Matematicko-fyzikální fakulta Univerzity Karlovy
Ústav formální a aplikované lingvistiky

Pravděpodobnostní model významových zápisů vět

Diplomová práce

Daniel Zeman

Vedoucí: RNDr. Jan Hajič, Dr.

Praha 1997

Děkuji vedoucímu diplomové práce Janu Hajičovi za jeho všestrannou pomoc, cenné připomínky a rady.

Poděkování patří též Kláře Matouškové za pečlivé přečtení rukopisu a připomínky k jeho srozumitelnosti, stylu i úpravě.

Souhlasím se zapůjčováním své diplomové práce.

Prohlašuji, že jsem práci vypracoval samostatně a použil pouze uvedené literatury.

V Praze 15.12.2000 Daniel Zeman

OBSAH

ZADÁNÍ

1. Vypracovat matematický model významových zápisů vět
2. Vypracovat program (C/C++) pro
   a) zjištění pravděpodobnosti daného zápisu na základě modelu
   b) generování zápisů na základě vedlejší informace
   c) zjištění hodnot entropie, křížové entropie a perplexity

Kapitola 1

1. ÚVOD

V různých aplikacích počítačové lingvistiky potřebujeme znát nejenom významy jednotlivých slov, ale i závislosti mezi slovy a jejich postavení ve větě. Zejména při strojovém překladu mezi dvěma přirozenými jazyky se neobejdeme bez důkladné analýzy zdrojového textu, a to jak morfologické, tak i syntaktické. Program, který by zvládnul větný rozbor, však může nalézt i jiné uplatnění: například při kontrole pravopisu, která kromě správných tvarů dokáže ohlídat i shodu větných členů v rodě, čísle a pádě.

Počítačová analýza textu v přirozeném jazyce musí zvládnout dvě základní úlohy. Jednak je potřeba zjistit, jaký význam má dané slovo ve větě, jakého je slovního druhu a jaké jsou jeho mluvnické kategorie (morfologická analýza). Druhým problémem je určení jednotlivých větných členů a stanovení závislostí mezi nimi (syntaktická analýza). Algoritmické řešení obou částí rozboru je ovšem značně obtížné, protože každý přirozený jazyk obsahuje řadu nejednoznačností. Například slovo „stát“ může být buď podstatné jméno v 1. pádě, nebo infinitiv slovesa, a to hned se dvěma různými významy („stát na návsi“ a „stát tisíc korun“). Ještě složitější je situace při syntaktické analýze, kde neexistuje žádné jednoznačné pravidlo, které by slovnímu tvaru přiřadilo jeho postavení ve větě. Určení větného členu závisí vždy i na ostatních slovech věty, případně i na kontextu dalších vět v okolí.

Provádí-li rozbor věty člověk, je vůči počítači ve výhodě: při zpracování textu je schopen vést v patrnosti obrovské množství informací o větách,

které předcházely, porovnávat tyto informace s textem, který už viděl v minulosti, a nacházet analogie. Člověk tak v naprosté většině nejednoznačných případů dokáže najít správné řešení. Neuspět může pouze u krátkých a z kontextu vytržených vět. (Např. v mluveném projevu nelze rozlišit věty „Syn je zdravý.“ a „Syn je zdraví.“.) Podotkněme však, že i v těch případech, kdy význam věty neumíme určit, existuje jediné správné řešení: je jím ten význam, který měl na mysli autor věty, když ji pronesl nebo napsal. Při počítačovém zpracování přirozeného jazyka musíme vždy počítat s určitým rizikem, že nalezený výsledek neodpovídá skutečnému významu věty.

Existují různé metody, jak provádět analýzu přirozeného jazyka; obecně můžeme rozlišit dva základní přístupy. První, *klasický,* se pokouší popsat jazyk co možná nejúplnější sadou pravidel a na jejich základě získávat informace o analyzované větě.[1] Druhý, *pravděpodobnostní* přístup, se spoléhá na relativní četnost zkoumaného jazykového jevu v již rozebraném textu. Může-li mít nějaká věta více různých významů, vezme se v úvahu ten nejpravděpodobnější.[2] Oba přístupy je možné i kombinovat, chceme-li pravděpodobnostní model dodatečně spoutat mluvnickými pravidly, která považujeme za neměnná.

Pravděpodobnostní metody jsou nasazovány při nejrůznějších lingvistických aplikacích zejména v poslední době, přičemž zkoumaným jazykem je nejčastěji angličtina.

---

[1] Příkladem může být práce [[13]], popisující analýzu českého textu při strojovém překladu do ruštiny. Rovněž [[15]], která se zabývá syntaktickou analýzou anglické a české věty, používá nepravděpodobnostní, tzv. transformační přístup.

[2] V [[1]] najdeme pokročilejší aplikaci pravděpodobnostních postupů v angličtině při rozpoznávání řeči. V [[7]] se stochastické metody využívají k automatickému generování morfologických značek (tagů) v češtině.

Cílem této práce je aplikovat pravděpodobnostní postupy na rozbor věty v českém jazyce.

K a p i t o l a 2

2. VĚTY A JEJICH ZÁPISY

Analýzou věty budeme rozumět převedení věty na její syntaktický zápis.[3] Mějme **větu** *S*, zadanou jako posloupnost slov $s_1, s_2, \mathrm{K}, s_n$. **Syntaktický zápis věty** je strom *M*, zadaný množinou vrcholů *V*, množinou hran *H* a informací o tom, který vrchol je kořenem stromu.[4] Nebude-li v následujícím textu řečeno jinak, budeme pojmy „strom“ a „syntaktický zápis“ ztotožňovat.

Vrchol v_i stromu *M* (s výjimkou kořene) odpovídá právě jednomu slovu věty *S*. Obsahuje základní tvar slova a další informace. Přesný tvar vrcholu stanovíme později. Prozatím je podstatné pouze to, že obsahuje i původní tvar slova a jeho pořadí ve větě, takže je možné z každého syntaktického zápisu jednoznačně zrekonstruovat větu, která mu dala vzniknout.

Každý strom má v kořeni zvláštní vrchol #. Tento vrchol neodpovídá žádnému slovu ve větě a slouží pouze k označení kořene. V každém stromu je právě jeden takový vrchol. Všechny ostatní vrcholy reprezentují jednotlivá

---

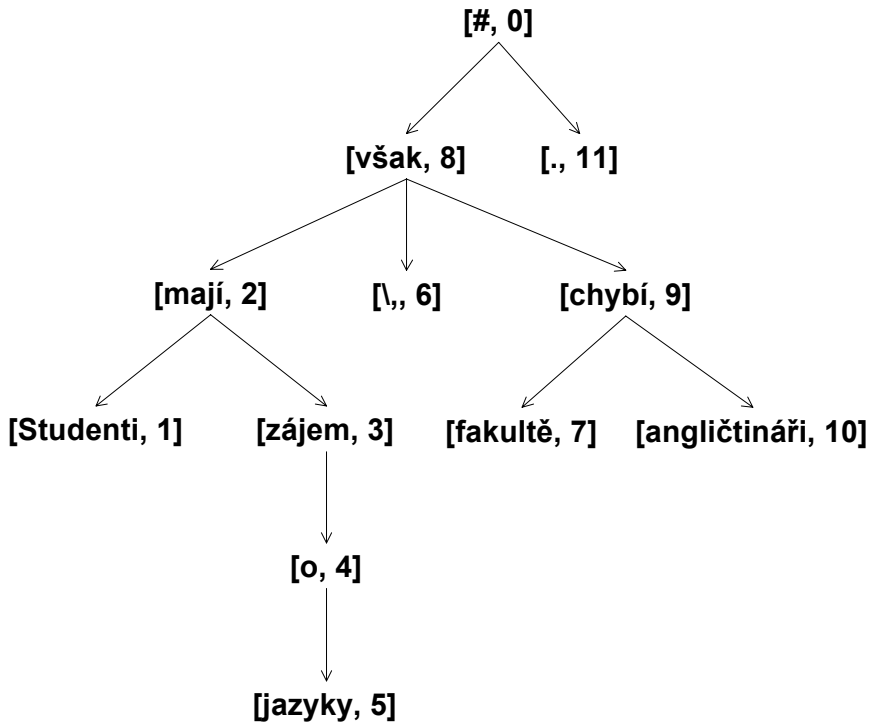
[3] Záměrně zde nahrazujeme pojem *významový zápis* z názvu a zadání této diplomové práce, protože data, která jsou v současné době k dispozici, ještě nepokrývají významovou rovinu jazyka (viz např. [[14]]). Zdůrazněme, že z našeho pohledu jde o pouhou kosmetickou úpravu, kterou se chceme terminologicky přiblížit běžně používanému vrstvovému modelu jazyka. Jak na významové, tak na povrchově syntaktické rovině se pracuje se závislostmi mezi větnými členy — závislostmi, které lze znázornit stromovou strukturou. Jestliže tedy v této kapitole definujeme syntaktický zápis jako strom, mohli bychom analogicky definovat i významový zápis věty. Na matematickém modelu a algoritmech, které budeme popisovat v dalších kapitolách, se nic nemění.

[4] Viz též [[11]], zejména články 2.1, 2.4 a 7.2.

slova věty. Každý vrchol leží na *i*-té úrovni, kde *i* je jeho vzdálenost od kořene. V rámci úrovně jsou slova uspořádána podle svého pořadí ve větě. Hrany považujeme za orientované směrem od kořene k listům.

Hrana h_j stromu M vyjadřuje vztah závislosti mezi dvěma větnými členy. Je to uspořádaná dvojice významů $\left[v_{\check{r}}, v_z\right]$, kde $v_{\check{r}}$ představuje řídící větný člen a v_z je závislý větný člen. Hrana ve stromu tedy odpovídá skladební dvojici ve větě. Poznamenejme, že z definice stromu nevyplývají žádná pravidla, která by předem říkala, jaké slovo smí na jakém záviset. Apriori tedy nevylučujeme ani takové závislosti, které se ve skutečné větě nemohou vyskytnout, nebo lépe řečeno, jejichž výskyt je velmi nepravděpodobný. Požadujeme však, aby každé slovo věty bylo právě jedním větným členem, čímž se poněkud odchylujeme od lingvistických představ o větném členu. V našem pojetí budou rovnoprávnými vrcholy stromu např. i předložky (nikoli celé předložkové vazby), ba dokonce i interpunkční znaménka, s nimiž musíme nakládat jako se slovy.

PŘÍKLAD: Věta „Studenti mají zájem o jazyky, fakultě však chybí angličtináři.“ má následující zápis:




```
[#,0]([však,8]([mají,2]([Studenti,1],[zájem,3]
([o,4]([jazyky,5]))),[\,,6],[chybí,9]([fakultě,7],
[angličtináři,10])),[.,11])
```

K a p i t o l a 3

3. PRAVDĚPODOBNOSTNÍ JAZYKOVÝ MODEL

Již jsme řekli, že naším hlavním úkolem bude nalézt ke známé větě *S* její syntaktický zápis *M*. Pro člověka, který je narozdíl od počítače schopen vnímat široký kontext věty, je to většinou deterministická úloha. Avšak počítač velmi rychle narazí na četné nejednoznačnosti v jazyce, pro které nebude mít k dispozici rozhodovací pravidlo. Je totiž značně obtížné předem podchytit všechna pravidla, která mohou být použita při rozboru věty. Gramatických zákonitostí je velké množství a není možné u všech předem říci, jaký vliv budou mít na určení závislostí ve větě. V přirozeném jazyce navíc existují četné nepravidelnosti, které se zákonům mluvnice vymykají, a je proto nutné popsat je explicitně.

Víme již, že vždy existuje jedno správné řešení, jeden strom *M*, odpovídající tomu významu věty *S*, který měl na mysli její autor. Počítač, který je schopen sledovat jen omezené množství souvislostí, dospěje téměř vždy k celé množině stromů, které pro danou větu přicházejí v úvahu. Je jistě přirozené, jestliže si z této množiny vybereme strom, který je pro danou větu nejpravděpodobnější. Podmínkou je, že budeme umět odhadnout pravděpodobnost stromu *M* v závislosti na větě *S*, nebo alespoň o každé dvojici stromů říci, který z nich je pravděpodobnější.

3.1. DEFINICE PRAVDĚPODOBNOSTI

Až dosud jsme používali pojem **pravděpodobnost** intuitivně a stačilo nám, že tušíme jeho význam a interpretaci. V níže uvedené definici chceme především říci, co rozumíme pravděpodobností syntaktického zápisu a pravděpodobností hrany. Nemáme ovšem v úmyslu suplovat literaturu o teorii pravděpodobnosti[5] a nebudeme zde rozebírat vlastnosti této míry, přestože v dalším textu předpokládáme jejich znalost.

Mějme množinu U, jejíž prvky nazveme **elementární jevy**. Vybereme-li náhodně jeden elementární jev, provádíme **pokus** α. Výsledkem pokusu je vybraný elementární jev; všechny výsledky považujeme za stejně pravděpodobné. Libovolná podmnožina A množiny U je potom **jev**, který nastane, jestliže výsledek pokusu α leží v množině A. **Pravděpodobnost** jevu A se definuje jako poměr počtu prvků množiny A (výsledků příznivých jevu A) k počtu prvků množiny U (všech možných výsledků):

$$p(A) = \frac{|A|}{|U|}$$

Víme-li předem, že výsledek pokusu bude ležet v množině B (splňovat podmínku pro jev B), potom nás místo obyčejné pravděpodobnosti $p(A)$ zajímá **podmíněná pravděpodobnost** $p(A|B)$, definovaná jako podíl počtu možných výsledků, které jsou příznivé oběma jevům, a počtu výsledků, které jsou příznivé jevu B:

[5] Viz též [[10]], kapitola I.

$$p(A|B) = \frac{|A \cap B|}{|B|}$$

Nás budou zajímat zejména dva jevy: výskyt stromu *M* velikosti *m* a výskyt hrany *h*.

V prvním případě vezmeme za elementární jevy syntaktické zápisy všech českých vět délky $n = m - 1$ slov, které kdy někdo pronesl nebo napsal. Dva různé výskyty téhož stromu jsou tedy dva různé elementární jevy. Očíslujeme-li všechny výskyty všech stromů, potom množina *U* obsahuje právě všechna takto vzniklá čísla. Počet elementárních jevů je konečný; zanedbáváme ovšem fakt, že jsme do českého jazyka připustili jen takové věty, které se v něm už někdy v minulosti vyskytly. Jev *A*, jehož pravděpodobnost sledujeme, v tomto případě říká: výsledkem pokusu α je výskyt stromu totožného se stromem *M*.

Zcela analogicky pak definujeme jev *B* jako výskyt hrany *h* v některé pronesené nebo napsané české větě. Do množiny $B \subset U$ patří všechna čísla, která označují výskyt hrany se stejným řídícím i závislým vrcholem, jako má hrana *h*. Existuje $|L|^2$ takových podmnožin množiny *U*, kde *L* je množina všech slovních tvarů v češtině[6]; tyto podmnožiny společně vytvářejí disjunktní pokrytí *U*. V dalším textu budeme ztotožňovat hranu *h* s jevem *B* a pravděpodobnost výskytu hrany *h* budeme označovat $p(h)$. Analogicky budeme postupovat i u stromů a dalších sledovaných prvků: jev, který hovoří o výskytu jistého stromu, budeme označovat stejným písmenem jako dotyčný strom.

---

[6] Zde předpokládáme, že vrchol stromu, tj. slovo, je reprezentováno slovním tvarem, jaký se objevil ve větě. Jak uvidíme v kapitole Kapitola 5 , může tomu být i jinak: za shodné můžeme považovat

(Poznámky pokračují na další straně.)

Tam, kde budeme hovořit o podmíněné pravděpodobnosti, musíme rozšířit definici elementárního jevu. Chceme-li znát pravděpodobnost $p(h_i|h_1,\text{K},h_{i-1})$ výskytu *i*-té hrany v závislosti na výskytu $i-1$ předcházejících hran, musíme za elementární jevy vzít všechny výskyty všech uspořádaných *i*-tic hran. Ještě před provedením pokusu α víme, že prvních $i-1$ hran v elementárním jevu, který bude jeho výsledkem, se bude shodovat s hranami h_1 až h_{i-1}. Podmiňující jev je množina všech elementárních jevů, které splňují tuto podmínku. Podmíněný jev je množina všech elementárních jevů, jejichž poslední hrana se shoduje s hranou h_i.

