Tartu Ülikool
Matemaatika-informaatikateaduskond
Arvutiteaduse instituut

Ain Isotamm

TRANSLAATORITE TEGEMISE SÜSTEEM

Tartu 2012

Toimetaja: Ahti Peder

Käesoleva õpiku väljaandmist on toetanud Euroopa Sotsiaalfondi projekt „Informaatika ja infotehnoloogia magistriõppekavade arendamine Tartu Ülikooli arvutiteaduse instituudis koostöös Eesti infotehnoloogiaettevõtetega“.

Küljendus: Ain Isotamm

Keeletoimetus ja trükk: Tartu Ülikooli Kirjastus

Autoriõigus: Ain Isotamm, 2012

ISBN 978-9949-19-970-9

Tartu Ülikooli Kirjastus
www.tyk.ee

SAATEKS

See raamat on mõeldud eeskätt TÜ matemaatika-informaatikateaduskonna üliõpilastele õppematerjaliks ainete „Automaadid, keeled ja translaatorid“ ja „Transleerimismeetodid“ omandamise hõlbustamiseks, ent mitte ainult – võib juhtuda, et sellest on abi ka rakendajale, kes peab realiseerima mingil masinal mingi (programmeerimis)keele, s.o. kirjutama programmide (*pro* tekstide) interpretaatori või kompilaatori[1].

Selle raamatu *esimene*[2] kaasautor pidanuks olema *Mati Tombak*, kes raamatu kavandamisel arvas, et ta katab teoreetilised teemad, ent kirjutamist alustades selgus, et selleks eeldatud aeg oli liiga optimistlik. Millest on siiralt kahju: minu teooria-alane tekst tugineb nende kaante vahel Matilt õpitule (esmatutvuseks oli ta loengukursus TRÜs 1974. a. sügisel, mis oli orienteeritud kõigile huvilistele). Ja kui matemaatikateaduskonnas oli paarkümmend aastat hiljem (kevadel 1998) vaja leida lektor uue kursuse „Kompilaatorid“ (teooria rakenduslik osa pluss realisatsioon) jaoks ja kui *Tombak* kutsus mind seda tegema[3], siis pidin nõustuma, kuivõrd polnuks eetiline keelduda. Samas polnud meil “elusat” mikroarvutitel töötavat *TTS*i (*M. Tombak*u eestvedamisel olid tehtud süsteemid *Minsk-32*, *Iskra*, *Apple*’i, *CM-4*, EC-*1020* ja *PC XT* jaoks) ja et valdkonda terviklikult tunnetada ning mitte piirduda õppetöös pelgalt tahvliga, suvel enne septembrit realiseerida *TTS*, kusjuures – taas tunnetuse huvides – ei tundnud ma huvi eelmiste realisatsioonide andmestruktuuride ja neil töötavate programmide vastu (mis olnuks ka võimatu, kuivõrd “vanad tegijad”, eeskätt *M. Tombak*, *Jaanus Pöial* ja *Viljo Soo* puhkasid).

Niisiis, teoreetiliselt oli kõik paigas ning realisatsiooniks võtsin endale vabad käed. Käesolevas raamatus püüan tutvustada teoreetilisi põhitõdesid ja üht võimalikku realisatsiooni ühe võimaliku variandi – eelnevusmeetodit kasutava alt-üles-analüüsi – näitel.

Süntaksorienteeritud transleerimise meetodeid on palju[4]; ma ei tea, et neid oleks suudetud vastuvaidlematult järjestada kehtestamaks etaloni. Seetõttu (ja sellegipärast, et meie õppevahendi eesmärk pole nende variantide tutvustamine, vaid näitamine, kuidas selline lähenemine toimib) kirjeldame allpool kontekstivabadele eelnevusgrammatikatele baseeruvate (programmeerimis)keelte sõnade (*resp.* programmide) analüüsi puude automaatset genereerimist ning variante (kui keel on programmeerimiskeel) nonde puude interpreteerimise kaudu kas programmi tulemuste väljastamist või neid resultaate arvutada võimaldava koodifaili (*.exe*-fail) genereerimist. Seejuures, tänapäeval ei üritata reeglina genereerida resultaatfailina masinkoodi, vaid teksti mõnes muus, juba efektiivselt realiseeritud programmeerimiskeeles (nt. *Forth*[5] või *assembler* – nagu ka nende kaante vahel näidatakse *Intel*i assembleri jaoks), või

---

[1] Nii pidi nt. *Ahti Peder* (selle raamatu toimetaja) oma magistritöö koostamisel kirjutama translaatori omakonstrueeritud keelele *Formula*. Ja selleks kasutas ta meie süsteemi, sj. oli oma roll ka juhendajal *Mati Tombakul.* Ning *Formula* polegi programmeerimiskeel tavamõttes (vt. lisa 12).

[2] *Teise* kaasautorina tegutses üsna kaua *Nikita Šipilov*, kes käsitles oma bakalaureusetöös [Šipilov] regulaarseid grammatikaid ning regulaaravaldisi tuvastavate automaatide genereerimist ja neist asjust pidi ta meie ühises raamatus ka kirjutama mõned peatükid pluss lisad; paraku muutusid aga temagi plaanid.

[3] Töötasin ikka veel majandusteaduskonnas ja olin õpetanud majanduskϋberneetika tudengitele mingis ulatuses (kriidi ja tahvli abil) seda ainet. Mati meeskonnas olin varem kirjutanud analüüsi hõreda puu moodustaja *TTS*i *Minsk-32* versioonile ning osalesin tagasihoidlikul määral ka projektis *Modula-2 → Forth.*

[4] Vt. nt. *Jaak Henno* ([Henno], lk. 101 jj.) raamatut.

[5] *Mati Tombak*u meeskond realiseeris nii translaatori *Modula-2 → Forth.*

mõni baitkood – meie *TTS*i jaoks tegid seda oma bakalaureusetöödes 2008. aasta kevadel *Henri Lakk* (*Java* baitkood) ning 2009. a. *Einar Pius* (*MONO* ja *.NET*).

Joonis s1. Ajurünnak koos Mati Tombakuga, 1974. aasta.

Mõteldes avaloengule matemaatikatudengite ees[1], püüdsin muidugi ette valmistuda, ja peaküsimuseks enda (ning oletatavasti auditooriumi) jaoks oli: miks on selline aine üldse õppekavas. Kuivõrd translaatorite kirjutamine oli juba ammu „tehasemeeskondade" pärusmaa ning tõenäosus, et mõni minu kuulaja peaks tulevikus tõepoolest sellises meeskonnas grupijuhina tööle hakkama, ei tundunud ei siis ega tundu ka nüüd kuigi suurena. Ent mõtlesin välja mõned põhjendused.

- Millegipärast on see teema vist kõikide arvestatava taseme IT-haridust andvate kõrgkoolide õppekavades (kui uskuda *Internet*'ti).
- Mulle tundub, et teadmata, kuidas mingit programmeerimiskeelt realiseeritakse, pole võimalik toda keelt kui programmeerimisvahendit lõpuni mõista[2]. Ja paradoksaalsel moel on kerge tekkida arvamusel, et *masin* (*pro* translaator) on „tark" ja suudab ise programmeerimisvigadega (nii süntaktiliste kui ka semantilistega) hakkama saada[3].
- Programmeerimiskeelte realiseerimise kursus haakub orgaaniliselt meie instituudi teiste baaskursustega, nagu „Programmeerimine", „Algoritmid ja andmestruktuurid

[1] Olemata haritud matemaatik; olen lõpetanud TRÜ rahanduse ja krediidi erialal 1965. a. ning TPI aspirantuuri statistika üldteooria alal, milles kaitsesin ka kandidaadikraadi 1972. a. Vilniuses. Pean nentima, et ma pisut kartsin oma kuulajaskonda (hoolimata tõigast, et olen 1971. aastast olnud professionaalne programmeerija).

[2] Hea sõber *Tõnis Kelder* põhjendas, miks ta võttis vastu *Mark Nemenman*i pakkumise teha *C*-kompilaator Minski tehase mikromasinale, järgmiselt: „Tahtsin teada, kuidas *C* toimib, ja parim variant selleks oli kompilaatori kirjutamine." (vt. ka [Isotamm 2009], lk. 37 jj.).

[3] Tegelikult olid viimased „täisinformatsiooniga" illusioonivabad programmeerijad need, kes kirjutasid masinkoodi ilma operatsioonisüsteemi toeta (nagu TPI meeskond Eesti Raadio arvutuskeskuses, kus masina *Razdan-3* tehasepoolseks toeks olid ainult primitiivsed sisend-/väljund- ja tüübiteisendusfunktsioonid (vt. nt. [Isotamm 2007], lk. 47 jj.)); operatsioonisüsteem peidab programmeerija eest vähem infot kui translaator. Ent – nii opsüsteem kui ka translaator on programmeerija jaoks „mudakiht" inimese ja „raua" vahel. Mõlemad teevad elu lihtsamaks, ent hägustavad tunnetust – eriti, kui ei tea, kuidas nad on tehtud, ja millised on sealjuures madalaima taseme programmid.

(A&A)“, „Programmeerimiskeeled“ ning „Transleerimismeetodid“. Tundub, et kõik need kursused on tervikliku tunnetuse saamiseks vältimatult vajalikud.

- Edasi, praktiliselt kõik A&A andmestruktuurid ja algoritmid on transleerimiskursuses kasutusel: stringid, ahelad, vektorid, massiivid, puud, magasinid, struktuurid, tabelid, järjestamine, paisksalvestus, automaadid jne. Seega, meie kursuse võtmine peaks toetama A&A praktilist väljundit (loe: tõsiseltvõetavust).
- Tegelikult tuleb pea igal professionaalil lahendada ülesandeid, kus algandmed on mingil moel organiseeritud ning nende protsessimiseks tundub otstarbekana kirjutada „preprotsessor“ – triviaalne translaator, saamaks mugavaid andmestruktuure.
- Mõnikord on spetsiifilise ülesande lahendamiseks kasulik programmeerida translaator (alates grammatika koostamisest, näiteks [Peder, Tombak, Isotamm] ja selle abil ülesanne lahendada [Peder, Tombak]).

Ja kümmekond aastat hiljem, lugedes *David Galles*i raamatu [Galles] 1. peatüki sissejuhatust (lk. 1), oli pigem meeldiv kui üllatav tõdeda, et ka *D. G.* küsib endalt, miks peaks noor bakaeelik raiskama aega kompilaatorite kirjutamise tehnikale teadmises, et ta sellega kunagi tegelema ei hakka. Ja et põhjendusi on selles raamatus kolm.

- Professionaal kasutab kompilaatorit iga päev ja see vahend ei tohi olla tema jaoks must kast, kui ta tahab kirjutada *efektiivset koodi*.
- Ehkki väga vähesed meist kirjutavad iga päev kompilaatoreid, kirjutavad paljud meist nende juppe, näiteks *HTML* või *XML* alamhulkade interpreteerimiseks või siis muude andmestruktuuride mugavaks teisendamiseks.
- Ja kolmandaks, kompilaatori kirjutamine sunnib meid olema kodus kõigis süsteemprogrammeerimise valdkondades (automaadid, keerukad andmestruktuurid jne. jne.).

Minu *TTS*-süsteemi esimene versioon valmis *MS-DOS*i keskkonnas (kasutades graafilise liidese valmistamiseks *Stan Milam*i vabavaralist menüüde tegemise süsteemi), teine versioon – *Window*si keskkonnas – valmis paar aastat hiljem, tollase tudengi *Gunnar Kudrjavets*i tungiva soovituse ja asjaliku teeleaitamise toel.

Minu *TTS*i silmatorkavaim vajakajäämine seisneb seigas, et *lekseemiklasside* (identifikaatorid ja konstandid reaalselt, stringid ja kommentaarid potentsiaalselt) elementide tuvastaja on skannerisse „sisse programmeeritud“, selmet kasutada regulaarseid grammatikaid ja nende defineeritud sõnade (regulaaravaldiste) tuvastamiseks genereeritavat automaati. Mul oli selle asja jaoks paar hägusat ideed ja minu õnneks hakkas see teema huvitama *Nikita Šipilov*it, kes sai oma bakalaureusetöös [Šipilov] rakendatava resultaadi. Nagu juba ülalpool mainitud, kavatsesime tema tulemusi nende kaante vahel tutvustada, ent paraku see plaan ei realiseerunud. Loodame *Nikita*ga siiski, et tulevikus saame selle vajakajäämise kompenseerida.

Joonis s2. Nikita Šipilov.

Me mõtlesime Nikitaga üsna pikalt, kuidas oleks hea raamat üles ehitada nii, et teda oleks õpikuna võimalikult hõlpus kasutada. Iseenesest on tegemist lihtsa kursusega, mis on täiesti õpitav, samas on ta aga küllaltki tehniline, uute teemade juurde asudes tuleb paratamatult raamatut tagasi lapata, üle lugemaks tähistusi ja definitsioone. Ja jõudsime järeldusele, et kõik

need ülelugemist vajavad asjad võiksid olla kontsentreeritult kohe sisukorra järel, kust oleks vajaminevat kerge leida ja mis omamoodi juhatab ka raamatu sisuldasa sisse. Selle osa nimeks sai *glossaarium*[1] ning kõik, mis seal on, kirjutatakse lahti järgnevates peatükkides. Ja leidsime, et miks ka mitte teha glossaariumist omakorda kontsentraat, millega täita raamatu sisekaas. Meile tundub, et selle aine eksamiks valmistuja, kui ta tahab spikreid teha[2], saaks neid kaasi (tagumisel on TTSi skeemid) ära kasutada.

Ja veel: meie raamatus on üpris tihti vihjatud programmeerijate baasharidusse kuuluvatele tehnikatele ja/või terminitele, nagu *magasin*, *LIFO*, *paisksalvestus*, *kahendpuu* jmt, rääkimata sellest, et näidete (ja lisatud programmide) keel on *C*. Paraku jäävad nood asjad nende kaante vahel selgitamata või lahti kirjutamata − raamatu maht on paratamatult piiratud[3].

Seetõttu peaks selle raamatu materjali omandamisel olema eeliseid neil, kes on läbinud *programmeerimise* (eriti *C*), *programmeerimiskeelte* ning *algoritmide ja andmestruktuuride* kursused − ent nood *pole* eeldusaineteks; nagu kogemused näitavad, on asi suurepäraselt omandatav ka asjast huvitatud ja sootuks muu taustaga noortel kolleegidel, näiteks ajaloolastel, majandustudengitel või filoloogidel[4] − kui püüda leida võimalikult humanitaarseid tegijaid.

Ülalpool andsime mõista, et käesolev raamat on mõeldud aitama kursuse „Automaadid, keeled ja translaatorid“ omandamist, ent paraku saime (loodetavasti) hakkama ainult viimase teemaga. Keeltega on asi lihtne, aitab *Chomsky* klassifikatsiooni omandamisest, ent automaatide teema ootab omaette käsitlust.

Ja veel, raamatus on üsna palju *C*-keelseid programmitekste. Me loodame, et neist on kasu ka programmeerimis- ja *A&A*-huvilistele[5], rääkimata neist, kes võtavad *C*-keele kursust.

[1] *Webster´s Dictionary* ([Webster's], lk. 535 jj.) sedastab, et glossaarium (*glossary*) on märksõnade ja nende selgituste loetelu raamatu *lõpus*, ent ei keela ka otsesõnu seda kuhugile mujale paigutada.

[2] Mis pole lõpuni hea, spikreid võib ja tuleb teha, aga nende kasutamisest tuleks hoiduda. Hea spikker on parim vahend õppimiseks, aga kui teda tulekski kasutada, siis on ta läbimõtlematult koostatud.

[3] Siiski, materjale, vähemalt selles ulatuses, nagu meie raamatust arusaamiseks vaja, on olemas ja saadaval. Meie raamatu temaatikat, ainekäsitlust ja stiili peaksid loodetavasti sisse juhatama raamatud [Isotamm, 2007] ja [Isotamm, 2009].

[4] „Toimetusele on nende isikute nimed teada ☺“. Samas, meie *MIT*-tudengitele hoiatuseks mõeldud märkus: tegemist ei ole loomulikust intelligentsist ja laialdaste kogemuste toel läbitava kursusega, vaid asi on ainult siis lihtne, kui sellega on piisavalt tegeletud.

[5] Tihti on algoritmi esitamise parim viis (kõrgtaseme keeles) programm.

Sisukord

Glossaarium 10

Üldmõisted 10

Kontekstivaba grammatika 11

Translaatorite tegemise süsteem (TTS, CC, СПТ) 14

1. Transleerimisülesande püstitus 15

1.1. Trigol 16

1.1.1. Süntaks 16

1.1.2. Trigol-programm 17

1.1.3. Operaatori puu 18

1.1.4. Näiteprogrammi operaatorite puude moodustamine 20

1.2. Transleerimise alamülesanded 23

2. Kontekstivabad grammatikad: sissejuhatus 25

2.1. Põhimõisted 25

2.2. Edasised kokkulepped 26

2.3. Mõistete defineerimine 26

2.4. Kontekstivabad grammatikad (KVG) 27

3. Eelnevusgrammatikad 33

3.1. Eelnevusrelatsioonid 34

3.2. Eelnevuskonfliktide likvideerimine 37

3.3. Pööratavate eelnevusgrammatikate analüsaator 42

4. Analüüsi puu moodustamine 47

4.1. Analüüsi puu 47

4.2. Analüüsi puu moodustamise juhtimine 50

4.2.1. Semantikast 50

4.2.2. Puusemantika 50

5. Konteksti kasutav analüüs 57
5.1. Sõltumatu kontekst 57
5.2. Sõltuv kontekst 59
5.3. Konteksti lisamine 61
5.4. Kokkuvõte 64
6. Translaatorite Tegemise Süsteem (TTS) 65
6.1. TTSi üldine skeem 66
6.2. TTSi esimene plokk: Konstruktor 69
6.2.1. Konstruktori logi 72
6.2.2. Konstruktori genereeritud tabelid 76
6.3. TTSi teine plokk: Analüsaator 82
6.3.1. Skanner 84
6.3.2. Parser 88
7. Translaator 92
7.1. Andmestruktuurid 92
7.2. Interpretaator 93
7.3. Kompilaator 106
7.3.1. Ettevalmistustööd 107
7.3.2. Näited 112
7.3.2.1. P6.tri 113
7.3.2.2. P4.tri 115
7.3.3. Optimeerimine 117
7.3.3.1. Trigol 118
7.3.3.2. Muud lihtsamad võimalused 122
Kokkuvõtteks 125
Lisad 127
Lisa 1. Oluliselt mitmene keel 127
Lisa 2. Valik grammatikaid 128

Lisa 3. Trigoli semantikafail 136
Lisa 4. TTSi päisfail 137
Lisa 5. TTSi juhtprogramm 143
Lisa 6. Konstruktor 151
Lisa 7. Analüsaator 200
Lisa 8. Trigoli interpretaator 228
Lisa 9. Trigoli kompilaator 243
Lisa 10. Trigoli kompilaatori abivahendid ja väljundid 269
Lisa 11. Trigoli laiendamine: Viktor Karabuti versioon 275
Lisa 12. Näide sõnast keeles Form8 280
Lisa 13. Kompilaatori optimeerimine: 282
Bootstrapping (Mati Tombaku slaidid) 282
Kasutatud materjalid 284
Indeks 287

Glossaarium

Üldmõisted

Sümbol on suvaline *string* (e. sõne). String pikkusega 0 on ε.

Tähestik (*alphabet*, *V*) on mingit printsiipi järgides moodustatud *sümbolite* hulk.

Mitteterminaalne (*nonterminal*) *tähestik* (V_N) on *defineeritavate mõistete* hulk.

Terminaalne (*terminal*) *tähestik* (V_T) on nende sümbolite hulk, mida saab kasutada antud grammatikaga defineeritud keele tekstides.

Programmeerimiskeele *mõisted* (iga $A \in V_N$) on näiteks *programm*, *operaatorid*, *operaator*, *märgend*, *omistamine*, *iflause*, *massiiv*, *suunamine*, *muutuja*, *tüüp*, *aritmeetiline avaldis* jmt.

Sümbolid, millest programmi nähtav tekst koosneb (nn. *lekseemid*, $l \in V_T$)[1], on näiteks järgmised: *reservsõnad* `GOTO`, `IF`, `THEN` jmt., *eraldajad* `;` `:` `:=` `+` `-` `*` `/` `<` `>` `<=` jmt. ning *lekseemiklassid* – identifikaatorid, konstandid, stringid ja kommentaarid.

$V = V_N \cup V_T$. Tähestik *V* on terminaalse ja mitteterminaalse tähestiku ühend, $V_N \cap V_T = \varnothing$.

V_T^* on tähestiku V_T baasil moodustatavate *sõnade* (*resp.* programmide) hulk.

V^* on tähestiku *V* baasil moodustatavate sõnade hulk. Seda hulka kasutatakse (esimeses lähenduses) mõistete defineerimiseks.

Λ (kr. lambda) on *tühisõna*. $\Lambda = \{\varepsilon\}$. $V^+ = V^* \setminus \Lambda$.

***Chomsky klassid*:**

klass 0: reeglid kujul $a \rightarrow b$, $a, b \in V^*$ (*rekursiivselt loenduvad keeled*, tuvastatavad *Turing*i masinaga, siia klassi kuuluvad ka *loomulikud keeled* (eesti, vene));

klass 1: *kontekstitundlikud keeled*; reeglid kujul $aAb \rightarrow acb$, $a, c, b \in V^*$, $A \in V_N$;

klass 2: *kontekstivabad keeled*, reeglid on kujul $A \rightarrow a$, kus $a \in V^*$ ja $A \in V_N$;

klass 3: *regulaarsed keeled*, reeglid on kujul $A \rightarrow a$ ($a \in V_T^*$ ja $A \in V_N$) ja $A \rightarrow Ba$ või $A \rightarrow aB$), $a \in V_T^*$ ja $A, B \in V_N$).

[1] *Ingl*. nimetus on (vahel) *token* − mis tähistab, nagu brittidel kombeks, väga paljusid erinevaid asju.

Kontekstivaba grammatika

Kontekstivaba grammatika (*KVG*) on järjestatud nelik (V_N, V_T, P, S), kus V_N on mitteterminaalne (mõistete) tähestik, V_T on terminaalne tähestik, P on produktsioonide hulk kujul $A \rightarrow x$, kus $A \in V_N$ ja $x \in V^*$ ja S on aksioom: fikseeritud täht tähestikust V_N, mida ei saa kasutada teiste mõistete defineerimiseks ja millest lähtudes saab genereerida kõik antud keele sõnad (programmid).

Olgu $x,y \in V^*$. Ütleme, et x-st on *vahetult tuletatav* y ($x \Rightarrow y$), kui $x=uAv$, $y=uzv$ ja $A \rightarrow z \in P$, $u,z,v \in V^*$.

Sõnast x *on tuletatav* sõna y ($x \overset{*}{\Rightarrow} y$), kui leidub selline sõnade jada $z_0,z_1,\ldots,z_k$ ($z_i \in V^*$), et $x=z_0$, $z_{i-1} \Rightarrow z_i$ ja $z_k=y$ $(1 \le i \le k)$. Naturaalarvu k nimetame *derivatsiooni pikkuseks* ja jada $z_0, z_1, \ldots, z_k$ *derivatsiooniks*.

Kanooniline derivatsioon (ingl. *right derivation*) on selline, kus igal sammul asendatakse kõige parempoolsem[1] mitteterminal (kanoonilisust tähistab k topeltjoontega noole all). Niisiis, derivatsiooni abil saab aksioomist S lähtudes genereerida iga antud keele sõna.

Olgu antud *KVG* nelikuga (V_N,V_T,P,S). Tema poolt defineeritud *keel* $\mathscr{L}(G)$ on sõnade x hulk

$\mathscr{L}(G)=\{x: S \overset{*}{\underset{k}{\Rightarrow}} x \,\&\, x \in V_T^*\}$.

Süntaksi analüüsi ülesanne on püstitatud nii: on antud *KVG* ja $x \in V_T^*$. Tuleb välja selgitada, kas $x \in \mathscr{L}(G)$, ja kui kuulub, siis leida sõna x derivatsioon aksioomist S.

Olgu G kontekstivaba grammatika ja jada $z_0, z_1,\ldots,z_k$ sõna x derivatsioon aksioomist S, see tähendab, $x_0 = S$ ja $z_k = x$. Jada $z_k, z_{k-1},\ldots,z_0$ nimetatakse sõna x *analüüsiks.*

Olgu antud $S=z_0 \Rightarrow z_1 \Rightarrow \ldots \Rightarrow z_k$, kus $z_k \in V_T^*$. Ilmselt $z_k \in \mathscr{L}(G)$. Öeldakse, et iga z_i (i=0,...,k) on *lause vorm. Lause vormi baas* (alus) on selle produktsiooni parem pool, mis analüüsi järgmisel sammul asendatakse produktsiooni vasaku poolega.

Kontekstivaba grammatika G *derivatsiooni puu* on järgmiste reeglite kohaselt märgendatud tippudega puu:

- üksainus tipp märgendiga S (s.o. aksioom) on derivatsiooni puu;
- kui D on derivatsiooni puu ja N tema tipp märgendiga $A \in V_N$ ning $A \rightarrow X_1,\ldots,X_p \in P$, saame moodustada uue derivatsiooni puu D', asendades tipu N märgendi uuega $A \rightarrow X_1,\ldots,X_p \in P$ ja lisades tipule N p alluvat, millede märgendeiks paneme (vasakult paremale) $X_1,\ldots,X_p$.

Derivatsiooni puu, mille kõik märgendid kuuluvad hulka $P \cup V_T$, on *analüüsi puu.*

[1] Parempoolsusele vihjab ingl. *right*, meie „kanooniline“ seondub pigem ristiusuga, kus „parem“ tähendab üksiti „õiget“, mis „vastab õigeusule või tunnustatud eeskirjadele“ [W: kanooniline].

Eelnevusrelatsioonid on defineeritud lähtuvalt tähestiku *V* sümbolite paiknemisest produktsioonide paremates pooltes; relatsioonide „eelneb“ ja „järgneb“ leidmiseks peame defineerima mitteterminalist *A* tuletatavate sõnade algus- ja lõpusümbolite hulgad *L*(*A*) ja *R*(*A*):

$L(A)=\{X: X\in V \,\&\, [\exists\phi: A\to Xu \vee \exists\phi: A\to Bu \,\&\, X\in L(B)]\};$
$R(A)=\{X: X\in V \,\&\, [\exists\phi: A\to uX \vee \exists\phi: A\to uB \,\&\, X\in R(B)]\}.$

Eelnevusrelatsioonid („ajastub”, „eelneb” ja „järgneb”) on defineeritud järgmiselt:

$X \doteq Y \equiv \{(X,Y): \exists\phi: A\to uXYv\}$ „X ajastub Y-ga“;
$X \lessdot Y \equiv \{(X,Y): \exists\phi: A\to uXBv \,\&\, Y\in L(B)\}$ „X eelneb Y-le“;
$X \gtrdot Y \equiv \{(X,Y): [\exists\phi: A\to uBYv \,\&\, X\in R(B)]\vee[\exists\phi: A\to uBCv \,\&\, X\in R(B) \,\&\, Y\in L(C)]\}$
„X-le järgneb Y“.

Kontekstivaba grammatikat nimetatakse *eelnevusgrammatikaks* (*EG*), kui suvalise kahe sümboli vahel tähestikust *V* kehtib ülimalt üks eelnevusrelatsioon.

Kui *KVG* pole *EG*, siis saab ta *teisendada eelnevusgrammatikaks*: $G=(V_N,V_T,P,S)$ teisendatakse $G'=(V_N',V_T,P',S)$ nii, et $\mathscr{L}(G)=\mathscr{L}(G')$. Keeled on *võrdsed*, kui nad genereerivad sama sõnade hulga.

P_1-konfliktiks nimetatakse situatsiooni, kus ($X, Y \in V$): $X \doteq \gtrdot Y \vee X \lessdot \gtrdot Y \vee X \lessdot \doteq \gtrdot Y$.
P_2-konfliktiks nimetatakse situatsiooni, kus $X \lessdot \doteq Y$.

P_2-konfliktide (*X,Y*) *allikad* on produktsioonid kujul $A\to xXYy$, mis asendatakse produktsiooniga $A\to xXD$ ja lisatakse $D\to Yy$.

P_1-konfliktide (*X,Z*) *allikad* on produktsioonid kujul $A\to xXZy$ või $A\to xXYy$ nii, et $Z\in L(Y)$, mis asendatakse produktsiooniga $A\to DZy$ või $A\to DYy$ ja lisatakse $D\to xX$.

Esmalt likvideeritakse *P_2*-konfliktid ja seejärel *P_1*-konfliktid.

Eeelnevusgrammatikate analüsaatori teoreetiline alus:

Kui $S \overset{*}{\underset{k}{\Rightarrow}} X_1,X_2,\ldots,X_kAT_1,\ldots,T_n \underset{k}{\Rightarrow} X_1,X_2,\ldots,X_kZ_1,\ldots,Z_mT_1,\ldots,T_n$, kus $X_i, Z_j\in V$, $T_h\in V_T$ ($i=1,\ldots,k$, $j=1,\ldots,m$ ja $h=1,\ldots,n$), siis jada $Z_1,\ldots,Z_m$ on lause vormi baas (viimasena toimis produktsioon $A\to Z_1,\ldots,Z_m$) ja kehtivad relatsioonid

$X_i \doteq X_{i+1} \vee X_i \lessdot X_{i+1}$ $(i=1..k-1)$;
$X_k \lessdot Z_1$;
$Z_j \doteq Z_{j+1}$ $(j=1..m-1)$;
$Z_m \gtrdot T_1$.

Eelnevusgrammatikat nimetatakse *pööratavaks* (ingl. *invertible*, vn. *обратимая*), kui kõik produktsioonide paremad pooled on unikaalsed.

Kui produktsioonide hulgas on kaks produktsiooni kujul $A\to x$ ja $B\to x$, siis tekib redutseerimisprobleem: kas analüüsil tuleb asendada lause vormi baas *x* mitteterminali *A* või *B*-ga. Sel

juhul saab kasutada mitteterminalide *A* ja *B piiratud kanoonilisi kontekste BRC*(1,1): analüüsisammu määravad üheselt lause vormi baas *x*, üks sümbol vasakult (magasinis) ja üks sümbol paremalt, milleks on järjekordne terminal sisendreast (*BRC* on akronüüm ingl. terminist *Bounded Right Context*). Formaalselt:

Kui $S \overset{*}{\underset{k}{\Rightarrow}} X_1,X_2,\ldots,X_kAT_1,\ldots,T_n \underset{k}{\Rightarrow} X_1,X_2,\ldots,X_kZ_1,\ldots,Z_mT_1,\ldots,T_n \quad (T_i \in V_T)$,

siis

$X_k \lessdot A \vee X_k \doteq A$ ja $A \lessdot T_1 \vee A \doteq T_1 \vee A \gtrdot T_1$.

Moodustatakse mitteterminali *A* vasaku ja parema konteksti hulgad *LC*(*A*) ja *RC*(*A*)[1]:

$LC(A)=\{X:[X \lessdot A \vee X \doteq A] \& X \in V\}$;
$RC(A)=\{T:[A \lessdot T \vee A \doteq T \vee A \gtrdot T] \& T \in V_T\}$;
$C_{1|1}(A)=LC(A) \times RC(A)$.

Hulk $C_{1|1}(A)$ on mitteterminali *A* (1|1)-sõltumatu kanooniline kontekst (loe „üks-kriips-üks“).

