Závislostní zachycení větné struktury v anotovaném syntaktickém korpusu (nástroje pro zajištění konzistence dat)

Jan Štěpánek

Disertační práce

Univerzita Karlova v Praze
Matematicko-fyzikální fakulta
Ústav formální a aplikované lingvistiky

5. října 2006

Školitel: Prof. PhDr. Jarmila Panevová, DrSc.
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta
Ústav formální a aplikované lingvistiky

Oponenti: Doc. RNDr. Jan Hajič, Dr.
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta
Ústav formální a aplikované lingvistiky

Prof. Patrice Pognan
Institut national des langues et civilisations orientales (INALCO)
Paříž

Obsah

Poděkování

Poděkování patří v první řadě všem, kteří se podíleli na vzniku **PDT** 2.0. Mezi programátory bych chtěl zvláště poděkovat Ondřeji Bojarovi, Jiřímu Havelkovi, Petru Pajasovi, Jiřímu Semeckému, Danu Zemanovi, Zdeňkovi Žabokrtskému a Janu Hajičovi, který se ovšem na projektu podílel jako organizátor a vedoucí. Za anotování a pomoc při opravách korpusu patří dík Alevtině Bémové, Václavě Benešové, Jakubu Dotlačilovi, Veronice Kolářové, Lucii Kučové, Markétě Lopatkové, Marii Mikulové, Magdě Razímové, Barboře Smrčkové, Kateřině Součkové, Zdeňce Urešové, Janě Vejvodové, Kateřině Veselé a Šárce Zikánové.

Další velký dík patří paní profesorce Jarmile Panevové, jejíž pomoc jsem ocenil zvláště při sepisování práce.

Tato práce by nemohla vzniknout nebýt existence Centra komputační lingvistiky (MŠMT: LN00A063) a projektu Integrace jazykových zdrojů za účelem extrakce informací z přirozených textů (GA AV ČR: 1ET101120503).

Úvod

Přesvědčení, podle kterého se značkováním korpus kazí, je už technicky i metodologicky překonané.
Eva Hajičová, Výzva (festšrift k padesátým narozeninám Nikiho Petkeviče)

Slovem *korpus* se v lingvistice označuje rozsáhlá sada textů v přirozeném jazyce, uložená v dnešní době většinou v elektronické podobě a zpracovatelná na počítači. Korpusy bývají různým způsobem strukturovány a anotovány, což znamená, že k textům samotným je přidána nějaká další informace, většinou lingvistické povahy, kterou do korpusu vkládají lidé (anotátoři), popřípadě ji přidává nějaká automatická procedura. Anotovaný korpus je pak zdrojem informací pro lingvistický výzkum, protože je v něm možné hledat výskyty nejrůznějších jevů či ověřovat formulované hypotézy; často se také korpusy používají k trénování statistických jazykových modelů.

Obsahem této práce je podrobný popis nástrojů, s jejichž pomocí byla sledována konzistence a integrita dat v korpusu a které pomáhaly odhalovat různé typy chyb, klasifikovaly je a případně i odstraňovaly. Anotování je totiž náročnou činností, neboť vyžaduje od anotátorů detailní znalost jevů, které jsou v korpusu značkovány. Jako při každé lidské práci i při ní dělají lidé občas chyby.

Kromě anotování je dalším zdrojem chyb zpracování anotace a zpracování korpusu vůbec: převody mezi datovými formáty, slévání informací z různých zdrojů či automatická anotace některého jevu. Chyby při anotování a zpracování korpusu mohou mít různě závažné důsledky – od drobných chyb v anotaci (nepřesná či neodpovídající interpretace jevu) až po závažnější chyby, které mohou mít destruktivní charakter (např. smazání části korpusu).

Pro anotaci bývají stanovena pravidla, která popisují, jakým způsobem zachytit různé lingvistické jevy. Při tvorbě těchto pravidel je nutné učinit řadu nejrůznějších lingvistických zobecnění, aproximací a ústupků, aby bylo možné zachytit co nejširší oblast jazyka a zároveň aby počet pravidel nepřesáhl únosné meze. Bývají stanovena i obecná pravidla pro technické řešení situací, jejichž řešení není pravidly přesně specifikováno. Vzhledem k složitosti jazyka však není často ani přesto zcela jasné, jak konkrétní jev v daném případě anotovat, rozhodnutí závisí na citu anotátora a jeho

pochopení textu. Tentýž text tak může být různými lidmi (nebo dokonce tímtéž člověkem s delším časovým odstupem) anotován různým způsobem, což vnáší do korpusu nejednotnost.

V případě Pražského závislostního korpusu, kterým se v této práci budeme zabývat, existovaly ještě další dvě příčiny chyb a inkonzistencí v datech. Do jisté míry se asi mohou vyskytovat i při anotování jiných korpusů, pro Pražský závislostní korpus však byly typické:

1. Pražský závislostní korpus popisuje texty na několika rovinách. Kromě nejnižší roviny obsahující samotné texty navazuje každá další rovina $\mathbf{r}_n$ na nižší rovinu $\mathbf{r}_{n-1}$, jejíž všechny chyby jsou zdrojem dalších chyb a inkonzistencí na rovině $\mathbf{r}_n$. Ty se navíc často nedají opravit jinak než opravou chyby na rovině $\mathbf{r}_{n-1}$. „Nultá“ rovina obsahující texty je však také plná chyb, zeugmat, anakolutů, překlepů a nejrůznějších nedostatků (např. kolečko používané jako odrážka při odsazování je zapsáno jako písmeno *o*). Mnohdy se zdá, že zdrojové texty neměly korektory či neprocházely redakční ani žádnou jinou úpravou.
2. Při anotaci složitějších rovin se instrukce pro anotátory měnily v průběhu anotování, většinou jako reakce na jejich dotazy a připomínky. Dříve anotovaná data, podle starých pravidel korektní, tak mohla být náhle plná chyb, aniž by se jakkoli změnila. Některá pravidla byla dokonce přesně specifikována až v době, kdy již byla anotační práce dokončena.

Je zřejmé, že je třeba průběžně sledovat, zda nenastaly některé závažné chyby (s důsledky jako smazání částí korpusu či změna obsahu souborů takovým způsobem, že je používané nástroje nadále nedokážou načíst). Nástroje k tomu účelu vyvinuté se však ukázaly vhodnými i ke kontrolám drobnějších chyb v anotaci – navíc bylo jejich použití jediným možným způsobem, jak tyto chyby alespoň nějakým způsobem sledovat, protože k ručnímu procházení dat nebyl dostatek lidských ani finančních zdrojů, natož času.

Pokud bylo výskytů určitého typu příliš mnoho pro ruční průchod a byl nalezen vhodný algoritmus opravy, byly nalezené chyby opravovány automaticky. Chyby s menším počtem výskytů nebo chyby zbylé po automatických opravách, které se neuplatnily ve sto procentech případů, byly ručně opravovány anotátory.

Mohlo by se zdát, že všech těchto méně závažných chyb nebude v porovnání s celkovou velikostí korpusu mnoho, a je tedy zbytečné se jimi dále zabývat. Pokud se však zaměříme na určitý lingvistický jev, klesne počet relevantních výskytů jen do řádu desítek či jednotek.[1)] Vzhledem k tomu,

[1)] Například v celém tektogramaticky anotovaném korpusu je jen 57 užití doplňku, který není ani jednou ze svých závislostí závislý na slovese, protože to je nominalizováno (tj. v podstatě nahrazeno substantivem nebo adjektivem s podobným významem).

kolik různých jevů korpus popisuje, se pro řidší z nich stává v podstatě každý jednotlivý výskyt důležitým.

Aby bylo možné hodnotit přínos poanotačních oprav, byly všechny změny v datech sledovány a kategorizovány. V každé kategorii byl pak sledován počet provedených oprav, popřípadě počet chyb, které v datech zůstaly, pokud bylo množství chyb příliš vysoké a oprava se nedala provést automaticky. Tato čísla nejlépe ilustrují, kolik práce bylo při opravách korpusu vykonáno a ve kterých oblastech anotátoři nejvíce chybovali.

Sledování oprav dat mělo pak zpětný účinek i na samotnou teorii, podle níž byl korpus vybudován. Oblasti, které se projevily jako nejproblematičtější, se ukázaly jako nedostatečně prozkoumané a popsané, a třebaže se v nich problémy většinou očekávaly, objevilo se i několik témat nových, jejichž soustavným studiem se dosud nikdo nezabýval. Korpus však nejen ukazuje nové oblasti výzkumu, ale sám pro každou z nich přímo předkládá základní studijní materiál, z něhož je možné při vědeckém bádání vycházet.

Následná analýza poanotačních oprav umožnila učinit ještě další závěry, které se obecně týkají metodologie oprav velkých souborů dat. Některé postupy se totiž ukázaly vhodnější než jiné, protože vedly k lepším výsledkům, rozdílné byly technické či kapacitní nároky jednotlivých metod a nepostradatelnou se stala také jakási hierarchie kontrol a oprav, která je klasifikovala podle různých kritérií a určovala jejich pořadí podle důležitosti.

Kapitola 1

Pražský závislostní korpus

I spadl příval, a přišly řeky, a váli větrové, a obořili se na ten dům, ale nepadl; nebo založen byl na skále.
Matouš, 7:25

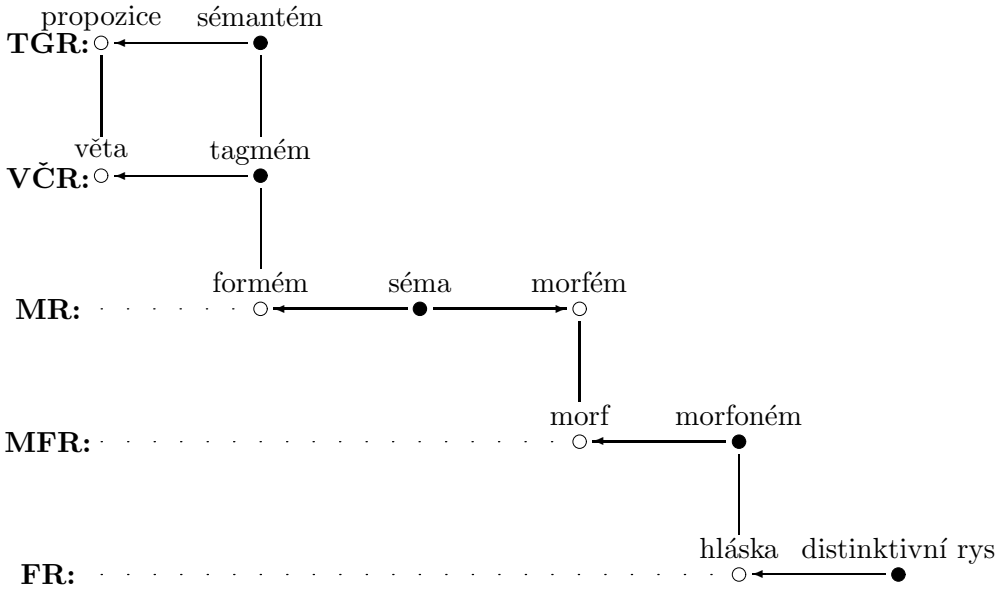
1.1 Teoretické základy

Podkladem pro Pražský závislostní korpus (Prague Dependency Treebank, **PDT**) byla teorie funkčního generativního popisu (poprvé souborně představená v [Sgall, 1967]), která bude v textu dále označována jen jako **FGP**. Lingvistickým popisem věty podle **FGP** je její reprezentace na několika rovinách (tzv. stratifikační přístup), podle původního návrhu bylo těchto rovin pět:

1. rovina fonetická
2. rovina morfonologická
3. rovina morfématická
4. rovina větněčlenská (povrchová syntax)
5. rovina tektogramatická (hloubková syntax)

Na každé rovině jsou definovány elementární jednotky, které se spojují pomocí kombinační relace C do jednotek komplexních. Mezi jednotkami rovin (jak elementárními, tak komplexními) pak existuje relace reprezentace R, která jednotlivé roviny propojuje (viz obrázek 1.1 na straně 6, relace C je znázorněna šipkou, relace R svislou čarou; elementární jednotky představují vyplněné kruhy).

Relaci reprezentace R lze také chápat jako vyjádření vztahu formy a funkce: směrem od tektogramatické roviny „dolů“ vyjadřuje formu (např. formou podmětu je zpravidla nominativ), opačným směrem představuje funkci (např. nominativ má funkci podmětu).



Obrázek 1.1: Schéma rozdělení lingvistického popisu na jednotlivé roviny podle **FGP**

Elementární jednotkou fonetické roviny je *distinktivní rys*. Příkladem distinktivních rysů jsou znělost, obouretnost, délka a další vlastnosti hlásek.

Jako *morfoném* si můžeme představit například dvojici hlásek *d*|*t*, která se vyskytuje v kmeni slova *hrad*, v němž se koncové *d* může v některých kontextech vyslovovat jako *t*. *Morf* je pak řetězec morfonémů, například právě *h-r-a-d*|*t*.

Morfém vyjadřuje funkci morfu, tedy gramatické kategorie, které příslušný morf reprezentuje, pokud jde např. o koncovku, nebo představuje kmen slova či pomocné slovo – příkladem morfému tak může být třeba dativ singuláru životného substantiva, jemuž odpovídají dvě různé formy, morfy *-u* nebo *-ovi*; jiným příkladem je morfém *matk-*|*matc-*|*matek*, odpovídající kmeni slova *matka*, nebo morfém *v*|*ve*, odpovídající předložce *v*. Morfémy se skládají z jednoho nebo více *sémat*, v případě kmenů slov a slov pomocných jde o jediné séma, v případě dativu singuláru životného substantiva jde o čtyři různá sémata, jimiž jsou hodnoty jednotlivých mluvnických kategorií. Sémata, na něž se rozpadají morfémy, se pak jiným způsobem skládají do *formémů*, které reprezentují jednotky větněčlenské roviny: formémem je například kombinace sématu *v* s lokálem, jehož funkcí může být třeba příslovečné určení místa.

Mezi *tagmémy* patří větné členy (podmět, přísudek aj.), *sémémy* (tj. lexikální tagmémy, které by dnes byly chápány jako jakési odkazy do

slovníku významových slov) a tzv. *sufixy* (např. přítomnost, minulost u sloves, číslo u substantiv).

Sémantémy jako elementární jednotky roviny jazykového významu (tektogramatické roviny) se podobají jednotkám roviny větněčlenské, lexikální sémantémy se nazývaly *sémoglyfy*. Větným členům odpovídají *funktory* (například aktor, efekt, adresát) a pro sufixy se nověji používá označení *gramatémy*. Funktory byly později podrobněji rozčleněny pomocí *subfunktorů*,[1)] které blíže určují drobnější významové odstíny (například pro funktor `LOC`, určení místa jako odpovědi na otázku *kde?*, můžeme rozlišit subfunktory `near` blízko čeho, `around` okolo čeho, `betw` mezi čím, `opp` proti čemu, `front` před čím a tak dále).

Popisem věty na rovině větněčlenské a tektogramatické je *závislostní strom*: jde o ohodnocený strom ve smyslu teorie grafů, který má navíc definováno lineární uspořádání na uzlech a jeden význačný uzel, který se nazývá *kořen*. Jednou z možných formalizací tohoto pojmu je pětice $T = (V, E, r, O, \mathcal{F})$, kde V je množina uzlů, E množina hran (tedy neuspořádaných dvojic uzlů, $E \subseteq \{\{u, v\}\,;\, u, v \in V \;\&\; u \neq v\}$) a $r \in V$ je kořen. Vlastnost *být stromem* se dá definovat mnoha způsoby, například tak, že každé dva uzly $u, v \in V$ lze spojit právě jednou cestou. Relace O je lineárním uspořádáním na V a funkce $\mathcal{F}$ je *ohodnocením* uzlů stromu.

Poněvadž každé dva uzly lze spojit právě jednou cestou, existuje pro každý uzel u jednoznačně určená cesta z kořene r do u. Délka této cesty se nazývá *hloubka* uzlu. *Rodičem* uzlu u se míní takový uzel, který je s ním spojen hranou a má hloubku o jedna menší než uzel u. Říkáme také, že každý uzel *závisí* na svém rodiči.

Závislostní stromy bývají popisovány i zobrazovány kořenem vzhůru. Každý uzel je zobrazen na úrovni, která odpovídá jeho hloubce (všechny uzly závislé na daném uzlu jsou zobrazovány ve stejné výšce pod svým rodičem). Pořadí uzlů zleva doprava odpovídá uspořádání O.

Definovat obor hodnot funkce $\mathcal{F}$ je poněkud složitější. Uzlu v závislostním stromě totiž neodpovídá ani elementární, ani komplexní jednotka příslušné roviny, ale skupina několika tagmémů, respektive sémantémů, která se nazývá *komplexní symbol*. Skládá se z jednoho tagmému (resp. sémantému) lexikálního, jednoho syntaktického (větný člen, resp. funktor) a různého počtu tagmémů (resp. sémantémů) morfologických (sufixů, resp. gramatémů). Každý komplexní symbol odpovídá jednomu významovému slovu (a to i takovému, které je případně ve vyjádření věty elidováno). Syntaktická funkce se komplexnímu symbolu přiřazuje uměle, teoreticky přísluší hraně (tj. dvojici komplexních symbolů), díky vlastnostem stromu však lze každou hranu jednoznačně reprezentovat uzlem s vyšší hloubkou.

[1)] Pro subfunktory se dříve používalo označení syntaktické gramatémy, termín subfunktor poprvé použili Razímová a Žabokrtský [2005].

Formální vztahy mezi rovinami nebyly popisovány *deklarativně*, tj. jako soubor podmínek, kterým musí prvky jednotlivých rovin vyhovovat, ale *procedurálně*, tedy jako seznamy operací, kterými lze dospět od jedné roviny k druhé. Tento přechod byl přitom popisován vždy směrem od tektogramatické roviny „dolů“, takže popisoval *generování* textu z tektogramatických reprezentací (podrobněji např. [Panevová, 1979]).

Za součást tektogramatické roviny byly později považovány také informace o *valenci*, tedy o tom, jaké aktanty a jaká obligatorní volná doplnění jednotlivá slova vyžadují ([Panevová, 1974], [Panevová, 1980]). Tato informace je reprezentována odkazem do valenčního slovníku, který navíc popisuje i formy, kterých mohou valenční členy nabývat – popisuje tedy vztah roviny tektogramatické k rovinám nižším.

Těžiště výzkumného zájmu leželo vždy v rovinách „vyšších“, a to ze dvou důvodů: jednak byly otázky týkající se fonetiky zpracovány dostatečně podrobně již ve starších lingvistických pracích (zejména pracích tzv. Pražské školy), jednak rovina morfonologická i fonetická slouží k popisu mluvené řeči, takže při zkoumání psaného textu nejsou v podstatě potřeba (musela by je nahradit jakási rovina, která by navazovala na rovinu morfématickou a popisovala by grafické vyjádření jednotlivých morfémů, tento vztah ale není příliš komplikovaný a dá se přesně popsat pomocí klasifikace slov do jednotlivých tvaroslovných paradigmat).

1.2 Rozdíly mezi FGP a PDT

Při budování **PDT** se sice vycházelo z teorie **FGP**, z metodologických i technických důvodů však byly mnohé předpoklady změněny a teorie byla na mnoha místech přizpůsobena okolnostem a požadavkům – ať již finančním, časovým, implementačním či jiným. Dalším zdrojem odchylek mezi teorií **FGP** a **PDT** je rovněž fakt, že anotace textu (postupuje se od rovin nižších k vyšším) je vlastně opačný proces než generování, které bylo teorií primárně popisováno. Mnoho informací, které přicházejí z textu, je možné zachovat i na rovinách, kde se s existencí podobného údaje nepočítalo, protože je jistě výhodnější mít popis bohatší, než ztrácet informace, které jsou díky opačnému pohledu na text dostupné.

Základním rozdílem je již samotné členění do rovin. **PDT** se skládá ze tří rovin popisu:

1. rovina morfologická
2. rovina analytická
3. rovina tektogramatická

Do systému svým způsobem patří i „nultá“ rovina slovních tvarů, obsahující původní text, rozdělený na jednotlivá slova (v technickém smyslu, tj. za slovo se považuje i interpunkční znaménko a podobně).

Morfologická rovina **PDT** odpovídá rovině morfématické z **FGP**. Jednotlivá sémata však nejsou slučována do morfémů, ale do ještě složitějších (tj. vícečlenných) jednotek, z nichž každá odpovídá právě jednomu slovu z anotovaného psaného textu (jde vlastně o morfémy, které se však sdruží dohromady, pokud odpovídají v textu jednomu slovu, takže se k sobě dostanou například kmen a koncovka). Gramatickým kategoriím odpovídají *tagy* neboli *morfologické značky*, pro homonymní tvary je vybrán vždy jen jeden z možných tagů. Zároveň je text rozdělen na věty. Příklad anotace jedné věty na morfologické rovině je na obrázku 1.2 (chybný tvar „*oživením*“ z původního textu je opraven na „*oživení*“).

Příklady označené [pdt] pocházejí přímo z **PDT**; příklady označené [zm] jsou věty, které sice také pocházejí z **PDT**, ale byly nějakým způsobem změněny (nejčastěji zkráceny); a konečně příklady označené [vym] jsou věty vymyšlené a uměle zkonstruované.

Forma	Lemma	Morfologická značka
Některé	`některý`	`PZFP1----------`
kontury	`kontura`	`NNFP1-----A----`
problému	`problém`	`NNIS2-----A----`
se	`se_^(zvr._zájmeno/částice)`	`P7-X4----------`
však	`však`	`J^------------`
po	`po-1`	`RR--6----------`
oživení	`oživení_^(*3it)`	`NNNS6-----A----`
Havlovým	`Havlův_;S_^(*3el)`	`AUIS7M---------`
projevem	`projev`	`NNIS7-----A----`
zdají	`zdát`	`VB-P---3P-AA---`
být	`být`	`Vf--------A----`
jasnější	`jasný`	`AAFP1----2A----`
.	.	`Z:------------`

Obrázek 1.2: Morfologická rovina. Příklad anotace věty „*Některé kontury problému se však po oživením Havlovým projevem zdají být jasnější.*“ [pdt]

Analytická a tektogramatická rovina jsou rovinami „strukturními“, reprezentací věty je na nich struktura (strom). Obě však závislostní strom do jisté míry podobným způsobem upravují, aby bylo možné zachytit koordinované a aponované konstrukce,[2)] jejichž zachycení nebylo teorií **FGP** přesně specifikováno (alespoň ne ve starších pracích). Řešení, které bylo použito, zachovává koordinační spojku nebo apoziční výraz jako uzel stromu, který je speciální hranou zavěšen na uzel, na kterém závisejí koordinované (resp. aponované) členy a který se nazývá *efektivní rodič* koordinovaných (resp. aponovaných) uzlů, a ty jsou opět speciálními hranami zavěšeny na koor-

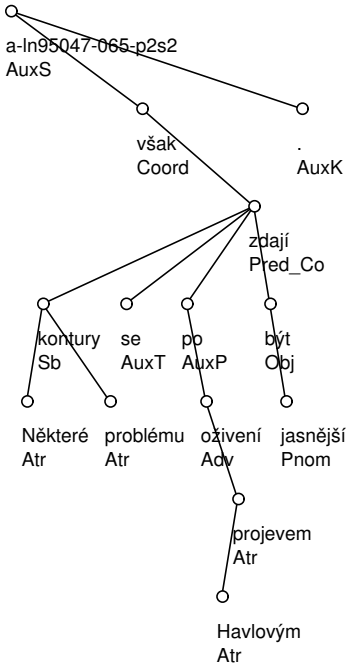
[2)] Na tektogramatické rovině mezi podobně zachycované konstrukce patří ještě matematické operace a intervaly, zachycované pomocí zvláštního funktoru `OPER`.

dinační spojku či apoziční výraz. Tyto speciální hrany neodpovídají relaci závislosti a naopak závislost mezi koordinovanými (resp. aponovanými) uzly a jejich efektivním rodičem není ve stromě reprezentována hranou. Kromě toho existuje ještě další druh hrany pro společná rozvití koordinovaných (resp. aponovaných) členů, podrobnosti viz dále v oddílu 2.2.

Méně významným rozdílem strukturních rovin oproti teorii je existence *technického kořene*, zvláštního uzlu, který reprezentuje celou větu a obsahuje o ní nejrůznější informace. Technický kořen je rodičem řídícího uzlu věty (tedy kořene podle teorie **FGP**) a je jediným typem uzlu, který může tyto informace obsahovat. Uzly ostatních typů (tj. uzly, které se mohou vyskytovat také v pozici synů) tyto informace naopak neobsahují nikdy. Kdyby neexistoval technický kořen, nemohly by se informace o větě zapisovat přímo do stromu, takže by se komplikovala struktura dat, nebo by se zapisovaly k řídícímu uzlu věty (tedy ke kořeni ve smyslu teorie **FGP**), čímž by se porušila jednoduchá hierarchie typů uzlů. Navíc by se značně zkomplikovala situace, kdy je třeba při anotování z některého uzlu vytvořit nový kořen a starý kořen učinit uzlem s nenulovou hloubkou: informace o větě by se musely přesunout z jednoho uzlu ke druhému a pro takovou operaci by musela existovat speciální metoda; naopak existence technického kořene činí z tohoto úkolu obyčejné převěšení uzlu. (K převěšování uzlu do pozice kořene věty ve smyslu teorie **FGP** docházelo například při anotování tektogramatického stromu pro větu z našeho příkladu, srov. spojku „*však*“ a sloveso „*zdát*“ na obrázcích 1.3 a 1.4).

Analytická rovina (viz obrázek 1.3) vzdáleně připomíná rovinu větněčlenskou, ta také byla při specifikaci analytické roviny zdrojem inspirace. I zde je popisem věty strom, který se však od závislostního stromu liší, i když odhlédneme od zachycení koordinací a apozic, protože ani tak neodpovídá každá jeho hrana závislosti. Důvodem je, že uzly stromu analytické roviny odpovídají jednoznačně jednotkám morfologické roviny, tj. jednotlivým slovům textu, a tak ani na této rovině nedochází k sloučení pomocných slov se slovy významovými, zatímco v teorii **FGP** by k tomu došlo již na rovině morfématické. Pomocná slova[3)] je proto potřeba ve stromě někam „zavěsit“, s výjimkou předložek a podřadicích spojek zpravidla pod významové slovo, s nímž by tvořila komplexní jednotku na větněčlenské rovině. Hrana od takového pomocného slova pak neodpovídá závislosti, ale je jen technickou reprezentací jiného vztahu mezi uzly. Typ hrany je reprezentován *analytickou funkcí*, která u závislostních hran označuje typ závislosti. Předložky a podřadicí spojky se na rozdíl od ostatních pomocných slov umisťovaly *nad* významové slovo, s nímž by na větněčlenské rovině tvořily formém. Hrany vedoucí z předložek a spojek tedy rovněž neodpovídají závislosti, ovšem pokud bychom spojili rodiče předložky s jejím

[3)] Stejně jako pomocná slova byla do stromu začleňována i interpunkce a další řetězce znaků, které se v textech vyskytovaly, ale které nespadají do kategorie pomocných slov.