3.2. FORMULACE ÚLOHY

V úvodu této kapitoly jsme řekli, že chceme nalézt strom, který je s největší pravděpodobností syntaktickým zápisem věty *S*. Označme $p(M|S)$ pravděpodobnost, že strom *M* je zápisem věty *S*, jestliže tuto větu známe. Potom budeme hledat

$$\arg\max_M p(M|S),$$

tedy takový strom *M*, pro nějž je $p(M|S)$ nejvyšší. Podle Bayesova vzorce[7] můžeme vyjádřit:

$$p(M|S) = \frac{p(S|M)\cdot p(M)}{p(S)}$$

---

také dva vrcholy, jejichž slova vznikla ze stejného slovníkového hesla, nebo náleží do stejných mluvnických kategorií.

[7] Viz [[6]], § 12.

kde $p(S|M)$ je pravděpodobnost, že věta, jejímž syntaktickým zápisem je strom *M*, je právě *S*; *p*(*M*) je pravděpodobnost výskytu stromu *M* (tj. pravděpodobnost, že strom *M* je syntaktickým zápisem vůbec nějaké věty) a *p*(*S*) je pravděpodobnost výskytu věty *S*. Z hlediska hledání nejpravděpodobnějšího stromu je ovšem poslední veličina pouhou konstantou, která výsledek nijak neovlivní:

$$\arg\max_M \frac{p(S|M) \cdot p(M)}{p(S)} = \arg\max_M (p(S|M) \cdot p(M))$$

Naším úkolem tedy bude odhadnout pravděpodobnosti $p(S|M)$ a $p(M)$.

3.3. Převod stromu na posloupnost hran

M je dán množinou vrcholů *V*, množinou hran *H* a kořenem. Protože *M* je strom (tedy souvislý graf), můžeme *V* odvodit z *H*. Dále můžeme z *V* odvodit, který vrchol je kořen, protože kořen má zvláštní tvar #. (I kdyby množina *H* byla prázdná, víme, že *V* obsahuje právě jeden vrchol, a tím je #.) Strom *M* je tedy dostatečně zadán už samotnou množinou hran, takže se zdá, že můžeme napsat $p(M) = p(H)$. Je-li tomu skutečně tak, dostává náš cíl poněkud jasnější obrysy, protože zatímco odhad pravděpodobnosti stromu z jeho relativní četnosti je vzhledem k obrovskému množství stromů těžko představitelná úloha, pravděpodobnost množiny lze stanovit jako součin podmíněných pravděpodobností jejích prvků.

Podíváme se na korespondenci mezi stromem *M* a jeho množinou hran *H* podrobněji. Můžeme definovat funkci *rozlozit*, která převede libovolný syntaktický zápis věty na množinu hran, a funkci *slozit*, která naopak z množiny hran udělá strom. Přitom chceme, aby převod byl v obou smě-

rech jednoznačný, tj. aby každému stromu funkce *rozlozit* přiřadila právě jednu množinu hran a funkce *slozit* aby každé množině hran přiřadila právě jeden strom.

Přímo se nám nabízí, abychom definovali první funkci takto: vezmeme strom M, vyjmeme z něj množinu hran, kterou je zadán, a ta bude výsledkem funkce *rozlozit*. Už z definice plyne, že každý strom má právě jednu takovou množinu. Navíc je uvedené zobrazení prosté, protože žádné dva různé stromy nejsou zadány stejnou množinou hran. Pokud by dva odlišné stromy měly stejnou množinu hran, musely by se lišit buď v množině vrcholů, nebo v kořeni. První případ by znamenal, že v množině V jednoho ze stromů leží vrchol, do kterého nevede žádná hrana. Pak by ale dotyčný graf nebyl strom. Zbývá tedy jediná možnost, že se stromy liší v tom, který vrchol je kořen. Jak ale vyplývá z naší definice, syntaktický zápis věty má kořen ve zvláštním vrcholu #, který je v každém stromu právě jeden. Uvedený případ proto nemůže nastat.

Protože funkce *rozlozit* je prostá, můžeme pro všechny množiny H z jejího oboru hodnot definovat funkci *slozit* jako *rozlozit*$^{-1}$. Zbude nám však mnoho množin hran, které leží mimo obor hodnot funkce *rozlozit*, protože nepopisují strom: například množiny, v nichž se vyskytuje hrana se stejným řídícím i závislým vrcholem. Pro tyto množiny musíme funkci *slozit* dodefinovat. Jestliže H je množina hran, které nemohou náležet stromu, potom za výsledek *slozit*(H) označíme zvláštní strom $M_{\#}$, který nereprezentuje žádnou skutečnou větu. Na $M_{\#}$ můžeme pohlížet jako na prázdný strom, obsahující pouze kořen. Právě tak ovšem můžeme definovat speciální strom s libovolným počtem vrcholů, jestliže žádný ze symbolů, kterými vrcholy označíme, nebude odpovídat žádnému slovnímu tvaru. Máme tedy zvláštní strom $M_{\#,n}$ pro každou třídu stromů, které mají n vrcholů; bude se nám to hodit později, až zúžíme obor na stromy s pevným počtem vrcholů.

Abychom si usnadnili implementaci, budeme dále požadovat, aby hrany v množině *H* byly uspořádány. *H* tedy nebude pouhá množina, ale posloupnost hran. Uspořádání by mělo vyplývat ze struktury stromu, např. může jít o pořadí hran při prohledávání do hloubky. Volbu uspořádání provedeme později. Za nepřípustné posloupnosti, vedoucí na strom $M_{\#}$, budeme pak považovat i ty množiny, jejichž hrany sice dávají strom, ale nejsou správně uspořádány.

Jestliže umíme převést strom na posloupnost hran a zpátky, nahradíme pravděpodobnost stromu *M* pravděpodobností jemu odpovídající posloupnosti *H*. Dopouštíme se tak značného zkreslení skutečnosti, protože největší pravděpodobnost dostane zvláštní strom $M_{\#}$, reprezentující všechny špatné posloupnosti. Ale poměry pravděpodobností ostatních stromů zůstanou zachovány, proto můžeme hledat takovou posloupnost *H*, že $slozit(H) \neq M_{\#}$ a součin $p(S|H).p(H)$ je nejvyšší možný:

$$\arg\max_M \left(p(S|M)\cdot p(M)\right) = slozit\left(\arg\max_H \left(p(S|H)\cdot p(H)\right)\right),$$

kde obor *H* je omezen na takové posloupnosti, že $slozit(H) \neq M_{\#}$.

3.4. VYLOUČENÍ ŠPATNÝCH STROMŮ

Jak jsme řekli v předchozí kapitole, syntaktický zápis věty obsahuje dostatečné množství informací, abychom mohli větu jednoznačně zrekonstruovat. Pro každý vrchol známe původní tvar slova, které zastupuje, i jeho pořadí ve větě. Můžeme tedy definovat funkci *syntéza*, která každému stromu *M* přiřadí větu *S*, jejímž syntaktickým zápisem strom je. (Duální operaci pak provádí funkce *analýza*, jejíž nalezení je vlastně cílem této práce.) Podobně jako funkce *slozit*, popsaná v předchozí části, musí se *syntéza* vypo-

řádat se stromy, které nejsou syntaktickým zápisem žádné věty. Jde zejména o zvláštní strom $M_{\#}$, ale jsou i další možnosti: například strom, ve kterém dvě slova budou mít uvedeno stejné pořadí, je neplatný. Pro všechny takové stromy dodefinujeme funkci *syntéza* a přiřadíme jim zvláštní větu $S_{\#}$, která se neshoduje s žádnou skutečnou větou v českém jazyce.

Teď už snadno určíme pravděpodobnost $p(S|H)$. Pomocí funkcí *slozit* a *syntéza* získáme větu S_H, ze které posloupnost H vznikla. Tu pak porovnáme s větou S a položíme $p(S|H) = 1$, jestliže $S = S_H$, a $p(S|H) = 0$ jinak. Tím jsme zaručili, že výsledná posloupnost hran, kterou dostaneme při větné analýze, je správně uspořádanou posloupností, že definuje strom, že tento strom je korektním syntaktickým zápisem věty a že tato věta je S. (Poslední podmínka znamená, že pro každé slovo věty lze nalézt vrchol stromu, u něhož je správně uveden tvar tohoto slova a jeho pořadí ve větě.) Při takto definované pravděpodobnosti $p(S|H)$ můžeme už bez doplňujících podmínek napsat:

$$\arg\max_M \left(p(S|M) \cdot p(M)\right) = slozit\left(\arg\max_H \left(p(S|H) \cdot p(H)\right)\right)$$

Všimněme si, že pravděpodobnost $p(S|H)$ hraje pouze formální, technickou roli. Rozděluje stromy na „dobré“ a „špatné“ podle toho, zda nějakým způsobem vybočují z námi vytyčených mezí, nikoli na základě jazykových zákonitostí. To je úkol pro poslední a nepochybně nejdůležitější veličinu v našem jazykovém modelu: pravděpodobnost $p(H)$.

3.5. PRAVDĚPODOBNOST POSLOUPNOSTI HRAN

Chceme stanovit pravděpodobnost stromu M reprezentovaného posloupností hran H tak, aby její poměr k pravděpodobnostem ostatních stromů pokud možno odpovídal poměru jejich relativních četností mezi všemi správně utvořenými syntaktickými stromy. Udělejme si nejprve představu, kolik takových stromů vlastně existuje.

Obecně lze vytvořit nekonečně mnoho různých grafů, které odpovídají definici stromu. Jinak je tomu, jestliže předem víme, kolik má mít strom vrcholů. Pomocí četnosti v trénovacích datech můžeme odhadnout pravděpodobnost $p(m)$, že náhodně vybraný strom má m vrcholů. Potom budeme hledat pravděpodobnost $p(M|m)$, přičemž možných stromů s m vrcholy je už jen konečně mnoho. Ve skutečnosti se v následujícím textu rovnou omezíme na stromy s m vrcholy, protože při praktickém použití budeme vždy dopředu vědět, kolik vrcholů má hledaný strom mít. Stromům s jiným počtem vrcholů můžeme bez obav přisoudit nulovou pravděpodobnost, protože nejsou syntaktickým zápisem věty, kterou rozebíráme (a platí pro ně $p(S|H) = 0$). Rezervujme nyní písmeno n pro počet slov ve větě, kterou rozebíráme; strom M bude mít $n+1$ vrcholů (pro každé slovo jeden vrchol a přídavný vrchol pro kořen).

Vrátíme se teď k naší otázce: kolik je stromů s $n+1$ vrcholy? Obecně nad množinou m vrcholů existuje m^{m-2} různých stromů.[8] Tento výsledek z teorie grafů ovšem předpokládá, že stromy jsou dány pouze množinou vrcholů a množinou hran; přitom se nebere ohled na to, který vrchol považujeme za kořen stromu. V našem případě je kořen dán předem jako pří-

[8] Důkaz je podán v [[11]], věta 7.2.4 a důsledek 7.2.5.

davný vrchol, z jehož tvaru už je patrné, že jde o kořen. Proto můžeme u uvedeného vztahu zůstat: máme $m = n + 1$ vrcholů a $(n+1)^{n-1}$ stromů. Např. pro větu o 6 slovech, kterou předem známe, je to 16 807 stromů za předpokladu, že ke každému slovu věty bude existovat právě jeden možný vrchol; pro libovolnou větu se stejným počtem slov bychom uvedené číslo museli navíc vynásobit počtem různých šestic slov, které český jazyk připouští. K tomuto ilustračnímu příkladu se ještě vrátíme; prozatím je pro nás důležité zjištění, že počet vrcholů (a tedy i hran) stromu *M* je předem dán.

Řekli jsme, že strom *M* budeme reprezentovat posloupností hran *H*; chceme znát pravděpodobnost této posloupnosti. Jinými slovy, chceme vědět, s jakou pravděpodobností se v *H* současně objeví hrany h_1 až h_n v daném pořadí:

$$p(H) = p(h_1) \cdot p(h_2|h_1) \cdot \mathrm{K} \cdot p(h_n|h_1, \mathrm{K}\ , h_{n-1})$$

Pravděpodobnosti v uvedeném vztahu jsou podmíněné, protože výskyt každé hrany je závislý na ostatních prvcích posloupnosti. Přitom můžeme rozlišit závislosti technického charakteru (např. že hrany v *H* nesmí tvořit cyklus) a závislosti charakteru jazykového. Jak je vidět, znalost podmíněných pravděpodobností by nám ušetřila i práci, kterou jsme si dali v předchozích odstavcích s odstraňováním špatných posloupností *H*: určení součinu podmíněných pravděpodobností je vlastně ekvivalentní stanovení přímo pravděpodobnosti stromu *M*. Ale my chceme zjišťovat rozdělení pravděpodobností z relativních četností jednotlivých hran v trénovacích datech. Nemůžeme si dovolit evidovat četnost každé hrany v závislosti na všech ostatních hranách ve větě. Znamenalo by to znát četnost každé možné *n*-tice hran; těch je takové množství, že i kdyby se v trénovacích datech vyskytla jen každá miliontá, nedokázali bychom všechny četnosti uložit. Hlavní problém však vyvstává právě na opačné straně: nikdy nenashromáž-

díme dostatek trénovacích dat, abychom mohli jednotlivé četnosti srovnávat. Porovnání četností dvou *n*-tic nám neřekne prakticky nic o tom, která z nich lépe odpovídá danému jazyku; dozvíme se pouze, která měla to štěstí a „strefila“ se do našich trénovacích dat, přičemž v naprosté většině případů to nebude ani jedna. Je zřejmé, že takto definovaný jazykový model by byl velmi nespolehlivý.

Faktorem, způsobujícím nespolehlivost modelu, je počet jeho **parametrů**, tedy potenciální počet různých hodnot, jejichž četnost chceme evidovat. Pro ilustraci: Za předpokladu, že každému slovnímu tvaru odpovídá právě jeden možný vrchol syntaktického stromu, že čeština má 6 000 000 slovních tvarů a, podobně jako v posledním příkladu, že délka české věty nepřesahuje 6 slov, potom výše uvedený model bude mít asi $7{,}8.10^{44}$ parametrů!

Budeme tedy předpokládat, že výskyt hrany *h* není závislý na výskytu ostatních hran ve stromu:

$$p\left(h_i \middle| h_1, \mathrm{K}\ , h_{i-1}\right) = p\left(h_i\right),$$

kde $p\left(h_i\right)$ je pravděpodobnost výskytu hrany h_i, tedy pravděpodobnost, že příslušný význam v_z v nějaké české větě závisí na příslušném významu $v_ř$. Tuto pravděpodobnost odhadneme relativní četností takové závislosti v trénovacích datech. Pravděpodobnost celé posloupnosti potom bude

$$p(H) = \prod_{i=1}^{n} p\left(h_i\right).$$

Počet parametrů modelu se tak sníží na počet možných dvojic vrcholů. Označíme-li *L* množinu všech entit, které se mohou stát vrcholem syntak-

tického stromu, potom počet parametrů bude $|L|^2$ (v uvedeném ilustračním příkladu je to $3{,}6.10^{13}$).

Ověřme, zda hodnota $p(H)$ splňuje podmínky pro pravděpodobnost, tj.:

- $p(H) \in \langle 0;1 \rangle$
- $\sum_{H} p(H) = 1$, kde H je libovolná posloupnost hran, jejichž vrcholy jsou prvky L.

První podmínka je zřejmě splněna, neboť součin čísel z intervalu $\langle 0;1 \rangle$ vždy zůstane v tomto intervalu.

Platnost druhé podmínky dokážeme níže. Zopakujme, že zde hovoříme o součtu pravděpodobností všech různých *n*-tic hran, nikoli všech stromů. Na součtu se mohou podílet i nenulové pravděpodobnosti takových *n*-tic, které strom nedávají; potom součet pravděpodobností těch *n*-tic, které jsou stromem, nutně bude menší než 1. Vzhledem k našemu cíli to však nevadí, jak jsme zjistili v předchozích částech této kapitoly.

