PML-TQ pamācība

Lauma Pretkalniņa
LU MII AILab
2023-10-27

Saturs

Ievads

PML-TQ (*PML Tree Query*) ir vaicājumu valoda, kas paredzēta, lai meklēšanas sistēmai dažādos veidos varētu raksturot ("izskaidrot"), kādu koka fragmentu ir nepieciešams atrast. Katrs vaicājums sastāv no divām daļām:

1. koka struktūras specifikācija ("selektors");
2. filtri atrasto struktūru apkopošanai, saskaitīšanai un parādīšanai dažādos griezumos.

Pirmā daļa ir obligāta, otrā daļa ir neobligāta un filtrus var kombinēt vairākus secīgi. Tālāk secīgi sekos izklāsts, kā konstruēt abas šīs vaicājuma daļas, sākot ar vienkāršākiem un turpinot ar sarežģītākiem vaicājumiem. Dažviet ar norādi "***Papildinfo***" ir pievienota padziļinātā informācija, kura ir tematiski saistīta, taču visticamāk būs labāk saprotama otrajā dokumenta lasīšanas reizē.

Piezīme par terminoloģiju - visā dokumentā ir lietoti grafu termini koks, sakne, virsotne, bērns, vecāks, utt. Reizēm ir lietoti arī relatīvie termini kā virsotne

augstāk vai zemāk - šie termini ir lietoti ar pieņēmumu, ka koka tiek zīmēts tā, kā to parasti dara TrEd un LVTB LINDAT vizualizācija - ar sakni pašā augšā, saknes bērniem mazliet zemāk, bērnu bērniem vēl mazliet zemāk, utt. T.i., virsotne augstāk ir virsotne relatīvi tuvāk saknei un virsotne zemāk ir virstne tālāk no saknes.

Šī pamācība izskaidro PML-TQ būtiskākos aspektus, taču tā nepārklāj visas vaicājumu valodas iespējas. Pamācība balstīta PML-TQ specifikācijā, taču papildu informācija (t.sk. vairākas ievadprezentācijas) pieejama arī Kārļa universitātes PML-TQ dokumentācijas apkopojuma lapā.

Visas šeit aprakstītās idejas var izmantot arī meklēšanai UD kokos. Uz UD nav attiecināms skaidrojums par frāzes veida konstrukcijām, šeit dotais virsotņu tipu uzskaitījums, kā arī piemēros lietotie lauku nosaukumi un lomu vērtības. UD kokiem var strādāt arī tādas PML-TQ specifikācijas daļas, kas no šī izklāsta ir izlaistas vienkāršuma vai nesaderības ar LVTB dēļ.

Pamācību veidota ar pieņēmumu, ka lasītājs iekļautos vaicājumus izmēģinās un tam būs pieejami vaicājumu rezultāti, tapēc tie šajā dokumentā nav dublēti. Pamācība testēta ar LVTB v2.11.

1. Kā veidojas koka selektors?

Koka selektora vispārīgā sintakse vienkāršojot ir šāda:

> *virsotnes_tips [virsotni raksturojošā pazīme #1, virsotni raksturojošā pazīme #2, utt.];*

t.i., ir jāzina kāds ir meklējamo virsotņu tips un kas vēl tai ir raksturīgs.

1.1. Virsotņu tipi

Virsotnes tips var būt viens no šiem:

- `a-root` - koka sakne;
- `a-node` - jebkura atkarību virsotne;
- `a-xinfo` - informācija par xVārda frāzi;
- `a-coordinfo` - informācija par vienlīdzīgo locekļu frāzi;
- `a-pmcinfo` - informācija par pieturzīmju konstrukcijas frāzi.

Tādejādi visvienkāršākie sintaktiski validie vaicājumi būtu šādi:

- `a-root []` - atrast visas, visas koku saknes bez izņēmuma;
- `a-node []` - atrast visas, visas koku nesaknes virsotnes bez izņēmuma.

Semikols aiz vaicājuma parasti nav obligāts, ja tālāk neseko filtri. Taču vairums PML-TQ rīku to mēdz pievienot paši, tapēc arī šajā dokumentā tas tiks pievienots.

Papildinfo. Arī vaicājumi `a-xinfo []`, `a-coordinfo []` un `a-pmcinfo []` ir derīgi un lietojami vaicājumi, taču LVTB lietotās stila lapas īpatnību dēļ šajos gadījumus atrastā virsotne LINDAT serverī netiek izcelta, ērtāk ir lietot attiecīgi vaicājumus `a-node [ a-xinfo [] ]`, `a-node [ a-coordinfo [] ]` un `a-node [ a-pmcinfo [] ]`. Ko tie nozīmē? Tas būs tūlīt nākamajā nodaļā :)

1.2. Virsotņu pazīmes

Ja atceramies vispārīgo vaicājuma šablonu no 1. nodaļas sākuma, tad šobrīd mums ir zināms, kas ir *virsotnes_tips*, un ir jāsaprot kvadrātiekavu daļa:

> *virsotnes_tips [virsotni raksturojošā pazīme #1, virsotni raksturojošā pazīme #2, utt.]*

Atrast visas atkarību virsotnes vai visus xVārdus ir patīkams vingrinājums, taču praksē parasti vajag ko specifiskāku. Tam ir paredzēta iespēja kvadrātiekavās uzskaidīt dažādas pazīmes, kas sašaurina atrasto virsotņu klāstu līdz tādām, kas atbilst šīm pazīmēm. Pazīmes tiek uzskaitītas patvaļīgā secībā, atdalot ar komatu, un meklēšanas sistēma tad piedāvā tādas virsotnes, kas vienlaicīgi apmierina visas norādītās pazīmju prasības. Pazīmes var būt dažādu veidu, un tās var brīvi kombinēt:

- virsotņu lauki;
- virsotņu bērni;
- virsotņu vecāki, senči, pēcteči, kaimiņi;
- pazīmju kombinācijas ar loģikas darbību `or` (disjunkcija) un `not` (noliegums) palīdzību;
- ar kvantifikatoriem modificētas pazīmes.

Pazīmes savā starpā ir jāatdala ar komatu. Aiz pēdējās pazīmes drīkst likt komatu, taču tas nav obligāts. Komata izlaišana starp divām pazīmēm izraisīs sintakses kļūdu un vaicājums nestrādās.

1.2.1. Virsotņu lauki

Katrai virsotnei ir vairāki lauki, piemēram, loma, lemma, forma, tags, kas raksturo šo virsotni. Jebkuram laukam vaicājumā vērtību ir iespējams norādīt vai nu precīzi (*lauka_vārds* = "*vērtība*", t.i., lietojot vienādības zīmi `=` un divpēdiņas `"`), vai ierobežot ar regulāro izteiksmi (*lauka_vārds* ~ '*izteiksme*', t.i., lietojot tildi `~` un vienpēdiņas `'`). Ir iespējams arī vērtību raksturot ar to, kam tā netbilst - atkal vai nu precīzi (*lauka_vārds* != "*vērtība*") vai ar regulāro izteiksmi (*lauka_vārds* !~ '*izteiksme*'), taču regulārās izteiksmes gadījumā ir jārēķinās, ka var uzrakstīt pareizu izteiksmi nevienādībai var būt ārkārtīgi piņķerīgi. Pieejamie lauki uzskaitīti [roles+phrasetypes.xlsx 2. lapā] un virsotnes tips nosaka, tieši kāda informācija šai virsotnei būs pieejama (vērtības ir aprakstītas LVTB "exceļos"), taču daži svarīgākie ir šādi:

- `a-node` virsotnēm laukā `role` ir pieejama atkarību loma, piemēram, `subj` teikumu priekšmetiem;
- lielākajai daļai `a-node` virsotņu ir vārdforma laukā `m/form`, pamatforma laukā `m/lemma` un morfoloģiskais tags laukā `m/tag`;
- `a-xinfo`, `a-coordinfo` un `a-pmcinfo` virsotnēm precīzāks apakštips ir pieejams attiecīgi laukā `xtype`, `coordtype` vai `pmctype`;
- `a-xinfo` un retums arī `a-coordinfo` virsotnēm frāzes kopējais morfoloģiskais tags pieejams laukā `tag`;
- pētījumu replikācijas vajadzībām noder fakts, ka visām `a-node` un `a-root` virsotnēm laukā `id` ir korpusa mērogā unikāls identifikators, kas palīdz jau reiz atrastu virsotni atrast vēlreiz.

Vienkāršākais vaicājums, ko šeit aplūkot, varētu būt vaicājums `a-node [ role = "subj" ]`: tas atlasa atkarību virsotnes (jo tips `a-node`) ar lomu `subj` jeb teikuma priekšmets. Šajā vaicājumā tiek pārbaudīta tieši viena pazīme (`role`) tieši uz precīzu vērtību. Ja nepieciešams norādīt vairākas vienlaikus spēkā esošas pazīmes, tās atdala ar komatu: vaicājumā `a-node [ role = "subj", m/lemma = "mamma" ]` papildus visam iepriekš izskaidrotajam vēl ir pievienots nosacījums `m/form = "mamma"` - tas nozīmē, ka tiek meklēti nevis jebkuri teikuma priekšmeti, bet tādi, kuru pamatforma (`m/lemma` lauks) ir vārds *mamma*. Savukārt piemērs `a-node [ role = "subj", m/tag ~ '^p'`

`]` parāda vaicājumu, kurā vienas pazīmes pieļaujamās vērtības ir specificētas ar regulāro izteiksmi - taga laukam `m/tag` ir jāsākas (simbols `^` izteiksmes sākumā) ar burtu `p`, t.i., tam jābūt vietniekvārdam. Tādejādi šis vaicājums atlasa virsotnes, kas nevis patvaļīgi teikuma priekšmeti, bet tādi, kas izteikti ar vietniekvārdu. To pašu darītu arī vaicājums `a-node [ m/tag ~ '^p', role = "subj" ]`, jo pazīmju secība nav svarīga.