Kui $A \rightarrow x$ ja $B \rightarrow x$ ning kehtib $C_{1|1}(A) \cap C_{1|1}(B) = \varnothing$, siis ütleme, et *G* on 1|1-*redutseeritav eelnevusgrammatika.*

Kui $A \rightarrow x$ ja $B \rightarrow x$ ning kehtib $C_{1|1}(A) \cap C_{1|1}(B) \neq \varnothing$, siis *G* pole 1|1-redutseeritav eelnevusgrammatika ning paljudel juhtudel aitab derivatsioonist *sõltuv kontekst.* Mitteterminali *A derivatsioonist sõltuv kontekst* $C_{1,1}(A)$ on defineeritud järgmiselt:

$C_{1,1}(A)=\{(X,T):S \overset{*}{\underset{k}{\Rightarrow}} uXATv \ \& \ Tv \in V_T^*\}$.

$C_{1,1}(A)$ arvutuseeskiri on järgmine:

$C_{1,1}(A)=\gamma_1(A) \cup \gamma_2(A) \cup \gamma_3(A) \cup \gamma_4(A)$;
$\gamma_1(A)=\{(X,T):B \rightarrow uXADv \in P \& [T=Dv \vee T \in L(D)]\}$;
$\gamma_2(A)=\{(X,T):B \rightarrow uXA \in P \& T \in RC(B)\}$;
$\gamma_3(A)=\{(X,T):B \rightarrow ADv \in P \& X \in LC(B) \& [T=Dv \vee T \in L(D)]\}$;
$\gamma_4(A)=\{(X,T):B \rightarrow A \in P \& (X,T) \in C_{1,1}(B)\}$.

Kui $A \rightarrow x$ ja $B \rightarrow x$ ning kehtib $C_{1,1}(A) \cap C_{1,1}(B) = \varnothing$, siis ütleme, et G on 1,1-redutseeritav[2] *eelnevusgrammatika.*

Kui sõltuv kontekst ei eristu, siis võib püüda konteksti juurde tuua (s.o. lisada), ja kui seegi võte ei aita, siis tuleb muuta keelt (loe: süntaksireegleid).

[1] Vasak kontekst on analüüsi ajal alati magasinis ja magasini elementide vahel pole kunagi relatsioone „puudub“ või „järgneb“. Parem kontekst on alati sisendrea momendi vasakpoolseim sümbol („aktiivne terminal“).
[2] Loe: „üks-koma-üks-redutseeritav“.

Translaatorite tegemise süsteem (TTS, CC, СПТ)

TTS koosneb kahest plokist: *Konstruktor* (mis saab ette produktsioonide hulga (ja semantika) ning genereerib *Analüsaatori*-tabelid) ja *Analüsaator*, mis kasutab *Konstruktor*i väljundit ja genereerib etteantud sõna (programmeerimiskeelte puhul programmi teksti) analüüsi puu ning (taas programmeerimiskeele puhul) lisaks vajalikud tabelid.

Kui sisendkeel on programmeerimiskeel, siis *translaatori* kirjutamiseks on kaks võimalust:

- kirjutada analüüsi puu *interpretaator*, mis läbib analüüsi puu ning leiab läbimise käigus programmi resultaadi(d);
- kirjutada *kompilaator* (masinkoodi, assemblerisse, sobivasse baitkoodi või kõrgtaseme keelde, nt. *C*), mis läbib samuti analüüsi puu ning käivitab vajadusel objektkeele kompilaatori (+linkeri) või (objektkeele) interpretaatori.

Interpretaatori ja kompilaatori kirjutamine pole *TTS*is automatiseeritud. Iga uue keele puhul tuleb üldjuhul lisada midagi olemasolevate *Trigol*-interpretaatori või -kompilaatori *C*-tekstidesse. Meie raamatu kontekstis on suvalise programmeerimiskeele kompileerimise väljunditeks olemas järgmised objektkeele prototüübid: *Intel*i *assembler*[1] ja kolm baitkoodi – *Java baitkood*, *.NET* ja *MONO*.

[1] *TTS*i esmaversioonis kasutasime assembler-translaatori ja linkerina 16-bitisele arhitektuurile orienteeritud *Borland*i vahendeid *TASM* (*Turbo Asm)* ja *TLINK* (*Turbo Linker*), mida oli võimeline kasutama ka 32-bitise arhitektuuriga masin. Ent see on juba ajalugu; tänapäevane 64-bitine arhitektuur neid rakendusi ei toeta. Sestap genereerib uue masina *Trigol*-kompilaator ühe võimaliku assembleri-keskkonna väljundit, *Microsoft*i paketi *MASM32* poolt pakutut. See pakett toetab 32-bitist lahendust 64-bitisel masinal ning on (teadmiseks *C*-huvilistele) üpris tihedalt integreeritud *C*-keelega. Lisaks, objektorienteeritud stiili kasutajatele on selles paketis nii vahendid kui ka tugi ka neile harjumuspärasele kodeerimisele. Nt. katkestuste kinnipüüdmine ja nende seostamine menüüdega. Tahtmata küll hakata toda paketti reklaamima, tuleb siiski lisada, et tal on võimas *Editor* (vististi kõiki vajaminevaid võimalusi pakkuv, graafilise liidese ja menüüsüsteemiga). Ja et tegemist on vabavaraga, siis alla laadida saab seda asja aadressilt

http://www.masm32.com/masmdl.htm

„MASM32“ on lahtikirjutatult „Macro Assembler“, 32-bitise masina koodi genereerija 64-bitise arhitektuuri jaoks. Makrovahendid annavad üpris palju lisavõimalusi, mainigem kas või keelte *C* ja *C++* toetamist.

1. Transleerimisülesande püstitus

Selles peatükis püüame teha sissejuhatuse programmeerimiskeele realiseerimise (s.o. translaatori kirjutamise) üldistesse probleemidesse, pakkumata täpseid lahendusi. Õppekeele rollis kasutame just selleks otstarbeks väljamõeldud väga lihtsat (ja nappide võimalustega) keelt *Trigol* (*Tri*viaalne Al*gol*)[1]. Ette rutates: see keel on näitekeeleks ka reaalse interpretaatori ja kompilaatori tutvustamisel meie raamatu vastavates peatükkides.

Ent alustagem mõistetest. *Translaator* (ingl. *translator* – 'tõlk') on programm, mis saab ette lähtekeeles kirjutatud programmi teksti − lähteprogrammi − ja tagab selle täitmise arvutil. Kui translaator tõlgib ühest programmeerimiskeelest teise, on tegemist *kompilaatoriga*, ja kui on tegemist lähteprogrammi keelele orienteeritud programmiga, mis teeb seda, mida lähteprogramm tahab, siis translaator töötab kui *interpretaator*. Seejuures on translaatoreid, mis võivad töötada mõlemas režiimis, näiteks *baitkoode* (vt. nt. [Isotamm, 2007] lk. 209 jj.) võib nii interpreteerida (tavaliselt) kui ka kompileerida (vajadusel).

Vastavalt programmeerimiskeelte klassifikatsioonile (vt. nt. [Isotamm 2007] lk. 10 jj.) on kompilaatoreid mitmeti liigitatud. On tavaline, et kõrgema taseme keelest tõlgitakse madalama taseme keelde (nt. assembler → masinkood), ent see pole nii ilma eranditeta. Näiteks masinkoodiaegsed paigaldajad toimisid samal, madalaimal tasemel, ja tänapäeval on tavalised translaatorid, mis tõlgivad uuest kõrgtaseme keelest vanasse, mille jaoks on olemas efektiivsed translaatorid, näiteks nii, et suvalise kõrgtaseme keele translaator genereerib *C*-teksti ja see antakse ette *C*-kompilaatorile.

Aga vana reegel kompilaatori jaoks on *kõrgem tase → madalam tase.* Ja kompilaatoreid on seda printsiipi järgides ka liigitatud:

- assembler: assemblerkood → masinkood;
- makroassembler: makroassemblerkood → assemblerkood;
- alates *FORTRAN*ist: kõrgtase → suvaline (tavaliselt madalam) tase.

Interpretaator on translaatori loomulik variant *probleemorienteeritud keelte* realiseerimiseks[2], näiteks enne infosüsteemide ja tabelarvutussüsteemide evitamist olulisel kohal olnud *aruannete generaatorite* (otsi võrgust *RPG*) või nüüdisaegsemate hüpertekstiredaktorite realiseerimisel. Sellist tüüpi translaator ei genereeri täidetavat koodifaili, vaid moodustab esmalt talle sobivad andmestruktuurid ning seejärel kasutab neid programmi täitmiseks. Samas, juba esimeste protseduurorienteeritud keelte realiseerimisel eelistati vahel interpreteerivat režiimi, eeskätt nende puhul, mis töötasid interaktiivselt (s.o. programmi käitumine sõltub dialoogist kasutajaga). Hea näide võiks olla *LISP*. Aga lisaks ka *Forth* ning *APL*. Või *Java.*

Ärme vastanda kompilaatoreid ja interpretaatoreid, need on kaks võrdväärset ja vastavalt otstarbele kasutatavat samaväärset varianti. Me käsitleme seda vastandamisprobleemi allpool ja korduvalt.

[1] Selle keele formaalne kirjeldus on lisas 2, ent eelnevusteisendused (neist tuleb juttu allpool) läbinult. Puhas *Trigol*i grammatika (originaalsüntaks) on esitatud jaotises 1.1.1.

[2] Probleemorienteeritud on nt. SQL. Meenutagem, et *interpreteerimine* on arvuti jaoks ainuvõimalik ja loomulik režiim: mis tahes aparatuurne protsessor *interpreteerib* masinkoodi.

1.1. Trigol

1.1.1. Süntaks

1950ndate keskpaiku jõudis *MIT* (*Massachusetts Institute of Technology*) teadlane, lingvist *Noam Chomsky*[1] – uurides inglise keele grammatikat – *generatiivse grammatika* kontseptsioonini. Esimeses selleteemalises publikatsioonis [Chomsky 1956] esitas ta „kolme mudeli" teooria. Aga formaalset *programmeerimiskeele* kirjeldust sel ajal polnud; oli küll juba *John W. Backus*e (*IBM*) eestvedamisel loodud *FORTRAN*, ent keele kirjeldus sarnanes pigem *assembler-keele* omaga. Viiekümnendate lõpus asus rahvusvaheline meeskond (USA ja Euroopa) välja töötama algoritmide publitseerimise keelt *Algol*, sinna meeskonda kaasati loomulikult ka *Backus*, kes oli tuttav *Chomsky* töödega, ning ta konstrueeris metakeele *kontekstivaba grammatika* kirjeldamiseks; *Algol-58* formaalse grammatika kohta tegi ta Zürichis ettekande [Backus]. Kuivõrd *Chomsky* formalismi oli teaduskirjanduses jõutud hakata nimetama *Chomsky normaalkujuks* (*CNF*, *Chomsky Normal Form*), siis hakati ka *Backus*e formalismi nimetama *Backus*e normaalkujuks (*BNF*), ent *Donald Knuth* osutas, et see pole korrektne – *Backus*e metakeel polnud normaalkuju, vaid kontekstivaba grammatika (liiase) kirjeldamise moodus. *Algol*i töörühma kuulunud *Peter Naur*[2] kirjeldas *Algol-60* süntaksi viisil, mis lihtsustas *Backus*e originaalvarianti üsna tunduvalt (publitseeriti *CACM* [3]is 1963. a. [Naur]) ning see seik võimaldas akronüümi *BNF* dešifreerida juba kui *Backus–Naur Form*.

*BNF*i (*N=Naur*) metakeel ise on lihtne: *mõisted* on eraldajate paari „<" ja „>" vahel, märki „::=" loetakse kui „on definitsiooni kohaselt" ning märk „|" eraldab sama mõiste alternatiivseid definitsioone. Kõik muud sõnad või märgid kujutavad endast kirjeldatava keele legaalseid sümboleid (reserv- ja/või võtmesõnad ning eraldajad[4]). Niisiis, *Trigol*i originaalsüntaks:

```
<programm> ::= # <operaatorid> #
<operaatorid> ::= <operaator> | <operaator>;<operaatorid>
<operaator> ::= <label>:<operaator> | <omistamine> | <iflause>
| <suunamine> | <lugemine> | <kirjutamine>
<label> ::= #i#
<omistamine> ::= <muutuja>:=<aritmav> | <muutuja:=<loogilav>
<muutuja> ::= #i#
<iflause> ::= <tingimus><operaator>
<suunamine ::=    GOTO <label>
<aritmav> ::= <yksliige>|<aritmav>+<yksliige>|<aritmav> -
<yksliige>
<yksliige> ::= <tegur>|<yksliige>*<tegur>|<yksliige>/<tegur>
<tegur> ::= #i#|#c#|(<aritmav>)
<loogilav> ::= <aritmav> < <aritmav> | <aritmav> > <aritmav> |<aritmav> <=
<aritmav> | <aritmav> >= <aritmav>| <aritmav> =/<aritmav>| <aritmav> =
<aritmav>
```

[1] Vt. nt. [Isotamm, 2007], lk. 235 jj.

[2] Taani programmeerija (snd 25.10.28, hariduselt astronoom); *BNF*i tõlgendus tema nime kaasates oli *P. N*-le vastukarva, ent *Donald Knuth*i autoriteet jäi peale. Vt. http://en.wikipedia.org/wiki/Peter_Naur 21.01.08)

[3] *Communications of the ACM* (*Assotiation for Computing Machinery*).

[4] Iga „normaalse" programmeerimiskeele programm võimaldab kasutada lisaks veel nelja tüüpi objekte: *identifikaatorid*, *konstandid*, *stringid* (*'sõned'*) ja *kommentaarid*. Nende klasside esindajate loomise reeglid esitatakse keele kirjelduses tavaliselt informaalselt, selgitava tekstina väljaspool süntaksi formaalset kirjeldust, ent viimases tuleb näidata, kus noid objekte saab (võib) kasutada. *Trigol*is on kasutatavad neist neljast kaks esimest, *identifikaatorid* (#i#) ja *konstandid* (#c#). Üldisem variant oleks nende klasside kirjeldamine regulaarsete grammatikatega.

```
<tingimus> ::= IF<loogilav>THEN
<lugemine> ::= READ #i#
<kirjutamine> ::= WRITE #i#
```

Trigol-keele mõisted on *programm*, *operaatorid*, *operaator*, *label*, *omistamine*, *iflause*, *suunamine*, *lugemine*, *kirjutamine*, *muutuja*, *aritmav*, *loogilav*, *tingimus*, *yksliige* ja *tegur*.

Sümbolid, millest programmi nähtav tekst koosneb (nn. *lekseemid*) on järgmised: *reservsõnad* `GOTO`,`IF`,`THEN`,`READ` ja `WRITE`, *eraldajad* `#` **`; : := + - * / ( ) < > <= >= /=`** ja **`=`** ning kaks *lekseemiklassi* – `#i#` ja `#c#`[1]. Lisaks tühikud, reavahetus ning tabulatsioon; neid ei tohi kasutada lekseemide sees, ent mujal kasutuspiiranguid pole, näiteks kahe lekseemi vahel *võib* (aga ei pruugi) olla suvaline arv tühikuid. Formaalne grammatika neid seiku ei kajasta.

Mainigem, et eraldaja *#* on *Trigol*is[2] nii üld- kui ka lekseemiklassi markeri rollis. Üldmarkerite vahel on kogu programmi tekst[3], lekseemimarkereid aga kasutatakse ainult programmeerimiskeele süntaksi defineerimisel; üldmarkeri rollis on see reaalselt programmi tekstis, lekseemiklassi markerina aga mitte.

1.1.2. Trigol-programm

Jätkame *Trigol*-programmiga *P6.tri*[4], mis sisestab naturaalarvu *n* klaviatuurilt, arvutab ta faktoriaali $F(n) \equiv n! = 1 \times 2 \times \ldots \times n\text{-}1 \times n$ ning väljastab selle väärtuse ekraanile:

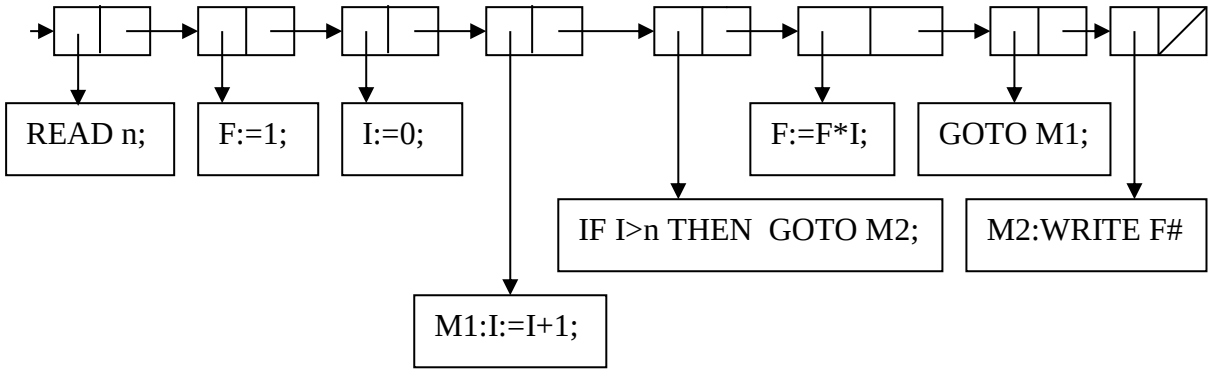
```
# READ n ; F := 1 ; I := 0 ;
M1 : I := I + 1 ;
IF I > n THEN GOTO M2 ;
F := F * I ;
GOTO M1 ;
M2 : WRITE F #
```

Meie[5] peame kirjutama translaatori, mis garanteerib *Trigol*-programmi lahendamise. Küsimus on, kuidas. Esimene idee võiks olla, et transleerime operaatorhaaval, neid on lihtne tuvastada – operaatorid lõpevad sümboliga „;“ (erandlikult viimane sümboliga „#“). Andmestruktuurina tundub loomulikuna *ahel*, mille lüli formaat on järgmine:

viit operaatorile	järgmine

P6.tri jaoks oleks see ahel kujutatud joonisel 1.1.2.a.

[1] Tavaliselt kirjeldatakse lekseemiklasside moodustamise reegleid verbaalselt. *Trigol*i identifikaator on suur- või väiketähega algav ja soovi korral pikem nimi, milles võib kasutada nii tähti kui ka numbreid ja ei midagi muud ning konstant võib olla ainult märgita täisarv.
[2] Ja üldmarkerina ka abstraktsetes garammatikates, see on vajalik tagamaks lihtsat analüsaatorit.
[3] Programmeerimiskeele puhul võib programmeerija neid trelle eirata, vajadusel lisab nad Skanner.
[4] Järgime üldaktsepteeritavat stiili, kus programmi nime laiend (meil *.tri*) viitab ilmutatult keelele, milles ta on kirjutatud; vrd. nt. *.c* (*C gcc* jaoks), .C (*C++*, *gcc*), *.asm* (*assembler*) jne.
[5] *Meie* oleme nende ridade kirjutaja ja nende ridade lugeja, kui teeme näo, et *meil* pole aimugi, kuidas selle tööga hakkama võiks saada.

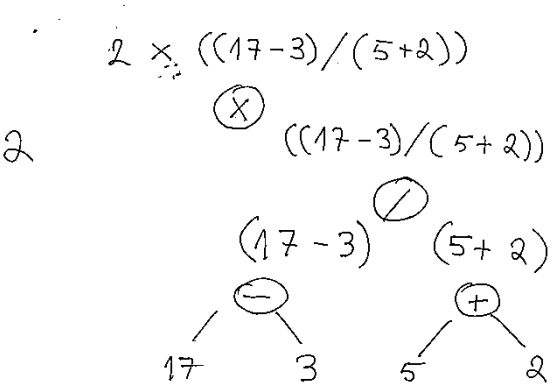


Joonis 1.1.2.a. *P6* esmane puu.

1.1.3. Operaatori puu

Jätkakem oma teadlikult naiivset mõttekäiku endiselt teksti tasemel. Usutavasti on lugejal teada kas juba läbitud ainekursustest (kas või *algoritmid ja andmestruktuurid* või *programmeerimiskeeled*) või muudest allikatest, et operaatorite töötlemiseks on ratsionaalne nende kujutamine *kahendpuude* abil; tuntuim näide on *aritmeetilise avaldise* viimine kahendpuu kujule ning tolle läbimine *LIFO*-magasini abil avaldise väärtuse leidmiseks (vt. nt. [Isotamm, 2007], lk. 226–231 jj.). Tuletagem puu konstrueerimise mõttekäiku meelde: ehitatava puu juure märgendiks saab kõige viimasena sooritatava tehte märk ning avaldise tekstist saame kaks uut stringi: esimese moodustavad sümbolid kuni tolle tehte märgini ja teise – märgist järgmisest sümbolist kuni avaldise lõpuni. Esimese poole baasil moodustame ehitatava kahendpuu juure vasaku alampuu ja teise poole baasil – parema alampuu ning (kui seda mõttekäiku nimetame algoritmiks) rakendame sama algoritmi rekursiivselt iga alampuu jaoks seni, kuni puu lehtedes on ainult operandid (muutujad või konstandid, meil klasside `#i#` ja `#c#` esindajad).

Näiteks, kui avaldiseks on *konstantavaldis* „2*((17-3)/(5+2))“, siis selle lahutamine alamstringideks toimub, nagu on esitatud joonisel 1.1.3.a.

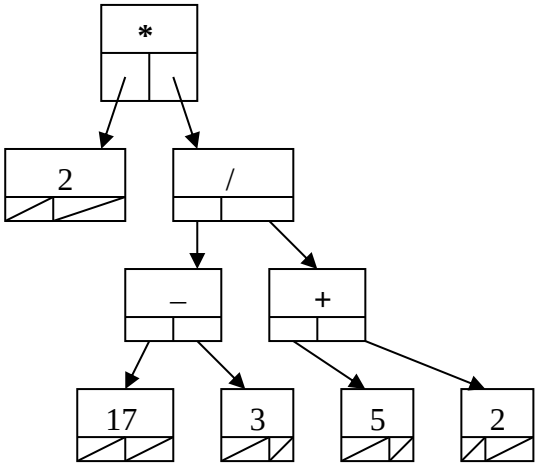


Joonis 1.1.3.a. Aritmeetilise avaldise käsitsitehtud kahendpuu.

Kahendpuu tipu kirjeldus meie näidete jaoks on *C*-keeles järgmine:

```
struct tipp{
      char label[10];//võtmesõna, tehtemärk või ident. või konst.
      struct tipp *vasak;   // viit vasakule alluvale
      struct tipp *parem;   // viit paremale alluvale
      };
```

Kahendpuu kui andmestruktuur tuleb selline:



Joonis 1.1.3.b. Aritmeetilise konstantavaldise (joonisel 1.1.3.a) kahendpuu.

Seda kahendpuud võime kasutada nii avaldise väärtuse arvutamiseks (*interpreteeriv režiim*) kui ka selleks vajalike käskude *kompileerimiseks.* Mõlemal juhul läbitakse puu lõppjärjekorras (*postorder*), lehtedes olevad operandid paigutatakse magasini, mitterippuvate tippude tehted sooritatakse magasini tipmiste elementide vahel[1] ning tehte resultaat kirjutatakse operandide asemele magasini tippu (interpretaator teeb seda „tegelikult“, kompilaator genereerib vastavad käsud). Näiteks, *interpretaatori* magasini seise illustreerib joonis 1.1.3.c.

			2			
	3		5	7		
	17	14	14	14	2	
2	2	2	2	2	2	4
push 2	push 17 3	– 17 3	push 5 2	+ 5 2	/ 14 7	* 2 2

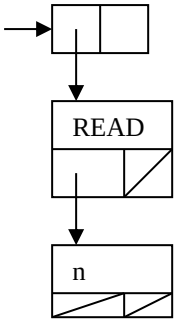
Joonis 1.1.3.c. Interpreteerimisaegse *LIFO*-magasini seisud.

[1] Tuletame meelde, et teine operand saadakse magasini tipust ja esimeseks on talle järgnev element: lahutamine ja jagamine pole kommutatiivsed tehted (vt. „– 3 17“).

1.1.4. Näiteprogrammi operaatorite puude moodustamine

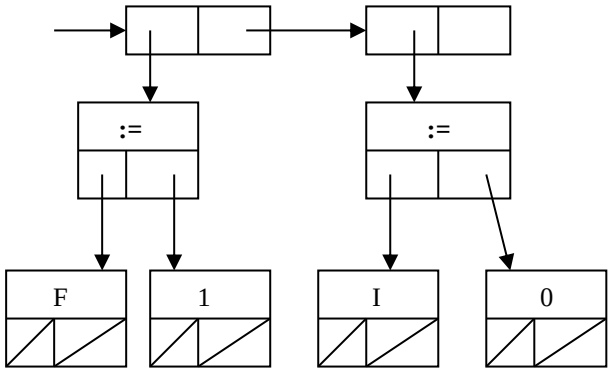
Julgustatult teadmisest, kui hõlpsasti on realiseeritav aritmeetilise avaldise kahendpuu, tundub perspektiivikana ehitada kõigile *Trigol*-operaatorite tekstidele peale kahendpuud – lootuses, et neid on mugav transleerimise käigus kasutada. Tugineme puu tegemisel *Trigol*-keele süntaksile ja lekseemiklasside #i# ja #c# esindajate kirjeldustele[1]. Seejuures kasutame nii *Trigol*i süntaksit kui ka lekseeme.

Niisiis, esimene operaator on `READ n.` See algab lekseemiga `READ;` ainus süntaksireegel, mille definitsiooniosa nii algab, on `<lugemine>::=READ #i#.` Ja saame oma ahela esimeses lülis viidatud operaatori asendada alampuuga (joonis 1.1.4.a, käsitledes lekseemi `READ` unaarse tehtena).



Joonis 1.1.4.a. Esimene operaator.

Järgmine operaator on `F:=1`. Definitsioonidest leiame, et omistamistehe **`:=`** on kasutusel omistamisoperaatoris; tegemist on binaarse tehtega. Ja et järgmine operaator on samuti lihtomistamine, siis paigutame sellegi puu samale joonisele (joonis 1.1.4.b).

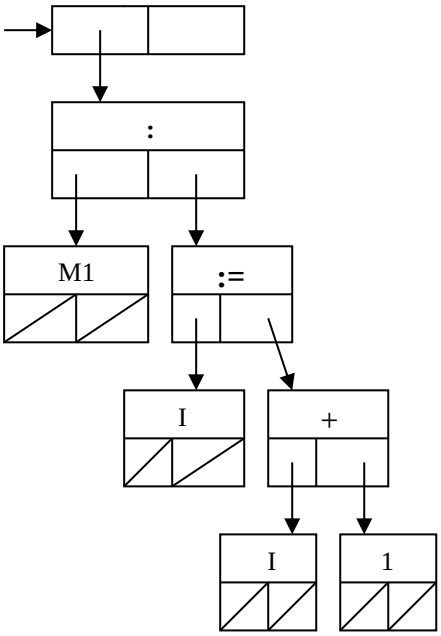


Joonis 1.1.4.b. Teine ja kolmas operaator.

Neljas operaator on pisut keerulisem: `M1:I:=I+1.` Teine lekseem on **`:`**. Süntaksist leiame, et operaatori defineerimise esimene variant on `<label>:<operaator>` (vältimaks täpitähti nimetasime *märgendit* võõrkeele abil). Puus võiksime kujutada *label*it binaarse tehtena,

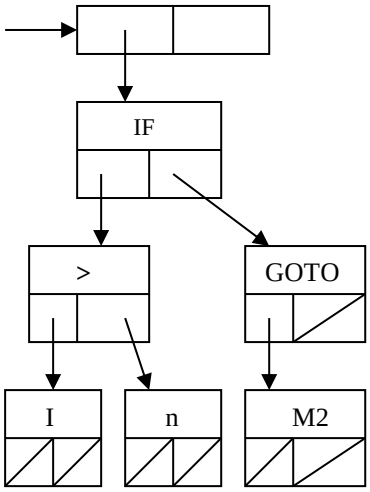
[1] Kui järjekordne tuvastatav tekstilõik pole ei ilmutatud kujul esitatud reservsõna ega ka eraldaja, siis olgem heausklikud, et ta on kas korrektne identifikaator (kui algab tähega) või konstant (kui algab numbriga). Ja lootkem, et pole tehtud süntaksivigu á la `READ WRITE := IF`. Selliste asjade vastu on meie naiivne translaator relvitu. Ja reaalne translaator peab nendega hakkama saama.

mille esimene operand on märgend ja teine – märgendatud operaatori alampuu. Sel juhul saame joonisel 1.1.4.c kujutatud pildi.



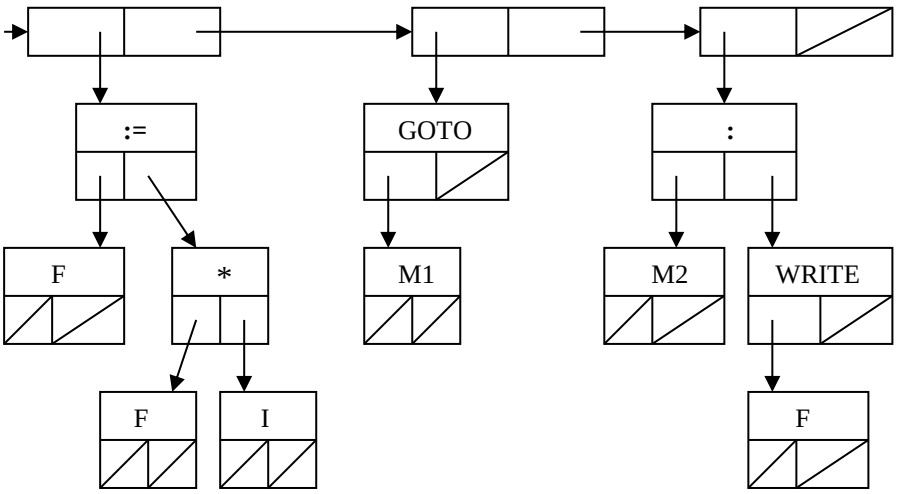
Joonis 1.1.4.c. Neljas operaator.

Viies operaator `IF I>n THEN GOTO M2` on selle programmi pikim ja keerulisim. Reservsõna `THEN` on ilmselgelt „müra“ (vt. nt. [Isotamm 2007], lk. 223 jj.), muud komponendid mitte. Võiksime oma loodava translaatori programmeerida nii, et *iflause* alampuu juur on *IF*-tipp, mille vasak alampuu esitab tingimuse ja parem alampuu väljendab operaatorit, mis täidetakse, kui vasaku alampuu tingimuse väärtus on *tõene.*



Joonis 1.1.4.d. Viies operaator.

Suunamine (`GOTO`) on unaarne tehe. Kui otsustame nii, siis selle operaatori alampuu on kujutatud joonisel 1.1.4.d. Ja et ülejäänud kolm operaatorit enam midagi põhimõtteliselt uut ei paku, siis esitame nende alampuud kõik ühel ja samal pildil (joonis 1.1.4.e).



Joonis 1.1.4.e. Viimased operaatorid.

Püüame tehtut pisut kommenteerida. Esiteks, kasutades *lõputunnust* (**;** või **#**), jagasime programmi teksti operaatoritele vastavateks alamstringideks ning iga operaatori teksti püüdsime omakorda „jupitada“, näiteks neljanda operaatori `M1:I:=I+1` teksti jagasime esiteks kaheks – `M1 ja I:=I+1` ja seejärel teise poole stringideks „`I`“ **:=** „`I+1`“**.** Mida viimasega teha, dikteeris tuvastatud tehtemärk „**+**“.