VĚTA 1: Nechť $p(h)$ je pravděpodobnost výskytu hrany h. Označme L^2 množinu všech hran. Potom pro libovolné přirozené číslo n platí:

$$\sum_{i=1}^{|L^2|^n} \prod_{j=1}^{n} p\left(h_{i,j}\right) = 1$$

DŮKAZ:

1) Věta platí pro $n = 1$. V tom případě jde totiž o prostý součet pravděpodobností všech hran, a ten musí být roven 1.

2) Jestliže věta platí pro $n = k$, pak platí i pro $n = k + 1$. Máme *k*-tice hran, každou z nich můžeme $|L|^2$ způsoby rozšířit na $(k+1)$-tici. Přitom pravděpodobnost původní *k*-tice vynásobíme pravděpodobností rozšiřující hrany. Součet pravděpodobností všech možných rozšíření jedné *k*-tice je roven pravděpodobnosti této *k*-tice:

$$\sum_{j=1}^{|L|^2} p(h_j) \cdot p(h_1, \mathrm{K}\ , h_k) = p(h_1, \mathrm{K}\ , h_k),$$

$$\text{protože} \sum_{j=1}^{|L|^2} p(h_j) = 1.$$

Jelikož jsme předpokládali, že věta platí pro $n = k$, je součet pravděpodobností všech *k*-tic roven 1, a tedy i součet pravděpodobností všech $(k+1)$-tic je roven 1. Tím je věta dokázána.

K uvedenému důkazu nepotřebujeme vědět, jestli je pravděpodobnost rozšiřující hrany h_{k+1} závislá na předchozích hranách. Stačí nám skutečnost, že součet pravděpodobností všech možných rozšiřujících hran dává 1. Bude-li pravděpodobnost hrany h_{k+1} podmíněna tím, jaké hrany se v posloupnosti objevily před ní, dojde pouze k přerozdělení pravděpodobností mezi kandidáty na $(k+1)$-ní pozici, ale jejich součet zůstane roven 1. Jestliže se tedy rozhodneme pracovat s nepodmíněnou pravděpodobností, nebudou tím narušeny nutné technické podmínky, které na pravděpodobnost klademe. I tak se ovšem námi stanovené hodnoty mohou od skutečných pravděpodobností značně lišit, takže otázku podmíněnosti ani nadále nepustíme ze zřetele.

3.6. N-GRAMOVÝ JAZYKOVÝ MODEL

Při hledání pravděpodobnosti posloupnosti hran jsme probrali dva extrémy. V prvním jsme předpokládali, že pravděpodobnost hrany h_i závisí na všech předchozích hranách h_1 až h_{i-1}. Řekli jsme, že výhodou takového modelu by byla jeho absolutní přesnost, nevýhodou pak vysoká nespolehlivost. Druhým extrémem byl předpoklad, že pravděpodobnost hrany h_i je zcela nezávislá na ostatních hranách ve stromu. Takový přístup zvyšuje spolehlivost modelu, ale také jeho nepřesnost, jelikož závislost dvou větných členů je zde zcela vytržena z kontextu zbytku věty. V pravděpodobnostním modelování jazyka se proto obvykle volí kompromis a předpokládá se, že zkoumaný jev závisí pouze na omezeném kontextu *k* jiných jevů. V našem případě by tedy pravděpodobnost výskytu hrany h_i závisela na výskytu k jiných hran ve stromu, kde $k < n$; hovoříme pak o $(k+1)$-gramovém jazykovém modelu:

$$p\left(h_i \middle| h_1, \text{K}, h_{i-1}\right) = p\left(h_i \middle| h_{i-k}, \text{K}, h_{i-1}\right)$$

Záměrně jsme použili výraz „jiné“ hrany, nikoli „předchozí“, protože volba kontextu zcela závisí na autorovi jazykového modelu. Pro příklad sáhněme do jiné úlohy jazykového modelování: chceme zjistit ke každému slovu daného textu jeho morfologické atributy. K tomu potřebujeme rozlišit významy u některých homonym - např. slovní tvar „je“ má jiné atributy ve větě „Otec je zaneprázdněn.“ a jiné ve větě „Potkal jsem je včera.“. Jazykový model nám může v rozhodování pomoci tím, že na základě kontextu přisoudí jednomu z obou významů slova „je“ vyšší pravděpodobnost. Hledáme tedy pravděpodobnost výskytu toho kterého významu slova „je“ v závislosti na výskytu *k* jiných slov, která zahrneme do kontextu. Jak však ukazuje uvedený příklad, volba kontextu nemusí být jednoduchá. V první větě význam dotyčného slova vyplývá především ze slova následujícího

(„je zaneprázdněn“), zatímco ve druhé větě je dostatečně určen slovy předcházejícími, především slovem „potkal“. Kontext tedy může obsahovat *k* předcházejících slov, *k* následujících slov, ale i mnohem komplikovanější kombinace: například poslední jmennou frázi v předcházejícím textu. Nemáme-li ovšem dobré důvody pro konstrukci složitějšího kontextu, postačí nám *k* předchozích slov. V praxi se ukazuje vhodné položit $k = 1$ (potom hovoříme o dvougramovém modelu, čili **bigramu**) nebo $k = 2$ (třígramový model, **trigram**). Vyšší *k* už příliš snižují spolehlivost modelu a nulové *k* (kde pravděpodobnost slova nezávisí na kontextu) je nepřesné.

Vraťme se však k naší úloze, tedy k hledání pravděpodobnosti syntaktických závislostí ve větě. Tento problém vykazuje vyšší kombinatorickou složitost než zmíněné morfologické modelování, proto se rovnou omezíme na kontext velikosti 1. Nemáme také žádné vodítko pro volbu kontextu, protože dosud získané zkušenosti se týkají modelování prostého textu, ne jeho syntaktických struktur. Můžeme ale s výhodou využít faktu, že máme hrany našeho syntaktického stromu uspořádány do posloupnosti. U každé hrany s výjimkou první můžeme uvažovat kontext té hrany, která jí v posloupnosti bezprostředně předcházela. Všimněme si, že jsme tím jeden ze stěžejních parametrů modelu ponechali na funkci *rozlozit*, která určuje uspořádání hran ve stromu. Konkrétní, prozatím odložená definice této funkce tak může významně ovlivnit výsledky větného rozboru.

Jestliže pravděpodobnost jedné hrany závisí na předchozí hraně, pak pravděpodobnost celé posloupnosti nově zapíšeme takto:

$$p(H) = p(h_1) \cdot p(h_2|h_1) \cdot \ldots \cdot p(h_n|h_{n-1}) = p(h_1) \cdot \prod_{i=2}^{n} p(h_i|h_{i-1})$$

Obdobně jako v důkazu Věty 1 lze ověřit, že součet všech takto definovaných pravděpodobností je roven 1, jestliže pro každé i, $1 \le i \le |L|^2$, platí $\sum_{j=1}^{|L|^2} p\left(h_j \middle| h_i\right) = 1$.

K a p i t o l a 4

4. ALGORITMUS

V předchozí kapitole jsme ukázali, že problém nalezení syntaktického zápisu věty lze převést na hledání nejpravděpodobnější posloupnosti hran, která reprezentuje danou větu. Přitom jsme opakovaně zdůraznili, že chceme trénovat příslušné pravděpodobnostní rozdělení na textu, který byl již syntakticky rozebrán jinou metodou, resp. kterému byly syntaktické struktury (syntaktické zápisy) přiřazeny ručně. V této části se na trénování podíváme podrobněji. Dotkneme se i dalších otázek spojených se získáním pravděpodobnostního rozdělení a popíšeme algoritmus, který za pomoci tohoto rozdělení převede na syntaktický zápis dosud nerozebranou českou větu. Nakonec se vrátíme k jazykovému modelu (tj. k rozložení pravděpodobností) a budeme ho testovat.

4.1. TRÉNOVÁNÍ

Základním kamenem našeho modelu je rozdělení pravděpodobností hran, neboli závislostních vztahů v české větě. Naším úkolem nyní bude získat toto rozdělení, čili stanovit pravděpodobnost každé myslitelné hrany.

Na začátku kapitoly Kapitola 3 jsme definovali pravděpodobnost jevu *A* pomocí elementárních jevů a pokusu α, při kterém náhodně jeden elementární jev vybereme. Definujme **relativní četnost** jevu *A* jako poměr počtu pokusů, po nichž nastal jev *A*, k celkovému počtu provedených pokusů:

$$r(A) = \frac{c(A)}{c(\alpha)}$$

Provedeme-li velké množství pokusů α, bude se relativní četnost jevu A blížit jeho pravděpodobnosti. Pokud tedy neznáme předem velikost množin U a A, což je náš případ, můžeme pravděpodobnost jevu odhadnout jeho relativní četností.

Analogicky odhadneme i podmíněnou pravděpodobnost $p(A|B)$ jako poměr počtu pokusů, po nichž nastal současně jev A i jev B, k počtu pokusů, po nichž nastal jev B:

$$r(A|B) = \frac{c(A,B)}{c(B)}$$

Pro nás dotyčné jevy představují výskyty jisté hrany h. Pokus α nasimulujeme tak, že vezmeme libovolný text, k němuž máme k dispozici syntaktické struktury vět, a náhodně z něj vybereme skladební dvojici (hranu). Tento postup opakujeme mnohokrát, přičemž si u jednotlivých hran pamatujeme četnosti jejich výskytu. Bez újmy na obecnosti můžeme také číst všechny hrany daného textu popořadě, protože v tak velké množině U, jakou jsme zvolili, je náhodnost výběru dostatečně zaručena už výběrem celého textu.

V první fázi, kterou nazveme trénovací, tedy budeme číst náhodně zvolený rozebraný text, tj. posloupnost stromů. To budou naše **trénovací data**. Čím větší množství dat získáme, tím lépe, protože tím spolehlivější bude náš model. Každou větu projdeme podle definovaného uspořádání hran ve stromu. Přitom budeme udržovat tabulku četností dvojic po sobě jdoucích

hran, tabulku četností samostatných hran a celkový počet přečtených hran.[9] Na základě těchto údajů dokážeme odhadnout pravděpodobnost libovolné hrany *h*, když víme, jaká hrana jí předcházela.

Všimněme si, že stále nepotřebujeme vědět, jak vypadá vrchol stromu a podle čeho určíme, že dva vrcholy, resp. dvě hrany, jsou stejné. V tomto smyslu jsou popisované metody univerzální: můžeme trénovat model nejen podle slovních tvarů, ale také například podle slovních druhů, mluvnických kategorií apod. Jediné omezení tkví v tom, že dotyčnou informaci, kterou prohlásíme za klíčovou pro odlišení dvou vrcholů, musíme mít k dispozici nejen při trénování, ale i později při analýze. Podrobněji se touto otázkou budeme zabývat v kapitole Kapitola 5 .

4.2. VYHLAZOVÁNÍ

Pravděpodobnostní rozdělení získané trénováním má jednu podstatnou nevýhodu. Výsledná tabulka bude velmi řídká - to znamená, že většina dvojic hran dostane nulovou pravděpodobnost. Až podle takového rozdělení budeme konstruovat syntaktický strom, budou hrany s nulovou pravděpodobností rovnou vyřazeny ze hry.

Potíž je v tom, že jen některé hrany vylučujeme právem, protože jsou v daném jazyce skutečně chybné. Mnoho dalších hran dostane nulu prostě proto, že se v našich trénovacích datech v daném kontextu nevyskytly, ačkoli obecně jejich výskyt vyloučen není. Při použití jazykového modelu se může stát, že dostaneme vedlejší informaci, která bude dotyčnou hranu ab-

[9] Je zřejmé, že funkce obou tabulek při vhodné implementaci může zastat tabulka jediná, to však na této úrovni abstrakce není podstatné.

solutně preferovat; s nulovou pravděpodobností se však taková informace nemůže uplatnit. Z toho důvodu se nulám v rozdělení chceme vyhnout a neviděným hranám přiřadit jistou velmi nízkou, avšak nenulovou hodnotu.

Jednou možností, jak toho dosáhnout, je přičíst ke všem absolutním četnostem v trénovacích datech relativně nízkou konstantu, např. 1. Bude-li tréninkový text obsahovat 10 000 hran, dostanou neznámé hrany pravděpodobnost $\frac{1}{10\,001}$. Tento přístup neklade žádné další nároky na zpracování dat, ale nijak nerozlišuje hrany, které se neobjevily, od těch, které se nikdy objevit nemohou.

Přesnější je jiná metoda, která kombinuje vícegramové rozdělení s méněgramovými. V našem konkrétním případě sledujeme výskyt hrany v kontextu jedné předcházející hrany. Chceme-li ve známém kontextu zjistit pravděpodobnost hrany, která se v trénovacích datech v tomto kontextu neobjevila, můžeme se pokusit zjistit její nepodmíněnou pravděpodobnost (bez ohledu na kontext). Mezery ve dvougramovém modelu tedy doplníme jednogramovým. Pokud ani to nepomůže a pravděpodobnost bude stále nulová, můžeme na zbylé hrany použít rovnoměrné rozdělení (odpovídá 0-gramovému modelu).

Protože součet takto získaných pravděpodobností přes všechny hrany není roven 1, musíme jednotlivým *i*-gramovým rozdělením přiřadit váhy $\lambda_i \in (0;1)$, kde $\sum \lambda_i = 1$. Pravděpodobnost hrany h_i pak určíme takto:

$$p(h_i|h_{i-1}) = \lambda_2 r(h_i|h_{i-1}) + \lambda_1 r(h_i) + \lambda_0 \frac{1}{|L|^2}$$

Tomuto způsobu odstranění nulových pravděpodobností říkáme **vyhlazování** jazykového modelu.

Zbývá zodpovědět otázku, kde vzít hodnoty jednotlivých vah λ_i. Dobrých výsledků lze dosáhnout i odhadem, přičemž bývá vhodné dodržet podmínku $\lambda_2 > \lambda_1 > \lambda_0$. Například můžeme zvolit $\lambda_2 = 0{,}99$, $\lambda_1 = 0{,}009$ a $\lambda_0 = 0{,}001$. Pokud máme k dispozici dostatek vzorových dat, můžeme iteračním algoritmem tyto hodnoty ještě více přiblížit realitě.[10]

Popíšeme zde jeden průchod algoritmu, kterým z hodnot λ_i, získaných předchozím průchodem algoritmu nebo odhadem, dostaneme nové, realističtější hodnoty λ_i'. Iterací tohoto kroku váhy konvergují k hodnotám $\overline{\lambda}_i$, které považujeme za optimální.

Algoritmus zjišťuje, nakolik se jednotlivá *i*-gramová rozdělení uplatnila při aplikaci na skutečná data. Míra uplatnění je dána jednak výchozími hodnotami vah, jednak dílčími pravděpodobnostmi, kterými se to které rozdělení podílelo na celkových pravděpodobnostech hran v textu. Pokud tedy naprostá většina dvojic hran dostává od bigramového rozdělení nulu, takže „zachraňovat situaci“ musí unigramové rozdělení svými četnostmi samostatných hran, bude toto rozdělení zvýhodněno vyšší hodnotou λ_1.

