stackrel{def}{=} t(q, a)$ pro $t(q, a) \notin \{+,-\}$, jinak nedefinováno,
- $s \stackrel{def}{=} 0$,
- $F \stackrel{def}{=} L$.

Potom pětice $(Q, \Sigma, \delta, s, F)$ je deterministický konečný automat odpovídající struktuře trie dané tabulkou pro funkci t. Počet jeho stavů a tím i počet uzlů trie může být minimalizován.

Implementační podrobnosti týkající se reprezentace ekvivalentních uzlů apod. uvedeme v příslušné části následující kapitoly.

Kapitola 5

Implementace morfologického analyzátoru

Morfologický analyzátor byl implementován na základě algoritmického popisu české formální morfologie [Osol–96]. Zvolili jsme slovníkový přístup, to znamená, že veškerá data potřebná ke správné funkci morfologického analyzátoru jsou uložena ve strojovém slovníku češtiny a v definičním souboru koncovkových množin a vzorů. Popis formátu těchto souborů bude náplní následujících dvou částí. Ve třetí části kapitoly podrobně popíšeme program `abin`, který převádí uvedené soubory do binárního tvaru a vytváří tak struktury dat využívané samotným morfologickým analyzátorem `ajka`. Popis implementace analyzátoru je uveden v poslední části této kapitoly. V úvodu ještě poznamenejme, že oba programy `abin` i `ajka` byly napsány v programovacím jazyce C podle normy ANSI.

5.1 Formát strojového slovníku češtiny

Strojový slovník češtiny jsme se pokusili navrhnout tak, aby byl uživatelsky co možná nejjednodušší a přitom bylo možné data v něm uložená dále používat pro jiná lingvistická bádání a experimenty. Jedná se o textový soubor, takže je snadno editovatelný běžnými textovými editory. Pro rozlišení od ostatních textových souborů jsme mu přidělili příponu `.dic`.

Základním požadavkem bylo, aby byl slovník, resp. data v něm, libovolně přemístitelný a bylo tak možné slovník rozdělit do více souborů.

Hlavní stavební jednotkou jakéhokoliv slovníku je heslo. Nejinak tomu je i v našem případě. Navíc lze v našem slovníku hesla jistým způsobem související sdružovat do sekcí, oddílů. Heslo jako takové má tři části. První dvě jsou povinné, třetí je fakultativní:

- část definovaná (lexikální), která obsahuje základní tvar slova (lemma), případně další tvary, pokud u nich dochází ke změně podoby kmene vzhledem ke tvaru základnímu

- část definující (gramatická) zahrnující informaci o vzoru, u sloves dále informaci o vidu, reflexivitě a možnosti tvoření negativní formy, u adjektiv o možnosti připojení za číselný prefix a tvoření negativní formy

- část obsahující prefixy, pokud slovní tvar může mít prefixy a přidáním prefixu se nezmění gramatická informace ve druhé části hesla

Podrobně se nyní budeme věnovat jednotlivým částem hesla. Lexikální část se může skládat z jednoho nebo více slovních tvarů. Více slovních tvarů se uvádí zpravidla tam, kde mezi jednotlivými tvary dochází ke změně podoby kmenového základu. Ostatně tyto změny jsou zahrnuty v rámci definice vzoru, takže při pořizování hesla by již měla být definující část známa. Počet slovních tvarů v lexikální části hesla tedy musí odpovídat počtu změn v podobě kmenového základu danému definicí vzoru. Prvním slovním tvarem však musí být tvar základní (lemma). Neohebná slova mají jediný tvar, považujeme jej tedy za tvar základní. Základní tvar ohebných slov závisí na slovním druhu. Rozhodnutí, které slovní tvary budou uváděny, je-li jich více, jsme učinili s ohledem na minimalizaci počtu výjimek. Ani tak však nebylo v našich silách se výjimkám ubránit. Přehledně požadované tvary zachycuje tabulka 5.1.

Slovní druh	**Základní tvar**	**Druhý tvar**
podstatná jména	1. p. j./mn. čísla	2. p. mn. čísla
přídavná jména	1. p. j. č. muž. rodu	2. st. odvozeného adverbia
zájmena	1. p. j./mn. čísla	2. p. j. čísla
číslovky	1. p. j./mn. čísla č. základní	1. p. j. čísla č. řadové
slovesa	infinitiv	rozkaz. zp. 2. os. j. čísla

Tabulka 5.1: Požadované tvary slov ve slovníku

Jednotlivé slovní tvary v lexikální části hesla, pokud jich je více, jsou odděleny čárkou. V této části hesla lze použít též zkráceného zápisu, zvláště v případech, kdy je možný dvojí pravopis. Jsou povoleny pouze dvě alternativy, které se píší do složených závorek a vzájemně se oddělují znakem „|“. Například dvojí pravopis slova *gymnasium* a *gymnázium* lze úsporněji zapsat do našeho strojového slovníku jako *gymn*{*as*|*áz*}*ium*.

Ve speciálním případě, kdy jde o sloveso, u něhož dochází v negativní formě ke změně podoby kmenového základu, se užívá za jednotlivými tvary symbolů „!“ a „@“. Jejich význam je následující. Vyskytne-li se znak „!“ za slovním tvarem, znamená to, že příslušná podoba kmenového základu obsažená v tomto tvaru je shodná s podobou kmenového základu negativní formy, tzn., že dodáním negativního prefixu *ne-* před daný slovní tvar vznikne korektní negativní forma. Jinými slovy, výskyt znaku „!“ indikuje, že slovní tvar může tvořit negativní formu jednoduše předsazením prefixu *ne-*. Výskyt znaku „@“ je povolen za tvary, které již jsou v negativní formě. Pak přítomnost tohoto znaku znamená, že kmenový základ obsažený v daném tvaru je korektní pouze pro negativní formu, nelze od něj tedy negativní prefix *ne-* odtrhnout. Chybí-li za slovním tvarem oba znaky, je tvar chápán jako pozitivní forma, jejíž kmenový základ nemůže být nikdy součástí formy negativní. Pro objasnění uveďme příklad.

```
hnát, nehnat@, hnal!, žeň! : hnát
```

Řádek znamená, že slovní tvar *hnát* je v pozitivní formě a jeho kmenový základ negativní formu tvořit nemůže (*nehnát* není korektní negativní formou). Opačně, tvar *nehnat* je korektní negativní formou, jeho kmenový základ ovšem není nikdy součástí pozitivní formy (slovo *hnat* není správně utvořeno). Zbylé dva tvary *hnal* a *žeň* obsahují kmenové základy, které mohou být součástí jak pozitivní, tak i negativní formy (*nehnal* i *nežeň* jsou korektní).

Na závěr k lexikální části hesla ještě dodejme, že je podstatná i jeho kapitalizace. Všímáme si dvou speciálních případů. Jedním jsou slova, která je nutno psát s velkým počátečním písmenem (např. vlastní jména) nebo se všemi písmeny velkými (některé zkratky, jako *ČR* apod.). V tomto případě je třeba uvést všechny tvary v lexikální části v korektní podobě. Slova, jako kupříkladu *CSc.*, jsme prozatím neuvažovali, tzn., že ve slovníku je vhodné je uvést ve správném pravopisu, nicméně jako korektní jsou analyzovány i jejich ne zcela správné tvary (např. *CSC.* a *CsC.*). Podobně i v druhém případě, kdy se jedná o zkratky, které se píší na konci s tečkou, je nutné je do slovníku zařadit v korektním tvaru, tj. s tečkou na konci.

K předchozímu přístupu nás vedla snaha o to, aby se v definované části hesla vyskytovaly jen správné tvary českých slov, a také nutnost alespoň výskyt velkého počátečního písmene zachytit. Například značná část českých příjmení vznikla z původních substantiv nebo adjektiv. Jelikož jsme se rozhodli analyzovat i ženská příjmení, je pro nás znalost velikosti prvního písmene podstatná. Například slova *Sedláček* a *sedláček* jsou obě analyzována jako správná s tím rozdílem, že v prvním případě jde o mužské příjmení, zatímco v druhém o životné maskulinum. Na druhé straně, první ze slov *Sedláčková* a *sedláčková* je správně utvořeno, druhé nikoliv. Zavedení dalšího speciálního znaku indikujícího nutnost psaní prvního velkého písmene se nám nezdálo být vhodné, neboť psaní vlastních jmen s velkým písmenem na začátku je mnohem přirozenější, než psaní všech písmen malých. Další speciální symbol navíc pouze ztěžuje čitelnost slovníku.

Další částí hesla je část gramatická, nebo též definující. Musí se vyskytovat na tomtéž řádku jako část definovaná, přičemž je od ní oddělena dvojtečkou. Definující část hesla obsahuje povinně vzor, pod který tvary v definované části hesla spadají a nepovinně některý ze speciálních symbolů:

- znak „!“ má smysl u sloves a adjektiv a značí možnost tvoření negativní formy. Pokud předchází speciální podoba definované části hesla pro slovesa, která znak „!“, případně znak „@“ již obsahuje, je znak „!“ v definující části ignorován;
- znak „%“ u sloves indikuje, že sloveso je reflexivní;
- znak „*“ znamená, že sloveso je dokonavé;
- znak „~“ má význam jen u adjektiv, kde umožňuje, aby adjektivum bylo připojeno za číslovku a tvořilo tak s ní kompozitum.

Uvedeme příklad s komentářem, aby použití výše uvedených symbolů bylo naprosto jasné. Uvažujme následující řádky strojového slovníku.

```
cejchovat : kupovat   !
chlubit   : bavit     %!
padnout   : klovnout  *!
metrákový : kovový    ~
sobecký   : otrocký   !
```

Dle hesel je sloveso *cejchovat* nedokonavé a může tvořit negativní formu *necejchovat*. Toto sloveso není reflexivní. Druhé heslo v pořadí nás informuje o tom, že sloveso *chlubit* reflexivní je (např. *chlubím se*) a opět se může vyskytovat i v negativní formě (*nechlub se*). Sloveso *chlubit* je nedokonavé, na rozdíl od slovesa *padnout* na třetím řádku, které se též vyskytuje v negativní formě. Přídavné jméno *metrákový* je příkladem adjektiv, která spolu s číslovkou mohou tvořit kompozita, jako např. *pětimetrákový* atd. Všimněme si, že negativní forma přípustná není. Poslední heslo, *sobecký*, upozorňuje na to, že i adjektiva mohou tvořit negativní formy, v tomto případě *nesobecký*, což značí vykřičník v gramatické části hesla.

Definující část tvoří konec prvního řádku hesla. Heslo buď takto končí nebo pokračuje na novém řádku prefigovaném znakem „^“. Tento nepovinný řádek (resp. řádky) obsahuje seznam prefixů, které mohou být předřazeny všem kmenovým základům obsaženým v tvarech lexikální části předchozího hesla, aniž se tím poruší platnost informace reprezentované gramatickou částí téhož hesla. Je patrné, že hlavní použití těchto řádků je ve spojení se slovesy. Ale i pro jiné slovní druhy lze tohoto zápisu využít.

Motivací zavedení tohoto způsobu uložení prefixů bylo usnadnění práce uživateli, který tak nemusí stále opisovat tvary s neměnným kmenovým základem pro různé prefixy, zmenšení rozsahu slovníku, ale hlavně možnost dalšího výzkumu a zjišťování závislostí výskytu jednotlivých prefixů u sloves. Takto je otevřena cesta k nalezení jistých pravidel určujících, které prefixy se s daným slovesem pojí a které nikoliv.