Mazliet sarežģītāks piemērs ar vairāku lauku specifikāciju: `a-node [ role = "subj", m/tag ~ '^p' ]`

Lielāku pētījumu ietvaros bieži gadās atlasīt noteiktus teikumus vai virsotnes pēc to identifikatoriem (`id` lauks), un tad pie tiem var vēlāk atgriezties ar vaicājumiem, kas specificē noteiktu identifikatoru, piemēram, teikumam `a-root [ id = "a-c1-p16s2" ]` vai atsevišķai virsotnei `a-node [ id = "a-c1-p16s3w4" ]`.

Papildinfo. PML-TQ lieto *perl* regulāro izteiksmju sintaksi ar izņēmumu, ka slīpsvītras `\` vietā jālieto dubultā slīpsvītra `\\`. To visu sīki un smalki aprakstīt ir ārpus šī dokumenta tvēruma (vismaz pagaidām), taču īsumā to var raksturot šādi:

- burti arbilst burtiem;
- `.` - jebkurš 1 patvaļīgs simbols;
- `[ab]` - viens simbols a vai b;
- `[a-z]` - viens simbols no intervāla: a, b, c, d... (bet `[ā-ž]` nestrādā);
- `[^abc]` - visi simboli, kas nav a, b vai c;
- `[^a-z]` - visi simboli, kas nav no intervāla a, b, c... (bet `[ā-ž]` nestrādā);
- `^` - simbolu virknes sākums;
- `$` - simbolu virknes beigas;
- `\\.` - punkts (analoģiski arī `\\(`, `\\\\`, `\\[`, `\\?`)
- `a?` - a vienu vai nevienu reizi;
- `a*` - a vienu, nevienu vai daudz reizes;
- `a+` - a vienu vai daudz reizes;
- pilnais saraksts ar regulāro izteiksmju iespējām ir piemēram, šeit (diemžēl pašam jāatceras dubultot slīpsvītras), lai gan sadaļa POSIX varētu nestrādāt, un īpaši ir ieteicams pievērst uzmanību sadaļai "Regular Expression Common Metacharacters" (jo tie ir simboli, kas noteikti nozīmē kaut citu, nevis sevi, tāpēc, lai dabūtu tiešām pašu simbolu, priekšā jāliek `\\`).

1.2.2. Virsotņu bērni

Lai gan ir jauki varēt atrast jebkuru virsotni pēc tās atkarību lomas, morfoloģiskā taga vai korpusa identifikatora, sintaktiski marķētā korpusa datos ir daudz vairāk informācijas un PMLTQ vaicājumu valodas lielākais spēks slēpjas iespējā norādīt, ka vēlamies atrast vairākas virsotnes ar specifisku savstarpējo novietojumu. Uz šo problēmu var skatīties divos soļos - vispirms sacerēt atsevišķus apakšvaicājumus tām virsotnēm, ko gribam atrast, un tad sakombinēt tos kopā vienā dižvaicājumā, kas ņem vērā šo virsotņu savstarpējo sakaru.

Pieņemsim, ka mēs vēlamies atrast ar lietvārdu izteiktus apzīmētājus, kas paskaidro kaut kādus teikuma priekšmetus. Kā zināms no iepriekšējās nodaļas, vaicājums teikuma priekšmetu atrašanai ir `a-node [ role = "subj" ]`. Vaicājums ar lietvārdu izteiktu atribūtu atrašanai ir `a-node [ m/tag ~ '^n', role = "attr" ]`. Katru no šiem vaicājumiem ir iespējams (pat ieteicams!) pirms apvienošanas pārbaudīt korpusā un pārliecināties, ka tie meklē tiešām atbilstošās virsotnes.

PML-TQ meklēšanas valoda uzskata, ka fakts, ka virsotnei ir kaut kāds bērns ar kaut kādām īpašībām, ir tāda pati kā pazīme kā fakts, ka virsotnei ir lauks ar noteiktu vērtību. Tapēc, lai iegūtu vaicājumu, kur apzīmētājs ir teikuma priekšmeta bērns, divi augšminētie vaicājumi kombinējas tādā veidā, ka bērna vaicājums kļūst vecāka vaicājumā par vēl vienu pazīmi:

```
a-node [ role = "subj", a-node [ m/tag ~ '^n', role = "attr" ] ]
```

Šādā veidā kombinējot mazākus vaicājumus, ir iespējams iegūt patvaļīgi sarežģītu struktūru vaicājumus. Piemēram, vaicājums, kas norāda divus dažādus bērnus? Vienkārši - pēc pirmā bērna specifikācijas ieliek komatu un liek galā nākamā bērna specifikāciju. Piemēram, iepriekš doto vaicājumu var papildināt šādi (PML-TQ var lietot vairākas atstarpes vienas vietā, lai padarītu sarežģītu vaicājumu lasāmāku):

```
a-node [ role = "subj",
    a-node [ m/tag ~ '^n', role = "attr" ],
    a-node [ m/tag ~ '^p', role = "attr" ] ]
```

un tagad vaicājums atradīs tādus teikuma priekšmetus, kurus paskaidro viens apzīmētājs vietniekvārds un viens apzīmētājs lietvārds jebkurā secībā, piemēram, tekstu *visa mana bērnība*. Savukārt tos pašu jaunpievienoto apakšvaicājumu `a-node [ m/tag ~ '^p', role = "attr" ]` ievietojot kā pazīmi nevis kvadrātiekavās pie "subj", bet gan vienu līmeni dziļāk - kvadrātiekavās pie "attr" mēs iegūstam atkal sintaktiski validu vaicājumu:

```
a-node [ role = "subj",
    a-node [ m/tag ~ '^n', role = "attr",
       a-node [ m/tag ~ '^p', role = "attr" ] ] ]
```

Šis vaicājums atrod tās pašas virsotnes, taču citā konfigurācijā. Tas vēl aiz vien meklē teikuma priekšmetu, ko paskaidro ar lietvārdu izteiktais apzīmētājs, taču ar vietniekvārdu izteiktajam apzīmētājam tagad ir jāpaskaidro ar lietvārdu izteiktais apzīmētājs, tapēc šoreiz starp atrodamajiem piemēriem būs *viņas rokas mājiens*.

NB par LVTB frāžu specifiku

Šeit ir piemēroti vēlreiz atgriezties pie 1.1. nodaļas papildinfo piebildes, ka, meklējot visus xVārdus (vai PMC, vai vienlīdzīgos locekļus) vaicājums `a-node [ a-xinfo [] ]` dos glītākus rezultātus nekā `a-xinfo []`. Kas te notiek? Atbilde slēpjas hibrīdo koku uzbūves niansēs. LVTB hibrīdajā gramatikā katrs xVārds, katra vienlīdzīgo locekļu konstrukcija un visi PMC, izņemot saknē esošo `sent` / `utter` ir iesaistīti atkarību kokā kā pilnvērtīgi atkarīgie. Katram xVardam, vienlīdzīgo konstrukcijai un PMC var būt gan frāzes sastāvdaļas, gan kopējie atkarīgie. LINDAT publiskajā skatā un TrEd pārskatīšanas skatā atšķirība starp frāzes veida konstrukciju sastāvdaļām un kopatkarīgajiem tiek parādīta ar šķautņu krāsu palīdzību - sastāvdaļas ir zaļas, zilas vai violetas, bet kopatkarīgie - brūni. Taču PML-TQ meklēšanas valoda neko nezina par krāsām un visu šo situāciju redz mazliet citādi.

Pirmkārt, PML-TQ redz, ka katrai atkarību virsotnei `a-node` vai nu piemīt reālie morfoloģijas lauki (tags, lemma un forma), vai nu redukcijas virsotne ar teikumā neminētā vārda raksturlielumiem (lauks `reduction`), vai arī saite uz kādu no frāzes veida konstrukcijas raksturvisotnēm - `a-xinfo`, `a-pmcinfo` vai `a-coordinfo`. No sintakses viedokļa šo saikni starp `a-node` un frāzes veida konstrukciju varētu saukt par "šī sintaktiskā loma tiek aizpildīta ar tādu un tādu frāzi", taču no PML-TQ viedokļa tā ir tāda pati vecāka un bērna attieksme kā citas, t.i., ja `a-node` nav

morfoloģijas lauku, tad viņai var būt bērns `a-xinfo`, `a-pmcinfo` vai `a-coordinfo`. Otrkārt, PML-TQ frāzes veida konstrukciju sastāvdaļas redz kā bērnus attiecīgi `a-xinfo`, `a-pmcinfo` vai `a-coordinfo` virsotnes bērnus, bet frāzes veida konstrukcijas kopējos atkarīgos - kā bērnus virsotnei tieši virs attiecīgi `a-xinfo`, `a-pmcinfo` vai `a-coordinfo`. Parasti šī virsotne ir `a-node`, taču katrā kokā ir tieši viens izņēmums - koka pašas augšējās pieturzīmju konstrukcijas (`sent` vai `utter`) vecāks ir `a-root`, tapēc meklējot PMC konstrukciju atkarīgos, ir jārēķinās, ka tie var būt bērni vai nu `a-node` vai `a-root` tipa virsotnēm (te, iespējams, noderēs nākamā nodaļā aprakstītās kaimiņu attieksmes).