V jednom kroku algoritmu projdeme celá data, která máme pro vyhlazování vyčleněna, přičemž sčítáme následující veličiny ($|H|$ je počet výskytů všech hran v datech vyčleněných pro vyhlazování):

$$\Lambda_2 = \sum_{i=1}^{|H|} \lambda_2 r\left(h_i \middle| h_{i-1}\right),$$

[10] Jde o speciální případ tzv. E-M algoritmu (z anglického *estimation-maximation*, tj. odhad-maximalizace). Viz též [[5]].

$$\Lambda_1 = \sum_{i=1}^{|H|} \lambda_1 r(h_i),$$

$$\Lambda_0 = \sum_{i=1}^{|H|} \lambda_0 \frac{1}{|L|^2} = \lambda_0 \frac{|H|}{|L|^2}.$$

Na konci kroku zjistíme pro jednotlivá rozdělení jejich „podíl na moci“, vyjádřený novými hodnotami vah:

$$\lambda_2' = \frac{\Lambda_2}{\Lambda_2 + \Lambda_1 + \Lambda_0},$$

$$\lambda_1' = \frac{\Lambda_1}{\Lambda_2 + \Lambda_1 + \Lambda_0},$$

$$\lambda_0' = \frac{\Lambda_0}{\Lambda_2 + \Lambda_1 + \Lambda_0}.$$

Intuitivně je zřejmé, že nové váhy budou dávat lepší výsledky v „realitě“. Tu zde reprezentují použitá data *H*, proto je důležité, aby se stylisticky podobala textu, který jsme použili pro trénování. Samotná trénovací data ovšem použít nemůžeme, potom by poměry jednotlivých vah vyšly 1:0:0. Osvědčuje se proto rozdělit vzorový text na dvě části. První použijeme pro trénování a druhou, menší, vyčleníme na vyhlazování - odtud pojem **vyčleněná data**.[11]

Pokud nemáme k dispozici dostatek trénovacích dat, zůstaneme u odhadnutých hodnot pro jednotlivé váhy.

[11] V anglické literatuře *held-out data*.

4.3. ANALÝZA

Dostáváme se k vyvrcholení naší práce - ke způsobu, jakým vybudujeme syntaktický zápis věty, jestliže známe tuto větu a rozdělení pravděpodobností jednotlivých hran.

V kapitole Kapitola 3 jsme zjistili, že hledáme výsledek výrazu

$$slozit\left(\arg\max_H \left(p(S|H)\cdot p(H)\right)\right), \text{ kde}$$

$$p(H)=\sum_{i=1}^{n} p\left(h_i|h_{i-1}\right) \text{ a}$$

$$p(S|H)=0 \text{ pro špatné posloupnosti hran, 1 pro ostatní.}$$

Z uvedeného vyplývá, že hledaná posloupnost hran musí mít maximální $p(H)$ při současném zachování „správnosti“, aby $p(S|H)=1$.

Problém maximalizace $p(H)$ je analogií hledání maximální kostry grafu.[12] Máme graf $G=(V,H_G)$, kde V je množina vrcholů a H_G je množina hran, a máme definovánu funkci $w(h)$, která každé hraně přiřadí „ohodnocení“. Chceme nalézt takový strom $M=(V,H)$, aby $\sum w(h)$ byla maximální. V našem případě je G úplný graf, V je množina slov výchozí věty (plus kořen #) a H_G obsahuje všechny uspořádané dvojice vrcholů z V. Hodnotící funkci představuje pravděpodobnost $p(h)$. Úloha o kostře požaduje maximalizaci součtu, zatímco my potřebujeme co možná nejvyšší součin. Postup, který ukážeme, lze však použít v obou případech, protože pravděpo-

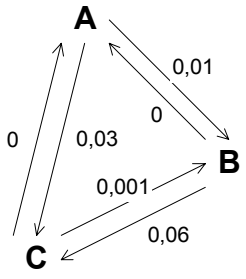
[12] Viz též [[11]], článek 7.3. a cvičení 7.B.

dobnosti jsou nezáporná čísla. (Kdyby $p(h)$ mohlo být záporné, závisel by výsledek součinu mj. na tom, jestli se záporných činitelů vyskytl lichý nebo sudý počet. U součtu nic podobného nehrozí.)

Pro hledání maximální kostry *M* použijeme **hladový algoritmus**. Začneme se stromem, obsahujícím jeden libovolný vrchol grafu *G* (například kořen #), a potom v každém kroku přidáme takovou hranu $[x;y]$, že $x \in M$, $y \notin M$ a $p([x;y])$ je nejvyšší možná. Skončíme v okamžiku, kdy nemůžeme přidat žádnou další hranu, protože strom *M* už obsahuje všechny vrcholy grafu *G*.

Dá se ukázat, že pro neorientované grafy je výsledek hladového algoritmu skutečně maximální; že je to kostra, plyne přímo ze způsobu konstrukce. Potíž je v tom, že my pracujeme s orientovanými grafy, kde nemusí platit $p([x;y]) = p([y;x])$. Následující obrázek je příkladem orientovaného grafu, na kterém námi popsaný algoritmus selže. (Graf je úplný; hrany, které nejsou zakresleny, mají pravděpodobnost libovolně blízkou nule.)

PŘÍKLAD 1:



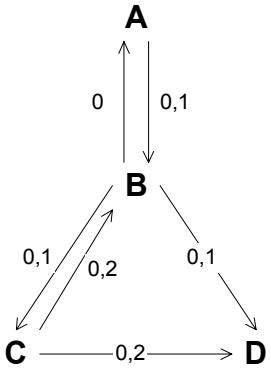
Jestliže A je kořen, potom algoritmus přidá nejdříve hranu [AC], potom hranu [AB]. Pravděpodobnost posloupnosti hran vyjde 0,0003. Vezmeme-li však hrany [AB] a [BC], dostaneme pravděpodobnost vyšší: 0,0006.

Domníváme se, že podobné případy nebudou výjimečné (jak často k nim dochází a jak moc se chyba projeví na výsledných stromech, to ukáží až praktické pokusy). Uvedeme proto ještě jednu variantu hladového algoritmu; podle zvoleného přístupu jí budeme říkat **komponentová**, zatímco dříve uvedenou verzi nazveme **sekvenční**.

Komponentový hladový algoritmus pracuje na principu slévání komponent grafu. Na začátku máme $n+1$ komponent, z nichž každá obsahuje právě jeden vrchol grafu *G*. V každém kroku pak přidáme hranu, která spojuje dvě existující komponenty v jedinou. Požadujeme tedy, aby řídící vrchol ležel v jiné komponentě než závislý a dále (pro orientované grafy) aby cílová komponenta byla napojena ve svém kořeni, tj. v takovém vrcholu, do kterého dosud nevede žádná jiná hrana. Ze všech hran, které splňují uvedené podmínky, vybereme nejpravděpodobnější. Po *n* krocích dostaneme jedinou komponentu obsahující všechny vrcholy grafu *G*, a tu prohlásíme za výslednou maximální kostru.

Pro neorientované grafy jsou obě uvedené varianty hladového algoritmu ekvivalentní a dávají stejné výsledky. U orientovaného grafu, uvedeného v příkladu 1, komponentová varianta narozdíl od sekvenční neselže: jako první vybere hranu [BC], potom hranu [AB]. Ale obecně ani tato verze nezaručuje dosažení maxima; protipříkladem může být následující graf. (Opět předpokládáme, že nezakreslené hrany mají zanedbatelnou pravděpodobnost.)

PŘÍKLAD 2:



V tomto případě komponentový algoritmus použije po řadě hrany [CB], [CD] a [BA]; správné řešení je [AB], [BC] a [CD].

Ani jedna z obou verzí hladového algoritmu tedy pro orientované grafy nezaručuje optimální výsledek. Přesto hladový algoritmus použijeme, protože neznáme žádný lepší způsob, jak maximum nalézt. Jistotu získáme pouze v případě, že probereme všechny stromy, které lze sestavit nad danou množinou vrcholů; takový postup by však měl exponenciální složitost. Hladový algoritmus naproti tomu bude mít v nejhorším případě složitost n^3. Nemáme dost pádný argument, abychom rozhodli mezi sekvenční nebo komponentovou variantou, proto vyzkoušíme obě. Posledně jmenovaná má ale podstatnou nevýhodu: v okamžiku přidávání hrany většinou neznáme hranu, která jí předchází v uspořádání definovaném funkcí *rozlozit*. Neznáme tedy kontext a nemůžeme použít podmíněné pravděpodobnosti. Proto budeme komponentovou variantu testovat pouze s jednogramovým modelem.

Nakonec musíme zaručit, aby $p(S|H) = 1$. Tj. že 1) H popisuje strom, 2) H je správně uspořádaná podle uspořádání daného funkcí *rozlozit* a 3) H odpovídá větě S. Poslední podmínka je triviálně splněna, protože při konstrukci používáme pouze slova z věty S. Obě verze hladového algoritmu

také garantují, že jejich výsledek je strom. K porušení druhé podmínky může dojít ve vícegramovém modelu, kde použité pravděpodobnosti závisí nejen na aktuální hraně, ale i na její předchůdkyni. (V jednogramovém modelu můžeme s utříděním posloupnosti vyčkat do okamžiku, kdy budeme znát všechny její členy.) Proto v běžném kroku sekvenčního algoritmu omezíme výběr na hrany, které jsou ve smyslu daného uspořádání „větší“ než naposledy přidaná hrana.

4.4. TESTOVÁNÍ

Nakonec se musíme zmínit o způsobu, jakým budeme jazykový model testovat a hodnotit jeho úspěšnost. Kiril Ribarov [[15]], který řeší naši úlohu jinými metodami, definuje chybovou funkci pro srovnání výsledků analýzy se správným řešením. Nemáme důvod nepoužít stejnou metodu — naopak shodný způsob hodnocení umožní srovnání úspěšnosti obou přístupů. Navíc máme k dispozici několik veličin z teorie informace (entropii, křížovou entropii a perplexitu), kterými budeme charakterizovat naše pravděpodobnostní rozdělení.

K testování potřebujeme další text, který už obsahuje syntaktickou strukturu a který stylisticky se podobá textu použitému pro trénování. Výchozí data tedy budeme případně dělit až na tři části: trénovací (*T*, ta bude nejrozsáhlejší), vyhlazovací (vyčleněnou, *H*) a testovací (*Z*). Z testovacích dat odstraníme syntaktické struktury a necháme je rozebrat naším programem. Potom vezmeme dvojici stromů M_1, M_2, kde první byl součástí testovacích dat a druhý z něj vzniknul odstraněním a opětovným vygenerováním syntaktické struktury. Pro každou takovou dvojici spočítáme hrany, které se vyskytly v obou stromech současně. Celkový počet shodných hran ve všech

dvojicích stromů ze *Z* vydělíme celkovým počtem všech hran v *Z* a získáme **úspěšnost**; její doplněk do jedničky pak bude **chyba**.

Jelikož nám pravděpodobnost dává ohodnocení každého stromu, můžeme také definovat vzdálenost, nebo lépe **rozdíl dvou stromů**. Je-li $\overline{p}(M)$ ohodnocení stromu *M*, potom rozdíl

$$d(M_1, M_2) = \overline{p}(M_1) - \overline{p}(M_2)$$

Abychom mohli porovnávat ohodnocení různě velkých stromů, nemůžeme za $\overline{p}(M)$ dosadit přímo součin pravděpodobností hran stromu *M*, musíme ho normalizovat vzhledem k počtu hran *n*. Získáme tak vlastně geometrický průměr pravděpodobností hran:

$$\overline{p}(M) = \prod_{i=1}^{n} \sqrt[n]{p(h_i)}$$

Pravděpodobnostní rozdělení můžeme charakterizovat veličinou zvanou **entropie**, kterou chápeme jako míru neurčitosti. Čím vyšší entropii má jazykový model, tím obtížněji se na jeho základě předpovídá budoucí text. Nejvyšší možná entropie má hodnotu $\log|L|^2$ a odpovídá rovnoměrnému rozdělení. Naopak nulovou entropii mají rozdělení, kde jeden jev je jistý a všechny ostatní nemožné. Entropie *H* pokusu α je definována následujícím vztahem:[13]

$$H(\alpha) = -\sum_{i=1}^{|L|^2} p(h_i) \log p(h_i)$$

[13] Viz též [[10]], kapitola II.

Nezáleží na tom, při jakém základu počítáme logaritmus; my budeme vždy používat logaritmus při základu 2, takže výsledek vyjde v bitech. To souvisí s jinou interpretací entropie, která říká, kolik bitů potřebujeme na zakódování libovolné hrany z T.

Jestliže pracujeme s podmíněnými pravděpodobnostmi, hodnota entropie by se měla snížit, protože znalost kontextu zvyšuje naši rozhodovací schopnost. Hovoříme pak o **podmíněné entropii** $H(\alpha|\beta)$.

$$H(\alpha|\beta) = \sum_{i=1}^{|L|^2} p(h_i) H(\alpha|h_i), \text{ tj.}$$

$$H = -\sum_{i=1}^{|L|^2} \sum_{j=1}^{|L|^2} p(h_i) p(h_j|h_i) \log p(h_j|h_i)$$

Pro porovnávání natrénovaného rozdělení s testovacími daty slouží tzv. **křížová entropie**. Vztah pro ni se podobá definici entropie, ale do prvního činitele místo pravděpodobnosti dvojice hran dosazujeme relativní četnost této dvojice v testovacích datech — $r_Z(h_i \wedge h_j)$:

$$\bar{H} = -\sum_{i=1}^{|L|^2} \sum_{j=1}^{|L|^2} r_Z(h_i \wedge h_j) \log p(h_j|h_i)$$

Konečně poslední veličinou, kterou budeme sledovat, je **perplexita**, definovaná vztahem

$$P = 2^H,$$

kde H je entropie. Perplexita nám říká, kolik různých hran bychom potřebovali, aby při rovnoměrném rozdělení pravděpodobností vyšla stejná entropie. To znamená, že při rovnoměrném rozdělení bychom měli jen

$|L|^2 = P$ různých hran. Perplexita nemůže vyjít větší než počet různých hran v trénovacích datech.

K a p i t o l a 5

5. IMPLEMENTACE A VÝSLEDKY

Nebudeme zde rozebírat všechny implementační detaily, neboť je z hlediska této práce nepovažujeme za zajímavé. Poznamenejme pouze, že v případě praktického nasazení je třeba implementaci tabulky pravděpodobností dobře promyslet, protože bude muset zvládnout obrovské množství dat. Co však v této kapitole chceme probrat, jsou některé parametry jazykového modelu, jejichž specifikaci jsme dosud odkládali.

Popsané metody nám na několika místech ponechávají volnost ve volbě konkrétních postupů či hodnot. Při výběru variant, které prezentujeme v následujících odstavcích, jsme byli od začátku vedeni a omezováni množstvím i tvarem dat, která máme k dispozici pro trénování.

Data pocházejí z Ústavu českého národního korpusu Filozofické fakulty UK. Korpus obsahuje v současné době texty nejrůznějšího původu a stylu v rozsahu 50 až 70 miliónů slov, přičemž část textů již byla syntakticky analyzována v rámci projektu *Formální reprezentace jazykových struktur*[14]. Spolu s daty jsme z tohoto projektu převzali i jejich jednotný formát, mj. atributy vrcholu a stavbu morfologické značky. Část, která je v současné době rozebrána, představuje pouhý zlomek všech korpusových dat. K dispozici máme 1150 vět, přičemž v jiných aplikacích pravděpodob-

[14] Grantový projekt GA ČR, vedený Evou Hajičovou, probíhá ve spolupráci Ústavu formální a aplikované lingvistiky MFF UK a Ústavu teoretické a komputační lingvistiky FF UK.

nostního modelování jazyka vedou k uspokojivým výsledkům řádově desetitisíce. Musíme se proto snažit volit takové metody, které jsou na množství dat méně náročné.

Na prvním místě jde o volbu velikosti kontextu, tj. gramu. Hodláme vyzkoušet 1- i 2-gramový model, budeme tedy uvažovat kontext nejvýše jedné předcházející hrany. Nemůžeme také opomenout vyhlazování: Zkombinujeme dvougramové rozdělení s jednogramovým a rovnoměrným, přičemž váhy jednotlivých rozdělení stanovíme odhadem. Položíme $\lambda_2 = 0{,}99$, $\lambda_1 = 0{,}009$ a $\lambda_0 = 0{,}001$ pro bigram, resp. $\lambda_1 = 0{,}99$ a $\lambda_0 = 0{,}01$ při testování unigramu.

Dále musíme určit způsob, jakým budou uspořádány hrany v syntaktickém stromu, tj. definovat funkce *rozlozit* a *slozit*. Toto uspořádání významně ovlivní volbu kontextu ve dvourozměrném rozdělení, a tím i přesnost získaného modelu. Nedovedeme však předem říci, které uspořádání bude dávat lepší či horší výsledky, proto vyzkoušíme dva způsoby, které se nám jeví nejpřirozenější: hrany budou uspořádány podle svých závislých vrcholů v pořadí daném procházením stromu do hloubky, resp. do šířky. Vrcholy na téže úrovni od kořene uspořádáme podle jejich pořadí (resp. pořadí jimi reprezentovaných slov) ve větě.