Pokud slovní tvar žádné prefixy nepřijímá, jednoduše tyto řádky ve slovníku neuvádíme. V případě, že slovní tvar je správný nejen s některými prefixy, ale i sám o sobě, tedy neprefigovaný, uvádí se do seznamu prefixů speciální symbol „_“. Ten určuje možnost prázdného prefixu. Jednotlivé prefixy, včetně „_“, jsou odděleny čárkami. Za posledním čárka samozřejmě chybí. Pro ilustraci může část strojového slovníku vypadat takto:

```
 kudy        : kde
^ _, ně, ni

 lézt, lez   : snést!
^ _, do, na, ob, od, o, po, pod, pood, popo, povy, poza
^ pro, pře, při, roz, s, u, v, vy, vyna, z, za, zpře
```

Příklad je dostatečně výstižný, proto se domníváme, že dalšího komentáře na tomto místě není třeba. Snad jenom fakt, že hesla ve slovníku není nutné nijak

uspořádávat, například podle abecedy. Mohou se prolínat jednotlivé slovní druhy, ohebné i neohebné. V tomto smyslu je formát slovníku dostatečně volný. Z příkladů je také vidět, že mezery jsou ignorovány, stejně tak prázdné řádky. I estetická stránka slovníku je tedy ponechána na uživateli. Ten si navíc může činit poznámky přímo do slovníku. Slouží pro to znak „#“, který takové poznámky uvozuje. Vše, co následuje za tímto znakem až do konce řádku, je ignorováno.

Ačkoliv je uživateli ponecháno na libovůli, jakým způsobem hesla ve slovníku uspořádá, zdálo se nám užitečné přeci jen některé základní „formátovací“ procesy uživateli usnadnit. Implementovali jsme proto převodní program tak, aby umožňoval sdružování hesel do jistých sekcí. Vycházeli jsme přitom z definující části hesla. Informace, které jsou společné všem heslům v uvažované sekci, lze umístit do tzv. *hlavičky sekce*.

Hlavičku sekce indikuje znak „$“. Za ním se mohou vyskytovat všechny informace jinak obsažené v gramatické části hesla. Význam je takový, že všechna hesla za hlavičkou až do výskytu nové definice hlavičky nebo konce souboru budou chápána, jako by měla informaci z hlavičky uloženu ve své vlastní definující části. O jakou informaci se jedná? Jde o vzor a dále o speciální symboly „!“, „%“, „*“ a „~“.

Zde je nutné vysvětlit některé z možností. Hlavička může být prázdná, pak její význam je stejný jako význam prázdného řádku, jedná se tedy pouze o vizuální oddělovač. Dále se v hlavičce může objevit pouze vzor. Tehdy se nemusí uvádět znak „:“ oddělující obě části hesla v případě, že definující část neobsahuje speciální znaky (je prázdná). Musí-li být tyto znaky v gramatické části hesla uvedeny, je nutné je dvojtečkou oddělit od lexikální části hesla. Další možností je, že hlavička obsahuje pouze speciální znaky. Pak heslo má klasickou strukturu, definující část je oddělena dvojtečkou a musí obsahovat vzor, případně s dodatečnými speciálními znaky dále specifikujícími gramatickou informaci hesla. Posledním případem je situace, kdy hlavička obsahuje vzor i s gramatickou informací v podobě speciálních znaků. Všechna následující hesla pak tuto informací zdědí do své definující části. Pokud je to veškerá informace potřebná ke korektní definici hesla, je gramatická část hesla prázdná, tedy znak „:“ nemusí být uveden. Jinak, jestliže je potřeba informaci dále specifikovat, provedeme to v definující části hesla oddělené dvojtečkou uvedením příslušných speciálních znaků, vzor zde již neuvádíme. Názorně ukazuje použití sekcí následující výsek ze slovníku.

```
$             # prázdná hlavička
patnáct : jedenáct
patník  : krk
ostrý   : moudrý!

$ metrový     # téhož vzoru
arový       : ~
drobnolistý
vyjádřený   : !
```

```
$ *           # dokonavá slovesa
padnout          : klovnout!
dotázat, dotaž   : vyvázat%! # některá jsou reflexivní
dovtípit, dovtip : dávit%!

$ chválit*!   # dokonavá téhož vzoru
docílit,docil
odvážit,odvaž # ve smyslu vážení
odvážit,odvaž : %  # reflexivní ve smyslu odvahy
naléčit,naléč
```

Jestliže dojde ke kolizi s těmito pravidly, je řádek při čtení programem `abin` ignorován a je vypsáno chybové hlášení na standardní chybový výstup. Stejně tak, dojde-li k duplikaci gramatické informace v hlavičce a definující části hesla.

Možnost sdružování hesel vidíme jako velmi užitečnou. Například je možné sdružovat hesla stejného vzoru, nebo hesla se stejnou gramatickou informací. Příkladem může být sdružení všech adjektiv, která nemohou tvořit negativní formu, popřípadě sloves, která jsou reflexivní, či dokonavá. Je možné i kombinovat, tj. například sdružit slovesa nedokonavá, která nejsou reflexivní, nebo adjektiva skloňující se podle vzoru *metrový*, která ovšem nemohou tvořit kompozitum s číslovkou apod. Takový slovník se stává podkladem pro další zajímavé lingvistické experimenty.

5.2 Formát definičního souboru koncovkových množin a vzorů

Definiční soubor koncovkových množin a vzorů obsahuje nejpodstatnější informaci týkající se morfologie českých slov, proto je tento soubor nejdůležitější součástí morfologického analyzátoru. Jak jeho název napovídá, obsahuje definice koncovkových množin a vzorů tak, jak byly navrženy v [Osol–96]. Také formát souboru byl z převážné části převzat z této práce. Je to textový soubor, lze jej proto editovat běžnými textovými editory, má implicitní příponu `.par`.

Záznamy souboru je možno rozdělit do dvou kategorií:

- definice koncovkové množiny
- definice vzoru

V následujícím textu postupně popíšeme strukturu obou těchto záznamů. Definice koncovkové množiny začíná na začátku řádku znakem „=“, za nímž následuje jednoznačný identifikátor koncovkové množiny (její jméno). Každá koncovková množina se skládá z bloků dvojic. Jednotlivé dvojice jsou uvedeny vždy na samostatném řádku, přičemž první člen dvojice tvoří koncovka, druhý člen pak znak odpovídající hodnotě jisté gramatické kategorie závislé na slovním druhu. Přesněji tuto závislost vyjadřuje tabulka 5.2 (příslušná gramatická kategorie je uvedena na posledním řádku). Dvojice je uzavřena do kulatých závorek a jednotlivé komponenty jsou odděleny čárkou.

Blok takovýchto dvojic je uvozen gramatickou značkou. Ta má podobu řetězce dvou až šesti znaků uzavřeného do hranatých závorek. Také tato uvozující značka musí být na samostatném řádku. Každé pozici ve značce odpovídá jistá gramatická kategorie. Opět je přiřazení gramatické kategorie k určité pozici ve značce závislé na druhu slova (podrobně viz tabulka 5.2). Znak uvedený na dané pozici pak reprezentuje hodnotu, kterou příslušná gramatická kategorie nabývá (viz příloha A).

Speciálními znaky, které se mohou vyskytnout na libovolné pozici značky kromě první pozice, jsou symboly „`.`“ a „_“. Znak „`.`“ se může vyskytnout pouze na jediné pozici ve značce. Tato pozice je předem určena pro každý slovní druh (viz též tabulka 5.2). Výskyt „`.`“ tak identifikuje pozici ve značce, kam bude přemístěn znak, jenž zastupuje hodnotu gramatické kategorie obsažené v druhém prvku každé z následujících dvojic. Každé dvojici tvořené koncovkou a hodnotou jisté gramatické kategorie tak odpovídá značka, která vznikne nahrazením znaku „`.`“ symbolem z druhé části dvojice ve značce uvozující blok, v němž se daná dvojice nalézá. Takto získáváme pro každou koncovku téměř úplnou gramatickou informaci.

Druhý speciální znak „_“ ve značce indikuje, že gramatická kategorie, která odpovídá pozici s tímto znakem, nemá definovanou hodnotu, protože například nemá smysl tuto kategorii pro daný slovní tvar zkoumat. Takže se tento znak může vyskytnout například na pozici odpovídající osobě ve značce, která uvozuje blok koncovek pro infinitiv slovesa. Je jasné, že určovat osobu nemá v případě infinitivu smysl.

Bylo by vhodné ještě dodat, že symbol „_“ lze použít i na místě kterékoliv komponenty dvojice koncovka, gramatická kategorie. Na místě koncovky znamená, že jde o nulovou koncovku (prázdný řetězec), v místě hodnoty gramatické kategorie značí, že je nedefinována.

Na vysvětlenou shlédněme tabulku 5.2. Mezi slovními druhy se vyskytují dvakrát adverbia. Poprvé se jedná o adverbia, která nelze automaticky vygenerovat z adjektiv, podruhé jde o adverbia, která z adjektiv automaticky generována jsou. Na předposledním řádku tabulky je uvedena struktura gramatické značky pro deverbativní substantiva, tj. substantiva odvozená ze sloves, poslední řádek tabulky se týká deverbativních adjektiv, tj. adjektiv taktéž odvozených ze sloves.

Nyní se podívejme kupř. na řádek pro přídavná jména. Na první pozici značky bude hodnota identifikující slovní druh, na druhé hodnota gramatické kategorie rod. Na třetí nalezneme hodnotu čísla. Čtvrtá pozice je obsazena tečkou, to znamená, že na tuto pozici se zkopíruje hodnota z druhé části dvojice. Příslušná gramatická kategorie, jejíž hodnotu tečka zastupuje, je pád, to je vidět z posledního sloupce tabulky. Na páté pozici je hodnota odpovídající stupni, šestá pozice je neobsazena (znak „—“). Přehledný popis symbolů, které představují konkrétní hodnoty jednotlivých gramatických kategorií, uvádíme v příloze A.

Pro úplnost ještě dodejme, že v definičním souboru koncovkových množin a vzorů se vyskytuje jedna speciální značka `[2CO.M]`. Je určena pro uvození bloku koncovek adejktiv, která mohou tvořit první část kompozit, např. *pražsko-* apod.

Slovní druh	Pozice v gramatické značce						Gram. kat.
	1.	2.	3.	4.	5.	6.	
Podst. jm.	sl. druh	rod	číslo	.	—	—	pád
Příd. jm.	sl. druh	rod	číslo	.	stupeň	—	pád
Zájmena	sl. druh	druh	rod	číslo	.	osoba	pád
Číslovky	sl. druh	druh	rod	číslo	.	—	pád
Slovesa	sl. druh	.	číslo	čas	způsob	vid	osoba
Příslovce	sl. druh	druh	stupeň	.	—	—	—
Předložky	sl. druh	.	—	—	—	—	pád
Spojky	sl. druh	druh	.	—	—	—	—
Částice	sl. druh	.	—	—	—	—	—
Citoslovce	sl. druh	.	—	—	—	—	—
Zkratky	sl. druh	.	—	—	—	—	—
Příslovce	sl. druh	druh	stupeň	.	—	—	—
Posesiva	sl. druh	rod	přir. rod	číslo	.	—	pád
Dev. subst.	sl. druh	rod	číslo	.	—	—	pád
Dev. adj.	sl. druh	rod	číslo	.	stupeň	—	pád

Tabulka 5.2: Struktura značky jednotlivých slovních druhů

Na závěr popisu záznamu definujícího koncovkovou množinu uvádíme odpovídající část definičního souboru pro ilustraci.

```
=V2
  [1IP.]
   (ů,2)
   (y,1)
   (ům,3)
   (y,4)
   (y,7)
  [1IS.]
   (u,3)
   (em,7)

=V24910F
  [1FS.]
   (_,1)
   (_,4)

=V2UA
  [1IS.]
   (u,2)
   (a,2)
```

Druhým typem záznamu v definičním souboru je definice vzoru. Podobně jako definice koncovkové množiny, začíná na novém řádku znakem „+“ následovaným jednoznačným jménem vzoru. Dále pokračují vždy samostatné řádky, které mají dvě povinné a jednu fakultativní část. Každý takový řádek musí obsahovat intersegment uzavřený z levé strany znakem „<“ a z pravé znakem „>“. Za ním pokračuje seznam jmen koncovkových množin. Jména jsou vzájemně oddělena čárkami.