Nentigem (ehkki me pole siiani midagi rääkinud *strateegiatest*), et programmi puu ehitamisel lähtusime *ülalt-alla-strateegiast*: püüdsime tuvastada ja puustruktuurile teisendada keele süntaksiga fikseeritud alamstringe nii, et igal järgmisel sammul tegeleksime eelmise sammu alamstringi uute osistega (kuni enam ositada polnud võimalik).

Ja veel: loodetavasti märkasid lugejad, kui „vabalt“ me otsustasime, kuidas programmi puu struktuuri kujundada. Põhjus on lihtne: kui me peame kirjutama translaatori, siis ei tohiks kedagi huvitada, kuidas me seda teeme – tellijat huvitab vaid, et translaator toimib ja teeb seda võimalikult efektiivselt (s.o. ruttu ja väheste ressurssidega). *Ergo*, andmestruktuurid genereerime ise ja just nii, et neid oleks interpretaatorit või kompilaatorit kirjutades võimalikult lihtne kasutada meil endil.

Kuidas asi toimib? Lugeja võiks läbi mängida suvalise operaatori alampuu *LIFO*-magasini abil *á la* operaatori puu (vt. joonis 1.1.3).

1.2. Transleerimise alamülesanded

Eelmises jaotises esitasime ühe võimaliku variandi *Trigol*-näiteprogrammi *P6.tri* edasiseks tõlkimiseks sobivana tunduvale kujule teisendamisest. Mõned märkused.

- Meil tuleb programmeerida *tuvastaja*[1], s.o. programm, mis on võimeline *.tri*-programmi teksti jagama *operaatoriteks* jt. väiksemateks süntaktilisteks ühikuteks kuni *lekseemide* tuvastamiseni, ja meil tuleb välja mõtelda sobivad andmestruktuurid selle töö resultaatide jaoks. Eelmises peatükis andsime mõista, et too struktuur võiks olla *puu*, ent võib arvata, et peaksime moodustama ka lekseemiklasside esindajate tabelid, *Trigol*i jaoks siis – *identifikaatorite* tabeli ja *konstantide* tabeli. Üldiselt, on teada, et tuvastajaid on mõistlik esitada *automaatidena*. Mainigem, et tuvastaja on võimeline vastama küsimusele, kas etteantud tekst on korrektne (s.o. reeglipärane) või ei. Tuvastaja ülesanne pole etteantud teksti põhjal andmestruktuuride (loe: puu ja tabelite) genereerimine[2].
- Me peaksime tutvuma erinevate võimalustega *programmi puu* ehitamiseks (s.o. tuvastaja pealisehituse mehhanismi valikuks); sobiva variandi valik sõltub eeskätt sellest, kas meie ülesanne on ühe konkreetse programmeerimiskeele efektiivne realiseerimine või on ette näha, et translaatorite tegemine võib osutuda meie põhitööks. Esimesel juhul võime kõik vajaliku n-ö. sisse programmeerida ja teha translaatorit võimalikult hoolikalt, et tulemus oleks efektiivne, s.o. kirjutajasõbralik ning ressursisäästlik (ressursi all mõtleme masinaaega ja vajaliku mälu mahtu). Teisel juhul teeme oma töö lihtsamaks, kui leiame meie ülesannete klassi jaoks sobiva *translaatorite tegemise süsteemi* (*TTS*, ingl. *Compiler Compilers*, *CC*, ja vn. *Система Построения Трансляторов, СПТ*)[3]. Meie õppevahend üritab näidata, kuidas sellist *TTS*i on võimalik kirjutada (seejuures fikseerime strateegia – alt üles – ja sellele kuulekate keelte alamklassi (kontekstivabad *BRC*-analüüsitavad eelnevusgrammatikad).
- Meie esimene (ja olulisim) eesmärk on teisendada lähteprogrammi tekst *puustruktuuriks*. Parafraseerides *Archimedes*t, kes olevat öelnud, et antagu talle toetuspunkt ja kang, siis ta liigutab Maakera, ütlevad programmeerijad, et „andke mulle puu ja ma teen mis iganes“. Etteruttavalt: on tehnikaid, mis garanteerivad oma (alam)klassi grammatikatel põhinevate programmeerimiskeelte programmide puude genereerimise (kusjuures see protsess on üldjuhul juhitav), ilma et *TTS*i kasutaja peaks midagigi programmeerima. Tema tahab saada programmi puud, ja *TTS* annab selle.
- *Kui* translaatori programmeerijal on garanteeritult käes adekvaatne programmi puu, siis teoreetiliselt on tal valida kahe variandi vahel: kas kirjutada *interpretaator* või *kompilaator* (s.o. kas interpreteerida programmi analüüsi puud ja väljastada resultaadid, või genereerida programmi tööks vajalikud käsud ja *.exe*-fail, mille käivitamine väljastab resultaadid). Ütlesime „teoreetiliselt“ sellepärast, et tavaliselt on keele autor(id) juba keelde sisse kirjutanud, mil viisil seda tuleb realiseerida.
- Peatugem siinkohal ühel üpris levinud anakronismil interpretaatorite ja kompilaatorite iseloomustamisel: tihti väidetakse, et kompileeritakse alati ja kohe masinkoodi (tänapäeval tegelikult assemblerkoodi või kõrgtasemekeelde) ja et interpreteerimisel täidetakse iga käsk kohe pärast ta transleerimist ja et nii analüüsitakse programmijuppe täitmise ajal korduvalt ning et interpreteerimise kiirus ei kannata seetõttu välja võrd-

[1] Ingl. *recognizer* või *scanner*, vn. *распознаватель*.

[2] Tähelepanu! Siin räägime *regulaarsest keelest*; kui laseme *automaadi* genereerida kontekstivaba grammatika baasil, siis on see (automaat) võimeline lisaks skaneerimisele ka analüüsiks, sh. vastava puu moodustamiseks.

[3] Vt. [Isotamm, 2007], lk. 239 jj. *TTS* tugineb *süntaksorienteeritud* lähenemisele, sj. *orienteerutakse* mingile kindlale grammatikate (alam)klassile.

lust kompilaatoriga. Kunagi perfokaartide ajastul realiseeriti mõned keeled tõepoolest sel moel, aga nood ajad on ammu möödas. Nii interpretaator kui ka kompilaator töötavad samade andmestruktuuridega (eeskätt programmi analüüsi puuga) ning ei saa olla juttugi lähteteksti kordustega analüüsist. Kui interpretaator on korralikult programmeeritud ja kui interpreteeritav keel pole interaktiivne[1], siis pole mingit põhjust, et kompileeritud programm peaks tingimata kiiremini töötama. Ent arvestades asjaoluga, et interpreteeritakse eeskätt interaktiivseid keeli, siis inimfaktorist tingitult on selliste programmide täitmine loomulikult aeglasem. Ent kompileeritud (sama) interaktiivne programm ei pruugi olla kiirem tolle programmi interpretaatorist, pigem otsustab lahendusaegade võrdlemistestide tulemused kasutaja reaktsiooniaeg.

- Allpool püüame näidata, et interpretaatori ja kompilaatori erinevused on programmeerija jaoks üpris marginaalsed. Ja *n*-kordse tsükli läbimisel teeb interpretaator tõepoolest *n* „ronimist“ analüüsi puus, ent kompileeritud programm täidab samamoodi *n* korda oma sama töö koodilõiku.
- Pole (üldiselt) vahet, kas programmi puu on saadud sisseprogrammeeritult või *TTS*ilt. Üldjuhul tuleb edasine programmeerida programmi puu läbimise loogikast lähtudes. Siiski, pea 50 aastat on üritatud siduda puu moodustamist „koodi genereerimisega“ (vt. nt. [Isotamm, 2007], lk. 299 jj.). Läbimurdvate tehnikateni paraku ei jõutud ja tänapäeval pole see uurimissuund enam aktuaalne. Me käsitleme seda valdkonda pisut põhjalikumalt jaotises 4.2.1.

[1] *Interaktiivne* keel toetab keele virtuaalarvuti ja kasutaja vahelist dialoogi programmi lahendamise ajal märksa olulisemal tasemel kui lähteandmete sisestamine või failide identifitseerimine. Nt. saab *LISP*-programmi lahendamise ajal sisestada (või programselt genereerida) uusi *LISP*-programmi fragmente, neid transleerida ja kasutada programmi jätkamisel. Või oluliselt lihtsamal juhul saab kasutaja esitada lahendamise ajal küsimusi programmi poolt avatud andmebaasi kohta ja vajadusel korrigeerida mainitud andmebaasi.

2. Kontekstivabad grammatikad: sissejuhatus

Grammatikate kirjeldamiseks pole välja kujunenud üht ja üldtunnustatud formalismi, erinevad autorid kasutavad erisugust sümboolikat ning erinevate grammatikaklasside käsitlemisel kasutatakse tihtipeale erinevaid variante (nii on nt. automaatide ning regulaarsete grammatikate käsitlus üpriski teistsugune kui kontekstivabade grammatikate sümboolika).

Järgnevalt tutvustame formaalsete keelte (grammatikate) põhimõisteid, kasutades kontekstivabade grammatikate kirjeldamiseks nende puhul tavalist (ent mitte reegliks kujunenud) formalismi.

2.1. Põhimõisted

String (vahel ka *sõne*, vn. *цепочка*) on suvaliste nähtavate märkide jada, pikkusega 0..*n* märki. *Tühja* stringi (pikkusega 0) tähistame kui ε (kr. *e*, nagu *empty* –'tühi'), ja Λ (kr. *lambda*) on tühjade stringide hulk: Λ={ε}. Vähem formaalselt, Λ on *tühisõna*.

Sümbol on suvaline string (sh. ε), mitte tingimata üksik märk (vrd. *riigi sümboolika*).

Tähestik (*alphabet, V*) on mingit printsiipi järgides moodustatud *sümbolite* hulk.

Mitteterminaalne (*nonterminal*) tähestik (V_N) on *defineeritavate mõistete hulk* .

Terminaalne (*terminal*) *tähestik* (V_T) on nende sümbolite hulk, mida saab kasutada antud grammatikaga defineeritud keele tekstides[1]. Selle tähestiku elemente nimetatakse vahel ka *terminal*ideks või *lekseem*ideks.

$V=V_N \cup V_T$. Tähestik *V* on terminaalse ja mitteterminaalse tähestiku ühend, sj. $V_N \cap V_T = \varnothing$.

Trigol-keele[2] *mõisted* (sümbolid hulgast V_N) on *programm*, *operaatorid*, *operaator*, *label*, *omistamine*, *iflause*, *suunamine*, *lugemine*, *kirjutamine*, *muutuja*, *aritmav*, *loogilav*, *tingimus*, *yksliige* ja *tegur*.

Sümbolid, millest programmi nähtav tekst koosneb (nn. *lekseemid*, V_T) on *Trigol*is järgmised: *reservsõnad* `GOTO, IF, THEN, READ ja WRITE`, *eraldajad* `# ; : := + - * / ( ) < > <= >= /=` ja `=` ning kaks *lekseemiklassi* – `#i#` ja `#c#`[3].

Identifikaator on näiteks *M1* ja konstant 221, kumbki neist ei kuulu ei mõistete ega reservsõnade hulka ja on seega tuvastatavad oma identifikaatorite ja konstantide klasside esindajatena, *M1* ∈ `#i#` ja 221 ∈ `#c#`.

[1] Kui tegemist on programmeerimiskeelega, siis neiks *tekstideks* on sellekeelsed programmid.
[2] Vt. 1. peatükk.
[3] *Identifikaatorid* ja *konstandid*; normaalne oleks neid defineerida *regulaarse grammatikaga*; *Trigol*is on nende tuvastaja sisse programmeeritud ning neisse klassidesse kuuluvate stringide moodustamise eeskirjad on esitatud verbaalselt (informaalselt).

2.2. Edasised kokkulepped

V_T^* on tähestiku V_T baasil moodustatavate sõnade[1] (lõppvariandis, loe programmide) hulk. Näiteks, meie P6.tri tekst kuulub keele *Trigol* V_T^* hulka.

V^* on tähestiku V baasil moodustatavate sõnade hulk. Seda hulka kasutatakse (esimeses lähenduses) mõistete defineerimiseks (taas *Trigol*):

```
<omistamine> ::= <muutuja>:=<aritmav>.
```

„omistamine“ on mõiste, mida defineeritakse stringiga

```
<muutuja>:=<aritmav>,
```

see string koosneb nii mõistete nimedest (*muutuja* ja *aritmav*[2]) kui ka terminaalse tähestiku sümbolist (:=); seega definitsiooni parem pool kuulub hulka V^*.

Λ on *tühisõna*. Edasises eeldame, et $\Lambda \notin V^*$ (s.o. keelame ära produktsioonid kujul $A \rightarrow)$[3]. Keelamise põhjus on puhtpragmaatiline: see tagab programmi puu moodustamise ajalise keerukuse tuntava alanemise.

2.3. Mõistete defineerimine

Formaalses grammatikas tuleb defineerida kõik mõisted. Defineerimiseks kasutatakse (tavaliselt, järgides *Chomsky*t), järgmist metakeelt (mis sai nimeks *Chomsky normaalkuju, CNF*):

```
<mõiste> → <kirjeldus>.
```

Veel kord, *Chomsky* klassifikatsioon pole standardiseeritud, s.o. allpoolesitatav formalism pole mingil moel standard. Ja too klassifikatsioon ise pole ka üldtunnustatud, *Interneti*-allikatest võib leida sootuks rohkem keelte klasse ja alamklasse, ent meie raamatu suunitlus *pole* ei formaalsete keelte ajaloo ega ka nende teooria arengu uurimine. Niisiis, „klassikaline *Chomsky* klassifikatsioon“ (täpselt sellist pole *N. Chomsky* artiklites, ent seda kasutatakse tavaliselt toda klassifikatsiooni esitavas kirjanduses, sestap „klassikaline“)[4]:

klass 0: reeglid kujul $a \rightarrow b$, $a, b \in V^*$ (*rekursiivselt loenduvad keeled*, tuvastatavad *Turing*i masinaga, siia klassi kuuluvad ka *loomulikud* (eesti, vene, ...) *keeled*);

klass 1: *kontekstitundlikud* keeled; reeglid kujul

$aAb \rightarrow acb$, $a,c,b \in V^*$, (lubatud, et a või $b = \varepsilon$, ent tavaliselt mitte, et $c=\varepsilon$), $A \in V_N$;

klass 2: *kontekstivabad keeled*, reeglid on kujul

[1] *Sõna* on sisuliselt sama mis *string* või *sõne*, ent antud kontekstis on oluline tunnus klassikuuluvus. Etteruttavalt, mingi programmeerimiskeele puhul iga selles keeles kirjutatud süntaktiliselt korrektne programm on formaalselt tolle keele *sõna*. Lõplike sõnade (programmide) tuletamise käigus moodustatavad *lause vormid* kuuluvad kõik hulka V^*.

[2] *aritmav* on mõistagi „aritmeetiline avaldis“.

[3] Kui seda seika tahetakse eriti rõhutada, võib kasutada tähistust V^+: $V^+ = V^* \setminus \Lambda$.

[4] Loe lisaks [Koit&Roosmaa], lk. 172 jj.

$A \to a$, kus $a \in V^*$ ja $A \in V_N$;

klass 3: *regulaarsed keeled*, reeglid on kujul

$A \to a$ ($a \in V_T^*$ ja $A \in V_N$) või $A \to Ba$ (või $A \to aB$), $a \in V_T^*$ ja $A, B \in V_N$.

2.4. Kontekstivabad grammatikad (KVG)

Kontekstivaba grammatika (*KVG*) on nelik (V_N, V_T, P, S), kus V_N on mitteterminaalne (mõistete) tähestik, V_T on terminaalne tähestik, P on produktsioonide hulk kujul $A \to x$, kus $A \in V_N$ ja $x \in V^*$ ja S on aksioom: fikseeritud täht tähestikust V_N, mida ei saa kasutada teiste mõistete defineerimiseks (ent mis ise *peab* olema defineeritud; programmeerimiskeelte puhul on loomulikuks aksioomiks mõiste *programm*).

Chomsky klassifikatsiooniga defineeritakse *generatiivseid* grammatikaid. Kuivõrd me keskendume (nüüd ja edaspidi) kontekstivabadele grammatikatele, siis generatiivne tähendab, et *aksioomist* lähtudes on võimalik genereerida kõik antud grammatikaga defineeritud keele sõnad: kui sõna on $x \in V_T^*$, siis on genereeritavad kõik need sõnad x, mida me saame *tuletada* aksioomist S produktsioonide abil. Näiteks, kui tegemist on *C*-keele grammatikaga, siis hulk V_T^* koosneb *kõikvõimalikest süntaksivigadeta*[1] *C-programmidest*. Allpool defineerime mõned mõisted, mida äsjaöeldu demonstreerimiseks vajame.

Seejuures kasutame näi(de)teks abstraktseid (teisisõnu õppe)grammatikaid, mis on süntaktiliselt korrektsed, ent semantiliselt sisutühjad. Nagu näiteks grammatika *g1.grm*[2]:

`S'->#`A'#	lihtsalt:	S->#A#
`A'->`B'`C'		A->BC
`B'->a		B->a
`B'->`B'a		B->Ba
`C'->b		C->b

Olgu $x,y \in V^*$. Ütleme, et x-st on *vahetult tuletatav* y ($x \Rightarrow y$), kui
$x=uAv$, $y=uzv$ ja $A \to z \in P$; $u,z,v \in V^*$, $A \in V_N$.

Sõnast x on *tuletatav* sõna y ($x \overset{*}{\Rightarrow} y$), kui leidub sõnade jada
$z_0, z_1, \ldots, z_k$ ($z_i \in V^*$) selline, et
$x=z_0$, $z_{i-1} \Rightarrow z_i$ ja $z_k=y$ ($1 \le i \le k$).
Naturaalarvu k nimetame *derivatsiooni*[3] *pikkuseks* ja jada $z_0, z_1, \ldots, z_k$ *derivatsiooniks*.

---

[1] Mõistagi kuuluvad sellesse hulka ka kõik semantikavigadega, ent süntaktiliselt korrektsed programmid.

[2] See grammatika on vormistatud vastavalt meie *TTS*i kokkulepetele. Esiteks, faili nimi (kujul *xxx.grm*) informeerib, et *keele* nimi on *xxx* ja et antud fail esitab selle keele produktsioonide hulka *P* – nime laiend on *grm*. Oma *TTS*i programmeerisime nii, et *õppegrammatikate* nimed algavad prefiksiga *g* (või *G*) ning antud keele *sõnade* (tolles keeles kirjutatud programmide) nimed tuleb kirjutada kujul *zzz.xxx*, järgides väljakujunenud tavasid. Nt, kui realiseerime oma *TTS*iga *C*-keele, siis tema grammatika nimi on *c.grm* ning *C*-programmi nimi on näiteks *mull.c* (*g1.grm-„programm*“ on nt nimega *w22.g1*). Ja teiseks, *mõistete* markerite paari „<…>“ asemel kasutame *apostroofide paari* ja seda mugavusest: nii saame vältida tehnilisi probleeme võrdlustehete (terminalide „<“ ja „>“) eristamisel metakeelsetest markeritest. Meie õppevahendi tekstis kasutame tavaliselt lihtsustatud varianti: mõisteid tähistavad suurtähed, kõik muud on terminalid (vt. veergu „lihtsalt“), ent reaalne *TTS* seda lihtsustust ei aktsepteeri.

[3] Derivatsioon (ingl. *derivation*) on eesti keeles *tuletamine* (vn. *приведение*).

Kanooniline derivatsioon (ingl. *right derivation*) on selline, kus igal sammul asendatakse kõige parempoolsem mitteterminal[1] (kanoonilisust tähistab *k* topeltjoontega noole all). Niisiis, derivatsiooni abil saab genereerida iga antud keele sõna − kui alustame tuletamisi aksioomist *S*. Loeme derivatsiooni lõpetatuks, kui viimases sõnas pole enam ainsatki mitteterminali (mõistet).

Ja veel, derivatsiooni $S=z_0 \Rightarrow z_1 \Rightarrow ... \Rightarrow z_k$ korral, kus $z_k \in V_T^*$, siis öeldakse, et z_i (i=0,..,k) on *lause vorm*[2].

Olgu antud *KVG* nelikuga (V_N,V_T,P,S). Tema poolt defineeritud *keel* $\mathscr{L}(G)$ on hulk

$\mathscr{L}(G)=\{x: S \overset{*}{\underset{k}{\Rightarrow}} x \,\&\, x \in V_T^*\}$.

See tähendab, et kontekstivaba grammatika poolt genereeritav keel on niisuguste sõnade *x* hulk, mis on kanooniliselt tuletatavad aksioomist *S* ning on moodustatud terminaalse tähestiku sümbolite baasil (mõtlemiseks: mitme katsega saaks nii genereerida ühe midagi tegeva programmi?).

Süntaksi analüüsi ülesanne on püstitatud järgmiselt: on antud *KVG* ja sõna *x*. Tuleb välja selgitada, kas $x \in \mathscr{L}(G)$, ja kui kuulub[3], siis leida sõna *x* derivatsioon aksioomist *S*. Olgu sellisel juhul jada $z_0, z_1, \ldots, z_k$ sõna *x* derivatsioon aksioomist *S*, see tähendab, $x_0 = S$ ja $z_k = x$. Jada z_k, $z_{k-1}, \ldots, z_0$ nimetatakse sõna *x analüüsiks.*

Selline analüüsiülesande püstitus on erialakirjanduses tuntud kui *alt-üles-analüüs*, mille eesmärk on taastada kanooniline derivatsioon aksioomist *S* sõnani *x*.

Meenutagem: meie eesmärk on ehitada programmi *analüüsi puu*, ja et analüüs on tagurpididerivatsioon, siis vaadakem, kuidas puid ehitatakse. Ehituseeskirjad on esitatud grammatika produktsioonidega. Alustagem tuttava tehnikaga − derivatsiooniga − mille käigus saame moodustada *puu* (täpsemalt, *derivatsiooni puu*).

Kontekxstivaba grammatika *G derivatsiooni puu* on järgmiste reeglite kohaselt märgendatud tippudega puu:

- üksainus tipp märgendiga *S* (s.o. aksioom) on derivatsiooni puu;
- kui *D* on derivatsiooni puu ja *N* tema tipp märgendiga $A \in V_N$ ning $A \rightarrow X_1,\ldots,X_p \in P$, saame moodustada uue derivatsiooni puu *D'*, asendades tipu *N* märgendi märgendiga $A \rightarrow X_1,\ldots,X_p \in P$ ja lisades tipule *N p* alluvat, millede märgendeiks paneme (vasakult paremale) $X_1,...,X_p$.

Derivatsiooni puu, mille *kõik* märgendid kuuluvad hulka $P \cup V_T$, on *analüüsi puu*[4].

---

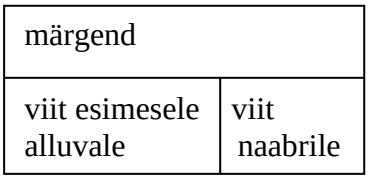
[1] Pisut etteruttavalt, kanoonilisus on oluline analüüsi aspektist. Nii garanteeritakse, et analüüsimagasini tipus on alati järgmist redutseerimissammu loomulikult võimaldav lause vormi baas.

[2] Ingl. *sentential form.* Et see termin ei tunduks eksitavana, võime kokku leppida, et mainitud *lause* on derivatsioon, mis koosneb *sõnadest* – lause vormidest.

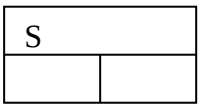
[3] Sõna *x ei kuulu* keelde, kui ei leidu derivatsiooni aksioomist *S* sõnani *x*: viimases on süntaksiviga või lubamatu lekseem (s.o. $x \notin V_T^*$).

[4] Kõikide mitterippuvate tippude märgendid on produktsioonid ja kõikide lehtede märgendid on terminalid.

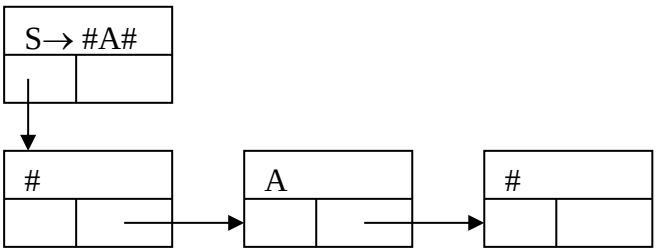
Näitena teeme kanoonilise derivatsiooni koos puu moodustamisega grammatika *G1* baasil. Puu tipu formaat olgu seejuures järgmine:



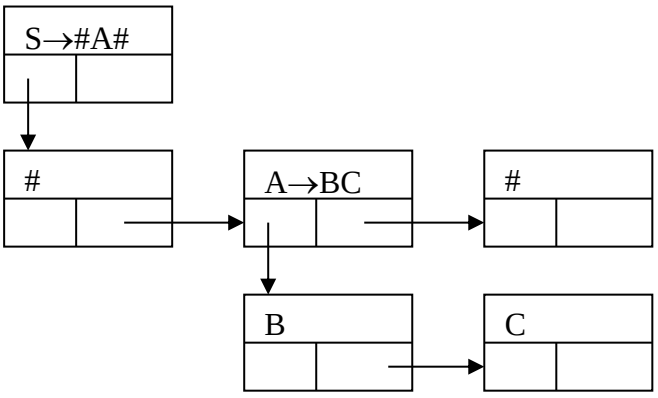
Esimene (ühetipuline) puu on selline:



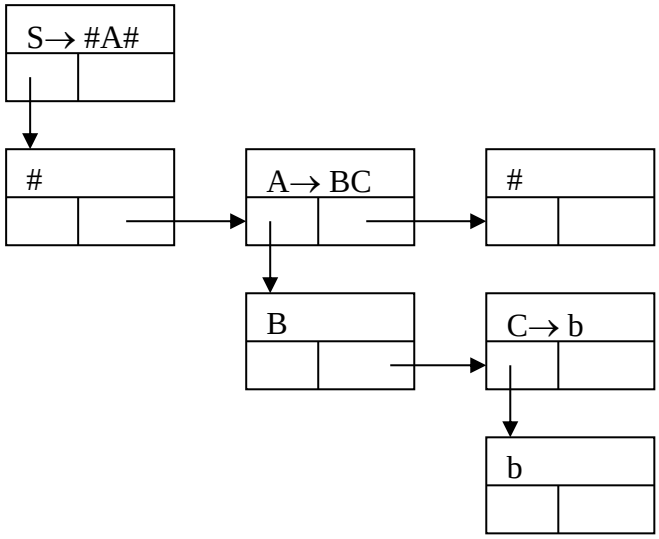
Asendame märgendi *S* märgendiga $S \rightarrow$ *#A#* ja kasvatame puud:



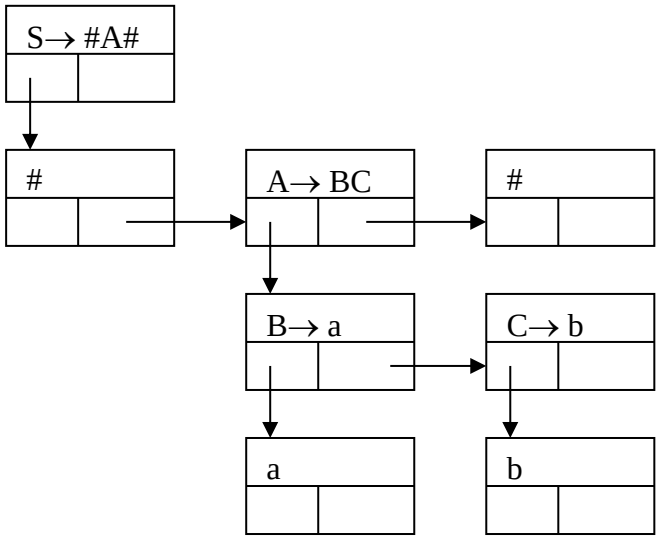
Järgmine samm (kui tahame jõuda analüüsi puuni) on samuti sunnitud: asendame märgendi *A* uuega: $A \rightarrow BC$, ja lisame tipud märgenditega *B* ning *C*:



Kuivõrd me tegeleme *kanoonilise* derivatsiooniga, siis jätkame tipuga, mille märgend on *C*:



Järgmisena asendame lause vormis mitteterminali *B*: $B \rightarrow a$.



Derivatsioon on lõppenud, puus pole enam ühtegi tippu, mis oleks märgendatud mitteterminaalse tähestiku sümboliga (mõistega), ja et kõikide mitterippuvate tippude märgendeiks on produktsioonid ning rippuvatel tippudel terminaalse tähestiku sümbolid, siis meie viimane puu on definitsiooni kohaselt sõna *# a b # analüüsi puu.*

Süntaksorienteeritud transleerimine on selline tehnika, kus etteantud programmi tekstist (formaalselt on see $x \in V_T^*$) taastatakse ta kanooniline derivatsioon aksioomist *S* ja ehitatakse analüüsi puu. Sellist tegevust nimetatakse programmi *analüüsiks*. Formaalselt on *analüüs* derivatsiooni vastandtehe: kui jada $S=z_0 \Rightarrow z_1 \Rightarrow z_2 \Rightarrow \ldots \Rightarrow z_k=x$ on sõna $x \in V_T^*$ *derivatsioon* aksioomist *S*, siis jada $z_k=x \in V_T^*, z_{k-1}, z_{k-2}, \ldots, z_1, z_0=S$ on sõna *x* analüüs.

Konkreetseid analüüsimeetodeid on välja töötatud (enamasti möödunud sajandi 60. ja 70. aastatel) palju. Põhimõtteliselt jagunevad nad kahte suurde klassi: *alt-üles*- ja *ülalt-alla*-meetodid; nagu nimetused vihjavad, püütakse esimesel juhul taastada derivatsiooni puu lehtedega (rippuvate tippudega) alustades, teisel puhul, vastupidi, puu juurest. Ning mõlemal variandil on hulgaliselt konkreetseid lahendusi. Teoreetiliselt on kõik meetodid huvipakkuvad, ent

pragmaatiliselt huvitavad on vaid need meetodid, mis garanteerivad lineaarse ajalise keerukusega[1] analüüsi. Selle saavutamiseks tuleb kontekstivabade grammatikate hulgast eraldada kiiret analüüsi võimaldav alamhulk; esimeseks lähenduseks on piirdumine *üheste* grammatikatega.

Grammatika on *ühene*, kui suvalise tema poolt defineeritud keele sõnani leidub parajasti üks kanooniline derivatsioon. Toome lihtsa näite (grammatika G10[2]):

```
S →A B
A →a A
A →b
B →B c
B →d
```

Sõnani *aabd* on ainult üks kanooniline derivatsioon:

S ⇒ A B ⇒ A d ⇒ a A d ⇒ a a A d ⇒ a a b d.

Modifitseerime grammatikat G10, saades grammatika G11:

```
S →A B
A →a A
A →b
A →A
B →B c
B →d
```

Lisandus produktsioon $A \rightarrow A$ — lihttsükkel (komplitseeritum variant on näiteks $A \rightarrow C$, $C \rightarrow D$, $D \rightarrow A$). Nüüd võime sõnani *aabd* teha kui tahes palju erinevaid kanoonilisi derivatsioone, näiteks $S \Rightarrow A\ B \Rightarrow A\ d \Rightarrow a\ A\ d \Rightarrow a\ a\ A\ d \Rightarrow a\ a\ A\ d \Rightarrow a\ a\ b\ d$, seega G11 pole ühene, s.o. ta on *mitmene*. Grammatika on *oluliselt mitmene*, kui temast ei leidu ühest varianti; kuivõrd G10 on grammatika G11 ühene variant, siis G11 pole oluliselt mitmene. Et grammatika ja tema poolt defineeritud keel oleksid (loodetavasti) ühesed, ei tohi produktsioonide hulk sisaldada tsükleid, ja teiseks, keelatud on tühisõna, see on produktsioon kujul $A \rightarrow$ (formaalselt $A \rightarrow \Lambda$)[3].