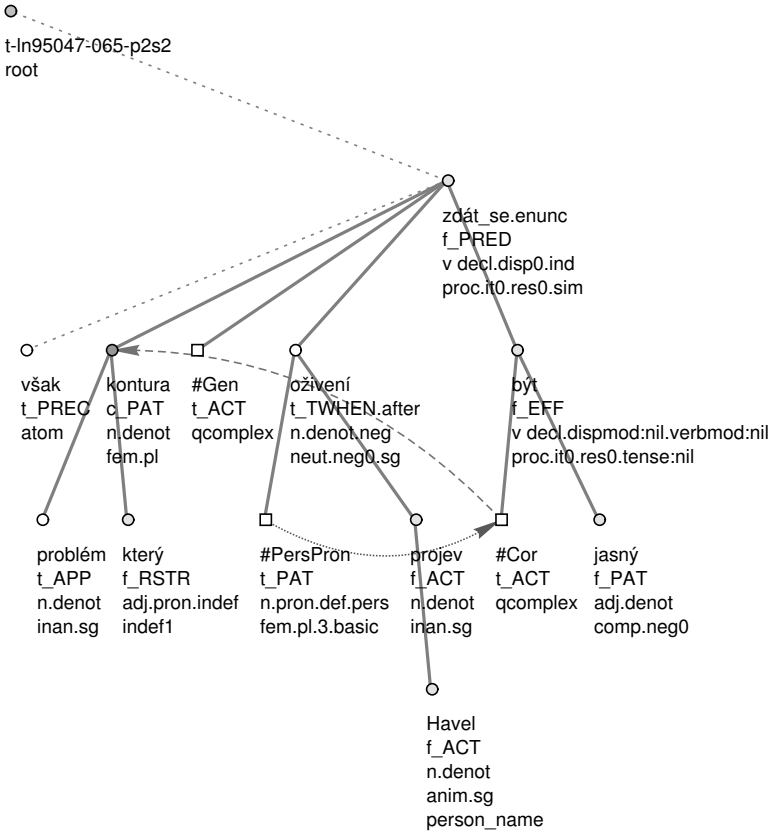


Obrázek 1.3: Analytická rovina: příklad stromu (pro stejnou větu jako na obrázku 1.2): „*Některé kontury problému se však po oživením Havlovým projevem zdají být jasnější.*“ [pdt]

synem, získali bychom dvojici slov v závislostním vztahu. Tato závislost není ve stromě na analytické rovině reprezentována hranou, ale vlastně dvojicí hran.

Dalším rozdílem mezi analytickou a větněčlenskou rovinou je popis elips. Jak již bylo uvedeno v předchozím odstavci, odpovídají uzly stromu na analytické rovině *jednoznačně* jednotkám morfologické roviny – nejenže žádná slova nebyla „mazána“, ale žádná nebyla ani přidávána. Elidované slovo se proto na analytické rovině neobjevuje, a pokud by na něm závisela nějaká další slova, zavěsí se hranou se speciální analytickou funkcí na jeho nejbližšího nevynechaného předka. Taková hrana pak opět neodpovídá závislosti.

Rovina tektogramatická (viz obrázek 1.4) sice nese stejný název jako jí odpovídající rovina **FGP**, ale i ona se od své teoretické předlohy liší. Podobně jako na analytické rovině existují i na rovině tektogramatické hrany, které neodpovídají relaci závislosti. Kromě koordinačních, apozičních a intervalových hran, o kterých jsme se již zmínili, existují ještě další „nezávislostní“ hrany: jimi byly do stromů začleňovány výrazy, jejichž syntaktické zařazení je nejasné – částice rozvíjející spojovací výrazy, slova



Obrázek 1.4: Tektogramatická rovina: příklad stromu (opět stejná věta jako na obrázcích 1.2 a 1.3): „*Některé kontury problému se však po oživením Havlovým projevem zdají být jasnější.*“ [pdt]

odkazující k předchozím výpovědím a podobně – a pro které neexistuje odpovídající gramatém, který by mohl být jejich funkcí.

Oproti teoretické předloze je u každého uzlu zaznamenáno, zda se nacházel již na nižších rovinách, nebo byl při anotování „vygenerován“ (nevyjádřené aktanty, elipsy), z pohledu generování tedy spíše to, zda bude mít uzel na nižší rovině nějakou odpovídající realizaci.

Kromě toho je součástí tektogramatické roviny anotace aktuálního členění (viz [Veselá a Havelka, 2003]), jehož začlenění do tektogramatické roviny se v původním návrhu **FGP** předpokládalo (viz například [Sgall a kol., 1986]). Projevuje se jednak klasifikací uzlů na kontextově zapojené, kontrastivně kontextově zapojené a kontextově nezapojené; jednak pořadím uzlů (relace O), které odpovídá míře výpovědního dynamismu.

Dále byla na tektogramatické rovině anotována *koreference* – vztažná, reflexivní a ukazovací zájmena a zájmena osobní ve třetí osobě mají vyznačen

svůj antecedent (viz [Kučová a kol., 2003]). Obdobným způsobem jsou označeny i vztahy kontroly – doplněný kontrolovaný člen odkazuje ke členu kontrolujícímu – a „druhá závislost“ doplňku – zatímco doplněk závisí na slovese, odkazuje zároveň na jméno, ke kterému se vztahuje.

Jak aktuální členění, tak koreference fakticky rozšiřují anotaci na tektogramatické rovině přes hranici věty (stejným způsobem lze nahlížet i na funktor `PREC` pro částice odkazující k předchozí výpovědi). Popis hloubkové stavby věty tedy není možný bez jejího zasazení do kontextu.

Prvky vyšších rovin obsahují také odkazy do rovin nižších, konkrétně na ty prvky nižších rovin, z nichž vznikly (odkazy jsou realizovány jednoznačnými identifikátory uzlů). Do jisté míry tak tyto odkazy odpovídají relaci reprezentace R teorie **FGP**, i když tato relace je z důvodu svého vzniku v průběhu anotování „orientována opačným směrem“: zatímco teorie **FGP** pojímala popis generativně, jako popis *syntézy*, je anotování naopak *analýzou*. Pokud by nás tedy zajímala inverzní relace, například množina uzlů tektogramatické roviny, která odpovídá nějakému danému uzlu roviny analytické, nezbývalo by nám než takovou množinu „spočítat“, tj. projít všechny uzly tektogramatické roviny z odpovídajícího dokumentu a zjistit, které z nich obsahují daný analytický uzel ve svém odkazu (ilustrací takové situace je například spojení „*červené a bílé víno*“ [vym], kde uzlu „*víno*“ analytické roviny odpovídají dva uzly roviny tektogramatické, které oba obsahují odkaz na tentýž analytický uzel).

Odhlédneme-li od všech těchto rozdílů, ani tak konečná realizace korpusu výše popsané teorii **FGP** plně neodpovídá, přičemž příčiny nenaplnění teoretických předpokladů jsou rozmanité. Hlavním důvodem je bezpochyby skutečnost, že v reálných datech narážíme často na věty,[4)] které se nacházejí kdesi v hraniční oblasti gramaticky správně tvořených vět (a někdy i daleko za ní). Budování korpusu tak při řešení problémů anotátorů zpětně přispělo k rozšíření teorie o popisy častých jevů, které dosud nebyly zpracovány – teorii **FGP** tedy nelze chápat jako uzavřenou a nepodléhající dalšímu vývoji.

Ani gramaticky správné a teorií pokryté věty však nebylo možné v praxi zachytit tak, jak by to teorie vyžadovala. Pro některé složitější teoretické konstrukce nebylo nalezeno odpovídající technické řešení, a to z nejrůznějších důvodů: k podrobné anotaci některých jevů nebylo dost času, anotátorů a peněz (takové jevy by se však způsobem odpovídajícím teorii daly zpracovat v budoucnosti). Jedná se například o subfunktory, které byly přiřazeny automaticky jen se zběžnou ruční kontrolou na základě ad hoc sestavených heuristik, které vycházely z nedokončených a neúplných analýz, neboť práce na tomto poli dosud zdaleka neskončila. Podobně byly přiřazovány i některé gramatémy, neboť přesné vymezení gramatémů v teorii chybělo či naopak data neposkytovala bez další anotace dostatek podkladů

[4)] Často je dokonce otázkou, zda podobné úseky textu nazývat větami, např. zápisy sportovních výsledků nebo šachových partií.

pro jednoznačné určení některých kategorií. Ani přiřazení t_lemmat, která víceméně odpovídají sémoglyfům, nebylo zcela důsledné, protože neexistují přesné specifikace slovníku tektogramatické roviny (ontologie). Mnohé rozdíly bychom z podobných důvodů našli i v anotaci valenčních rámců (tzv. PDT-vallex).

Vybudováním závislostního korpusu tak byla teorie **FGP** podrobena jakési „zkoušce ohněm“, protože věty v ohromném množství reálných textů nabízely téměř všechny možné jazykové jevy. Se všemi si museli anotátoři nakonec poradit, často po dlouhých diskuzích, mnohdy se přijaté řešení ukázalo nevyhovující a muselo se hledat nové. Teorie jako taková však ve zkoušce bezpochyby obstála, většina problémů se navíc vyskytla právě v těch oblastech, kde byly těžkosti předpokládány.

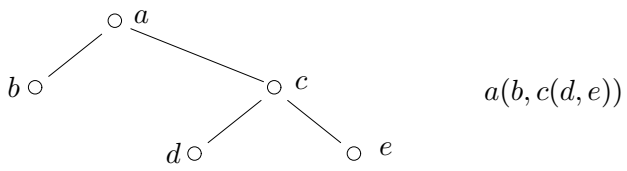
1.3 Anotační schémata

Anotační schéma je abstraktní datový model, který charakterizuje, jakým způsobem budou organizována data a jak v nich budou zachycovány jevy, které jsou anotovány. Anotační schéma není totéž co formát dat, i když v případě **PDT** odpovídá starému anotačnímu schématu formát **FS** a novému **PML**. Konkrétním formátem, který je pouhou implementací schématu, se v tomto oddílu zabývat nebudeme.

Ani samotní anotátoři nemusejí být s anotačním schématem podrobně seznámeni. Nezáleží jim totiž na tom, jakým způsobem jsou v datech zachyceny šipky, které zobrazuje anotační nástroj, jejich úkolem je pouze zaručit, že šipky spojují správné uzly. Právě proto však mohou anotátoři při řešení nestandardních situací porušit anotační schéma a vytvořit v datech chyby.

Staré anotační schéma bylo navrženo v době, kdy začínalo anotování na analytické rovině. Jelikož jednotky analytické roviny odpovídají jednotkám roviny morfologické jedna ku jedné (a vztah morfologické roviny k původnímu textu byl chápán velmi podobně) a analytická rovina byla v tu chvíli jedinou, která do dat vnášela jinou než lineární strukturu, odpovídající pořadí slov v textu, nebylo třeba se při tvorbě anotačního schématu příliš zabývat vztahy mezi rovinami. Text byl rozdělen na soubory[5)] – úseky o pravidelné délce padesáti vět, každý takový soubor obsahoval posloupnost reprezentací jednotlivých vět. Často se proto stávalo, že dvě na sebe navazující věty, které byly později na tektogramatické rovině propojeny koreferenčními odkazy či odkazy na kopírované uzly, ležely každá v jiném souboru.

[5)] Mohlo by se zdát, že dělení na soubory souvisí spíše s formátem než s anotačním schématem. Formát sám však žádné takové dělení nevyžadoval, takže toto rozhodnutí řadíme k vlastnostem anotačního schématu, jinak by bylo třeba mezi formátem a anotačním schématem zavádět ještě další úroveň, „technickou realizaci“ schématu.



Obrázek 1.5: Linearizace stromu

Věta byla reprezentována „na všech rovinách najednou“, tj. v podstatě tak, že analytické struktuře odpovídala struktura dat (neuspořádaný strom lze jednoduše lineárně zapsat za použití závorkování: za každý uzel se do závorek zapíše seznam jeho dětí, viz obrázek 1.5), morfologické informace byly uloženy u jednotlivých uzlů a pořadí slov ve větě bylo určeno číslem přiřazeným každému uzlu (morfologické pořadové číslo): na analytické rovině totiž mohly existovat neprojektivní stromy, tj. takové stromy, které se pomocí závorkování zapsat nedají, pokud dodržujeme pořadí uzlů. (Strom na obrázku by se dal zapsat také jako řetězec $(b)a((d)c(e))$, ze kterého lze zjistit i pořadí uzlů, ale protože neprojektivní stromy nelze linearizovat ani tak, tento zápis se nepoužíval.)

Uzly původně neměly ani jednoznačné identifikátory, protože na ně nebylo nutno nijak odkazovat. Ve chvíli, kdy bylo potřeba anotovat morfologické informace odděleně a později je do dat „přilévat“ (*merge*), ukázala se existence identifikátorů nezbytnou.

Pro potřeby analytické anotace bylo staré schéma dostačující. Problémy nastaly až tehdy, když se data začala anotovat na tektogramatické rovině. Vycházelo se z předpokladu, že uzly vystupující na různých rovinách jsou vlastně stále tytéž objekty, na každé rovině jsou však relevantní odlišné sady jejich atributů. Jakákoliv změna se tedy potenciálně týkala *všech rovin najednou*, pokud se například přidával nebo mazal uzel, dělo se tak na všech rovinách naráz. Obě stromové struktury navíc nemohly být zachyceny linearizovaným způsobem (jako na obrázku 1.5): nestačilo by ani zavedení dvou druhů závorek – např. stromy $a(b, c)$ a $[a, b]c$ se sice ještě dají zapsat zároveň jako $[a(b]c)$, ale pro $a(b, c)$ a $b[a, c]$ již takový zápis neexistuje. Další komplikací byly časté případy, kdy jednomu uzlu jedné roviny odpovídalo několik uzlů roviny jiné, a kromě toho na tektogramatické rovině existovalo mnoho uzlů, které neměly na analytické rovině žádný protějšek.

Jako řešení byla přijata jen drobná změna anotačního schématu: struktura souborů pak odpovídala tektogramatickým stromům a pro pořadí uzlů na tektogramatické rovině byla používána nová tektogramatická pořadová čísla, zatímco analytická struktura byla reprezentována vždy morfologickým

FS: forma, analytická funkce, tektogramatické lema, funktor, morfologické pořadové číslo, tektogramatické pořadové číslo, analytický rodič, skrytý uzel

```
[#10,AuxS,#10,SENT,0,0](
  [vzdal,Pred,vzdát_se,PRED,3,3,1](
    [Černý,Sb,černý,ACT,1,1,3],
    [se,AuxT,se,2,2,3,hide]
  ),[.,AuxK,&Period;,4,4,0,hide]
)
```

CSTS: forma, lema, tag, analytická funkce, analytické pořadové číslo, analytický rodič

```
<s id="ln95048:053-p6s48">
 <f cap>Černý<l>černý-1_^(barva)
    <t>AAMS1----1A----<A>Sb<r>1<g>3
 <f>se<l>se_^(zvr._zájmeno/částice)
    <t>P7-X4----------<A>AuxT<r>2<g>3
 <f>vzdal<l>vzdát
    <t>VpYS---XR-AA---<A>Pred<r>3<g>0
 <d>.<l>.
    <t>Z:-------------<A>AuxK<r>4<g>0
```

Obrázek 1.6: Příklady formátů **FS** a **CSTS** (zjednodušeno): na analytického rodiče je vždy uveden odkaz pomocí jeho pořadového čísla: „*Černý se vzdal.*“ [zm]

pořadovým číslem rodiče.[6)] Přidané uzly tektogramatické roviny, které neodpovídaly na analytické rovině žádnému uzlu, neměly svůj rodič uveden. Naopak uzly analytické roviny, které neměly na tektogramatické rovině svůj protějšek (interpunkce, předložky, spojky, pomocná a modální slovesa apod.) byly na tektogramatické rovině označeny jako *skryté uzly*.

Tato změna však nebyla příliš šťastná. Pokud bylo při pozdějších opravách nutné změnit pořadová čísla jednotek nižší roviny (například při vložení nového slova či při jeho odstranění, což byly poměrně časté jevy při opravách špatného rozdělení vět nebo slov), musela se zakódovaná struktura celá přepočítat. Kdyby se namísto pořadového čísla struktura zapisovala pomocí identifikátoru uzlu, část práce bychom si ušetřili, ale problémy by se opakovaly, pokud bychom přesouvali uzly z jedné věty do druhé (mohlo by se totiž stát, že některý z přesunutých tektogramatických uzlů by měl některý ze zůstavších coby svého analytického syna).

Skrývání uzlů také způsobovalo těžkosti, i když zde byla na vině spíše lidská nepozornost. Před skrytím uzlu bylo nutné převěsit všechny jeho děti na jiné viditelné (neskryté) uzly, které nebyly jeho potomky. Pokud se totiž skryl uzel, skryli se všichni jeho potomci spolu s ním. Tato chyba se bohužel objevila v automatické proceduře, která generovala tektogramatické stromy z analytických, aby se anotátorům ulehčila práce (viz [Böhmová, 2001]). Chyba však byla objevena až ve chvíli, kdy již byla větší část anotací dokončena, takže nebylo možné automatickou proceduru znovu pustit.

[6)] Ve starším formátu **CSTS**, který sloužil při anotování morfologické roviny, a jemuž proto odpovídalo ještě jednodušší anotační schéma, byla pomocí odkazů reprezentována již analytická rovina. Struktura souboru byla proto nerekurzivní a odpovídala pořadí slov v textu – oba starší formáty ukazuje obrázek 1.6.

Nové anotační schéma bylo dokončeno a implementováno až po skončení anotování a oprav závislostního korpusu. Pokud by bylo ke změně došlo v průběhu prací, bylo by to znamenalo značné zdržení, než by všichni pracovníci získali nová data, nové nástroje a přeškolili se. Při opravách **PDT** se proto musely řešit mnohé problémy, které by při použití nového schématu vůbec nenastaly nebo by byly řešitelné o mnoho snadněji.

Nové anotační schéma předpokládá, že soubory odpovídají dokumentům (k pojmu souboru viz také poznámku 5 na straně 14), a tedy je každý soubor uzavřený na veškeré odkazy (kromě odkazů do nižších rovin, jak hned vysvětlíme). Každá rovina popisu má své vlastní uzly, které na sebe směrem dolů odkazují pomocí identifikátorů. Nejnižší dvě roviny, označované písmeny **w** (slovní rovina) a **m** (morfologická rovina), obsahují své jednotky v pořadí odpovídajícím jejich pořadí povrchovému, na morfologické rovině jsou ke každému slovu přiřazené morfologické informace. Zápis věty na analytické (**a**) i tektogramatické (**t**) rovině svou strukturou odpovídá struktuře stromu, pořadí sourozenců je libovolné (pořadí uzlů je určeno pořadovým číslem). Příklad zápisu věty „*Čtvrť pro diplomaty*“ v novém formátu **PML** je uveden na obrázku 1.7: jednotlivé roviny jsou od sebe oddělené, propojené pouze pomocí odkazů (na obrázku naznačeny šipkami). Podrobnou specifikaci formátu **PML** lze nalézt v [Pajas a Štěpánek, 2005].

Změna pořadí uzlů na jedné rovině se tedy nijak nedotýká rovin ostatních. Rovněž přidání uzlu (které v sobě často nese i nutné přečíslování pořadových čísel uzlů následujících přidaný uzel) se značně zjednodušuje, protože se vždy týká jen jedné roviny. Mazání uzlu zůstává poměrně komplikovaným, protože je vždy třeba na všech vyšších rovinách zkontrolovat, že na smazaný uzel neexistuje odkaz, odpadá však nutnost přepočítávat odkazy kódující analytickou strukturu. Přidávání uzlu, které svou povahou odpovídá procesu anotování, kdy byly vlastně všechny uzly postupně „přidávány“, tedy musí probíhat zdola nahoru, od roviny slovní k rovině tektogramatické, kdežto mazání uzlu by mělo probíhat směrem opačným, aby se na každém kroku zachovala konzistence odkazů do nižších rovin.

Obecnými požadavky na anotační schémata se zabývá například článek [Skut a kol., 1997]. Ukážeme, že nové anotační schéma **PDT** jim vyhovuje lépe než staré; jediný požadavek, který nesplňuje, lze navíc snadno zpochybnit. Autoři článku formulují následující požadavky na anotační schémata:

Deskriptivita: gramatické jevy mají být zachycovány, ne vysvětlovány.

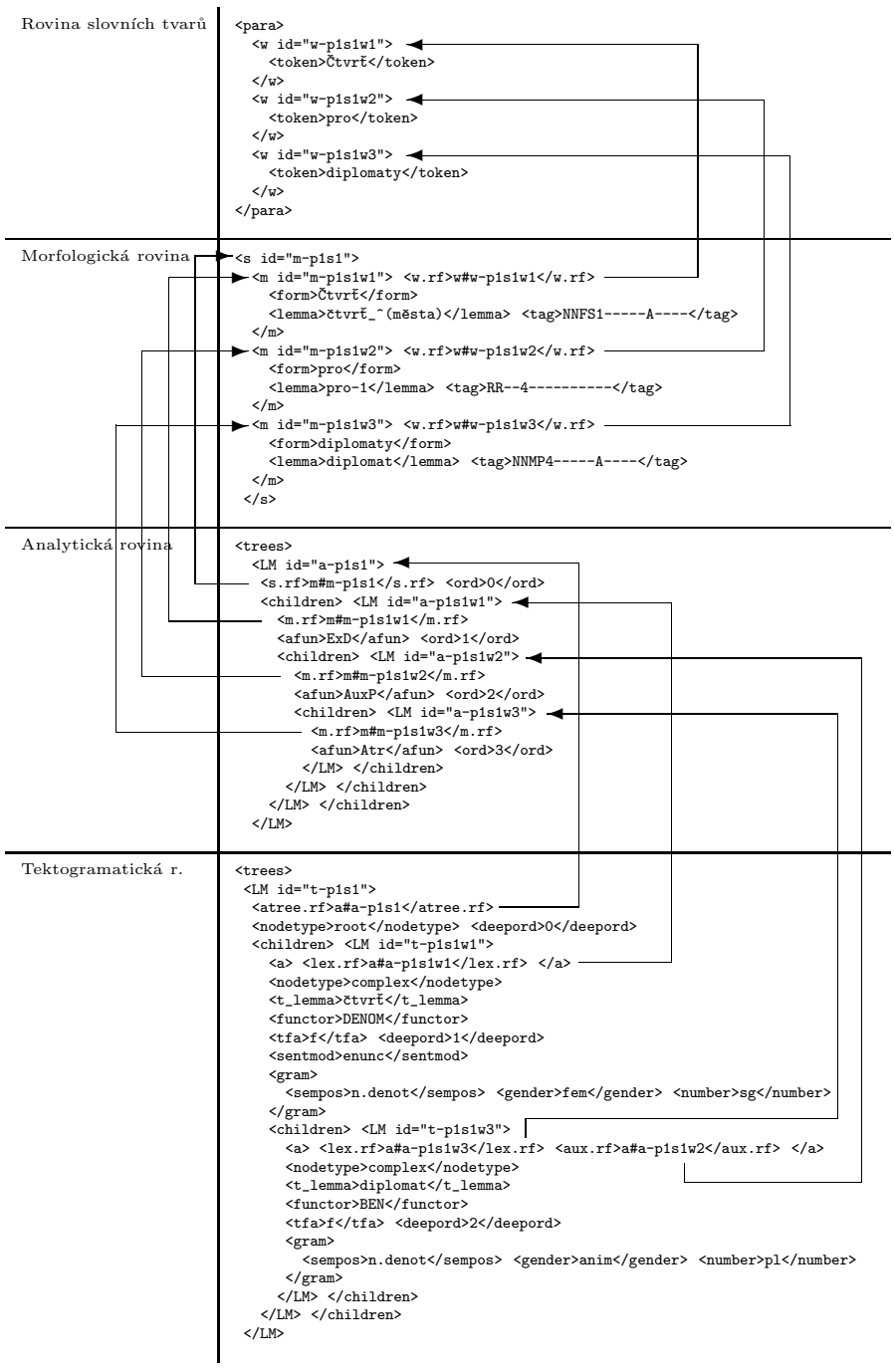
Nezávislost na teorii: anotace by neměla být ovlivněna žádnou teorií, jednotlivé teoretické interpretace by však z ní měly být odvoditelné.

Multistratální reprezentace: jednotlivé roviny popisu by měly být jasně oddělené.

Data jsou výchozí: schéma by mělo umožňovat popis všech jevů, které se v textu vyskytnou. Nejednoznačnosti by měl rozhodovat člověk.

Prvnímu a poslednímu požadavku vyhovují obě schémata **PDT**. Druhý požadavek je diskutabilní, v článku samotném se hned v následujícím odstavci uvádí bezkontextová gramatika se stopami jako typický příklad syntaktické struktury, závislostní stromy používané na analytické rovině však nejsou s frázovými strukturami ekvivalentní, protože nevyznačují stopy (převoditelnost analytických závislostních stromů a frázových struktur rozebírá článek [Xia a Palmer, 2001][7]). Co se oddělení rovin popisu týče, je nové anotační schéma **PDT** důslednější než staré a citovaným požadavkům tedy vyhovuje lépe.

[7] Tektogramatické závislostní stromy stopy víceméně obsahují, protože všechny elidované aktanty jsou do stromu doplněny, takže jejich převod na frázové struktury by měl být jednodušší. Převod opačným směrem je popsán v [Žabokrtský a Kučerová, 2002].



Obrázek 1.7: Příklad formátu PML (zjednodušeno) „*Čtvrť pro diplomaty*“[zm]

Kapitola 2

Nástroje

> Vím, že ty časy nelze dnes nikomu vylíčit. Ostatně stroj dosud pracuje a jeho činy hovoří.
> *Franz Kafka, V kárném táboře*

Hlavním nástrojem pro práci na závislostním korpusu **PDT** byl program **TrEd** („Tree Editor“, [Pajas]), vytvořený Petrem Pajasem v programovacím jazyce Perl ([Wall a kol., 1996]). Oproti dříve používaným nástrojům měl hned několik výrazných výhod:

- Jazyk Perl včetně knihovny Tk, kterou **TrEd** hojně používá pro grafické zobrazování stromů, se dá zkompilovat jak pro operační systém Microsoft Windows, tak Linux (a mnoho dalších). Anotační i opravné práce bylo proto možné provádět na nejrůznějších počítačích bez nutnosti měnit operační systém a anotátoři mohli dál pracovat v prostředí, na něž byli zvyklí.
- Jazyk Perl je jazykem interpretovaným. To sice na jednu stranu přináší jisté zpomalení při provádění programů, na druhou stranu však tato vlastnost umožňuje programu **TrEd** spouštět takzvaná *makra*, tj. vlastně další programy v jazyce Perl, používající knihovny pro práci s korpusovými daty. Protože jazyk Perl není kompilován, nemusí být žádné z maker předem známo a uživatel může při jejich tvorbě využívat celou šíři programovacího jazyka.
- Zdrojový kód programu **TrEd** je volně dostupný a dostatečně dokumentovaný, takže je možné ho různě rozšiřovat a přizpůsobovat novým požadavkům (například při vzniku nového anotačního schématu).

Nadto jsou všechny prostředky pro práci s korpusem objektově orientovány, tedy alespoň do té míry, jakou dovoluje jazyk Perl (výhodou objektově orientovaného přístupu je především snadné propojení s vizuálním editorem). Hlavním objektem je uzel (třída `FSNode`), mezi jehož hlavní metody patří:

`parent`: Vrací rodiče uzlu.

`root`: Vrací kořen stromu, do nějž uzel náleží.

`level`: Vrací hloubku uzlu.

`lbrother, rbrother`: Vrací levého a pravého bratra uzlu.

`firstson`: Vrací prvního syna uzlu zleva.

`following`: Vrací následující uzel stromu při průchodu do hloubky (tzv. průchod „pre-order“): Pokud existuje nějaký syn uzlu n, vrátí prvního zleva (jako metoda `firstson`), jinak pokud existuje pravý bratr, vrátí jej (`rbrother`), jinak najde nejbližšího předka uzlu n (tj. předka s největší možnou hloubkou), který má pravého bratra, a tohoto bratra vrátí. Pokud neexistuje žádný z těchto uzlů (tj. uzel je poslední v pořadí průchodu), vrací metoda hodnotu `undef`.

`children`: Vrací seznam dětí uzlu.

`descendants`: Vrací seznam následníků uzlu (tranzitivní obdoba metody `children`).

Metod je samozřejmě mnohem víc, tyto však budeme nejčastěji potřebovat při popisu jednotlivých kontrol; také je z nich dobře patrné, jak byl implementován závislostní strom. Jeden z uzlů je vždy obsažen ve speciální proměnné `$this`, která označuje *aktuální* uzel – na něm stojí „kurzor“ v případě použití grafického editoru. V rámci souboru je každý strom reprezentován svým kořenem, soubor je pak seznamem kořenů s jedním vyznačeným aktuálním kořenem.

2.1 `Btred` a `ntred`

Program **TrEd** byl používán hlavně k anotování dat na tektogramatické rovině a k ručním opravám dat na všech rovinách. Pro automatické prohledávání a opravování korpusu nebylo jeho grafické prostředí potřeba, což vedlo ke vzniku programu **btred**, který poskytuje programátorům stejné možnosti pro psaní maker jako **TrEd**, ale je možné ho spouštět z příkazové řádky. Program si sám postupně otevře všechny zadané soubory a na každém z nich spustí určené makro (je dokonce možné zadávat, zda se má makro provádět vždy jen jednou pro celý soubor, postupně pro každý strom, nebo dokonce postupně pro každý jednotlivý uzel).