Tātad, bruņojoties ar šīm detaļām, varam secināt, ka `a-node [ a-xinfo [] ]`, `a-node [ a-coordinfo [] ]` un `a-node [ a-pmcinfo [] ]` ir tas vaicājumu pamats, kas atrod kopā gan atkarību virsotni, gan frāzes veida konstrukciju, kas izpilda šo atkarību lomu, un šo pamatu tālāk mēs varam apaudzēt atkal ar visādu papildu informāciju.

Piemēram, ir iespējams atrast noteikta veida atkarību, piemēram, teikuma priekšmetu, kas ir izteikts ar noteikta vieda frāzi, piemēram, prievārda frāzi. `role = subj` ir mums jau pazīstams lauka nosacījums un, atbilstoši LVTB specifikācijas dokumentiem un iepriekš izklāstītajai loģikai, lauks `role` ir pieejams virsotnei ar tipu `a-node`. Tātad vaicājums `a-node [role = "subj", a-xinfo [] ]` atradīs teikuma priekšmetus, kas izteikti ar jebkādu xVārdu. Savukārt xVārda tipu norāda lauks `xtype`, tapēc, lai sašaurinātu vaicājumu uz prievārda frāzēm, mums jāpievieno nosacījums `xtype = "xPrep"`. Kopā sanāk šāds vaicājums:

```
a-node [ role = "subj",
    a-xinfo [ xtype = "xPrep" ] ]
```

Līdzīgā veidā ir iespējams rīkoties arī pārējām frāzes veida konstrukcijām, lai specificētu to atkarību lomas un tipus, taču ir iespējams iet arī tālāk - ņemot vērā, ka PML-TQ ir pēc būtības rekursīvi simetriska valoda, mēs varam ņemt palīgā iepriekš aprakstīto un papildināt šo vaicājumu ar papildus nosacījumiem. Piemēram, kas jādara, ja mēs vēlētos atrast no šiem ar prievārdu izteiktajiem subjektiem tikai tos, kas izteikti ar prievārdu *ap*? Vispirms jāizdomā, kā atrast prievādu *ap* - tam der vaicājums `a-node [ m/tag ~ '^s', m/lemma = "ap" ]`, kas saka, ka lemmai jābūt *ap*, bet tagam jāsākas ar *s*, kas nozīmē prievārdus. Tālāk jāsaprot, kur šo vaicājumu pievienot jau esošajam - tā, kā *ap* būs x-vārda sastāvdaļa, tad šī *ap* virsotne būs `a-xinfo` bērns. Kopā sanāk šāds vaicājums:

```
a-node [ role = "subj",
    a-xinfo [ xtype = "xPrep",
       a-node [ m/tag ~ '^s', m/lemma = "ap" ] ] ]
```

Ir iespējams veidot vaicājumus arī par frāzes kopīgajiem vai sastāvdaļu atkarīgajiem. Šeit mūs mazliet ierobežo izvēlētais piemērs - prievārdiem LVTB parasti atkarīgo nav, taču varbūt kāds atkarīgais varētu būt visai prievārda frāzei kopumā? Nebūsim prasīgi, meklēsim jebkuru atkarīgo bez papildu prasībām, t.i., `a-node []`. Frāzes kopīgie atkarīgie ir bērni virsotnei virs `a-xinfo` / `a-coordinfo` / `a-pmcinfo` virsotnes, tapēc vaicājums kopā sanāk šāds:

```
a-node [ role = "subj",
    a-xinfo [ xtype = "xPrep",
       a-node [ m/tag ~ '^s', m/lemma = "ap" ] ],
    a-node [] ]
```

un pamācības rakstīšanas brīdī šim vaicājumam LVTB ir viens vienīgs rezultāts, teikums *Šīs lietas izmeklēšanas gaitā dažādās valūtās ir arestēti ap 400 000 latu, kas ar tiesas lēmumiem atzīti par noziedzīgi iegūtu mantu.*

Tā kā LVTB prievārdiem nav atkarīgo, tad, lai nodemonstrētu, kā pie `subj` + `xPrep` + `ap` vaicājuma pielikt frāzes konstrukcijas atkarīgos, nākas padomāt par to, ka prievārda frāzei bez prievārda ir vēl viena sastāvdaļa - pamatvārds. Varbūt to tad paskaidro kāds atkarīgais? Tas nozīmētu, ka mums vaicājums jāpapildina tā, lai `a-xinfo` būtu vēl viens bērns, kas nav *ap*, tad tai virsotnei bērns būtu tas mūsu meklētais atkarīgais:

```
a-node [ role = "subj",
    a-xinfo [ xtype = "xPrep",
       a-node [ m/tag ~ '^s', m/lemma = "ap" ],
       a-node [ a-node [] ] ] ]
```

Pamācības rakstīšanas brīdī šī vaicājuma rezultāts ir tas pats arestēto latu teikums, kas iepriekšējam vaicājumam, taču to, ka atrasts ir pēc atšķirīgiem kritērijiem, parāda tas, ka meklēšanas rezultātā ir izceltas citas virsotnes.

Kopumā šādi ir iespējams veidot patvaļīgi dziļus sazarotus apakškoku aprakstus, taču sarežģītie vaicājumu valodas lietojumi var novest situācijā, kad ir grūti saprast, vai dotā konstrukcija korpusā tiešām nav atrasta, vai vaicājums nav īsti pareizs. Tapēc ir vērts katru vaicājumu veidot iteratīvi, liekot klāt pa vienai jaunai virsotnei un pārbaudot pa vidu atrasto, vai tas vēl jo projām atbilst gaidītajam.

1.2.3. Virsotņu vecāki, senči, pēcteči, kaimiņi un citas "radniecības"

Ja iepriekšējā nodaļa bija veltīta, grafu terminoloģijā runājot, virsotņu bērnu specificēšanai un tam kā grafa vecāka-bērna attieksme saistās ar sintaktiskajām attieksmēm - atkarībām un frāzēm -, tad šajā nodaļā paskatīsimies uz citām grafu teorijas un/vai novietojuma attieksmēm (lai nejūk ar sintakses attieksmēm, turpmāk PMLTQ attieksmes sauksim par relācijām). Virsotni raksturojoša pazīme var būt ne tikai vienkārši citas virsotnes vaicājums iepriekšējā nodaļā aprakstītajā veidā (*virsotnes_tips* [...]), bet arī citas virsotnes vaicājums kopā relācijas tipu formā:

> *relācijas tips virsotnes_tips [...]*

Piemēram, vaicājumā `a-node [role = "attr", parent a-node [ role = "subj" ] ]` relācijas tips ir `parent`.

PML-TQ ļauj izmantot šādas grafu un vispārējā novietojuma relācijas:

- `child` - virsotne pazīmē (kvadrātiekavās iekšā) ir bērns virsotnei, kuras pazīme tā ir (virsotnei ārpus kvadrātiekavām); ekvivalents tam, ka vispār relācijas tipu nenorāda; skaidrots iepriekšējā nodaļā.
- `parent` - apgrieztā (inversā) relācija `child` relācijai, t.i., virsotne pazīmē (kvadrātiekavās) ir vecāks un virsotne, kuras pazīme tā ir (t.i. ārpus kvadrātiekavām) ir bērns.
- `ancestor` - virsotne pazīmē ir sencis virsotnei, kuras pazīme tā ir - vecāks, vecvecāks, utt.
- `descendant` - virsotne pazīmē ir pēctecis virsotnei, kuras pazīme tā ir - bērns, mazbērns, utt; t.i. inversā relācija `ancestor`.
- `sibling` - abas virsotnes ir vienas virsotnes bērni.
- `order-follows` un `order-precedes` - virsotne pazīmē vārdu secībā ir pirms vai pēc virsotnes, kuras pazīme tā ir (viena koka ietvaros). *NB* Pamācības

rakstīšanas brīdī PML-TQ realizācijās šajā vietā ir iezagusies nekonsekvence (autoriem paziņota), tapēc meklēšana ar šīm relācijām koku serveros (ar tīmekļa saskarni vai caur TrEd) strādā otrādi nekā meklēšana ar TrEd failos savā datorā:
    - TrEd lokālajā meklēšanā ar `order-follows` atzīmētā virsotne pazīmē tiek meklēta *pēc* virsotnes, kuras pazīme tā ir, bet `order-precedes` gadījumā pazīmes virsotne ir *pirms*. Šādi ir aprakstīts arī PML-TQ specifikācijas dokumentā.
    - Koku servera meklēšanā ir otrādi - ar `order-follows` atzīmētā virsotne pazīmē tiek meklēta *pirms*, bet `order-precedes` - *pēc*.
- `same-tree-as` - abas virsotnes atrodas jebkur vienā kokā.
- `same-document-as` - abas virsotnes atrodas jebkur vienā dokumentā (failā).