Järgnevas tutvume analüüsi algoritmiga. See kasutab *LIFO*-tüüpi magasini, kuhu võime lisada (operatsiooniga *push*) järjekordse sisendrea-lekseemi või võtta midagi ette magasini tipmise elemendi (või tipmiste elementidega), kasutades operaatorit *pop*. Kusjuures mõlemal juhul juhib otsustamist lekseemi ja/või magasinielementide vaheline suhe. Noid suhteid nimetatakse *eelnevusrelatsioonideks* ning nad on tuvastatavad *KVG* produktsioonide hulga *P* baasil: nende abil me leiame magasinist lause vormi baasi – mingi produktsiooni parema poole − et see asendada magasinis tolle produktsiooni vasaku poolega, s.o. defineeritud mõistega. Siinkohal tasuks meelde tuletada, kuidas ehitatakse derivatsiooni puud ja millist rolli mängib selle juures lause vormi baas.

---

[1] Kui programmiteksti lekseemide arv on *n*, siis lineaarne ajaline keerukus on väljendatav kui $O(n)$. Vt. nt. [Isotamm, 2009], lk. 74 jj.

[2] Grammatikate nimed on üldiselt suvalised, isegi *TTS*i kasutades. Enamasti on grammatikate „raamatunimed" ja samanimelised *TTS*-nimed kattuvad, ent ei pruugi. Seejuures mitte pahatahtlikkusest.

[3] Probleem on (teoreetiliselt) tõsisem: me teame, mis tingimustel (nt. tsükkel ja tühisõna) grammatika ja tema poolt defineeritud keel on garanteeritult mitmesed, ent *pole teada algoritmi*, mis teeks *üldjuhul* kindlaks, kas *KVG* on ühene või ei ole. Johtuvalt meie raamatu pragmaatilisest suunitlusest me noid probleeme ei käsitle. Siiski, lisas 1 on toodud näide oluliselt mitmesest keelest.

Eelnevusrelatsioone kasutav analüüsimeetod kitsendab kontekstivabade grammatikate klassi − vastavat alamklassi nimetatakse *eelnevusgrammatikate klassiks* ja kuulumaks sellesse klassi, peab grammatika rahuldama mitmeid lisatingimusi. Ja, nagu juba öeldud, me käsitleme neid asju edaspidi ja allpool.

3. Eelnevusgrammatikad

Kui grammatika pole ühene, siis analüsaatori teoreetiline ajalise keerukuse hinnang on $O(n^3)$ – n on programmi lekseemide arv, ja ühese grammatika puhul on too hinnang üldjuhul $O(n^2)$. Saavutamaks lineaarset kiirushinnangut $O(n)$ on välja töötatud veelgi rohkem kitsendusi, üks paljudest on piirdumine *eelnevusgrammatikate* klassiga[1]. Publikatsioone järgides oli esimene selle grammatikate klassi formaliseerija ja terminoloogia rajaja *Niklaus Wirth* [Wirth].

Alustame lihtsa näitega. Ülalpool oli meil grammatika G1. Derivatsioon sõnani *#ab#* on:

S ⇒ #A# ⇒ #BC# ⇒ #Bb# ⇒ #ab#.

Sõna *#ab#* analüüs on järgmiste lause vormide jada:

#ab# ⇒ #Bb# ⇒ #BC# ⇒ #A# ⇒ S.

Igas *i*-ndas (*i=k,...,1*) analüüsi lause vormis z_i (z_k=*#ab#*, z_1=*#A#*) on allakriipsutatud lause vormi *baas* – selle produktsiooni parem pool, mis järgmisel sammul asendatakse produktsiooni vasaku poolega.

Lause vormi baasi otsimist nimetatakse *detekteerimiseks*[2] ning järgmise lause vormi moodustamist (detekteeritud baasi (mingi produktsiooni parema poole) asendamist produktsiooni vasema poolega) – *redutseerimiseks* (eesti keeles võiks see olla *taandamine*).

Detekteerimisülesannet võimaldab lahendada tähestiku V ($V=V_N \cup V_T$) elementide omavaheliste eelnevus-järgnevustingimuste (*eelnevusrelatsioonide*) fikseerimine, tuginedes produktsioonide parematele pooltele. Mainitud relatsioonid on "eelneb", "ajastub" ja "järgneb"[3] ($\lessdot$, $\doteq$ ja $\gtrdot$). Etteruttavalt, kui *abc* on lause vormi baas, siis magasinis $\# \lessdot \ldots \lessdot a \doteq b \doteq c \gtrdot T_i$, mis on vasakpoolseim jooksev terminal väljaspool magasini.

Ja kui ülalpool nentisime, et analüüsi eesmärk on kanoonilise derivatsiooni taastamine (ja derivatsiooni puu alt-üles-ehi-tamine), siis nood eelnevusrelatsioonid on paika pandud võimaldamaks tuvastada produktsioonide $A \rightarrow x \in P$ paremaid pooli, $(x=X_1 \ldots X_p \in V^*)$.

[1] Meie valisime selle meetodi kahel põhjusel: esiteks, Eestis oli see esimene teadasaadud süntaksorienteeritud *TTS*i kirjutamise vahend, ja teiseks, eeskätt *Mati Tomba*ku eestvedamisel on just seda strateegiat järgiv *TTS* realiseeritud paljudel erinevatel masinatel (nt. *Minsk*-32, *Iskra*, *CM*-4, *Apple*, *EC-1022*, *PC AT*, aga mitte enam *Windows-PC*, see jäi nende ridade autori teha).

[2] Vrd. *detektiiv*.

[3] Relatsioonide eestikeelsed nimetused pärinevad 70-ndate aastate algusest *Leo Võhandu*lt. Paradoksaalsel moel pole relatsioonidel nimetusi ei inglise ega vene keeles, rahuldutakse vastavate sümbolitega. Vaene lektor tahvli ees…

3.1. Eelnevusrelatsioonid

Eelnevusrelatsioonid on defineeritud lähtuvalt tähestiku *V* sümbolite paiknemisest produktsioonide paremates pooltes[1]; relatsioonide *eelneb* ja *järgneb* leidmiseks peame defineerima mitteterminalist *A* tuletatavate sõnade algus- ja lõpusümbolite hulgad *L*(*A*) ja *R*(*A*)[2]:

$L(A)=\{X: X\in V \,\&\, [\exists\phi: A\rightarrow Xu \vee \exists\phi: A\rightarrow Bu \,\&\, X\in L(B)]\};$
$R(A)=\{X: X\in V \,\&\, [\exists\phi: A\rightarrow uX \vee \exists\phi: A\rightarrow uB \,\&\, X\in R(B)]\}.$

Eelnevusrelatsioonid on defineeritud järgmiselt:

$X \doteq Y \equiv \{(X,Y): \exists\phi: A\rightarrow uXYv\};$
$X \lessdot Y \equiv \{(X,Y): \exists\phi: A\rightarrow uXBv \,\&\, Y\in L(B)\};$
$X \gtrdot Y \equiv \{(X,Y): [\exists\phi: A\rightarrow uBYv \,\&\, X\in R(B)] \vee [\exists\phi: A\rightarrow uBCv \,\&\, X\in R(B) \,\&\, Y\in L(C)]\};$
$X \odot Y \equiv$ „X ja Y vahel puudub relatsioon“.

Kontekstivaba grammatikat nimetatakse *eelnevusgrammatikaks* siis, kui suvalise kahe sümboli vahel tähestikust *V* kehtib ülimalt üks eelnevusrelatsioon.

Eelnevusmeetodit (analüüsiks) kasutava *KVG* testimine[3] algab *eelnevusmaatriksi* koostamisest. See on $|V| \times |V|$ - maatriks, esimesed read ja veerud on terminaalsetele, järgmised mitteterminaalsetele sümbolitele – miks selles järjekorras? Keele terminaalsete sümbolite hulk on keele defineerimisega fikseeritud, ent mõistete arv mitte. Me käsitleme seda pisut hiljem, grammatika modifitseerimise juures (vt. *KVG* teisendamine eelnevusgrammatikaks, osas 3.2, asja huvides võime esialgseid mõisteid ositada, ilma et keel muutuks).

Kasutagem näitena taas grammatika *G1* produktsioonide hulka:

```
S -> # A #
A -> B C
B -> a
B -> B a
C -> b
```

Siin on tähestiku *V* pikkuseks 7 ($|V|=|V_N|+|V_T|$), ent eelnevusmaatriksis on 6×6 välja[4]:

---

[1] Ainsa tagamõttega võimaldada analüüsi käigus tuvastada lause vormi baasi, s.o. mingi produktsiooni paremat poolt.

[2] Tähistused on valitud ingliskeelsete sõnade *L*(*eftmost*) ja *R*(*ightmost*) algustähtede järgi. Tõlkida võime nii: *A vasakpoolseimate* ja *parempoolseimate* sümbolite hulgad, *A*-st tuletatavate sõnade alg- ja lõppsümbolid. Nende otstarbeks on aidata tuvastada lause vormi baasi algust ja lõppu analüüsiaegses magasinis eelnevusrelatsioonide abil. Vt. jaotist 3.3.

[3] *Testime* esimeses lähenduses, edaspidi kasutame eelnevusmaatriksit sõna analüüsimisel (vajadusel ka $C_{1|1}(A)$ arvutamisel). Ilma liialdusteta on eelnevusmaatriks analüsaatori tähtsaim vahend.

[4] Aksioomil *S* pole definitsiooni kohaselt relatsioone ülejäänud tähestiku *V* sümbolitega, seetõttu võime ta oma õppepiltidelt välja jätta. Reaalsetes andmestruktuurides on ta omal kohal, ehkki 0-rea ja -veeruga.

	#	a	b	A	B	C
#						
a						
b						
A						
B						
C						

Tuvastamaks eelnevusrelatsioone, peame esmalt leidma mõistete jaoks hulgad *L*( ) ja *R*( ):

	L( )	R( )
S	#	#
A	B, a	C, b
B	a, B	a
C	b	b

Käsitsi lahendamiseks[1] on osutunud riskivabamaks variandiks relatsioonide leidmise järjekord „ajastub“, „eelneb“ ja „järgneb“. Niisiis, esmalt „ajastub“:

produktsioonist $S \rightarrow \#A\#$ saame, et $\# \doteq A$ ja $A \doteq \#$;
produktsioonist $A \rightarrow BC$, et $B \doteq C$;
produktsioonist $B \rightarrow Ba$, et $B \doteq a$.

	#	a	b	A	B	C
#				$\doteq$		
a						
b						
A	$\doteq$					

[1] See on kontrolltööde ja eksamivariant, paraku.

B		≐				≐
C						

Teiseks, „eelneb“:

```
# ⋖ L(A)={B,a}:  # ⋖ B, # ⋖ a;

B ⋖ L(C)={b}:  B ⋖ b.
```

Rohkem selle relatsiooni allikaid pole ja eelnevusmaatriks on nüüd järgmine:

	#	a	b	A	B	C
#		⋖		≐	⋖	
a						
b						
A	≐					
B		≐	⋖			≐
C						

Ja viimasena, „järgneb“:

```
R(A)={C,b} ⋗ # : C ⋗ #, b ⋗ #;
R(B)={a} ⋗ C: a ⋗ C;
R(B)={a} ⋗ L(C)={b}: a ⋗ b;
R(B)={a} ⋗ a: a ⋗ a.
```

Kanname needki relatsioonid maatriksisse.

	#	a	b	A	B	C
#		⋖		≐	⋖	
a		⋗	⋗			⋗
b	⋗					
A	≐					
B		≐	⋖			≐
C	⋗					

Kõik eelnevusrelatsioonid on nüüd tuvastatud ja kantud maatriksisse (meie joonistel tähistab relatsiooni puudumist (⊙) tühi lahter). Nentigem: kuivõrd suvalise kahe tähestiku *V* elemendi vahel ei kehti üle ühe eelnevusrelatsiooni, siis *G1* on *eelnevusgrammatika.* Mis tähendab: selle grammatika poolt defineeritud keele sõnu saab standardselt analüüsida (loe: ehitada nende analüüsi puid ja selle töö käigus kontrollida nende sõnade süntaktilist õigsust).

> Kasutades grammatikat *G1*, näitlikustame eelnevusrelatsioonide olemust, kirjutades neid derivatsiooni käigus lause vormidesse – analüüsi ajal hakataksegi neid just nii kasutama. Niisiis:
>
> $S \Rightarrow \#\doteq A\doteq\# \Rightarrow \#\lessdot B\doteq C\gtrdot\# \Rightarrow \#\lessdot B\lessdot b\gtrdot\# \Rightarrow \#\lessdot B\doteq a\gtrdot b\# \Rightarrow \#\lessdot a\gtrdot ab\#$

Näeme, et kanoonilise derivatsiooni järjekordsel (alates teisest, kui markerid on paigas) sammul asendatud mitteterminali parem pool on alati relatsioonide $\lessdot$ ja $\gtrdot$ vahel. Ja analüüs on sellele tõigale rajatud –, aga seda näeme allpool.

Juhul, kui mõne paari vahel hulgast *V* kehtib samal ajal üle ühe eelnevusrelatsiooni, on tegemist *eelnevuskonfliktiga*, ja nende konfliktide lahendamiseks (niisuguse grammatika teisendamiseks eelnevusgrammatikaks) on olemas algoritm, käsitleme seda järgmises jaotises. Eelnevuskonfliktidega *KVG* tuleb teisendada *EG*ks, kuivõrd lineaarse kiirusega analüüsi jaoks on algoritm teada vaid eelnevuskonfliktide puudumisel.

3.2. Eelnevuskonfliktide likvideerimine[1]

Eelnevusgrammatika (*EG*) on selline *KVG*, kus tähestiku *V* mis tahes kahe elemendi vahel kehtib ülimalt üks eelnevusrelatsioon. Kui *KVG pole* eelnevusgrammatika, siis saab ta selleks alati *programselt* teisendada. Situatsiooni, kus *V* kahe elemendi vahel kehtib samal ajal rohkem kui üks eelnevusrelatsioon, nimetatakse *eelnevuskonfliktiks.* Pragmaatilistel kaalutlustel jagatakse need konfliktid kahte klassi: *P1*- ja *P2*-konfliktid[2]. Vajadusel teeb *Konstruktor*[3] eelnevusteisendused (s.o. likvideerib eelnevuskonfliktid) automaatselt: grammatika $G=(V_N,V_T, P,S)$ teisendatakse grammatikaks $G'=(V_N',V_T,P',S)$ nii, et
$\mathscr{L}(G)=\mathscr{L}(G')$[4].

P1-konfliktiks nimetatakse situatsiooni, kus ($X, Y \in V$): $X\doteq\gtrdot Y$, $X\lessdot\gtrdot Y$ või $X\lessdot\doteq\gtrdot Y$.

P2-konfliktiks nimetatakse situatsiooni, kus $X\lessdot\doteq Y$.

Joonisel 3.2.b on esitatud eelnevuskonfliktide likvideerimise algoritmi plokkskeem. Nagu sealt näha, likvideeritakse esmalt P2-konfliktid, seejärel P1-konfliktid. Põhjus peitub konfliktitüüpide allikates; P2-tüüpi konfliktide kõrvaldamise käigus võib lisanduda uusi P1-konfliktide allikaid, P1-konfliktide likvideerimisel sellist kõrvalefekti pole. Asi on selles, et sümbolite *X* ja *Z* vahelise P1-konflikti allikateks on produktsioonid kujul

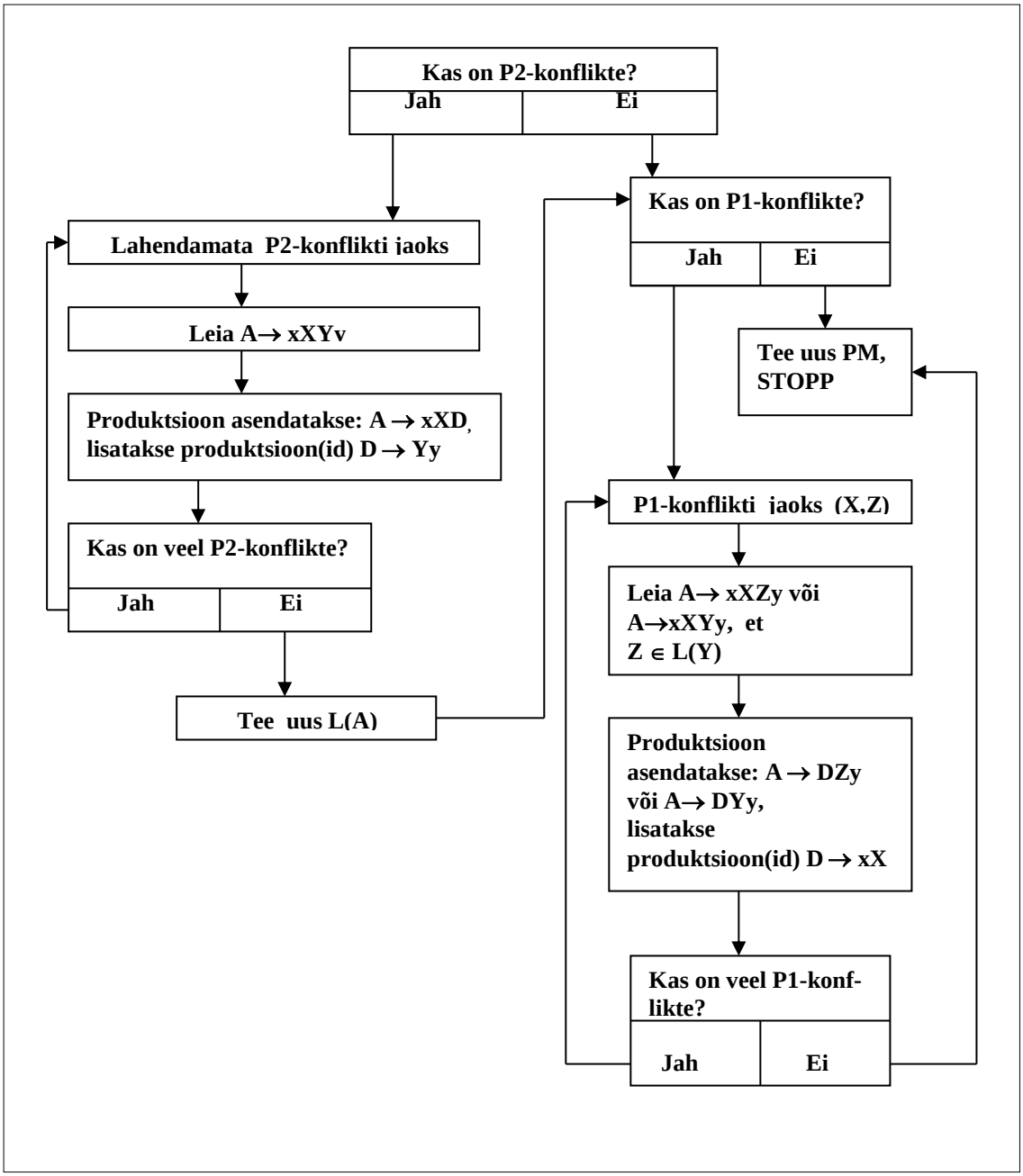
$A \rightarrow xXZy$ või $A \rightarrow xXYy$, nii et $Z \in L(Y)$ (mõistagi, $Y \in V_N$)

---

[1] Siin esitatav on *Mati Tombaku* algoritm (vt. [Tombak, 1976]).
[2] Originaalis on *eelnevus Precedence.* (*P*). Võib oletada (või meelde jätta), et *P1* tähendab, et konflikti kõrvaldamiseks asendatakse punktis *uX.Yv* või *uX.Zv* esimene pool, s.o. *uX*, ja *P2*, et asendatakse teine pool, s.o. *Yv* või *Zv.* Ning *eelnevusmaatriks* on inglise keeles *Precedence Matrix* (plokkskeemis *PM*).
[3] Silmas on peetud *translaatorite tegemise süsteemi* esimest plokki; *TTS*i käsitleme hiljem detailsemalt.
[4] Keeled on *võrdsed*, kui nende grammatikad genereerivad sama sõnade hulga.

ning P2-konfliktide likvideerimisel lisanduvad uued produktsioonid, kus paremas pooles on seni mitteesinenud sümbolite paar (*X*, *D*), kus $D \in V_N$. Kui $Z \in L(D)$, siis ongi tegemist võimaliku uute *P1*-konfliktide allikaga.

Konfliktide allikad leiame produktsioonide parematest pooltest: P2 (*X*,*Y*) puhul produktsioonidest kujul $A \rightarrow uXYv$ punktis *X.Y* ning P1 (*X*,*Z*) puhul punktis $A \rightarrow uX.Zv$ või $A \rightarrow uXYv$, kus $Z \epsilon L(Y)$. Konfliktide likvideerimiseks muudetakse produktsioone: mainitud punktides: asendatakse kas punktist vasem (P1, esimene) või parem (P2, teine) definitsioonipool, modifitseerides konflikti põhjustanud produktsiooni ning lisades uusi mitteterminale ning nende definitsioone (P2 puhul $A \rightarrow uXYv$ tehakse $A \rightarrow uXA'$ ja lisatakse $A' \rightarrow Yv$ ning P1 puhul $A \rightarrow uXZv$ tehakse $A \rightarrow A'Zv$ (või $A \rightarrow A'Yy$) ja lisatakse $A' \rightarrow uX$.



Joonis 3.2.b. Eelnevuskonfliktide likvideerimine.

Illustreerime seniesitatut abstraktse *KVG G3* näitel. *G3* produktsioonide hulk on järgmine:

```
T → # S #
S → a A a
S → b A b
S → a B b
S → b B a
A → 1
B → 1
A → A c
B → d B
```

Mitteterminalide derivatsiooniga saavutatavate lause vormide vasak- ja parempoolseimate sümbolite hulgad on järgmised (järgnevas on enamasti kopeeritud *TTS*i *HTML*-formaadis väljastatud protokolli, ent ingliskeelne tekst on asendatud eestikeelsega; kui maatriksi väljal on sümbol '*', siis see tähendab, et rea *A* veeru sümbol $s \in L(A)$ või $s \in R(A)$, näiteks $b \in L(S)$):

Hulk L( ):

sümbol	#	a	b	1	c	d	T	S	A	B
7.T	*	0	0	0	0	0	0	0	0	0
8.S	0	*	*	0	0	0	0	0	0	0
9.A	0	0	0	*	0	0	0	*	0	
10.B	0	0	0	*	0	*	0	0	0	0

Hulk R( ):

sümbol	#	a	b	1	c	d	T	S	A	B
7.T	*	0	0	0	0	0	0	0	0	0
8.S	0	*	*	0	0	0	0	0	0	0
9.A	0	0	0	*	*	0	0	0	0	0
10.B	0	0	0	*	0	0	0	0	0	*

*TTS*i logis on eelnevusmaatriksis ka aksioomi (*T*) rida ja veerg, kuivõrd ka seda sümbolit indekseeritakse (vt. rea numbrit). Mõistagi, aksioomil ei saa olla relatsioone teiste tähestiku *V* sümbolitega. Füüsilisel tasemel kodeeritakse eelnevusrelatsioone (meie *TTS*is) nagu joonisel 3.2.a; koodid on valitud mõttega, et eelnevusmaatriksi lahtrit *x* saaks alati täita tehtega $x \vee = \Re$, kus $\Re$ on relatsioon maatriksi rea- ja veerusümboli vahel (omistamisavaldise keel on *C*).

sümbol	***kood₂***	***tehe***	***kood₁₀***
⊙	000		0
≐	001		1
⋖	010		2
⋗	100		4
⋖ ≐	011	010∨001	3
⋗≐	101	100∨001	5
⋖ ⋗	110	100∨010	6
⋖ ≐ ⋗	111	010∨001∨100	7

Joonis 3.2.a. Eelnevusrelatsioonide koodid ja kodeerimine.

Eelnevusmaatriks:

sümbol	#	a	b	1	c	d	T	S	A	B
1.#	0	<	<	0	0	0	0	=	0	0
2.a	>	0	0	<	0	<	0	0	**3**	=
3.b	>	0	0	<	0	<	0	0	**3**	=
4.1	0	>	>	0	>	0	0	0	0	0
5.c	0	>	>	0	>	0	0	0	0	0
6.d	0	0	0	<	0	<	0	0	0	=
7.T	0	0	0	0	0	0	0	0	0	0
8.S	=	0	0	0	0	0	0	0	0	0
9.A	0	=	=	0	=	0	0	0	0	0
10.B	0	**5**	**5**	0	0	0	0	0	0	0

Siin on eelnevusrelatsioonid tähistatud lihtsustatult, näiteks ajastumine sümboli ≐ asemel "=" ning eelnevuskonfliktid on maatriksis kodeeritud kaheksandkoodis (5 on ⋗≐ ja 3 on ⋖≐). Niisiis, kuivõrd saime neli eelnevuskonflikti, siis peame nentima, et meie grammatika *G3* on küll kontekstivaba, ent mitte eelnevusgrammatika. Järgnevalt tuleb likvideerida eelnevus-konfliktid, esmalt *P2*-tüüpi – vt. joonist 3.2.b.

*TTS*i logis kajastub eelnevuskonfliktide likvideerimine järgmiselt (uue mõiste sümboliks saab ümberdefineeritava mõiste sümbol+sufiksiksina asenduse järjekorranumber *j* (*j*=1,2,…)):

P2-konflikt: a <•=• `A'
Allikas on produktsioon P2: `S' -> a `A' a
Lisan uue mitteterminali S1 ja produktsiooni: P10: `S1' -> `A' a
Asendan produktsiooni P2: `S' -> a `S1'
P2-konflikt: b <•=• `A'
Allikas on produktsioon P3: `S' -> b `A' b
Lisan uue mitteterminali S2 ja produktsiooni P11: `S2' -> `A' b
Asendan produktsiooni P3: `S' -> b `S2'
Edasi, vastavalt ülaltoodud algoritmile, tehakse uus *Leftmost*-hulk, avastamaks võimalikke uusi P1-konfliktide allikaid.

Uus hulk L( )

sümbol	#	a	b	1	c	d	T	S	A	B	S1	S2
7.T	*	0	0	0	0	0	0	0	0	0	0	0
8.S	0	*	*	0	0	0	0	0	0	0	0	0
9.A	0	0	0	*	0	0	0	0	*	0	0	0
10.B	0	0	0	*	0	*	0	0	0	0	0	0
11.S1	0	0	0	*	0	0	0	0	*	0	0	0
12.S2	0	0	0	*	0	0	0	0	*	0	0	0

Nüüd likvideeritakse P1-konfliktid:

P1-konflikt: `B' =••> a
Allikas on produktsioon P5: `S' -> b `B' a
Lisan uue mitteterminali S3 ja produktsiooni P12 : `S3' -> b `B'
Asendan produktsiooni P5: `S' -> `S3' a
P1-konflikt: `B' =••> b
Allikas on produktsioon P4: `S' -> a `B' b
Lisan uue mitteterminali S4 ja produktsiooni P13: `S4' -> a `B'
Asendan produktsiooni P4: `S' -> `S4' b

Uus grammatika

P 1: `T' -> # `S' #
P 2: `S' -> a `S1'
P 3: `S' -> b `S2'
P 4: `S' -> `S4' b
P 5: `S' -> `S3' a
P 6: `A' -> 1
P 7: `B' -> 1
P 8: `A' -> `A' c
P 9: `B' -> d `B'
P10: `S1' -> `A' a
P11: `S2' -> `A' b
P12: `S3' -> b `B'
P13: `S4' -> a `B'

Modifitseeritud grammatika jaoks eelnevusmaatriksi tegemist alustame taas mõistetest tuletatavate sõnade vasak- ja parempoolseimate sümbolite hulkade tuvastamisega:

Hulk L( )

sümb.	#	a	b	1	c	d	T	S	A	B	S1	S2	S3	S4
7.T	*	0	0	0	0	0	0	0	0	0	0	0	0	0
8.S	0	*	*	0	0	0	0	0	0	0	0	0	*	*
9.A	0	0	0	*	0	0	0	0	*	0	0	0	0	0
10.B	0	0	0	*	0	*	0	0	0	0	0	0	0	0
11.S1	0	0	0	*	0	0	0	0	*	0	0	0	0	0
12.S2	0	0	0	*	0	0	0	0	*	0	0	0	0	0
13.S3	0	0	*	0	0	0	0	0	0	0	0	0	0	0
14.S4	0	*	0	0	0	0	0	0	0	0	0	0	0	0

Hulk R( )

sümb.	#	a	b	1	c	d	T	S	A	B	S1	S2	S3	S4
7.T	*	0	0	0	0	0	0	0	0	0	0	0	0	0
8.S	0	*	*	0	0	0	0	0	0	0	*	*	0	0
9.A	0	0	0	*	*	0	0	0	0	0	0	0	0	0
10.B	0	0	0	*	0	0	0	0	0	*	0	0	0	0
11.S1	0	*	0	0	0	0	0	0	0	0	0	0	0	0
12.S2	0	0	*	0	0	0	0	0	0	0	0	0	0	0
13.S3	0	0	0	*	0	0	0	0	0	*	0	0	0	0
14.S4	0	0	0	*	0	0	0	0	0	*	0	0	0	0

Uus eelnevusmaatriks on järgmine:

sümbol	#	a	b	1	c	d	T	S	A	B	S1	S2	S3	S4
1.#	0	<	<	0	0	0	0	=	0	0	0	0	<	<
2.a	>	0	0	<	0	<	0	0	<	=	=	0	0	0
3.b	>	0	0	<	0	<	0	0	<	=	0	=	0	0
4.1	0	>	>	0	>	0	0	0	0	0	0	0	0	0
5.c	0	>	>	0	>	0	0	0	0	0	0	0	0	0
6.d	0	0	0	<	0	<	0	0	0	=	0	0	0	0
7.T	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8.S	=	0	0	0	0	0	0	0	0	0	0	0	0	0
9.A	0	=	=	0	=	0	0	0	0	0	0	0	0	0
10.B	0	>	>	0	0	0	0	0	0	0	0	0	0	0
11.S1	>	0	0	0	0	0	0	0	0	0	0	0	0	0
12.S2	>	0	0	0	0	0	0	0	0	0	0	0	0	0
13.S3	0	=	0	0	0	0	0	0	0	0	0	0	0	0
14.S4	0	0	=	0	0	0	0	0	0	0	0	0	0	0

Nagu näeme, õnnestusid eelnevusteisendused esimesel katsel. Rõhutagem veel kord, et vajadusel teisendab *TTS*i esimene plokk *Konstruktor KVG* eelnevusgrammatikaks. Mida me võime nentida grammatika *G3* kohta: *G3* on kontekstivaba eelnevusgrammatika, ent mitte *pööratav*: nii *P6* kui ka *P7* parem pool on 1. See seik põhjustab redutseerimisprobleemi: me ei tea, kas see 1 tuleb redutseerida mõisteks *A* või *B*.