Je nutné, aby jména koncovkových množin uvedená za intersegmentem již byla známa, tj. definice koncovkových množin, na které se pomocí jejich jmen odkazujeme, musí předcházet definici tohoto vzoru. V případě, že některý z intersegmentů má vliv na změnu slovního druhu, je zapotřebí tento intersegment uvést na samostatný řádek s koncovkovými množinami příslušejícími k tomuto druhu.

Stejně tak z důvodu rychlejší analýzy jsme se dohodli, že první koncovka první koncovkové množiny prvního intersegmentu bude příslušet základnímu tvaru pro odpovídající slovní druh. Pokud některý intersegment kromě změny slovního druhu způsobí i změnu základního tvaru, je nevyhnutelné toto pravidlo dodržet. Tedy opět první koncovka první koncovkové množiny v prvním intersegmentu pro tento druh patří základnímu tvaru.

Za seznamem koncovkových množin může následovat seznam postfixů, které by měly být shodné pro všechny intersegmenty daného vzoru, jedná-li se skutečně o postfixy. Postfixy od koncovkových množin oddělujeme znakem „&“. Jako v každém seznamu i v seznamu postfixů jsou jednotlivé prvky odděleny čárkou. V případě, že slovní tvar může kromě některých postfixů existovat i samostatně, bez postfixu, uvádíme jako první prvek seznamu postfixů znak „_“. Podobně jako u koncovkových množin, lze tohoto znaku použít na místě intersegmentu, kde znamená prázdný intersegment.

Definice vzoru končí prázdným řádkem. V tomto případě má prázdný řádek svůj význam. Slouží totiž k oddělení vzorů. Existují případy, kdy vzor zachycuje změnu podoby kmenového základu. Těchto změn může být ve slově pro jeden základní tvar více. Obvykle změna podoby kmenového základu znamená změnu hodnot některých gramatických kategorií, kterou je třeba zachytit společně se změnou podoby kmenového základu. Tato situace byla v [Osol–96] vyřešena zavedním dalšího vzoru pro každou změnu podoby kmenového základu. Současně je ale nezbytné uchovat informaci o tom, že dané podoby kmenového základu a jím příslušející vzory patří k jednomu a témuž základnímu tvaru. Proto se v takovém případě jednotlivé vzory patřící k jednotlivým alternativám podoby kmenového základu uvádějí těsně po sobě, a to bez vynechávání volných řádků. Pracovně jsme takové skupiny vzorů nazvali *vícevzory* a skupiny různých podob kmenových základů jako *alternativní* nebo *sdružené*. Celou situaci ilustruje následující příklad.

```
+génius
  <us> V13X
  <_> V13M,VOVE,VVE,VQM
  <ův> PRIVL1X
  <ov> PRIVL1
```

```
# dvojvzor
+daněk
  <ěk> V13X
+daňků
  <k> V1,VOVE,VVU
  <kův> PRIVL1X
  <kov> PRIVL1
  <c> VI,VQM
```

Formát definičního souboru není nijak pevně dán. Musí být zachováno pouze pravidlo, aby definice koncovkové množiny předcházela odkazu na ni. Je tedy možné uspořádat soubor tak, že nejprve obsahuje definice koncovkových množin a teprve po nich definice vzorů. Nebo lze vždy pro každý slovní druh uvést jak koncovkové množiny, tak definice vzorů. Ani na pořadí vzorů nezáleží, vyjma vícevzorů. Pořadí definic koncovkových množin může být též libovolné. Opět se ignorují mezery, nikoliv však prázdné řádky, které v jistých případech význam mají, jak jsme popisovali výše. Uživateli je povoleno dopisovat poznámky do souboru, pokud před ně uvede znak „#“. Počínaje tímto znakem je zbytek řádku ignorován.

5.3 Program `abin`

Hlavním úkolem programu `abin` je získání informací uložených v definičním souboru koncovkových množin a vzorů a strojovém slovníku češtiny, jejich transformace do tvaru přijatelného pro morfologický analyzátor `ajka` a následné uložení do binárních souborů. Tyto binární soubory jsou pak hlavní datovou základnou pro samotný morfologický analyzátor `ajka`.

Činnost programu `abin` lze rozdělit do několika fází, které bychom rádi v následujícím textu této části více či méně rozebrali a vysvětlili:

1. Uložení morfologické informace z definičního souboru koncovkových množin a vzorů do binárního souboru se standardní příponou `.mrf`:
   (a) Načtení definičního souboru koncovkových množin a vzorů
   (b) Uložení vytvořených datových struktur do binárního `.mrf` souboru
2. Uložení informace o kmenových základech ze strojového slovníku do binárního souboru se standardní příponou `.stm`:
   (a) Načtení strojového slovníku češtiny
   (b) Vytvoření struktury trie pro kmenové základy
   (c) Optimalizace struktury trie
   (d) Minimalizace struktury trie
   (e) Uložení vytvořených datových struktur do binárního `.stm` souboru

Proces načtení definičního souboru koncovkových množin a vzorů v sobě neobsahuje z programátorského hlediska žádné problémy, které by bylo nutné řešit nestandardními cestami. Soubor je zpracováván řádek po řádku, po prvotním předzpracování, kdy se vypustí nevýznamné mezery, popřípadě text, který je uvozen jako poznámka, jsou postupně plněny interní datové struktury, které v podstatě reflektují strukturu textového souboru, resp. strukturu záznamů pro definici koncovkové množiny a vzoru.

Struktura interních dat není příliš důležitá. Podstatný je spíše výsledek, tedy struktura dat, která jsou uložena v binárním `.mrf` souboru. Tato data by totiž měla být efektivně uložena jednak z hlediska velikosti a jednak z hlediska rychlého přístupu k nim a jejich následného využití. Vycházeli jsme od počátku z faktu, že data budou analyzátorem pouze čtena, nikoliv transformována. Proto jsme se snažili při návrhu jejich struktury zohlednit zejména rychlý přístup. Navíc jejich velikost, máme na mysli velikost morfologických dat z definičního souboru, byla nepatrná vzhledem k celkovému objemu dat. Jejich efektivnějším uložením z hlediska paměťové náročnosti bychom tedy příliš nezískali, nehledě na to, že by se tím, dle našeho názoru, výrazně zhoršila jejich přístupnost.

Museli jsme mít na paměti, jakým způsobem bude morfologický analyzátor pracovat. Ostatně jeho činnost napovídá již struktura záznamů v definičním souboru. Odtud a z algoritmu morfologické analýzy je zjevné, že podstatným prvkem je vzor, jakožto pravidlo kombinovatelnosti intersegmentů a koncovek. Ke vzoru tudíž musí být zajištěn přímý přístup. Naopak, jednotlivé intersegmenty ve vzoru nutně musí být čteny sekvenčně (chceme-li obsáhnout všechny možnosti, což je případ lemmatizace). Co se týče seznamů koncovkových množin, zjistili jsme, že v každém slovním druhu se velmi často opakují stejné sekvence koncovkových množin. To nás vedlo k tomu, že u každého intersegmentu ukládáme index do jakéhosi seznamu seznamů koncovkových množin. Tudíž přístup k těmto seznamům musí být opět přímý, stejně jako přístup k jednotlivým koncovkovým množinám. Seznamy koncovkových množin je nutné ze stejného důvodu jako u intersegmentů procházet sekvenčně.

S postfixy je situace obdobná jako s koncovkovými množinami u intersegmentů. Opět se opakují podobné sekvence. Neukládáme proto jednotlivé postfixy, ale celé jejich seznamy. Přístup k jednotlivým seznamům postfixů proto musí být přímý, jednotlivé postfixy je nutné číst sekvenčně.

Z jednotlivých intersegmentů u vzoru se přímo dostaneme k definici koncovkové množiny. I zde, z důvodu úplnosti a nutnosti vyzkoušení všech možných alternativ koncovek potřebného při přiřazování všech možných gramatických informací k analyzovanému slovnímu tvaru, stačí procházet bloky dvojic koncovek a hodnot gramatických kategorií sekvenčně se současným zapamatováním si značky tento blok uvozující. Jak značky a koncovky, tak i hodnoty gramatických kategorií se opakují. Proto opět ukládáme jen repertoár různých typů a na každém místě výskytu jen odkaz na příslušný typ.

Načítání strojového slovníku češtiny tak triviální, jako bylo prosté načítání definičního souboru koncovkových množin a vzorů, není. Provádějí se podobné akce

jako v předchozím případě, tj. odstranění nevýznamných mezer, poznámek apod. Kromě toho je potřeba expandovat slova, u nichž se vyskytuje možnost dvojího pravopisu, jak jsme popisovali výše v příkladu se slovem *gymnasium*. Expanzí získáme dvě hesla slovníku, která se liší pouze v lexikální části.

Strojový slovník je načítán po heslech. Z každého tvaru v lexikální části se nejdříve získá kmenový základ. Toto lze učinit proto, že ve chvíli načítání slovníku je již načten definiční soubor, a jsou tedy známy všechny vzory a koncovkové množiny. Proto musí načtení definičního souboru předcházet načtení slovníku. Z gramatické části hesla se získá jméno vzoru, k němuž daný slovní tvar patří. V případě vícevzorů tedy musí počet slovních tvarů právě odpovídat počtu vzorů ve vícevzoru.

Kromě kmenového základu lze z lexikální části hesla ještě získat informaci o pravopisu slova z hlediska psaní malých a velkých písmen. Po přečtení slovního tvaru je třeba zařadit tvar do některé ze čtyř kategorií, které pro tento účel uvažujeme:

- kategorie č. 0 – všechna písmena ve slově jsou malá
- kategorie č. 1 – některá písmena jsou malá, některá velká
- kategorie č. 2 – první písmeno musí být velké
- kategorie č. 3 – všechna písmena musí být velká

Pokud jde o speciální případ, kdy heslo označuje sloveso, u kterého dochází ke změně podoby kmene u negativní formy, tedy definovaná část hesla obsahuje znaky „@“ a „!“, získáme z definované části hesla ještě další informaci, a sice o nutnosti (zde je potřeba navíc odtrhnout negativní prefix *ne-* ze slovního tvaru), resp. možnosti tvoření negace.

Z gramatické části hesla je extrahována informace o vzoru, pod nějž slovní tvar spadá. V případě vícevzorů spadá první slovní tvar z definované části hesla pod první vzor vícevzoru, druhý slovní tvar pod druhý vzor ve vícevzoru atd. Dále gramatická část hesla obsahuje informaci o možnosti tvoření negativní formy, reflexivitě, dokonavosti a možnosti tvoření kompozita s číslovkou.

Interně se v průběhu načítání ukládají kmenové základy. Každý kmenový základ může spadat pod více vzorů. U kmenového základu se tedy navíc uchovávají seznamy vzorů, pod nějž kmenový základ spadá. Položka takového seznamu vzorů potom obsahuje identifikaci vzoru (jeho interní číslo), gramatickou informaci o slovním tvaru příslušejícím k danému vzoru a v případě vícevzorů navíc informaci o alternativním kmenovém základu. Gramatická informace nyní zabírá jednu slabiku (8 bitů). Nejvyšších pět bitů po řadě odpovídá možnosti tvoření negativní formy, reflexivitě, možnosti tvoření kompozita s číslovkou, dokonavosti a nutnosti tvoření negativní formy. Třetí nejnižší bit nemá dosud přiřazen žádný význam a nejnižší dva bity slouží k uložení čísla kategorie, do které slovní tvar tohoto vzoru s ohledem na psaní malých a velkých písmen náleží.