Konečně nejdůležitější rozhodnutí tkví v tom, jak přesně bude vypadat vrchol stromu a podle kterých rysů budeme při trénování a při analýze určovat, že dva vrcholy, resp. dvě hrany, jsou ekvivalentní.

5.1. VRCHOL STROMU

Vrchol stromu obsahuje řadu informací o konkrétním slovu, přičemž pouze některé atributy jsou využitelné pro naši syntaktickou analýzu. Tvar vrcholu je předem dán použitými daty (viz úvod této kapitoly).

Vrchol lze popsat jako *strukturu rysů* (*feature structure*), která obsahuje až 12 atributů. Čtyři z nich pro nás mohou být zajímavé:

- Slovní tvar, tak jak se vyskytl ve větě. Je důležitý především pro rekonstrukci původní věty.

- Pořadí slova ve větě. Rovněž slouží k rekonstrukci původní věty.

- Slovníkové heslo, ze kterého slovní tvar mohl vzniknout skloňováním nebo časováním. Nezřídka máme hned několik možností, z čeho mohl být tvar odvozen, a bez znalosti kontextu se neumíme mezi nimi rozhodnout. Lze si představit další jazykový model, který bude hledat nejpravděpodobnější alternativu; v datech, která máme k dispozici, jsou pro každý vrchol vyjmenovány všechny možnosti.

- Morfologická značka (*tag*), která popisuje mluvnické kategorie daného slovního tvaru (rod, číslo, pád, osobu, čas…). I značky mívají několik variant, přičemž ke každé možnosti můžeme nalézt takovou variantu hesla, která v kombinaci se značkou dává slovní tvar ve vrcholu uvedený. Získat správnou variantu morfologické značky ke slovnímu tvaru lze také pomocí pravděpodobnostního modelu[15]; my zatím bohužel nemáme takto zjednoznačněná data k dispozici.

[15] Příkladem takového automatického značkování (*tagování*) může být [[7]].

Důležitá je otázka, jak poznáme, že dva vrcholy jsou shodné. Odpověď na ni zásadním způsobem ovlivní počet parametrů jazykového modelu, tj. počet různých dvojic hran, které se mohou vyskytnout v datech. Chtěli bychom každý vrchol reprezentovat takovým atributem nebo skupinou atributů, která nabývá relativně nízkého počtu hodnot, a o které se současně domníváme, že je nějak svázána se závislostí vrcholu na jiném větném členu. Přitom tiše předpokládáme, že hodnoty dotyčných atributů budeme mít k dispozici i v okamžiku analýzy.

Z uvedených položek nejdříve vyloučíme pořadí, neboť intuitivně tušíme, že nemá valný vliv na skladbu věty. Spíše než absolutní umístění by nás mohla zajímat vzdálenost dvou slov od sebe, ale ve stávajícím modelu ji neumíme využít.

Zbývající atributy porovnáme podle potenciálního počtu parametrů, které přinesou. Různých tvarů je v češtině asi 6 000 000, hesel zhruba 40 000 a značek jen necelých 1800. Protože nepracujeme s vrcholy, ale s hranami, resp. s dvojicemi hran, musíme k získání počtu parametrů to které číslo ještě umocnit na druhou, resp. na čtvrtou. Odtud je zřejmé, že při našem malém množství dat musíme model trénovat podle značek.

Upozorněme zároveň, že toto rozhodnutí má i výrazné nevýhody. Kromě nižší přesnosti je to zejména fakt, že jeden vrchol může obsahovat několik variant morfologických značek. Taková nejednoznačnost není ve srovnatelných aplikacích obvyklá a v úvahách předchozích kapitol jsme s ní proto nepočítali. Jejím řešením se budeme podrobněji zabývat níže v této kapitole.

5.2. MORFOLOGICKÉ ZNAČKY

Vzhledem k tomu, že značky budou jediným údajem, na který redukujeme vrchol stromu, podíváme se podrobněji na jejich tvar.[16]

Morfologická značka je posloupnost velkých písmen a číslic (výjimečně může obsahovat i pomlčku), jejíž první písmeno udává slovní druh a na dalších místech jsou popsány mluvnické kategorie, případně poddruh slova. Jak je ovšem vidět z následujících tabulek, hodnoty značek lze pohodlně zakódovat do 4-bytových čísel, čímž se správa tabulky četností usnadní.

Jméno kategorie	Jméno proměnné	Hodnota	Vysvětlení
slovní druh	**k**	**N**	podstatné jméno
		A	přídavné jméno
		P	zájmeno
		C	číslovka
		V	sloveso
		O	příslovce
		R	předložka
		J	spojka
		T	částice
		I	citoslovce
		Z	zvláštní (např. interpunkce)
		X	nerozlišitelný
rod	**g**	**M**	mužský životný
		I	mužský neživotný
		F	ženský
		N	střední
		Y	M nebo I
		T	I nebo F
		W	I nebo N
		H	F nebo N
		Q	F nebo N ve zvláštních případech
		Z	M, I nebo N
		X	nerozlišitelný

[16] V [[7]] i [[15]] je uveden systém značek pro češtinu, který se poněkud liší od námi používaných dat.

číslo	**n**	**S**	jednotné
		D	dvojné
		P	množné
		X	nerozlišitelné
pád	**c**	**1 až 7**	1. až 7.
		X	nerozlišitelný
stupeň	**d**	**1 až 3**	1. až 3.
zápor	**a**	**A**	kladné
		N	záporné
osoba	**p**	**1 až 3**	1. až 3.
	f	**1 až 2**	pouze 1. a 2.
čas	**t**	**P**	přítomný
		M	minulý
		F	budoucí
slovesný rod	**s**	**A**	činný
		P	trpný
způsob	**m**	**N**	oznamovací
		R	rozkazovací
		C	podmiňovací (jen některá slovesa)

Za značkou se může vyskytnout ještě informace o tom, že daný tvar se používá pouze při konkrétním stylu projevu:

Přípona	**Vysvětlení**
—1	jiná varianta, méně častá
—2	jiná varianta, velmi málo častá, archaická nebo knižní
—3	velmi archaické, popř. hovorové
—5	hovorové, tolerováno ve veřejném mluveném i psaném projevu
—6	hovorové, většinou mluvené, nemělo by se používat ve veřejném projevu
—7	hovorové, jako 6, ale méně preferováno mluvčími
—9	zvláštní použití (např. po jistých předložkách)

Následující tabulka přináší orientační výčet povolených kombinací atributů. Jiné značky než zde uvedené nejsou povoleny. Neplatí však, že všechny kombinace, které lze z tabulky odvodit, skutečně existují.

Některá písmena neurčují mluvnické kategorie, ale blíže specifikují třídu slov v rámci slovního druhu.

Značka	Vysvětlení, poznámka
Ngnca	podstatná jména (jen výjimečně a=N)
Agncda	přídavná jména
ACgn[c]a	jmenný tvar přídavného jména (rád, zdráv), pouze: **ACYSa**, **ACQXa** (za **ACFSa**, **ACNPa** nebo **ACYS4a**), **ACNSa**, **ACMPa**, **ACTPa**, **ACFS4a**
ASMgnc	přídavná jména přivlastňovací mužská (otcův)
ASFgnc	přídavná jména přivlastňovací ženská (matčin)
AVGgnca	přídavná jména slovesná přítomná (dělající)
AVVgnca	přídavná jména slovesná minulá (dodělavší)
A1gn	(svůj, nesvůj, tentam - jsou-li použita jako A)
PPfnc	zájmena osobní (já, ty)
PPfCnc	zájmena osobní zkrácená (mi, mě)
PP3gnc	zájmena osobní samostatná (on, ona, ono, jeho, jemu…)
PP3Cgnc	zájmena osobní zkrácená (ho, mu)
PP3Rgnc	zájmena osobní po předložce (něho)
PPD	zájmena osobní s předložkou (naň, proň)
PPA2S1	zájmeno osobní se slovesem jsi (tys)
PRnc	zájmeno osobní zvratné samostatné (sebe, sobě)
PRCnc	zájmeno osobní zvratné zkrácené (se, si; n=X)
PRACncp	zájmeno osobní zvratné se slovesem jsi (ses, sis)
PSfngnc	zájmena přivlastňovací (můj, tvůj); první číslo je „vnitřní“ (můj/náš), druhé „předmětné“ (můj/moji)
PS3gngnc	zájmena přivlastňovací (jeho, její), první rod a číslo jsou vnitřní
PRSgnc	zájmeno přivlastňovací zvratné (svůj)
PDgnc	zájmena ukazovací (ten, tento, tamten)
PLgnc	zájmeno (sám, všechen)
PQFgnc	zájmena tázací nebo vztažná s rodem a číslem (jaký, který)
PQKgc	zájmena tázací nebo vztažná bez čísla (kdo, kdož; g=M)
PQCc	zájmena tázací nebo vztažná bez rodu a čísla (co)
PQD	zájmena tázací nebo vztažná s předložkou (nač, oč)
PQAcp	zájmena tázací nebo vztažná se slovesem jsi (kdos, cos)
PAEgnc	zájmena vztažná (jenž)
PAERgnc	zájmena vztažná po předložce (něhož)
PEc	zájmena vztažná bez rodu a čísla (což)
PSEgngnc	zájmena vztažná přivlastňovací (jehož)
PIFgnc	zájmena neurčitá (nějaký)
PIc	zájmena neurčitá bez rodu a čísla (něco)
PNFgnc	zájmena záporná (nijaký)
PNc	zájmena záporná bez rodu a čísla (nic)
CGgnc	číslovky základní (jeden, dva, tři, čtyři)
CBnc	číslovky základní bez rodu (pět a více)
CFgnc	číslovky - zlomky (g=F)
CRgnc	číslovky řadové (první, třetí, pátý)
CDgnc **CD1gnc** **CD2nc** **CDJnc**	číslovky druhové (dvojí, trojí, paterý)
CQFgnc	číslovky tázací nebo vztažné (kolikátý)

CQc	číslovky tázací bez rodu a čísla (kolik)
CIFgnc	číslovky neurčité (několikátý)
CIc	číslovky neurčité bez rodu a čísla (tolik)
CM	číslovky násobné (-krát)
CQM	číslovky násobné tázací (kolikrát)
CIM	číslovky násobné neurčité (několikrát)
CX	číslovky římské
VFa	slovesa, infinitiv (dělat)
VPnpa	slovesa, t=P (dělá)
VPEnpa	slovesa, t=P, s „nebť“ (děláť)
VRgna	slovesa, t=M (dělal)
VREgna	slovesa, t=M, s „nebť“ (dělalť)
VRAgnpa	slovesa, t=M, se „jsi“ (dělals)
VSgn[c]a	slovesa, s=P (udělán)), pouze: **VSYSa**, **VSQXa** (za **VSFSa**, **VSNPa** nebo **VSYS4a**), **VSNSa**, **VSMPa**, **VSTPa**, **VSFS4a**
VSAgnpa	slovesa, s=P, se „jsi“ (uděláns)
VUnpa	slovesa, t=F (udělám)
VUEnpa	slovesa, t=F, s „nebť“ (udělámť)
VMnpa	slovesa, m=R (dělej)
VCnp	slovesa, m=C (by)
VGnga	slovesa, přechodník, t=P (dělaje)
VVnga	slovesa, přechodník, t=M (dělav)
DGda	příslovce
DB	příslovce bez stupně a záporu
Rc	předložky
RVc	předložky s přidanou samohláskou (ke, ku)
RF	první část složeného předložkového výrazu (vzhledem)
JE	spojky souřadící, stejná úroveň koordinace
JC	spojky souřadící matematické (krát, děleno)
JS	spojky podřadící
JVnp	spojky podřadící slovesné (aby, kdyby)
T	částice (ano, ne, ať, kéž, nechť)
I	citoslovce
HYPH	první část složeného výrazu spojeného pomlčkou
ABBR[k]	zkratka daného slovního druhu
NOMORPH	neanalyzovaná slova, např. citace v cizím jazyku
ZNUM	číslo
ZIP	interpunkce
ZSB	identifikace věty (kořen stromu)

Jak jsme se již zmínili výše, tabulka nepostihuje všechna omezení, která jsou ve skutečnosti na hodnoty konkrétních atributů v konkrétních značkách kladena. V praxi se tedy nevyskytuje všech 48 308 značek, které ta-

bulka připouští, ale pouhých 1786.[17] To je naše hodnota čísla $|L|$, které jsme používali v předchozích kapitolách. Potom dostáváme $3{,}19.10^6$ různých hran.

5.3. TRÉNOVÁNÍ PODLE HRAN S VÍCE VARIANTAMI MORFOLOGICKÝCH ZNAČEK

Při trénování si z každé hrany zapamatujeme značku řídícího a značku závislého vrcholu. Vyskytne-li se v trénovacích datech hrana, jejíž řídící vrchol obsahuje i a závislý vrchol obsahuje j variant morfologických značek[18], musíme si zapamatovat všech $i \cdot j$ možných kombinací, které z nich lze vytvořit. Současně však musíme mít na paměti, že tyto kombinace dohromady reprezentují jediný výskyt hrany v trénovacích datech. Aby byl zachován poměr relativních četností vůči ostatním hranám, musíme daný výskyt rovnoměrně rozdělit mezi jednotlivé kombinace značek, které se v hraně vyskytly.

PŘÍKLAD: V trénovacích datech se vyskytne hrana s řídícím vrcholem `[solí, NFS7A|NFP2A|VPX3A]` a závislým vrcholem `[bílou, AFS41A|AFS71A]`. Tato hrana ve skutečnosti odpovídá právě jedné z kombinací `NFS7A-AFS41A`, `NFS7A-AFS71A`, `NFP2A-AFS41A`, `NFP2A-AFS71A`, `VPX3A-AFS41A` a `VPX3A-AFS71A`. Místo abychom zvýšili četnost některé hrany v tabulce o 1, zvýšíme četnost každé z uvedených dvojic o $\frac{1}{6}$.

---

[17] Tabulka neuvažuje přípony `–1` až `–9`. Pokud je vezmeme v úvahu, vzroste počet možných značek na 2578.

[18] V trénovacích datech měl každý vrchol průměrně více než 2 morfologické značky.

Při větném rozboru pak budeme hledat pravděpodobnost, že se ve větě vyskytla některá z možných kombinací, tj. pravděpodobnost disjunkce jevů:

$$p(h_1 \vee \mathrm{K} \vee h_k) = p(h_1) + \mathrm{K} + p(h_k) - p(h_1 \wedge \mathrm{K} \wedge h_k)$$

Poslední člen (pravděpodobnost průniku jevů) je nulový, jestliže se žádná kombinace značek nevyskytne v jedné hraně více než jednou. Ve skutečných datech sice k opakování značek dochází, avšak z důvodů, které pro naši úlohu nejsou relevantní. Proto postačí, když při čtení atributů vrcholu zaevidujeme každou značku nejvýše jednou. Potom se pravděpodobnost hrany bude rovnat součtu pravděpodobností jednotlivých kombinací.

Obdobně musíme postupovat i u hran, které tvoří kontext. Jestliže aktuální hrana h_i obsahuje kombinace $e_{i,1}$ až e_{i,k_i} a kontext h_{i-1} obsahuje kombinace $e_{i-1,1}$ až $e_{i-1,k_{i-1}}$, potom při trénování zaregistrujeme každou kombinaci z aktuální hrany postupně v kontextu všech kombinací z kontextové hrany, vždy s četností $\frac{1}{k_i \cdot k_{i-1}}$.