Aga *kui* kontekstivaba eelnevusgrammatika *on* pööratav, siis saame kasutada triviaalset analüsaatorit (automaati, mis abstraheerudes põhiülesandest − analüüsi puu moodustamisest − tegeleb analüüsi ülesande esimese poolega: kas etteantud sõna (*resp*. programm) $x \in \mathscr{L}(G)$?).

3.3. Pööratavate eelnevusgrammatikate analüsaator

Lihtsaim *analüsaator* (sõna *x* analüüsi puu ehitaja ja süntaksivigade avastaja) on konstrueeritav, kui eelnevusgrammatika ongi *pööratav*. Selline on ka meie näitegrammatika *G1*. Niisiis, sõna (programmi, kui tegemist on programmeerimiskeelega) analüüs (ja analüüsi puu ehitamine, ent sellest hiljem) baseerub mõistetel *lause vorm* ja *lause vormi baas*[1].

Kui $S \overset{*}{\underset{\kappa}{\Rightarrow}} X_1, X_2, \ldots, X_k A T_1, \ldots, T_n \underset{k}{\Rightarrow} X_1, X_2, \ldots, X_k Z_1, \ldots, Z_m T_1, \ldots, T_n$,
kus $X_i, Z_j \in V$, $T_h \in V_T$, (*i=1,...,k, j=1,...,m* ja *h=1,...,n*), siis jada $Z_1, \ldots, Z_m$ on lause vormi *baas* (viimasena rakendati produktsiooni $A \rightarrow Z_1, \ldots, Z_m$), ja kehtivad relatsioonid:

$X_i \doteq X_{i+1} \vee X_i \lessdot X_{i+1}$ *(i=1,...,k-1);*
$X_k \lessdot Z_1$;
$Z_j \doteq Z_{j+1}$ *(j=1,...,m-1);*
$Z_m \gtrdot T_1$.

[1] Ehkki ülalpool oli meil sellest juttu, siis siin esitame vajaliku formalismi.

Programmi analüüs toimub *LIFO*-tüüpi magasini abil ning seda juhivad eelnevusrelatsioonid.

Kui relatsioon $\Re$ magasini tipmise elemendi ja jooksva lekseemi vahel on „eelneb“ või „ajastub“, siis kantakse too lekseem (koos relatsiooniga $\Re$) magasini ja aktiveeritakse järgmine lekseem, kui aga relatsioon on „järgneb“, siis täidetakse järgmised kaks sammu:

- *detekteerimine*: magasini tipust alates otsitakse magasinist esimest relatsiooni „eelneb“ – nii tuvastatakse *lause vormi baas*i algus;
- *redutseerimine*: magasinis asendatakse lause vormi baasi moodustavad sümbolid – mingi produktsiooni parem pool – tolle produktsiooni vasakul poolel oleva mitteterminaliga; paika pannakse ka relatsioon magasini viimasena paika jäänud sümboli ja lisatud mitteterminali vahel. Redutseerimine on lihtne, kui detekteeritud lause vormi baas on produktsioonide hulgas unikaalne. Selles jaotises vaatlemegi just seda lihtsat juhtu.

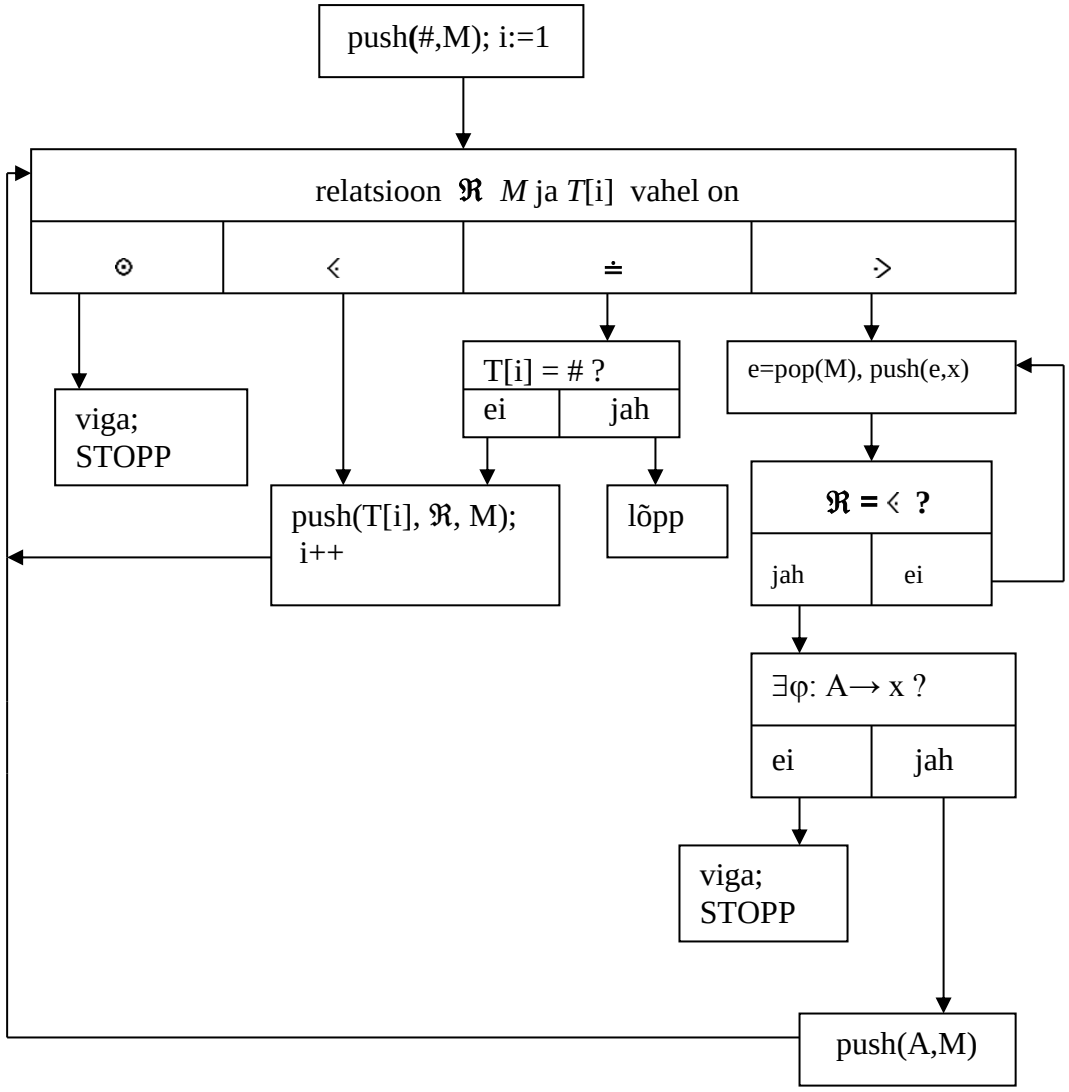
Esitame siinkohal selle lihtsaima analüsaatori algoritmi joonisel 3.3.b ja magasini formaadi joonisel 3.3.a. *Analüsaator* kasutab kolmetraktilist[1] *LIFO*-tüüpi magasini *M*: esimeses traktis on sümbol (lekseem või redutseeritud mõiste), teises relatsioon magasinis eelneva ja antud elemendi vahel ja kolmandas (võimalik) viit ehitatava analüüsi puu tipule (esialgu me ei käsitle midagi analüüsi puuga seonduvat).

trakt	sisu
1	sümbol $s_j \in (V_T \cup V_N)$ j=0,..magasini tipu indeks (s_0=#)
2	relatsioon s_{j-1} $\Re$ s_j j=1,.. magasini tipu indeks
3	viit analüüsi puu tipule või ∅

Joonis 3.3.a. *Analüsaatori* kolmetraktiline magasin.

Operaatorid *push* ja *pop* lisavad ja eemaldavad *LIFO*-magasini tipmisi elemente: *push*(*s*, $\Re$, *M*) kirjutab *M* esimestesse traktidesse sümboli *s* ning relatsiooni magasini eelmise sümboli ja sümboli *s* vahel. *LIFO*-magasin *x* on lause vormi baasi kogumiseks, tolle baasi saab sealt kätte *FIFO*-magasini kasutades (mida plokkskeem ei kajasta). Plokkskeemis on vektor *T* sisendrida: kodeeritud lekseemide jada (*T*[0]=#) ning *i* on selle vektori juurdepääsuindeks.

[1] Mitmetraktilist magasini indekseeritakse ühe ja sama indeksiga; iga trakt on realiseeritav omaette, sobivat tüüpi vektorina.



.

Joonis 3.3.b. *Analüsaator* pööratavate abstraktsete[1] eelnevusgrammatikate jaoks, ilma puuta.

Meie õppe- (abstraktne) grammatika G1 on pööratav ja me konstrueerisime selle grammatikaga defineeritud keele $\mathscr{L}(G1)$ sõna *#ab#* analüüsi puu (järjestikuste kanooniliste derivatsioonide abil). Puu moodustamist analüüsi käigus käsitleme järgmises alapunktis, siin aga mängime läbi pööratavate eelnevusgrammatikate analüsaatori algoritmi; kuivõrd puutegevused on välja lülitatud[2], piirdume kahetraktilise magasiniga. Joonistel on toodud vasakul magasini jooksev seis ning paremal – etteantud sõna veel vaatlemata osa. Analüüs algab markeri # magasini panekuga:

#

a	b	#

[1] Kui grammatika kirjeldab *programmeerimiskeelt*, siis analüsaator on mõnevõrra keerulisem: me ei tohi vea puhul analüüsi katkestada, vaid peame valima mingi jätkamisstrateegia. Viga ei tohi päädida STOP-iga.

[2] Sisuliselt tegeleme analüüsiülesande esimese poolega: *tuvastame*, kas etteantud string kujutab endast grammatikaga defineeritud keele sõna või ei.

Niisiis, esimene samm: magasini tipus on marker # ja sisendreas on ootamas string „*ab#*“. Relatsioon $\mathfrak{R}$ sümbolite # ja *a* vahel on $\lessdot$, see ja *a* lähevad magasini:

#	a
	$\lessdot$

h	#

Edasi, $a \gtrdot b$. Sümbol *a* peab olema (kui analüüsitav sõna on korrektne) mingi produktsiooni parem pool; tõepoolest, selline leidub: $B \rightarrow a$. Asendame magasinis sümboli *a* sümboliga *B* (see on redutseerimine) ja paneme paika relatsiooni # ja *B* vahel.

$B \lessdot b$. Seega, nii sisendrea sümbol *b* kui ka relatsioon sümbolite *B* ja *b* vahel tuleb kanda magasini:

#	B
	$\lessdot$

h	#

#	B	b
	$\lessdot$	$\lessdot$

#

Edasi, $b \gtrdot$ #. Lause vormi baas on *b*; produktsioonide hulgas on $C \rightarrow b$ ning redutseerime: asendame magasinis sümboli *b* sümboliga *C*, üksiti kirjutame teise trakti relatsiooni „ajastub“:

#	B	C
	$\lessdot$	$\doteq$

#

Magasini tipmisele sümbolile *C* järgneb ($\gtrdot$) sisendrea aktiivne sümbol #. Seega, lause vormi baas on jada *BC* ning magasinis asendatakse see sümboliga *A* (vastavalt produktsioonile $A \rightarrow BC$); sümbol # ajastub *A*-ga, seegi relatsioon läheb magasini (2. trakti):

#	A
	$\doteq$

#

Relatsioon *A* ja # vahel on „ajastub“; analüsaatori algoritmi järgides on see lõppolek: etteantud sõna on aktsepteeritud. Teisisõnu, sõna *#ab#* kuulub keelde $\mathscr{L}(G1)$.

Lõpetagem see alapeatükk üleseletamisega, miks on oluline, et derivatsioon oleks tingimata kanooniline. Äsja selgitasime, et analüüs on derivatsiooni vastandoperatsioon, ja kui derivatsioon oleks juhuslik (s.o. igal sammul asendatakse suvaline mitteterminal), siis oleks analüsaatorit raske realiseerida. Kanoonilise derivatsiooni eelis kõigi muude variantide ees peitub seigas, et sel juhul on garanteeritud, et lause vormi baas on analüüsi ajal alati leitav magasini tipust alustades (operatsiooniga *pop* kuni relatsioonini „eelneb“).

Järgmises peatükis tutvustame analüüsi puu ehitamise tehnikat. Esmalt vaatame, kuidas genereeritakse täispuu, mis taastab üksüheselt derivatsiooni puu moodustamise käigu, ja et see tee on ilmselgelt liiane ja raiskav (demonstreerime selle väite paikapidavust), siis tutvume lihtsa tehnikaga analüüsi puu moodustamise käigu kontrollimiseks nii, et analüüsi puusse genereeritakse ainult need tipud, millel on translaatori jaoks hädavajalik roll[1].

Kordame üle: analüüsi peamine eesmärk on saada sõna (programmi) analüüsi puu, mille läbimist saab programmeerida kui tolle programmi *interpretaatorit*, mis teeb selle töö käigus resultaadid, või kui *kompilaatorit*, mis genereerib programmi täitmise koodi (*.exe-faili*).

Ja lisaks näeme, et interpretaatorit ja kompilaatorit pole mõtet vastandada, nad mõlemad on efektiivsed ning kasutuskõlblikud variandid programmeerimiskeele realiseerimiseks.

[1] Teeme „pügatud“ või „hõreda“ analüüsi puu.

4. Analüüsi puu moodustamine

4.1. Analüüsi puu

Eelmises peatükis käsitlesime *tuvastamist*, s.o. algoritmi, mis kas aktsepteerib etteantud stringi mingi grammatikaga antud keele sõnana või ei. Tuletame meelde: just nii on sõnastatud programmi (sõna) süntaksi analüüsi esimene alamülesanne: *Kas sõna* $x \in \mathscr{L}(G)$?

Teise alamülesande sõnastasime umbes nii: *Kui sõna kuulub keelde, siis ehita selle sõna (süntaktilise) analüüsi puu*. Analüüsi puu[1] ehitamise näide oli meil esitatud juba ülalpool, kui demonstreerisime grammatika *G*1 näitel derivatsiooni puu moodustamist (kordame, analüüsi puu on selline derivatsiooni puu, mille lehtede (rippuvate tippude) märgendid on terminalid ja kõikide mitterippuvate tippude märgendid on produktsioonid).

Praktilise analüüsi puus märgendame mitterippuvaid tippe mitte produktsioonidega, vaid nende produktsioonide vasakute poolte (mitteterminaalsete) sümbolitega, just nii, nagu toimime redutseerimisel: lause vormi baas $x \rightarrow$ *mõiste (*mitteterminal *A*, kui $A \rightarrow x \in P$)[2].

Analüüsi puu moodustamise idee on iseenesest lihtne; puu tipu formaati võime viitadele keskendudes kujutada nagu joonisel 4.1.a.

sümbol $\in V_T \cup V_N$	viit $\uparrow$
viit $\downarrow$	viit $\rightarrow$

Joonis 4.1.a. Analüüsi puu tipu abstraktne formaat.

Ülesviit ($\uparrow$) on väärtustatud vaid alluvahela (mille esimesele lülile ülemustipus viitab $\downarrow$) viimases lülis (kus väli $\rightarrow$ on Ø). Mõistagi võib alluvahel koosneda ka ainult ühest lülist (või olla tühi). Ülesviit[3] võimaldab läbida puud lõppjärjekorras ilma magasini ja rekursioonita (seda varianti on lihtne programmeerida ja töö kiirus on parem)[4]. Ja veel, väljal *sümbol* on nüüd mitterippuvas tipus produktsiooni asemel selle vasaku poole mitteterminal[5].

[1] (Keele)vaistlikult tahaks kirjutada „analüüsipuu“, ent see oleks eksitav: kokkukirjutatud termin vihjaks justkui analüüsi abivahendile, instrumendile analüüsimiseks, ent „analüüsi puu“ on selle analüüsi tulemus.
[2] Nii teeme oma õpikunäidetes, tegelik tipuformaat on pisut informatiivsem − aga sellest hiljem.
[3] Mõistagi, selline andmestruktuur pole enam *puu*, vaid *graaf*, ent „otstarve pühendab abinõu“.
[4] Vt. nt. [Isotamm 2009], lk. 172 jj.
[5] Redutseerimisel me nii toimimegi, magasini kirjutatakse „mitteterminal”. Translaator kasutab tipuformaati, kus informatiivne roll on semantika koodil, aga sellest tuleb juttu hiljem, siin mainime, et semantikat saab anda terminalidele ja produktsioonidele.

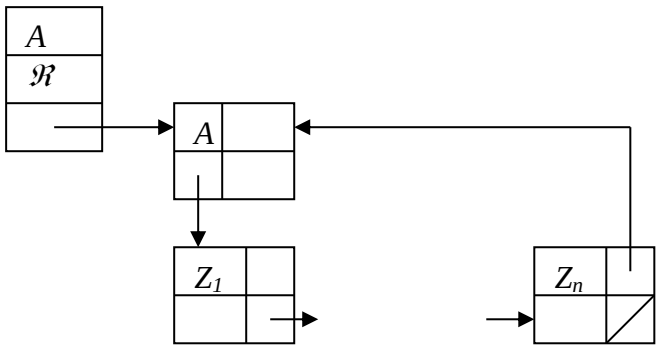
Analüüsi puu saame moodustada kas täisvariandis (nii taastame kanoonilise derivatsiooni tegeliku puu) või „hõredal“ (“pügatud”) kujul, genereerides ainult need puu tipud, millel on sisuline ehk semantiline kaal. Täisvariandi algoritm on järgmine:

- $a = T_i \in V_T$ kandmisega magasini kaasneb puu uue tipu X_a moodustamine; viit sellele tipule kirjutatakse magasini M kolmandasse trakti;
- pärast *detekteerimist* (s.o. lause vormi baasi $x=z_1,\ldots,z_n$ leidmist)[1] on meil olukord, mis on esitatud joonisel 4.1.b;

z_1	z_2	. . .	z_n
$\lessdot$	$\doteq$	. . .	$\doteq$
viit X_{z1}	viit X_{z2}	. . .	viit X_{zn}

Joonis 4.1.b. Magasini tipmises osas on lause vormi baas $z_1,\ldots,z_n$.

- lause vormi baasist viidatud analüüsi puu tipud ühendatakse alates tipust X_{z1} “$\rightarrow$”-viitade abil ahelasse (viit selle ahela peale on endiselt seotud magasinis z_1-ga);
- redutseerimise käigus (tuletagem meelde, siis asendatakse magasinis lause vormi baas $z_1,\ldots,z_n = x$ mitteterminaliga A ($A \rightarrow x \in P$)) moodustatakse uus puu tipp N_A (kus sümbol on A), sellele allutatakse ($\downarrow$-viida abil) ahel $X_{z1},\ldots,X_{zn}$ ja selle ahela viimasesse tippu kirjutatakse ülesviit tipule N_A ning tolle tipu N_A aadress kirjutatakse magasini 3. trakti kohale, kus esimeses traktis on sümbol A (joonis 4.1.c).

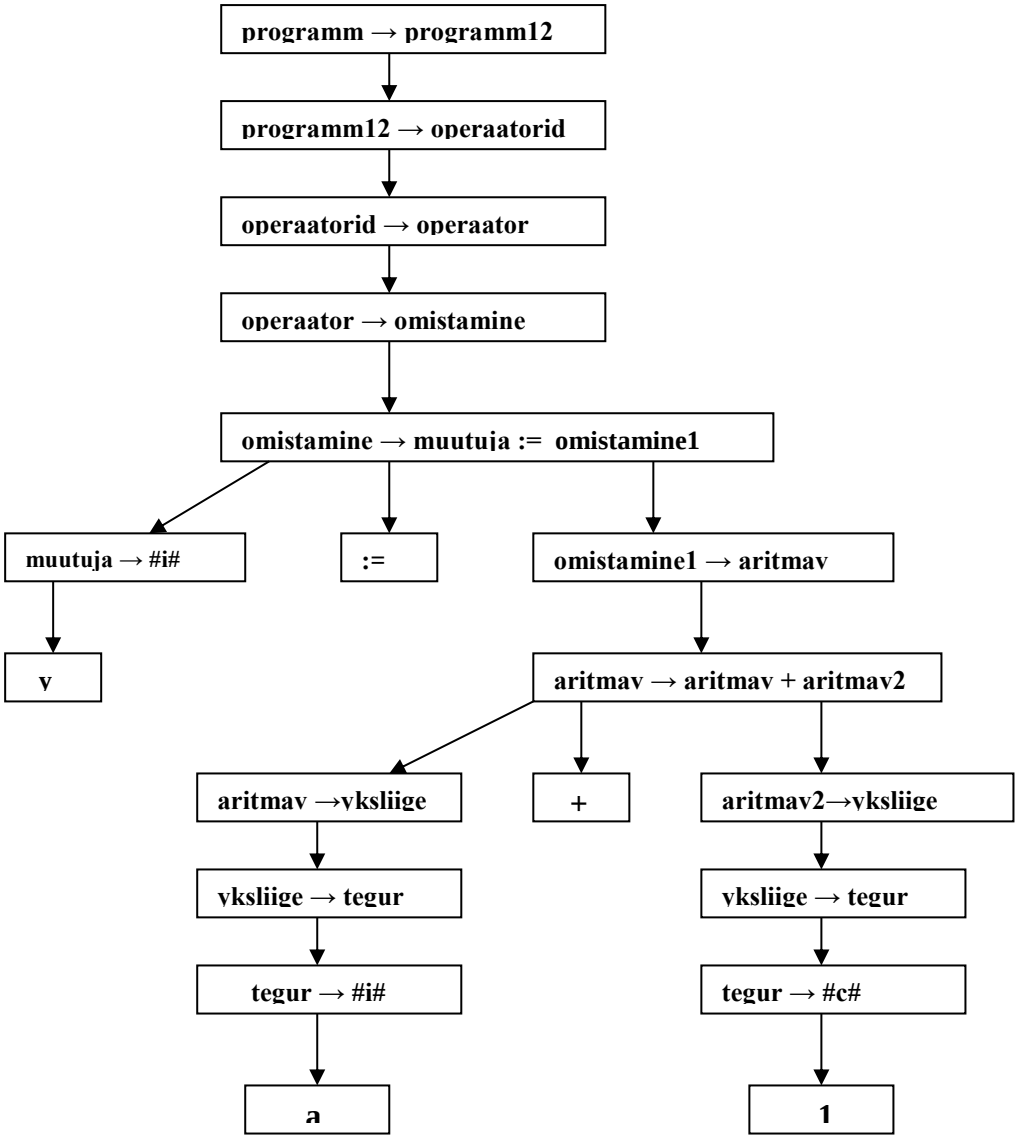


Joonis 4.1.c. Puu fragment pärast redutseerimist.

Täispuu on tugevasti liiane. Näiteks, üheoperaatorilise *Trigol*-programmi *y:=a+1* derivatsioon aksioomist *programm* on järgmine (lihtsuse mõttes on ära jäetud apostroofid mõistete ümber ja algus- ja lõpumarkerid ning on kasutatud grammatika *eelnevusteisendused läbinud versiooni* (vt. lisa 2, *TRI.GRM*)):

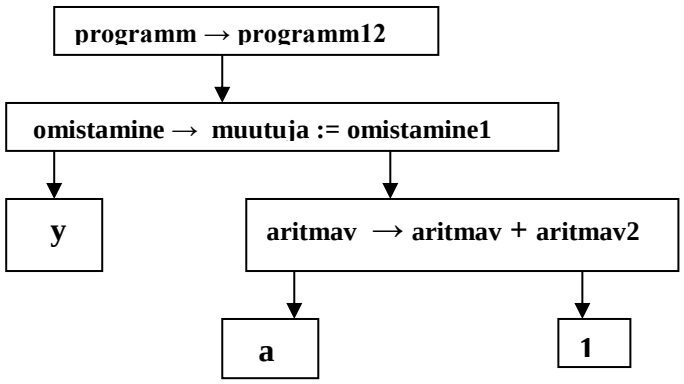
`programm` ⇒ `programm12` ⇒ `operaatorid` ⇒ `operaator` ⇒ `omistamine` ⇒
⇒ `muutuja` := `omistamine1` ⇒ `muutuja` := `aritmav` ⇒
⇒ `muutuja` := `aritmav` + `aritmav2` ⇒ `muutuja` := `aritmav` + `yksliige` ⇒
⇒ `muutuja` := `aritmav` + `tegur` ⇒ `muutuja` := `aritmav` + `#c#` ⇒
⇒ `muutuja` := `aritmav` + `1` ⇒
⇒ `muutuja` := `yksliige` + `1` ⇒ `muutuja` := `tegur` + `1`
⇒ `muutuja` := `#i#` + `1` ⇒ `muutuja` := `a` + `1` ⇒ `#i#` := `a` + `1` ⇒ `y`:=`a` + `1`

[1] z_i (i=1..n) on kas terminal a või mitteterminal (saadud redutseerimise tulemusena).



Joonis 4.1.d. Sõna *y:=a* + 1 derivatsiooni lõplik puu.

Analüüsi täielik puu on kujutatud joonisel 4.1.d ning liigsetest tippudest vabastatud puu on joonisel 4.1.e.



Joonis 4.1.e. Sõna *y:=a+1* analüüsi hõre puu.

4.2. Analüüsi puu moodustamise juhtimine

4.2.1. Semantikast

Analüüsi puu moodustamise kirjeldamisel kasutame terminit *semantika*, mõeldes *puusemantikat*: kas mingi terminaalse tähestiku sümbol või produktsioon produktsioonide hulgast *P* on programmi (sõna) analüüsi puus translaatori (interpretaatori või kompilaatori) jaoks müra või oluline. Suures plaanis tähistab semantika keele süntaktiliste konstruktsioonide tähendust, ekvivalentsisuhte loomist erinevate keelte süntaktiliselt erinevate konstruktsioonide vahel. Siin käsitletud puusemantika on nii triviaalne kui ka utilitaarne. See on efektiivne ja toimib. Ent süntaksorienteeritud transleerimismeetodite (mis panid aluse *TTS*-ile (*translaatorite tegemise süsteem*, ingl. *CC = Compiler Compiler*, vn. *СПТ = Система Построения Трансляторов*)) kiire ja edukas realiseerimine suunasid teoreetikud vähemalt aastakümneks (1970ndad) otsima meetodit, millega saaks produktsioonidega siduda kompileerimiseeskirjad (s.o. *päris-semantika*).

Sel teemal publitseeriti palju monograafiaid ja veel rohkem artikleid (ning kaitsti väitekirju). Huviline võiks vastavaid materjale otsida märksõnade *denotatsioonsemantika*, *W-grammatikad*, *Viini meetod* või *atribuuttehnika* alt, meie raamatu kaante vahele nad ei mahu eeskätt sellel põhjusel, et nad jäid praktilise väljundita. Tolle seiga peamine põhjus tundub olevat selles, et semantika on oluliselt abstraktsem (ja seega vähem formaliseeritav) mõiste kui süntaks, ja ligilähedaselt sama üldise meetodi kui *süntaksorienteeritud analüüs* realiseerimine osutus saavutamatuks. Ja teiseks, kuivõrd tol ajal olid masinad tavaliselt unikaalsed, siis semantikat püüti esitada masinorienteeritud keel(t)es ning otsitav väljund oli liiga detailne, isegi niivõrd, et masinorienteeritud semantika lisamine produktsioonidele osutus töömahukamaks kui vastava konkreetse (masinkoodi)kompilaatori kirjutamine.

Ent tõlkimisel keelest *A* keelde *B* (mis pole *üldine* variant), kui need mõlemad on protseduurorienteeritud keeled, õnnestus edukalt rakendada semantikat, mille abil sai automaatselt (s.o. programselt) genereerida keeles *A* kirjutatud programmi ekvivalent keeles *B* (*NB*! teksti tasemel). Näiteks sobib TÜs loodud translaator *Modula-2* → *Forth* (*M. Tombak, J. Pöial, V. Soo* jt.), kus esimese produktsioonidega seoti semantikana ekvivalentsed *Forth*-keele sõnad või konstruktsioonid (mis üldjuhul lisati laiendustena *Forth*-sõnastikku). Kui „normaalne“ *TTS*-tehnika näeb ette, et süsteemi vahendid garanteerivad programmide analüüsi puu ja puu interpreteerimine tuleb käsitsi programmeerida (kirjutada kas interpretaator või kompilaator), siis mainitud translaator väljastas perfektse *Forth*-teksti automaatselt (seda muidugi suuresti näiliselt, kuivõrd enamik semantikaks kasutatavaid sõnu tuli eelnevalt *Forth*is programmeerida) ja tihti olid nad spetsiaalselt *Modula-2* iseärasustele orienteeritud. Seega, sel juhul programmeeriti *semantika*, selmet programmeerida kompilaator.

4.2.2. Puusemantika

Puu moodustamist juhitakse meie versioonis nn. *semantiliste muutujate* abil ─ need on naturaalarvud σ_x ($\sigma_x \geq 0$), mis seotakse lihtsa keele abil kas produktsioonide või terminaalse tähestiku sümbolitega (meenutagem, et lekseemiklassid ─ näiteks *identifikaatorid* (*Trigoli grammatikas* tähistatud kui #i#) ja *konstandid* (#c#) ─ kuuluvad ka nende hulka).

Kui produktsiooni või terminali *x* korral $\sigma_x = 0$, siis vastavaid tippe analüüsi puusse ei lisata. Meile juba mõnevõrra tuttava keele *Trigol* semantika on toodud lisas 3. Selle esitamiseks on kasutatud lihtsat metakeelt; selgitame seda mainitud lisa abil. Semantikafaili kolm esimest rida on sellised:

```
4=1    $ #i#
11=2   $ #c#
p13=10  $ omistamine -> muutuja := omistamine1
```

Sümboliga *$* algab kommentaar, mis kehtib rea lõpuni, kusjuures too marker hakkab tööle alates esimesest esinemisest reas. Semantika omistamine toimub ridade alguses. Terminalidele semantika andmiseks viidatakse terminaalse tähestiku V_T elemendile selle järjekorranumbriga tähestikus V_T (need väljastab *Konstruktori* logi).

Niisiis, identifikaatorite klassi (#i# järjekorranumber terminalide seas on 4) elementide semantikakood on 1 (s.o. $\sigma_{\#i\#} = 1$) ning konstantidel (#c# jrk-nr on 11) on see 2 ($\sigma_{\#c\#} = 2$).

Produktsioonid nummerdatakse selles järjekorras, nagu nad on esitatud hulgas *P*. Metakeeles viidatakse produktsioonile kujul *Pi*, kus *i* on produktsiooni järjekorranumber ($\sigma_{P13} = 10$).

Analüüsi hõre puu moodustatakse põhimõtteliselt samuti nagu täispuu, ent nüanssides on erinevusi.

- $T_i \in V_T$ kandmisega magasini kaasneb puu uue tipu *X* moodustamine ainult siis, kui $\sigma_{T[i]} > 0$ (viit sellele tipule kirjutatakse magasini *M* kolmandasse trakti) ja muidu uut tippu ei tehta ning magasini 3. trakti element = Ø.
- Pärast *detekteerimist* (s.o. lause vormi baasi $x=z_1,\ldots,z_n$ leidmist) on meil näiteks olukord nagu joonisel 4.2.a.

z_1	z_2	. . .	z_n
⩽	≐	. . .	≐
viit $X_{z1} \vee$ Ø	viit $X_{z2} \vee$ Ø	. . .	viit $X_{zn} \vee$ Ø

Joonis 4.2.a. Magasinis on lause vormi baas (3. trakt viitab hõreda puu tippudele).