Informaci o alternativním kmenovém základu se pokusíme vyložit podrobněji. Jedná se o situaci, kdy v gramatické části hesla je uveden první vzor vícevzoru. V lexikální části je tudíž příslušný počet slovních tvarů, z každého z nich máme osamostatněný kmenový základ. Pro kmenový základ příslušný slovnímu tvaru pro první vzor vícevzoru je nutné uložit informaci o všech alternativních podobách kmenových základů. Například pro čtyřvzor se spolu s prvním kmenovým základem ukládá informace o třech alternativních kmenových základech příslušných druhému, třetímu a čtvrtému vzoru v tomto čtyřvzoru. V ostatních kmenových základech patřících ke druhému, třetímu, ... až poslednímu vzoru vícevzoru pak stačí ukládat jen informaci o alternativním kmenovém základu spadajícím pod první vzor vícevzoru.

Nyní již víme, jakou informaci o alternativních podobách kmenového základu budeme ukládat, nevíme však jak. Rozhodli jsme se pro netradiční způsob. Zjistili jsme, že ve změnách podob kmenových základů lze vypozorovat jisté pravidelnosti. Tyto pravidelnosti jsou obvykle reprezentovány samotným vzorem. Z toho plyne, že u všech slov stejného vzoru dochází k totožným změnám podob kmenového základu. Proto by nebylo efektivní z hlediska paměťové náročnosti pro daný kmenový základ uchovávat jeho alternativní podobu v celé její velikosti. Stačí, když si poznamenáme, na které pozici (písmeni) kmenového základu dojde ke změně. Z alternativního kmenového základu je pak pro uložení podstatná pouze část počínaje právě touto pozicí až do konce. Např. pro slova *Achilles*, *Achillů* s kmenovými základy *achilles*, *achill* uložíme u prvního kmenového základu pouze číslo pozice, tedy 6 (číslujeme od 0) a prázdný řetězec, neboť kmenový základ *achill* má délku 6. U kmenového základu *achill* opět uložíme číslo 6 jako pozici, na které dochází ke změně a dále zbytek alternativní podoby kmenového základu, tj. řetězec *es*.

Tento přístup nám umožní sdílet řetězce odpovídající alternativním podobám kmenových základů, které se pro různá slova často opakují. U kmenového základu pak kromě pozice změny nalezneme identifikaci alternativní podoby kmene, tj. index do tabulky řetězců vystihujících tyto alternace.

Samotný proces načtení hesla strojového slovníku lze shrnout asi takto: po získaní kmenového základu, jeho pravopisu, vzoru a gramatické informace se hledá kmenový základ v paměti. Jestliže není nalezen, je přidán jako nový kmenový základ se seznamem obsahujícím jediný vzor a jemu příslušnou gramatickou informaci, popř. alternativní kmenové základy. Je-li ovšem kmenový základ nalezen, je prohledáván jeho seznam vzorů. Mohou nastat opět dvě možnosti. Buď je vzor včetně gramatické informace a případných alternativních podob kmenového základu nalezen, pak jde o duplikaci dat a heslo se ignoruje a je vypsáno varovné hlášení na standardní chybový výstup, nebo je vzor se zmíněnými dalšími informacemi přidán do tohoto seznamu.

Pokud heslo obsahuje ještě třetí nepovinnou část, seznam prefixů, je před každý slovní tvar v definované části hesla dodán postupně každý prefix z tohoto seznamu. Zpracování prefigovaných slovních tvarů probíhá tímtéž způsobem jako v případě bez prefixů.

Po načtení celého strojového slovníku, kdy už je jasné, že do seznamů vzorů u kmenových základů nebudou přibývat nové vzory, se opět provede založení tabulky – pole těchto seznamů vzorů ovšem vzájemně různých. I v tomto případě se ukazuje, že mnoho kmenových základů patří k téže množině vzorů, tj. jejich seznamy vzorů jsou shodné. Proto u kmenového základu dále uchováváme pouze index do tabulky seznamů vzorů. Dále se provede přiřazení kódů písmenům obsaženým v kmenových základech. Přiřazení respektuje uspořádání těchto znaků v ASCII tabulce v kódování ISO8859-2 s tím, že kódy jsou přidělovány od nuly. Pro češtinu vystačíme s 41 různými kódy (0–40). Tím je vše připraveno pro následné vybudování struktury trie.

Proces budování struktury trie není příliš složitý, zvláště zúročíme-li poznatky teoretičtěji zaměřených kapitol této práce. Zopakujme, že cílem, který jsme si vytkli, je schopnost *m*-árního větvení najednou. Vzpomeneme-li na definici trie (viz str. 24), jsou jednotlivé uzly *m*-árními vektory. V implementaci ctíme tuto definici, a proto každý uzel tvoří bitová mapa. Jedná se o posloupnost 6 slabik (48 bitů). Jednotlivým bitům odpovídají kódy příslušných písmen (uvažujeme pouze malá písmena). V současné implementaci odpovídá nejvyššímu bitu nejnižší kód písmene, tedy kód písmene *a*. Kód písmene tedy říká, který bit „zleva“ je onomu písmeni přiřazen, číslujeme-li bity v bitmapě „zleva“ od nuly. Je-li příslušný bit nastaven na hodnotu 1, znamená to, že uzel obsahující tuto bitmapu má syna odpovídajícího písmenu, které zmíněný bit zastupuje.

Zjištění, zda daný uzel má syna, jenž odpovídá jistému písmeni, je tudíž velmi jednoduché a efektivní. Bitmapa daného uzlu je v jediné strojové instrukci podrobena testu na hodnotu určitého bitu. Nulový výsledek říká, že hledaný syn neexistuje, nenulový výsledek znamená, že tento syn existuje. V situaci, kdy syn existuje, se ocitáme na poloviční cestě k dosažení *m*-árního větvení najednou. Zbývá už jen rychlý přechod na daného syna.

Vzpomeňme si na uložení stromu do šířky (viz str. 21, obrázek 3.8). Díky tomuto způsobu uložení, kdy všichni bratři jsou vždy uloženi bezprostředně vedle sebe, stačí k okamžitému přechodu na hledaný uzel pouze informace obsažená v bitmapě. Položka `INFO` nyní obsahuje bitmapu, položka `FSON` ukazatel na nejstaršího syna. Položka `LAST` původně indikující nejmladšího syna (bratra) se nyní díky bitmapě stává zbytečnou.

Co vše lze z bitmapy vyčíst? Počet bitů v bitmapě nastavených na hodnotu 1 odpovídá počtu synů daného uzlu. Z tohoto důvodu je zbytečná položka `LAST`. Existuje-li syn daného uzlu, jeho adresa se spočítá jednoduše tak, že k adrese, na kterou ukazuje `FSON` se přičte velikost paměti, kterou zabírají starší bratři hledaného syna. Starší bratři totiž našeho syna bezprostředně předcházejí. Velikost paměti, kterou zabírají, je dána jejich počtem vynásobeným velikostí uzlu. Z tohoto vyplývá, že *všechny uzly musí mít tutéž velikost*. Počet starších bratrů uzlu je roven počtu jedničkových bitů předcházejících v bitmapě otce bit příslušející tomuto uzlu.

Ve skutečnosti se ukazuje, že též bitmapy v jednotlivých uzlech se často opakují. V každém uzlu proto položka `INFO` neobsahuje vlastní bitmapu, ale index

do jakési tabulky bitmap, kde jsou pochopitelně uloženy jen vzájemně různé typy bitmap. Jedním z možných vysvětlení je kupříkladu triviální fakt, že po souhlásce obvykle následuje samohláska. Proto bitmapy uzlů reprezentujících samohlásky, respektive souhlásky, budou podobné. Situace je samozřejmě mnohem složitější. Po samohlásce může následovat samohláska, např. ve slově *auto*, po souhlásce souhláska, jako ve slově *proč*. Na druhou stranu ovšem nelze popřít, že výskyt bigramu *aa*, *dd* nebo *pp* je v českém slově naprosto ojedinělý. Tuto situaci poněkud komplikují zkratky, u nichž se pochopitelně takto exotické skupiny písmen objevují nezřídka. I přesto jisté souvislosti mezi písmeny, která mohou stát vedle sebe, nepochybně existují.

Při samotném procesu budování trie jsme vycházeli z faktu, že slovník bude používán jako statický. Nebylo tudíž důvodu budovat trie inkrementálně. Inkrementální způsob budování trie je paměťově náročnější než neikrementální způsob. Na druhé straně je rychlost budování trie v inkrementálním případě mnohem vyšší. Jestliže předem „odsoudíme“ slovník ke statickému použití, předpokládáme, že jej jednorázově vytvoříme a dále do něj nebudeme zasahovat. Časová ztráta způsobená výběrem neinkrementální metody pro jeho vytvoření v tomto ohledu nehraje podstatnější roli.

Podrobně popisovat budování trie nemá, dle našeho názoru, valného smyslu. Ostatně samotná struktura trie, resp. uložení klíčů v ní, napovídá, jakým způsobem proces jejího budování probíhá. V našem případě jsou klíči kmenové základy českých slov převedené na malá písmena. Podstatou vyhledávací struktury trie je sdílení společných prefixů. Na základě této vlastnosti je také struktura budována.

Seznam všech kmenových základů se nejprve setřídí podle abecedy. Na počátku se za společný prefix prohlásí prázdný řetězec. Prochází se celý seznam kmenových základů a poznamenává se změna prvního písmene. Po projití seznamu se tak zjistí všechna možná první písmena, tedy větvení na úrovni kořene. Toto větvení je reprezentováno bitmapou, v níž jsou nastaveny příslušné bity. V další iteraci se zvýší délka prefixu na 1 znak. Vezme se tedy první písmeno prvního kmenového základu a to se prohlásí za společný prefix. Opět se projde seznam kmenových základů s tím, že se nyní testuje druhé písmeno.

Speciálním případem písmene je znak indikující konec slova. Tomuto znaku neodpovídá žádný bit v bitmapě, přesto je potřeba poznamenat, že aktuální uzel je koncový, tedy jeho prvním synem je list. Interně proto s každým uzlem uchováváme i jeho typ. Do listu umisťujeme index do tabulky seznamů vzorů, který určuje seznam vzorů, pod které daný kmenový základ spadá.

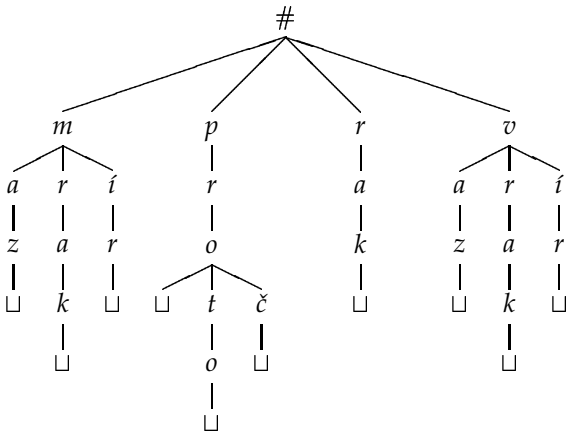
Při průchodu mohou nyní nastat dvě situace. Buď se změní druhé písmeno kmenového základu, tj. písmeno následující společnému prefixu, nebo se změní společný prefix. V prvním případě se jedná pouze o větvení v tomtéž uzlu (poslední uzel společného prefixu) a při každé změně se pouze nastaví příslušný bit v bitmapě na hodnotu 1. Ve druhém případě, kdy prefix kmenového základu neodpovídá dosud společnému prefixu, je potřeba uložit právě rozpracovanou bitmapu (uzel) a za společný prefix prohlásit prefix kmenového základu, ve kterém došlo ke změně.