Při analýze je situace komplikovanější. Pravděpodobnost hrany odhadujeme jako četnost dvojice „hrana plus kontext“, dělenou četností kontextu:

$$p(h_i | h_{i-1}) = \frac{c(h_i \wedge h_{i-1})}{c(h_{i-1})}$$

Protože dotyčné četnosti jsou součtem četností dílčích kombinací značek, dostáváme:

$$p\left(h_i \middle| h_{i-1}\right) = \frac{\sum_{j=1}^{k_i} \sum_{l=1}^{k_{i-1}} c\left(e_{i,j} \wedge e_{i-1,l}\right)}{\sum_{l=1}^{k_{i-1}} c\left(e_{i-1,l}\right)}$$

Z předchozího výkladu vysvítá, do jaké míry nejednoznačné morfologické značky komplikují algoritmus. Časová složitost se nebezpečně blíží prahu únosnosti zejména u dvougramového rozdělení, neboť pro každou hranu musíme projít všechny její kombinace spolu se všemi kombinacemi hrany z kontextu. Jelikož některá slova můžou obsahovat i 27 možných variant morfologické značky, může se nám stát, že zjištění jediné pravděpodobnosti si vyžádá přes půl miliónu přístupů do tabulky!

5.4. VÝSLEDKY TESTOVÁNÍ

Jak jsme uvedli v popisu pokusných dat na začátku této kapitoly, máme k dispozici 1150 vět. Sto z nich jsme rezervovali pro testování, zbytek jsme použili jako trénovací množinu. Z takto rozdělených dat jsme vybírali různě velké vzorky pro jednotlivé pokusy.

TABULKA 1: V této tabulce přinášíme základní charakteristiky jednotlivých natrénovaných modelů. Hodnota *T* udává počet trénovacích vět. Počet parametrů udává počet různých *i*-tic hran, které se vyskytly v trénovacích datech při *i*-gramovém modelování. Tento počet je vyšší než počet jednotlivých hran, a to i u jednogramového modelu, protože některé hrany se skládaly z více kombinací morfologických značek. Perplexita v posledním sloupci koresponduje s počtem hran ve druhém sloupci (viz též interpretaci perplexity v kapitole Kapitola 4).

Tabulka entropií	**počet hran**	**počet parametrů**	**entropie**	**perplexita**
unigram, T=50	684	3127	9,43	689
unigram, T=250	3960	10 305	10,64	1599
unigram, T=1050	18 015	23 099	11,12	2227
bigram do hloubky, T=50	684	78 026	2,51	6
bigram do hloubky, T=250	3960	439 964	3,64	13
bigram do hloubky, T=1050	18 015	1 901 858		
bigram do šířky, T=50	684	51 775	2,70	7
bigram do šířky, T=250	3960	456 723	3,88	15
bigram do šířky, T=1050	18 015	1 943 864		

TABULKA 2: Úspěšnost měřená podle značek. Za správnou závislostní hranu se považuje taková, jejíž řídící i závislý vrchol mají stejnou značku jako některá hrana původního stromu. Jestliže jeden nebo oba vrcholy připouštějí více variant značek, započte se pouze poměrný počet správných hran, připadající na správné kombinace značek. Takto zjištěná úspěšnost by měla být srovnatelná s výsledky práce [[15]]. Hodnota *T* opět udává počet trénovacích vět, *Z* je počet testovacích vět.

Tabulka úspěšnosti měřené podle značek	**T = 50 Z = 5**	**T = 250 Z = 25**	**T = 1050 Z = 100**
unigram, sekvenční algoritmus	29 %	37 %	36 %
unigram, komponentový algoritmus	28 %	33 %	38 %
bigram do hloubky	31 %	29 %	36 %
bigram do šířky	27 %	30 %	34 %

TABULKA 3: Úspěšnost měřená podle slov. Za správnou hranu se považuje taková, jejíž řídící i závislý vrchol reprezentují totéž slovo. Pokud se ve větě vyskytlo stejné slovo dvakrát, musí odpovídat i pořadí výskytu (slovosled). Takto zjištěná úspěšnost nemůže být vyšší než při porovnávání značek. Lépe však vystihuje to, čím je naše snažení motivováno: správnost rozboru vět.

Tabulka úspěšnosti měřené podle slov	**T = 50 Z = 5**	**T = 250 Z = 25**	**T = 1050 Z = 100**
unigram, sekvenční algoritmus	25 %	32 %	31 %
unigram, komponentový algoritmus	25 %	27 %	32 %
bigram do hloubky	29 %	22 %	30 %
bigram do šířky	23 %	23 %	27 %

TABULKA 4: Křížová entropie. Čím vyšší hodnota byla zaznamenána, tím větší počet prvků testovacích dat nebyl znám z dat trénovacích.

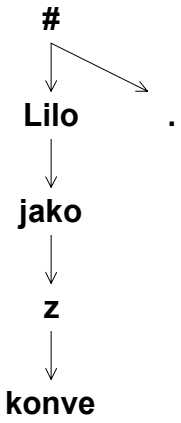
Tabulka křížových entropií	**T = 50 Z = 5**	**T = 250 Z = 25**	**T = 1050 Z = 100**
unigram	21,68	14,84	12,60
bigram do hloubky	21,98	10,81	
bigram do šířky	21,45	11,11	

Jak je vidět, výsledky nejsou příliš povzbuzující. Co z nich můžeme zjistit? Především to, že sekvenční i komponentová varianta hladového algoritmu jsou v podstatě rovnocenné, pokud jde o úspěšnost. Avšak komponentová verze nespolupracuje s jiným než jednogramovým rozdělením a její výpočet je časově náročnější.

Překvapením je nižší úspěšnost bigramů ve srovnání s unigramy. Mohlo by to znamenat, že kontext hrany, tak jak jsme ho definovali, má příliš malý vliv na výskyt této hrany; do popředí pak vystupuje nevýhoda bigramu, spočívající v omezení nabídky hran v daném kroku jejich uspořádáním do posloupnosti.

Srovnáme-li úspěšnost pravděpodobnostního modelu s transformačním přístupem Kirila Ribarova [[15]], je naše metoda nejen horší než Ribarovovy konečné výsledky, ale, což je zarážející, nedosahuje ani úspěšnosti tzv. *počátečního závorkování*, tedy takového stromu, kde bez ohledu na obsah každé slovo závisí na svém předchůdci ve větě, pouze poslední slovo (obvykle tečka za větou) závisí opět na kořeni. Příslušný strom pak připomíná

obrácenou jedničku: například věta „Lilo jako z konve.“ má zápis `#(Lilo(jako(z(konve))),.)`.



Jak ovšem vyplývá z následující tabulky, úspěšnost jedničkových stromů pro námi použitá data je výrazně nižší nejen oproti [[15]], ale nepřevyšuje ani úspěšnost pravděpodobnostního modelu.

TABULKA 5: Srovnání značkové úspěšnosti s jedničkovými stromy. V prvním sloupci je úspěšnost počátečního závorkování podle [[15]], ve druhém je tatáž hodnota naměřená na datech, z nichž vycházejí výsledky předchozích tabulek. Ve třetím sloupci je nejlepší výsledek naší metody a ve čtvrtém je nejlepší výsledek [[15]].

Tabulka srovnání úspěšností s jedničkovými stromy	**úz1R**	**úz1**	**úz**	**úzR**
Z = 5	44 %	34 %		
Z = 25	45 %	35 %	38 %	69 %
Z = 100		34 %		

Důležitá je otázka, zda se na chybovosti naší metody podílejí větší měrou ústupky, které jsme učinili při budování modelu, nebo nedokonalé hledání nejpravděpodobnějšího stromu (hladový algoritmus). Provedeme tedy ještě dvě další pozorování. Nejdříve zjistíme, zda „správné“ stromy (tj. předlohy

z testovacích dat) dostávají v daném modelu vyšší pravděpodobnost než námi vygenerované stromy chybné. A posléze se podíváme, zda se úspěšnost zvýší, když data pro testování přímo vybereme z trénovací množiny (nebude-li tomu tak, potom za chabé výsledky odpovídá hladový algoritmus).

TABULKA 6: Průměrný rozdíl mezi námi vygenerovaným stromem M a předlohou z testovacích dat M_0, resp. počátečním závorkováním M_1. Rozdíl jsme definovali v kapitole Kapitola 4 : kladné číslo znamená, že náš strom je pravděpodobnější. Čísla n a n_0 resp. n_1 označují po řadě počet testovacích vět a počet případů, kdy náš strom dostal vyšší pravděpodobnost než předloha, resp. počáteční závorkování (jedničkový strom).

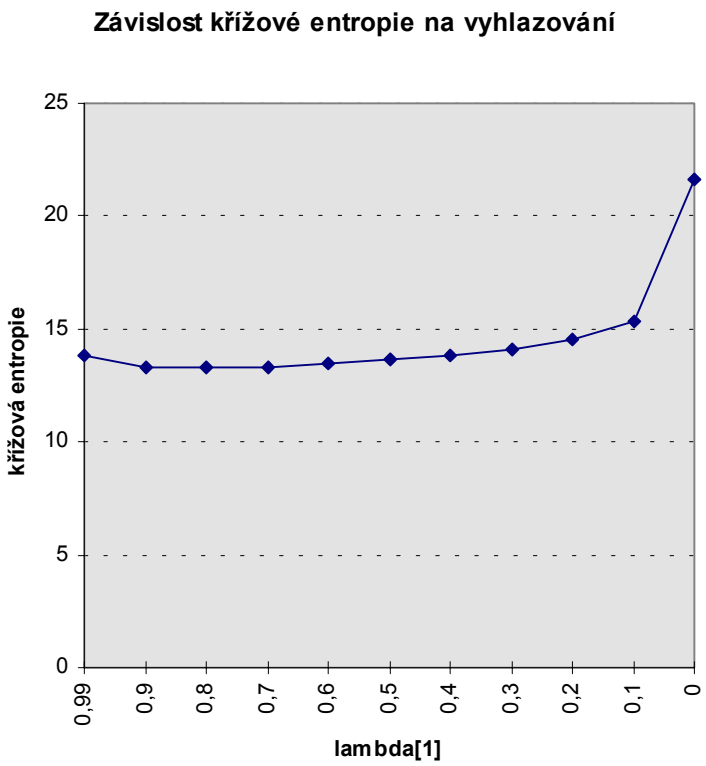
Tabulka průměrných rozdílů stromů	n	n_0	n_1	$d(M,M_0)$	$d(M,M_1)$
unigram, T = 50	5	5	5	$+6{,}0.10^{-3}$	$+6{,}3.10^{-3}$
unigram, T = 250	25	22	24	$+2{,}9.10^{-3}$	$+5{,}8.10^{-3}$
unigram, T = 1050	100	89	91	$+2{,}8.10^{-3}$	$+6{,}0.10^{-3}$
bigram do hloubky, T = 50	5	5	5	$+1{,}8.10^{-3}$	$+1{,}8.10^{-3}$
bigram do šířky, T = 50	5	5	5	$+2{,}0.10^{-3}$	$+2{,}1.10^{-3}$

TABULKA 7: Výsledky testování na trénovacích datech. Za normálních okolností nesmí testovací data být podmnožinou trénovacích, protože takový test by nebyl objektivní. Nám jde však právě o to, srovnat výsledky trénovacích dat s výše uvedenými testy.

Tabulka výsledků testu na trénovacích datech	**značková úspěšnost**	**slovní úspěšnost**	**křížová entropie**
unigram, T = 50, Z = 5	53 %	45 %	9,71
unigram, T = 250, Z = 25	43 %	37 %	10,73
unigram, T = 1050, Z = 100	43 %	39 %	11,19
bigram do hloubky, T = 50, Z = 5	50 %	43 %	0,24
bigram do šířky, T = 50, Z = 5	50 %	39 %	0,57

TABULKA 8: Posledním z „objasňovacích“ experimentů jsme se pokusili zjistit, zda problém nízké úspěšnosti netkví v tom, že vyhlazovací váhy λ

jsme pouze odhadli. Pokus jsme provedli s jednogramovým modelem, natrénovaným na 250 větách; jako obvykle jsme testovali vzorek 25 vět. Přitom jsme postupně snižovali hodnotu λ_1 a odpovídajícím způsobem zvyšovali λ_0. Ukázalo se, že hodnota tohoto parametru má jen minimální vliv na křížovou entropii; na počet chyb neměla až do hodnot velmi blízkých nule vliv žádný. Z toho vyplývá, že ani jemnější určení tohoto parametru (pomocí EM algoritmu - viz kapitolu Kapitola 4) úspěšnost daného modelu nezlepší.



K a p i t o l a 6

6. ZÁVĚR

S daty, která jsou v současnosti k dispozici, se ani jedna z vyzkoušených variant pravděpodobnostního modelu neukazuje jako nejlepší metoda pro syntaktickou analýzu české věty. Provedená pozorování ukazují spíše na nedostatky v modelu samotném (rozdělení pravděpodobností) než na špatné hledání (hladový algoritmus).

Toto zjištění ovšem neznamená, že musíme pravděpodobnostní přístup k syntaktické analýze zcela zavrhnout. Na naší cestě jsme provedli různá zjednodušení, dílem motivovaná snahou snížit časovou a prostorovou náročnost výpočtu, dílem vynucená právě množstvím a povahou dat. Shrňme tedy nakonec body, ve kterých vidíme možné zlepšení, a které by tudíž mohly být předmětem dalšího zkoumání. (Pořadí bodů nemusí nutně odpovídat jejich důležitosti.)

1. Použitý model ignoruje závislostní vztahy mezi lexikálními jednotkami (využívá pouze vztahy mezi morfologickými značkami). Například čárka ve větě je označena stejně jako jakákoli jiná interpunkce, ačkoli zcela zřejmě se syntakticky chová jinak než například tečka na konci věty. Toto zjednodušení bylo nutné vzhledem k malému množství použitelných trénovacích dat. „Ideální“ model by měl vzít v úvahu kombinaci vztahů lexikálních jednotek i morfologických značek mezi řídícím a závislým uzlem a podle možností využít alespoň bigramů.
2. Fakt, že jeden vrchol může mít více variant morfologické značky, je pravděpodobně velkým zdrojem nepřesností. Například slovní tvar „V“

může být nejen předložka na začátku věty, ale i římská číslovka 5; v našem modelu však obě z toho vyvozené skupiny hran získají stejnou váhu. Vad tohoto druhu lze jmenovat celou řadu. Velmi často se z tvaru jmen dá usuzovat hned na několik kombinací čísel a pádů, případně rodů, ve kterých se dotyčné slovo může nacházet. Přitom právě tyto atributy hrají významnou roli při určování závislostních vztahů (shoda u přívlastků, shoda přísudku s podmětem). Proto se domníváme, že až bude k dispozici jednoznačná morfologická analýza slov v datech, přesnost výpočtu se zvýší. Současně by měly klesnout výpočetní nároky algoritmu. Už dříve jsme poukázali na přímo katastrofální vliv, který má nejednoznačnost značek na časovou složitost analýzy i trénování. Po disambiguaci by však měly klesnout i požadavky na prostor, protože z rozdělení pravděpodobností by zmizelo velké množství zcela nesmyslných hran.

3. V některých případech bychom mohli použít méně, v jiných více specializované značky. Např. rozlišení jednotlivých druhů zájmen by na syntaxi nemělo mít velký vliv, důležitý je především jejich rod, číslo a pád. Na druhé straně je velký rozdíl mezi čárkou rozdělující věty v souvětí a tečkou na konci věty, v námi používané sadě značek je však oběma přisouzen kód `ZIP`. (viz též bod 1).
4. Váhy pro vyhlazení modelu jsme stanovili odhadem. Jak jsme uvedli v komentáři k tabulce 8, v současných podmínkách nemá toto zjednodušení vliv na úspěšnost modelu. Určité výkyvy křížové entropie, prezentované v téže tabulce, však signalizují, že při řádově větším množství dat můžeme očekávat zesílení vlivu vyhlazovacích vah λ. Potom by mohlo být dosaženo jistého zlepšení, pokud by váhy byly „natrénovány“ algoritmem popsaným v kapitole Kapitola 4 .
5. V modelu jsme nevyužili všechny informace, které o větě máme. Za vhodné považujeme zejména uvážit slovosled, tj. minimalizovat počet

hran, které porušují podmínku tzv. projektivity[19], a začlenit vhodným způsobem i vzdálenost řídícího a závislého vrcholu ve větě.