Velmi brzy se však ukázalo, že práce s programem **btred** je neúnosně pomalá. Průchod makra všemi uzly tektogramatické roviny **PDT** trval téměř deset minut, a to i v případě, že makro neprovádělo žádnou akci – nejvíce času totiž zabralo otevírání (a v případě některých formátů také parsing) více než tisícovky souborů. Pokud makro ještě navíc něco skutečně počítalo či dokonce ukládalo změněné soubory, doba výpočtu se několikanásobně prodlužovala. Za takových podmínek nebylo téměř možné psát složitější makra, protože ladění trvalo několik hodin. Bohužel nebylo vždy možné

používat k ladění menší vzorek korpusu, protože (jak již bylo zmíněno v úvodu na straně 2) některé jevy jsou velmi řídké a mají v korpusu jen několik výskytů.

Možným řešením tohoto nedostatku by bylo načíst soubory do paměti jen poprvé a nechat je načtené pro další použití. Celý korpus byl však tak velký, že se do paměti jednoho počítače nevešel. Podobné úvahy vedly nakonec ke vzniku programu **ntred**, tedy síťové verze programu **btred**: ten spočívá v tom, že se nejprve na několika počítačích spustí tzv. *servery*, z nichž každý si načte část dat do paměti, a potom je možné na nich spouštět makra z příkazové řádky přes tzv. *hub*, který s nimi komunikuje – posílá jim zadání a sbírá odpovědi.

Rozdělování dat jednotlivým serverům se původně řídilo srovnávacím testem (*benchmark*), který se snažil zjistit výkon serverů a rozdělit jim data proporcionálně odpovídajícím způsobem. Sestrojit test, který by výkon serverů dobře odhadl, je však velmi těžké. Nakonec se ukázalo, že optimální je rozesílat serverům soubory v dávkách – jakmile server načte do paměti jednu dávku, vyšle požadavek *hubu* a ten mu poskytne další.

Program **ntred** má však také několik nevýhod. Za prvé nefunguje v prostředí operačního systému Microsoft Windows (nepracuje síťové rozhraní); za druhé se výstupy z některých maker musejí ještě dále zpracovávat (například pokud makro počítá počet výskytů určitého jevu, v případě spuštění programem **btred** vrací skutečně tento počet, v případě spuštění programem **ntred** však vrátí několik čísel, totiž částečné součty od jednotlivých serverů, které je pak ještě nutno sečíst). Kromě toho je třeba, aby na všech serverech běžely kompatibilní verze systému a jazyka Perl (a stejné verze programu **btred**) a všechny měly přístup ke společnému systému souborů.

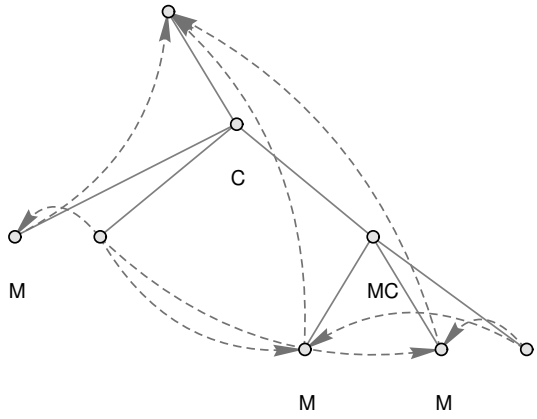
Používání programu **ntred** podstatně zrychlilo práci: počáteční načítání trvá jen několik minut (záleží na počtu serverů, při příliš velkém počtu serverů se však začíná načítání opět zpomalovat vytížením sítě); jednoduché makro, které pouze prochází všemi uzly, běží poté jen několik sekund. Urychlení práce umožnilo vývoj složitějších maker, bez nichž by nemohla existovat celá sestava kontrolních skriptů, maker a programů,[1)] o níž pojednává kapitola 3.

2.2 Spojovací konstrukce

Při vyhledávání jakéhokoliv „lingvistického“ jevu v korpusu **PDT** brzy narazíme na to, že metody `parent` a `children` nám k zjišťování vztahu závislosti nestačí – jak již bylo uvedeno v oddílu 1.2, ne každá hrana

[1)] V poslední fázi kontrol korpusu před vydáním CD trval jeden běh kontrol několik hodin, což by bez síťové verze znamenalo několik dní.

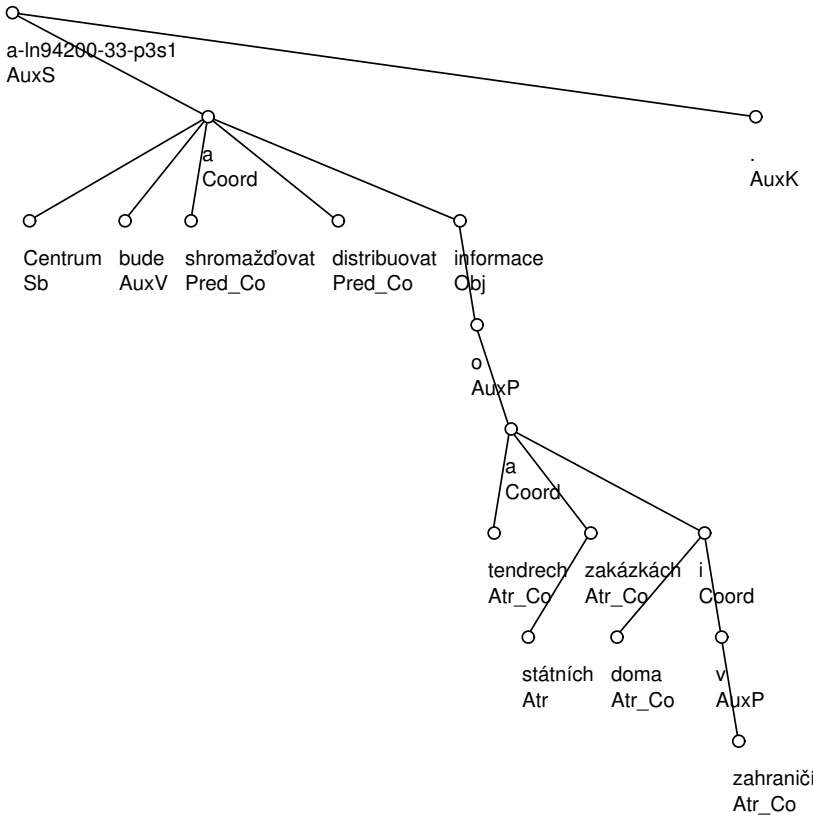
odpovídá relaci závislosti, a naopak ne každá relace závislosti je realizována nějakou hranou.



Obrázek 2.1: Spojovací konstrukce

Na analytické i tektogramatické rovině každý koordinační či apoziční uzel (a na tektogramatické rovině i uzel s funktorem `OPER`) vlastně zaujímá ve stromě místo, na kterém by měli viset jeho synové s příznakem členů takové konstrukce. (V dalším textu budeme koordinační a apoziční konstrukce a konstrukce s funktorem `OPER` nazývat souhrnně *spojovací* konstrukce, abychom se vyhnuli neustálému vyjmenovávání všech tří typů. Uzel odpovídající koordinační (apoziční...) spojce bude nazýván *spojovací uzel*, uzly koordinované (aponované...) budeme nazývat *členy* spojovací konstrukce.) Pokud něco společně rozvíjí členy spojovací konstrukce, je to zavěšeno opět na spojovací uzel, který je zastupuje, ovšem bez příznaku členu konstrukce. Mohli bychom si tedy představit, že kterýkoliv z členů spojovací konstrukce vlastně visí na místě spojovacího uzlu, tedy má jeho rodiče za svého rodiče a jeho syny neoznačené za členy konstrukce za své syny. Takovým rodičům a synům říkáme *efektivní*. Pravidlo pro zachycení spojovacích konstrukcí se však používá rekurzivně, tj. i spojovací uzel může být zároveň členem spojovací konstrukce, takže členy konstrukce, jejímž je spojovacím členem, nemají za efektivní rodiče jeho rodiče, ale opět až jeho efektivní rodiče (viz příklad na obrázku 2.1, spojovací uzel je označen písmenem C, členy spojovací konstrukce písmenem M, relace závislosti je znázorněna šrafovanou šipkou; na obrázku není zachycena poslední možná kombinace, kdy je společné rozvití samo spojovací konstrukcí, protože by se již tak komplikovaný obrázek stal zcela nepřehledným). Algoritmus pro hledání efektivních synů či rodičů by tedy měl být rekurzivní.

Na analytické rovině je situace ještě navíc komplikována tím, že nad jakýmkoliv uzlem může viset předložka či podřadicí spojka – v kombinaci se spojovací konstrukcí dokonce nad i pod spojovacím uzlem (např. „*s Petrem a Pavlem*“, „*se ctí, ale bez peněz*“ [vym], viz také následující odstavec). Předložky a podřadicí spojky budeme nazývat *průchozí uzly* (protože při hledání efektivních rodičů a synů se jimi „prochází“).



Obrázek 2.2: Spojovací konstrukce na analytické rovině: „*Centrum bude shromažďovat a distribuovat informace o tendrech a státních zakázkách doma i v zahraničí.*“ [pdt]

Příklad složitější spojovací konstrukce je uveden na obrázku 2.2: slovesa „*shromažďovat*“ a „*distribuovat*“ jsou koordinována spojkou „*a*“, slovo „*centrum*“ je jejich společným podmětem (subjektem, `Sb`), slovo „*informace*“ je jejich společným předmětem (objektem, `Obj`). Také pomocné sloveso „*bude*“ je zavěšeno jako jejich společné rozvití, podle analytické funkce však poznáme, že se jedná o technickou hranu, jeho zavěšení jen vyjadřuje, že se pomocné sloveso vztahuje k oběma infinitivům. Slovní tvary „*tendrech*“ a „*zakázkách*“ jsou rovněž koordinovány spojkou „*a*“. Všimněme si,

že předložka „*o*“ je zavěšena jako *rodič* koordinační spojky, takže se vztahuje k oběma koordinovaným substantivům. Naproti tomu předložka „*v*“ je přímo rodičem slova „*zahraničí*“, protože nepatří zároveň ke slovu „*doma*“, které je se slovem „*zahraničí*“ koordinováno spojkou „*i*“. Celá konstrukce „*doma i v zahraničí*“ je přitom společným rozvitím koordinace „*tendrech a zakázkách*“.

2.2.1 Hledání efektivních synů a rodičů

Jak tedy hledat efektivní syny? Než se ponoříme do programů, pokusme se algoritmus vyjádřit prostými slovy: mezi efektivní syny uzlu patří jeho synové, kteří nejsou spojovacími uzly; všechny členy spojovacích konstrukcí, jejichž spojovací uzly jeho syny jsou; a konečně, je-li uzel členem spojovací konstrukce, pak také její společná rozvití (pokud je společným rozvitím spojovací uzel, je třeba ho opět nahradit členy spojovací konstrukce), a pokud je i spojovací uzel této konstrukce členem další spojovací konstrukce, tak i její společná rozvití, atd.

```
1   sub GetEChildren ($$) {
2     my ($node,$dive) = @_;
3     return if IsConn($node);
4     my @sons;
5     my $from;
6     push @sons,_FilterEChildren ($node,$dive,0,0);
7     if ($node->{is_member}) {
8       while ($node->{is_member} or !IsConn ($node)) {
9         $from = $node;
10        $node = $node->parent;
11        push @sons,_FilterEChildren ($node,$dive,0,$from);
12      }
13    }
14    return @sons;
15  }
```

Výpis programu 2.1: Funkce `GetEChildren`

Výpis programu 2.1 ukazuje zjednodušený kód funkce `GetEChildren`, která pro zadaný uzel vrací seznam jeho efektivních synů. Druhý parametr, proměnná `$dive`, je ukazatelem na funkci, která vrací jedničku pro průchozí uzly a nulu pro ostatní (na tektogramatické rovině je tento parametr zbytečný, protože průchozí uzly neexistují, takže je kód funkce o něco jednodušší).

V první řadě si uvědomme, že efektivní syny má smysl hledat pouze u uzlů, které se nějak účastní relace závislosti. Řádek 3 je odmítnutím takové otázky pro spojovací uzly (pro něž funkce `IsConn` vrací jedničku).

Vlastní hledání probíhá takto: jako první se projdou synové daného uzlu (řádek 6), funkce `_FilterEChildren`, která je popsána níže, odpovídá za „přeskakování“ průchozích uzlů a případné procházení spojovacích konstrukcí, pokud se mezi syny daného uzlu objeví spojovací uzel. Pokud je sám daný uzel členem spojovací konstrukce (řádek 7), jsou jeho efektivními syny i všechna společná rozvití všech spojovacích uzlů této konstrukce, které jsou jeho předky. Za procházení těchto spojovacích uzlů odpovídá cyklus na řádcích 8–12, který pro každý z nich volá opět funkci `_FilterEChildren`, tentokrát ovšem ještě s informací, který syn byl již v cyklu zpracován (předchozí hodnota proměnné `$node`): ten je třeba z filtrování vynechat, protože již filtrem prošel.

Zde prezentovaný kód se spoléhá na to, že všechny spojovací konstrukce jsou v datech zachyceny správně, jinak by namísto podmínky na řádku 8 musel obsahovat podmínku mnohem složitější a v případě nalezení neočekávaného uzlu ohlásit chybu (a tak tomu v průběhu poanotačních kontrol skutečně bylo). Takto ovšem první člen disjunkce spoléhá na to, že vždy narazí na spojovací uzel, který je členem spojovací konstrukce, a druhý člen pouze přeskakuje průchozí uzly. Rovněž jsou vynechány další části kódu, které nastavovaly defaultní funkci pro přeskakování průchozích uzlů, kontrolovaly existenci výchozího uzlu a podobně.

Filtrovací funkci `_FilterEChildren` ukazuje výpis programu 2.2. První dva její argumenty odpovídají argumentům funkce `GetEChildren`. Třetí argument `$suff` udává, zda v dané chvíli mají být zahrnovány členy spojovacích konstrukcí, či naopak uzly, které členy spojovacích konstrukcí nejsou. Filtr je vlastně automat, který se pohybuje stromem (jeho „hlava“ přitom čte uzel, na který ukazuje proměnná `$node`) a nachází se vždy v jednom ze dvou stavů: buď hledá skutečné syny či společná rozvití, tedy uzly, které nejsou členy spojovacích konstrukcí, nebo naopak hledá členy spojovací konstrukce, která se nacházela na místě syna či společného rozvití. V proměnné `$from` je uložen uzel, z kterého automat přišel a který se má při filtrování vynechat, jak již bylo popsáno v popisu funkce `GetEChildren`.

Pokud je daný uzel spojovacím (řádek 7) a odpovídá stavu indikovanému proměnnou `$suff` (řádky 8 a 9), přidáme mezi efektivní syny původního uzlu všechny jeho syny, které vrátí filtr, volaný tentokrát ve stavu `$suff` = 1 (řádek 10), tedy všechny uzly, které tento spojovací uzel „zastupuje“. Jinak pokud je daný uzel průchozí a má nějaké syny (řádek 11), pustíme na ně další filtr ve stejném stavu, v jakém se zrovna filtr nachází – průchozí uzly nemění stav filtru (řádek 12). Poslední možností je, že narazíme na „obyčejný“ uzel, v tom případě ho zařadíme mezi efektivní syny původního uzlu, pokud odpovídá stavu filtru (řádky 13–16). Řádek 17 je krokem cyklu, který postupně prochází všechny děti původního uzlu.

```
1  sub _FilterEChildren ($$$$) {
2    my ($node,$dive,$suff,$from) = @_;
3    my @sons;
4    $node = $node->firstson;
5    while ($node) {
6      unless ($node == $from) {
7        if (IsConn ($node) and
8             (!$suff && !$node->{is_member}
9             or $suff && $node->{is_member})) {
10         push @sons,_FilterEChildren ($node,$dive,1,0)
11       } elsif (&$dive ($node) and $node->firstson) {
12         push @sons,_FilterEChildren ($node,$dive,$suff,0);
13       } elsif (!$suff && !$node->{is_member}
14                or $suff && $node->{is_member}) {
15         push @sons,$node;
16       }
17     }
18     $node = $node->rbrother;
19   }
20   return @sons;
21 }
```

Výpis programu 2.2: Funkce `_FilterEChildren`

```
1  sub GetEParents ($$) {
2    my ($node,$through) = @_;
3    if ($node->{is_member}) {
4      while ($node->{is_member} or !IsConn ($node)) {
5        $node = $node->parent;
6      }
7    }
8    if (&$through ($node->parent)) {
9      while (&$through ($node->parent)) {
10       $node = $node->parent;
11     }
12   }
13   $node = $node->parent;
14   return $node if !IsConn ($node);
15   return _Expand ($node,$through);
16 }
```

Výpis programu 2.3: Funkce `GetEParents`

```
1   sub _Expand ($$) {
2     my ($node,$through) = @_;
3     my @toCheck = $node->children;
4     my @checked;
5     while (@toCheck) {
6       @toCheck = map {
7         if (&$through ($_)) { $_->children }
8         elsif (IsConn ($_) && $_->{is_member}) {
9           _Expand ($_,$through)
10        } elsif ($_->{is_member}) {
11          push @checked,$_;
12          ()
13        }
14        else { () }
15      } @toCheck;
16    }
17    return @checked;
18  }
```

Výpis programu 2.4: Funkce `_Expand`

Hledání efektivních rodičů je obdobné. Pokud uzel není členem spojovací konstrukce a jeho rodič není spojovacím uzlem, je rodič uzlu jeho rodičem efektivním. Jestliže je rodič spojovacím uzlem a uzel není členem spojovací konstrukce, je uzel společným rozvitím a jeho efektivními rodiči jsou členy spojovací konstrukce. Jestliže je uzel členem spojovací konstrukce, je jeho efektivním rodičem rodič nejvyššího spojovacího uzlu této konstrukce, pokud to není spojovací uzel. Pokud je, budou efektivními rodiči uzlu členy jeho spojovací konstrukce.

Implementace algoritmu se podobá implementaci hledání efektivních synů. I zde existuje hlavní funkce (`GetEParents`, výpis programu 2.3), která volá rekurzivní pomocnou funkci (`_Expand`, výpis programu 2.4). Hlavní funkce pracuje následujícím způsobem:

Pokud je daný uzel členem nějaké spojovací konstrukce, nalezne funkce `GetEParents` nejprve spojovací uzel s nejmenší hloubkou, který daný uzel „zastupuje“ (řádky 3–7, podmínka na řádku 4 je opět zjednodušená jako v případě podmínky na řádku 8 v kódu funkce `GetEChildren`, výpis programu 2.1), v opačném případě uzel „zastupuje“ sám sebe. V dalším kroku funkce popřípadě přeskočí průchozí uzly (řádky 8–12, všimněme si, že podmínka se ptá na rodiče uzlu, nikoliv na uzel sám, po skončení cyklu tedy ukazuje proměnná `$node` na průchozí uzel s nejmenší hloubkou nad zástupným uzlem). Rodič takto nalezeného uzlu (řádek 13) je buď přímo efektivním rodičem původního uzlu, a to tehdy, když není spojovacím uzlem

(řádek 14), nebo je spojovacím uzlem konstrukce, jejíž všechny členy jsou efektivními rodiči původního uzlu. Členy konstrukce vrací pomocná funkce `_Expand`.

Funkce `_Expand` (výpis programu 2.4) pracuje s dvěma pomocnými seznamy, `@toCheck` a `@checked`. První vždy obsahuje uzly, které je ještě třeba projít, do druhého se v průběhu výpočtu ukládají nalezené členy spojovací konstrukce. Jako první projdou procedurou synové daného spojovacího uzlu (řádek 3). Cyklus samotný pak vypadá následovně: každý průchozí uzel je v seznamu `@toCheck` nahrazen svými dětmi (řádek 7); každý spojovací uzel, který je zároveň členem spojovací konstrukce, je rekurzivně expandován (řádky 8 a 9); ostatní členy spojovací konstrukce se zařadí mezi nalezené uzly a ze seznamu se vymažou (řádky 10–13); ostatní uzly se rovněž vymažou (řádek 14). Tento cyklus se provádí tak dlouho, dokud v seznamu `@toCheck` zbývají nějaké uzly.

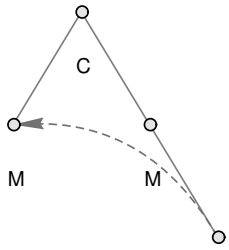
2.2.2 Jiné způsoby zachycení

Jak lze z výše uvedeného popisu vidět, jsou funkce pro zjišťování efektivních rodičů a synů poměrně komplikované. Položme si otázku, zda příčinou této složitosti není nevhodný návrh zachycení spojovacích konstrukcí, tj. zda by se v případě nějakého jiného zachycení těchto konstrukcí nedali nalézt efektivní rodiče a synové jednodušeji.

Jisté zjednodušení by přinesla rezignace na zachycení složité vnitřní struktury spojovacích konstrukcí, které se do sebe vnořují. Tím bychom však přišli o informaci, která může být pro pozdější výzkum spojovacích konstrukcí zajímavá (navíc se nám zdá téměř nepochybné, že ve spojeních typu „*kladivo a hřebíky nebo hrábě a lopata*“ [vym] skutečně jde o koordinaci koordinací).

Další možností, jak zachytit spojovací konstrukce, by bylo „předpočítat“ pro každý uzel seznam jeho efektivních synů a rodičů a tento seznam uložit do jeho atributu. Při každém převěšení uzlu, při změně funktoru ze spojovacího uzlu na jiný či z jiného uzlu na spojovací nebo při změně hodnoty atributu `is_member`, který označuje členy spojovacích konstrukcí, by se tato informace musela znovu přepočítávat, a to nejen u všech uzlů, které byly dříve členy spojovací konstrukce spolu s měněným uzlem, ale i u všech, které se změnou do takové konstrukce teprve mají dostat. Součástí spojovacích konstrukcí je navíc přes 15 procent uzlů, takže by se značně zvětšil objem dat.

Tomuto přístupu se blíží zachycení koordinací v německém syntaktickém korpusu **TIGER** (viz [Brants a Hansen, 2002]), který však nepoužívá stromy závislostní, ale frázové; jeho anotační schéma navíc nezachycuje strukturu stromů strukturou souboru jako **PDT** (tj. nepoužívá linearizovaný zápis stromů), ale pouze pomocí odkazů, takže se tak jako tak musí celá struktura stromu při parsingu dat počítat. S odhlédnutím od těchto odlišností je

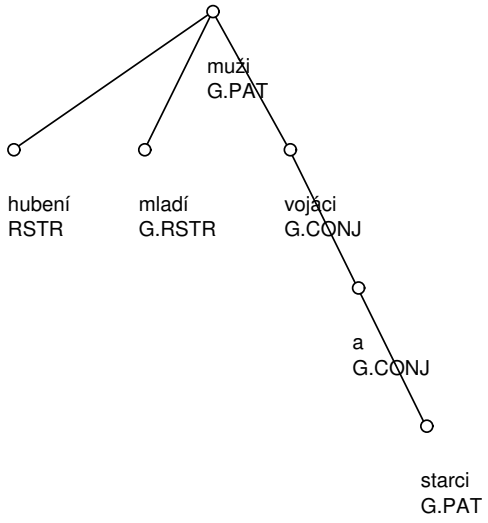


Obrázek 2.3: Zachycení společného rozvití na způsob korpusu **TIGER**. Šipka představuje sekundární hranu, C je koordinační uzel, M členy koordinace.

hlavní rozdíl mezi **PDT** a korpusem **TIGER** v zachycení společného rozvití – zavěšení společného rozvití v **PDT** bylo popsáno v oddílu 2.2; v korpusu **TIGER** se zachycuje následujícím způsobem: pokud si představíme, že koordinační spojka rozděluje větu na části v_1 a v_2, přičemž do části v_i vždy patří množina koordinovaných členů m_i, pak společné rozvití patřící do části v_i visí vždy na nejbližším členu množiny m_i a k ostatním členům koordinace od něj vedou tzv. *sekundární hrany* (tedy další odkazy jiného typu). Pokud bychom použili tuto ideu (viz obr. 2.3) pro spojovací konstrukce v **PDT** a předpočítávali pouze stromovou strukturu (tj. nikoliv sekundární hrany ani tabulku identifikátorů uzlů), bylo by hledání efektivních synů ještě o něco komplikovanější než za současného řešení: při hledání efektivních synů členu m spojovací konstrukce bychom museli zkontrolovat všechny syny všech členů této konstrukce, zda od nich nevede sekundární hrana k uzlu m (což znamená oproti metodě `GetEChildren` prohledávání do větší hloubky). Hledání efektivních rodičů uzlu n by odpovídalo nalezení druhého uzlu sekundární hrany, tj. uzlu s příslušným identifikátorem (hledání by se dalo omezit na členy spojovací konstrukce, jichž je rodič uzlu n členem, pak by byla složitost hledání stejná jako v případě funkce `GetEParents`).

Opět jiným způsobem zachycuje koordinaci Meľčuk, který také podobně jako **PDT** používá závislostní stromy (viz [Meľčuk, 1988]). První člen koordinace je zavěšen na uzlu, na kterém závisí, ostatní členy koordinace visí postupně na sobě a poslední z nich je od předposledního oddělen koordinačním uzlem (viz obrázek 2.4). Hrany procházející koordinační spojkou jsou označeny specifickou funkcí. Aby se odlišila společná rozvití od rozvití jednotlivých členů koordinace, zavádí Meľčuk *seskupování (grouping)*: společné rozvití je zavěšeno na prvním členu koordinace, ale není označeno jako člen seskupení[2)] (u uzlu „*hubení*“ chybí příznak G); naopak rozvití členu koordinace, které není rozvitím společným, za člen seskupení označeno je (příznak G u slova „*mladí*“). Samotný příznak, zda je daný uzel

[2)] Meľčuk nepoužívá žádný příznak pro označení členů seskupení, vyznačuje seskupení pomocí závorek kolem celého řetězu slov.



Obrázek 2.4: Použití analogie Meľčukova seskupování: „*Hubení mladí muži, vojáci a starci*“ [vym]

členem seskupení (v příkladu označený G) by ovšem obecně k zachycení spojovacích konstrukcí nestačil: pro skládání spojovacích konstrukcí je totiž potřeba do sebe seskupení vnořovat, a je tedy třeba přinejmenším označit, kolika seskupení je daný uzel členem (nebo každé seskupení jednoznačně označit identifikátorem, alespoň v rámci věty).

Kdybychom k vyznačování seskupení sahali pouze tehdy, je-li to třeba (tj. jestliže bychom například předpokládali, že pokud není seskupení vyznačeno, nemá žádná společná rozvití, a vyznačovali bychom pouze ta seskupení, která nějaké mají), stačilo by tento příznak na tektogramatické rovině vyplnit v méně než devíti tisících konstrukcí (číslo odpovídá počtu spojovacích konstrukcí se společným rozvitím) v celkem téměř osmnácti tisících případů (číslo odpovídá počtu uzlů, kterým by se musel vyznačit příznak seskupení), což je podstatně méně, než bezmála osmdesát tisíc uzlů s vyplněným atributem `is_member` v současných datech. Objem dat by se tedy dokonce zmenšil.

Hledání efektivních rodičů by sice bylo o něco pomalejší, ale toto zpomalení by se týkalo pouze společných rozvití, která by byla snadno identifikovatelná podle odlišné hodnoty atributu pro označení seskupení oproti rodiči. Hledání efektivních synů členů spojovacích konstrukcí označených za členy seskupení by bylo srovnatelně náročné: stačilo by najít první člen této konstrukce a najít jeho syny, kteří nejsou označeni za členy seskupení.