Takto se postupuje iterativně dále, v každé iteraci se zvýší délka společného prefixu o jeden znak. Jakmile jsou všechny kmenové základy kratší než prohlášený společný prefix, struktura trie je vybudována.

Při průchodu kmenovými základy se neustále udržují dvě podstatná čísla. Je to počet synů všech starších bratrů uzlu, jehož bitmapa se aktuálně zpracovává a počet jeho mladších bratrů. Součet těchto čísel vynásobený velikostí uzlu totiž udává vzdálenost mezi aktuálním uzlem a jeho prvním synem. Tento součet uložíme do položky `FSON`, která společně s indexem do tabulky bitmap v položce `INFO` tvoří uzel.

Takovýto opakovaný průchod přes všechny kmenové základy zajistí zároveň i to, že uzly jsou uloženy ve správném pořadí, tedy v pořadí odpovídajícím průchodu stromem do šířky. Po vybudování obsahuje trie dva typy vnitřních uzlů, tj. *koncové*, v nichž končí kmenový základ, který současně může, ale nemusí být prefixem jiných kmenových základů a *nekoncové*, v nichž nelze jinak, než pokračovat dále. Kromě vnitřních uzlů obsahuje trie *listy*, v nichž je uložena informace o vzorech kmenových základů (index do tabulky seznamů vzorů). Například obrázek 5.1 zobrazuje trie s 26 vnitřními uzly a 10 listy. Deset z vnitřních uzlů je koncových.

Samozřejmě, že v implementaci je potřeba zařídit speciální případy na začátku a konci seznamu kmenových základů, nulování bitmapy na začátku každé iterace a podobně. Popisování těchto podrobností by však pouze znepřehlednilo základní myšlenku, s jakou se trie buduje, která je nyní, doufejme, jasná.



Obrázek 5.1: Příklad trie kmenových základů

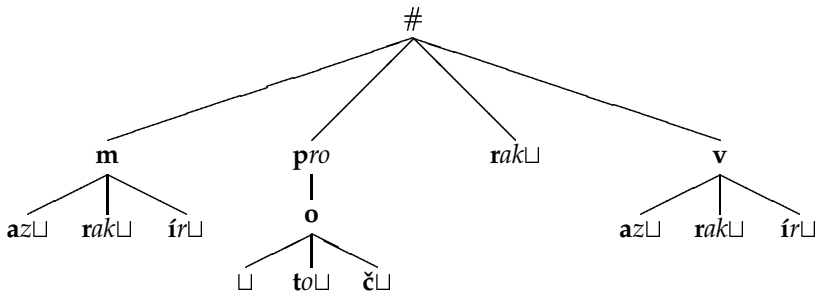
Poté, co je struktura trie vybudována, nastává proces její optimalizace. Pod optimalizací rozumíme transformaci, jejímž úkolem je zejména zmenšit paměťové nároky, tedy snížit počet uzlů trie. Hlavním problémem při uložení do šířky je hloubka stromu. Viděli jsme (viz tabulka 3.1), že pro uložení do šířky jsou vhod-

nější „široké“ stromy. Cílem optimalizace je tedy zmenšení hloubky trie.

K největší neefektivitě dochází v uzlech, které mají právě jednoho syna, neboť při uložení do šířky vždy v položce `FSON` uchováváme ukazatel na prvního syna, byť by byl jediným. Navíc bitmapa takového uzlu obsahuje právě jednu jedničku, ostatní bity jsou nulové (samozřejmě, že v případě, kdy je jediným následníkem uzlu list, je bitmapa nulová). Pokud po sobě následuje takovýchto uzlů více, mluvíme pracovně o *cestě*. Právě cesty činí uložení do šířky velmi neefektivním. Rozhodli jsme se proto cesty eliminovat.

Podívejme se na obrázek 5.1. Například třetí syn kořene, uzel *r*, má jediného syna *a*, ten má jediného syna *k* a konečně tento má jediného syna ␣. Jinými slovy, jakmile kmenový základ začíná na písmeno *r*, pak to může být jedině kmenový základ *rak* a žádný jiný. Řečeno ještě jiným způsobem, již po prvním písmeni *r* je jasné, že následující jediná správná sekvence písmen je *ak*. Podobně, je-li prvním písmenem *p*, jediným možným následným řetězcem je *ro* a dále buď konec slova, nebo písmeno *t*, nebo *č*. Opět, jakmile je třetím písmenem *t*, musí čtvrtým (a zároveň posledním) být *o*.

Cílem eliminace cest je uložení informace o tom, jaká jediná správná sekvence písmen může následovat, do prvních uzlů cest a vypuštění uzlů, které cestu tvoří. V případě, že cesta končí listem, pak navíc požadujeme uložení informace z tohoto listu do prvního uzlu cesty. Námi uvažovaný strom (viz obrázek 5.1) po eliminaci cest znázorňuje obrázek 5.2 (příslušná cesta je naznačena netučným písmem v uzlu).



Obrázek 5.2: Eliminace cest

Cesty se jako obvykle opakují. Opět jsou tedy uloženy separátně a sdíleny. V každém prvním uzlu cesty neuchováváme přímo příslušné následující řetězce písmen, ale pouze indexy určující typ řetězce uloženého v jakési tabulce cest. V této tabulce jsou cesty uloženy jako klasické řetězce.

Samotný algoritmus nalezení cest v trie není příliš složitý. Co je na něm důležité, je fakt, že zároveň při hledání cest nastavuje informaci v uzlech na patřičné hodnoty. Tak vznikne v podstatě šest typů uzlů, které sice musí mít shodnou velikost, v našem případě 4 slabiky (32 bitů), ale informace v nich uložená má odlišný

význam. Transformaci uzlů při eliminaci cest i jednotlivé typy uzlů s jejich strukturou přehledně vyjadřuje tabulka 5.3.

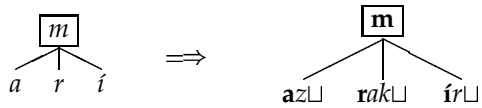
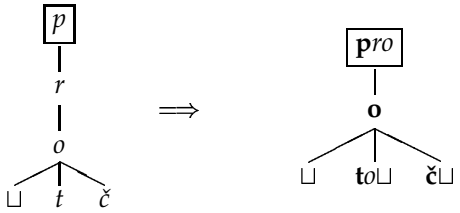
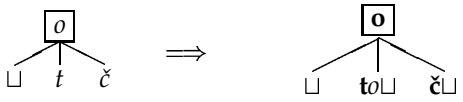
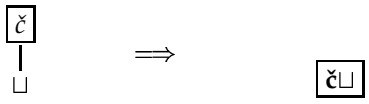
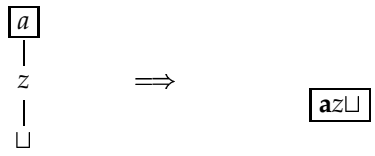
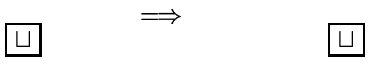
V prvním sloupci tabulky je uveden typ uzlu (transformace). Druhý sloupec tabulky tuto transformaci znázorňuje. Vlevo je zobrazen v rámečku první uzel cesty tak, jak byl v původním stromu (viz obrázek 5.1), vpravo je zakreslena situace po eliminaci cesty (viz obrázek 5.2). Třetí sloupec tabulky obsahuje popis struktury uzlu po transformaci. V horním řádku je uvedena velikost jednotlivých položek uzlu v bitech, pod ním je naznačen obsah příslušné položky.

Vidíme, že první položka uzlu zabírá dva bity. Hodnoty těchto bitů mají následující význam. Vyšší bit udává, zda uzel je koncovým, tzn. je posledním vnitřním uzlem, popř. listem ve větvi určující nějaký kmenový základ. Druhý nejvyšší bit indikuje, zda uzel je prvním uzlem cesty. Hodnoty těchto dvou bitů určují obsah druhé a třetí položky uzlu. V první položce může být uložen buď index do tabulky bitmap `BMAP`, nebo index do tabulky cest `PATH`, případně může být tato položka nulová (všechny bity mají hodnotu 0) `NULL`. Druhá položka obsahuje buď vzdálenost `FSON` mezi daným uzlem a jeho nejstarším synem, což je počet uzlů mezi nimi, nebo index do tabulky seznamů vzorů `LPAR`.

Jen pro srovnání uveďme, že původní strom obsahoval 36 uzlů, strom po eliminaci cest pouze 15. Došlo tedy ke značnému snížení počtu uzlů. Dále se výrazně zmenšila hloubka stromu. Při podrobnější analýze bychom zjistili, že počet $*$-uzlů při uložení tohoto stromu do hloubky tak, jak ukazuje obrázek 3.6 je 9, zatímco při uložení do šířky (viz obrázek 3.10) pouze 5. To znamená, že uložení tohoto stromu způsobem odpovídajícím průchodu do hloubky se stává oproti uložení do šířky velmi neefektivním. Tento fakt pouze potvrzuje správnost našeho výběru. Nehledě na to, že strom bez cest nyní takřka přesně odpovídá tabulkovému zápisu trie, jenž využívá vyhledávácí algoritmus (viz tabulka 3.2 a str. 24).

Nyní se pokusíme vysvětlit princip algoritmu pro eliminaci cest a nastavení správné informace v uzlech. Podstatou algoritmu je průchod trie do hloubky (viz str. 18). Algoritmus se opírá zejména o tři proměnné. Proměnnou `path`, v níž je uložena aktuální cesta (řetězec písmen), proměnnou `first`, což je ukazatel identifikující první uzel aktuální cesty a konečně proměnnou `succ`, která obsahuje vzdálenost (číslo udávající počet mezilehlých uzlů) `first` od uzlu právě zpracovávaného. Na počátku jsou všechny proměnné nulové. Při výběru uzlu ze zásobníku mohou nastat čtyři případy. Buď má uzel jediného syna, kterým je list, nebo má právě dva syny, z nichž jeden je list, nebo má právě jednoho syna, který není listem, a nebo má více než jednoho syna a přitom se nejedná o druhou možnost, tj má-li právě dva syny, pak mezi nimi není list.

V prvním případě závisí další akce na tom, zda proměnná `path` je prázdná, či nikoliv. Pokud není prázdná, pak list je posledním uzlem cesty, cesta v něm končí. Uložíme cestu do tabulky cest, pokud tam již není. Typ uzlu `first` nastavíme na hodnotu 3, jeho první položku naplníme indexem právě ukončené cesty v tabulce cest a druhou položku indexem do tabulky seznamů vzorů převzatým z listu (syna). Proměnnou `path` vynulujeme. V případě, že cesta `path` je prázdná, nastavíme typ právě vybraného uzlu na 2 a jeho druhou položku na index do

Typ	Transformace uzlu při eliminaci cest	Struktura uzlu
1		2 b | 14 b | 16 b 00 | BMAP | FSON
2		2 b | 14 b | 16 b 01 | PATH | FSON
3		2 b | 14 b | 16 b 10 | BMAP | FSON
4		2 b | 14 b | 16 b 10 | NULL | LPAR
5		2 b | 14 b | 16 b 11 | PATH | LPAR
6		2 b | 14 b | 16 b 00 | NULL | LPAR

Tabulka 5.3: Transformace uzlů a jejich struktura

tabulky seznamů vzorů převzatý z listu (syna). Nakonec, bez ohledu na stav proměnné `path`, vynulujeme všechny tři proměnné `path`, `first` i `succ`.

Pokud má uzel právě dva následníky, z nichž jeden je list, pak v případě, že proměnná `path` má alespoň dva znaky, uložíme cestu a nastavíme položky uzlu `first` takto: typ bude roven 1, první položka bude obsahovat index do tabulky cest a druhá položka hodnotu proměnné `succ`. Bez ohledu na délku proměnné `path` vynulujeme všechny tři proměnné.