- Lause vormi baasist viidatud analüüsi puu tipud ühendatakse alates tipust X_{z1} "→"-viitade abil ahelasse (viit tolle ahela peale on endiselt seotud magasinis z_1-ga[1]) — mõistagi ainult siis, kui viit tipule pole Ø. Võib juhtuda, et tulemuseks on tühi ahel. Ja nüüdsel juhul peame alati kontrollima, kas ahelasse kantava tipu "→"- viit = Ø — vastasel korral tuleb ahelasse lisada tollest tipust algav alamahel (vt. järgmine samm, *redutseerimine*) ja ahela järgmine tipp lisatakse juba sellele alamahelale.
- *Redutseerimise* käigus, kui magasinis asendatakse lause vormi baas $z_1,\ldots,z_n = x$ mitteterminaliga *A*, kus (võimalik, et konteksti arvestades) kehtib $A \rightarrow x$. Kui $\sigma_{A\rightarrow x} > 0$, siis moodustatakse uus puu tipp N_A (kus sümbol on *A*), sellele allutatakse (↓-viida abil) ahel $X_{z1},\ldots,X_{zn}$[2] ja selle ahela viimasesse tippu kirjutatakse ülesviit tipule N_A ning tolle tipu N_A aadress kirjutatakse magasini kolmandasse trakti kohale, kus esimeses

[1] Ka siis, kui z_1 enda puuviit on tühi; lause vormi baasi esimese elemendiga seotakse alati (mittetühja) ahela esimene lüli.

[2] See ahel võib olla tühi ja sel juhul on ka ahelala viit tühi. Ning mõistagi mingeid muidki viitu ei ole.

traktis on sümbol *A*. Kui aga $\sigma_{A\to x} = 0$, siis tippu N_A ei tehta ning magasinis seotakse sümboliga *A* ahel $X_{z1},...,X_{zn}$.

Toome paar näidet, tuginedes meile juba tuttavale abstraktsele grammatikale *G1*:

```
S →# A #
A →B C
B →a
B →B a
C →b
```

Eelnevusmaatriks on järgmine:

	#	a	b	A	B	C
#		⋖		≐	⋖	
a		⋗	⋗			⋗
b	⋗					
A	≐					
B		≐	⋖			≐
C	⋗					

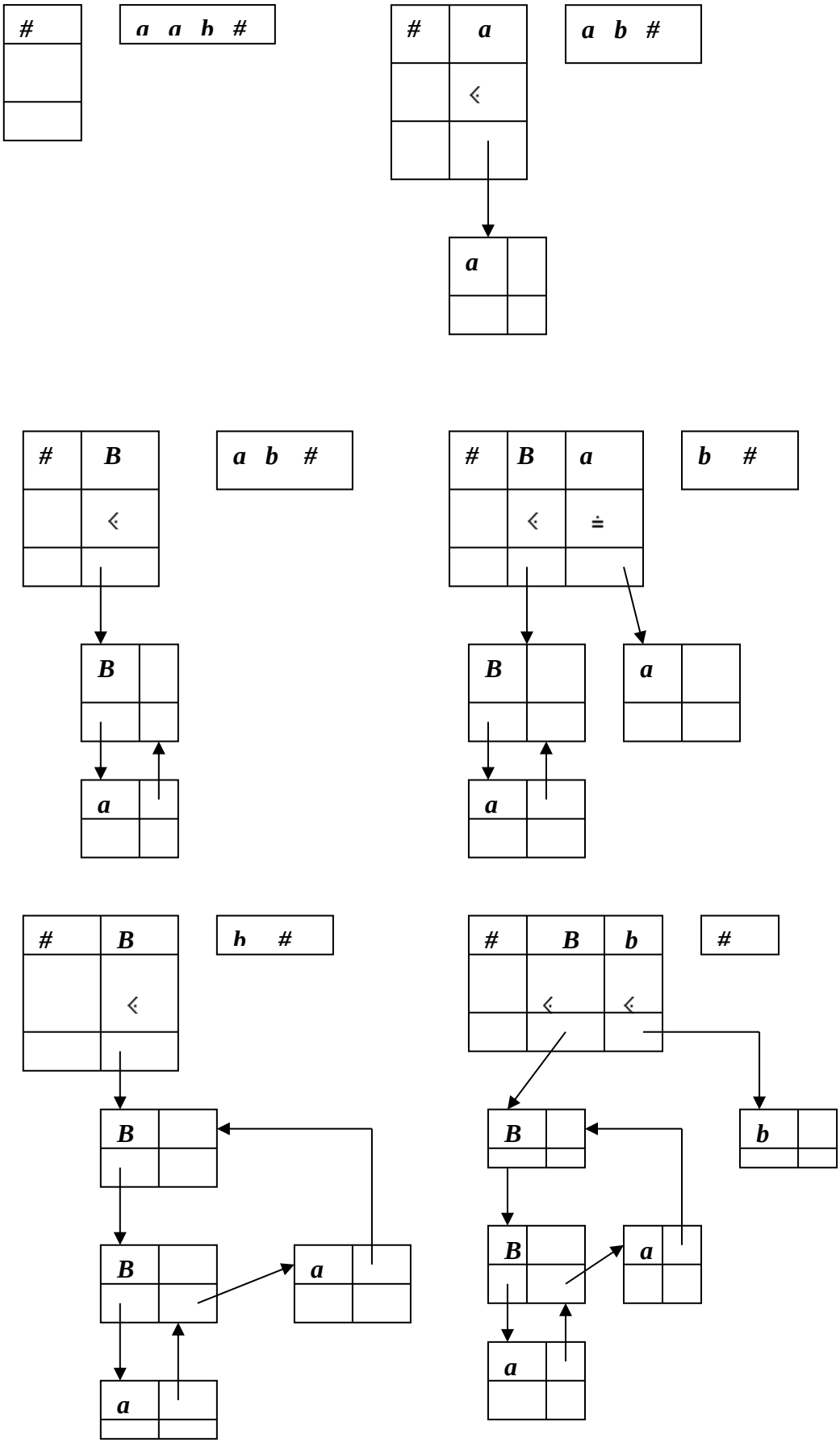
Joonis 4.2.b. G1 eelnevusmaatriks.

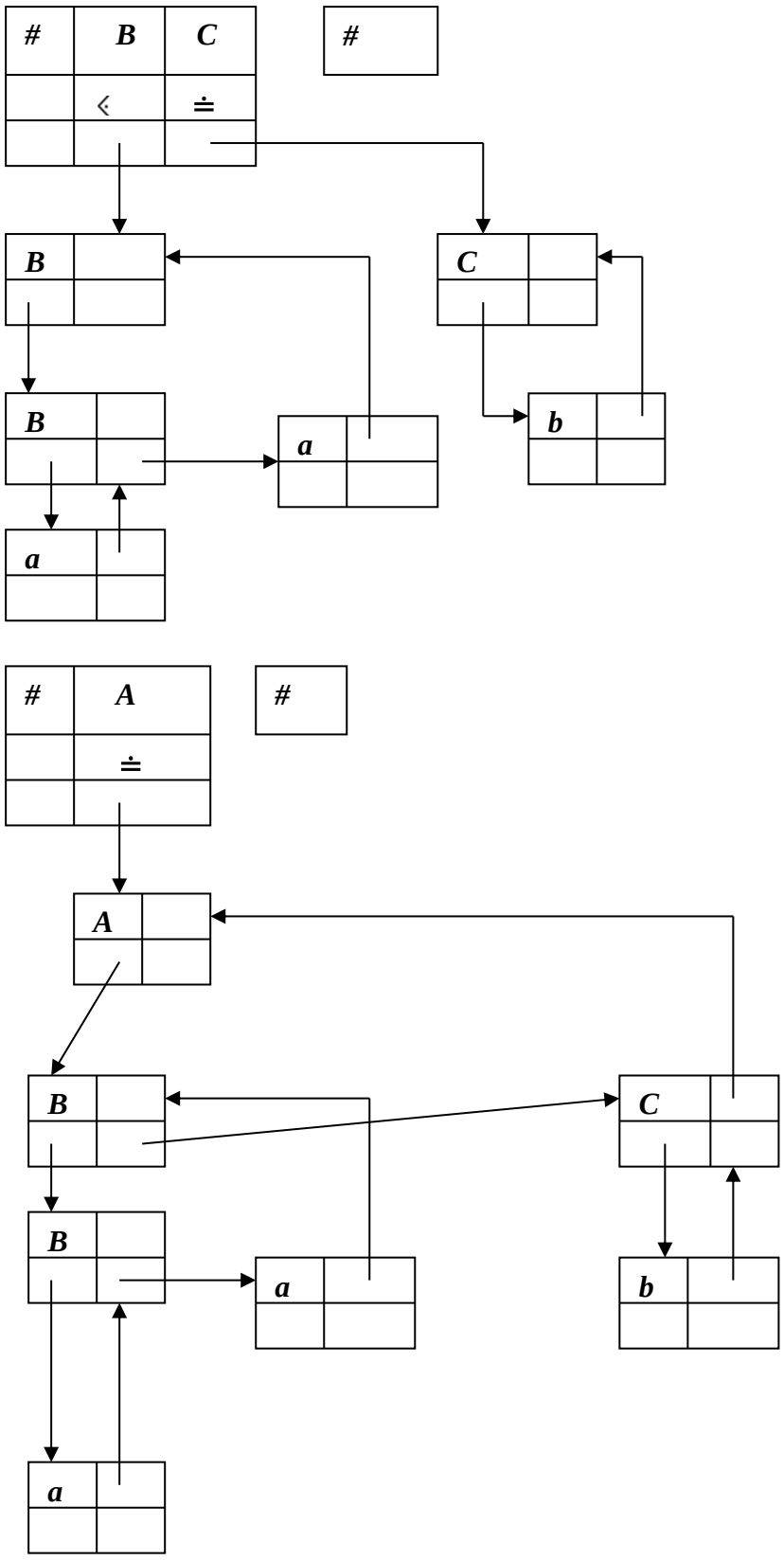
Analüüsime sõna #aab#, kusjuures esmalt teeme täispuu: semantika on kõigil terminalidel peale # ja kõigil produktsioonidel. Analüüsi kulg on nähtav joonisel 4.2.c, mida tuleb lugeda vasakult paremale ja ülalt alla (s.o. ridahaaval), episoodide kaupa.

Iga episood koosneb põhimõtteliselt kolmest komponendist: vasakus ülanurgas on *Analüsaatori* kolmetraktiline magasin ja sellest paremal sisendrida ─ analüüsitava sõna (veel) magasini kandmata sümbolid (*T[i],...,T[n]=#, T[0]=#).* Magasini kolmandas traktis on (kui puusemantika nõuab) viit antud magasinielemendiga seotud tipule sõna analüüsi puus (või Ø).

Episoodid on dikteeritud *Analüsaatori* algoritmi poolt. Kui $T[i] \Rightarrow M$, siis vastavas episoodis on näidatud magasini *M* uus seis, kui aga magasini olek muutus redutseerimise käigus, siis seis pärast kõiki asjakohaseid tegevusi (võimalik puutippude aheldamine ning lause vormi baasi *x* asendamine mitteterminaliga *A* ($A\to x\in P$).

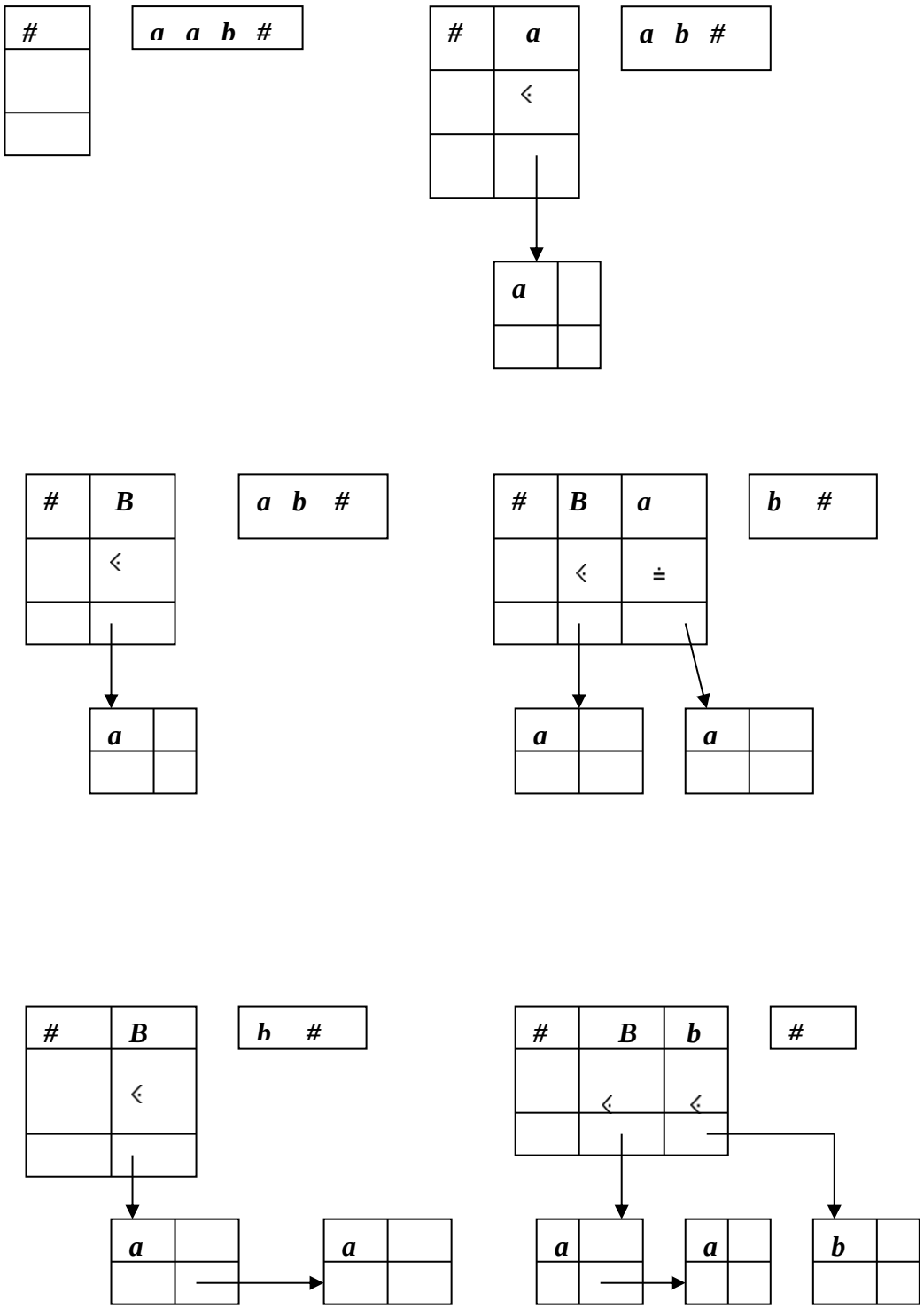
Niisiis, sõna #aab# analüüs (esmalt täis- ja siis pügatud puuga) − vt. jooniseid 4.2.c ja 4.2.d allpool.

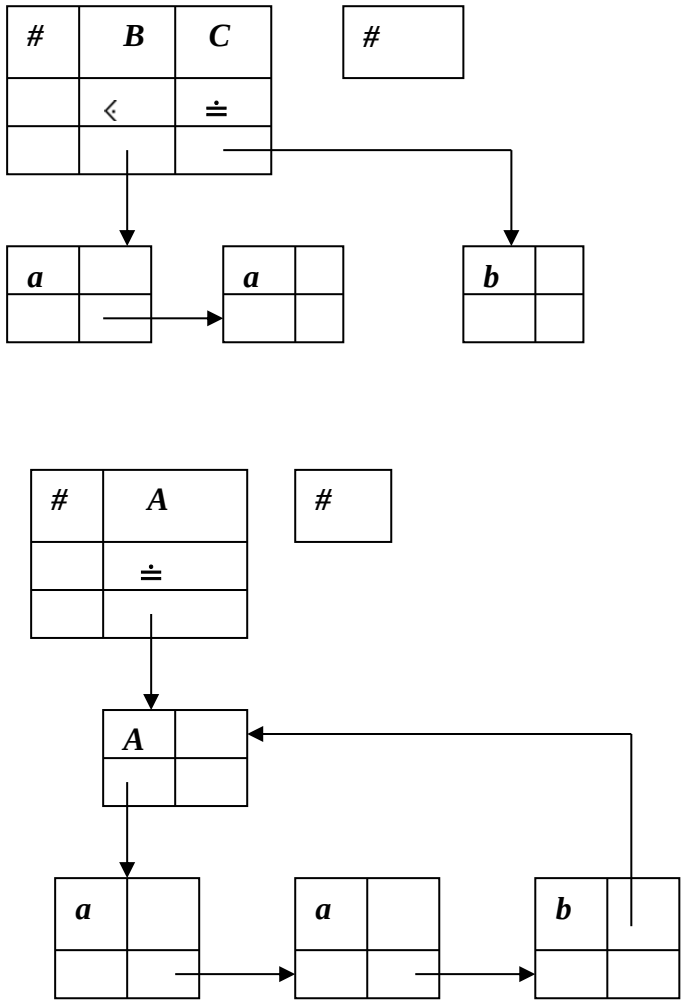

#
a a b #
a
a b #
a
B
a b #
B
a
B a
b #
B
a
a
B
b #
B
B
a
a
B b
#
B
B
a
a
b



Joonis 4.2.c. Sõna #aab# analüüsi täispuu.

Olgu „puusemantika” terminalidel *a* ja *b* ning produktsioonil $A \rightarrow B\ C$. Hõreda puu moodustamine on jälgitav joonisel 4.2.d.


#
a a b #
#
a
a b #
a
#
B
a b #
#
B
a
b #
a
a
a
#
B
b #
#
B
b
#
a
a
a
a
b



Joonis 4.2.d. Sõna #aab# analüüsi hõreda puu moodustamine.

5. Konteksti kasutav analüüs

Me oskame analüüsida *pööratavate eelnevusgrammatikatega* defineeritud keelte *sõnu*: nois grammatikates on kõikide produktsioonide paremad pooled unikaalsed. Paraku, me võime grammatika defineerida küll *pööratavana*, ent raske on grammatikat defineerida nii, et ta oleks garanteeritult *eelnevusgrammatika*, ja eelnevusteisenduste käigus juhtub sageli, et lisanduvate produktsioonide (definitsioonide) paremad pooled pole kõik enam unikaalsed. Formaalselt esineb situatsioon

$A \to x$ ja $B \to x$ $(x \in V^*)$.

Ning küsimus on, kummaks mõisteks tuleb redutseerida lause vormi baas *x*. Peab see olema *A* või *B*? Tähistagem neid mitteterminale (*A* ja *B*) koos tundmatuna: *X*.

Lause vormi baas *x* on analüüsi ajal magasinis ning analüsaator peab selle asemele kirjutama mitteterminali *X*. Seejuures, eelmise magasinis oleva sümboli *L* relatsioon on *x* algusega alati „eelneb" ning toda baasi hakkame magasinist otsima siis, kui *x* lõpusümboli relatsioon „lugemispea" ees oleva aktiivse lekseemiga (loomulikult terminali T_i) on „järgneb", need võimalikud terminalid T_i moodustavad *X parema konteksti RC*(*X*) *hulga* ja (teoreetiliselt) võimalikud sümbolid L_j on tolle mitteterminali *X vasaku konteksti LC*(*X*) hulk. See tõdemus annab võimaluse arvutada mitteterminali *X* paremaid ja vasakuid kontekste eelnevusmaatriksit kasutades; nimetagem seda tehnikat *sõltumatu*[1] konteksti arvutamiseks.

5.1. Sõltumatu kontekst

Kui produktsioonide hulgas on kaks produktsiooni kujul $A \to x$ ja $B \to x$ ($x \epsilon V^*$), siis tekib redutseerimisprobleem: kas analüüsil tuleb asendada lause vormi baas *x* mitteterminaliga *A* või *B*. Sel juhul saab kasutada mitteterminali *piiratud kanoonilist konteksti BRC*(1,1): analüüsisammu määravad üheselt lause vormi baas *x*, üks sümbol temast vasakult ja üks sümbol paremalt (*BRC* on ingliskeelse termini *Bounded Right Context* akronüüm). Formaalselt:

kui $S \overset{*}{\underset{\kappa}{\Rightarrow}} X_1, X_2, \ldots, X_k A T_1, \ldots, T_n \underset{\mathbf{k}}{\Rightarrow} X_1, X_2, \ldots, X_k Z_1, \ldots, Z_m T_1, \ldots, T_n$ $(T_i \in V_T)$, siis:

$X_k \lessdot A \vee X_k \doteq A$ ja

$A \lessdot T_1 \vee A \doteq T_1 \vee A \gtrdot T_1$.

Moodustatakse mitteterminali *A* vasaku ja parema konteksti hulgad *LC*(*A*) ja *RC*(*A*)[2]:

$LC(A) = \{X : [X \lessdot A \vee X \doteq A] \,\&\, X \in V\}$;

$RC(A) = \{T : [A \lessdot T \vee A \doteq T \vee A \gtrdot T] \,\&\, T \in V_T\}$;

$C_{1|1}(A) = LC(A) \times RC(A)$[3].

Hulk $C_{1|1}(A)$ on mitteterminali *A* (1|1)-sõltumatu kanooniline kontekst (loe „üks-kriips-üks"). Kui $A \to x$ ja $B \to x$ ning kehtib $C_{1|1}(A) \cap C_{1|1}(B) = \varnothing$, siis ütleme, et *G* on 1|1-*redutseeritav eelnevusgrammatika*[1]. Näiteks toome grammatika *G7*:

---

[1] *Derivatsioonist* sõltumatu; kasutame ainult eelnevusmaatriksit, kus on sümbolitevahelised *formaalsed* relatsioonid.

[2] Vasak kontekst on analüüsi ajal alati magasinis ja magasini elementide vahel pole kunagi relatsiooni „järgneb". Parem kontekst on alati sisendrea momendi vasakpoolseim sümbol (aktiivne terminal). Nii hea seisu tagab *kanooniline derivatsioon.*

[3] Tehe on mitme nimega. See on kas otsekorrutus, *Cartesius*e või *Descartes*'i ristkorrutus. *Renatus Cartesius* on prantsuse matemaatiku *René Descartes*'i (31.03.1596–11.02.1650) nime tollal tavaks olnud latiniseerimise tulem [wRD]. Tehte resultaadiks on operandhulkade elementide kordusteta paaride hulk.

```
T → # S #
S → a A
S → b B
A → 0 A 1
A → C 1
B → C 1
C → 1
```

See grammatika *pole pööratav*, kuivõrd nii *A* kui ka *B* on defineeritud samamoodi: *C1*. Vaatame, kas tegemist on eelnevusgrammatikaga, ja kui on, siis kas ta on *1|1*-redutseeritavav. Alustame mõistete (mitteterminalide) *L*- ja *R*-hulkade moodustamisega ja jätkame eelnevusmaatriksiga:

L-R-hulgad

	L( )	R( )
T	#	#
S	a, b	A, B, 1
A	0, C, 1	1
B	C, 1	1
C	1	1

Eelnevusmaatriks

	#	a	b	0	1	S	A	B	C
#		⋖	⋖			≐			
a				⋖	⋖		≐		⋖
b					⋖			≐	⋖
0				⋖	⋖		≐		⋖
1	⋗				⋗				
S	≐								
A	⋗				≐				
B	⋗								
C					≐				

Nentigem, et *G7* on eelnevusgrammatika. Järgmise sammuna leiame mitteterminalide *A* ja *B* (kuivõrd nad on samamoodi defineeritud) sõltumatute kontekstide hulgad:

$LC(A)=\{a,0\}$, $RC(A)=\{\#,1\}$ ja $C_{1|1}(A)=\{(a,\#),(a,1),(0,\#),(0,1)\}$;
$LC(B)=\{b\}$, $RC(B)=\{\#\}$ ja $C_{1|1}(B)=\{(b,\#)\}$.

Kuivõrd $C_{1|1}(A) \cap C_{1|1}(B) = \varnothing$, võtame teadmiseks, et *A* ja *B* derivatsioonist sõltumatud kontekstid *eristuvad* ja *G7* on 1|1-redutseeritav eelnevusgrammatika.

Genereerime kaks keele $\mathscr{L}(G7)$ sõna: esimese, kasutades produktsiooni $A \rightarrow C1$, ja teise, kasutades $B \rightarrow C1$:

$T \Rightarrow \#S\# \Rightarrow \#aA\# \Rightarrow \#aC1\# \Rightarrow \#a11\#$ ja
$T \Rightarrow \#S\# \Rightarrow \#bB\# \Rightarrow \#bC1\# \Rightarrow \#b11\#$.

[1] Loe: „üks-kriips-üks-redutseeritav“…

Mängime nende sõnade analüüsid läbi; kasutame seejuures lihtsustatud pilti –: me ei näita magasini „puutrakti“, sümbolid ja relatsioonid paigutame ühte ritta ning magasini piirina kasutame sümbolit „|“: sellest vasemal on magasin ja paremal sisendrida. Niisiis, esimese sõna analüüs:

```
# ⋖ a |1 1 #
# ⋖ a ⋖ 1 |1 #
# ⋖ a ⋖ C | 1 #
# ⋖ a ⋖ C ≐ 1 | #
```

Järgmisel sammul leiame, et 1 ⋗ #, seega magasini tipus on lause vormi baas *C* 1 ning peame otsustama, kas see tuleb redutseerida mitteterminaliks *A* või *B*. Valitava mitteterminali vasaku konteksti sümbol on *a* ning et $a \in LC(A) = \{a, 0\}$ ja $a \notin LC(B)$, siis valime *A*:

```
# ⋖ a ≐ A | #
```

Et A ⋗ #, siis järgmine lause vorm on #S# ja analüüs on lõppenud tõdemusega, et etteantud sõna kuulub keelde $\mathscr{L}(G7)$. Analüüsigem ka teist sõna:

```
# ⋖ b |1 1 #
# ⋖ b ⋖ 1 |1 #
# ⋖ b ⋖ C | 1 #
# ⋖ b ⋖ C ≐ 1 | #
```

Siingi leiame, et 1 ⋗ #, seega magasini tipus on lause vormi baas „*C* 1“ ning peame taas otsustama, kas see tuleb redutseerida mitteterminaliks *A* või *B*. Valitava mitteterminali vasaku konteksti sümbol on *b* ning et $b \in LC(B)$ ja et $LC(A) = \{a, 0\}$, siis valime *B*:

```
# ⋖ b ≐ B | #.
```

Et B ⋗ #, siis järgmine lause vorm on #S# ja analüüs on lõppenud tõdemusega, et ka see etteantud sõna kuulub keelde $\mathscr{L}(G7)$.

5.2. Sõltuv kontekst

Kui $A \to x$ ja $B \to x$ ning kehtib $C_{1|1}(A) \cap C_{1|1}(B) \neq \varnothing$, siis *G* pole 1|1-redutseeritav eelnevus-grammatika ning paljudel juhtudel aitab *derivatsioonist sõltuv* kontekst[1] (kasutatakse reaal-seid võimalikke derivatsioone, sealjuures neid tegelikult läbi tegemata). Vaatame näiteks järgmist grammatikat (*G8*):

```
T → # S #
S → a A a
S → b A b
S → a B b
S → b B a
A → 1
B → 1
```

A ja *B* sõltumatu kontekst kattub täielikult: $C_{1|1}(A) = C_{1|1}(B) = \{(a,a),(a,b),(b,a),(b,b)\}$.

[1] Sõltuva konteksti autor on *Mati Tombak* (vt. [Tombak, 1980]). $C_{1,1}(A) \subset C_{1|1}(A)$. $C_{1|1}(A) \setminus C_{1,1}(A)$ on derivatsioonidega saavutamatu kontekst.

Ent keeles $\mathscr{L}(G8)$ on ainult 4 sõna:

$T \Rightarrow \#S\# \Rightarrow \#aAa\# \Rightarrow \#a1a\#;$

$T \Rightarrow \#S\# \Rightarrow \#bAb\# \Rightarrow \#b1b\#;$

$T \Rightarrow \#S\# \Rightarrow \#aBb\# \Rightarrow \#a1b\#;$

$T \Rightarrow \#S\# \Rightarrow \#bBa\# \Rightarrow \#b1a\#.$

Näeme, et *A* tegelik kontekst on {(*a, a*),(*b,b*)} ning *B* kontekst on {(*a,b*),(*b,a*)}.

Mitteterminali *A* derivatsioonist *sõltuv kontekst* $C_{1,1}(A)$ on defineeritud järgmiselt:

$$C_{1,1}(A)=\{(X,T):S \overset{*}{\underset{\kappa}{\Rightarrow}} uXATv \ \&\ Tv\in V_T^*\}.$$

$C_{1,1}(A)$ arvutuseeskiri on järgmine:

$$C_{1,1}(A)=\gamma_1(A)\cup\gamma_2(A)\cup\gamma_3(A)\cup\gamma_4(A);$$
$$\gamma_1(A)=\{(X,T):B\rightarrow uXADv\in P\&[T=Dv\vee T\in L(D)]\};$$
$$\gamma_2(A)=\{(X,T):B\rightarrow uXA\in P\&T\in RC(B)\};$$
$$\gamma_3(A)=\{(X,T):B\rightarrow ADv\in P\&X\in LC(B)\&[T=Dv\vee T\in L(D)]\};$$
$$\gamma_4(A)=\{(X,T):B\rightarrow A\in P\&(X,T)\in C_{1,1}(B)\}.$$

Kui $A\rightarrow x$ ja $B\rightarrow x$ ning kehtib $C_{1,1}(A)\cap C_{1,1}(B)=\emptyset$, siis ütleme, et *G* on *1,1-redutseeritav eelnevusgrammatika*[1].

Järgnevalt kommenteerime derivatsioonist sõltuva konteksti alamhulkade γ_i (i=1..4) leidmist.

Hulgaga γ_1(A) puhul probleeme pole: produktsiooni paremas pooles on mitteterminalil *A* nii vasem kui ka parem naaber, mis moodustavadki vasaku ja parema konteksti sümbolite paari juhul, kui parempoolne naaber *D* on terminal. Kui *D* on mitteterminal, siis paremasse konteksti kuuluvad tolle mitteterminali *leftmost*-hulga kõik *n* terminaalset elementi ($n\geq1$), ja nii saame *n* paari.

Hulga γ_2(A) puhul on vasak kontekst samuti ilmutatud kujul produktsiooni paremas pooles antud, ent parema konteksti moodustavad produktsiooni vasakus pooles oleva (defineeritava) mitteterminali *B* parema konteksti *n* elementi (saame *n* paari). NB! Mõeldud on *B* sõltumatut paremat konteksti (mis saadakse eelnevusmaatriksist ja rekursiooni ei ole).

Hulga γ_3(A) parema konteksti hulga sümbolid saame samuti nagu γ_1(A) puhul ning vasaku konteksti sümbolid nagu γ_2(A) puhul, selle erinevusega, et kasutame *B* sõltumatu vasaku konteksti hulga elemente.