Zinot `sibling` relāciju, ir iespējams vienā elegantā vaicājumā atrast visus PMC konstrukciju (t.i., `a-pmcinfo`) kopējos atkarīgos. Kā jau iepriekšējā nodaļā secinājām, `a-pmcinfo` vecāks var būt vai nu virsotne ar tipu `a-node` vai ar tipu `a-root`, kuras bērni (tā pat kā pats `a-pmcinfo`) tālāk ir PMC konstrukcijas kopējie atkarīgie. Tātad šos atkarīgos ar `a-pmcinfo` saista `sibling` relācija! Tādejādi, ja mums nepieviešamas lietvārda virsotnes (`m/tag ~ '^n'`), kas paskaidro visu PMC konstrukciju, tad neatkarīgi no novietojuma - pie saknes vai zemāk kokā - mūs glābs šāds vaicājums:

```
a-pmcinfo [ sibling a-node [ m/tag ~ '^n' ] ]
```

Tāpat kā ar iepriekēšējā nodaļā aprakstītajo vecāka bērna relāciju, arī ar šīm ir iespējams specificēt patvaļīgi sarežģītu struktūru vaicājumus, t.sk., ir iespējams vienā vaicājumā izmantot vairākas dažādas relācijas, piemēram iepriekšējo vaicājumu atkal var papildināt ar precizējumu, ka mēs gribētu, lai tiem lietvārdiem vēl ir pakārtoti atribūti (`a-node [ role = "attr" ]`)

```
a-pmcinfo [ sibling a-node [ m/tag ~ '^n',
    a-node [ role = "attr" ] ] ]
```

Papildinfo. Atsevišķos gadījumos PML-TQ specifikācija ļauj ievietot neobligātu precizējumu figūriekavās tieši aiz relācijas tipa. Šī eobligātā precizējuma sintakse vispārīgi ir `{skaitlis,skaitlis}` un tā lietojamība ir atkarīga no relācijas tipa:

- Relācijām `child`, `parent`, `same-tree-as` un `same-document-as` šis precizējums nav iespējams vispār.
- Relācijām `order-follows` un `order-precedes` tas LVTB dod nepilnīgus rezultātus, tapēc to labāk nelietot apkopojošos pētījumos.
- Relācijām `ancestor` un `descentant` precizējums figūriekavās ļauj ierobežot, cik tālu virsotnes viena no otras var būt, piemēram, ja vaicājums `a-node [role = "attr", parent a-node [ role = "subj" ] ]` atrod tos apzīmētājus, kuru tiešais vecāks ir teikuma priekšmets, bet vaicājums `a-node [role = "attr", ancestor a-node [ role = "subj" ] ]` - tos apzīmētājus, virs kuriem patvaļīgi tālu ceļā uz sakni ir teikuma priekšmets, tad vaicājums `a-node [ role = "attr", ancestor{1,2} a-node [ role = "subj" ] ]` - tikai tos apzīmētājus, kam teikuma priekšmets ir vecāks vai vecvecāks.
- Relācijai `sibling` precizējums `{,-1}` nozīmē, ka virsotne pazīmē atrodas pirms virsotnes, kuras pazīme tā ir, bet precizējums `{1,}` - ka pēc. Pārējie figūriekavu precizējumi dod LVTB nepilnīgus rezultātus, ko labāk nelietot.

1.2.4. Pazīmju kombinēšana ar loģikas darbībām

Līdz šim aprakstītais materiāls ļauj pievienot vēl un vēl arvien precīzākus papildu nosacījumus, jo komats `,` starp pazīmēm pēc būtības strādā kā loģikas *un* operators, t.i., ja virsotnei ir norādītas vairākas ar komatu atdalītas pazīmes, tad atrod tās, kam izpildās visas pazīmju prasības vienlaicīgi. Taču PML-TQ piedāvā piedāvā arī iespēju saistīt pazīmes ar loģikas *vai* operaturu (sintakse: *pazīme_* `or` *_pazīme*), un atrast virsotnes, kam izpildās, jebkura viena no prasītajām pazīmēm. Piemēram, šāds vaicājums atrod visus verbus kokā, gan reducētus, gan izteiktus:

```
a-node [ m/tag ~ '^v' or reduction ~ '^v' ]
```

Pazīmes, ko saista `or`, var būt dažādas - virsotņu lauki, bērni, utt., piemēram, šādi iepriekšējo vaicājumu var papildināt ar atļauju rezultātos iekļaut arī saliktos izteicējus:

```
a-node [ m/tag ~ '^v' or reduction ~ '^v' or a-xinfo [ xtype = "xPred" ] ]
```

Šo darbību ir iespējams kombinēt arī ar parasto komata lietojumu, piemēram, lūk, šādi nodaļas pirmais vaicājums ir sašaurināms līdz tādiem reducētiem vai nereducētiem verbiem, kas ir `spc` lomā:

```
a-node [ role = "spc", ( m/tag ~ '^v' or reduction ~ '^v' ) ]
```

Taču, ja vēlamies ar `or` saistīt, nevis atsevišķas darbības, bet to kopas, tad `or` "iekšienē" ir PMLTQ prasa lietot nevis komatu, bet `and`, t.i., ja mūs iepriekšējā vaicājumā interesētu visi reducētie verbi, bet no nereducētajiem tikai *būt*, tad to rakstītu šādi:

```
a-node [ role = "spc", ( m/tag ~ '^v' and m/lemma = "būt" or reduction ~ '^v' ) ]
```

Lietojot vairāk kā vienu pazīmes sasaistošo loģikas darbību veidu (ne tikai `,`, `and` un `or`, bet arī vēlāk aprakstīto `!`), ir jāpievērš uzmanība operāciju precedencei jeb iekavu, t.i., `()`, lietojumam. Tā pat kā matemātikā `2*3+4` nav tas pats, kas `2*(3+4)`, tāpat PMLTQ, ja nelieto iekavas, vispirms mēģina `and`, tad `or` un visbeidzot laikam `,`. Jāatzīst, ka par komatu es neesmu pilnīgi droša, jo tieši tā nodaļa oficiālajā PML-TQ dokumentācijā iztrūkst. Taču vismaz meklēšanas vide TrEd iekšienē diezgan palīdzīgi cenšās salikt iekavas, kā nu ir sapratusi, tapēc var uzrakstīt vaicājumu, palaist meklēt pirmo koku, un tad pārbaudīt, vai tas, kā vaicājums ir ticis automātiski saiekavots, atbilst plānotajai vaicājuma būtībai.

Papildinfo. PML-TQ piedāvā arī `!` operatoru (to mēģina izpildīt pirms `and`, `or`, `,`), kuru, pievienojot pirms pazīmes, tiek meklēta šīs pazīmes noliegtā pazīme, piemēram, šāds vaicājums meklētu tādas virsotnes, kas tieši nav `spc` lomā:

```
a-node [ !role = "spc", ( m/tag ~ '^v' and m/lemma = "būt" or reduction ~ '^v' ) ]
```

taču ar šī operatora lietošanu datu apkopošanas vaicājumos ir jābūt īpaši uzmanīgam par tā precīzo nozīmi. Piemēram, šis vaicājums atrod visas `spc` virsotnes, kam neviens bērns nav ar lomu `subj`

```
a-node [ role = "spc",
    ! a-node [ role = "subj"] ]
```

Savukārt šis vaicājums atrod virsotņu pārīšus, kur augšējai virsotnei ir `spc` un apakšējai virsotnei ir ne`subj` loma, taču **negarantējot, ka** tai `spc` virsotnei **nav vēl kāds cits bērns**, kuram tomēr ir tā `subj` loma.

```
a-node [ role = "spc",
    a-node [ !role = "subj"] ]
```

1.2.5. Pazīmju kvantifikatori

Reizēm ir nepieciešams atrast vairākus praktiski vienādus pazīmes eksemplārus, piemēram, teikuma priekšmetu, ar diviem apzīmētājiem. Līdzšinējais izklāsts ļauj izveidot vaicājumu `a-node [ role = "subj", a-node [ role = "attr" ], a-node [ role = "attr" ] ]`, taču šim vaicājumam ir būtisks trūkums - tas dod redundantus rezultātus, piemēram, *daži sēņu veidi* tiek atrasts divreiz. Kapēc tā notiek? - tapēc ka meklēšanas dzinējs atrod gan tādu atbilsmi, kur pirmais `attr` ir *daži* un otrais - *sēņu*, gan tādu, kur pirmais `attr` ir *sēņu*, bet otrais *daži* (kā jau minēts iepriekš, pazīmju secība vaicājumā *nenosaka* struktūru secību kokā). Šai problēmai ir vairāki risinājumi, taču viens no tiem ir izmantot pazīmju kvantifikatoru. Multiplikatoru raksta pirms relācijas tipa, ja to lieto, vai pirms virsotnes tipa, ja relācijas tips nav norādīts. T.i., vispārīgā sintakse ir:

> *kvantifikators relācijas tips virsotnes_tips [...]*

Kvantifikatori var būt:

- precīzi apjoma ierobežojumi - `2x` (tieši divreiz), `3x` (tieši trīsreiz), `4x` (tieši četrreiz), utt.;
- minimālā apjoma ierobežojumi - `2+x` (vismaz divi), `3+x` (vismaz trīs), `4+x` (vismaz četri), utt.;
- maksimālā apjoma ierobežojumi - `10-x` (ne vairāk par 10), `4-x` (ne vairāk par četri), utt.;
- noliegums `0x`, kas nozīmē, ka attiecīgajā vietā šādas pazīmes eksemplāru nav vispār.