Zdá se dokonce, že zachycení spojovacích konstrukcí s použitím Meľčukových seskupení by bylo vhodnější i pro anotování: mnohem menší

četnost by měla častá chyba anotátorů **PDT**, kteří často zapomínali označit člen spojovací konstrukce příslušným atributem, takže byl anotován jako společné rozvití. Častěji by se pak asi vyskytovala opačná chyba, kdy je společné rozvití zachyceno pouze jako rozvití prvního členu koordinace – tuto chybu však můžeme považovat v jistém smyslu za *menší*, protože společné rozvití je zavěšeno alespoň na jednom z uzlů, na nichž skutečně závisí, kdežto v případě první chyby není žádná ze zachycených relací závislosti pravdivá. Jednoznačné rozhodnutí, zda se v daném případě jedná o společné rozvití či nikoliv, navíc v mnohých případech ani neexistuje, protože jsou možné obě interpretace.

Jedinou větší nevýhodou Mel̆čukova přístupu je nemožnost zachytit členy spojovací konstrukce s různými funkcemi, protože pokud je konstrukce vícečlenná, mají všechny členy kromě prvního a posledního funktor, odpovídající typu konstrukce (např. hodnota `CONJ` použitá na obrázku 2.4). Tomu by se však dalo předejít zavedením technických uzlů, které by měly tento funktor a oddělovaly by jednotlivé členy spojovacích konstrukcí, z nichž by již každý mohl mít svůj vlastní funktor.

Domníváme se však, že hlavním argumentem pro zachycení spojovacích konstrukcí způsobem, který byl použit v **PDT**, je skutečnost, že k popisu libovolně složité struktury stále vystačíme se stromem a jediným atributem `is_member`, který nabývá pouze hodnot 0 nebo 1. Mel̆čukovo seskupování by nás naproti tomu nutilo používat buď atribut s počtem seskupení, kterých je uzel členem, nebo dokonce odkazy na identifikátory seskupení. Navíc spojovací konstrukce zachycená Mel̆čukovým způsobem upřednostňuje svůj první člen, takže vlastně „svádí“ k zanedbávání členů ostatních (i když pro tento přístup podává Mel̆čuk několik závažných argumentů). Naopak řešení použité v **PDT** „nutí“ uživatele používat dříve popsané funkce, protože by jinak ztratil informaci o všech členech spojovací konstrukce.

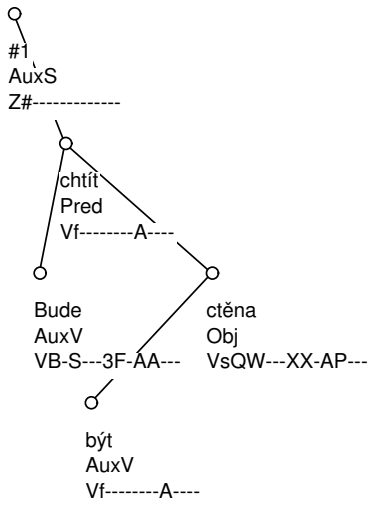
Tato pozorování tedy můžeme shrnout: pokud chceme zachovat zachycení složité struktury spojovacích konstrukcí, nechceme se vzdát stromu coby reprezentace věty a nechceme příliš zvětšovat objem dat, můžeme sice stále nalézt několik různých způsobů zachycení spojovacích konstrukcí, při hledání efektivních rodičů a synů však budeme v takovém případě vždy muset rekurzivně procházet množiny uzlů v podstatě stejné velikosti jako v současnosti, a tedy si práci podstatně nezjednodušíme.

Dodejme ještě, že na analytické rovině je situace ještě složitější: v určitých situacích mohli anotátoři používat tzv. *kombinované funkce*, kterými se označuje člen, který může stát v rámci věty ve více funkcích, aniž by se tím podstatně měnil její význam, tj. věta není striktně homonymní v závislosti na funkci tohoto členu (např. „*Věnoval dětem peníze na dům*“ [vym], kde „*na dům*“ může záviset jak na slovese „*věnoval*“, tak na substantivu „*peníze*“). Člen s kombinovanou funkcí je zavěšen na nejhlubším uzlu, ke kterému se může vztahovat, a podle určitých pravidel může záviset na jeho rodiči či dokonce na více jeho předcích. Zahrnout hledání uzlů

s kombinovanými funkcemi do funkce na hledání efektivních synů by znamenalo drastické zpomalení, protože hloubka zanoření je u těchto konstrukcí neomezená a neexistuje rychlý způsob, jak se ujistit, že se v podstromě daného uzlu nevyskytuje člen s kombinovanou funkcí, který na daném uzlu závisí. Naštěstí tvoří uzly s kombinovanými funkcemi jen necelou čvrtinu procenta všech analytických uzlů, takže pokud nám nejde o jednotlivé případy a zabýváme se jevem, který má dostatečný počet výskytů, dají se uzly s kombinovanými funkcemi zanedbat. V opačném případě jejich závislosti můžeme řešit zvlášť ve zvláštní funkci.

2.3 Další nástroje

Řešení spojovacích konstrukcí nebylo jediným lingvisticky motivovaným problémem. V jistém smyslu podobné bylo například stanovení mluvnických kategorií složených slovesných tvarů (užitečné jednak na analytické rovině, např. pro hledání objektů vyjádřených infinitivem, jednak při udělování gramatémů slovesům na rovině tektogramatické). Situaci ilustruje obrázek 2.5: sloveso „*chtít*“ není predikátem v infinitivu, třebaže tomu odpovídá jeho tag, mezi jeho syny je totiž pomocné sloveso „*bude*“ v budoucím čase. Podobně „*ctěna*“ není slovesem v pasivním tvaru, ale v infinitivu, což je opět dáno nikoliv tagem slovesa, ale jeho kombinací s pomocným slovesem „*být*“ v infinitivu. Na analytické rovině bylo proto nutné hledat mezi dětmi slovesných uzlů pomocná slovesa, na rovině tektogramatické vede na tato slovesa odkaz odpovídající relaci R teorie **FGP**.



Obrázek 2.5: Analytická struktura složeného slovesného tvaru: „*Bude chtít být ctěna*“ [vym]

Kromě toho existovaly desítky různých pomocných nástrojů (většinou kratších programů v jazyce Perl nebo skriptů pro Bourne Again SHell) pro rozmanité úkoly: třídění výstupů, vyhledávání uzlů v datech podle jejich identifikátoru či „slévání“ různých anotací dohromady. Všechny nástroje, které sloužily ke kontrolám a opravám **PDT**, byly spouštěny v operačním systému Linux. Mohly tak bezprostředně využívat výsledků programu **ntred** a daly se snadno spojovat pomocí rour a pojmenovaných rour. Všechny nástroje proto musely dodržovat jisté konvence, aby se daly libovolně kombinovat, např. pokud měl být výstupem programu seznam míst, na nichž se vyskytoval sledovaný jev, muselo být místo označeno způsobem, který se dá předat programu **TrEd** jako seznam souborů, tj. vždy jeden výskyt na jeden řádek (označení pozice uzlu vracela funkce `Position`). Editor potom mohl skákat aktuálním uzlem po jednotlivých výskytech jevu. Pokud výstup obsahoval nějakou další informaci (např. klasifikoval jednotlivé výskyty), musela být tato informace na každém řádku vždy *před* označením pozice a oddělena od ní tabulátorem, aby se na výstup dal spustit další nástroj, program `rcut` („right cut“, obdoba programu `cut`, která ale počítá sloupce zprava). Podobým způsobem byly upravovány i další programy, aby je bylo možné snáze používat pro zpracování dat (například program `join` si nedokáže poradit se vstupem, který obsahuje duplicitní hodnoty ve sledovaných polích).

Specifickým problémem bylo slévání anotací. Anotování valenčního slovníku, koreference a aktuálního členění totiž probíhaly současně s opravami (pokud při slévání došlo k chybě, protože se data příliš změnila, muselo se problematické místo opravit ručně). V případě, kdy se předem vědělo, že změny budou významné, byl sestaven seznam „nedotknutelných“ dat (realizovaný jako seznam identifikátorů technických kořenů), pro který existovaly další nástroje, jednak umožňující dotázat se přímo z prostředí editoru **TrEd**, zda je editovaný soubor „dotknutelný“, jednak krátké programy, které rozdělovaly seznam pozic na dotknutelné a nedotknutelné.

Při zpracování valenčního slovníku, který byl reprezentován jako **XML** dokument (viz bod „Valence“ na straně 60 a [Hajič a kol., 2003]), jsme používali program **xsh** Petra Pajase. Jde o nástroj pro zpracování **XML** souborů, který se podobá unixovému shellu a ke struktuře dokumentu přistupuje podobně jako shell k systému souborů, jenže navíc dovoluje používat výrazy jazyka XPath a kód v jazyce Perl. Tento nástroj byl použit rovněž při převodu dat do nového formátu **PML**, založeného na **XML**.

Další drobné nástroje sloužily k sledování počtu změn dat v průběhu oprav. Podrobněji budou popsány spolu se svými výsledky v oddílu 3.2.2. Podobně, tedy teprve spolu se svými výsledky, budou popsány i další úžeji zaměřené nástroje, používané při kontrolách a při jejich sledování.

Jak již bylo řečeno, editor **TrEd** je určen k úpravám anotací na analytické a tektogramatické rovině. Při kontrolách dat se však často naráželo i na chyby na rovinách nižších, morfologické či slovní. **TrEd** sice umožňuje měnit hodnoty atributů těchto rovin, ale některé z nich jsou velmi složité

a vyžadují další nástroje – jedná se hlavně o atributy `lemma` a `tag`, pro jejichž anotování je nezbytná morfologická analýza (podrobný popis viz [Hajič, 2004]). Program pro morfologickou analýzu ovšem vyžaduje vstup ve formátu **CSTS** (viz obrázek 1.6) a ve stejném formátu vrací výstup, takže musel být „obalen“ dalším programem, aby s ním mohl **TrEd** komunikovat a nabízet anotátorovi možná lemata a tagy pro každé označené slovo v „lidsky“ pochopitelném formátu.

Kapitola 3

Kontroly

Apaišunam – nechuť k vyhledávání chyb znamená nehledat
chyby nebo je zbytečně neopravovat.
Bhagavadgíta

Po vydání první verze **PDT**, která vyšla koncem srpna 2001, se data nějakou dobu vůbec neměnila. Uživatelé **PDT** však postupně přicházeli s připomínkami, které nakonec vedly k tomu, že se data začala znovu měnit. Asi měsíc po vydání CD byly nejdříve odstraněny duplicitní identifikátory vět, o další měsíc později byly jednoznačné identifikátory přiřazeny všem uzlům – tehdy se již počítalo s tím, že data budou znovu vydána v další verzi korpusu. Nová verze se však od první měla lišit hlavně existencí tektogramatické roviny, s opravami nižších rovin se příliš nepočítalo.

Asi rok a půl po vydání první verze **PDT** byla data podrobně zkoumána při hledání „povrchových“ valenčních rámců sloves ([Ondruška, Panevová a Štěpánek, 2003], práce byla představena v březnu 2003). Protože výsledky hledání byly ručně procházeny, bylo nalezeno mnoho chyb na analytické i morfologické rovině. Data však byla opravena teprve v průběhu dalšího půlroku, kdy také vznikly funkce pro hledání efektivních synů a rodičů (viz oddíl 2.2.1), s jejichž použitím byl experiment opakován, aby se získaly přesnější výsledky. Ručně bylo opraveno asi 550 chyb, hlavně na morfologické a analytické rovině, několik desítek systematických chyb bylo opraveno automaticky (opravy probíhaly až do února 2004).

Zároveň s tím však již od léta 2003 probíhaly první kontroly dat na tektogramatické rovině: soubory přicházely od anotátorů postupně, zároveň na datech probíhalo anotování koreference a aktuálního členění. Neustálý vývoj anotačního nástroje **TrEd** (viz kapitola 2) si občas vyžádal i drobné změny anotačního schématu nebo formátu dat. Nové verze se ovšem nedostávaly anotátorům do rukou ihned ani ke všem ve stejnou dobu, takže bylo třeba neustále ověřovat, že všechna data odpovídají poslední specifikaci.

Když vznikla první makra pro kontrolu formátu dat, ukázalo se, že by se podobným způsobem dalo kontrolovat i dodržování složitějších anotačních

pravidel. Začaly vznikat první „lingvistické“ kontroly, a jak se zdokonalovaly nástroje, které se v nich používaly, objevovaly se stále nové a obtížnější úkoly.

Velká část kontrolních maker vznikla na základě anotačního manuálu [Mikulová a kol., 2005]. Pro většinu pravidel v něm uvedených existovala makra ověřující jejich dodržování. Část maker vznikla po náhodném objevení chyby, protože bylo třeba zjistit, zda se chyba nevyskytuje v datech opakovaně. Mnoho kontrolních maker vzniklo díky „automatickému hledači chyb“, který navrhl a implementoval Zdeněk Žabokrtský (srov. [Dickinson, 2005]). Jednalo se o makro, které vycházelo z následujícího tvrzení: *Jestliže najdeme dva podstromy s totožným povrchovým pokrytím, měly by být co do struktury a ohodnocení hran identické až na funktor kořene.* Skript vypisoval a třídil všechny typy podstromů, které sice měly totožné povrchové pokrytí,[1)] ale jejich struktura či ohodnocení hran se lišily. Při prohlížení takových skupin s větší četností bylo objeveno mnoho druhů chyb na obou strukturních rovinách.

3.1 Typy kontrol

Kontrolní makra je možné srovnávat a rozdělovat podle nejrůznějších hledisek. Časová posloupnost a důvod vzniku byly již naznačeny v předchozích odstavcích, v následujících pododdílech budou makra rozdělena podle toho, které části dat se nejvíce týkala – buď šlo o kontrolu technického rázu, která kontrolovala formát souboru, nebo o kontrolu zaměřenou na nějaký lingvistický jev. Taková kontrolní makra je pak dále možné rozdělit podle toho, na které rovině byly primárně opravovány chyby, i když v mnoha případech zjišťovala makra informace z více rovin najednou. Chyby nalezené makrem pak nebývaly vždy všechny stejného druhu a jejich příčiny mohly ležet na kterékoliv z makrem sledovaných rovin.

Pořadí, v jakém byla kontrolní makra spouštěna, nebylo zcela libovolné. V první řadě bylo třeba zkontrolovat existenci a formát souborů, jinak by další kontroly neměly smysl. Mnoho analytických kontrol se spoléhalo na informace z morfologické roviny, většina tektogramatických kontrol využívala všechny ostatní roviny. Bylo tedy vhodné pouštět kontroly pro „nižší“ roviny dříve. Podobné závislosti kontrol vznikaly i v rámci jednotlivých rovin: například na tektogramatické rovině neměla smysl žádná kontrola využívající analytickou rovinu, dokud nebyly zkontrolovány odkazy do analytické roviny.

V následujícím textu nejsou kontrolní makra uvedena zdaleka všechna. Snažili jsme se vybrat ta, která byla úspěšná a nacházela větší počty chyb, a zároveň taková, která odkrývala nečekaná úskalí či další problémy, které bylo třeba řešit.

[1)] Povrchovým pokrytím nemusíme rozumět pouze posloupnost slovních tvarů, ale také např. posloupnost morfologických tagů, dvojic lema–tag apod.

V době, kdy byla kontrolní makra psána a používána k opravám dat, byla tříděna podle toho, jaký byl jejich úkol. Existoval seznam jevů, který byl neustále rozšiřován a v němž každý jev obdržel své číslo. Všechna makra, která se jevu týkala, měla pak v názvu totéž číslo. Existovaly tři základní skupiny maker:

`find`: Tato skupina obsahovala makra, jejichž úkolem bylo hledat „podezřelé“ výskyty nějakého jevu, tedy místa, kde se pravděpodobně objevila hledaná chyba. Pro každý jev bylo makro z této skupiny psáno jako první. Pokud byl jeho výstup krátký (chyba měla jen několik výskytů), opravovaly se jednotlivé výskyty ručně. Pokud bylo nalezených problematických míst mnoho (přes několik set), procházelo se ručně jen několik desítek výskytů v náhodném pořadí, aby bylo možné rozhodnout, zda je třeba makro ještě předělat, protože považovalo za chyby i některé správné konstrukce, nebo zda je možné napsat makro ze skupiny `fix`, které by většinu chyb opravilo automaticky. Existovaly ale bohužel i problémy, které pro rozhodnutí o tom, zda je daná konstrukce správná či nikoliv, potřebovaly lidský zásah (často se jednalo o sémantická kritéria). V takovém případě musel anotátor procházet velký počet vět, z nichž mnohé byly anotovány správně. Pokud to bylo možné, zobrazovaly se v takovém případě anotátorovi věty nějak uspořádané, aby mohl přeskakovat bloky správných vět a jeho práce se urychlila (např. při kontrole všech `DIR1` vyjádřených příslovcem mohl anotátor přeskočit stiskem jedné klávesy všechny výskyty adverbia „*odtud*“, protože se dalo předpokládat, že jsou všechna anotovaná správně; viz také bod „Poznámky anotátorů“ na straně 65).

`fix`: Do skupiny `fix` patřila makra, která měnila data, tj. opravovala příslušnou chybu. Tato skupina byla nejmenší, protože mnoho problémů se automaticky opravit nedalo nebo byla ruční oprava rychlejší než ladění složitého makra.

`check`: Makra v této skupině fungovala stejně, jako makra typu `find`, ale obsahovala navíc seznam výjimek, tj. většinou identifikátorů uzlů, které odpovídající makro ze skupiny `find` označilo za chyby, i když o chyby nešlo. Data byla v pořádku, když žádné z maker typu `check` nevracelo žádný výstup – data neobsahovala žádnou chybu. Pro mnohé jevy však v této skupině makro neexistovalo, protože se nepodařilo rozhodnout sporné případy, na které narazili anotátoři v průběhu ručních oprav, nebo protože výjimek by bylo příliš mnoho.

Všechna makra obsahovala stručnou dokumentaci, která popisovala hledaný či opravovaný jev a případné složitější kroky programu. Každé makro z těchto tří skupin bylo standardním způsobem spustitelné (tj. jako makro programu **ntred** bez dalších argumentů) a vracelo seznamy chyb ve standardním formátu (který byl popsán v oddílu 2.3). Kromě toho ještě existovala skupina `misc`, která obsahovala další pomocná makra a programy,

například pro případy, kdy pro provedení kontroly nestačil jeden průchod daty.

Skupina	Počet maker	Autoři největšího počtu maker
`find`	469	Jan Štěpánek (139), Zdeněk Žabokrtský (122)
`fix`	135	Petr Pajas (59), Jan Štěpánek (38)
`check`	196	Jan Štěpánek (67), Zdeněk Žabokrtský (62)

Tabulka 3.1: Počty kontrol v jednotlivých skupinách

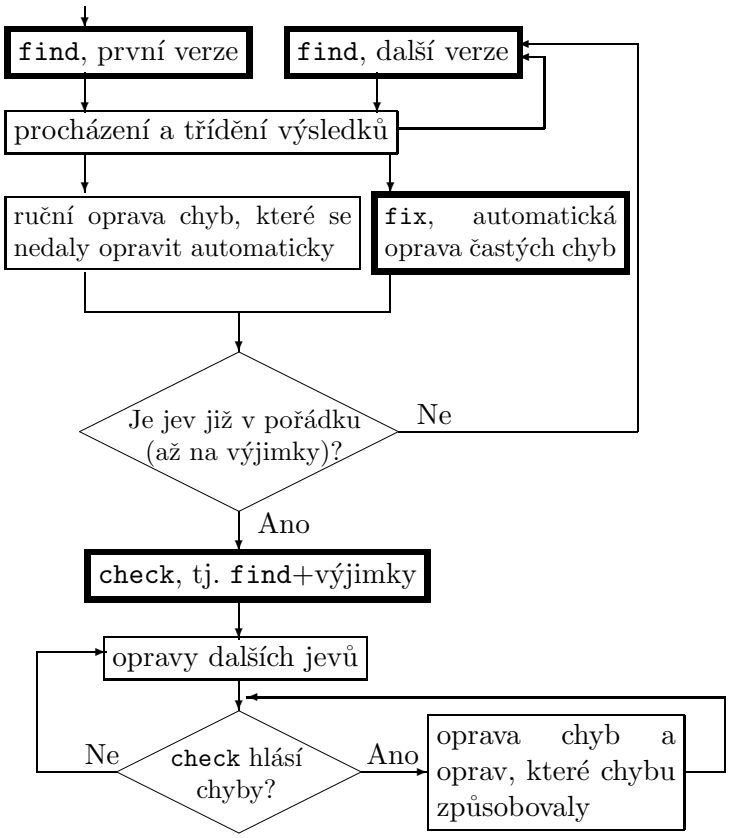
Počty maker v jednotlivých skupinách ukazuje tabulka 3.1 spolu s uvedením autorů největšího počtu maker v každé skupině.

Makra ze skupiny `check` byla opakovaně spouštěna, aby se periodicky ověřovalo, že se v datech při opravách neobjevují nové chyby. Pokud se nějaké objevily (a to se stalo téměř pokaždé), museli je anotátoři ručně opravit. (Popsaný způsob samozřejmě teoreticky nezajišťuje, že se počet nalezených chyb postupně snižuje. Mohla by existovat oprava, která by pokaždé způsobila nové chyby, které by se nedaly opravit bez toho, aby opět vznikaly chyby nové. Jedinou pojistkou proti tomu bylo pravidelné procházení výstupů všech kontrol člověkem.)

Schéma postupu při vytváření maker pro jeden jev je znázorněno na obrázku 3.1, tučné rámečky označují vznik nebo změnu makra. Lepšímu pochopení celého procesu může pomoci i následující fiktivní příklad: Představme si, že jsme se rozhodli zkontrolovat, zda pád slova pod předložkou na analytické rovině vždy odpovídá pádu vyžadovanému předložkou.[2)] Vytvoříme první verzi makra `find`, která bude vypisovat pozici každého uzlu analytické roviny, který reprezentuje předložku (tj. má analytickou funkci `AuxP`), pokud na páté pozici jeho tagu (ta odpovídá pádu) nebude stejný znak jako na páté pozici tagu některého z jeho synů. Výsledných pozic bude velmi mnoho, takže se rozhodneme je nějakým způsobem roztřídit, třeba právě podle dvojice [pád předložky, pád syna]. Zjistíme, že nejčastější neshodou je dvojice [4,-], následují dvojice [2,-], [7,-] a [6,X]. Jedná se tedy vždy o situaci, kdy syn předložky pád vůbec nevyjadřuje nebo je nesklonný. Pohledem do dat zjistíme, že situaci odpovídají dva případy:

- Synem předložky je spojovací uzel, jednotlivé členy spojovací konstrukce již mohou mít správný tvar. Další verze makra proto musí procházet spojovací konstrukce a hledat, ke kterým uzlům se předložka vztahuje (srov. s oddílem 2.2).

[2)] Podobná kontrola se částečně prováděla již při opravách **PDT** 1.0, částečně při kontrolách valence (viz bod „Valence“ na straně 60).



Obrázek 3.1: Schéma postupu kontroly jednotlivými fázemi

- Syn předložky je nesklonný, popř. se vůbec nejedná o jméno („*Hrál hlavní roli v Nedotýkat se*“ [vym]). Další verze makra proto musí ignorovat všechny případy, kdy slovo patřící k předložce nevyjadřuje pád.

Nová verze makra sice sníží počet neshod, docela v pořádku však ještě nebude. Náhodně vybraný vzorek dat nám ukáže další dva zdroje chyb:

- Předložka je poslední částí víceslovné předložky a mezi její děti tedy patří i nějaká další součást složené předložky, která zdánlivě porušuje shodu pádů (např. „*ve shodě s*“: nejvyšším uzlem je „*s*“, které vyžaduje sedmý pád, ale „*ve*“ a „*shodě*“, přestože jsou jeho syny, ho nemají: mají uveden šestý). Další verze makra tedy musí ignorovat součásti složených předložek.
- Slovo, které považujeme za předložku, má sice analytickou funkci `AuxP`, ale nejedná se o předložku, nýbrž o jiný slovní druh, který je součástí složené předložky (např. „*rámci*“ ve složené předložce „*v rámci*“).

Morfologická analýza nám nedává žádnou informaci, jaký pád taková předložka vyžaduje, další verze makra musí tedy takové případy ignorovat.

- Dejme tomu, že slovní tvar „*oceli*“ je špatně zachycen ve slovníku morfologického analyzátoru. Ten mu vždy předepisuje třetí pád, zatímco ve skutečnosti se může jednat také o pád druhý, pátý a šestý. Jedná se tedy o chybu na morfologické rovině. Musíme vytvořit makro typu `fix`, které opraví tag všem výskytům slovního tvaru „*oceli*“ podle předložky (např. „*o oceli*“ bude vždy šestý pád), výskyty bez předložky musí projít ručně anotátor.

Představme si, že nová verze makra a dat už vrací jen jedinou chybu: „*V ¿¿Dětem¡¡ autor vyjadřuje svůj vztah k matce*“ [vym]. Jedná se o výjimku, vytvoříme tedy makro typu `check` podle poslední verze makra typu `find`, které bude ignorovat tento konkrétní případ. Makro typu `check` budeme opakovaně pouštět v průběhu dalších oprav, abychom zaručili, že opravy žádných dalších jevů nezpůsobí neshodu v pádě mezi předložkou a příslušným jménem.

3.1.1 Technické kontroly

Ty kontroly, které nesouvisejí s lingvistickou teorií, ale s formátem dat, anotačním schématem či datovou reprezentací,[3] označujeme jako technické kontroly. Jde o základní kontroly, které podmiňují funkčnost jakýchkoliv dalších maker a dotazů, takže jejich opravy měly nejvyšší prioritu.

Mezi jinými patřila mezi technické kontroly tato makra:

Počet souborů: Tato kontrola zjišťovala, zda je počet datových souborů stále stejný. Sloužila k odhalování situací, kdy došlo omylem ke smazání, přesunutí či přejmenování některého souboru. „Zmizení“ souboru mohlo také způsobit nestandardní ukončení některého nástroje v okamžiku, kdy se chystal změněný soubor uložit.

Formát souborů: Tato kontrola pouze zjišťovala, zda program **btred** dokáže načíst všechny soubory. Příčinou chyb mohla být nedůsledná změna anotačního schématu či náhodné přepsání obsahu souboru.

Hodnoty atributů: Některé atributy uzlů směly nabývat hodnot pouze z omezené množiny: některé atributy byly číselné, jiné výčtové (jejich možné hodnoty byly dány výčtem). Seznam možných hodnot se však u některých výčtových atributů v průběhu anotování měnil (např. při rušení a přidávání

[3] Opravy nalezených technických chyb však často měly lingvistické důsledky, například proto, že byla nalezena špatná formulace anotačního pravidla v manuálu či se nově objevily „ztracené“ uzly, které bylo třeba doanotovat.

funktorů). Tektogramatické lema bylo částečně výčtovým atributem: mohlo nabývat libovolné hodnoty,[4] pokud však byla tato hodnota speciální (tj. začínala znakem # v novém anotačním schématu), musela se nacházet v seznamu zástupných lemat. Mezi zástupná lemata patří například `#Comma` pro čárku, `#PersPron` pro osobní a posesivní zájmena či `#Gen` pro doplněný všeobecný aktant.

Jednoznačnost identifikátorů: Identifikátory uzlů musely být jednoznačné. Tato kontrola nemohla běžet pouze jako makro programu **ntred**, protože bylo třeba porovnat výstupy všech běžících serverů, které měly načtená data. Kontrolní skript tedy patřil do skupiny `misc`.

Platnost odkazů: Všechny odkazy na uzly byly realizovány uvedením identifikátorů uzlů, na které se odkazovalo. Pro každý atribut, který odkazy obsahoval, existovalo kontrolní makro, které zjišťovalo, zda uzly, na které se odkazuje, skutečně existují. Makra tohoto typu také patřila do skupiny `misc`, protože úloha tohoto typu nemohla být obecně realizována jako jeden běh makra programu **ntred**: bylo třeba získat ze všech souborů seznam V identifikátorů všech uzlů a seznam R všech odkazů a ověřit, že neexistuje $i \in R - V$. Oba seznamy mohlo vracet jedno makro, jejich porovnání však již musel provádět další program, který získal výsledky od všech běžících serverů.