Hulga γ_4(A) leidmise programmeerimisel tuleb olla ettevaatlik, kuivõrd meid huvitava mitteterminali *A* sõltuva konteksti arvutamiseks peame leidma produktsiooni vasaku poole sümboli *B* sõltuva konteksti hulga; teoreetiliselt on võimalik „surmahaarde“ situatsioon: *B* sõltuva konteksti leidmine võib nõuda mingi teise mõiste sõltuva konteksti leidmist jne ning ringiga

[1] Loetakse: “üks-koma-üks-redutseeritav…”.

võime jõuda seisuni, kus vajame juba jälle *A* sõltuvat konteksti[1]. Testime grammatikat *G*8 (vt. üle-eelmine lk.):

	L( )	R( )
T	#	#
S	a, b	a, b
A	1	1
B	1	1

	#	a	b	1	S	A	B
#		⋖	⋖		≐		
a	⋗			⋖		≐	≐
b	⋗			⋖		≐	≐
1		⋗	⋗				
S	≐						
A		≐	≐				
B		≐	≐				

$LC(A)=\{a,b\}$; $RC(A)=\{a,b\}$; $C_{1|1}(A)=\{(a,a),(a,b),(b,a),(b,b)\}$;
$LC(B)=\{a,b\}$; $RC(B)=\{a,b\}$; $C_{1|1}(B)=\{(a,a),(a,b),(b,a),(b,b)\}$ ja
$C_{1|1}(A)=C_{1|1}(B)$.

Niisiis, *A* ja *B* sõltumatu kontekst on sama. Arvutame nende mitteterminalide sõltuvad kontekstid. Nii *A* kui ka *B* sõltuv kontekst koosneb ainult komponendist γ_1:

$C_{1,1}(A)=\gamma_1(A)=\{(a,a),(b,b)\}$ ja $C_{1,1}(B)=\gamma_1(B)=\{(a,\ b),(b,a)\}$.

Ja et nood hulgad ei lõiku, siis nentigem, et *A* ja *B* (derivatsioonist) sõltuvad kontekstid eristuvad ning *G8* on *1,1-redutseeritav eelnevusgrammatika.* Analüsaator käitub samuti nagu sõltumatut konteksti kasutaval juhul: kui vasak kontekst eristub, siis redutseeritakse, kui mitte, siis kasutatakse paremat konteksti ja redutseeritakse.

Analüüsigem sõna: *#a1a#* (jaotise 4.1 lõpus kasutatud simulatsiooni järgides):

$\#\ \lessdot\ a\ \lessdot\ 1|\ a\ \#$.

Kehtib relatsioon $1 \gtrdot a$, seega „1“ on lause vormi baas – aga see on nii *A* kui ka *B* definitsioonide paremaks pooleks. Sümbol *a* kuulub nii *A* kui ka *B* vasakusse konteksti, ent paar (*a, a*) on ainult mitteterminali *A* sõltuva konteksti hulgas – seega, uus lause vorm on *#aAa#*. Ja sõna *#b1a#* analüüs viib vormini *#bBa#*.

5.3. Konteksti lisamine

Siiski, võib juhtuda, et ka derivatsioonist sõltuv kontekst ei eristu. Kui siiani kõik lisatööd (eelnevuskonfliktide likvideerimine, sõltumatu ja sõltuva konteksti arvutamine) on *TTS*i plokki *Konstruktor* „sisse programmeeritud“, siis antud juhul on jätkamise võimaluste otsimine ja leidmine problemaatiline[2]. Üldjuhul tuleb sellises situatsioonis muuta keelt (mõistagi süntaksit), ent on teada vähemalt üks tehnika, mis võib (keelt muutmata) aidata (aga ei pruugi

[1] Ilmselt pole surmahaare võimalik *ühese* grammatika puhul; ülalpool nentisime, et üldiselt pole teada algoritmi, mis tuvastaks, kas grammatika *on* ühene või mitmene, ja et grammatika *pole* garanteeritult ühene, kui tema produktsioonide hulk võimaldab tühisõna kasutamist või kui seal esineb tsükkel kujul $A \rightarrow B,\ B \rightarrow C,\ldots X \rightarrow A$. Just selline tsükkel võib viia surmahaardeni, kui teda on vaja hulga γ_4 arvutamiseks. *Konstruktor* kui programm peab surmahaardeks valmis olema, ent sellise olukorra põhjustab alati inimene, kes kirjutab grammatika.

[2] Siiani pole teada algoritmi, mis alati garanteeriks oodatud tulemuse.

seda teha). See võimalus pole *Konstruktoris* (garanteeritud resultaadi puudumisel) realiseeritud, seega tuleb seda „käsitsi“ proovida, modifitseerides ise produktsioonide hulka.

Teisisõnu: inimene saab kasutada mõningaid läbiproovitud võtteid, ent tulemus pole üldjuhul ennustatav. Siin on mitmeid intuitiivseid tehnikaid, tutvustame neist üht võtet nimetusega “*vasak faktoriseerimine*”[1].

Idee seisneb selles, et kui vasak kontekst (s.o. analüüsi ajal lause vormi baasile magasinis eelnev sümbol X) on mitme (n, $LC(A_i)= X$, $i=1..n$, $X \epsilon V$) mitteterminalidest A_i „konkurendi“ puhul sama, siis lisatakse produktsioonide hulka uued definitsioonid $B_i \rightarrow X$ ja korrigeeritakse originaalproduktsioone. Toome ühe eduka näite (*edu* sõltub sellest, kas õnnestub lahku lüüa vasakut konteksti). Illustreerime seda tehnikat grammatika *G41* näite (allpool on TTSi logi) abil:

```
P 1: `T' -> # `S' #
P 2: `S' -> a `A'
P 3: `S' -> b `B'
P 4: `A' -> 0 `A' 1
P 5: `A' -> 0 1
P 6: `B' -> 0 `B' 1 1
P 7: `B' -> 0 1 1
```

Ilmselt on tegemist kontekstivaba grammatikaga; ent paraku pole ta eelnevusgrammatika ning pärast eelnevuskonfliktide likvideerimist selgub ka, et tegemist pole pööratava eelnevusgrammatikaga, ja lõpuks selgub, et ei sõltumatu ega ka sõltuv kontekst ei eristu (vastavad testid palume lugejal endal teha); pärast eelnevusteisendusi on grammatika järgmine:

```
P 1: `T' -> # `S' #
P 2: `S' -> a `A'
P 3: `S' -> b `B'
P 4: `A' -> 0 `A' 1
P 5: `A' -> 0 1
P 6: `B' -> `B2' 1
P 7: `B' -> `B1' 1
P 8: `B1' -> 0 1
P 9: `B2' -> 0 `B' 1
```

*TTS*i logi lõpp on järgmine:

independent context didn’t help us. I’ll try to use the dependent one. (sõltumatu kontekst ei eristu)
I’ll find the subsets of dependent context of `A’ (leian *A* sõltuva konteksti)
gamma1: the source is the production
`P=4 `A' -> 0 `A' 1`
`{0,1}`
gamma2: the source is the production
`P=2 `S' -> a `A'`
`{a,#}`
The set of dependent context of `A’:
`{a,#} {0,1}`
I’ll find the subsets of dependent context of `B1’ (leian *B*1 sõltuva konteksti)

[1] Seegi võte pärineb *Mati Tombak*ult. *Faktoriseerimine* on produktsiooni parema poole ositamine (ja uu(t)e mõiste(te) lisamine), midagi analoogilist eelnevuskonfliktide likvideerimise stratifikatsioonidele.

```
gamma3: the source is the production
P=7 `B' -> `B1' 1
{b,1} {0,1}
The set of dependent context of `B1':
{b,1} {0,1}
test_dep_con A and B1
common: {0 , 1}
dependent context is not different: A and B1
```
(*A* ja *B*1 sõltuvad kontekstid ei eristu)

Niisiis, ka sõltuv kontekst ei eristu. Nii *A* kui ka *B1* vasakus kontekstis on terminal "0"; vasaku faktoriseerimise ideed järgides tuleb kriitilistes punktides kasutada umbes sama tehnikat nagu *P1*-eelnevuskonfliktide likvideerimisel, ent defineerides seejuures nood terminalid "0" erinevalt – nii, nagu on tehtud allpool (vt. kaht viimast produktsiooni):

```
`T'->#`S'#
`S'->a`A'
`S'->b`B'
`A'->`C'`A'1
`A'->0 1
`B'->`D'`B'1 1
`B'->0 1 1
`C'->0
`D'->0
```

See grammatika pole eelnevusgrammatika (soovitame lugejal selles veenduda ning teha eelnevusteisendused); pärast eelnevusteisendusi, mille käigus lisandus uus produktsioon

```
`B1' → 0,
```

on *A* ja *B* sõltumatute kontekstide hulgad järgmised:

```
LC(A)={a,C}; RC(A)={#,1}; C1|1(A)={(a,#),(a,1),(C,#),(C,1)};
LC(B1)={b,D}; RC(B1)={1}; C1|1(B1)={(b,1),(D,1)}.
```

Mitteterminalide *A* ja *B*1 sõltumatud kontekstid eristuvad. Teine uurimist vajav paar on $C \rightarrow 0$ ja $D \rightarrow 0$:

```
LC(C)={a,C}; RC(C)={0}; C1|1(C)={(a,0),(C,0)};
LC(D)={b,D}; RC(D)={0}; C1|1(D)={(b,0),(D,0)}
```

ja ka mitteterminalide *C* ja *D* sõltumatud kontekstid eristuvad. Niisiis, grammatika *G41.grm* on *BRC*(1|1)-redutseeritav eelnevusgrammatika. Ja tõdegem, et praegusel juhul oli *vasakust faktoriseerimisest* abi.

Ülalpool osutasime tõigale, et see võte ei pruugi olla alati tulemuslik. Vaadeldud näites oli parem kontekst „paigas" ja vasaku konteksti lisamine („juurde toomine") oli võimalik; kui see nii aga ei ole, siis tuleb tavaliselt muuta keelt (defineerides konfliktseid mõisteid teisiti).

Kommenteerigem äsjakäsitletud tehnikat. Mida me ka ei teinud, ei saanud me lahti kontekstipaarist *(0,1)*. Produktsioone vaadates on selge, et meie probleemide põhjused on kätketud neisse, kus produktsiooni paremas pooles on paar *(0, mitteterminal):*

```
P 4: `A' -> 0 `A' 1
P 6: `B' -> 0 `B' 1 1
```

Miks nii: vaatame derivatsioone, kus kasutame teid, mis viivad *P4* või *P6*-ni:

```
T⇒#S#⇒#a A#⇒#a0A1#⇒... ja
T⇒#S#⇒#b B#⇒#b0B11#⇒...
```

Ilmselt võib vasaku konteksti „0“ olla pärit just nendest *P4* ja *P6*-st (*P5* ja *P7* ei tule arvesse, kuivõrd nende paremates pooltes olid „0“ ja „1“ vahetud naabrid, nende vahel pole mitteterminale). Ja pärast eelnevusteisendusi näeme, et too vasaku konteksti „0“ käib ikka ja alati kaasas. Niisiis, püüame vasaku konteksti nulli ära peita uute mõistete (mida defineerime kui mõisted → 0) abil ja nii „kõrgel“ kui võimalik, s.o. produktsioonides *P4* ja *P6*. Teisisõnu, „lööme vasaku konteksti lahku“ ja lisame sellega konfliktsetele mõistetele uue ja erineva vasaku konteksti.

5.4. Kokkuvõte

Ülalpool käsitlesime tingimusi, mis peavad olema täidetud võimaldamaks kontekstivabadele eelnevusgrammatikatele tuginevat nonde poolt defineeritud keelte sõnade *BRC*-analüüsi. Resümeerigem.

- Kontekstivaba grammatika (*KVG*) peab olema eelnevusgrammatika (*EG*, tähestiku $V=V_N \cup V_T$ mis tahes kahe sümboli vahel võib kehtida ülimalt üks eelnevusrelatsioon); eksisteerib korrektne keelt säilitav algoritm suvalise *KVG* teisendamiseks *KV EG*-ks.
- Kui *KV*(*E*)*G* pole *pööratav* (s.o. produktsioonide paremad pooled pole unikaalsed: hulgas *P* on vähemalt üks paar kujul $A \rightarrow x$ ja $B \rightarrow x$), siis tuleb leida mitteterminalide *A* ja *B* derivatsioonist sõltumatu konteksti hulgad $C_{1|1}(A)$ ja $C_{1|1}(B)$; kui need eristuvad, on antud grammatika analüüsitav, kui aga ei, siis tuleb leida nende derivatsioonist sõltuvad kontekstid $C_{1,1}(A)$ ja $C_{1,1}(B)$. Kui need hulgad ei lõiku, siis on *G* analüüsitav, kui aga lõikuvad, siis jätkamiseks antud grammatikaga puudub üldine algoritm (*Konstruktor* lõpetab töö tulemusteta ja *Analüsaator* ei ole võimeline töötama).
- Kui kontekstivaba eelnevusgrammatika pole *BRC*-analüüsitav, siis võib püüda konteksti juurde tuua (selle tegevuse jaoks pole teada resultatiivset algoritmi ning tulemuseni jõudmist tuleb käsitsi üritada, teada on vähemalt üks (vahel) edukas tehnika).
- Kui grammatika vastab nõuetele, siis tagab *TTS* selle grammatikaga defineeritud keele sõnade (programmide) analüüsi puude juhitava moodustamise − seda teeb *Analüsaator* − ja identifikaatorite ning konstantide tabelite tegemise, kui tegu on programmeerimiskeelega. Ja sama *TTS*i programm saab hakkama nii abstraktsete kui ka reaalsete programmeerimiskeelte grammatikatega.

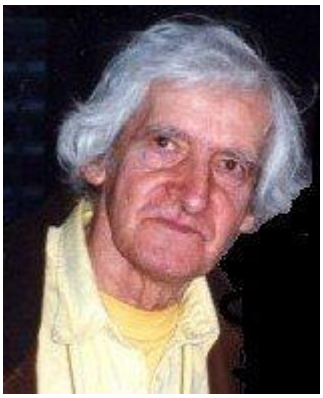
Niisiis, kui *KVG* kuulub *BRC*-analüüsitavate eelnevusgrammatikate hulka, siis tollele grammatikale tugineva keele $\mathscr{L}(G)$ translaator on tehtav (programmeeritav) *translaatorite tegemise süsteemi* tehnikat kasutades: hinnanguliselt üle poole rutiinsest tööst teeb mainitud süsteem ise ära, see töö päädib etteantud sõna (programmi) analüüsi puu (ja vajalike tabelite) väljastamisega. Ülejäänu (s.o. analüüsi puu → resultaat[1]) tuleb paraku ise programmeerida.

Me tutvustame selle lähenemisviisi üht võimalikku varianti allpool.

[1] See resultaat tähendab kas interpretaatori või kompilaatori kirjutamist iga järjekordse programmeerimiskeele jaoks.

6. Translaatorite Tegemise Süsteem (TTS)

Translaatorite Tegemise Süsteem on üldistatult programmide pakett, mille sisendiks on programmeerimiskeele $\mathscr{L}$ süntaksi reeglite kogum (kas *Backus*e-*Naur*i notatsioonis (*BNF*) või selle mingis modifikatsioonis, näiteks *produktsioonide keeles* esitatult) ja väljundiks on ideaalis tolle keele translaator (s.o. töötav programm) või reaalsemalt, keeles $\mathscr{L}$ kirjutatud programm-mide analüüsi puud. *TTS*i teerajaja on *A. Brooker* (joonis 6.a).



Anthony (Tony) Brooker[1] lõpetas *Imperial College*'i Inglismaal 1945. a. ja kutsuti 1951.a. Manchesteri Ülikooli tööle, täitmaks *Alan Turing*ist jäänud vakantsi. Ta oli tegev kolme tolleaegse tippmasina (*Mark* I, *Mercury* ja *Atlas* − kõik briti masinad) väljatöötamisel. 1954. a. evitas ta *Mark I autocode*'i, mida britid peavad esimeseks sammuks kõrgtasemekeelte leiutamise teel (kõik me teame, et esimene oli *J. Backus*e *FORTRAN*, aga kas oli?). Järgnevalt tegeles *TB* ujupunktaritmeetika riistvaraga, ent "oma" masinate assemblerite baasil ja masinast sõltumatute keelte survel jõudis ta *Compiler Compiler*i kontseptsioonini, ta lähtus *BNF*-süntaksi defineerimisest ja töötas välja viie tolleaegse masina jaoks sobiva vahekeele formaadi.

Ja termin *CC* (*Compiler-Compiler*) pärineb just *Anthony Brooker*ilt.

Joonis 6.a. *Anthony Brooker*.

Esimesed andmed süntaksorienteeritud (seejuures orienteeritud mingile kontekstivabade grammatikate alamklassile) transleerimisest pärinevad 1960. aastaist. Noil aegadel pidi transleerimisresultaat olema (meie mõistes) *.exe-fail*, toona täidetav masinkoodiprogramm. See oli ideaal, selle tulemuseni süntaksorienteeritud tehnikad (kuhu lisati ideaali saavutamiseks semantilised atribuudid) arvestatavale tasemele ei jõudnud. Ent *Translaatorite Tegemise Süsteemid* (*TTS*, vn. *Система Построения Трансляторов – СПТ*) arendati järgmise paarikümne aasta jooksul üpris arvestatavale tasemele[2]. Kusjuures masinkoodi-väljundi asemele asus objektmasina *assembler*, mis on protsessororienteeritud ning täielikult sõltuv riistvarast. Ületamaks erinevate platvormide (loe: protsessorite (ja võimalik, et opsüsteemide)) põhjustatud raskusi tarkvara mobiilsuse saavutamisel, jõuti lõpuks üldlevinud lahenduseni: tänapäevased *TTS*id on orienteeritud kasutama väljundkeelena (*objektkeelena*) kõrgtaseme keelt[3] (sellist, mille jaoks on olemas efektiivne translaator) [*c-list*]. Uuem tendents on kasutada vahekeelena (objektkeelena) *baitkoodi* (ingl. *byte-code*), omamoodi universaalse abstraktse arvuti (baitmasina) masinkoodi/assemblerit[4].

Kuivõrd võrdväärseid analüüsimeetodeid oli palju: ülalt-alla-tehnikad, alt-üles-tehnikad, erinevad kontekstiarvestamise tehnikad jne[5], siis mingit *TTS*-standardit ei kehtestatud. Selles

[1] Vt. [wTB, wCC, TB,W:Man]. Siinkirjutajal on kahju, et ta ei leidnud *TB* 50-ndate aastate pilti. Abi polnud *Google*'ist (sellenimelisi on liiga palju) ega ka *Manchesteri Ülikoolist*, kes pakkus sama pilti, mis meil juba on.

[2] Eeskätt teoreetilisel tasemel, tööstuslikud kompilaatorid on reeglina tehtud „põlve otsas", orienteerudes ühelt poolt keelele, teiselt poolt op-süsteemile ja protsessorile. Termini *CC* asemel kasutatakse tänapäeval tihti asjakohasemat *analüsaatorite generaatorit* (*Parser Generator*).

[3] Nt. *C*, *C++*, *C#*, *Java*, *Python*, *Delphi* jmt.

[4] Vt. nt. [Isotamm, 2007], lk. 208–220; vahet pole, on see *Java* baitkood, *.NET* või *MONO*. Me ei tutvusta objektkeele kõrgtasemeversiooni, mõttega, et lugeja saaks aduda tegelikku viimase taseme keele vajalikkust. Protsessor suudab interpreteerida *ainult* masinkoodi. Nt *Trigol* → *C* ei avaks madala taseme transleerimist mingilgi määral. Kusjuures, see (kompilaator *C*-sse) võiks olla bakalaureusetöö teemaks.

[5] Siia ei mahu nonde tehnikate käsitlus, sestap soovitame huvilistele otsida asjakohast informatsiooni iseseisvalt. Meetodite paljus johtus teema aktuaalsusest, see oli ajakirjades ja konverentsidel ja dissertatsioonides esiplaanil.

peatükis keskendume ühele paljudest võimalustest − eelnevusgrammatikatele (*N. Wirth*i kui selle meetodi formaliseerija töödele) põhinevale alt-üles-realisatsioonile – mille eeliseid ülejäänute ees (kui neid on) on raske tõestada.

6.1. TTSi üldine skeem

Allpool tutvustatava *TTS*i ülesehitus järgib *Mati Tombak*u ideid, ent füüsiline realisatsioon teadlikult mitte. Grammatikate klass on kitsendatud kontekstivabade eelnevusgrammatikatega, mida on võimalik redutseerida *BRC*(1,1)-tehnikaga (mitteterminali *A* kontekstiks on tema definitsiooni $A \rightarrow x$ parema poole *x* jaoks üks sümbol vasakult (magasinis) ja üks sümbol paremalt, sisendreast).

Translaatori tegijale pakub süsteem "programmeerimisvaba" tuge kahe plokiga: need on *Konstruktor* ja *Analüsaator*. Esimene testib etteantud grammatikat (Kas on eelnevusgrammatika? Kas on pööratav? Kas on *BRC*-redutseeritav?), saades ette grammatika produktsioonide hulga *P* (ja võimalik, et ka semantikafaili) ning väljastades asjakohased tabelid. Ideaalis töötab *Konstruktor* üks kord iga uue grammatika jaoks, reaalselt läheb vaja rohkem "jooksutamisi". Sisulisel tasemel peab translaatori kirjutaja sekkuma ainult siis, kui derivatsioonist sõltuv kontekst ei eristu, kõik muu teeb *Konstruktor* kui programm. Muuhulgas teeb[1] ta ka grammatika viimasele versioonile (see saadi pärast võimalikke eelnevusteisendusi) tugineva semantikafaili, mis tagab analüüsi täispuu genereerimise; hõreda puu saamiseks piisab, kui tolles failis liigsed read märkida (semantikafaili) kommentaarideks (kommentaari markeriks on sümbol „$", mis kehtib reavahetuseni). Seda faili võib modifitseerida kuni rahuldava tulemuseni jõudmiseni.

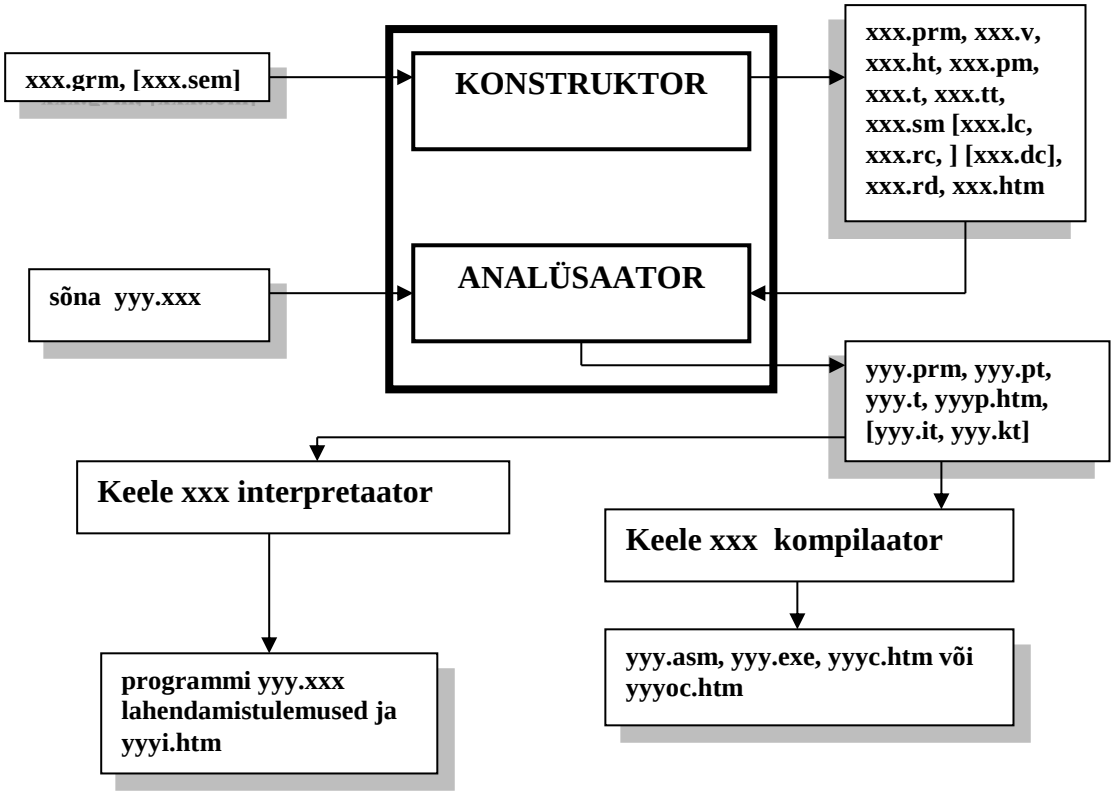
*TTS*i üldine skeem on toodud joonisel 6.1.a. Kui tegeleme vaid abstraktsete (õppe)grammatikatega, siis *TTS* teebki kõik, mis vaja. Programmeerimiskeele realiseerimine on komplitseeritum. Aga mingem samm-sammult edasi, alustades lihtvariandist.

*TTS*i startimiseks tuleb käivitada vabavaraline fail *tts.exe* ning avaneb joonisel 6.1.b toodud pilt (süsteem on allalaaditav aadressilt http://www.cs.ut.ee/~isotamm → Automaadid, keeled ja translaatorid).

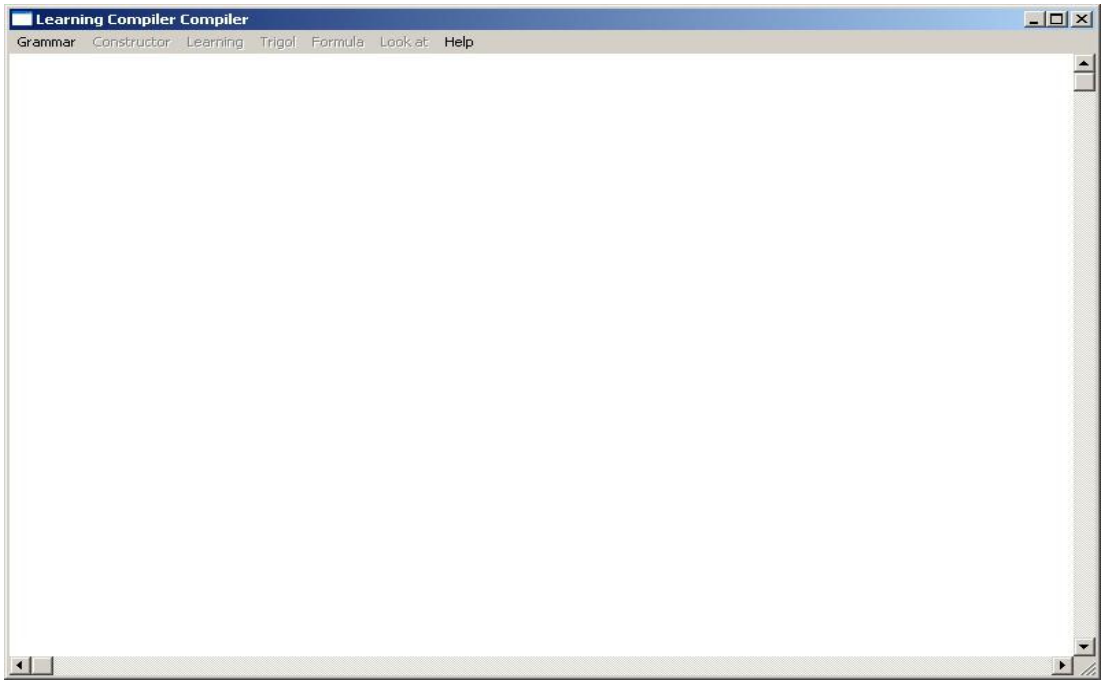
Meie *TTS*is on esmalähenduses avatavad ainult kaks rippmenüüd: grammatika ja „Help" – viimase pealkiri on mõneti eksitav, kuivõrd seal avaneb kogu *TTS*i elektrooniline manuaal. Ülejäänud rippmenüüde aktiveerimiseks tuleb avada menüü „Grammar" (vt. joonis 6.1c). Miks nii: edasised variandid on grammatika-orienteeritud ning ilma grammatikat fikseerimata (ja vajadusel *Konstruktorit* lahendamata) ei saa aktiveerida järgmiseid samme, milleks on sõna (programmi) valimine analüüsiks ning see, mida analüüsi puuga edasi teha. Kas piirdudagi puuga, nagu on ainuvõimalik abstraktsete (õppe)grammatikate puhul või valime − meie raamatu näitel − edasise tegevuse, s.o. kirjutame kas analüüsi puu otseses mõttes interpretaatori või siis sellise interpretaatori, mis genereerib teksti mingis muus programmeerimiskeeles[2].

[1] Seda vaid siis, kui kasutaja pole ise semantikafaili teinud – mida me ei soovitagi teha, genereeritu on mugavam.

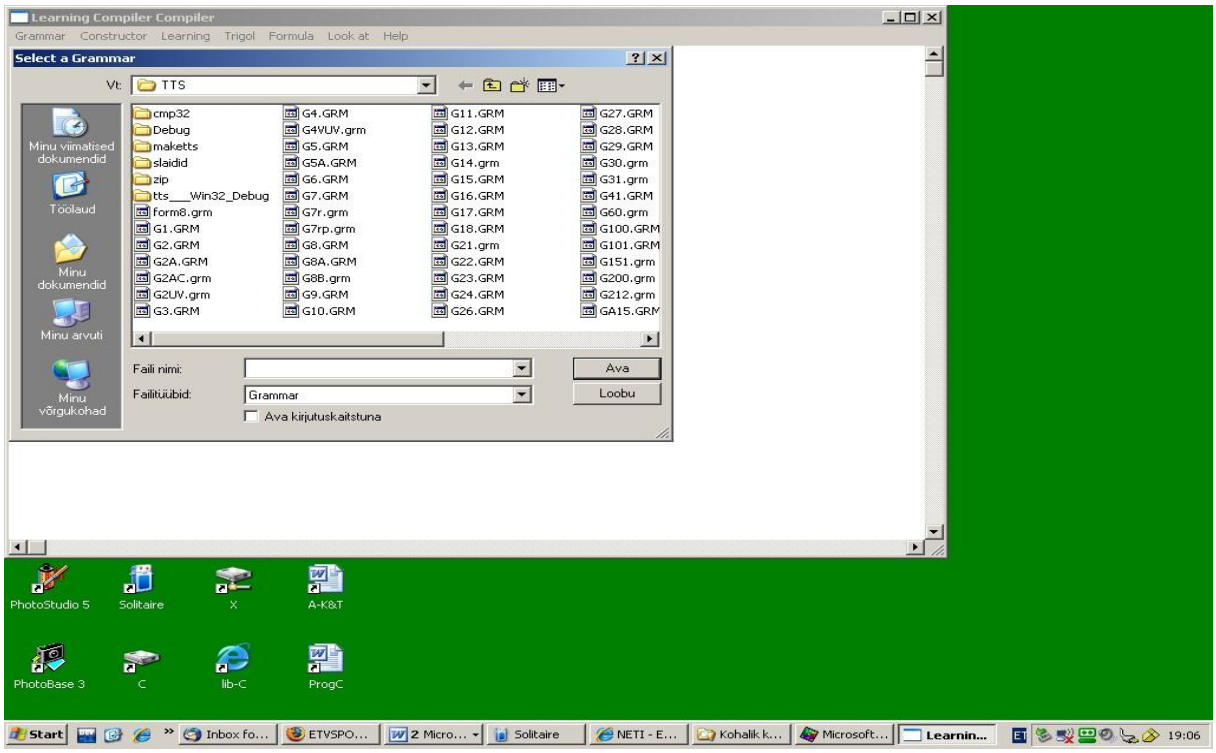
[2] Nende kaante vahel on ainsaks muuks keeleks *Intel*i assembler, ent *Henri Lakk* lisas *Java baitkoodi* ning *Einar Pius .NET*i ja *MONO*. Need variandid jäid sellest raamatust kahjuks välja.



Joonis 6.1.a. *TTS*i üldine skeem.