Tas ļauj nodaļas sākuma vaicājumu par diviem atribūtiem pārrakstīt kā `a-node [ role = "subj", 2+x a-node [ role = "attr" ] ]`, ja vēlme ir atrast visus teikuma priekšmetus ar *vismaz* 2 atribūtiem vai kā `a-node [ role = "subj", 2x a-node [ role = "attr" ] ]`, ja mērķis ir tieši divi atribūti. LVTB ir iespējams pārliecināties, ka nodaļas pirmais vaicājums tiešām atdod skaitliski visvairāk rezultātu (atkārtošanās dēļ) un `2+x` vaicājums dod mazāk par pirmo, bet vairāk par `2x` vaicājumu.

Šis mehānisms kopā ar iepriekšējā nodaļā aprakstīto `same-tree-as` relāciju ļauj konstruēt tādus vaicājumus, kas atrod relatīvi mazus kokus, piemēram, ievietošanai prezentācijas slaidā. Teiksim, ka mēs vēlamies atrast skaistu piemēru ar saliktu izteicēju. Standarta vaicājums visiem saliktajiem izteicējiem ir `a-node [ a-xinfo [ xtype = "xPred" ] ]`. Ja mēs pieņemam, ka vēlamies atrast tādu koku, kurā bez šī izteicēja ir vēl ne vairāk kā 15 atkarību virsotnes, tad vaicājumu var papildināt šādi:

```
a-node [ a-xinfo [ xtype = "xPred" ],
    15-x same-tree-as a-node [] ]
```

Virsotnes, ko atrod kvantificētas pazīmes, netiek uzskatītas par vaicājuma rezultāta daļu, tapēc tās netiek iekrāsotas saskarnē, parādot rezultātu (un šis ietekmēs arī 2. nodaļā aprakstīto).

1.3. Relatīvie nosaukumi jeb mainīgo vārdi un vairākas relācijas vienā virsotņu pārī

Lai varētu ķerties pie informācijas, kas iegūta ar kādu lielisku vaicājumu, apkopošanas, ir nepieciešams izskaidrot, kā atsaukties uz vaicājumā atrastajām koku fragmentu virsotnēm. Jebkurai nekvantificētai virsotnei vaicājumā ir iespējams piešķirt relatīvo nosaukumu. Vispārīgā sintakse mainīgo nosaukumiem ir:

> *relācijas tips virsotnes_tips $virsotnes_vārds := [...]*

Virsotnes vārdam ir jāatbilst *perl* nerezervēto mainīgo vārdu sintaksei, taču šīs pamācības ietvaros uzskatīsim, ka labs virsotnes vārds ir tāds, kas:

1. sākas ar `$`;
2. turpinās ar vienu lielu vai mazu angļu alfabēta burtu;
3. tālāk satur tikai angļu alfabēta burtus, ciparus, `-` un `_`.

Relatīvos virsotņu vārdus var izmantot, lai vaicājumā nodefinētu vairākas relācijas starp tām pašām divām nozīmēm. Piemēram pieņemsim, ka mums ir vaicājums, kas atrod apzīmētājus, kas paskaidro lietvārdus, un mēs gribētu to sašaurināt tikai uz apzīmētājiem aiz paskaidrojamā vārda. Vaicājums apzīmētājam pie lietvārda ir `a-node [ m/tag ~ '^n', a-node [ role = "attr" ] ]`, un relācija, kas varētu palīdzēt mums definēt secību, ir `order-follows`, ja meklē lokālajos failos, vai `order-precedes`, ja meklē koku serverī (sk. skaidrojumu par kļūdu šīs relācijas realizācijā nodaļā par virsotņu "radniecību"). Šai problēmai neder vienkārši vaicājumi `a-node [ m/tag ~ '^n', order-follows a-node [ role = "attr" ] ]` / `a-node [ m/tag ~ '^n', order-precedes a-node [ role = "attr" ] ]`, jo tad atrod vienkārši jebkurus apzīmētājus, kas atrodas teikumā aiz lietvārda, pazaudējot lietvārda un apzīmētāja sintaktisko sakaru, tā vietā būtu nepieciešams starp abām virsotnēm norādīt vienlaikus divas relācijas. Vispirms pārrakstām sākotnējo, vispārējo, lietvārdu paskaidrojošā apzīmētāja vaicājumu, nemainot tā funkcionalitāti, bet ieviešot relatīvos virsotņu vārdu apzīmētaja virsotnei:

```
a-node [ m/tag ~ '^n',
    a-node $attribute := [ role = "attr" ] ]
```

Tagad ir iespējams uz jauno vārdu `$attribute` atsaukties vēlreiz, tāpēc var pievienot vēl vienu pazīmi, kas norāda `order-follows` vai `order-precedes` relāciju, atkarībā no tā kurā meklēšanas situacijā vaicājums tiek lietots:

```
a-node [ m/tag ~ '^n',
    a-node $attribute := [ role = "attr" ],
    order-follows $attribute ]
```

un

```
a-node [ m/tag ~ '^n',
    a-node $attribute := [ role = "attr" ],
    order-precedes $attribute ]
```

Mainīgo vārdus var ieviest vairākām vai pat visām virsotnēm, vienīgi tad tie nedrīkst būt dažādām virsotnēm vienādi. Šāds vaicājums koku serverī atradīs lietvārdus, kuriem atkarīgais apstāklis ir pirms viņiem, bet apzīmētājs pēc (*īpaši situācijās ar*

raksturu, ja raksturu ir `xPrep` lomā `attr`), bet lokālajos failos - lietvārdus, pirms kuriem ir atkarīgais apzīmētājs, bet pēc - atkarīgais apstāklis (*divas reizes nedēļā*):

```
a-node [ m/tag ~ '^n',
    a-node $attribute := [ role = "attr" ],
    a-node $adverbial := [ role = "adv" ],
    order-precedes $attribute,
    order-follows $adverbial ]
```

2. Kā apkopot un parādīt atlasīto informāciju?

Kad interesējošā koka fragmenta selektors ir uzrakstīts, reizēm izrādās, ka rezultātu ir par daudz, lai tos apskatītu, tāpēc ir nepieciešams veidot informācijas apkopojumus, kas kaut kaut kādā veidā saskaita vai izdrukā. Tāpat, ja mērķis ir veikt padziļinātu kvalitatīvo analīzi, var patraucēt tas, ka meklēšanas servera rezultāti tiek atgriezti nestabilā, t.i., dažādās reizēs atšķirīgā secībā, tapēc var gribēties vai nu tos sakārtot pēc striktiem kritērijiem, vai iegūt virsotņu unikālo identifikatoru sarakstu, kuru saglabāt citur, lai tajā atzīmētu kādas piezīmes par paveikto utt. Šādas darbības PML-TQ ļauj veikt ar dažādu filtru palīdzību. Filtrs ir komanda, ko pievieno aiz selektora, vai aiz cita filtra, un katru filtru ievada simboli `>>`:

> *virsotnes_tips [virsotni raksturojošā pazīme #1, virsotni raksturojošā pazīme #2, utt.]; >> pirmais filtrs >> nākamais filtrs >> utt.*

Iespējams, pirmais noderīgākais filtrs ir atrasto saskaitīšanas filtrs `count()`. Piemēram, meklējot saliktus izteicējus `xPred` ar vaicājumu `a-node [ a-xinfo [ xtype = "xPred" ] ]` LINDAT serverī, nākas saskarties ar iebūvēto apjoma ierobežojumu 10 000 teikumu apjomā, taču saskaitīšanas filtrs ļauj uzzināt, cik atrasto rezultātu ir:

```
a-node [ a-xinfo [ xtype = "xPred" ] ]
>> count()
```

Taču vispār filtri ļauj veikt dažādu veidu darbības. Katrs filtrs pēc būtības ir apstrādes cikls - tas saņem ieejā datus (sarakstu ar selektoram atbilstošajām vietāk korpusā vai iepriekšējā filtra rezultātus tabulā), secīgi iziet tiem cauri, aplūkojot katru ieejas datu elementu - koka fragmentu vai tablas rindiņu, un tad kaut ko ar to izdara, piemēram, kaut kur pieskaita vai sagatavo par to izdruku. Vienkāršākie filtri katru ieejas elementu pārtaisa par vienu tabulas rindiņu izejā, tapēc vispārīgi runājot filtra rezultāts ir tabula, bet ir arī tādi filtri, kas var apkopot vairākus ieejas elementus kopā un tad izejā var sanākt mazāka tabula - piemēram, iepriekšējā rindkopā dotais filtrs sakopo visus visus ieejas atradumus vienkārši vienā skaitlī.