Lingvistický kořen: Pod technickým kořenem na tektogramatické rovině musel vždy viset právě jeden uzel – lingvistický kořen věty. „Prázdné“ věty (tj. technický kořen bez potomků) se mazaly; pokud měl naopak technický kořen několik synů, mohlo jít buď o chybu anotace, nebo bylo třeba větu rozdělit na několik vět, což byla poměrně složitá operace: pro nové věty se musely vytvořit nové technické kořeny s novými unikátními identifikátory.[5] Přesouvání uzlů z jedné věty do druhé se muselo provádět velmi opatrně, protože uzly byly zároveň objekty všech rovin najednou (viz oddíl 1.3), na každé rovině mohlo však být pořadí uzlů jiné – spojitý úsek uzlů analytické roviny tak nemusel na rovině tektogramatické odpovídat rovněž spojitému úseku. Po přesunutí uzlů bylo třeba ještě přepočítat pořadová čísla uzlů na všech rovinách.

[4] Teoreticky by mohl existovat slovník všech použitých tektogramatických lemat, takže by se jednalo o výčtový atribut; pak by se ale nedala data rozšířit o libovolnou novou větu obsahující dosud neznámé slovo. Kromě toho by byl slovník natolik rozsáhlý, že by jeho implementace drasticky zpomalovala práci programů **TrEd** i **btred** a neúnosně zvyšovala jejich paměťovou náročnost.

[5] Identifikátor se měnil i původní větě. Přidávaným uzlům se totiž identifikátory generovaly z identifikátoru kořene, takže kdyby se ten nezměnil, mohl by nový přidaný uzel dostat identifikátor, který už dříve dostal nějaký jiný uzel přesunutý do sousední věty.

Skrytá interpunkce: Uzly odpovídající interpunkci se na tektogramatické rovině neobjevovaly, pokud nešlo o spojovací uzly. Občas se stávalo, že při opravách takto zachycené spojovací konstrukce anotátor usoudil, že se o spojovací konstrukci nejedná, ale zapomněl interpunkční uzel skrýt (tj. smazat na tektogramatické rovině).

Kořen první v pořadí: Pořadové číslo technického kořene věty na všech rovinách muselo být 0. Pokud tomu tak nebylo, musel s větou někdo manipulovat nedovoleným způsobem, což často znamenalo i množství dalších chyb.

Skryté přidané uzly: Jako „přidané“ se označovaly uzly, které anotátoři generovali na tektogramatické rovině: většinou šlo o valenční doplnění vypuštěná na povrchu či o různé typy elips. Jako skryté mohly však být označeny pouze uzly analytické roviny, kterým na tektogramatické rovině neodpovídal žádný uzel (tj. pomocná slova). Při označení uzlu za skrytý byli společně s ním skryti i všichni jeho potomci – a pokud byl mezi nimi i nějaký přidaný uzel, toto makro ho objevilo.

Chyba automatické procedury: Jak již bylo naznačeno v oddílu 1.3, byla analytická data před začátkem anotování tektogramatické roviny automaticky převáděna na jakýsi prototyp tektogramatických dat automatickou procedurou (viz [Böhmová, 2001]). Tato procedura však bohužel občas skrývala velké úseky vět, které měly zůstat viditelné. Pomocí nejrůznějších heuristik se tato kontrola snažila takové uzly odhalit (podezřelá byla např. všechna skrytá plnovýznamová slova). Stejné chyby se mohl při nepozorném anotování dopustit i člověk, pokud nechal skrýt uzel, mezi jehož potomky byly uzly, které skryty být neměly.

Odkazy na nižší rovinu: Kontroly odkazů (viz oddíl 1.2) tvořily v podstatě samostatnou podskupinu technických kontrol, chyby v této oblasti hledalo 35 různých maker typu `find`. Složitost odkazů si lépe představíme, jestliže si rozebereme větu „*Velkou váhu mají ovšem i obavy z toho, že by se země helvetského kříže musela v budoucnu zříci své neutrality*“ [pdt]. Uzel s lematem `zříci_se` odkazuje nejen na slovo „*zříci*“, ale také na slova „*z*“, „*toho*“, „*že*“, „*by*“, „*se*“ a „*musela*“.

Prvním problémem bylo, že se s odkazy do nižší roviny původně vůbec nepočítalo. Při skrývání pomocných slov se do atributu `fw` („function word“) uzlu významového slova, k němuž skrývané slovo patřilo, doplňovala forma pomocného slova. Bohužel, anotátoři hodnotu tohoto atributu příliš nesledovali, takže se často stávalo, že se funkční slova objevila jinde, než měla, nebo se ocitla v atributu `fw` jiného pomocného slova, které bylo později

skryto, takže první skryté slovo zmizelo beze stopy. Když byl později vypracován systém odkazů z tektogramatické do analytické roviny, bylo třeba na všechny skryté uzly „dopočítat“ odkazy – tj. automaticky (pro ruční průchod byl jejich počet příliš velký) odhadnout, která pomocná slova patří ke kterým slovům významovým.

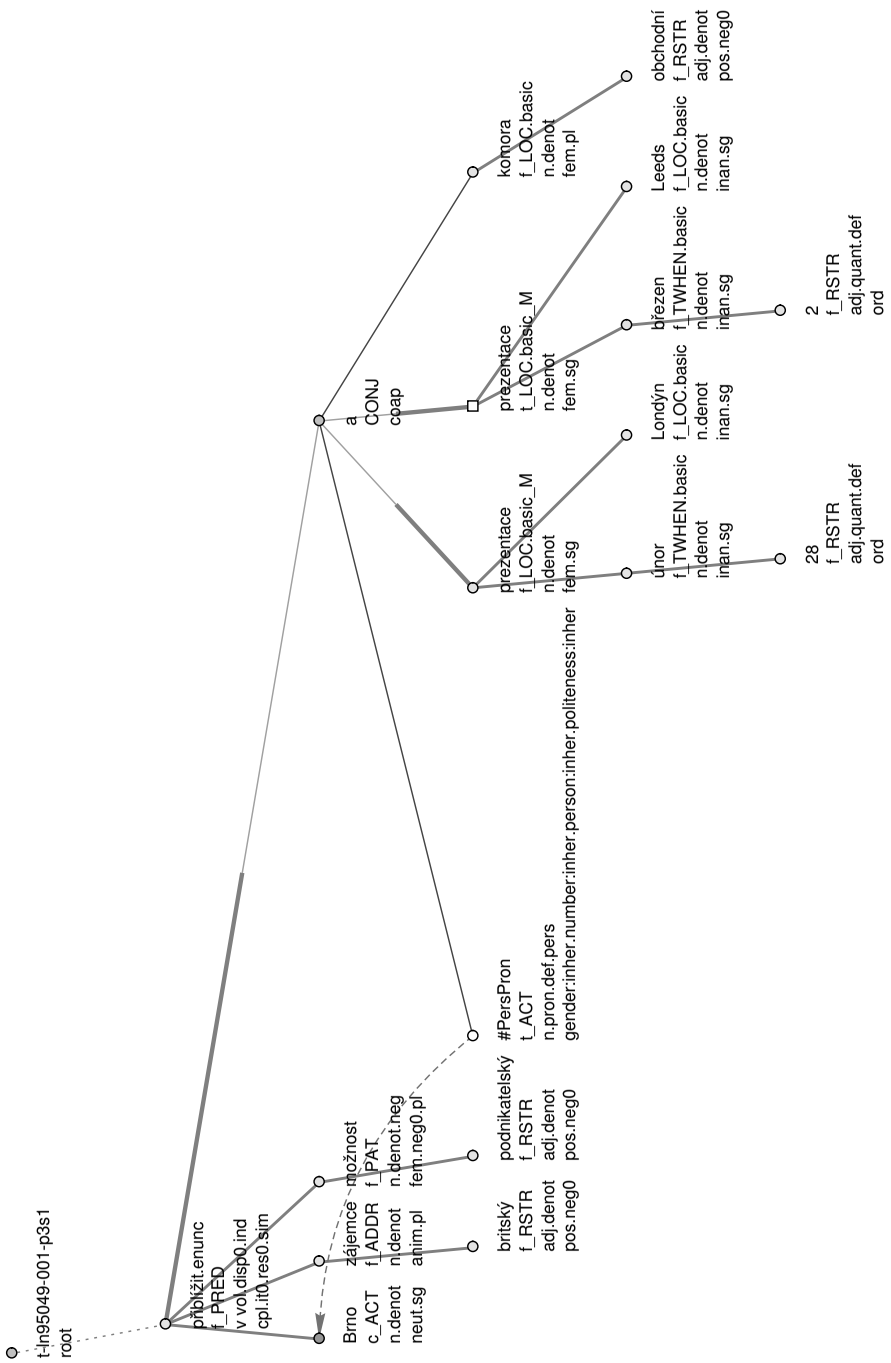
Pro tektogramatické uzly, které vznikly přímo z analytických (tj. nešlo o uzly kopírované ani generované) existovaly heuristiky, které se na základě analytické struktury snažily odhadnout, která pomocná slova k nim patří (atribut `fw` se nedal použít, protože jeho hodnoty byly spíše zavádějící). Situaci komplikovaly spojovací konstrukce, elipsy a složené předložky – chápání složených předložek na analytické a tektogramatické rovině bylo totiž nezávislé, takže se mohlo stát, že určitá skupina slov byla na analytické rovině anotována jako složená předložka (všechny uzly měly analytickou funkci `AuxP`), ale na tektogramatické rovině některému ze slov této skupiny odpovídal uzel; či naopak některé slovo, považované na analytické rovině za plnovýznamové, mohlo být na tektogramatické rovině považováno za součást složené předložky a skryto (resp. smazáno).

Pro kopírované uzly existovala další heuristika, která předpokládala, že originální uzly již mají korektně přiřazené odkazy. Ani v tomto případě nešlo o jednoduchý úkol, jak si můžeme ukázat na následujících dvou příkladech:

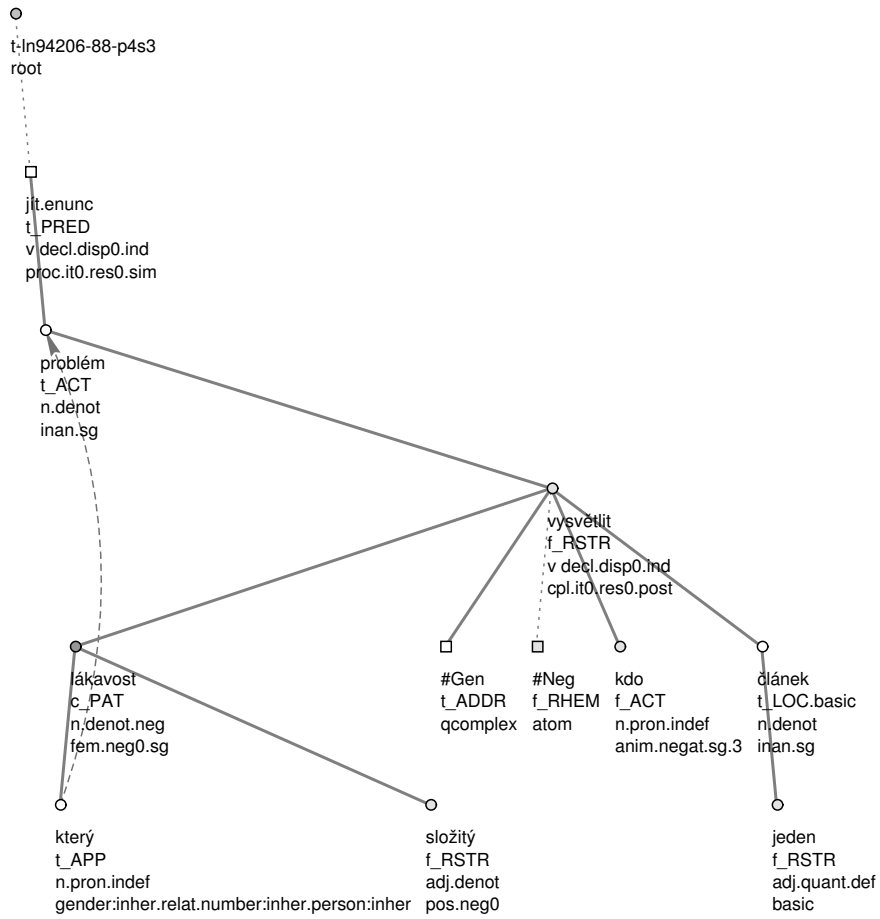
Na obrázku 3.2 je vidět, že v tektogramatickém stromě se na rozdíl od původní věty objevuje slovo „*prezentace*“ dvakrát: druhý uzel má příznak `is_generated` (uzel je zobrazen jako čtvereček). Kromě analytického uzlu pro slovo „*prezentace*“ (který je na analytické rovině jen jeden) by měly oba uzly odkazovat také na předložku „*na*“, která má vliv na jejich funktor `LOC`. Zdálo by se tedy, že u všech kopírovaných uzlů stačilo doplnit odkazy na stejné uzly, na které odkazoval originální uzel.

Situace je však složitější. Kopírovaný uzel „*jít*“ se v následujících větách chová jinak: „*Laikovi se brzy zdá, že nejde o pouhý laciný švindl, nýbrž o naprosto legitimní problém. O problém, jehož složitou lákavost nevysvětlí nikdo v jednom článku.*“ Uzel „*jít*“ v první větě bude odkazovat (kromě slova „*nejde*“) také na spojku „*že*“, která má vliv na jeho funktor `PAT`. V druhé větě bude sloveso „*jít*“ zkopírované (viz obrázek 3.3), ovšem na spojku „*že*“ by již v tomto případě odkazovat nemělo, protože jeho funktorem je `PRED` – odkaz by měl vést pouze k uzlu „*nejde*“ (kopírovaný uzel však nebude považován za negativní, protože pod ním nebude uzel s tektogramatickým lematem `#Neg`).

Jako kritérium se tedy spíše nabízí shodný funktor originálu a kopie, ale ani tato metoda nebyla docela spolehlivá: existovaly případy, kdy kopírovaný uzel měl jiný funktor, ale měl odkazovat na stejné pomocné slovo (například slovo „*Leonhard*“ se ve spojení „*debata a posezení s Leonhardem*“ [zm] vyskytuje na tektogramatické rovině dvakrát, jednou s funktorem `ACMP` jako syn uzlu „*posezení*“ a jednou s funktorem `ADDR` jako syn uzlu „*debata*“).



Obrázek 3.2: Ilustrace k odstavci Odkazy do analytické roviny: „*Brno chce přiblížit britským zájemcům podnikatelské možnosti na své prezentaci v obchodních komorách 28. února v Londýně a 2. března v Leedsu.*“ [pdt]



Obrázek 3.3: Ilustrace k odstavci Odkazy do analytické roviny: „*O problém, jehož složitou lákavost nevysvětlí nikdo v jednom článku.*“ [pdt]

Nakonec byly ze seznamu odkazů u kopírovaných uzlů smazány ty, které vedly k pomocným slovům, na něž odkazoval i originální uzel, pokud rodiče obou uzlů (kopie i originálu) neměli stejné tektogramatické lema – nejednalo se tedy o tentýž uzel ani o originál a kopii jednoho uzlu (příklad s „*Leonhardem*“ však tato heuristika řešila špatně).

Kromě toho se ručně procházely všechny uzly, od kterých vedly odkazy na více předložek nebo spojek, které ale na analytické rovině netvořily podstrom (tj. nešlo o víceslovné výrazy). V takových případech obvykle jeden z pomocných uzlů patřil pouze k originálu a druhý pouze ke kopii. Stačilo proto některý z odkazů smazat, ale nebylo tomu tak vždy (například „*ve spolupráci s Ruskem, Německem a se Spojenými státy*“ [vym] – druhá

předložka „*se*“ tvoří složenou předložku se slovy „*ve spolupráci*“, přestože na analytické rovině je umístěna jinde).

Dále se v mnoha případech na tektogramatické rovině zavěšovaly počítané předměty jako rodiče číselných výrazů – opačně, než tomu bylo na analytické rovině. Automatické heuristiky obvykle přiřazovaly předložky k číselným výrazům, musely tedy být dalším makrem přesouvány k počítaným předmětům. Podobný případ nastával při elipse jména, které však bylo rozvito adjektivem (například „*maso s červeným vínem a ryba s bílým*“ [vym]): na tektogramatické rovině bylo jméno doplněno, ale na předložku vedl odkaz od adjektiva (protože bylo jejím synem na analytické rovině). Všechny předložky byly proto od adjektiv rozvíjejících kopírovaná jména přesunuty k těmto jménům.

Posledním velkým problémem souvisejícím s odkazy na analytickou rovinu byla složená lemata. Každé složené tektogramatické lema (tj. lema obsahující znak _) mělo odkazovat na všechna slova, z kterých je tvořeno. Pokud se nějakou součást lematu nepodařilo najít mezi uzly, na které vedl odkaz, šlo o chybu. Existovala dokonce heuristika, která uměla chybějící slova dohledat. Jistou komplikací bylo, že se některá složená lemata měnila, aby byla srozumitelnější, takže bylo těžké hledat jejich součásti (v datech je `pamatovat_si` místo `pamatovat_se`, které by odpovídalo lematům jednotlivých slov, podobně je dnes `počínaje_konče` namísto původně užívaného `počínat_končit`).

Těsně před dokončením oprav anotace **PDT** zbývalo stále ještě přes 500 analytických uzlů,[6)] na které z tektogramatické roviny nevedl žádný odkaz. Odkazy na tyto uzly byly doplněny ručně.

Existenci uzlů, na které se odkazovalo, kontrolovalo makro ze skupiny popsané v bodě „Platnost odkazů“ na straně 43.

Chybná segmentace vět: Při anotování morfologické roviny prošel každý soubor rukama dvou anotátorů (soubory byly anotovány paralelně), při anotování analytické roviny rukama dalšího anotátora a při anotování tektogramatické roviny ho procházeli dokonce nejméně tři anotátoři (první anotoval strukturu s funktory, druhý koreferenci a třetí aktuální větné členění). Všichni mohli různými způsoby upozorňovat na chyby, které nedokázali sami opravit – nejčastější z nich bylo špatné rozdělení vět: někde byla tečka za zkratkou chybně považována za tečku za větou, jinde byl nadpis, který nekončil žádným interpunkčním znaménkem, připojen k první větě článku. Přesto zůstaly tyto chyby na mnoha místech neoznačené. Důvodů, proč tomu tak bylo, je několik: Upozorňování na chyby nebylo v prvních fázích anotování povinné, protože se předpokládalo, že ani automatická procedura, která bude jednou texty anotovat, nebude umět všechny takové

[6)] Do tohoto čísla není započítáno několik tisíc uzlů odpovídajících interpunkci, na kterou odkazy z tektogramatické roviny nevedou.

chyby sama opravit. Některé chyby se opakovaly v určitých souborech tak často, že je anotátoři přestali označovat jednotlivě – neoznačené výskyty se pak však daly jen těžko najít automaticky. Jiné chyby byly patrné teprve při čtení souvislejších úseků textu než jednotlivých vět, takže je mohli anotátoři snadno přehlédnout. (Posledním důvodem – doufejme, že nepříliš častým – pak mohla být nedůslednost anotátora.)

Kromě míst, která anotátoři přímo označili, se daly špatně rozdělené věty hledat pomocí nejrůznějších odhadů: podezřelé byly všechny věty, v nichž se vyskytovala předložka, za níž až do konce věty nenásledovalo slovo v pádě, který předložka vyžaduje; podezřelé byly věty, které končily čárkou či jiným znakem, který obvykle věty neukončuje; a za podezřelé byly považovány i věty začínající malým písmenem (o rozdělování vět srov. také bod „Lingvistický kořen“ na straně 43).

Spojená a rozdělená slova: Některá slova byla ve zdrojových textech napsána vedle sebe bez mezery, jiná byla naopak chybně rozdělena na několik slov, přestože mělo jít o slovo jediné. Ani tyto chyby neoznačili anotátoři důsledně všechny, pro jejich hledání se však nepodařilo napsat žádné makro. Dalším zdrojem chyb v dělení slov byl tzv. *tokenizer*, tedy program, který ve zdrojovém textu označuje jednotlivá slova: ten například často považoval datum (např. 30. 5.) za číslo s desetinnou tečkou (30.5). Proto byla všechna čísla s desetinnou tečkou, jejichž celá část byla menší než 32 a část za tečkou menší než 13, ručně zkontrolována. Rozdělování a spojování slov bylo poměrně obtížnou operací, jak už bylo poznamenáno v oddílu 1.3.

Špatně označené změny forem: Každé slovo v korpusu má zachovanou původní formu, v níž se v textu před začátkem anotování vyskytovalo. Za jistých okolností však byla slovům přidávána nová forma, a to buď opravená (v případě, že v původním slově byl překlep), nebo normalizovaná (normalizace se týkala čísel, byly vynechávány mezery v řádech tisíců, desetinná čárka se měnila na tečku a podobně).

Důvod, proč byla forma slova změněna, se ukládal jako hodnota atributů `formtype` a `origftype`. Přesná definice hodnot těchto atributů ale chyběla, navíc se některé hodnoty ztrácely během konverze mezi různými formáty dat.

Po vytvoření nového anotačního schématu byly tyto atributy sloučeny do nového atributu `form_change` a jeho hodnoty byly přegenerovány podle toho, jak se forma slova lišila mezi morfologickou rovinou a rovinou slovních tvarů.

3.1.2 Morfologické kontroly

Mezi morfologické kontroly, tedy opravy anotace na morfologické rovině, by se v první řadě daly zařadit opravy, prováděné při druhém opakování

experimentu popsaného v [Ondruška, Panevová a Štěpánek, 2003]. Podrobně byly procházeny objekty sloves na analytické rovině a jich se také týkala většina oprav: šlo hlavně o špatně určené pády podstatných jmen a zájmen, popřípadě zcela špatně určené slovní druhy. Kromě oprav dat přinesla tato kontrola i řadu oprav v morfologickém slovníku, který neznal některá slova či přiřazoval některým tvarům nesprávné kategorie.

Později byla napsána ještě další makra, která kontrolovala morfologickou rovinu:

Imperativy a vokativy: Mnohé slovní tvary, které se dají interpretovat jako imperativ nebo vokativ, tak byly často označkovány, přestože se o ně v daném kontextu nejednalo (tak byl „*otec*“ označen jako imperativ od „*otéci*“, „*čas*“ od „*časit*“ a častá značka autora „*haš*“ byla značkována jako imperativ od „*hasit*“; v případě vokativů se jednalo hlavně o tvary homonymní s nominativem či cizí slova). Všechny vokativy a imperativy, které připouštěly i jinou interpretaci (na základě morfologické analýzy nebo úsudku anotátorů), byly ručně zkontrolovány.

Lokál bez předložky: V českém jazyce se lokál nemůže objevit bez předložky. Tato kontrola hledala všechny výskyty lokálu, kterým ve větě nepředchází předložka pojící se s šestým pádem. Na analytické rovině se dalo zároveň testovat, zda je taková předložka rodičem daného lokálu, a na tektogramatické rovině musel od uzlu odpovídajícího jménu v šestém pádě vést na předložku odkaz do analytické roviny.

Negace: Vyznačování negace na morfologické rovině nebylo zcela konzistentní (například slovo „*neustálý*“ bylo lematizováno jako „*ustálý*“). U některých slov se dokonce informace o negaci docela ztrácela, což způsobovalo problémy na tektogramatické rovině, kde je negace v jistých případech reprezentována generovaným uzlem s tektogramatickým lematem `#Neg`. Na základě výstupu z tohoto makra byl sestaven seznam problematických slov začínajících na „ne-“, která pak byla opravována (např. lema „*japný*“ na „*nejapný*“, „*smyslný*“ na „*nesmyslný*“ a podobně).

Shoda adjektivního přívlastku s řídícím jménem: Všechny adjektivní přívlastky (analytická funkce `Atr`), které se lišily pádem, číslem nebo rodem od svého rodiče, byly ručně procházeny a většinou byla opravována jejich morfologická informace. V některých případech však bylo příčinou chyby špatné zavěšení uzlu nebo nesprávně přiřazená analytická funkce.

Podobným způsobem se kontrolovala také shoda křestního jména s příjmením, alespoň pro ta jména, která měla vyplněný příslušný morfologický příznak (znak `Y` a `S` v morfologickém lematu, viz [Zeman a kol., 2005]).

Selhání pravidel: Všechna data byla také otestována programem pro pravidlovou disambiguaci (viz [Květoň, 2006]). Všechna místa, která pravidla označila za chybná, pak byla ručně procházena a opravována. Pomocí pravidel byly objeveny chyby nejrůznějších druhů, převažovaly špatně určené tagy (většinou šlo o drobné změny v rodě či životnosti, které snadno uniknou pozornosti anotátora) a chybějící čárky v souvětích.

3.1.3 Analytické kontroly

Mezi analytické kontroly opět patří opravy objektů sloves v rámci prohloubení pokusu popsaného v [Ondruška, Panevová a Štěpánek, 2003]. Několik set uzlů bylo převěšeno (nejčastější chybou bylo, že anotátor nezavěsil objekt pod správné sloveso, ale pod jiné, které je spolu s ním součástí složeného slovesného tvaru, například v konstrukci „*chtěl říci, že se mu líbí*“ [vym] byla vedlejší věta zavěšena na „*chtěl*“ místo na „*říci*“). Často bylo také za objekt označeno příslovečné určení.

Ve fázi intenzivních kontrol **PDT** byla vytvořena ještě další makra, která kontrolovala analytickou rovinu, například následující:

Spojovací konstrukce: Každý spojovací uzel odpovídá nějaké spojovací konstrukci, a tedy pro každý spojovací uzel musejí existovat nějaké členy této konstrukce (výjimkou z tohoto pravidla byla věta „*Ale...*“, která obsahovala pouze spojovací uzel a koncovou interpunkci). Podobně musí nad každým uzlem, označeným za člen spojovací konstrukce, existovat nějaký spojovací uzel. Pokud tomu tak není, selhávají funkce pro hledání efektivních synů a rodičů (popsané v pododdílu 2.2.1), takže jakékoliv složitější dotazy, které se ptají po struktuře, vracejí špatné odpovědi.

Na analytické rovině nesměly být členy jedné spojovací konstrukce uzly s různými analytickými funkcemi (s výjimkou funkce `ExD`, která jen označovala, že uzel s příslušnou analytickou funkcí ve stromě chybí). Pokud se tak stalo, jednalo se většinou o špatné zachycení společného rozvití, které bylo vinou nepozornosti označeno jako člen spojovací konstrukce (obzvlášť častá byla tato chyba v případech, kdy samo společné rozvití bylo spojovací konstrukcí – její spojovací uzel měl mít analytickou funkci `Coord`, ale dostal vizuálně nepříliš odlišnou funkci `Coord_Co`, která byla navíc přiřazována velmi podobnou klávesovou zkratkou).

Možné chyby byly hledány také mezi společnými rozvitími, která stála mezi členy spojovací konstrukce – typické společné rozvití stojí před všemi členy nebo za nimi.

Řídící uzel pro atribut: Uzly s analytickou funkcí `Atr` směly mít rodiče pouze některých slovních druhů. Klasická formulace (srov. například [Havránek a Jedlička, 1981]), že „přívlastek je větný člen, který závisí na

podstatném jméně“, je ovšem nepřesná – platí buď tehdy, chápeme-li podstatné jméno jako hloubkový slovní druh, nebo pokud ji považujeme za odpověď na otázku, co může záviset na podstatném jméně. Hloubkové slovní druhy však byly generovány na tektogramatické rovině až na konci anotačního procesu, takže se ke kontrole analytické roviny nedaly použít; a cílem kontroly nebylo hledat, kdy má substantivum mezi syny něco jiného než atribut,[7] ale kdy je rodičem atributu něco, co jím být nemůže.

Šlo tedy v podstatě o vyjmenování slov, která se mohou stát hloubkovými substantivy: na analytické rovině mohou být rodičem přívlastku vedle podstatného jména také některé číslovky a zájmena (pokud jsou substantivní povahy), cizí slova a v podstatě jakékoliv slovo, jestliže sám přívlastek začíná velkým písmenem (například název podniku „*Pražská teplárenská*“ se rozebíral tak, že na řídícím adjektivu „*teplárenská*“ visel atribut „*Pražská*“). Číslovky, které se syntakticky chovají jako jména, se daly rozeznat podle druhé pozice tagu (tzv. *subPOS*, slovní poddruh), u zájmen bylo však rozlišení ještě komplikovanější a některá slova musela být vyjmenována jednotlivě (například neurčitým zájmenům začíná tag znaky `PZ`, zájmeno substantivní povahy „*někdo*“ má tag `PZM-1----------` a zájmeno adjektivní povahy „*nějaký*“ může dostat tag `PZYS1----------`, který se od prvního liší jen znakem na pozici čísla a rodu[8]).