6. V práci [[15]] nebyl strom budován od začátku. Místo toho byla věta deterministicky uspořádána do výchozího stromu (tzv. *počáteční závorkování*) a ten byl při analýze měněn. Úspěšnost je v takovém případě zdola omezena úspěšností počátečního závorkování, která, jak jsme se sami přesvědčili, není zanedbatelná. Bylo by určitě zajímavé odpovídajícím způsobem upravit náš algoritmus pro analýzu a výsledek porovnat s výsledky prezentovanými v této práci.
7. Trénovací data obsahovala chyby — rozuměj chyby vzhledem k pravidlům, podle nichž větu analyzuje člověk, protože jinak je „pravda“ dána právě obsahem trénovacích dat. Za předpokladu, že tytéž chyby se se stejnou frekvencí objeví i v testovacím textu, by jejich existence neměla být na závadu. (Příklady: špatně určená hranice věty; věta rozebraná do řetízku, kde každé slovo závisí na slovu, které mu ve větě předchází.)

[19] Viz např. [[14]], [[16]].

LITERATURA

[1] [BAHL ET AL. 1989]
Lalit Bahl, Peter Brown, Peter de Souza, Robert Mercer: *A Tree-Based Statistical Language Model for Natural Language Speech Recognition.* In: IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 37, No. 7, July 1989. Yorktown Heights 1989.

[2] [BÖHMOVÁ 1995]
Alena Böhmová: *Grafický editor závislostních struktur přirozeného jazyka (diplomová práce).* Matematicko-fyzikální fakulta Univerzity Karlovy, Praha 1995.

[3] [BROWN ET AL. 1993]
Peter Brown, Stephen della Pietra, Vincent della Pietra, Robert Mercer: *The Mathematics of Machine Translation: Parameter Estimation.* In: Computational Linguistics, 19 (2): pp. 263-312, 1993.

[4] [COVER, THOMAS 1991]
Thomas Cover, Joy Thomas: *Elements of Information Theory.* John Wiley & Sons, Inc., New York 1991.

[5] [DEMPSTER ET AL. 1976]
Arthur Dempster, N. M. Laird, Donald Rubin: *Maximum Likelihood from Incomplete Data via the EM Algorithm.* In: Journal of the Royal Statistical Society, Series B, Volume 39, pp. 1-38. London 1977.

[6] [GNEDENKO, HINČIN 1952]
Борис Владимирович Гнеденко, Александр Яковлевич Хинчин: *Элементарное введение в теорию вероятностей.* Государственное издательство технико-теоретической литературы, Москва 1952.
Boris Vladimirovič Gnedenko, Aleksandr Jakovlevič Hinčin: *Elementární úvod do teorie pravděpodobnosti.* SNTL, Praha 1954.

[7] [HLADKÁ 1994]
Barbora Hladká: *Počítačové vybavení pro zpracování velkých českých textových korpusů (diplomová práce).* Matematicko-fyzikální fakulta Univerzity Karlovy, Praha 1994.

[8] [HOLAN ET AL. 1995]
Tomáš Holan, Vladislav Kuboň, Martin Plátek: *An Implementation of Syntactic Analysis of Czech.* In: Fourth International Workshop on Parsing Technologies. Praha, 1995.

[9] [CHOMSKY 1957]
Noam Chomsky: *Syntactic Structures.* Mouton & Co. Publishers, Den Haag 1957.
Noam Chomsky: *Syntaktické struktury.* Academia, Praha 1966.

[10] [JAGLOM, JAGLOM 1960]
Авика Моисеевич Яглом, Исаак Моисеевич Яглом: *Вероятность и информация.* Государственное издательство физико-математической литературы, Москва 1960.
Avika Moiseevič Jaglom, Isaak Moiseevič Jaglom: *Pravděpodobnost a informace.* Nakladatelství Československé akademie věd, Praha 1964.

[11] [NEŠETŘIL 1979]
Jaroslav Nešetřil: *Teorie grafů.* SNTL - Nakladatelství technické literatury, n.p., Praha 1979.

[12] [OLIVA 1983]
Karel Oliva: *Syntaktická analýza češtiny na počítači.* In: Proceedings of the software seminary SOFSEM '83, VÚSEI AR, Bratislava 1983.

[13] [OLIVA 1989]
Karel Oliva: *A Parser for Czech Implemented in Systems Q.* Explicitní popis jazyka a automatické zpracování textu XVI. Matematicko-fyzikální fakulta Univerzity Karlovy, Praha 1989.

[14] [PETKEVIČ 1997]
Vladimír Petkevič: *Underlying Structure of Sentence Based on Dependency.* Filozofická fakulta Univerzity Karlovy, Praha 1997.

[15] [RIBAROV 1996]
Kiril Ribarov: *Automatická tvorba gramatiky přirozeného jazyka (diplomová práce)*. Matematicko-fyzikální fakulta Univerzity Karlovy, Praha 1996.

[16] [SGALL ET AL. 1969]
Petr Sgall, L. Nebeský, Alla Goralčíková, Eva Hajičová: *A Functional Approach to Syntax in Generative Description of Language.* New York 1969.

P ø í l o h a A

UŽIVATELSKÁ DOKUMENTACE K PROGRAMU

Součástí této diplomové práce je přes 9200 řádků zdrojového programového kódu, který lze nalézt na přiložené disketě; vzhledem k tomu, že zajímavost uvedeného materiálu neodpovídá jeho rozsahu, považujeme za zbytečné přikládat ho v tištěné podobě. Zde se omezíme na uživatelskou dokumentaci, tedy na popis použití programu a otázky spojené s kompatibilitou a instalací.

KOMPATIBILITA

Program byl testován na počítači s procesorem slučitelným s Intel 486 s 8 MB paměti. Vzhledem k tomu, že jde o 32-bitovou aplikaci, je vyloučeno jeho fungování na procesorech 286 a starších.

Použitý operační systém je MS-DOS verze 7.0, tj. Microsoft Windows 95 bez využití oken. Program volá pouze funkce ze standardní céčkové knihovny, dostupné i na Unixu, čímž se usnadní přenos na tento operační systém. Při přenosu na Unix je nutné zajistit (např. pomocí FTP), aby všechny soubory byly pojmenovány malými písmeny a aby dosovské konce řádků (CR LF) byly nahrazeny unixovými (LF). Soubor *makefile* je nutné přejmenovat na *makedos* a *makeunix* na *makefile*.

Zdrojový kód lze přeložit překladači Watcom C++ verze 10.5 a G++ (DJGPP) verze 2.0; použití jiných překladačů C++ nebylo vyzkoušeno.

Soubor *makefile* s informacemi o celém projektu předpokládá použití GNU Make verze 3.73.

Přestože Unix i Windows 95 umožňují použití dlouhých jmen souboru, délka jmen v tomto projektu zůstává omezena na 8+3 znaky, protože GNU Make pro DOS ještě dlouhá jména nezvládá.

Veškeré komentáře ve zdrojovém kódu i v *makefile* a hlášení programu jsou v češtině a používají kódovou stránku Windows 1250 (Střední Evropa). Všechna hlášení, kterými program oblažuje uživatele (s výjimkou ladicích výpisů, které se v konečné verzi programu nepřekládají), jsou soustředěna ve dvou programových modulech, takže není problém z nich odstranit diakritiku nebo je lokalizovat do jiného jazyka. Na kódové stránce dat, se kterými program pracuje, nezáleží; je pouze třeba, aby vstup pro analýzu používal stejné kódování jako trénovací data. Texty přiložené k programu (z nich pocházejí výsledky naší práce) jsou kódovány podle unixového standardu ISO 8859-2 (Latin 2).

INSTALACE

Zkopírujte obsah diskety včetně adresářové struktury na pevný disk, otevřete adresář `Sntx` a spusťte `make`. Předpokládá se, že tímto příkazem je v systému dostupný GNU Make verze 3.73. Dále musí být dostupný příkaz `gcc` pro vyvolání integrovaného překladače GNU C/C++. Chcete-li použít překladač Watcom, musíte nejdříve v souboru *makefile* změnit nastavení proměnné `prekladac = watcom`.

SPOUŠTĚNÍ

Způsob volání programu lze zjednodušeně popsat takto:

```
sntx [-akce=(trenink|analyza|synteza|test)] [-gram=gram]
[-vahy=λgram:...:λ0] [-rozklad=(hloubka|sirka|libovolne)]
[-budovani=(sekvencne|komponentove|do_jednicky)]
[-vstup=soubor] [-vystup=soubor] [-filtr=soubor]
[-hlaseni=soubor] [-ovladani=soubor]
[-rozsah=[vynechat:]použít] [-pouze_cist=(ano|ne)]
[-prepsat|-pripojit] [-pouze_tvary=(ano|ne)] [operand ...]
```

Ve skutečnosti je syntaxe příkazového řádku benevolentnější, než vyplývá z předlohy. Volby mohou začínat nejen znakem `'-'`, ale i `'/'`, který se někdy používá v Dosu. (Naopak operandy žádným z těchto znaků začínat nesmějí.) Název volby může od její hodnoty oddělovat nejen znak `'='`, ale i `':'`, což je potřeba, mají-li být volby předány prostřednictvím proměnné v prostředí programu. Není-li u jednotlivých voleb řečeno něco jiného, může být zadána libovolná kombinace voleb a operandů v libovolném pořadí. V názvech voleb se nerozlišují velká a malá písmena, což ale neplatí o hodnotách voleb ani o operandech.

Zvláštní význam mají operandy začínající znakem `'@'`. Text, který následuje za tímto znakem, se považuje za jméno proměnné, resp. souboru s dalšími volbami a operandy. Zejména se tím zbavujeme problému s omezenou délkou příkazového řádku pod Dosem. Program se nejprve pokusí najít ve svém prostředí proměnnou daného jména, a pokud taková neexistuje, bude jméno považováno za jméno souboru. Jednotlivé volby a operandy jsou potom čteny z proměnné, resp. souboru, jako by se vyskytly přímo na příkazovém řádku. Po přečtení celého obsahu proměnné či souboru se program vrátí na příkazový řádek.

Implicitní nastavení je rozepsáno u jednotlivých voleb. Vlastní implicitní nastavení můžete určit obsahem proměnné `SNTX` v systémovém prostředí

programu. Program naběhne se svým výchozím nastavením, potom přečte obsah proměnné SNTX a následovně čte zleva příkazový řádek. Jsou-li některá pozdější nastavení v konfliktu s předchozími, dostane přednost poslední přečtená hodnota.

-akce=*akce*

Základní volba, která určuje funkci programu. Výchozí nastavení této volby je `trenink`.

Je-li *akce*=**trenink**, program si na základě již rozebraného textu vytvoří („natrénuje“) jazykový model. Na vstupu se očekává soubor typu **.vzv* se syntaktickými zápisy vět - viz popis formátu souborů. Na výstup pošle natrénované rozdělení pravděpodobností ve formátu **.rpr*. Místo volby `-akce=trenink` lze použít zkratku `-al`.

Je-li *akce*=**analyza**, program převádí nerozebrané věty na jejich syntaktické zápisy. Na vstupu se očekává soubor typu **.vet* se syntakticky nerozebranými větami - viz popis formátu souborů. Na vstupu pro filtr (viz dále) se očekává soubor typu **.rpr*, vzniklý při trénovací fázi. Na výstup se pošle soubor se syntaktickými zápisy vět ve formátu **.vzv*. Místo volby `-akce=analyza` lze použít zkratku `-aa`.

Je-li *akce*=**synteza**, program převádí syntaktické zápisy vět zpět na nerozebrané věty, tj. odstraní syntaktickou strukturu. Můžeme tak získat např. testovací data: výstup syntézy předáme jako vstup do analýzy a její výstup porovnáme s původními vstupními daty. Na vstupu syntézy se očekává soubor typu **.vzv* (viz popis formátu souborů), na výstup se pošle soubor ve formátu **.vet*. Místo volby `-akce=synteza` lze použít zkratku `-as`.

Je-li *akce*=**test**, program otestuje natrénovaný jazykový model. Na vstupu očekává soubor typu **.vzv* (viz popis formátu souborů), ze kterého odstraní syntaktickou strukturu (obdobně jako při syntéze), výsledek protáhne analýzou a její výstup porovná s původními vstupními daty. Na vstupu pro filtr přitom vyžaduje soubor typu **.rpr*, vzniklý při trénovací fázi. Výsledek analýzy se uloží pouze do dočasného souboru a na výstup se místo něj pošle zpráva o výsledku srovnání (ve formátu obyčejného textového souboru — **.txt*). Místo volby `-akce=test` lze použít zkratku `-at`.

`-vstup`=*soubor*

Udává jméno souboru, ze kterého se má číst vstup. Jaká data jsou očekávána na vstupu, je popsáno výše u jednotlivých akcí. Je-li místo jména souboru uveden znak `'-'`, vstup programu se napojí na standardní vstup (`stdin`). To je také výchozí nastavení. Místo volby `-vstup` lze použít zkratku `-i`.

`-vystup`=*soubor*

Udává jméno souboru, do kterého se má psát výstup. Jaká data se objeví na výstupu, je popsáno výše u jednotlivých akcí. Je-li místo jména souboru uveden znak `'-'`, výstup programu se přesměruje na standardní výstup (`stdout`). To je také výchozí nastavení. Místo volby `-vystup` lze použít zkratku `-o`.

`-filtr`=*soubor*

Udává jméno souboru, ze kterého se má číst „filtr“, tedy soubor dat, která blíže určují programem prováděnou akci. Jaká data jsou očekávána ve filtru, je popsáno výše u jednotlivých akcí. Je-li místo jména souboru uveden znak `'-'`, filtr se přečte ze standardního vstupu (`stdin`). To je také výchozí nastavení. Místo volby `-filtr` lze použít zkratku `-f`.

-hlaseni=*soubor*

Udává jméno souboru, do kterého se mají psát chybová hlášení a všechny další zprávy, které nepatří do hlavního výstupu programu, jak je definován u jednotlivých akcí. Tato volba má význam zejména pod Dosem, kde není možné přesměrovat standardní chybový výstup. Je-li místo jména souboru uveden znak `'-'`, hlášení programu se přesměrují na standardní chybový výstup (`stderr`). To je také výchozí nastavení. Místo volby `-hlaseni` lze použít zkratku `-e`.

-ovladani=*soubor*

Udává jméno souboru, ze kterého má být program ovládán. Zejména jsou zde očekávány reakce na chybová hlášení. Je-li místo jména souboru uveden znak `'-'`, ovládací vstup programu se napojí na standardní vstup (`stdin`). To je také výchozí nastavení. Místo volby `-ovladani` lze použít zkratku `-c`.

-gram=*n*

Nastavuje velikost kontextu, na který se bere zřetel při trénování modelu. Do kontextu právě přidávané hrany se zahrne *n*`-1` předcházejících hran. Při *n*`=0` se bez ohledu na trénovací data vytvoří model s rovnoměrným rozdělením pravděpodobností. Tato volba nemá žádný vliv při analýze a testování, kdy se velikost kontextu zjišťuje z již natrénovaného rozdělení, předaného jako filtr. Současná implementace některých částí programu nepočítá s gramem vyšším než 2! Implicitní hodnota je 2 (bigram). Pro unigram a bigram lze také použít zkratky `-g1`, resp. `-g2`.

-vahy=*hodnota*`[:`*hodnota*`:…]`

Definuje váhy pro vyhlazování. Jednotlivé hodnoty jsou odděleny dvojtečkami; jako první se uvádí váha nejvíce-gramového rozdělení, jako poslední se uvádí váha rovnoměrného rozdělení. Uživatel

nese odpovědnost za to, že počet vah bude odpovídat nastavení volby `-gram`, že jednotlivé váhy budou ležet v intervalu $\langle 0;1 \rangle$ a že jejich součet bude 1. Celá a desetinná místa se oddělují tečkou, nikoli čárkou. Výchozí nastavení vah je `0.99:0.009:0.001`.