Filtriem ir trīs daļas, taču ne visas ir vienlaicīgi obligātas:

1) priekšapstrāde (ar atslēgvārdiem `for`, `over`) ļauj ieejas datus sagrupēt pēc kaut kādiem kritērijiem, lai transformācija skatītos uz tiem kopā kā vienu vienību, nevis katru atsevišķi; 2) transformācija (ar reizēm izlaistu atslēgvārdu `give`) pārveido katru ieejas vienību par vienu rindu izejas tabulā; 3) pēcapstrāde - kārtošana (ar atslēgvārdu `sort by`)

Vispirms aplūkosim transformācijas daļu, jo to var lietot vienu pašu, bez pārējām.

2.1. Filtra transformācija

Ja filtram nav nekādas priekšapstrādes un pirms tā nav citi filtri, tad filtra transformācijas daļa nostrādā vienu reizi uz katru ar selektoru atrasto koka fragmentu. Tā ļauj izdrukāt noteiktus virsotņu laukus, saskaitīt virsotņu bērnus, utt. Vispārīgā sintakse filtra transfomācijai ir ar komatu atdalīts izejas tabulas kolonnas uzskaitījums, t.i.:

> `>>` *`virsotnes_lauks_vai_funkcija, cits_virsotnes_lauks_vai_funkcija, vēl_cits_virsotnes_lauks_vai_funkcija, utt.`*

Lai atsauktos uz kādas selektora atrastās virsotnes kādu konkrētu lauku, nepieciešams šai virsotnei piešķirt mainīgā nosaukumu (sk. 1.3. nodaļu), un tad izmantot `.`:

> *`$virsotnes_vārds.lauks`*

Piemēram, ja tiek plānots veidot plašāku pētījumu par teikuma priekšmetiem, kas ir izteikti ar prievārdisku konstrukciju, tad fakts, ka LINDAT servera meklēšanas sistēma atgriež rezultātus jauktā kārtībā var patraucēt. Tapēc, lai atgrieztos pie jau aplūkotiem piemēriem, var izveidot virsotņu identifikatoriem, lūk, šādā veidā.

1) Ņem vaicājumu (selektoru), kas atrod interesējošos teikuma priekšmetus: `a-node [ role = "subj", a-xinfo [ xtype = "xPrep" ] ]`. 2) Pievieno `a-node` virsotnē mainīgā vārdu *`teikPr`*: `a-node $teikPr := [ role = "subj", a-xinfo [ xtype = "xPrep" ] ]`. 3) Izmantojot `$teikPr` un faktu, ka identifikators glabājas laukā `id`, var uzrakstīt šādu filtru:

```
a-node $teikPr := [
  role = "subj",
  a-xinfo [ xtype = "xPrep" ] ]
>> $teikPr.id
```

Šī vaicājuma rezultāts ir tabula ar vienu kolonnu, un šajā kolonnā ir uzskaitīti visi korpusā atrastie selektoram atbilstošo `subj` virsotņu identifikatori.

Varam iet arī tālāk un rezultātu tabulu papildināt ar citām kolonnām, kurās būtu vēl kāda noderīga informācija. Piemēram, `a-xinfo` virsotnēm ir lauks `tag`, kas satur sīkāku prepozicionālās konstrukcijas marķējumu. Iekļausim rezultātā arī to! Lai to izdarītu, ir jāievieš mainīgā nosaukums arī `a-xinfo` virsotnei, sauksim to par *xVaards*, un pēc tam filtru papildina ar vēl vienu lauka norādi aiz komata:

```
a-node $teikPr := [
  role = "subj",
  a-xinfo $xVaards := [ xtype = "xPrep" ] ]
>> $teikPr.id, $xVaards.tag
```

PML-TQ piedēvā arī vairākas datu apstrādes funkcijas, ko var šajos filtros izmantot. Plašāks saraksts pieejams PML-TQ specifikācijas dokumentā, galvenās grupas ir šādas:

- matemātiskās darbības `+`, `-`, `*` utt. un teksta virkņu konkatenācijas darbība `&`;
- teksta apstrādes funkcijas kā `substr()` (izgriezt gabaliņu no garāka vārda vai taga), `match()` (pārbaudīt, vai vārds vai tags atbilst regulārajai izteiksmei, var lietot noteikta gabaliņa atrašanai un iegūšanai), `replace()` un `substitute()` (aizvietot gabaliņu no vārda ar kaut ko citu), `lower()` un `upper()` (pārvērst virkni uz mazajiem vai lielajiem burtiem), `concat()` (tā pati konkatenācija, bet ar patvaļīgu skaitu salīmējamo);

- ar koka struktūru saistītas funkcijas, piemēram, `descendants()` (virsotnes pēcteči), `depth()` (virsotnes attālums līdz saknei), `sons()` (virsotnes bērni);
- matemātiskās funkcijas datu apkopošanai `count()`, `min()`, `max()`, `sum()` utt., kas visbiežāk lietojamas, ja pirms transformācijas dati ir sagrupēti (sk. 2.3. nodaļu), taču `count()` funkciju var lietot arī vienkārši atrasto datu saskaitīšanai kā 2. nodaļas sākumā.

LINDAT severī starp funkcijas nosaukumu un atverošajām iekavām nedrīkst lietot atstarpi, tas izraisa sintakses kļūdu.

Piemēram, vaicājumam par teikuma priekšmetiem filtru var papildināt ar informāciju par prepozīcijas konstrukcijas sastāvdaļu skaitu, lūk, šādi:

```
a-node $teikPr := [
  role = "subj",
  a-xinfo $xVaards := [ xtype = "xPrep" ] ]
>> $teikPr.id, $xVaards.tag, sons($xVaards)
```

Ja vēlamies uzzināt arī prepozīcijas frāzes atkarīgo skaitu, tad jāatceras, ka PML-TQ meklēšanas sistēma arī xVārda virsotni uzskata par teikuma priekšmeta virsotnes bērnu, lai gan tas nav īsts atkarīgais, tapēc jāatņem viens, lai iegūtu īstu atkarīgo skaitu:

```
a-node $teikPr := [
  role = "subj",
  a-xinfo $xVaards := [ xtype = "xPrep" ] ]
>> $teikPr.id, $xVaards.tag, sons($xVaards), sons($teikPr)-1
```

Pamācības rakstīšanas brīdī tieši vienam no visiem atrastajiem teikuma priekšmetiem ir viens atkarīgais.

Papildinfo. Ja tiek veidota sarežģītāka filtrēšana ar vairākiem filtriem, tad pirmais filtrs atsaucas uz mainīgo vārdiem selektorā, bet katrā nākamajā filtrā, lai atsauktos uz iepriekšējā filtra rezultāta rindas šūnām lieto attiecīgi `$1` pirmajai šūnai, `$2` - otrajai šūnai, utt. Piemēram, iepriekšējo vaicājumu var papildināt ar vēl vienu filtru, kas katru tabulas rindu noformulē kā teikumu, palīgā ņemot teksta virkņu salīmēšanas (konkatenēšanas) darbību `&`:

```
a-node $teikPr := [
  role = "subj",
  a-xinfo $xVaards := [ xtype = "xPrep" ] ]
>> $teikPr.id, $xVaards.tag, sons($xVaards), sons($teikPr)-1
>> 'Virsotnei ' & $1 & ' ar tagu ' & $2 & ' ir ' & $3 & ' sastāvdaļas un ' & $4 & '
atkarīgie.'
```

2.2. Filtra pēcapstrādes solis

2.2.1. Kārtošana

Filtru rezultātu tabulas var sakārtot, aiz filtra tranformācijas pievienojot atslēgvārdu `sort by` un aiz tā uzskaitot kārtošanas kritērijus:

```
>> filtra_transformācija sort by galvenais_kritērijs, otrais_kritērijs, utt.
```

Kritērijs sastāv no norādes par filtra rezultāta tabulas kolonnu (`$1` ir pirmā kolonna, `$2` - otrā kolonna, `$3` - trešā,utt.) un neobligātas norādes par kārtošanas

secību `asc` augošai secībai vai `desc` dilstošai (nenorādot secību, kārto augoši).

Piemēram, iepriekšējā nodaļā izveidoto teikuma priekšmetu tabulu šādi varētu sakārtot vispirms pēc atkarīgo skaita dilstoši un tad vienāda atkarīgo skaita gadījumā pēc virsotnes identifikatora alfabētiski augoši:

```
a-node $teikPr := [
  role = "subj",
  a-xinfo $xVaards := [ xtype = "xPrep" ] ]
>> $teikPr.id, $xVaards.tag, sons($xVaards), sons($teikPr)-1
   sort by $4 desc, $1
```

2.2.2 Dublikātu atmešana

Pieņemsim, ka mēs vēlamies veikt nelielu pētījumu par saliktiem izteicējiem `xPred`, un ka mums ir svarīgi tikai apkopot, kas korpusā vispār ir sastopams - bez identifikatoriem un skaitiem. Pamēģināsim šādu vaicājumu - atrast xVārdus, katram atrastajam izdrukāt tagu un sakārtot pēc taga:

```
a-node $izt := [
  a-xinfo $xVaards := [ xtype = "xPred" ] ]
>> $xVaards.tag
   sort by $1
```

Šī vairāk kā 14 tūkstošu rindiņu garā tabula ir ļoti neuzskatāma, jo daudzas rindas tajā vienādas. To palīdz atrisināt atslēgvārds `distinct` ko norāda tieši pēc `>>` (vai pēc `give` gadījumos, kad tiek lietots šis atslēgvārds):

```
a-node $izt := [
  a-xinfo $xVaards := [ xtype = "xPred" ] ]
>> distinct $xVaards.tag
   sort by $1
```

Šis vaicājums dod tikai unikālās rindas, un tajā pamācības rakstīšanas brīdī ir mazāk par trim tūkstošiem rindu.