Tečky za zkratkami: Tečky za zkratkami a řadovými číslovkami byly na analytické rovině anotovány odlišně od teček na konci věty – dostávaly speciální analytickou funkci `AuxG` (grafický symbol) a zavěšovaly se na uzel odpovídající zkratce. Pokud tečka plnila obě funkce najednou (zkratka stále ve větě jako poslední), měla být anotovaná jako tečka za zkratkou. Právě tečky za zkratkami na koncích vět byly často zachyceny špatně, v horším případě byla dokonce věta rozdělena na místě tečky za zkratkou a bylo třeba ji spojit s následující větou.

Předložky a spojky: Předložky a spojky se dají rozeznat jak podle tagu (tag spojky začíná písmenem `J` a tag předložky písmenem `R`), tak podle analytické funkce (`AuxP` pro předložky a `AuxC` pro podřadicí spojky). Pokud byl nalezen uzel s tagem předložky nebo spojky, ale s jinou analytickou funkcí, nebo pokud byl nalezen uzel s funkcí předložky či spojky, ale jiným tagem, jednalo se většinou o chybu. Výjimky tvořily hlavně složené předložky, jinak se muselo zacházet také se spojkami souřadicími.

[7] Mezi syny substantiva byl kromě atributu v pořádku i doplněk s analytickou funkcí `Atv`.
[8] Znak `S` označuje singulár, znak `Y` znamená, že rod může být jak mužský životný, tak neživotný.

3.1.4 Tektogramatické kontroly

Tektogramatické kontroly tvořily největší část celé sady testovacích maker. Bylo tomu tak jednak proto, že tektogramatická rovina sama o sobě je mnohem složitější než ostatní roviny, jednak proto, že práce na ostatních rovinách skončila již před několika lety, kdežto na tektogramatické rovině se dosud pracovalo, takže se jí věnovalo podstatně více lidí než rovinám ostatním.

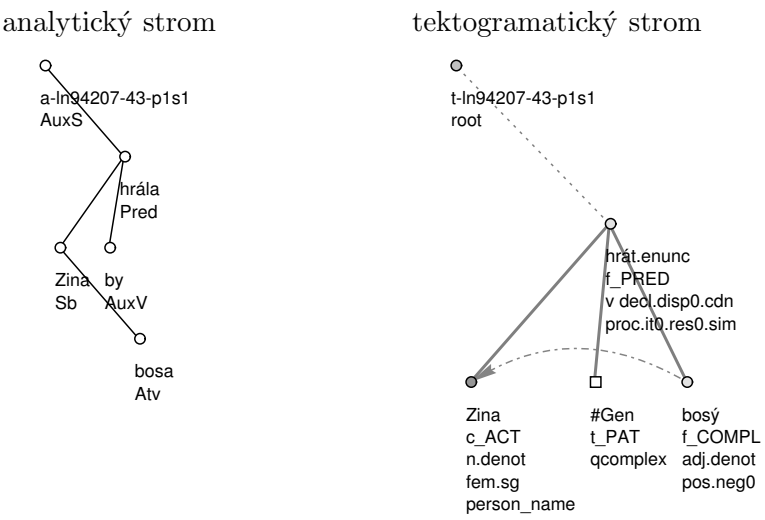
Podobně jako v předchozích pododdílech následuje seznam některých maker, která prováděla tektogramatické kontroly:

Doplněk: Doplněk závisí na dvou slovech najednou: na slovese a jméně. Na analytické rovině se doplňky zavěšovaly na jméno, předpokládalo se totiž, že příslušné sloveso se dá dohledat mezi předchůdci jména. Automatická procedura převádějící analytické stromy na tektogramatické (viz [Böhmová, 2001]) však doplňky nepřevěšovala, takže bylo úkolem anotátorů najít příslušné sloveso, doplněk na ně převěsit a ke jménu nakreslit od doplňku šipku (tj. do atributu `compl.rf` uložit identifikátor odkazovaného uzlu) – [Mikulová a kol., 2005, str. 356]. Mělo tedy platit, že pokud měl doplněk analytickou funkci `Atv`, a tedy visel na analytické rovině na jméně, vedla od něj na tektogramatické rovině šipka k uzlu odpovídajícímu tomuto jménu (viz příklad na obrázku 3.4). Pokud měl naopak analytickou funkci `AtvV`, jméno nebylo na povrchu vyjádřeno a doplněk tedy visel na slovese. Na tektogramatické rovině pak visel na uzlu odpovídajícím tomuto slovesu a šipka od něj vedla k nějakému doplněnému uzlu (viz obrázek 3.5). I z tohoto pravidla ovšem existovaly výjimky (např. některá slova s funktorem `COMPL` nebyla na analytické rovině vůbec považována za doplňky).

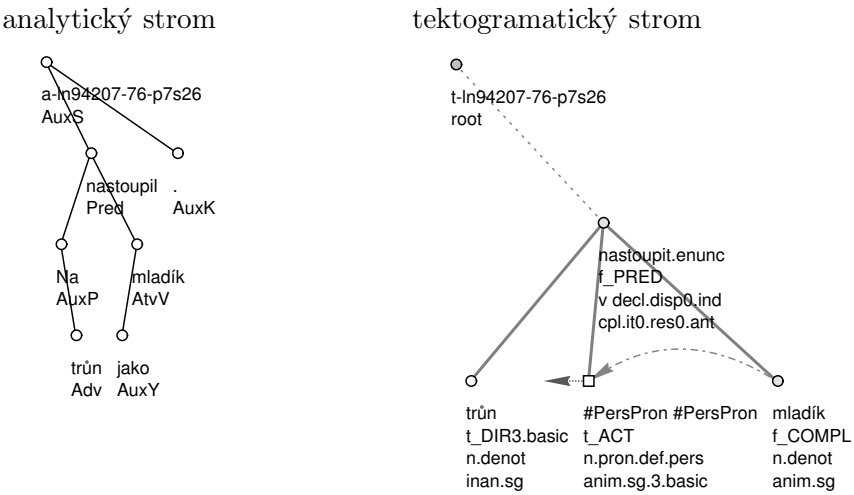
Na tektogramatické rovině mělo být efektivním rodičem doplňku vždy sloveso. Jak již však bylo poznamenáno v poznámce 1 na straně 2, nemuselo být toto sloveso vždy slovesem ve smyslu morfologickém, dokonce ani ve smyslu hloubkového slovního druhu. Pro kontrolu zavěšení doplňku se proto používalo makro ze skupiny `check`, které obsahovalo seznam výjimek, kdy doplněk visí na jiném slovním druhu. Pro každý takový doplněk bylo v makru zapsáno, který uzel byl jeho rodičem a ke kterému uzlu od něj vedla „šipka“ reprezentující druhou závislost. Pokud by se tedy anotace takového doplňku změnila, kontrolou by neprošel, anotátoři by ho museli znovu prohlédnout a zapsat novou situaci do seznamu výjimek.

Další kontrolní makro ověřovalo, že pod každým slovesným doplňkem (např. přechodník, vedlejší věta uvozená spojkou „*jak*“, trpné příčestí) je přidaný aktant s tektogramatickým lematem `#Cor`. Od tohoto aktantu navíc musí vést koreference ke stejnému jménu, na něž odkazuje doplněk svou druhou závislostí – viz příklad na obrázku 3.6.

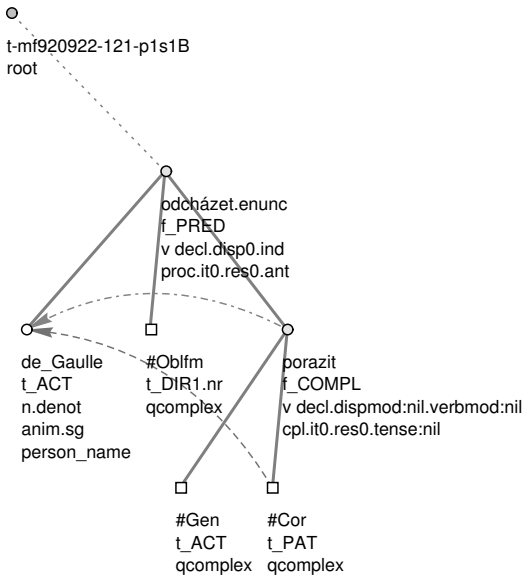
Existenci uzlů, na které se odkazovalo, kontrolovalo makro ze skupiny popsané v bodě „Platnost odkazů“ na straně 43.



Obrázek 3.4: Doplněk s analytickou funkcí `Atv`: „*Zina by hrála bosa.*“ [zm]



Obrázek 3.5: Doplněk s analytickou funkcí `AtvV`: „*Na trůn nastoupil jako mladík.*“ [pdt]



Obrázek 3.6: Doplněk vyjádřený slovesem: „*De Gaulle odcházel poražen.*“ [pdt]

Seznamy forem: Makro vypisovalo pro každý uzel jeho funktor a formu jemu odpovídajícího slova včetně předložek a spojek, na které tento uzel odkazoval. To znamená, že ze seznamu odkazů do analytické roviny muselo makro odstranit pomocná a modální slovesa a všechny uzly, které se podílely na tektogramatickém lematu daného uzlu (např. zvratné „*se*“ v případě reflexiva tantum). Výstup makra zpracovával další skript, který vstup třídil podle funktorů, takže pro každý funktor existoval seznam všech forem, kterými je v datech realizován. Tyto seznamy četli anotátoři a hledali „podezřelé“ formy (například bezpředložkové jméno v druhém pádě jako forma funktoru `DIR1`, příslovečného určení místa jako odpovědi na otázku „odkud?“).

Příklad seznamu, který anotátoři procházeli, je v tabulce 3.2. Seznam byl anotátorům předkládán ve formátu **HTML** a každé slovo uvedené ve sloupci příkladů bylo hypertextovým odkazem na větu, v níž se vyskytovalo, takže si anotátor mohl snadno zobrazit kontext, v němž se příslušná forma v datech objevila.

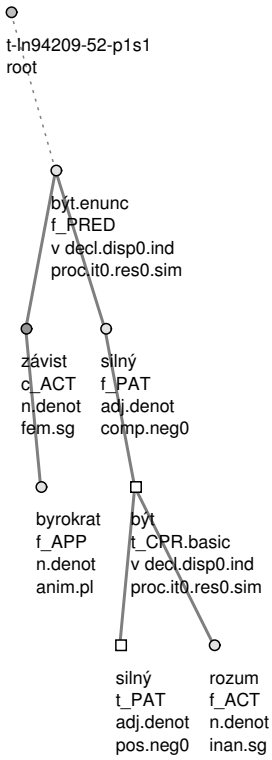
Tabulky forem pro všechny funktory se nepodařilo projít a opravit, taková práce totiž téměř odpovídá procházení celého korpusu. Zkontrolovány ale byly všechny funktory, ke kterým se generovaly subfunktory, aby nedocházelo k přiřazení špatného subfunktoru. Na základě zkušeností, které měli anotátoři s tabulkami, byly vytipovány nejčastější zdroje chyb a

Forma	příklady	počet
1	*doprava* poplatek *spor hranice podpora* Kaiserslautern *událost událost folklor spotřeba*	10
2	světa osobitého *sbor* tisíc *trať úvěr strana* rozbušek surovin *strana*	30
3	standardu	1
4	*podívaná*	1
6	*kolo*	1
7	Spojením *soupeř* dopravou	3
X	Tchaj *ten ten ten ten* ČR *ten*	7
adj	mála mála mála mála mnoha mála	6
adv	odtud odkud Zezadu odtud odkud odkud odkud odtud odtud odkud	80
na#4	Slovensko	1
od#2	Lublaně domů nás něj města vědy puku Brna aukce trendu	208
při#6	vybavení	1
s#2	stolu stolu stolu stolu	4
nezjištěná	*tam tady tam tady tam tam tady tam tam tam*	962
v#6	Maggioře	1
vfin	jsou vybíráme	2
z#2	desky úst Terezína příslušníků činnosti lístku Berlína života nich nich	4048
z_strana#2	hráček zemí diváků	3
z_ten#vfin	připisuje napsal	2
z_ten_že#vfin	pracovala	1
z_řada#2	demokratů	1
zpod#2	minisukní	1
zpoza#2	branky skříně vápna	3
zprostřed#2	dvorku	1

Tabulka 3.2: Příklad tabulky forem pro funktor `DIR1`: čísla znamenají pády, X označuje slova bez pádu, adj adjektiva, adv adverbia a vfin slovesa v určitém tvaru. Slova sázená kurzívou odpovídají přidaným uzlům, jejichž forma není příliš zajímavá, protože mohla být kopírována ze zcela odlišného kontextu.

opraveny pomocí dalších maker. Nalezené chyby byly opět všech možných druhů: špatně přiřazené funktory, chyby v morfologické anotaci, chyby v odkazech na analytickou rovinu či chyby v tektogramatické struktuře.

Srovnání: Funktor srovnání `CPR` (viz [Mikulová a kol., 2005, str. 669]) má širokou škálu možných realizací: od jednoduchých konstrukcí jako „*naproti tomu*“ či „*v porovnání s minulým rokem*“ až po komplikované srovnání zachycené na tektogramatické rovině v podobě vedlejší věty jako v příkladu na obrázku 3.7. Právě zachycování konstrukcí se spojkami „*jako*“ a „*než*“ bylo poměrně složité: na příkladu je vidět, že se věta „*Závist byrokratů je silnější než rozum*“ rozebírá tak, jako by bez elipsy zněla „*Závist by-*



Obrázek 3.7: Srovnání: „*Závist byrokratů je silnější než rozum*“ [pdt]

rokratů je silnější, než je silný rozum“. Některé uzly se kopírovaly, jindy se vytvářely uzly se zástupnými lematy (třeba v případech, kdy by kopírování uzlu nedávalo smysl, například ve větě „*Je stejný jako já*“ [vym] by bylo nesmyslné zopakovat slovo „*stejný*“).

Složitost anotačních pravidel pro srovnání byla příčinou častých chyb anotátorů (některá pravidla navíc vznikala až dodatečně v době, kdy již byla větší část korpusu oanotovaná). Existovalo několik maker, která se snažila v zachycení srovnání nalézt chyby: Jedno z nich ověřovalo, že sloveso s funktorem `CPR` odkazuje do analytické roviny na spojku „*jako*“ nebo „*než*“ (vzhledem k častým elipsám totiž heuristiky, popsané v bodě „Odkazy na nižší rovinu“ na straně 44 a následujících, nedokázaly tyto spojky přiřadit ke správným uzlům). Další makro kontrolovalo, zda je kopírované sloveso v doplněné vedlejší větě shodné se slovesem věty hlavní a zda obě odkazují na tentýž uzel analytické roviny. Kopírovanému slovesu se také musely doplnit valenční členy, to ale dokázalo ověřit i makro pro kontrolu valence (viz bod „Valence“ na straně 60).

Zrušené a nové funktory: Jak již bylo zmíněno v bodě „Hodnoty atributů“ na straně 42, v průběhu anotování tektogramatické roviny byly přidávány nové funktory a rušeny některé staré – například funktor `CTERF`, který měl označovat kontrafaktuál, byl nahrazen obecnějším funktorem `COND` označujícím podmínku, protože rozdíl mezi podmínkou reálnou a nereálnou lze rozlišit pomocí hodnoty gramatému `verbmod` (podmínka reálná má hodnotu `ind`, nereálná `cdn` – nereálná podmínka má typicky formu kondicionálu). Funktor `NORM`, který měl označovat normu, byl sloučen s funktorem `CRIT`, protože hranice mezi těmito dvěma funktory byla příliš nejasná a v mnohých případech sporná. Podobně byl zrušen funktor `DES` pro deskriptivní přívlastek a sloučen s funktorem `RSTR` pro přívlastek restriktivní, protože tyto dva druhy přívlastku nemají rozdílné formální vyjádření a jejich rozlišení je zhusta otázkou interpretace.

Nově byly zavedeny například funktory `OPER`, `CPHR`, `CM` a `AUTH`. Při opravách dat se musely zrušené funktory nahradit jinými (což bylo možné provést automaticky, pokud se funktor nahrazoval vždy tímtéž novým funktorem nebo pokud pro výběr nového funktoru existovala jednoznačná kritéria), kdežto v případě nových funktorů se musely ručně procházet situace, kde jejich použití přicházelo v úvahu.

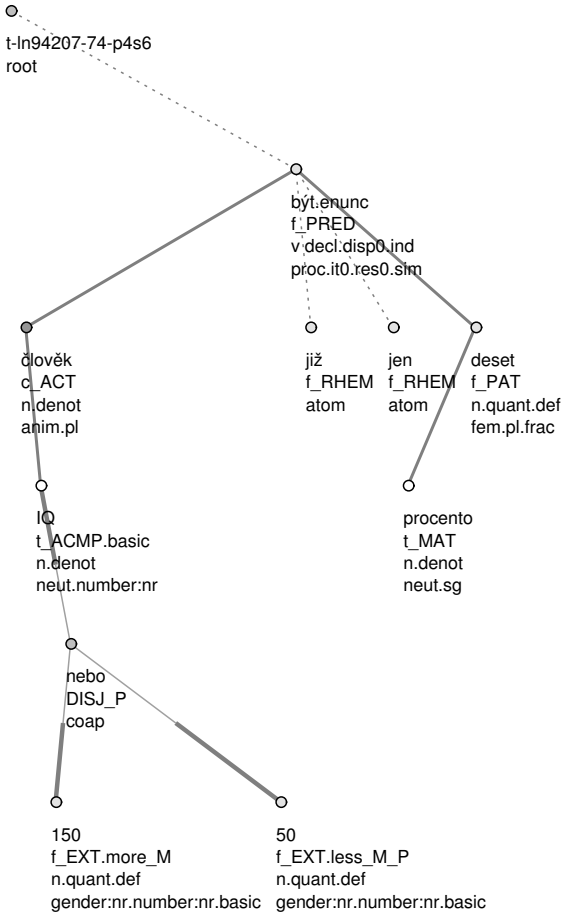
Do jisté míry obdobným případem bylo i rozhodnutí zrušit zachycování spojky „*zatímco*“ jako souřadicí spojky s funktorem `CONFR`. Všechny výskyty musel projít anotátor ručně, aby rozhodl, zda se opravené struktuře přiřadí funktor `CONTRD` („*Ceny energie v Rakousku pro malospotřebitele se v lednu zvýšily o 0,6 procenta v porovnání s lednem loňského roku, zatímco v prosinci* ***stouply*** *v meziročním srovnání o dvě procenta.*“ [pdt]) nebo `TPAR` („*Prostě jen vyčká následujícího rána, a zatímco Sára ještě* ***spí****, zapřáhne osla.*“ [pdt]).

Vícenásobné funktory: Anotátoři měli v průběhu anotování možnost přiřadit uzlu seznam funktorů (obvykle dvojici), pokud si nebyli jisti, který z nich by měl uzel jednoznačně dostat. Některé specifické kombinace funktorů vyjadřovaly, že jde o stavovou konstrukci (atribut `is_state` v novém anotačním schématu), zbývající kombinace musely být opraveny ručně.

Spojovací konstrukce: Kontroly spojovacích konstrukcí na tektogramatické rovině byly podobné kontrolám těchto konstrukcí na rovině analytické (viz strana 51). Hlavním rozdílem bylo, že na tektogramatické rovině mohly být členy spojovací konstrukce i uzly s různými funktory, takže bylo obtížnější odhadnout, kdy se jedná o chybně zachycenou konstrukci.

Pro spojovací uzly byl vytvořen seznam přípustných lemat. Pokud se na spojovací konstrukci vedle slova ze seznamu podílelo ještě nějaké další slovo, bylo označeno funktorem `CM` („coordination modifier“) a zavěšeno na spojovací uzel. Mezi nejčastější taková slova patřilo „*i*“, „*sice*“, „*tedy*“ nebo „*nejen*“.

Parenteze: Na analytické rovině se parenteze vyznačovala příznakem u řídícího členu podstromu, který byl do věty vložen. Na tektogramatické rovině se jako součást parenteze označoval každý uzel zvlášť (viz [Mikulová a kol., 2005, str. 277]), aby se daly přesněji zachytit konstrukce, kdy vložené uzly netvoří podstrom – viz obrázek 3.8: slova v závorce jsou označena jako součást parenteze (písmeno P u uzlů „*nebo*“ a „*50*“), ale slova „*nad 150*“ nikoliv.



Obrázek 3.8: Parenteze: „*Lidí s IQ nad 150 (nebo pod 50) jsou již jen desetiny procenta.*“ [pdt]

Jedno z maker pro kontrolu parenteze ověřovalo, že uzly označené jako parentetické odpovídají na povrchu souvislému úseku textu ohraničenému nějakými grafickými symboly. Zdaleka ne každé slovo, které bylo napsané v závorce, však bylo anotováno jako parenteze: kupříkladu vysvětlení nebo zavedení zkratky se anotovalo jako apozice („*Po volbách vstoupil do*

Občanské demokratické aliance (ODA)“ [vym]). Zdálo by se, že tyto konstrukce se dají strojově rozeznat podle toho, že jednotlivá písmena zkratky odpovídají počátečním písmenům názvu, ani takové pravidlo však není zcela spolehlivé (srov. „*Organizace zemí vyvážejících ropu (OPEC)*“ [vym]). Nejasné případy procházeli ručně anotátoři.

Šablony: Jako šablony byly označovány vzory stromů pro skupiny slov bez zjevných závislostních vztahů – bibliografické údaje, časové údaje, čísla zákonů, adresy a podobně (viz [Mikulová a kol., 2005, str. 840]). V širším pojetí se mezi šablony řadily i číslovky, protože mezi jednotlivými slovy víceslovných číslovek bychom těžko hledali závislost. Makra hledala podezřelé povrchové řetězce a kontrolovala, zda tektogramatická struktura odpovídá šabloně.

Cizí fráze a frazémy: Pro zachycování cizích frází platila jednoduchá pravidla, takže se jejich kontroly podobaly kontrolám šablon. Cizí fráze se daly rozeznat podle morfologických příznaků nebo funktoru `FPHR` (viz [Mikulová a kol., 2005, str. 798]).

České frazémy byly označovány funktory `DPHR` a `CPHR`, jejichž kontrola spadala pod kontrolu valence – viz následující bod.

Valence: V datech jsou valenční rámce vyznačeny jako odkazy na identifikátory rámců. Byl vypracován *valenční slovník*, realizovaný jako rozsáhlý **XML** dokument, který obsahuje všechny rámce vyskytnuvší se v **PDT** (viz [Hajič a kol., 2003] a [Mikulová a kol., 2005, str. 89]). Ten byl členěn na elementy odpovídající jednotlivým lematům a každý z nich se dále dělil na jednotlivé rámce. Každé lema mělo přiřazený jednoznačný identifikátor, na jehož základě byly generovány identifikátory rámců (k identifikátoru lematu se přidávala zkratka anotátora a pořadové číslo rámce pro dané lema).

Rámce mohli přiřazovat uzlům všichni anotátoři, takže se slovník v průběhu anotování často a složitě měnil. Přibývaly do něj nové rámce pro známá lemata i nová lemata s novými rámci. Dále se rámce nejrůznějším způsobem měnily, do rámce mohl například přibýt nový fakultativní člen. Anotování navíc probíhalo paralelně, tj. každý anotátor měl svou kopii valenčního slovníku, kterou podle anotovaných dat upravoval; anotátoři své kopie slovníku pravidelně odevzdávali a ty se slévaly vždy do další verze slovníku, která byla anotátory používána v následujícím období.

Slévání jednotlivých valenčních slovníků od anotátorů bylo obtížnou operací. Jedním z problémů slévání bylo zaručení jednoznačnosti identifikátorů lemat a rámců. Pokud více anotátorů přidalo do slovníku ve stejné době totéž lema, bylo ve výsledném slovníku zapsáno jen jednou pod jedním z identifikátorů, ostatní identifikátory však musely být pro pozdější použití zakázány, jinak by mohlo dojít k následující situaci: Dva anotátoři by přidali

do slovníku stejné lema, jeden s identifikátorem l_1 a rámcem $l_1a_1f_1$, druhý s identifikátorem l_2 a rámcem $l_2a_2f_2$. Po slití by zbylo lema ve slovníku pouze s identifikátorem l_1 a s rámci $l_1a_1f_1$ a $l_2a_2f_2$. Nezakázaný identifikátor l_2 by tedy mohlo dostat nějaké další lema, což by sice ještě stále nevadilo, ale některý z rámců takového slova by mohl dostat identifikátor $l_2a_2f_2$, který by porušoval požadavek jednoznačnosti.[9)]

```
<word POS="V" lemma="blahopřát" id="v-w155">
  <valency_frames>
    <frame hereditary_used="5" status="reviewed" used="5" id="v-w155f1">
      <example>blahopřát vítězům  k vítězství</example>
      <note>gratulovat</note>
      <frame_elements>
        <element functor="ACT" type="oblig">
          <form>
            <node afun="unspecified" agreement="0" case="1" inherits="1"
             neg="unspecified"/>
          </form>
        </element>
        <element functor="PAT" type="oblig">
          <form>
            <node afun="unspecified" agreement="0" inherits="0"
             lemma="k-1" neg="unspecified">
              <node afun="unspecified" agreement="0" case="3"
               inherits="1" neg="unspecified"/>
            </node>
          </form>
        </element>
        <element functor="ADDR" type="oblig">
          <form>
            <node afun="unspecified" agreement="0" case="3" inherits="1"
             neg="unspecified"/>
          </form>
        </element>
      </frame_elements>
      <local_history/>
    </frame>
  </valency_frames>
</word>
```

Obrázek 3.9: Ukázka valenčního slovníku

Často se také stávalo, že dva anotátoři přidali do svých kopií slovníku ke stejnému lematu tentýž rámec pod různými identifikátory. Při slévání byly všechny takové skupiny rámců hledány a pro každou z nich byl vybrán jeden reprezentující rámec. Všechny ostatní rámce byly označeny

[9)] Podobně byly řešeny identifikátory přidávaných uzlů při rozdělování vět, srov. poznámku 5 na straně 43.

jako *nahrazené* („substituted“) tímto reprezentujícím rámcem. Reprezentující rámec mohl být časem sám nahrazen dalším, takže vlastně vznikaly posloupnosti nahrazených rámců. Při kontrolách valenčního slovníku bylo proto nutné také sledovat, zda každá posloupnost nahrazených rámců končí nějakým platným rámcem.

Chybně navržené rámce se označovaly jako *zrušené* („obsolete“), při kontrole dat a valenčního slovníku se pak všechny jejich výskyty v datech musely nahradit jinými rámci.