Jestliže uzel má právě jednoho syna, který není listem, přidá se písmeno, kterému vybraný uzel odpovídá, na konec proměnné `path` a zvýší se hodnota `succ` o vzdálenost mezi tímto uzlem a jeho jediným synem. Pokud má nyní proměnná `path` délku 1, jedná se o první uzel na cestě, proto se nastaví `first` tak, aby ukazovala na tento vybraný uzel.

V posledním případě, kdy má uzel více než jednoho syna, se za předpokladu délky `path` vyšší než 1 uloží cesta a nastaví se typ uzlu `first` na 1, jeho první položka se naplní indexem do tabulky cest a druhá hodnotu proměnné `succ`. Taktéž se za tohoto předpokladu vynulují všechny tři proměnné `path`, `first` a `succ`.

I po eliminaci cest nás napadne přirozená otázka, zda nelze počet uzlů trie snížit ještě více. Při pohledu na obrázek 5.2 zjistíme, že v trie existují „podobné“ uzly. Například třikrát se v trie vyskytuje uzel označený **r***ak*␣. Podobně i uzly **m** a **v** jsou si podobné v tom smyslu, že mají stejné následníky. Docházíme k závěru, že kdyby např. na místě uzlu **m** byl uzel, jenž by nějakým způsobem naznačil, že celý podstrom pod ním je ekvivalentní podstromu pod uzlem **v**, mohli bychom z trie celý podstrom pod uzlem **m** vypustit, neboť jeho věrná kopie se nachází pod uzlem **v**. Tímto vypuštěním uzlů z podstromu se opět sníží paměťové nároky trie.

Otázkou je, jak tyto „podobné“ uzly spolehlivě a přitom efektivně najít. Odpověď je jednoduchá. Na trie lze pohlížet jako na deterministický konečný automat, vzájemnou korespondenci jsme vyložili v části 4.7. Pro DKA již máme k dispozici nástroj, pomocí nějž lze „podobné“ uzly, v terminologii DKA ekvivalentní stavy, nalézt. Je jím minimalizační algoritmus (viz str. 36).

Minimalizační algoritmus je implementován takřka beze zbytku stejně, jako jsme jej uvedli v části 4.4. Pro ekvivalenci stavů je ovšem potřeba podmínku v její definici zesílit. Nestačí pouze, aby na totéž slovo bylo možné buď z obou stavů dospět do stavu koncového, nebo z obou dospět do stavu nekoncového, ale navíc je v případě, kdy z obou stavů dospějeme do stavu koncového, nutná rovnost indexů, které identifikují seznam vzorů, pod které daný kmenový základ spadá. Nestačí tedy pouze akceptovat kmenový základ, ale také je nutné jej akceptovat se stejnou gramatickou informací.

Počáteční rozklad stavů je tedy poněkud jemnější. Uzly se rozdělí do tříd podle typu tak, že uzly typu 2 a 3 spadnou do třídy, kterou určuje jejich index do tabulky seznamů vzorů. Co záznam v takové tabulce, to jedna třída. Uzly typu 0 spadnou do další třídy, uzly typu 1 do jiné. Na počátku jsou tedy uzly rozděleny do počtu tříd, který odpovídá počtu různých seznamů vzorů plus 2. Dále algoritmus pracuje tak, jak bylo dříve vysvětleno, tj. každému uzlu se vždy přiřadí aktuální třída, do které spadá.

Spolu s každou třídou je uchováván její reprezentant a počet jejích prvků. Volba reprezentanta je velmi důležitá. Zvolili jsme ze všech uzlů třídy jako reprezentanta takový uzel, který je v uložení do šířky nejvíce vzdálen kořeni. Tím jsme se vyhnuli problémům, které mohou nastat při jiné volbě reprezentanta. Musíme si uvědomit, že reprezentant je tím uzlem, na který budou v budoucnu ukazovat všechny ostatní uzly třídy.

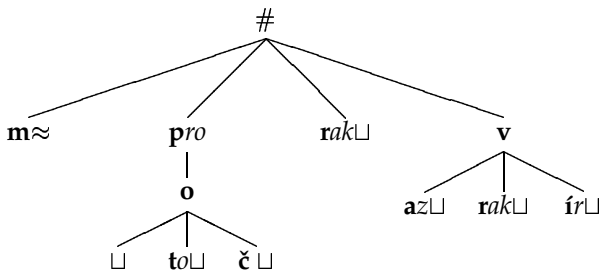
Předpokládejme, že jistá třída má n prvků setříděných vzestupně podle vzdálenosti od kořene. Je-li n sudé, bude reprezentantem $(n/2 + 1)$-ní uzel, jinak jím bude $\lceil n/2 \rceil$-tý uzel. Kdybychom přijali toto pravidlo, podívejme se na obrázek 5.2, jaké nastanou problémy. Vidíme, že uzly **r***ak*␣ spadají do téže třídy, přičemž jejich pořadí je takové, že prvním je syn kořene, druhým syn uzlu **m** a třetím syn uzlu **v**. Tedy jako reprezentant třídy je vybrán druhý uzel, syn uzlu **m**. Ostatní uzly třídy, zejména syn kořene, budou tedy na něj ukazovat. Nicméně je také vidět, že uzly **m** a **v** patří též do stejné třídy (v tomto pořadí), jsou ekvivalentní. Podle pravidla je za reprezentanta vybrán uzel **v**. Tedy uzel **m** bude ukazovat na uzel **v** s tím, že podstrom pod **m** se vypustí. Jenže v tomto vypuštěném podstromu se nachází uzel **r***ak*␣, na nějž ukazuje mj. uzel **r***ak*␣ jakožto syn kořene. Ukazatel pak ukazuje nesprávně a celý strom se hroutí. Proto je potřeba volit reprezentanta obezřetně. Může jím být buď nejbližší nebo nejvzdálenější uzel ke kořeni v dané třídě. Zvolili jsme druhou alternativu pouze z důvodu, aby ukazatele nebyly záporné.

Po skončení algoritmu má každý uzel přiřazenu třídu, do které spadá; u každé třídy je známa její mohutnost a reprezentant. Nezbývá, než označit ekvivalentní uzly a uložit do nich ukazatel na reprezentanta, s nímž jsou ekvivalentní. Ekvivalentními uzly jsou právě ty prvky tříd (s mohutností větší než 1), které jsou různé od reprezentanta dané třídy. Ani ekvivalentní uzly nesmí být větší než 4 slabiky. Ukazatele na reprezentanty jsou však, vzhledem k jejich volbě, příliš dlouhé, než aby na jejich uložení stačily 2 slabiky, což byla velikost ukazatele v neekvivalentním uzlu. Proto je struktura uzlu ekvivalentního poněkud odlišná. Má pouze dvě položky. První zabírá 12 bitů, které jsou vždy všechny nastaveny na hodnotu 1. Tím se odliší od všech ostatních uzlů všech typů za předpokladu, že počet různých cest nebude větší než 16368. Druhá položka zabírá zbylých 20 bitů a slouží pro uložení ukazatele na reprezentanta. Opět ukazatelem rozumíme počet uzlů, které příslušnou dvojici uzlů oddělují.

Ze struktury ekvivalentního uzlu je patrné, že nemá smysl za ekvivalentní prohlásit uzly typu 3 a 2 s nulovou bitmapou. Tyto uzly jsou totiž listy a tedy nejsou kořenem žádného podstromu, který by se dal vypustit. Proto např. uzel **r***ak*␣ jakožto syn kořene není označen za ekvivalentní uzel. Obrázek 5.3 znázorňuje strom trie, jehož optimalizovanou verzi ukazuje obrázek 5.2, po procesu minimalizace. Ekvivalentní uzel je označen symbolem $\approx$. Počet uzlů trie se tak dále snížil o tři. Celkem klesl počet uzlů v trie z původních 36 na třetinu, tj. 12 uzlů.

Strom trie podrobený optimalizaci a minimalizaci je připravený pro uložení do binárního `.stm` souboru spolu s dalšími informacemi.

Na závěr této části ještě dodáme poněkud praktičtější informace o programu `abin`. Zejména to, s jakými parametry se spouští apod. Výčet všech současných



Obrázek 5.3: Trie po minimalizaci

možností programu `abin` je kompaktně vyjádřen následujícím řádkem, který znázorňuje možný obsah příkazového řádku, chceme-li program spustit.

```
abin [-h] [-p parf | -d dicf | -m mrff | -s stmf]
```

Po zadání volby `-h` se vypíše krátká informace o správném použití příkazu. Jinak je činnost programu taková, že ze souboru `parf`, který je implicitně nastaven na `ajka.par` vytvoří binární soubor `mrff`, implicitně nastavený na `ajka.mrf` a dále ze souboru `dicf`, kterým je implicitně standardní vstup, vytvoří binární soubor `stmf`, implicitně nastavený na `ajka.stm`. Všechny volby jsou nepovinné, při jejich absenci se použije příslušná implicitní hodnota. Při nesprávné syntaxi se vypíše nápověda.

5.4 Morfologický analyzátor češtiny `ajka`

Implementace morfologického analyzátoru `jazyka` zcela vychází z algoritmu pro morfologickou analýzu, který jsme popsali v části 2.4. Poněvadž jsme zvolili slovníkový přístup, je značná část morfologické informace obsažena v datech.

Hlavními datovými zdroji pro morfologický analyzátor jsou dva binární soubory. Jeden obsahuje definice koncovkových množin a vzorů, tedy morfologickou informaci (implicitně pojmenovaný `ajka.mrf`), druhý pak je binární podobou vlastního strojového slovníku a obsahuje kmenové základy českých slov (implicitně se nazývá `ajka.stm`).

První akcí, která se provede po spuštění programu, je načtení dat uložených ve zmíněných binárních souborech. Protože předpokládáme, že program bude spouštěn opakovaně v cyklech zvláště při zpracování více textových souborů, zvolili jsme se z důvodu rychlejšího startu analyzátoru takové uložení dat, aby analyzátoru stačilo pouze jejich načtení. Poté je činnost analyzátoru dána algoritmem z úvodu práce. Navíc jsme implementovali ještě některé další možnosti morfologické analýzy, které jsme na počátku explicitně neuvedli.

Pokusili jsme se implementovat analýzu kompozit dvojího druhu. Jednak jde o kompozita více adjektiv, např. *československý*, *kanadskoamerický*, *českomoravsko-*

slezský apod. A dále o kompozita, ve kterých první část tvoří číslovka a druhou adjektivum. Sem patří např. tyto slovní tvary: *dvěstěpadesátičlenný*, *stokorunový*, *triadvacetilitrový*.

Při analýze kompozit prvního typu se navíc pouze ověřuje, zda analyzovaný kandidát na kmenový základ slova není adjektivem speciálního tvaru (tj. končící na *-sko*, *-cko* apod.). Koncovkové množiny příslušné takovýmto intersegmentům jsou v definičním souboru koncovkových množin a vzorů označeny speciální gramatickou značkou. Pokud se zjistí, že jde o takové adjektivum, analýza pokračuje jakoby od začátku, ovšem tentokráte jen na zbývající části slova. Pokud analýza zbytku slova proběhne v pořádku, tzn., že je analyzováno adjektivum 1. stupně, kompozitum je akceptováno s tím, že za základní tvar se prohlásí lemma nejvýznamnějšího (posledního) adjektiva nesoucího též gramatický význam.