Atslēgvārds `distinct` strādā unikālo tabulas rindu atlasīšanai arī tad, ja tabulā ir vairāk par vienu kolonnu, piemēram, šādi var atlasīt visas dažādās saliktā izteicēja tagu un lomu kombinācijas:

```
a-node $izt := [
  a-xinfo $xVaards := [ xtype = "xPred" ] ]
>> distinct $xVaards.tag, $izt.role
   sort by $2, $1
```

2.3. Filtra priekšapstrādes solis - grupēšana

Ja filtra transformācija ir filtra daļa, kas strādā ar ieejas datiem 1:1 attiecībā - viens koka fragments vai viena tabulas rinda kļūst par vienu jaunu tabulas rindu, tad šajā nodaļā apskatīsim iespējas no vairākiem datu elementiem radīt vienu tabulas rindu, piemēram, kaut ko saskaitot, atlasot unikālās vērtības, vai daļu rindu atmetot.

Atslēgvārds `for` ļauj pēc noteiktiem kritērijiem sadalīt ieejas datus (koku fragmentus vai tabulas rindas) sadalīt paciņās un darbināt filtra transformāciju tā,

ka tā ar vienu piegājienu apstrādā katru šo paciņu, nevis (kā 2.1 nodaļā) katru atsevišķo ieejas datu vienību. Tādejādi grupēšana ir tas mehānisms, kas ļauj sastādīt dažādus pārskatus formā "parādība X korpusā sastopama Y reizes".

Vispārējā grupēšanas sintakes ir šāda:

> `>> for pirmais_kritērijs, otrais_kritērijs, utt. give transformācija`

Katrs kritērijs tāpat kā transformācijā var būt kāds virsotnes lauks vai kāda no PML-TQ iebūvētajām datu apstrādes funkcijām, kas pieminētas 2.1. nodaļā. Ja filtrā tiek lietota `for` grupēšana, tad transformāciju jāsāk ar atslēgvārdu `give` (citādi tas nav obligāts), un transformācijā ir jāatsaucas nevis uz koka selektorā nodefinētajiem mainīgo nosaukumiem, bet gan uz grupēšanas kritērijiem - pirmais kritērijs ir `$1`, otrais `$2`, utt.

Formālā līmenī tas aprakstās diezgan sarežģīti, tapēc izpratnei nepieciešami piemēri. Vispirms saskaitīsim, cik korpusā ir dažādu xVārdu. xVārdus vispār atrod ar selektoru `a-node [ a-xinfo [ ] ]`. Pievienojam filtru, kas tikai izvada xVārda tipu, sanāk: `a-node [ a-xinfo $xVaards := [ ] ] >> $xVaards.xtype`, un mēs iegūstam ļoti, ļoti garu tabulu ar xVārdu tipiem. Tā kā mūs interesē skaits katram no šiem tipiem, tad patiesībā šis `$xVaards.type` ir precīzi tas kritērijs, pēc kura mēs gribētu grupēt, tāpēc `for` daļa mūsu apkopojošajam vaicājumam būs `for $xVaards.xtype`. Taču ar grupēšanu vien nepietiek, vēl vajag arī kaut ko izdrukāt uz ekrāna, tapēc mums būs `give` ar divām sastāvdaļām - `$1` būs atsauce uz filtra pirmo kritēriju `$xVaards.xtype` un reāli nodrošinās iespēju mums uzzināt, ko tad mēs te skaitām, savukārt `count()` ir funkcija, kas saskaita visu noteiktā grupā:

```
a-node [
  a-xinfo $xVaards := [ ] ]
>> for $xVaards.xtype
   give $1, count()
   sort by $2 desc
```

Un visbeidzot piemērā filtram galā ir pielikts arī filtrs, kas atsaucās uz transformācijas (t.i., `give` rindiņas) otro elementu `$2`, kas ir skaits, un sakārto pēc tā dilstoši. Pamācības rakstīšanas brīdī divi populārākie xVārdu tipi ir saliktie izteicēji un prievārdiskas konstrukcijas - katrs ar vairāk kā 10 tūkstošiem gadījumu.

Vēl viens piemērs. Ar selektoru `a-node $darbV := [ m/tag ~ '^v..([^p]|p[up]).*' ]` no korpusa var iegūt darbības vārdus finītajās formās, nelokāmos un daļēji lokāmos divdabjus (t.i., visu, kas nav lokāmie divdabji). Ar grupēšanas priekšapstrādi `for $darbV.m/lemma` sadala iegūtos rezultātus čupiņās pēc lemmām; ar `give $1, count()` katrai čupiņai iedod lemmu (jo pirmais `for` kritērijs ir lemma, `$1` atsaucas uz pirmo kritēriju iepriekšējā daļā vai filtrā) un kopskaitu; ar `sort by $2 desc` sakārto dilstoši pēc skaita (jo otrā `give` kolonna ir lemma, `$2` atsaucas uz otro kritēriju iepriekšējā daļā vai filtrā):

```
a-node $darbV := [
  m/tag ~ '^v..([^p]|p[up]).*' ]
>> for $darbV.m/lemma
   give $1, count()
   sort by $2 desc
```

Grupēt var arī **pēc vairāku nosacījumu kopuma**, piemēram, vaicājumu par xVārdu tipiem ir iespējams detalizēt, apkopojot pēc xVārda tipa un taga. Lai to izdarītu, ir (1), obligāti, grupēšanas daļā jāiekļauj norāde uz tagu `$xVaards.tag`, un (2), vēlams, transformācijas daļā iekļaut norādi arī šo jauno grupēšanas elementu izvadīt uz ekrāna. Šos nosacījumus pieliekot katrai atbilstošajai daļai galā, sanāk, lūk, šādi :

```
a-node [
  a-xinfo $xVaards := [ ] ]
>> for $xVaards.xtype, $xVaards.tag
   give $1, count(), $2
   sort by $2 desc
```

Transformācijas daļā ar komatu atdalīto kritēriju uzskaitījums nosaka, kādā secībā uz ekrāna parādīsies kolonnas un kādā secībā tās būs pieejamas kārtošanas pēcapstrādei `sort` un nākamajiem filtriem. Tapēc, ja gribās, lai uz ekrāna izvadītajā tabulā skaits ir pēdējā kolonna, bet kārtošana vēl aiz vien notiek pēc skaita, tad vaicājums jāpārveido šādi:

```
a-node [
  a-xinfo $xVaards := [ ] ]
>> for $xVaards.xtype, $xVaards.tag
   give $1, $2, count()
   sort by $3 desc
```

Grupēt var arī **sarežīgākiem kritērijiem**. Piemēram, iepriekšējā vaicājuma tabula ir pārāk detalizēta, jo tagi ir ārkārtīgi daudzveidīgi, taču būtu interesanti sagrupēt pēc xVārda apakštipa jeb taga daļas kvadrātiekavās! Lai to izdarītu, par pamatu var ņemt iepriekšējo vaicājumu, taču ir kaut kā jāizgriež no `$xVaards.tag` satura kvadrātiekavu daļa. Te mums palīgā nāks PML-TQ teksta virkņu apstrādes funkcija `match()` - tai kā pirmais arguments jānorāda, kur ņemt teksta virkni, un otrais - kādu regulāro izteiksmi tur meklēt, bet rezultātā tā dod pirmo atrasto teksta fragmentu, kas atbilst regulārajai izteiksmei. Regulārā izteiksme, kas apraksta kvadrātiekavu daļu tagā, ir `'\\[.*?\\]'`. Vērts atzīmēt, ka, lai vai cik sarežģīts būtu grupēšanas (`for`) otrais kritērijs, transformācija (`give`) tāpat uz to atsaucas kā vienkārši `$2`.

```
a-node [
  a-xinfo $xVaards := [ ] ]
>> for $xVaards.xtype, match($xVaards.tag, '\\[.*?\\]')
   give $1, $2, count()
   sort by $3 desc
```

Papildinfo. Pēdējam dotajam vaicājumam ir viens trūkums - no tā vispār ir izlaisti tie xVārdi, kam taga daļas kvadrātiekavās nav. To izraisa tas, ka `match()` funkcija tagos bez kvadrātiekavām neko neatrod, un par šo neatrašanu atdod tik ārkārtīgi noraidošu atbildi, ka grupēšanā šie gadījumi tiek nogrupēti nost kā neatbilstoši. Ņemot vērāk, ka, pat lietojot vienāršu transformācijas filtru `give $xVaards.xtype, match($xVaards.tag, '\\[.*?\\]')` bez grupēšanas, gadījumi bez kvadrātiekavām tagos tiek izkļauti no rezultāta, varbūt tur ir kāda PML-TQ sistēmas kļūda vai dīvainība. Lai vai kā, šīs zudības var apiet ar aprisinājumu, kurā laukam `$xVaards.tag` vispirms pielīmē (ar `&`) galā vienkārši tukšas kvadrātiekavas `[]`, jo tad gadījumos, kad tagā nav īsto kvadrātiekavu ar apakštipu, `match()` funkcija atrasīs vismaz šīs svaigi pielīmētās:

```
a-node [
  a-xinfo $xVaards := [ ] ]
>> for $xVaards.xtype, match($xVaards.tag & '[]', '\\[.*?\\]')
   give $1, $2, count()
   sort by $3 desc
```

Un tiešām - atkarībā no iepriekšējā rezultāta šeit tabulas sākumgalā parādās `namedEnt` ar `[]` otrajā kolonnā.