Valenční rámec byl ve slovníku zapsán jako seznam aktantů a obligatorních volných členů. Každý prvek seznamu obsahoval popis analytické struktury, která mu odpovídala (viz obrázek 3.9, na rámci „*blahopřát někomu k něčemu*“ je vidět zachycení složitější analytické struktury odpovídající patientu slovesa; z identifikátorů rámců byl odstraněn kód anotátora). Při kontrole forem členů valenčních rámců však bylo ještě třeba pamatovat na to, že formy jednotlivých členů se za určitých podmínek mění – typickým příkladem je pasivizace, kdy dochází ke změně formy aktoru a patientu. Kontrolní makro obsahovalo desítky transformačních pravidel (viz [Pajas, připravováno]), která mohla měnit formy jednotlivých členů rámce (dodejme jen pro představu, že popis všech pravidel má přes 550 řádek). Příkladem takového pravidla jsou následující řádky, jde o pravidlo platné pro slovesa v pasivním tvaru s pomocným slovesem „*být*“:

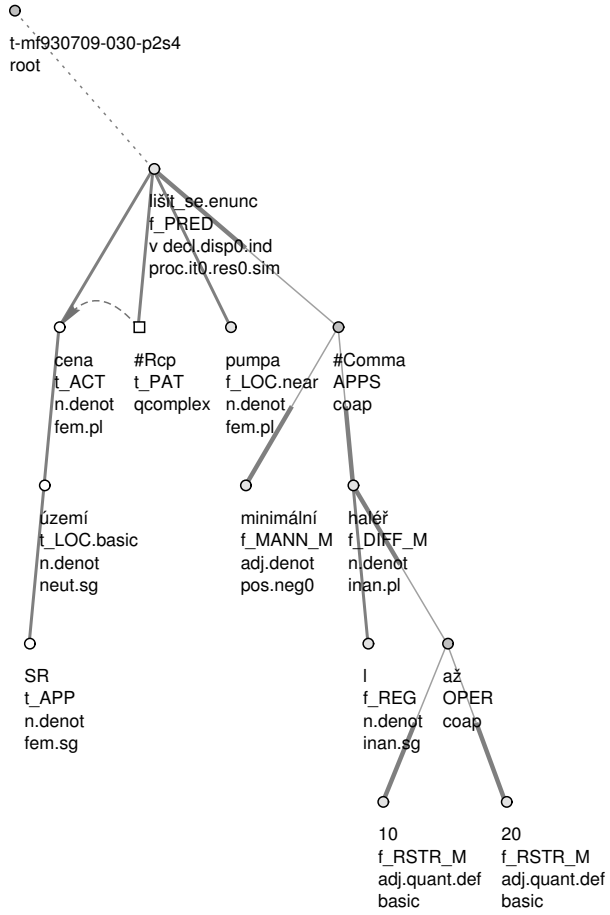
```
ACT(.1), PAT(.4), EFF/.4\[{jako,jakožto}(\/AuxY)?[.:]?\])$/  =>
-ACT(.1), +ACT(.7;od-1[.2]), -PAT(.4), +PAT(.1),
EFF/^.4(\[{jako,jakožto}(\/AuxY)?[.:]?\])$/.1${1}/
```

Pravidlo říká, že první pád aktora se nahradí sedmým pádem nebo předložkou „*od*“ s druhým pádem, čtvrtý pád patientu se nahradí prvním a čtvrtý pád efektu se nahradí rovněž prvním pádem (se zachovanými spojovacími výrazy „*jako*“ nebo „*jakožto*“). Netransformovaný rámec odpovídá třeba větě „*Podnik zaměstnal studenty jako brigádníky*“ [vym], po použití pravidla mu vyhovuje její pasivní obdoba „*Studenti byli podnikem zaměstnáni jako brigádníci*“ [vym].

Ani tato rozsáhlá transformační pravidla však nedokázala zajistit, aby rámce odpovídaly formě ve všech případech. 88 případů bylo označeno za výjimky, na něž se kontrola valence neaplikovala, protože podle anotátorů byl sice rámec v pořádku, ale formálně se tato korespondence nedala ověřit (nejčastějšími slovy, u nichž se nepodařilo ověřit správnost rámce, byla slovesa „*informovat*“[10)] a „*být*“; důvodem selhání kontrolního makra mohla být složitá analytická struktura nebo transformace, která se v datech vyskytovala v tak malém počtu, že bylo zbytečné ji popisovat pravidlem).

[10)] Šlo o poměrně časté konstrukce type „*Jak jsme v našem listě již informovali, udělá...*“ [zm], kde je sloveso „*informovat*“ označeno jako část parenteze a má doplněný patiens, od něhož vede koreferenční odkaz ke slovesu „*udělá*“.

Reciprocita: Příklad reciprocity je na obrázku 3.10: sloveso „*lišit se*“ vyžaduje podle valenčního slovníku aktora v prvním pádě a patiens v předložkovém pádě `od+2`. V uvedené větě však patiens na povrchu chybí a nedá se jednoduše doplnit (třebaže rozumíme, že se ceny liší navzájem jedna od druhé). Proto je doplněn uzel s lematem `#Rcp`, který na slovo „*ceny*“ odkazuje koreferenční šipkou. Kontroly zachycení reciprocity se proto částečně překrývaly s kontrolami zachycení koreference.



Obrázek 3.10: Reciprocita: „*Ceny na území SR se liší u pump minimálně, o 10 až 20 haléřů na litr.*“ [pdt]

Způsob anotace reciprocity (viz [Mikulová a kol., 2005, str. 157]) byl navržen až velmi pozdě, takže v datech nebyla téměř nikde zachycena. Všechna místa, která již byla anotována a vyznačení reciprocity v nich chybělo, se proto musela vyhledat. Používalo se několik heuristických maker, za kandidáty na doplnění recipročního aktantu byla například považována

všechna aktivní slovesa, která se někde jinde v datech vyskytla jako rodič uzlu s lematem `#Rcp` ve funkci některého aktantu a která mezi svými efektivními syny takový aktant neměla. Za podezřelé byly brány také případy splňující následující podmínky:

- na analytické rovině bylo synem uzlu odpovídajícího slovesu zájmeno „*se*“,
- některý aktant byl doplněný uzel[11] (tj. nebyl realizován na povrchu),
- aktor měl formu buď koordinace dvou nominativů („*Jan a Marie se líbají*“ [vym]) nebo nominativu v plurálu („*Milenci se líbají*“ [vym]), popřípadě mohlo jít o specifická kolektiva („*Dvojice se líbala*“ [vym]).

Koreference: Koreference (viz [Kučová a kol., 2003] a [Mikulová a kol., 2005, str. 935]) byla v datech realizována jako odkaz na identifikátor antecedentu (nebo seznam identifikátorů, pokud bylo antecedentů více, jako v případě vět „*Petr potkal Marii. Šli do kina.*“ [vym]). Existenci uzlů, na které se odkazovalo, kontrolovalo makro ze skupiny popsané v bodě „Platnost odkazů“ na straně 43.

Šipka pro gramatickou koreferenci musela vést od každého uzlu s tektogramatickým lematem `#Cor` či `#QCor` a od každého uzlu s tektogramatickým lematem `#PersPron`, pokud odpovídal zájmenu ve třetí osobě. Mnoho takových uzlů přidávali anotátoři při opravách valence. Protože však nebyli pověřeni současným anotováním koreferenčních vztahů, tento typ koreference často chyběl.

Koreferenční odkaz musel také vždy vést od uzlu s lematem `#Rcp` (viz předchozí bod), a to ke slovu vyjadřujícímu oba aktanty nebo ke spojovacímu uzlu, pokud byly tyto aktanty vyjádřeny spojovací konstrukcí.

Na tektogramatické lema `#Cor` a kontrolu obecně bylo zaměřeno několik maker. Ta obvykle procházela uzly podle seznamů slov (sloves i substantiv), která mohla být rozvita infinitivem s nevyjádřeným kontrolovaným aktantem. Výskyty těchto slov s infinitivním rozvitím bez kontrolovaného členu označovala tato makra za potenciálně chybné.

Vztažné a obsahové věty: S makry pro kontrolu koreference úzce souviselo makro kontrolující vztažné a obsahové věty: bylo postaveno na pozorování, že věty uvozené zájmeny „*který*“ a „*jaký*“ jsou buď restriktivní („*Je to člověk, který se umí dobře obléknout*“ [vym]), nebo obsahové („*Neodpověděl na otázku, který poslanec hlasoval proti*“ [vym]). V případě restriktivní vztažné věty mělo její řídící sloveso funktor `RSTR` a vztažné zájmeno se muselo shodovat v rodě a čísle s rodičem řídícího slovesa. K němu také musela vést od vztažného zájmena koreferenční šipka. V případě obsahové

[11] Při opravách valenčních rámců byl totiž namísto tehdy neexistujícího uzlu s lematem `#Rcp` často doplňován uzel s lematem `#Gen` nebo jiným podobným.

věty mělo funktor RSTR samo vztažné zájmeno a shodovalo se v rodě a čísle se svým rodičem.

Aktuální členění: Aktuální členění (viz [Veselá a Havelka, 2003] a [Mikulová a kol., 2005, str. 1051]) bylo poměrně samostatnou a složitou oblastí, jeho kontrola zahrnovala téměř třicet maker. Při anotování všech ostatních jevů docházelo často k převěšování uzlů, které mohlo anotaci aktuálního členění učinit chybnou, nebo k přidávání uzlů nových, které v době anotování aktuálního členění ještě neexistovaly. Každý nový uzel musel proto opravit anotátor aktuálního členění.

Anotace kontextové zapojenosti uzlů (atribut tfa) musela být v souladu s tektogramatickým pořadím uzlů: zjednodušeně řečeno, uzly označené t nebo c musely stát vlevo od svého rodiče, uzly označené f vpravo. Každá věta musela navíc obsahovat alespoň jeden uzel s hodnotou f atributu tfa, tedy nějaké ohnisko – to, co věta sděluje.

Pro úzkou souvislost s pořadím uzlů byly do kontrol aktuálního členění zařazeny i testy projektivity stromů. Podle teoretických předpokladů měly být totiž všechny tektogramatické stromy projektivní, ale vlivem technických řešení, jimiž se zachycovaly nejrůznější složité konstrukce na pomezí závislosti, se tento předpoklad nepodařilo udržet. Všechny neprojektivní konstrukce alespoň prošel anotátor a rozhodl, zda se strom skutečně nedá projektivizovat.

Vliv na aktuální členění měly také rematizátory – slova, která nějakou část věty zdůrazňují a vymezují ohnisko či kontrastivní základ („*i*“, „*jen*“, „*také*“, „*už*“ a další). Slova, která mohla dostat funktor RHEM, byla dána seznamem, žádné jiné slovo nemohlo být za rematizátor označeno. Za *dosah* rematizátoru, tedy jím vymezené ohnisko, byli považováni všichni jeho praví bratři včetně svých potomků; pokud navíc bylo rodičem rematizátoru sloveso, které podle tektogramatického pořadí uzlů stálo vpravo od něj, patřilo do dosahu i ono. Pro anotaci atributu tfa uzlů v dosahu rematizátoru platila různá pravidla a omezení, která se rovněž kontrolovala skriptem.

Od rematizátorů se liší modifikátory souřadného spojení (funktor CM, „coordination modifier“), jimiž se často mohla stát stejná slova jako rematizátory. Jejich funkcí však není označit ohnisko, ale blíže významově specifikovat koordinační nebo apoziční vztah. Některé z modifikátorů byly původně považovány za součásti složených spojek, ty však byly později rovněž omezeny seznamem. Bylo proto třeba změnit tektogramatická lemata spojek a původně skryté uzly odpovídající modifikátorům vrátit zpět na tektogramatickou rovinu a doanotovat je.

Poznámky anotátorů: Pro ukládání poznámek anotátorů sloužil pomocný atribut, který byl v konečné verzi **PDT** z dat odstraněn. V průběhu

oprav přibývaly stále další poznámky, například když anotátoři koreference narazili na chybu v anotaci struktury, kterou sami nemohli opravit.

Počet poznámek anotátorů se pohyboval nad pěti tisíci. Žádná striktní pravidla, jak označovat chyby stejného druhu, nebyla bohužel stanovena, takže automatické zpracování poznámek bylo velmi obtížné. K vytipování poznámek, které označovaly podobnou chybu, se používaly nejrůznější odhady: nejprve se sečetlo, kolikrát se která poznámka v datech vyskytla; pak se pro každý typ poznámky hledala automatická oprava postupně od nejčetnějších k méně četným. Podobným způsobem se sčítala i jednotlivá slova objevující se v poznámkách, takže se do stejné kategorie dostaly poznámky „morfologie“, „chyba morfologie“, „špatná morfologie“ a podobně. (Podobným způsobem se třídily výstupy z nejrůznějších kontrolních maker, pokud bylo nalezených chyb velmi mnoho. Do stejné skupiny se v takových případech dostávaly chyby týkající se stejného funktoru, lematu a podobně.)

Nejpočetnější kategorií poznámek byly poznámky jednopísmenné. Většinu z nich používali anotátoři k označení nesprávného použití malého nebo velkého písmene na začátku tektogramatického lematu (např. slovo „*Čermákem*“ dostalo lema `čermák` a anotátor uvedl jako poznámku pouze „Č“). U všech uzlů s jednopísmennou poznámkou proto makro ze skupiny `fix` měnilo počáteční písmeno lematu, pokud bylo až na velikost stejné jako písmeno v poznámce. Jednoslovné poznámky často obsahovaly stejné slovo jako lema, pouze se změněnou velikostí prvního písmene, i ty se opravovaly stejným způsobem. Podobně se řešily také poznámky jako „velké písmeno“, „název“ a podobně.

Mnoho poznámek se sice podařilo zařadit do nějaké kategorie, ale pro tuto kategorii se již nepodařilo naprogramovat automatickou opravu. Příslušné uzly pak procházel ručně anotátor, který označenou chybu opravoval. V poslední fázi oprav ještě stále zbývalo více než tisíc poznámek, které nepatřily do žádné kategorie. Na jejich ruční procházení již nebylo dost času a označené uzly tak zůstaly v datech nezkontrolované.[12)]

Gramatémy: Gramatémy (viz [Razímová a Žabokrtský, 2005] a [Mikulová a kol., 2005, str. 40]) byly v datech částečně anotovány ručně (vid obouvidých sloves, rod osobních zájmen, číslo a osoba neurčitých zájmen, rozdíl mezi „*vy*“ pro množné číslo a vykáním, hloubkové číslo pomnožných substantiv a použití komparativu běžného nebo absolutního) a částečně generovány automaticky podle morfologických hodnot a odkazů do nižší roviny (všechny ostatní, např. odkaz pomocí atributu `a/aux.rf` na modální sloveso mohl u infinitivu ovlivnit jednak gramatém modality a jednak také jeho gra-

[12)] To ovšem neznamená, že v datech je více než tisíc chyb, o kterých víme. Mnoho chyb bylo opraveno při jiné příležitosti, při které ale nebyla smazána poznámka; jindy mohl být pocit autora poznámky, že jde o chybu, mylný.

matém času). Ve speciálních případech mohli anotátoři označit, že hodnota některého gramatému neodpovídá té, která by se automaticky generovala z povrchového tvaru slova (například ve větě „*Kéž by to tam padalo celé jaro.*“ [zm] je u uzlu s lematem `padat` označeno, že gramatém `sentmod` má mít hodnotu `desid`).

Gramatémy se anotovaly až v poslední fázi anotačního procesu, protože k jejich přiřazování byla třeba plně anotovaná data, která se již dále neměnila. Z toho důvodu nezbývalo příliš času na kontroly gramatémů, třebaže navrhnout testy je snadné: nemělo by se například nikdy stát, že gramatém `tense` nemá hodnotu `post`, pokud uzel odkazuje do analytické roviny na sloveso v budoucím čase. Podobně by se daly kontrolovat kombinace časů pomocných sloves, na která se odkazuje od tektogramatických uzlů – např. kombinace budoucí a minulý čas není v češtině pro složený slovesný tvar možná.

Tektogramatická lemata: Kontrola složených tektogramatických lemat (viz [Mikulová a kol., 2005, str. 14]) byla již popsána v bodě „Odkazy na nižší rovinu“ na straně 44 a následujících. Další kontrola hledala mezi lematy potenciální jména, která tak ale nebyla označena (a tektogramatické lema tedy začínalo malým písmenem). Podezřelá byla všechna slova, která začínala velkým písmenem a vyskytovala se ve větě na jiném než prvním místě (přičemž se nepočítaly uvozovky a další grafické značky na začátku věty), jejichž lema začínalo malým písmenem.

Vlastní jména: U každého vlastního jména by měl být vyplněn atribut `is_name_of_person` hodnotou 1 (viz [Mikulová a kol., 2005, str. 778]). Všechny kandidáty procházel anotátor, který rozhodoval, zda se jedná o vlastní jméno.

Přímá řeč a uvozovky: Řídící uzly přímé řeči mají atribut `is_dsp_root` vyplněný hodnotou 1 (viz [Mikulová a kol., 2005, str. 652]). Tento atribut se používal například při kontrole valence – některá slovesa nepřipouštějí jako své doplnění libovolnou větu, ale právě přímou řeč (například slovesa „*dodat*“, „*dotázat se*“, „*hlásit*“, „*říci*“ a podobně). Pokud na místě doplnění přímou řečí stálo sloveso, které nemělo vyplněný atribut `is_dsp_root`, jednalo se většinou o chybu. Všechna místa, kde se požadavky valenčního slovníku neshodovaly s anotací přímé řeči, se ručně procházela a opravovala se jak data, tak valenční slovník.

V datech je rovněž označen každý úsek textu, který se ocitl v uvozovkách – atribut `quot` mají vyplněný všechny uzly, které odpovídají slovům uvnitř uvozovek (viz [Mikulová a kol., 2005, str. 929]). Každý takový úsek je označen jednoznačným identifikátorem a typem: rozlišuje se přímá řeč (atribut `quot/type` má hodnotu `dsp`), citát (`citation`), metajazykové užití

(meta), název (title) a ostatní (other). Anotování prováděl ručně anotátor, kontrolovala se jednoznačnost identifikátorů označených úseků.

V poslední fázi práce na **PDT** 2.0 byl navržen test, který uvádí tyto dva jevy do souvislosti: nemělo by se totiž stát, že nějaký uzel je označený jako quot/type=dsp, jeho rodič má atribut quot prázdný, ale uzel není označen jako kořen přímé řeči. V datech byly objeveny tři chyby, jednalo se o uzly, které se ve stromech objevily až po ukončení anotace atributu quot, ale jeho hodnota u nich nebyla doplněna.

Subfunktory: Přiřazení subfunktorů (viz [Mikulová a kol., 2005, str. 595]) se týkalo pouze vybraných funktorů. Proběhlo automaticky, jediným kritériem pro rozhodnutí o subfunktoru byly povrchové formy a lemata slov, na něž se odkazovalo pomocí referencí do nižší roviny.

subfunktor	seznam forem
basic	adv, adj, na+6, v+6
near	u+2, blízko+2, blízko+3, v_blízkost+2, nedaleko+2, poblíž+2, po_bok+2, při+6, k+3, vedle+2
around	kolem+2, okolo+2
along	podél+2, podle+2
betw	mezi+7
above	nad+7
below	pod+7
center	uprostřed+2, vprostřed+2, prostřed+2
opp	naproti+3, proti+3, přes+4
elsew	mimo+4, stranou+2, vně+2
front	před+7, tvář_v_tvář+3
abstr	v_oblast+2, v_obor+2
in	uvnitř+2
behind	za+7

Tabulka 3.3: Seznam subfunktorů pro funktor LOC

Jako příklad uvádí tabulka 3.3 seznam subfunktorů pro funktor LOC (pád odpovídá formě slova, na něž vede odkaz pomocí atributu a/lex.rf, ostatní slova mají uvedené lema). Každý uzel s funktorem LOC dostal přiřazen právě jeden subfunktor. Hodnota basic byla přiřazována i všem příslovcím, protože u nich je bližší specifikace významu funktoru dána lexikálně, a nikoliv subfunktorem.

3.2 Počty změn

Bylo by asi marné snažit se najít objektivní hodnocení úspěšnosti sestavy kontrolních maker a programů. Rozsah dat a jejich složitost neumožňují stanovit jednoznačná kritéria správnosti, nadto se mnoho sporných bodů nachází v dosud nedostatečně prozkoumaných oblastech, takže ani není možné o nich rozhodnout. O mnohých nedostatcích stávajících dat víme, protože jsme jejich řešení museli odložit z časových důvodů – ať už nebylo dost času na provedení opravy, nebo na podrobnější zpracování celého jevu. Věříme, že v rámci dalších projektů budou tyto chyby postupně odstraňovány a kvalita dat se bude dále zvyšovat.

Alespoň některé aspekty oprav **PDT** se však dají kvantifikovat a porovnat – a právě tím se budeme zabývat v tomto oddílu. Půjde hlavně o počty změn v datech, které opravy způsobily, a o jejich klasifikaci, popřípadě o porovnání různých verzí dat a zdrojů, které se k jejich anotaci používaly.

3.2.1 Morfologická analýza a data

Morfologický analyzátor je program, který každé slovní formě nabízí všechna možná lemata a pro každé z nich všechny možné tagy, které se jí dají přiřadit. Při anotování morfologické roviny vybírali anotátoři ze všech nabízených možností tu správnou, pokud však správná možnost mezi nabízenými nebyla, mohli ji doplnit sami. Při opravách dat se rovněž používal morfologický analyzátor (viz závěr oddílu 2.3), v mnohých případech však museli i tady anotátoři doplňovat lemata a tagy, které slovník morfologického analyzátoru neobsahoval.

Kategorie	výskytů	typů
Známé lema, ale tag vyžaduje jiné	238	124
Známé lema, ale tag neznámý	5 249	2 710
Neznámé lema, ale tag možný s jiným	2 370	1 133
Neznámé lema, ale tag možný s podobným	6 150	1 031
Neznámé lema i tag	20 535	13 913

Tabulka 3.4: Rozdíly mezi morfologickou anotací a hodnotami nabízenými morfologickým analyzátorem

Tabulka 3.4 přináší porovnání stavu morfologických dat po skončení oprav a poslední verze morfologického analyzátoru. Jsou v ní shrnuty výsledky porovnání morfologických lemat a tagů přiřazených všem slovům v **PDT** s lematy a tagy, které nejnovější verze morfologického analyzátoru nabízela. Jednotlivé kategorie odchylek dat a možností nabízených analyzátorem mají následující význam:

Známé lema, ale tag vyžaduje jiné: Morfologický analyzátor umožňuje dané formě přiřadit stejné lema, jako je uvedeno v dat-

ech, pro toto lema však nenabízí stejný tag. Tag, který je v datech uveden, by forma mohla podle analyzátoru dostat, pokud by jí bylo přiřazeno jiné lema. Nejvíce výskytů měly krátké zkratky, nejčastější byla „*V*“, která dostala v datech jednadvacetkrát lemma `V-4_:B_;K` s tagem `NNFXX-----A---8`, což je pro poslední verzi analyzátoru nepřípustná kombinace (toto lema totiž podle analyzátoru odpovídá zkratce za adjektivum s tagem `AAXXX----1A---8`).

Známé lema, ale tag neznámý: Morfologický analyzátor nabízí pro danou formu stejné lema, jako je uvedeno v datech. Naproti tomu tag, který je uveden v datech, pro tuto formu analyzátor nenabízí (ani s žádným jiným lematem). Nejvíce výskytů mají zkratky a cizí jména – v ruční morfologii je obvykle do tagu doplněna některá morfologická kategorie, kterou analyzátor považuje za neurčitelnou. Nejčastější je zkratka „*S*“ a jméno „*Ševardnadze*“.

Neznámé lema, ale tag možný s jiným: Morfologický analyzátor neumožňuje přiřadit dané formě lema, které je uvedené v datech. Pokud by však tato forma dostala jiné lema, nabízel by pro ně analyzátor stejný tag, jaký je uveden v datech. Jde hlavně o nepříliš frekventovaná slova, kterým se změnilo lema ve slovníku morfologického analyzátoru – například pro slovo „*šíři*“ nabízí morfologická analýza lemata `šíř_^(jeden_z_rozměrů)` a `šíře_^(jeden_z_rozměrů)`, zatímco v datech je přiřazeno dřívější `šířka_^(jeden_z_rozměrů)`.

Neznámé lema, ale tag možný s podobným: Tato možnost je podobná předchozímu případu. Lema, které by podle analyzátoru odpovídalo tagu, který je uveden v datech, je navíc téměř shodné s lematem, kterým byla forma anotována: liší se pouze koncová část lematu, která obsahuje vysvětlivky, zkratky sémantických či stylových kategorií nebo klasifikuje homonyma. Jedná se o poměrně častá slova, hlavně o různá zachycení zkratek (například „*tzv*“ má v datech lemma `takzvaný_:B`, zatímco morfologická analýza nabízí `takzvaný_:B_,x`, znak `x` přitom jen indikuje, že se dané slovníkové heslo nemá používat při syntéze) nebo slova, u nichž se změnily některé kategorie (například jméno „*Miroslav*“ dostávalo původně lema `Miroslav_;Y` – křestní jméno, kdežto v novější verzi morfologické analýzy dostává lema `Miroslav_;G_;Y` – křestní jméno nebo geografický název).

Neznámé lema i tag: Morfologický analyzátor nenabízí pro danou formu žádnou analýzu. Jde o nejrůznější zkratky a cizí slova, zvláštní skupinu tvoří první části složenin jako „*česko-německý*“ či „*izraelsko-palestinský*“, kterým přísluší tag `A2--------A----`, který jim anotátoři morfologické roviny správně přiřadili. Takový tag však morfologická analýza vůbec nezná.

Při získávání *počtu typů* počítáme na rozdíl od počtu výskytů každou formu jen jednou bez ohledu na to, kolikrát se v datech objevuje. Nejvyšší

poměr mezi počtem jednotlivých výskytů a počtem typů se objevuje v kategorii podobných lemat, jde tedy o slova s poměrně vysokými počty výskytů. Ostatní kategorie obvykle obsahují jen několik slov s větším počtem výskytů, zbytek vyplňují nejrůznější slova s jediným výskytem, jak lze ostatně usuzovat z druhého Zipfova zákona (viz např. [Baayen, 2001]).

3.2.2 Změny v tektogramatických datech

Jak již bylo uvedeno v úvodu pododdílu 3.1.4, největší pozornost byla při poanotačních opravách korpusu (a v této práci) věnována tektogramatické rovině. Díky tomu se nejvíce úprav týkalo tektogramatických dat, kterými ovšem nerozumíme pouze tektogramatickou rovinu, ale všechny roviny anotace na datech, která byla na tektogramatické rovině anotována. Mnohé chyby na nižších rovinách se totiž daly automaticky vyhledat teprve při konfrontaci s tektogramatickou anotací.

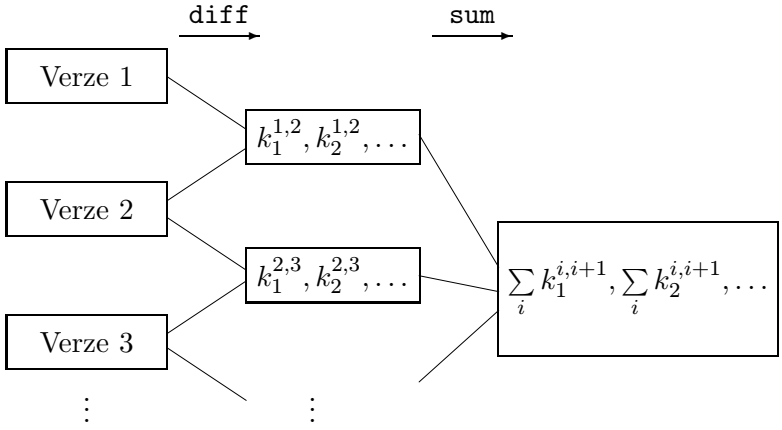
Některé chyby v datech se podařilo objevit až dlouhou dobu po tom, co vznikly. Protože část z nich mohla mít destruktivní charakter (smazaná část uzlů apod.), bylo třeba zachovávat i starší verze dat, aby se z nich dala popřípadě zrekonstruovat jejich původní podoba.

Jednotlivé verze dat byly zpočátku ukládány nepravidelně; v pozdějším období, kdy se data intenzivně měnila, se nová verze dat vytvářela každý týden (celkem vzniklo 25 verzí dat). Jednotlivé verze dat se daly porovnávat, tj. bylo možné sledovat, kolik práce se v různých oblastech anotace udělalo.

Pro sledování změn v datech existovalo několik krátkých programů. Prvním z nich bylo makro `changes` pro **ntred**, které procházelo všechny uzly tektogramatických dat a pro každý z nich zaznamenávalo hodnoty sledovaných atributů, identifikátor rodiče, identifikátor analytického rodiče, pokud uzel odpovídal uzlu analytické roviny, a identifikátor předcházejícího uzlu v tektogramatickém slovosledu. Další program `changes-diff` požadoval na vstupu dva výstupy z makra `changes` (typicky šlo o výstupy ze dvou po sobě jdoucích verzí dat), jeho úkolem bylo spočítat změny v jednotlivých sledovaných kategoriích. Poslední program `changes-sum` pak požadoval na vstupu libovolný počet výstupů programu `changes-diff`, pro každou kategorii sčítal jednotlivé dílčí počty změn a vracel jejich celkový součet. (Schéma práce programů pro sledování změn je na obrázku 3.11, k_i představují počty změn v jednotlivých kategoriích.)