Analýza kompozit druhého typu předpokládá dvě části. Jednak musí úspěšně proběhnout analýza číslovky. Experimentálně jsme se do analyzátoru pokusili zařadit jakousi gramatiku pro složené číslovky. Je-li číslovka v souladu s gramatikou, tj. je v příslušném tvaru, analýza pokračuje dále obdobným způsobem jako u kompozit prvního typu. Zbytkem slova opět musí být adjektivum 1. stupně, které ovšem je schopno přijímat číselné prefixy. Identifikace takových adjektiv je zajištěna uvedením speciálního symbolu v gramatické části hesla příslušného kmenového základu ve slovníku. Také koncovkové množiny číslovek patřičných tvarů jsou označeny speciální gramatickou značkou v definičním souboru. Základní tvar i gramatické kategorie řídí opět adjektivum.

Pro uživatele bude jistě přínosnější popis možností morfologického analyzátoru a způsobů, jakými lze jeho chování ovlivnit. Snažili jsme se zachovat podstatné rysy používaného lemmatizátoru `LEMMA` [Ševe–95]. Poněvadž byl za pomocí tohoto nástroje označkován jeden z korpusů na fakultě a rádi bychom v započaté práci značkování i nadále pokračovali, bylo nezbytné zachovat formát gramatických značek, které `LEMMA` používá (viz příloha A). Protože značkování probíhá automaticky, ani výstup analyzátoru se z důvodu snadnějšího zpracování výsledků příliš od výstupu programu `LEMMA` neliší. Také v interaktivním režimu jsme v podstatě zachovali „prostředí“, na které je uživatel zvyklý z analyzátoru `LEMMA`.

Na následujícím řádku uvádíme v kompaktní formě možnosti, jak může vypadat příkazový řádek pro spuštění programu `ajka`.

```
ajka [-h] [-b | -a] [-l lopt] [-m mrff | -s stmf | txtf]
```

Po zadání volby `-h` se vypíše krátká informace o správném použití programu. Jinak program pracuje ve dvou režimech v závislosti na tom, zda byl zadán název textového souboru `txtf`, jehož obsah má být analyzován. V případě, že jméno souboru zadáno bylo, mluvíme o dávkovém režimu a předpokládáme, že zadaný soubor je textový, v němž se na každém řádku vyskytuje právě jedno slovo. Jinak mluvíme o režimu interaktivním.

Práce s programem v interaktivním režimu je velmi jednoduchá. Uživatel má možnost napsat slovo, které má být analyzováno. V závislosti na zvoleném módu (uvedením buď `-b`, nebo `-a` na příkazovém řádku) se vypíše výsledek. Volba `-b`

znamená, že výstup bude v úspornější formě. Volba `-a` znamená, že ke každému základnímu tvaru se navíc budou generovat i všechny možné odvozené slovní tvary. Práce s analyzátorem v interaktivním režimu se ukončuje zadáním speciálního znaku „#“.

Volba `-l` slouží k zadání řetězce čísel `lopt`, který ovlivňuje volbu základního tvaru u různých druhů slov. Názorně jednotlivé možnosti vystihuje tabulka 5.4.

`lopt`	**základní tvar při dané volbě**	**základní tvar jinak**
0	žádný speciální tvar není vyžadován	totéž jako při `lopt`=„norm“
1	včetně prefixu *ne-*	neprefigovaný
2	včetně postfixu	bez postfixu
3	posesivum pro posesiva	substantivum
4	číslovka téhož druhu pro číslovky	číslovka základní
5	deverbativum pro deverbativa	sloveso
6	adverbium pro adv. generovaná z adj.	adjektivum
mini	totéž jako při `lopt`=„12“	totéž jako při `lopt`=„norm“
norm	totéž jako při `lopt`=„23“	totéž jako při `lopt`=„norm“
maxi	totéž jako při `lopt`=„23456“	totéž jako při `lopt`=„norm“

Tabulka 5.4: Různé volby lemmatizace

Konečně, volbami `-m` a `-s` lze specifikovat soubory, které se mají při analýze použít jako zdroj morfologické informace (soubor `mrff`) a jako slovník kmenových základů (soubor `stmf`).

Všechny volby jsou nepovinné, implicitní nastavení je dáno následovně. Pro oba režimy je nastaven normální mód odpovídající módu v interaktivním režimu bez uvedení jakýchkoliv voleb. Hodnota `lopt` je rovna „norm“. Soubory se hledají v pracovním adresáři pod názvy `ajka.mrf` a `ajka.stm`. Přepnutí režimu mezi interaktivním a dávkovým je dáno použitím jména analyzovaného souboru `txtf`. Příklady použití a formát výstupu v jednotlivých případech uvádíme v příloze B.

5.5 Automatické určování vzorů českých slov

Při zpracovávání definičního souboru koncovkových množin a vzorů jsme zjistili, že data v něm uložená by bylo možné s výhodou použít při určování vzorů českých slov. Nemáme samozřejmě na mysli vzory klasicky uváděné, ale vzory, které byly navrženy v práci [Osol–96] a jsou jednou ze součástí morfologického analyzátoru.

Běžný uživatel totiž není schopen se v takovém množství vzorů orientovat. V případě, kdy o analyzovaném slově bezpečně ví, že je správné, a přesto bylo zamítnuto analyzátorem z důvodu, že analyzátor slovo nezná, cítíme potřebu přidání nového slova do slovníku. Ovšem pro tento úkon je nezbytné znát, ke kterému vzoru slovo patří, popřípadě vědět, že slovo se skloňuje (časuje) stejným

způsobem jako některé jiné slovo, které již analyzátor zná. To nás motivovalo ke snaze alespoň se pokusit navrhnout experimentální nástroj, který by uživateli proces určení vzoru slova co nejvíce usnadnil.

Vzor, tak jak byl definován, reprezentuje množinu všech možných kombinací intersegmentů a koncovek. Dále budeme pro tyto kombinace používat pracovního pojmenování *konec slova*. V tomto smyslu tedy vzor identifikuje právě korektní konce slov, které lze připojit za kmenový základ slova k tomuto vzoru příslušející.

Pro určení vzoru slova je nezbytná jeho segmentace na kmenový základ a konec slova. Víme, že kmenový základ je z pohledu jediného vzoru neměnný, nicméně v případě, kdy jde o některý ze vzorů vícevzoru, je třeba s alternací kmenového základu počítat. Zde nastávají jisté komplikace, proto jsme zatím od plně autotmatického nástroje pro určování vzorů ustoupili. Východisko vidíme v nástroji, který by byl v interakci s uživatelem. Uživatel by měl podle nás hrát úlohu verifikátora, který jednotlivé slovní tvary navržené pro dané gramatické kategorie počítačem podle své jazykové kompetence buď označí za správné, nebo je odmítne.

Samozřejmě, že obecně je otázka přiřazení vzoru k zadanému slovu velmi nesnadná. Zejména je nutné přistoupit na fakt, že určení vzoru je závislé na druhu slova. Už jen proto, že u různých slovních druhů je potřeba určit obecně různé gramatické kategorie. Od začátku jsme se pokusili situaci slovních druhů zjednodušit. Uvědomme si, že z deseti základních slovních druhů je polovina neohebných. Systém vzorů pro tyto slovní druhy je tedy dostatečně chudý na to, aby orientace v nich nebyla příliš obtížná. Neohebné slovní druhy proto neuvažujeme.

Z ohebných slovních druhů jsou zájmena a číslovky relativně uzavřenou množinou slovních tvarů. Při dostatečně reprezentativním slovníku by se problém neznalosti kmenového základu neměl objevit. Předpokládáme, že v budoucnu budeme mít takový úplný slovník k dispozici. Nová slova k číslovkám a zájmenům by tedy do takového slovníku přibývat neměla.

Kde skutečně má význam uvažovat o poměrně častém výskytu nových slov, jsou podstatná jména, přídavná jména a slovesa. Na tyto slovní druhy se při určování vzorů zaměřujeme. V současné době sice uvažujeme, že všechny vzory mají potenciálně stejnou šanci být vzorem předloženého slova, ale do budoucna by zřejmě bylo vhodné některé vyloženě archaické vzory neuvažovat. Domníváme se, že k takovým vzorům nová slova přibývat nebudou.

Jestliže má jít o nástroj, se kterým bude uživatel komunikovat, záleží velmi na uživatelském rozhraní. Jelikož v současnosti existuje programový nástroj, který je podobným způsobem orientován (jedná se o program `ced` Mgr. Marka Vebera), zahrnuli jsme nástroj pro určování vzorů do tohoto systému a získali tak jednotné a příjemné uživatelské prostředí. Diskutovali jsme možnost co nejjednoduššího algoritmu a rozhodli se provést experiment se zmíněnými konci slov.

Základní myšlenka je asi následující. Pro jednoduchost příkladu zvolme substantiva. Interně je u každého substantivního vzoru uloženo 14 možných konců slov. Sedm jich je pro jednotlivé pády singuláru, druhá polovina pro plurál. V současné době počítáme i s alternativami, takže jde v podstatě o čtrnáct záznamů, které obsahují minimálně jeden konec slova. Např. 3. pád singuláru substantiva

pán má dvě možné podoby *pán-u* a *pán-ovi*. Uživatel zadá alespoň jeden slovní tvar substantiva, o němž bezpečně ví, že je správný. Pro substantiva tedy vyplní některé ze 14 políček tabulky, která odpovídají postupně jednotlivým pádům jednotného a množného čísla. Samozřejmě, že může vyplnit i více políček, čím více, tím jednodušší bude analýza. Může též specifikovat, jakého rodu právě předložené slovo je.

Po zafixování jednoho tvaru proběhne „*pattern matching*" . Není to nic jiného, než že se prochází všechny substantivní vzory (při zadaném rodu jen ty, které podmínku na rod splňují). Ty vzory, u nichž dojde ke shodě konce slova na místě, které odpovídá příslušnému pádu zafixovaného slovního tvaru s postfixem (posledními písmeny) tohoto tvaru, se označí za kandidáty. Počateční úsek slova po odtržení postfixu, u nějž došlo ke shodě s koncem slova ve vzoru, se prohlásí za kmenový základ.

Spolu s množinou kandidátů na vzory máme i množiny jim příslušejících konců slov. Nabídneme proto uživateli ostatní tvary, jako by se jednalo o slovo skloňované podle prvního z kandidátů na vzor. Uživatel má v tuto chvíli před sebou tabulku se 14 políčky; některé z nich sám vyplnil, obsahuje tedy korektní tvar pro daný pád, ostatní políčka byla doplněna počítačem podle definice prvního kandidáta na vzor. Tyto tvary čekají na uživatele, aby je verifikoval. Může se stát, že ani jeden tvar není v souladu s jazykovou kompetencí uživatele. V takovém případě jsou uživatelem tvary prvního kandidáta na vzor zamítnuty a počítač proto vyplní tabulku způsobem odpovídajícím dalšímu kandidátovi na vzor.

Další možností je, že všechny tvary slova ve všech pádech jsou správné, pak jsme vzor úspěšně nalezli. Poslední alternativou je situace, kdy některé tvary korektní jsou, jiné nikoliv. Nyní je na uživateli, aby opět správné tvary označil – zafixoval. Poté se z množiny kandidátů na vzory vyloučí ty, u nichž na příslušných pozicích nedojde ke shodě s postfixem zafixovaného tvaru a konce slova. Zároveň je možné vyloučit kandidáta, který poskytnul tvary aktuálně zobrazené v tabulce a nabídnout tvary jiného ze zbylých kandidátů.

Tento postup se opakuje tak dlouho, dokud není vzor nalezen, nebo nejsou vyčerpány všechny možnosti. Uživatel přitom může tvary nejen prohlašovat za správné a nevyhovující, ale rovněž smí odmítnuté tvary korigovat. Korekce je dovoleno provést též u více slovních tvarů. Pokaždé, když uživatel není spokojen, vyžádá si od počítače další alternativu.