`-rozklad`=*`metoda`*

Určuje, jakým způsobem se strom rozkládá na posloupnost hran, a tím definuje uspořádání hran. Zvolená metoda ovlivňuje procházení stromu při trénování (a tím i volbu hran, které tvoří kontext hrany právě přidávané) a při analýze / testování (volba kontextu a vyloučení hran, které jsou ve smyslu daného uspořádání „menší“ než naposledy přidaná hrana). Výchozí nastavení této volby je `hloubka`.

Je-li *`metoda`*=**`hloubka`**, hrany jsou uspořádány podle pořadí svého závislého vrcholu při procházení stromu do hloubky. (Procházení do hloubky: začne se v kořeni, potom se v každém kroku přednostně vezme první syn naposledy navštíveného uzlu. Nemá-li syny, vezme se bratr po jeho pravé straně. Pokud ani ten neexistuje, přejde se k otci aktuálního vrcholu, hledá se jeho pravý bratr atd.) Místo volby `-rozklad=hloubka` lze použít zkratku `-rh`.

Je-li *`metoda`*=**`sirka`**, hrany jsou uspořádány podle pořadí svého závislého vrcholu při procházení stromu do šířky. (Procházení do šířky: začne se v kořeni, potom se v každém kroku přednostně vezme pravý bratr nebo bratranec naposledy navštíveného uzlu. Pokud neexistuje, hledá se nejlevější uzel na další úrovni od kořene.) Místo volby `-rozklad=sirka` lze použít zkratku `-rs`.

Je-li *`metoda`*=**`libovolne`**, uspořádání hran není definováno a nelze říci, která hrana tvoří kontext ve vícegramovém modelu. Tato

metoda má však smysl při současném nastavení `-gram=1`, protože pak při analýze není výběr hran omezen jejich uspořádáním. Místo volby `-rozklad=libovolne` lze použít zkratku `-rl`.

`-budovani`=*`metoda`*

Určuje, jakým způsobem se buduje strom při analýze. Výchozí nastavení této volby je `sekvencne`.

Je-li *`metoda`*=**`sekvencne`**, použije se sekvenční hladový algoritmus (viz též kapitolu Kapitola 4 , část „Analýza“). Místo volby `-budovani=sekvencne` lze použít zkratku `-bs`.

Je-li *`metoda`*=**`komponentove`**, použije se komponentový hladový algoritmus (viz též kapitolu Kapitola 4 , část „Analýza“). Místo volby `-budovani=komponentove` lze použít zkratku `-bk`.

Je-li *`metoda`*=**`do_jednicky`**, úplně se vynechá pravděpodobnostní model a vybuduje se tzv. jedničkový strom (viz též kapitolu Kapitola 5 , část „Výsledky“). Místo volby `-budovani=do_jednicky` lze použít zkratku `-bj`.

`-rozsah`=[*`vynechat`*`:`]*`použít`*

Umožňuje použít pouze část dat ze vstupního souboru. Číslo *`vynechat`* udává počet vět od začátku souboru, které se mají přeskočit bez provedení zvolené akce. Číslo *`použít`* udává počet vět, s nimiž se má akce provést, a jež bezprostředně následují za vynechanými.

`-pouze_cist`=`ano|ne`

Je-li nastaven tento přepínač, program pouze přečte vstupní data a skončí. Lze tak zkontrolovat, zda vstupní soubor neobsahuje syntaktické chyby (viz popis formátu souborů v příloze Pøíloha B). Im-

plicitně je tato volba vypnuta. Pokud je uvedena bez =ano nebo =ne, předpokládá se „ano“.

-prepsat

-pripojit

Je-li při trénování zapnuta volba -prepsat, výstupní model se vytvoří nově i v případě, že výstupní soubor už existuje. Dosavadní obsah výstupního souboru se přemaže. Je-li zapnuta volba -pripojit, program nejdříve přečte již existující rozdělení ze souboru, daného volbou -vystup, potom k němu přidává další hrany z trénovacích dat a nakonec vše opět uloží do původního souboru. Obě volby mají vliv pouze na trénink, ne na jiné akce. Výchozí nastavení je -prepsat.

-pouze_tvary=ano|ne

Je-li nastaven tento přepínač, budou ve výstupním souboru typu **.vet* nebo **.vzv* uvedeny pouze slovní tvary místo kompletního popisu vrcholu v hranatých závorkách. Implicitně je tato volba vypnuta. Pokud je uvedena bez =ano nebo =ne, předpokládá se „ano“.

PŘÍKLADY:

```
sntx -al -g1 -i=data\BL101JS.vzv -o=uni01.rpr

sntx -at -f=uni01.rpr -vahy=0.99:0.01 -rozklad=libovolne
-i=data\BLD04EB.vzv -rozsah=5

sntx -aa -f=bis01.rpr -rs -i=BLD04EB.vzv -o=BLD04EB.vet
```

P ø í l o h a B

FORMÁT SOUBORŮ

Pro popis syntaxe dat v jednotlivých souborech použijeme rozšířenou bezkontextovou gramatiku. Zapíšeme ji jako množinu pravidel, kterými se přepisuje vždy jeden neterminální symbol (levá část pravidla) na posloupnost neterminálních symbolů a/nebo skupin terminálních symbolů. Postupnou aplikací pravidel na počáteční neterminální symbol `<soubor>` pak dostaneme posloupnost terminálních symbolů (znaků), která odpovídá jednomu z možných tvarů souboru.

Každé přepisovací pravidlo se skládá z levé a pravé části. Levou část tvoří neterminál, který je pravidlem definován. Pravou část tvoří definice tohoto neterminálu. Může obsahovat jiné neterminály a/nebo skupiny terminálů, přičemž terminály vždy odpovídají znakům, očekávaným na vstupu. Místo neterminálů je na vstupu očekávána posloupnost terminálů, která z nich vznikne postupnou aplikací přepisovacích pravidel. Až sem je naše gramatika obyčejnou bezkontextovou gramatikou. Z praktických důvodů ovšem povolíme některá rozšíření, abychom si usnadnili zadávání gramatiky.

Především je častým jevem, že vstupní data mohou na různých místech obsahovat mezery, popřípadě též tabulátory, konce řádků apod., aniž by těmto znakům příslušel nějaký syntaktický význam. Kdybychom měli do gramatiky zahrnout mezery na všech místech, na kterých se mohou vyskytnout, gramatiku by to značně zesložitilo a znepřehlednilo. Proto definujeme dva druhy pravidel. První bude předpokládat, že mezi neterminály a skupinami terminálů na jeho pravé straně se může vyskytnout libovolný počet „me-

zer“; znaky, které se považují za mezeru, definujeme zvláštním neterminálem `<mezera>`. V pravidlech druhého typu mezery povoleny nebudou, pokud na jejich pravé straně nebude explicitně uveden tento zvláštní neterminál. Symboly na pravé straně pravidel typu 2 tedy musí na vstupu následovat bezprostředně za sebou. Všimněme si, že ani jeden druh pravidel neumožňuje implicitně vkládat mezery dovnitř do skupin terminálních symbolů.

Další usnadnění práce nám přinese povolení operátorů na pravé straně pravidel. Například operátor logické disjunkce nám umožní spojit dvě pravidla se stejnými levými stranami v jedno (viz níže). Operandem může být vždy buď neterminál, nebo skupina terminálů, nebo podvýraz uzavřený v závorkách. Nebude-li mezi dvěma operandy uveden žádný operátor, budeme implicitně dosazovat operátor nekomutativní logické konjunkce. To znamená, že na vstupu se očekávají oba operandy v pořadí, v jakém jsou uvedeny v pravidle. Operátory jsou uvedeny sestupně podle své priority.

`::=` Odděluje levou a pravou část pravidla typu 1 (s mezerami).

→ Odděluje levou a pravou část pravidla typu 2 (bez mezer).

`<` Ohraničuje zleva neterminální symbol.

`>` Ohraničuje zprava neterminální symbol.

`"` Ohraničuje zleva nebo zprava skupinu terminálních symbolů.

`[` Ohraničuje zleva množinu znaků. Množina se může vyskytnout uvnitř skupiny terminálů a na jejím místě se na vstupu očekává právě jeden z jejích prvků. Prázdná množina označuje libovolný znak.

`]` Ohraničuje zprava množinu znaků.

`-` Uvnitř množiny nahrazuje všechny znaky, které mají kód vyšší než znak nalevo od něj a nižší než znak napravo. Není-li znak nalevo uveden, tvoří dolní hranici intervalu znak s nejnižším možným kó-

dem. Není-li uveden znak napravo, tvoří horní hranici intervalu znak s nejvyšším možným kódem. Množina `[-]` je tedy rovna množině `[]`.

`!` Volitelně rozděluje množinu znaků na část s povolenými znaky a část se zakázanými znaky. Místo množiny se na vstupu očekává libovolný znak, který je uveden před vykřičníkem a není uveden za ním. Vykřičník se také může vyskytnout mimo množinu znaků jako operátor (viz níže).

`\` Uvozuje zápis zvláštního terminálního symbolu. Zvláštní terminální symboly jsou popsány níže. Vyskytne-li se zpětné lomítko v takové kombinaci s jiným znakem, jaká není níže popsána, považuje se tato kombinace za zápis obyčejného terminálního symbolu — znaku za lomítkem. Standardně se za terminální symbol považuje jeden znak a ten je také očekáván na vstupu.

`\\` Zpětné lomítko jako terminální symbol, bez svého zvláštního významu.

`\"` Uvozovky jako terminální symbol, bez svého zvláštního významu.

`\[` Levá hranatá závorka jako terminální symbol, bez svého zvláštního významu.

`\]` Pravá hranatá závorka jako prvek množiny, bez svého zvláštního významu.

`\-` Pomlčka jako prvek množiny, bez svého zvláštního významu.

`\!` Vykřičník bez svého zvláštního významu.

`\t` Tabulátor (TAB). Obvykle znak s kódem 9.

`\r` Návrat na začátek řádku (CR). Obvykle znak s kódem 13.

`\n` Konec řádku (LF). Obvykle znak s kódem 10.

(Ohraničuje zleva podvýraz, který má vystupovat jako samostatný operand nějakého operátoru. Závorky se používají pro změnu priority operátorů.

) Ohraničuje podvýraz zprava.

! Unární prefixový operátor. Smí se vyskytnout pouze před operandem, který reprezentuje jediný vstupní znak. Na vstupu je potom očekáván libovolný znak, na nějž se operand nemůže expandovat.

* Unární postfixový operátor. Jeho operand se na vstupu může opakovat n-krát, kde $n \geq 0$.

+ Unární postfixový operátor. Jeho operand se na vstupu může opakovat n-krát, kde $n \geq 1$.

? Unární postfixový operátor. Jeho operand se na vstupu může vyskytnout jednou nebo vůbec.

Mezera nebo prázdný znak mezi symboly jsou považovány za nekomutativní logickou konjunkci. Na vstupu se očekávají oba operandy v uvedeném pořadí.

& Operátor komutativní logické konjunkce. Na vstupu se očekávají oba operandy, ale v libovolném pořadí.

| Operátor logické disjunkce. Na vstupu se očekává právě jeden z jeho operandů.

MORFOLOGICKY ROZEBRANÝ TEXT (VĚTY, *.*VET*)

Textový soubor. Jde vlastně o pomocný formát, který jsme definovali, abychom mohli trénovat model morfologickými značkami místo slovníkových hesel. Odvodili jsme tento formát z níže uvedeného a předem daného for-

mátu souborů **.vzv* tak, že jsme z nich odstranili stromovou strukturu a jednotlivé vrcholy jsme uspořádali lineárně podle jejich pořadí ve větě.

```
<soubor *.vet> ::= <věta>*

<věta> ::= "(" <vrchol>+ ")"

<vrchol> ::= "\[" <heslo> "," <morfologická značka> ","
     <tvar> ("," <jiný atribut>)* "," <pořadí> "\]"

<heslo> ::= <varianta hesla> ("|" <varianta hesla>)*

<varianta hesla> ::= <atom>

<morfologická značka> ::= <varianta morfologické značky>
     ("|" <varianta morfologické značky>)* ("|" "...")?

<varianta morfologické značky> → <znak v morfologické
     značce>*

<znak v morfologické značce> → "[A-Z1-9\-]"

<tvar> ::= <atom>

<jiný atribut> → <znak v první části jiného atributu>*
     <znak ve druhé části jiného atributu>* // má-li první část pre-
     fix "ord", musí se skončit, protože pak nejde o <jiný atribut>, ale o
     <pořadí>

<znak v první části jiného atributu> → "[!=,\]]" | ("\\"
     "[=,\]]"?)

<znak ve druhé části jiného atributu> → "[!,\]]" | ("\\"
     "[,\]]"?)

<pořadí> ::= "ord" "=" <číslo>

<atom> → (<znak v atomu> | <zneškodněný oddělovač>)*

<znak v atomu> → !("\\" | <oddělovač> | <mezera>)

<zneškodněný oddělovač> → "\\" (<oddělovač>)?

<číslo> → "[0123456789]"+

<oddělovač> → "[=\|,\]]"

<zalomení řádku> → "\\\r\n" | "\\\n"
```

```
<konec řádku> → "\r\n" | "\n"

<mezera> → "[ \t\r\n]" | <zalomení řádku>
```

ULOŽENÝ JAZYKOVÝ MODEL (ROZDĚLENÍ PRAVDĚPODOBNOSTÍ, *.*RPR*)

Binární soubor. Vzniká při trénovací fázi programu a používá se při analýze. Formát tohoto souboru je vnitřní záležitostí programu, proto ho zde nespecifikujeme.

SYNTAKTICKY ROZEBRANÝ TEXT (VÝZNAMOVÉ ZÁPISY VĚT, *.*VZV* RESP. *.*FS*)

Textový soubor. Jeho formát vychází z předem daného tvaru souborů *.*fs*, které jsme měli k dispozici pro trénování (viz informace o použitých datech, kapitola Kapitola 5 , část „Výsledky“). Zde uvedená gramatika však není formální specifikací souborů *.*fs*, neboť tu jsme k dispozici neměli. Gramatika spíše říká, jak na dotyčná data pohlíží náš program. Některé informace v datech obsažené, se kterými program nepracuje, zde nejsou detailně rozebrány. Vnitřní struktura morfologických značek je popsána v kapitole Kapitola 5 a zde ji už nerozvádíme. Stromy — syntaktické zápisy vět jsou zapsány pomocí hierarchie závorek.

```
<soubor *.vzv> ::= <záhlaví> <strom>* <zápatí>

<záhlaví> ::= <řádek záhlaví>*

<řádek záhlaví> ::= "@" ("[]" | <zalomení řádku>)* <konec
     řádku>

<strom> ::= <vrchol> "(" <seznam> ")"
```

```
<seznam> ::= <strom> ("," <strom>)*

<vrchol> ::= "\[" <heslo> "," <morfologická značka> ","
     <tvar> ("," <jiný atribut>)* "," <pořadí> "\]"

<heslo> ::= <varianta hesla> ("|" <varianta hesla>)*

<varianta hesla> ::= <atom>

<morfologická značka> ::= <varianta morfologické značky>
     ("|" <varianta morfologické značky>)* ("|" "...")?

<varianta morfologické značky> → <znak v morfologické
     značce>*

<znak v morfologické značce> → "[A-Z1-9\-]"

<tvar> ::= <atom>

<jiný atribut> → <znak v první části jiného atributu>*
     <znak ve druhé části jiného atributu>* // má-li první část pre-
     fix "ord", musí se skončit, protože pak nejde o <jiný atribut>, ale o
     <pořadí>

<znak v první části jiného atributu> → "[!=,\]]" | ("\\"
     "[=,\]]"?)

<znak ve druhé části jiného atributu> → "[!,\]]" | ("\\"
     "[,\]]"?)

<pořadí> ::= "ord" "=" <číslo>

<atom> → (<znak v atomu> | <zneškodněný oddělovač>)*

<znak v atomu> → !("\\" | <oddělovač> | <mezera>)

<zneškodněný oddělovač> → "\\" (<oddělovač>)?

<číslo> → "[0123456789]"+

<oddělovač> → "[=\|,\]]"

<zalomení řádku> → "\\\r\n" | "\\\n"

<konec řádku> → "\r\n" | "\n"

<mezera> → "[ \t\r\n]" | <zalomení řádku>
```