Samotný výstup z programu `changes-sum` se však musel dále upravovat. V prvních verzích dat existovalo množství uzlů, jejichž identifikátory nebyly jedinečné – program tedy nemusel jednoznačně určit, zda porovnává v obou verzích tentýž uzel, a mohl všechny rozdíly mezi špatně spárovanými uzly považovat za změny v datech. Pro takové verze se tedy musely všechny počty změn proporcionálně zmenšit, aby tato chyba neměla na výsledek vliv.

Některé další kategorie nemělo smysl sledovat v době, kdy jejich anotace probíhala odděleně (např. koreference). Byla-li anotace takové kategorie k



Obrázek 3.11: Schéma počítání změn

přilita do dat ve verzi v, muselo se od počtu změn odečíst číslo $k^{v-1,v}$, které neznamenalo změny, ale právě přilití anotace do dat.

Upravené hodnoty jsou předloženy v tabulce 3.5. Přestože jsou čísla uvedena s přesností na jednotky, je třeba je chápat jen jako velmi přibližné odhady – změna v jedné kategorii často vyvolávala změny i v dalších kategoriích a korekce jednotlivých počtů byly často jen těžko přesněji odhadnutelné. Významy jednotlivých sloupců a řádků si vysvětlíme podrobněji:

Počet změn: Tento sloupec uvádí počet změn v dané kategorii, v případě většiny kategorií upravený oproti výstupu programu `changes-sum` tak, jak bylo popsáno výše.

Procent všech: Udává, kolik procent ze všech tektogramatických uzlů odpovídá uvedenému počtu změn. Znak = označuje případ, kdy hodnota v tomto sloupci je stejná, jako hodnota ve sloupci „Procent relevantních“ (viz další bod).

Procent relevantních: Pro každou kategorii existuje jistá množina uzlů, které se daná kategorie může týkat (například sledování rodiče nemá smysl pro technický kořen). Pokud je touto množinou právě množina všech tektogramatických uzlů, je v předchozím sloupci uvedena značka =. V případě přidaných a smazaných uzlů je počet změn v závorce, protože se nedá říci, že „1,4 % uzlů bylo smazáno“ – celkový počet uzlů se neustále měnil, takže se dá těžko určit, z jakého základu by se procenta měla počítat. Navíc přidávání a mazání uzlů souviselo nejčastěji s anotací valence, takže počet přidaných uzlů rozhodně nevypovídá o chybách v původně anotovaných datech.

Pro tektogramatickou strukturu byla za relevantní považována množina všech uzlů kromě technických kořenů, stejně tak pro funktor. Pro spojovací konstrukce se za relevantní považovaly všechny uzly,

Kategorie	Počet změn	% všech	% relevantních	Počet rozdílů
Přidané uzly	20 222	=	(2,4)	17 867
Smazané uzly	11 486	=	(1,4)	9 131
Forma	2 281	=	0,3	723
Důvod změny formy (`formtype`)	5 796	=	0,7	3 810
Důvod změny formy (`origfkind`)	2 829	=	0,3	2 449
M-lema	12 957	=	1,5	8 886
Tag	29 573	=	4,0	22 796
Analytická struktura	4 973	=	0,6	3 001
Analytická funkce	4 453	=	0,5	2 873
Tektogramatická struktura	37 865	4,1	5,6	32 642
Funktor	37 330	5,2	5,5	31 918
T-lema	25 609	=	3,8	22 085
Skryto	2 884	=	0,3	1 623
Odkryto	6 278	=	0,7	4 048
Operand	399	0,0	25,9	249
Memberof	11 330	1,2	42,5	5 626
Odkazy do analytické roviny	54 173	6,0	42,1	42 365
Koreference – odkazy	6 348	0,7	13,4	@
Koreference – typ	5 440	0,6	11,5	@
Exofora, segment	278	0,0	1,0	@
Aktuální členění	27 249	3,0	6,3	@
Hloubkový slovosled	51 383	=	5,6	@

Tabulka 3.5: Počty změn

které byly v poslední verzi dat součástí nějaké spojovací konstrukce. Podobně pro odkazy do analytické roviny tvořily množinu relevantních uzlů všechny uzly, které ve finální verzi dat obsahovaly nějaký odkaz do analytické roviny, a relevantní množinou pro koreferenci byly uzly, od kterých v této verzi vedla kterákoliv koreferenční šipka. Pro aktuální členění se za relevantní považovala množina uzlů, pro které mělo smysl uvažovat o jedné z hodnot `t`, `f` nebo `c` pro atribut `tfa`.

Počet rozdílů: Tento sloupec odpovídá výstupu programu `changes-diff`, který má na vstupu pro každou kategorii tu verzi dat, v které se poprvé objevila, a jejich verzi poslední. Číslo je vždy menší než počet změn – mohlo se totiž stát, že některému uzlu se daná kategorie měnila několikrát, takže zatímco v počtu změn je započítána každá úprava, v počtu rozdílů odpovídá těmto změnám pouze jediný rozdíl (nebo dokonce žádný, jestliže konečná hodnota dané kategorie byla stejná jako

její hodnota počáteční). Rozhodně tedy nelze říci, že rozdíl mezi počtem změn a hodnotou tohoto sloupce odpovídá „zbytečné práci, protože se data nakonec stejně vrátila do původního stavu“ – spíše ukazuje, že změny probíhaly opakovaně na týchž místech, která bychom mohli označit za „problematická“. Na toto číslo mělo vliv i přidávání a mazání uzlů – mohlo se totiž stát, že byl smazán nějaký uzel a později byl přidán uzel nový se stejným identifikátorem, což se mohlo v počtu rozdílů započítat do všech kategorií. Čísla byla proto odhadem zmenšena podle toho, kolik se mazalo a přidávalo uzlů.

Znak @ je uveden tam, kde byl počet změn téměř shodný s počtem rozdílů, protože se s anotací uvedené kategorie začalo později nebo byla do dat přilévána až v závěru práce.

Přidané a smazané uzly: Za přidaný uzel považoval program každý uzel, na který narazil ve verzi v, pokud ve verzi $v-1$ nenašel uzel se stejným identifikátorem. U smazaných uzlů tomu bylo naopak.

Forma, změny formy, m-lema, tag: Změny morfologických atributů souvisely hlavně s kontrolami zmíněnými v pododdílu 3.1.2 a s kontrolou valence (pododdíl 3.1.4). Za povšimnutí jistě stojí, že tag se změnil u plných čtyř procent uzlů, což je již srovnatelné číslo s chybovostí nejúspěšnějších statistických taggerů češtiny (viz např. [Vidová-Hladká, 2000]).

Analytická struktura a funkce: Analytická struktura byla reprezentována identifikátorem analytického rodiče, analytická funkce hodnotou atributu `afun`.

Tektogramatická struktura a funkce, t-lema: Tektogramatická struktura byla reprezentována identifikátorem rodiče na tektogramatické rovině, funktor a tektogramatické lema odpovídajícími atributy.

Skrývání uzlů: Skryté uzly odpovídají podle nového anotačního schématu (viz oddíl 1.3) uzlům analytické roviny, na které nevede odkaz pomocí atributu `a/lex.rf`. U každého uzlu bylo makrem `changes.ntred` zaznamenáno, zda byl skrytý či nikoliv.

Spojovací konstrukce: Atributy `memberof` a `operand` odpovídaly v podstatě dnešnímu atributu `is_member`. Ten má hodnotu 1, pokud je daný uzel členem spojovací konstrukce, původní atributy navíc rozlišovaly typ spojovací konstrukce, a to jednak tím, že pro koordinaci a apozici existoval jeden a pro konstrukce se spojovacím uzlem s funktorem `OPER` druhý, ale také tím, že atribut `memberof` obsahoval informaci o typu spojovací konstrukce. Typ spojovací konstrukce se ale dá jednoznačně určit z funktoru spojovacího uzlu, takže se v novém anotačním schématu již zvlášť neukládá.

Odkazy do analytické roviny: Seznam odkazů na analytické uzly se zaznamenával uspořádaný, aby za změnu nebylo považována pouhá změna pořadí odkazů.

Koreference: Ve starém anotačním schématu (viz oddíl 1.3) byly koreferenční šipky reprezentovány dvěma atributy: jeden obsahoval seznam identifikátorů a druhý seznam odpovídajících typů koreferenčních vztahů (tj. k prvnímu identifikátoru seznamu vedla šipka takového typu, který byl v seznamu jako první, a tak dále). Změna typu koreference bude asi ve většině případů znamenat přidání koreference tam, kde předtím žádná nebyla, rozdíl mezi počtem odkazů a počtem typů pak více méně představuje ty případy, kdy byl sice zachycen koreferenční vztah, ale jako antecedent byl označen nesprávný uzel.

Aktuální členění: Sledována byla jednak hodnota atributu `tfa` a jednak hloubkový slovosled. Ke sledování změn slovosledu byl zaznamenáván identifikátor předcházejícího uzlu – zjištěný počet změn se proto musel vydělit třemi, protože přesun uzlu se většinou projevil změnou tohoto údaje u tří uzlů: u uzlu samotného, u jeho původního následníka a u jeho nového následníka (pokud byl uzel před přesunem nebo po něm na konci věty, projevila se změna pouze u dvou uzlů, takové případy zanedbáváme).[13)]

V tabulce jsou kategorie, které reprezentují stejný jev nebo jeho různé aspekty, zobrazeny bez dělicích čar (a ve vysvětlivkách jsou sloučeny do společných bodů). Všimněme si, že v naprosté většině těchto „nadkategorií" jsou počty změn v jednotlivých kategoriích podobné, zatímco jedna nadkategorie se od ostatních může podstatně lišit.

Jistou výjimkou je nízký počet změn u zvláštních případů koreference (exofora a segment) ve srovnání s klasickými případy, jejich absolutní počet je však tak nízký, že jeho vypovídací hodnota je téměř nulová (podobně jako u atributu `operand`). Nepoměr panuje také mezi počtem skrytých a odkrytých uzlů, jeho příčinou je již dříve zmíněná chyba automatické převodní procedury (pododdíl 3.1.1), která za jistých okolností skrývala velké části vět, které měly zůstat viditelné. Rovněž oprav v morfologickém tagu je o mnoho více než oprav v ostatních morfologických atributech – to však zřejmě souvisí s tím, že morfologický tag obsahuje větší množství informace (takže by se dal rozdělit až na patnáct atributů). Tag má navíc pro každou formu velký počet možných hodnot, takže existuje větší riziko chyby, a jeho kontrola byla součástí velkého množství analytických i tektogramatických oprav včetně kontrol valence, takže byl také často opravován.

Za povšimnutí také jistě stojí ty oblasti, v nichž bylo provedeno nejvíce oprav (a pokud připustíme, že kontroly dat byly dostatečně rozsáhlé, můžeme také říci, že jde o oblasti, kde anotátoři nejvíce chybovali). Jedná se

[13)] Přidaný uzel se naopak mohl projevit až jako pět změn: dvě změny při jeho přidání a tři při jeho pozdějším zařazení na správné místo anotátorem aktuálního členění.

v první řadě o odkazy do analytické roviny, na jejichž opravy bylo zaměřeno mnoho maker (viz bod „Odkazy na nižší rovinu“ na straně 44). Přesný význam odkazů byl většině anotátorů dlouhou dobu neznámý, původně se s nimi dokonce nepočítalo vůbec. Jedná se navíc o jev, který se dá v prostředí grafického editoru jen těžko vizualizovat – uvádí totiž do vztahu prvky různých rovin a často dokonce i z různých vět. Proto jeho anotování i kontrola vyžaduje velké úsilí a soustředění.

Další velmi problematickou oblastí byly spojovací konstrukce. Příčinu těžkostí můžeme opět hledat v tom, že spojovací konstrukce se nedají snadno zobrazit. Pokud jsou zachycené tak, jak je popsáno v kapitole 2.2, tak vlastně naopak zobrazenou strukturu narušují a komplikují. Anotování složitějších vnořených spojovacích konstrukcí se společnými rozvitími je, podobně jako anotování odkazů do analytické roviny, činností náročnou a vyčerpávající.

Zdaleka ne všechny chyby se podařilo opravit. Některé konstrukce byly natolik složité, že nebylo možné je kontrolovat automaticky, ale zároveň se vyskytovaly v datech tak často, že nebylo v silách anotátorů je všechny projít v čase, který byl k dispozici na dokončení projektu. Takové konstrukce zůstaly v datech zachycené chybně, ale o chybách se ví a opravovat se budou průběžně tak, aby je další verze **PDT** již neobsahovala.

Celkový počet oprav, uvedený v tabulce 3.5, je 361 136. Pokud by se každá oprava týkala jiného uzlu, bylo by opravami po první anotaci změněno přes 43 % uzlů. Domníváme se, že toto číslo ukazuje, že poanotační opravy dat měly smysl, a doufáme, že díky nim jsou data v podstatně lepším stavu, než v jakém byla po dokončení anotování.

Závěr

> Myslím, že na tomto místě už je třeba přestat, bezejmenný ostrov v cizích mořích je patrně hoden toho, aby na něj ten, kdo se vrátil, myslel za nocí beze spánku a aby mu jednou věnoval knihu, v níž by zazpíval...
>
> *Michal Ajvaz, Zlatý věk*

Podíl práce

Práce takového rozsahu, jako je vytváření korpusu o více než sto tisíci větách, nemůže být dílem jednoho člověka, ale je naopak nutně dílem kolektivním. Vzhledem ke složitosti jednotlivých dílčích částí není možné počítat ani srovnávat, do jaké míry se konkrétní pracovníci na anotovaném korpusu jako celku podíleli.

Přesto by asi bylo vhodné popsat, kterým částem korpusu se podstatněji věnoval autor této práce. Nebudeme přitom brát v potaz celou řadu úkolů jako sepisování dokumentace, příprava vydání kompaktního disku či vytváření nejrůznějších programů a skriptů, protože se o nich v této práci nezmiňujeme.

V první řadě je autor práce tvůrcem funkcí popsaných v pododdílu 2.2.1, a to od návrhu algoritmu až po samotnou implementaci. Jak je zřejmé z tabulky 3.1, je autor práce také tvůrcem velkého počtu kontrolních maker. Ani v tomto případě se nejednalo jen o programování, ale bylo třeba nejprve navrhovat, které jevy by měly být kontrolními makry sledovány a jak.

Kromě toho se autor práce podílel na návrhu nového anotačního schématu, popsaného v oddílu 1.3. Navrhl a implementoval též nástroje na sledování počtu změn v datech popsané v oddílu 3.2.2 a jeho dílem je rovněž propojení morfologického analyzátoru s editorem **TrEd** (viz oddíl 2.3). Dále vyvinul nebo se podílel na vývoji drobných pomocných programů, z nichž byly v oddílu 2.3 jmenovány nástroje `rcut` a `join`.

Někteří další pracovníci, kteří se na tvorbě **PDT** podíleli a jejichž činnost úžeji souvisela s předmětem této práce, jsou vyjmenováni v oddílu Poděkování na straně iii.

Zpracování dat

Přesvědčení, že poanotační opravy významnou měrou přispěly k udržení konzistence dat a k jejich kvalitě, k němuž jsme dospěli na konci kapitoly 3, je možné zobecnit pro práci s velkými objemy dat nejrůznějšího zaměření: Nejúčinnější způsob, jak udržet konzistenci dat a minimalizovat jejich chybovost, je periodicky ověřovat platnost invariantů, které by v nich měly platit. Pokud je možné některé takové kontroly provádět již v průběhu vytváření dat, může zpětná vazba pomoci zlepšit jeho úroveň a ušetřit tak mnoho dodatečné práce při opravách chyb.[14)]

Sada kontrolních procedur pak nejen zaručí integritu dat, ale také může sloužit jako zdroj pro porovnávání dat v různých fázích jejich vývoje. Snadno tak lze srovnávat přínos jednotlivých kontrolních maker.

Najít obecný návod na vytváření kontrolních procedur asi není možné. Každý korpus má svá specifika, která mají svůj původ v jazyce či jazycích, které zachycuje, v datovém formátu, který k tomu používá, v množině jevů, které popisuje, a v teoretickém rámci, který k popisu používá. Kontrola korpusu anglických vět s frázovou syntaktickou analýzou by například musela zaručit správné užití neterminálních symbolů, zatímco shodu přívlastku s řídícím jménem by bylo nesmyslné kontrolovat.

V případě **PDT** byl jediným „automatickým“ nástrojem na generování kontrolních procedur automatický hledač chyb, popsaný v oddílu 3.1. Podobný algoritmus se dá snadno použít na libovolná anotovaná data, pro která platí, že každým dvěma stejným úsekům na nižší rovině by měly odpovídat dva stejné úseky na rovině vyšší. Výstup z automatického nástroje však bývá nutno dále zpracovat a vytřídit skutečné chyby, k tomu je možné použít dále popsaný obecný postup pro procházení velkého množství výskytů.

Při budování korpusu se obvykle formulují pravidla, jimiž se anotátoři řídí. Velkou část kontrolních procedur můžeme tedy získat tak, že je necháme ověřovat dodržování těchto pravidel. Zvláštní pozornost je přitom třeba věnovat pravidlům, která popisují zachycení jevů, jejichž intuitivní chápání v jistém smyslu překračuje rámec zvoleného formalismu, tedy popisům arbitrárního způsobu reprezentace jevů, k jejichž zachycení, které by intuici odpovídalo, by bylo třeba formalismus nějakým způsobem rozšiřovat a měnit (např. zachycení srovnání v **PDT**, viz [Mikulová a kol., 2005, str. 669]). Chybám v anotaci je možno předcházet již při konstrukci anotačních pravidel: je třeba dbát na to, aby se různé jevy anotovaly dostatečně odlišně, čímž se sníží riziko vzniku chyby způsobené „přehlédnutím“ nějakého

[14)] Některé chyby by mohl kontrolovat přímo anotační nástroj. Všechna kontrolní makra v něm však být implementována nemohou: jednak některé kontroly vyžadují přítomnost všech anotovaných dat, která anotátor nemusí mít k dispozici, jednak by neustálé kontrolování velmi zpomalovalo práci.

důležitého detailu, kterým se zachycení daného jevu liší od zachycení jevu jiného.

Další rizikovou oblast tvoří jevy, které nejsou v anotačních nástrojích snadno vizualizovatelné (v případě **PDT** např. zachycení koordinací, viz oddíl 2.2). Jedná se většinou o vlastnosti vyššího řádu, tj. o relace na množinách anotovaných objektů (v případě koordinací jde o množiny analytických, resp. tektogramatických uzlů). Mnohým chybám tohoto druhu je proto možné předejít vhodným návrhem anotačních nástrojů, pokud budou schopny komplexní jevy nějakým způsobem zobrazit.

Podobnou skupinu tvoří také vztahy mezi rovinami (srov. bod „Odkazy na nižší rovinu“ na straně 44). Anotátoři jedné roviny nemusejí podrobně znát anotační pravidla platná pro jinou rovinu, a tedy ani nemusejí být schopni kontrolovat jejich vzájemný vztah během anotování. V případě **PDT** používala pro svou práci více než jednu rovinu většina kontrolních maker, třebaže jsou rozdělena do kategorií podle rovin (viz první odstavec oddílu 3.1). Pohled „napříč rovinami“ umožňuje hledat chyby tam, kde by se v rámci žádné z rovin samostatně nedaly najít.

Problematickým se může jevit i samotné zpracování výstupu z kontrolní procedury. Pokud je nalezených chyb příliš mnoho, nedají se všechny procházet ručně. Příklad možného postupu je uveden v odstavci „Poznámky anotátorů“ na straně 65: jednotlivé výskyty je třeba třídit podle relevantního atributu, aby se podobné chyby dostaly k sobě a mohlo pro ně být nalezeno automatické řešení. Při ručním procházení takto setříděných výskytů má anotátor možnost „přeskočit“ skupinu, která je za chybnou považována mylně, a může pokračovat opravami v dalších skupinách.

Dva pohledy

Jak je z jednotlivých částí práce patrno, bylo pro řešení popsaných problémů často nutno využívat jak znalosti o jazyce, tak znalosti z oblasti algoritmů a počítačů. Tento rys je jednou z hlavních charakteristik této práce: přináší totiž dva pohledy na projekt **PDT**.

Jeden z nich je pohledem *lingvistickým*, který sleduje ověření teorie **FGP** jejím „nasazením“ v praxi. Sleduje jednotlivé jazykové jevy tak, jak jsou využívány v jazyce, zpracovány v teorii a zaznamenány v korpusu.

Druhý pohled je pohledem *informatickým*. Sleduje nástroje, kterými je korpus vytvářen a zpracováván, a pozoruje či porovnává postupy a algoritmy, které jsou pro implementaci lingvisticky motivovaných úloh navrhovány a používány.

Spojení těchto dvou pohledů přináší nové podněty pro obě oblasti a navzájem je inspiruje a obohacuje. Právě v těžišti těchto zdánlivě odlehlých oblastí, v tomto průsečíku iluzorních mimoběžek leží hlavní důraz této práce.

Rejstřík

U každého termínu nebo zkratky je uvedeno číslo stránky, na které je vysvětlen. Na takové straně se většinou objevuje poprvé. Pokud je termín vysvětlován (tj. jeho různé aspekty jsou vysvětlovány) na více místech, jsou uvedeny všechny stránky.

English Summary

Capturing a Sentence Structure by a Dependency Relation in an Annotated Syntactical Corpus (Tools Guaranteeing Data Consistence)

This work describes various methods and tools used during the annotation and post-annotation checking of Prague Dependency Treebank 2.0 data. The annotation process of the treebank was complicated by several factors: for example, the corpus was divided into several layers that must reflect each other. Moreover, the annotation rules changed and evolved during the annotation. In addition, some parts of the data were being annotated separately and in parallel and had to be merged with the data later. Conversion of the data from an old format to a new one was another source of possible problems besides omnipresent human inadvertence.

In the first chapter, the theoretical background of the Prague Dependency Treebank is given. Differences between theoretical assumptions and their realization in the treebank are described together with various aspects of annotation schemata and data format.

The second chapter describes the tools used for annotation and during the post-annotation checking. Special attention is paid to coordination- and apposition-like constructions: several possible approaches are compared and functions for resolving complex structures (encoded according to the used approach) are presented.

The third chapter expounds the post-annotation checking procedures used to ensure data integrity and correctness. The procedures are classified by several criteria and many linguistically inspired examples are given. In the last part of the chapter, the impact of the checking procedures is measured by counting the number of changes in the data, the procedures are compared and their contribution and usefulness is discussed.

Použitá literatura

R. Harald Baayen. *Word Frequency Distributions*, 18. svazek řady *Text, Speech and Language Technology*. Kluwer, Dordrecht, Nizozemí, 2001.

Sabine Brants a Silvia Hansen. Developments in the TIGER Annotation Scheme and their Realization in the Corpus. In *Proceedings of the Third Conference on Language Resources and Evaluation (LREC 2002)*, strany 1643–1649, Las Palmas, 2002.

Alena Böhmová. Automatic Procedures in Tectogrammatical Tagging. *The Prague Bulletin of Mathematical Linguistics*, 76:23–34, 2001.

Markus Dickinson. *Error detection and correction in annotated corpora*. Disertační práce, The Ohio State University, 2005. URL `[http://ling.osu.edu/~dickinso/papers/diss/]`.

Jan Hajič. *Disambiguation of Rich Inflection (Computational Morphology of Czech)*. Karolinum, Praha, 2004.

Jan Hajič, Jarmila Panevová, Zdeňka Urešová, Alevtina Bémová, Veronika Kolářová a Petr Pajas. PDT-VALLEX: Creating a Large-coverage Valency Lexicon for Treebank Annotation. In Joakim Nivre a Erhard Hinrichs, editoři, *Proceedings of the Second Workshop on Treebanks and Linguistic Theories*, 9. svazek řady *Mathematical Modeling in Physics, Engineering and Cognitive Sciences*, strany 57–68, Växjö, 2003. Växjö University Press.

Bohuslav Havránek a Alois Jedlička. *Česká mluvnice*. SPN, Praha, čtvrté vydání, 1981.

Lucie Kučová, Veronika Kolářová, Zdeněk Žabokrtský, Petr Pajas a Oliver Čulo. Anotování koreference v Pražském závislostním korpusu. Technická zpráva 19, ÚFAL/CKL MFF UK, Praha, 2003.

Pavel Květoň. *Rule based morphological disambiguation*. Disertační práce, Matematicko-fyzikální fakulta UK, Praha, 2006.

Igor Meľčuk. *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany, 1988.

Marie Mikulová, Alevtina Bémová, Jan Hajič, Eva Hajičová, Jiří Havelka, Veronika Kolářová, Markéta Lopatková, Petr Pajas, Jarmila Panevová, Magda Razímová, Petr Sgall, Jan Štěpánek, Zdeňka Urešová, Kateřina Veselá, Zdeněk Žabokrtský a Lucie Kučová. Anotace na tektogramatické rovině Pražského závislostního korpusu. Anotátorská příručka. Technická zpráva 28, ÚFAL MFF UK, Praha, 2005.

Roman Ondruška, Jarmila Panevová a Jan Štěpánek. An Exploitation of the Prague Dependency Treebank: A Valency Case. In Kiril Simov a Petya Osenova, editoři, *Proceedings of the Workshop on Shallow Processing of Large Corpora (SproLaC 2003)*, strany 69–77, Lancaster, 2003. UCREL, Lancaster University.

Petr Pajas. Dokumentace programu TrEd. Na internetu nebo jako součást CD-ROM PDT 2.0 připravovaného k vydání Linguistic Data Consortium, Pennsylvania. URL `[http://ufal.mff.cuni.cz/pdt2.0/doc/tools/tred/]`.

Petr Pajas. Kontroly valence v PDT 2.0. Připravováno.

Petr Pajas a Jan Štěpánek. A Generic XML-Based Format for Structured Linguistic Annotation and Its Application to Prague Dependency Treebank 2.0. Technická zpráva 29, ÚFAL MFF UK, Praha, 2005.

Jarmila Panevová. On Verbal Frames in Functional Generative Description. *The Prague Bulletin of Mathematical Linguistics*, 22:3–40, 1974.

Jarmila Panevová. *Transducing Components of Functional Generative Description 1 (From Tectogrammatics to Morphemics)*, IV. svazek řady *Explizite Bechreibung der Sprache und automatische Textbearbeitung*. Matematicko-fyzikální fakulta UK, Praha, 1979.

Jarmila Panevová. *Formy a funkce ve stavbě české věty*. Academia, Praha, 1980.

Magda Razímová a Zdeněk Žabokrtský. Morphological Meanings in the Prague Dependency Treebank 2.0. In Václav Matoušek, Pavel Mautner a Tomáš Pavelka, editoři, *LNCS/Lecture Notes in Artificial Intelligence/Proceedings of Text, Speech and Dialogue*, strany 148–155. Springer Verlag, 2005.

Petr Sgall. *Generativní popis jazyka a česká deklinace*. Academia, Praha, 1967.

Petr Sgall, Eva Hajičová a Jarmila Panevová. *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. Academia, Praha, 1986.

Wojciech Skut, Brigitte Krenn, Thorsten Brants a Hans Uszkoreit. An Annotation Scheme for Free Word Order Languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP)*, strany 88–94, Washington, D. C., 1997. URL [`http://citeseer.ist.psu.edu/article/skut97annotation.html`].

Kateřina Veselá a Jiří Havelka. Anotování aktuálního členění věty v Pražském závislostním korpusu. Technická zpráva 20, ÚFAL/CKL MFF UK, Praha, 2003.

Barbora Vidová-Hladká. *Czech Language Tagging*. Disertační práce, Matematicko-fyzikální fakulta UK, Praha, 2000.

Larry Wall, Tom Christiansen a Randal L. Schwartz. *Programming Perl*. O'Reilly, Sebastopol, druhé vydání, 1996.

Fei Xia a Martha Palmer. Converting Dependency Structures to Phrase Structures. In *Proceedings of the HLT Conference 2001, First International Conference on Human Language Technology Research*, strany 61–65, San Francisco, 2001.

Dan Zeman, Jiří Hana, Hana Hanová, Jan Hajič, Barbora Hladká a Emil Jeřábek. A Manual for Morphological Annotation, 2nd edition. Technická zpráva 27, ÚFAL MFF UK, Praha, 2005.

Zdeněk Žabokrtský a Ivona Kučerová. Transforming Penn Treebank Phrase Trees into (Praguian) Tectogrammatical Dependency Trees. *The Prague Bulletin of Mathematical Linguistics*, 78:77–94, 2002.