Technické detaily implementace nemá smysl zde uvádět už jen proto, že systém se neustále vyvíjí. Cílem bylo vyložit princip, na kterém je algoritmus pro určení vzoru slova založen. Tento princip je zároveň nezávislý na slovním druhu. Přesto však konkrétní vzhled tabulky, do níž se tvary doplňují, se slovním druhem souvisí. U adjektiv je nutno přidat stupeň, resp. informaci, zda adjektivum stupňovat lze. U sloves se jedná o tabulku pro časování. Přidává se vid, osoba, způsob a čas.

Naše pojetí algoritmu pro určování vzorů není jediné možné. Přirozenějším způsobem by možná bylo systematické pokládání otázek uživateli, které by v podstatě odpovídalo slovním tvarům, které jednotlivé vzory odlišují. Tímto způsobem byly také vzory vytvořeny a definovány. Diskutovat by se dalo o tom, který způsob

je lepší. Jisté ovšem je to, že námi předložený algoritmus je dostatečně jednoduchý a pro uživatele je velmi příjemný. Pokládáním otázek bychom ke stejnému závěru zřejmě dospěli také. Při zvolení takového přístupu by samozřejmě vyvstaly další otázky týkající se minimalizace počtu otázek apod. Implementace by navíc v tomto případě byla mnohem složitější a vyžadovala by jistou dávku lingvistického vzdělání uživatele.

Existuje ještě jeden důvod, proč jsme se rozhodli porovnávat celé konce slov bez ohledu na jejich segmentaci. Zajímalo nás totiž, zda tímto způsobem některé vzory nesplynou v jeden. Odpověď na tuto otázku se v současnosti pokoušíme hledat a stane se zřejmě i předmětem našeho zájmu v blízké budoucnosti.

Kapitola 6

Závěr a směry budoucího vývoje

Předkládaná práce se zaměřila na automatickou morfologickou analýzu českého jazyka. Při návrhu algoritmu vlastní morfologické analýzy jsme vycházeli z algoritmického popisu české formální morfologie [Osol–96]. Námi navržený algoritmus je však o poznání efektivnější a zejména vhodnější pro praktickou implementaci.

Poněvadž jsme zvolili řešení založené na slovníku, vyvstala otázka efektivního uložení kmenových základů českých slov. Podařilo se nám navrhnout a implementovat uložení jazykových dat pomocí konečného automatu, který jsme pro účely morfologické analýzy reprezentovali jako vyhledávací strukturu trie. Tento kombinovaný přístup nám umožnil dosáhnout velmi dobrého výsledku: prostorovou složitost jsme omezili minimalizací automatu a další optimalizací, kterou jsme pro náš způsob uložení navrhli, časovou složitost pak využitím vyhledávací struktury trie, která se díky svým vlastnostem jeví jako nejvhodnější pro typ dat, s nimiž pracujeme.

K výsledkům práce patří i návrh formátu strojového slovníku češtiny a definičního souboru koncovkových množin a vzorů. V rámci řešení práce byl navržen a implementován program pro jejich převod do binárního tvaru.

Diplomová práce vrcholí návrhem a implementací morfologického analyzátoru češtiny, který současně plní úlohu lemmatizátoru a značkovače korpusu. Po provedení drobné úpravy jej lze efektivně využít i jako korektor překlepů.

Výsledkem morfologické analýzy je buď zamítnutí slova s tím, že se nejedná o správně utvořené české slovo, nebo akceptování s dodatečným určením příslušných gramatických kategorií, popř. určením základního tvaru. Vzhledem k tomu, že gramatické významy tvarů slov jsou namnoze jejich syntaktickými funkcemi, je takto otevřena cesta k dalšímu využití automatické morfologické analýzy pro analýzu syntaktickou.

Automatické generování morfologických tvarů lze použít v dalších oblastech strojového zpracování přirozeného jazyka, jako je například strojový překlad. Slovník kmenových základů se navíc může stát základnou pro různé experimenty z oblasti sémantiky. Je možné jej podrobit transformaci na derivační slovník češtiny.

Směry budoucího vývoje lze rozdělit do dvou skupin. Na prvním místě jde o teoretický výzkum v oblasti počítačového zpracování přirozeného jazyka zaměřený

zejména na slovotvorbu a její případné aplikace. Druhou oblastí našeho zájmu bude řešení praktičtějších problémů souvisejících s co možná nejširším použitím implementovaného morfologického analyzátoru.

Cesta k dalším lingvistickým experimentům s reálnými daty se otevřela již pouhým návrhem formátu strojového slovníku. Ten by se tak mohlo v budoucnu podařit transformovat na derivační slovník češtiny. Motivací je na jedné straně možnost studia slovotvorných procesů na poměrně reprezentativním datovém materiálu a dále praktický požadavek dalšího snížení objemu dat ve slovníku. V případě úspěšného nalezení souboru pravidel popisujících tvorbu kmene (kmenového základu) z kořene českého slova lze slovník kmenů nahradit slovníkem kořenů.

Dalším zdrojem inspirace, taktéž motivovaným praktickým požadavkem snížení velikosti dat, je analýza prefixů, které se mohou pojit s příslušným slovotvorným základem. Součástí řešení tohoto problému by se mohla stát také jakási sémantická klasifikace prefixů, tzn. vybudování systému prefixů nesoucích určité významy v kombinaci s určitými slovotvornými základy a jejich lexikálními významy.

Ke druhé skupině problémů, kterým bychom se v budoucnu rádi věnovali, patří rozšíření možností implementovaného analyzátoru. Hlavním vytčeným cílem by se stala analýza kolokací. Domníváme se, že by bylo vhodné řešit tento problém v rámci existujícího analyzátoru, zejména proto, že počítáme s propojením morfologické analýzy s analýzou syntaktickou. Analýza kolokací jakožto součást morfologické analýzy se v tomto ohledu stává nevyhnutelnou.

Spolu s kolokacemi se okamžitě dostavuje problematika zlepšení předzpracování textu určeného k analýze. Současný požadavek jednoho slova na samostatném řádku textu v souvislosti s kolokacemi musí ustoupit. Taktéž pevný formát výstupu analyzátoru by bylo vhodné zpřístupnit uživateli, který by pak byl schopen pomocí jistých parametrů určovat jeho podobu. Jednou z nejpalčivějších potřeb se zřejmě stane nutnost podpory uživatelského slovníku a nástroje pro jeho budování přidáváním nových slov.

Literatura

[Čerm–95] Čermák, F., Blatná, R.: Manuál lexikografie, Nakladatelství H&H, Praha, 1995

[Knu1–73] Knuth, D. E.: The Art of Computer Programming, Vol. 1, Fundamental Algorithms, Addison Wesley, 1973

[Knu3–73] Knuth, D. E.: The Art of Computer Programming, Vol. 3, Sorting and Searching, Addison Wesley, 1973

[Koze–97] Kozen, D. C.: Automata and Computability, Springer-Verlag New York, Inc., New York, 1997

[Osol–96] Osolsobě, K.: Algoritmický popis české formální morfologie a strojový slovník češtiny, disertační práce, FF MU Brno, 1996

[MČII–86] Petr, J.: Mluvnice češtiny II., Academia, Praha, 1986

[Ševe–95] Ševeček, P.: `LEMMA`, lemmatizátor pro češtinu, spustitelný program, Brno, 1995

Příloha A

Přehled symbolů programu `ajka`

Slovní druh – k	
Substantiva	1
Adjektiva	2
Zájmena	3
Číslovky	4
Slovesa	5
Příslovce	6
Předložky	7
Spojky	8
Částice	9
Citoslovce	0
Zkratky	A
Adverbia gen. z adj.	B
Posesiva	C
Deverbativní subst.	D
Deverbativní adj.	E

Rod – g	
Mužský životný	M
Mužský neživotný	I
Ženský	F
Střední	N
Libovolný	X
Muž. než. + střední	Y
Mužský + střední	U
Životný (kdo)	P
Neživotný (co)	T

Číslo – n	
Jednotné	S
Množné	P

Stupeň – d	
Nominativ	1
Komparativ	2
Superlativ	3

Pád – c	
1. pád	1
2. pád	2
3. pád	3
4. pád	4
5. pád	5
6. pád	6
7. pád	7

Osoba – p	
1. osoba	1
2. osoba	2
3. osoba	3

Čas – t	
Minulý	M
Přítomný	P
Budoucí	F

Způsob – m	
Infinitiv	F
Indikativ	I
Imperativ	R
Participium	P
Transgresiv	T
Kondicionál	C
Konjunktiv	K

Vid – a	
Perfektum	P
Imperfektum	I

Negace – e	
Afirmace	A
Negace	N

Druhy zájmen – x	
Osobní	P
Přivlastňovací	O
Ukazovací	D
Tázací	Q
Vztažná	R
Neurčitá	U
Záporná	N
Reflexivní	X

Druhy číslovek – x	
Základní	C
Řadové	O
Druhové	R
Názvy jmen	N
Názvy zlomů	D
Názvy n-tic	T

Druhy příslovcí – x	
Místa	L
Času	T
Způsobu	M
Míry	Q
Zřetele	R
Příčiny	C
Modální	D
Stavová	S

Druhy spojek – x	
Souřadné	C
Podřadné	S

Přirozený rod – h	
Mužský	M
Ženský	F

—

Příloha B

Ukázky výstupu programu `ajka`

```
$ ajka
ajka>jez
<s> j-ez  (776)
   <l> jíst
   <c> k5eAp2nSmRaI
<s> jez  (40)
   <l> jez
   <c> k1gInSc1
   <c> k1gInSc4
ajka>#

$ ajka -b
ajka>jez
jez <l>jíst <c>k5eAp2nSmRaI
jez <l>jez <c>k1gInSc1 <c>k1gInSc4
ajka>#

$ ajka
ajka>otcovi
<s> ot-c-ovi  (119)
   <l> otec
   <c> k1gMnSc3
   <c> k1gMnSc6
<s> ot-cov-i  (119)
   <l> otcův
   <c> kCgMhMnPc1
ajka>#

$ ajka -b -l mini
ajka>otcovi
otcovi <l>otec <c>k1gMnSc3 <c>k1gMnSc6
otcovi <l>otec <c>kCgMhMnPc1
ajka>#
```

```
$ ajka -a
ajka>jez
<s> j-ez  (776)
   <l> jíst
   <c> k5eAp2nSmRaI
jíst      jísti     jím       jíš       jí
jíme      jíte      jedí      jedl      jedla
jedlo     jedli     jedly     jeden     jedena
jedeno    jedeni    jedeny    jeda      jedouc
jedouce   jez       jezme     jezte     jedení
jedením   jedeních  jedeními  jedený    jedeného
jedené    jedenému  jedeném   jedeným   jedených
jedenými  jedenýma  jedená    jedenou   jedenu
jedcí     jedcího   jedcímu   jedcím    jedcích
jedcími   jedcíma
<s> jez  (40)
   <l> jez
   <c> k1gInSc1
   <c> k1gInSc4
jez     jezů    jezy    jezům   jezu    jezem
jezech  jeze
ajka>#

$ ajka -b -l 12456
ajka>otcovi
otcovi <l>otec <c>k1gMnSc3 <c>k1gMnSc6
otcovi <l>otec <c>kCgMhMnPc1
ajka>#

$ ajka -b -l 456
ajka>jedení
jedení <l>jedení <c>kDgNnSc1 <c>kDgNnSc2 <c>kDgNnSc3
<c>kDgNnSc4 <c>kDgNnSc5 <c>kDgNnSc6 <c>kDgNnPc1
<c>kDgNnPc2 <c>kDgNnPc4 <c>kDgNnPc5
jedení <l>jedený <c>kEeAgMnPc1d1 <c>kEeAgMnPc5d1
ajka>#

$ ajka -l 16
ajka>nejneuvěřitelněji
<s> nej-ne-uvěřiteln-ěji  (411)
   <l> neuvěřitelně
   <c> kBxMeNd3
ajka>#
```