Papildinfo 2. Grupēt ir iespējams arī filtra transformācijas daļā ar atslēgvārdu `over`, to lietojot kādas funkcijas, kas spēj apstrādāt grupas, piemēram, `count()`, `max()` vai `concat()` iekšienē formā "parametrs, ko apstrādāt `over` parametrs, kura vērtība ir fiksēta". Piemēram, šis vaicājums saskaita cik reižu sastopama katra teikuma priekšmetu lemma.

```
a-node $teikPr := [
  role = "subj", m/lemma ~ '.+' ]
>> distinct $teikPr.m/lemma, count($teikPr.id over $teikPr.m/lemma)
```

Sīkāk par to attiecīgajā PMLTQ dokumentācijas sadaļā.

2.4. Tik labs vaicājums, bet tomēr atrod drusku par daudz?

Atslēgvārds `filter` ļauj no no vaicājuma rezultāta izmest jebkuras rindas, kas atbilst kādam paša definētam nosacījumam. To lieto kā veselu atsevišķu filtru (t.i., sākumā jāliek `>>`) bez priekšapstrādes un pēcapstrādes soļiem. Pieņemsim, ka mēs gribam atkal pētīt teikuma priekšmetus. Lūk, šāds vaicājums drukā tabulu ar teikumu priekšmetu identifikatoriem, tagiem un lemmām, pieņemot, ka mūs interesē tikai īsti vārdi ar īstām lemmām, nevis kaut kādi saliktie teikuma priekšmeti:

```
a-node $teikPr := [
  role = "subj", m/lemma ~ '.+' ]
>> $teikPr.id, $teikPr.m/tag, $teikPr.m/lemma
```

Tabula ir gara un mēs attopamies, ka mūs nemaz neinteresē daudzie vietniekvārdi tajā. Var mēģināt izmainīt selektoru, lai tādus nemeklē, taču var arī beigās pielikt `filter` tipa filtru, lai rindas, ko negribam redzēt, vienkārši izmet! Filtrs veidojas tā: tags ir otrajā kolonā, tapēc jāfiltrē pēc `$2`, būšanu vietniekvārdam apraksta taga sākšanās ar p, jeb regulārā izteiksme `^p`, bet neatbilšanu regulārajai izteiksmei ir `!~`, kopā `$2 !~ '^p'`. Viss vaicājums kopā sanāk, lūk, šādi:

```
a-node $teikPr := [
  role = "subj", m/lemma ~ '.+' ]
>> $teikPr.id, $teikPr.m/tag, $teikPr.m/lemma
>> filter $2 !~ '^p'
```

Arī šos filtrus ir iespējams kombinēt. Piemēram, varbūt šodien mēs pētīsim tieši teikuma priekšmetus, kas sākas ar lielo burtu, bet tomēr nav arī vietniekvārdi? Lūk:

```
a-node $teikPr := [
  role = "subj", m/lemma ~ '.+' ]
>> $teikPr.id, $teikPr.m/tag, $teikPr.m/lemma
>> filter $2 !~ '^p'
>> filter $3 ~ '^[A-ZĀČĒĢĪĶĻŅŌŖŠŪŽ]'
```

Bieži `filter` funkcionalitāti var aizstāt ar sarežģītāku selektoru, kas vienkārši neatrod to, ko mums nevajadzēja, taču reizēm šajā nodaļā aprakstītais ceļš ir vienkāršāks.

Papildinfo. Kad runa ir par atlasošām vai salīdzinošām darbībām ar tukšiem laukiem, ir jāuzmanās, tie var "maģiski" parādīties un pazust no kopējiem rezultātiem. Piemēram, šis vaicājums iekļaus arī saliktos teikuma priekšmetus, 2.11. versijā kopā ir 25032 rezultātu.

```
a-node $teikPr := [
  role = "subj" ]
>> $teikPr.id, $teikPr.m/tag, $teikPr.m/lemma
```

Taču neviens no šiem diviem vaicājumiem rindas ar tukšiem tagiem neiedod:

```
a-node $teikPr := [
  role = "subj" ]
>> $teikPr.id, $teikPr.m/tag, $teikPr.m/lemma
>> filter $2 !~ '^p'
```

```
a-node $teikPr := [
  role = "subj"]
>> $teikPr.id, $teikPr.m/tag, $teikPr.m/lemma
>> filter $2 ~ '^p'
```

tapēc 2.11. versijā tie dod attiecīgi 7498 un 14878 rezultātus, kas kopā ir tikai 22376.

3. Piezīmes, padomi

3.1. Ķēpīga, bet noderīga skaitīšana

Pat tad, ja tiešais pētījuma mērķis nav saskaitīt visu atrasto kopā, ir vērts paskatīties arī, ko sacerētais vaicājums dod, kad tam galā pievieno `>> count()`, t.i., filtru, kas saskaita atrastos rezultātus. Tas palīdz gan novērtēt darba apjomu, gan izprast tādus piņķerīgus gadījumus kā 2.4. un 2.3. nodaļas papildinformācijas sadaļās redzamos gadījumus. Sevišķi lietderīgi kopapjomu paskatīties ir, ja pētījumu plānots sadalīt apakškopās. Piemēram, iedomāsimies, ka jāpēta izteicēji. Ir viegli iedomāties, ka mums būs tādi izteicēji, kas ir izteikti ar reālu vārdu (`a-node [ role = "pred", m/form ~ '.+' ]`), un ir tādi, kas izteikti ar xVārdu (`a-node [ role = "pred", a-xinfo [ xtype = "xPred" ] ]`). Taču, ja mēs paskatamies skaitus ar, lūk, šādiem vaicājumiem...

```
a-node [ role = "pred", m/form ~ '.+' ]
>> count()
```

Rezultāts: 15685 izteicēji ar reāliem vārdiem (v2.11)

```
a-node [ role = "pred", a-xinfo [ xtype = "xPred" ] ]
>> count()
```

Rezultāts: 13141 izteicēji ar xVārdiem (v2.11)

```
a-node [ role = "pred" ]
>> count()
```

Rezultāts: 31773 izteicēji kopā (v2.11)

... tad parādās, ka ne visi izteicēji ir ņemti vērā, jo ir 2419 ar vienlīdzīgiem locekļiem izteikti izteicēji, 45 ar PMC, 9 cita veida xVārdi un 474 pilnīgi reducēti izteicēji bez vārdformas (kopā ir 611 izteicēji ar aizpildītu redukcijas lauku, no kuriem 137 ir arī reāla tekstvienība tekstā) (visi skaitļi no v2.11).

3.2. Tukši un netukši virsotņu lauki

Atlasīt virsotnes, kam kāds lauks ir netukšs, var, šo lauku salīdzinot ar regulāro izteiksmi `.+` (jebkurš simbols vismaz vienu reizi). Tātad zināt, ka virsotnei vispār piemīt jebkāda vārdforma, var šādi: `a-node [ m/form ~ '.+' ]`. Ka virsotnei nav vārdformas - šādi: `a-node [ m/form !~ '.+' ]`.

3.3. Vaicājumu piemēri

Šajā nodaļā tiks apkopoti sarežģītu, taču dzīvē noderīgu vaicājumu piemēri, kurus var pētīt arī kā uzdevumus par šajā pamācībā stāstīto. Nodaļu plānots pakāpeniski papildināt.

3.3.1 SPK

Vaicājumi, kas atrod visus nereducētos, ar tekstvienību izteiktos SPK, saskaita, cik reižu katra vieda tags ir lietots, un uzskaita, ar kādām lemmām katrs tags ir lietots. Pirmais ir, iespējams, vienkāršāk iedomājams, otrais dod glītāku tabulu.

```
a-node $spc := [ role = "spc", m/form ~ '.+', reduction !~ '.+']
>> distinct $spc.m/tag, count($spc.id over $spc.m/tag), concat($spc.m/lemma & ', '
over $spc.m/tag)
   sort by $2 desc
```

```
a-node $spc := [ role = "spc", m/form ~ '.+', reduction !~ '.+']
>> for $spc.m/tag, $spc.m/lemma
   give distinct $1, count(), $2
>> distinct $1, sum($2 over $1), concat($3 & ', ' over $1)
   sort by $2 desc
```

3.3.2 Redukcijas marķējuma sadalīšana

Divi vaicājumi, kas funkcionāli dara to pašu - izveido tabulu, kur norādīti redukcijas tagi, redukcijas formas un šo pārīšu kopskaiti. Pirmais ir vienkāršāk iedomājams, otrais dod glītāku tabulu.

```
a-node $red := [ reduction ~ '.+']
>> for match($red.reduction, '^[^ (]+'), match($red.reduction, '\\(.+\\)')
   give $1, $2, count()
   sort by $3 desc
```

```
a-node $red := [ reduction ~ '.+']
>> for match($red.reduction, '^[^ (]+'), substr(match($red.reduction, '\\([^)]+'), 1)
```

```
    give $1, $2, count()
    sort by $3 desc
```