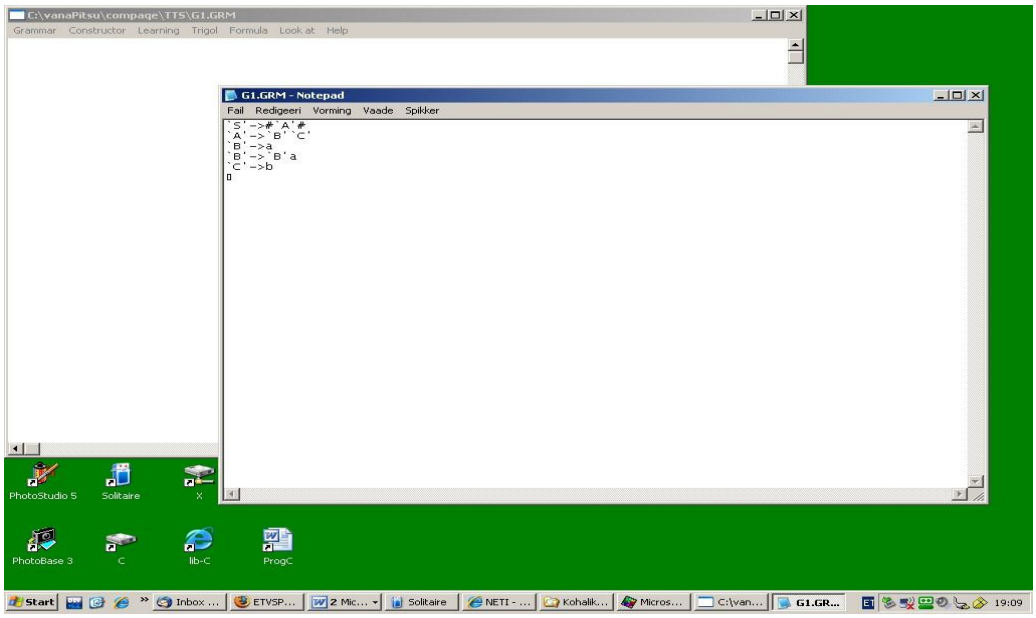


Joonis 6.1.b. TTSi startimise ekraanipilt.

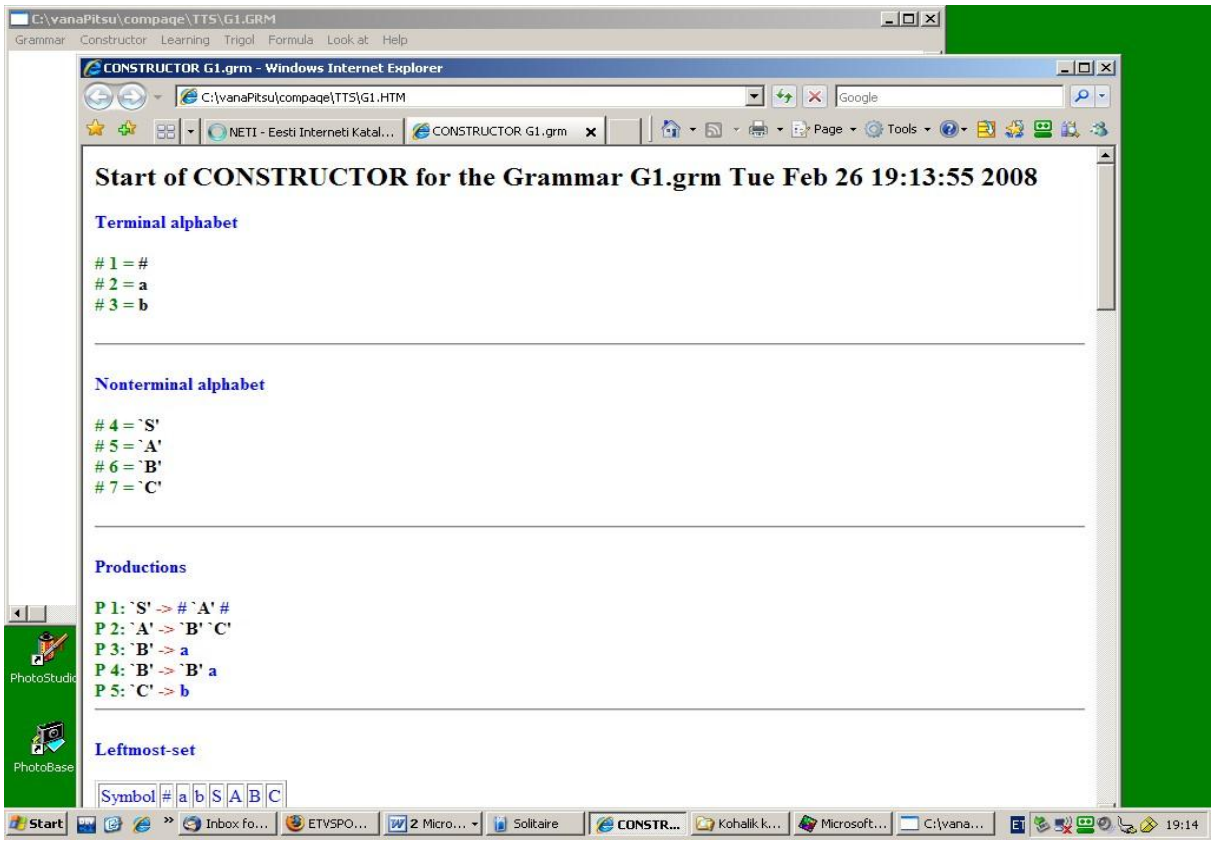


Joonis 6.1.c. Grammatika valimise võimaldamine.

*TTS*i genereeritud andmestruktuuride visualiseerimine tugineb samuti ja loomulikult grammatikale ja keelele ning fikseerimata grammatikat, võib aktiivne menüü tekitada vaid müra.



Joonis 6.1.d. Valiti *G1.grm.*



Joonis 6.1.e. *Konstruktori* logifaili kuvamise algus.

Teine plokk – *Analüsaator* – saab ette *Konstruktori* väljundtabelid[1] ning lähtekeelse *programmi*, väljundiks on kas veateated või andmestruktuurid: analüüsi (hõre) puu ning identifikaatorite ja konstantide tabelid[2]. Ka see plokk on tagatud *TTS*iga – programmeerija ei pea koodi kirjutama.

Kolmas plokk tuleb programmeerida: analüsaatorilt on saadud (hõre) puu ja analüsaatori tabelid ning kirjutatakse kas interpretaator või kompilaator − siis, kui keel on *programmeerimiskeel.*

6.2. TTSi esimene plokk: *Konstruktor*

Konstruktor on alati orienteeritud *analüüsimeetodile*: ta peab genereerima analüüsi garanteerivad andmestruktuurid. Me ei hakka siingi käsitlema paljusid (sisuldasa võrdväärseid) variante, vaid keskendume Eestis 1970ndatel (*Mati Tombaku*[3]) valitud lahendusele – eelnevusgrammatikatele ning *BRC*-analüsaatorile; sellist tehnikat järgib ka meie näite-*TTS*. *Konstruktori* jäme algoritm on esitatud joonisel 6.2.a. Peen (täpne) algoritm on lisades esitatud *C*-programmiga. Allpool esitame näiteks valitud abstraktse grammatika *G8.grm* (mis

[1] Neid tabeleid saab vaadata rippmenüüst nupu *look at* abil, samast menüüst on kättesaadavad ka *TTS*i programmide *C*-tekstid (mis leiduvad ka meie raamatu lisades). Need on vabalt allalaaditavad ja muudetavad.

[2] Need siis, kui tegemist on programmeerimiskeelega. Abstraktse grammatika puhul noid tabeleid mõistagi ei tekitata.

[3] Meie jaoks mõneti üllatuslikult leidis *Mati Tombak* paar aastat tagasi, et *BRC*-analüüs tuleks meie kursuses asendada *LR*-analüüsiga, vältimaks tülikat kontekstiarvutamistehnikat. Ideed kaalutakse.

on *Konstruktori* jaoks üpris komplitseeritud) *Konstruktori* logifaili *G8.htm* ning genereeritud reaalsed andmestruktuurid (fail *g8rd.htm*), viimased koos raamatu jaoks lisatud kommentaaridega.

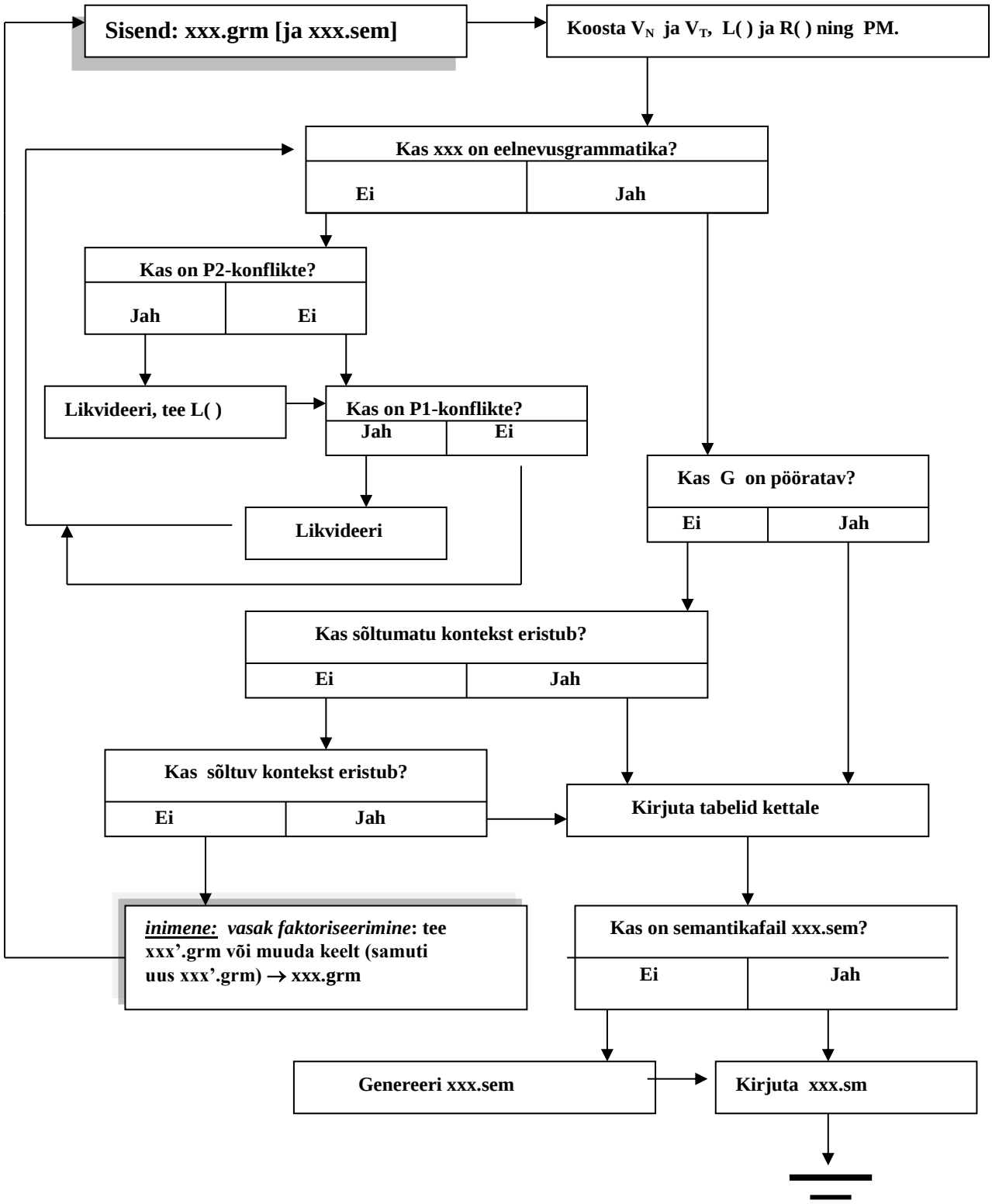
Niisiis, me teame, et *Konstruktor* saab ette kontekstivaba grammatika produktsioonide hulga *P* ja võib saada semantikafaili ning genereerib järgmiste *TTS*i plokkide (*Skanner* ja *Parser*) jaoks vajaliku hulga tabeleid. Abstraktsel tasemel me teame, mis informatsiooni need esitavad: eelnevusmaatriks kui analüsaatori tähtsaim tabel, vajadusel sõltumatu *LC*( ) ja *RC*( ) ja sõltuva konteksti *DC*( ) tabelid[1]. Ent neile lisaks on *Konstruktoril* täisinformatsioon *Analüsaatorile* redutseerimiseks ning *Skannerile* lekseemide tuvastamiseks; neid seiku pole me siiani tähtsustanud, pluss veel üht-teist vajalikku[2].

Joonisel 6.2a on esitatud *Konstruktori* üsnagi detailne algoritm. Rõhutagem veel kord, et kõikide tegevustega saab *Konstruktor* ise hakkama − kuni seigani, et sõltuv kontekst ei eristu ning vaja on *inimese* sekkumist. Mainime veel, et *Konstruktori* teooria on tõestatud teoreemidega ning *konteksti lisamine* on paraku niisuguse korrektse toeta.

Tollel joonisel kasutame paljusid failide nimesid; allpool seletame nad lahti, kasutades menüüs *look up* pakutavatest varianti *Files*, kus juba tuttava grammatika *G8* jaoks tuleb valida *G8rd.htm*[3].

Jaotises 6.2.1 reprodutseerime meie näitegrammatika *G8 Konstruktori* logi.

[1] Ent *L*( ) ja *R*( )-hulgad huvitavad ainult *Konstruktorit* ning neid salvestada pole mõtet.
[2] Me ei käsitle siin situatsiooni, kus *KVG* tuli teisendada *eelnevusgrammatikaks* ning esialgsed hulgad V_N ja *P* kirjutati üle − needki on *Konstruktori* väljundandmed, ent *Analüsaator* „ei tea“ noist teisendustest midagi.
[3] „*rd*“ on „*real data* (*structures*)“ – tegelikud, füüsilised andmestruktuurid.



Joonis 6.2.a. *Konstruktori* algoritm.

6.2.1. Konstruktori logi

Start of CONSTRUCTOR for the Grammar G8.grm Wed Jun 01 20:15:07 2011

Terminal alphabet
1 =
2 = a
3 = b
4 = 1

Nonterminal alphabet
5 = `T'
6 = `S'
7 = `A'
8 = `B'

Productions
P 1: `T' -> # `S' #
P 2: `S' -> a `A' a
P 3: `S' -> b `A' b
P 4: `S' -> a `B' b
P 5: `S' -> b `B' a
P 6: `A' -> 1
P 7: `B' -> 1

Leftmost-set

Symbol	#	a	b	1	T	S	A	B
5.T	*	0	0	0	0	0	0	0
6.S	0	*	*	0	0	0	0	0
7.A	0	0	0	*	0	0	0	0
8.B	0	0	0	*	0	0	0	0

Rightmost-set

Symbol	#	a	b	1	T	S	A	B
5.T	*	0	0	0	0	0	0	0
6.S	0	*	*	0	0	0	0	0
7.A	0	0	0	*	0	0	0	0
8.B	0	0	0	*	0	0	0	0

Leftmost & rightmost sets
`T' leftmost set: `#'
`T' rightmost set: #

`S' leftmost set: `a' , `b'
`S' rightmost set: a , b

`A' leftmost set: `1'
`A' rightmost set: 1

`B' leftmost set: `1'
`B' rightmost set: 1

Precedence matrix

Symbol	#	a	b	1	T	S	A	B
1.#	0	<	<	0	0	=	0	0
2.a	>	0	0	<	0	0	=	=
3.b	>	0	0	<	0	0	=	=
4.1	0	>	>	0	0	0	0	0
5.T	0	0	0	0	0	0	0	0
6.S	=	0	0	0	0	0	0	0
7.A	0	=	=	0	0	0	0	0
8.B	0	=	=	0	0	0	0	0

The relationships of symbol #1 #:

<• a	<• b	=• `S'

The relationships of symbol #2 a:

•> #	<• 1	=• `A'	=• `B'

The relationships of symbol #3 b:

•> #	<• 1	=• `A'	=• `B'

The relationships of symbol #4 1:

•> a	•> b

The relationships of symbol #5 `T':

The relationships of symbol #6 `S':

=• #

The relationships of symbol #7 `A':

=• a	=• b

The relationships of symbol #8 `B':

=• a	=• b

Grammar G8.grm is a precedence grammar

Grammar G8.grm is not invertible

Left Context

Symbol	#	a	b	1	T	S	A	B
5.T	0	0	0	0	0	0	0	0
6.S	*	0	0	0	0	0	0	0
7.A	0	*	*	0	0	0	0	0
8.B	0	*	*	0	0	0	0	0

Right Context

Symbol	#	a	b	1
5.T	0	0	0	0
6.S	*	0	0	0
7.A	0	*	*	0
8.B	0	*	*	0

Independent context
`S' left context: #
`S' right context: #

`A' left context: a , b
`A' right context: a , b

`B' left context: a , b
`B' right context: a , b

Equivalent definitions:
`A' —> 1 & `B' —> 1
`A' left context: a , b
`A' right context: a , b

`B' left context: a , b
`B' right context: a , b

The independent context of `A' and `B' is not different

---

independent context didn't help us. I'll try to use the dependent one.
I'll find the subsets of dependent context of `A'
gamma1: the source is the production
P=2 `S' -> a `A' a
{a , a}
gamma1: the source is the production
P=3 `S' -> b `A' b
{b , b}

The set of dependent context of `A':
{a , a} {b , b}

I'll find the subsets of dependent context of `B'
gamma1: the source is the production
P=4 `S' -> a `B' b
{a , b}
gamma1: the source is the production
P=5 `S' -> b `B' a
{b , a}

The set of dependent context of `B':
{a , b} {b , a}

test_dep_con A and B

dependent context of `A' and `B' is different

Grammar G8.grm is BRC(1,1)-reducible

Dependent context
dependent context of `A':
{a , a} {b , b}
dependent context of `B':
{a , b} {b , a}

Semantics

Semantics file is G8.sem
#=1
a=2
b=3
1=4
P1=5 $P 1: `T' -> # `S' #
P2=6 $P 2: `S' -> a `A' a
P3=7 $P 3: `S' -> b `A' b
P4=8 $P 4: `S' -> a `B' b
P5=9 $P 5: `S' -> b `B' a
P6=10 $P 6: `A' -> 1
P7=11 $P 7: `B' -> 1

Result tables

File	Size
G8.prm	28
G8.pm	81
G8.t	180

G8.tt	100
G8.ht	1316
G8.sm	52
G8.v	464
G8.lc	81
G8.rc	81
G8.dc	128

Look at result tables

Finish of CONSTRUCTOR Wed Jun 01 20:15:07 2011

Konstruktori logifaili viimane pakkumine oli, et „vaata resultaattabeleid“. Esitame need järgnevas alapeatükis.

6.2.2. Konstruktori genereeritud tabelid

CONSTRUCTOR TABLES of the Grammar G8.grm (G8 *Konstruktori* tabelid)

- Parameters
- Alphabet V
- Scanner Table
- Tree of the Alphabet V
- Reduce Table
- Precedence Matrix
- Left Context
- Right Context
- Dependent Context
- Semantics

Parameters (*Konstruktori* (ja *Parseri*) parameetrite fail)

File G8.prm

```
struct parm{
        int nr   //tähestiku V pikkus
        int tnr  //tähestiku V_T pikkus
        int BRC  //0: G on pööratav
        int Pnr  //Produktsioonide arv
        int dep  //1: sõltuv kontekst
        int itl  //identifikaatorite arv (Parser)
        int ktl  //konstantide arv (Parser)
};

nr=8 tnr=4 BRC=1 Pnr=8 dep=1 itl=0 ktl=0
```

Alphabet V (tähestik *V*, esmalt V_T ja siis V_N)

File G8.*t*

```
# a b 1 T S A B
```
See on tähestiku *V* sümbolite vektor (elemendid on viidad tekstidele).

1 2 3 4 5 6 7 8 Need on *V* elementide meie poolt lisatud indeksid, hõlbustamaks andmestruktuuride lugemist järgnevalt toodud tabelites. Näiteks, „T[4]“ on sümbol „1“ ja „T[7]“ on sümbol „*A*“.

Scanner Table (Skaneerimistabel, vektor, mida kasutab *Skanner* lekseemide tuvastamiseks)

File G8.tt

```
# a b 1
```

See on redutseeritud eelmisest vektorist, viidad on järjestatud V_T elementide pikkuste mittekasvavas järjekorras. Meie grammatika pole selle vektori demonstreerimiseks üldse mitte sobiv, sestap toome siin ära *Trigol*i vastava vektori:

```
WRITE THEN READ GOTO #c# #i# /= IF := <= >= ( ) < > ; : + = - * / #
```

Tree of the Alphabet V (tähestiku *V* puu)

Tähestiku *V* puu on üles ehitatud otsimis-kahendpuuna (vt. nt. [Isotamm, 2009], lk. 162 jj.), kus võtmeteks on tähestikku *V* moodustavad stringid ning kuhu kantakse tippe kindlas järjekorras: esmalt terminaalse tähestiku tähed, seejärel mõisted ja − analüüsi ajal − identifikaatorid ja konstandid[1]. Puu tippu kirjeldab struktuur *D*, mis omakorda kasutab mõistete jaoks struktuuri *R*. Niisiis, tähestiku *V* *Konstruktorilt* saadud puu:

```
 D 00681C50: tunnus=1 code=1 L=1 loc=0 icx=0 left=00000000 right=00681BB0
def=00000000   (#)
 D 00681930: tunnus=1 code=4 L=1 loc=0 icx=0 left=00000000 right=00681890
def=00000000   (1)
 D 00681630: tunnus=0 code=7 L=1 loc=0 icx=0 left=00000000 right=006812B0
def=00682F00   (A)
 R 00682F00: P=6 sem=0 n=1 d=00681030 alt=00000000
 d 00681030: 4
 D 006812B0: tunnus=0 code=8 L=1 loc=0 icx=0 left=00000000 right=00000000
def=00682E20   (B)
 R 00682E20: P=7 sem=0 n=1 d=00682DF0 alt=00000000
 d 00682DF0: 4
 D 006817F0: tunnus=0 code=6 L=1 loc=0 icx=0 left=00681630 right=00000000
def=00681060   (S)
 R 00681060: P=5 sem=0 n=3 d=00682FE0 alt=00681220
 d 00682FE0: 3 8 2
 R 00681220: P=4 sem=0 n=3 d=006811E0 alt=006813E0
 d 006811E0: 2 8 3
 R 006813E0: P=3 sem=0 n=3 d=006813A0 alt=006815A0
 d 006813A0: 3 7 3
 R 006815A0: P=2 sem=0 n=3 d=00681560 alt=00000000
 d 00681560: 2 7 2
 D 00681890: tunnus=0 code=5 L=1 loc=0 icx=0 left=006817F0 right=00000000
def=00681760   (T)
 R 00681760: P=1 sem=0 n=3 d=00681720 alt=00000000
```

```
 d 00681720: 1 6 1
 D 00681BB0: tunnus=1 code=2 L=1 loc=0 icx=0 left=00681930 right=00681B10
def=00000000   (a)
 D 00681B10: tunnus=1 code=3 L=1 loc=0 icx=0 left=00000000 right=00000000
def=00000000   (b)
```

Reduce Table (redutseerimistabel)

Redutseerimistabel on *paisktabel*, mille kirjete võtmeks on produktsioonide paremad pooled (mõistagi, kodeeritult) − analüüsist teame neid *lause vormide baasidena*. Viit *col* on tavaline paisktabeli põrkeviit (kaks erinevat võtit annavad sama paiskaadressi), ent viit *same* on kasutusel siis, kui kaks (või rohkem) mitteterminali on samamoodi defineeritud ning redutseerimisel tuleb kasutada mõiste konteksti. Kui *nc*=0, siis kasutab *Analüsaator* tabelit *xxx.lc* ja vajadusel *xxx.rc*, aga kui *nc*=1, siis tuleb kasutada andmestruktuuri *xxx.dc* − derivatsioonist sõltuvat konteksti.

File G8.ht

```
struct h{
        int P;          /* produktsiooni jrk-nr */
        int n;          /* definitsiooni pikkus  */
        int NT;         /* defineeritav mitteterminal  */
        int *d;         /* definitsioon: hash-võti */
        int nc;         /* 1 - sõltuv kontekst */
        char *c;        /* reservis */
        struct h *same; /* sama parem pool */
        struct h *col;  /* põrkeviit */
};
```

```
 h 00682AD0: P=6 n=1 NT=7 d=00682AA0 nc=1 same=00682A50 col=00000000  (A)
 d 00682AA0: 4
 h 00682A50: P=7 n=1 NT=8 d=00682A20 nc=1 same=00000000 col=00000000  (B)
 d 00682A20: 4
 h 00682DA0: P=1 n=3 NT=5 d=00682D60 nc=0 same=00000000 col=00000000  (T)
 d 00682D60: 1 6 1
 h 00682B60: P=2 n=3 NT=6 d=00682B20 nc=0 same=00000000 col=00682BF0  (S)
 d 00682B20: 2 7 2
 h 00682BF0: P=3 n=3 NT=6 d=00682BB0 nc=0 same=00000000 col=00000000  (S)
 d 00682BB0: 3 7 3
 h 00682C80: P=4 n=3 NT=6 d=00682C40 nc=0 same=00000000 col=00682D10  (S)
 d 00682C40: 2 8 3
 h 00682D10: P=5 n=3 NT=6 d=00682CD0 nc=0 same=00000000 col=00000000  (S)
 d 00682CD0: 3 8 2
```

Precedence Matrix (eelnevusmaatriks)

Tuletagem meelde eelnevusrelatsioonide koodid, neid kasutatakse eelnevusmaatriksis:

sümbol	***kood$_2$***	***tehe***	***kood$_8$***
⊙	000		0
≐	001		1
⋖	010		2
⋗	100		4
⋖ ≐	011	010∨001	3
⋗≐	101	100∨001	5
⋖ ⋗	110	100∨010	6
⋖ ≐ ⋗	111	010∨001∨100	7

File G8.*pm*

Nr	1	2	3	4	5	6	7	8
1	0	2	2	0	0	1	0	0
2	4	0	0	2	0	0	1	1
3	4	0	0	2	0	0	1	1
4	0	4	4	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0
7	0	1	1	0	0	0	0	0
8	0	1	1	0	0	0	0	0

Järgmised tabelid − failid − on maatriksid, mis esitavad mitteterminalide sõltumatute kontekstide ($C_{1|1}(A)$ ja $C_{1|1}(B)$) hulkasid; $V_N[6]=S$, $V_N[7]=A$ ja $V_N[8]=B$. $V_T[2]=a$ ja $V_T[3]=b$. $V_T[1]=\#$.

Left Context
File G8.lc

Nr	1	2	3	4	5	6	7	8
6	1	0	0	0		0	0	0
7	0	1	1	0		0	0	0
8	0	1	1	0	0		0	0

Right Context
File G8.rc

Nr	1	2	3	4
6	1	0	0	0
7	0	1	1	0
8	0	1	1	0

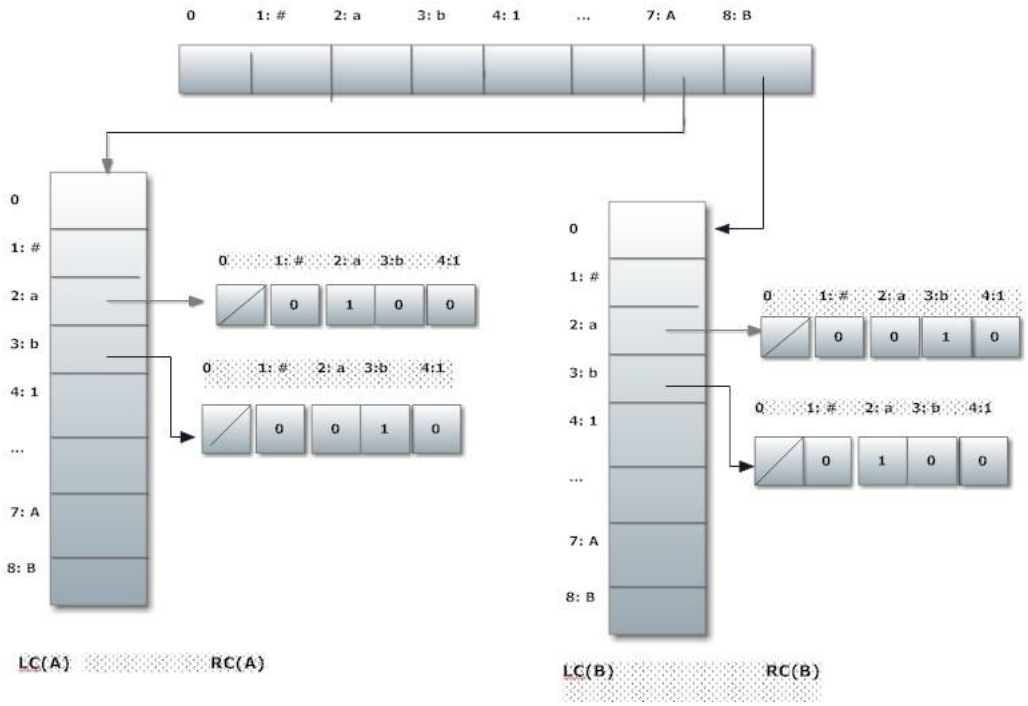
Ridades on mitteterminalid $A \in V_N$; vasaku konteksti $LC(A)$ maatriksis on veerud kõigile tähestiku V elementidele ja parema konteksti omas ainult terminalidele. Kui veerusümbol kuulub mitteterminali konteksti hulka, siis on vastavas lahtris väärtus 1, muidu aga 0.

Dependent Context (sõltuv kontekst)

File G8.dc

See on derivatsioonist sõltuva konteksti tabel, kus vasak ja parem kontekst pole lahus nagu sõltumatu konteksti puhul, vaid kus on olulised vasaku ja parema konteksti *paarid.* Sestap oli ka pisut raskusi sobiva (ja mugava) andmestruktuuri leidmisel. Me ei osanud midagi paremat välja mõtelda kui kolmetasemeline struktuur, kus vektor *DEP* (pikkusega $|V|$, tähestiku V pikkus) viitab mitteterminalidega seotud indeksitega väljadelt sama pikkadele ($|V|$) vektoritele, kus vasaku konteksti element (kui selle indeksiga element kuulub antud mitteterminali vasaku konteksti hulka) viitab vektorile pikkusega $|V_T|$, kus baidi väärtus on 1, kui vastava indeksiga terminal on uuritava mitteterminali paremas kontekstis ja muidu 0. See on struktuur formaalse kirjelduseta, sestap esitame esmalt ta mälutõmmise (ridade lõpus on toodud raamatu-selgitus) ja seejärel „puu"-skeemi joonisel 6.2.b.

```
DEP[7]=00682840      A
L[2]=00682800        aϵLC(A)
rc[2]=1              aϵRC(A)
L[3]=006827C0        bϵLC(A)
rc[3]=1              bϵRC(A)
DEP[8]=00682770      B
L[2]=00682730        aϵLC(B)
rc[3]=1              bϵRC(B)
L[3]=006826F0        bϵLC(B)
rc[2]=1              aϵRC(B)
```



Joonis 6.2.b. Sõltuv kontekst.

Semantics (semantika on kõigil terminalidel ja produktsioonidel)

File G8.sm

0 1 2 3 4 5 6 7 8 9 10 11

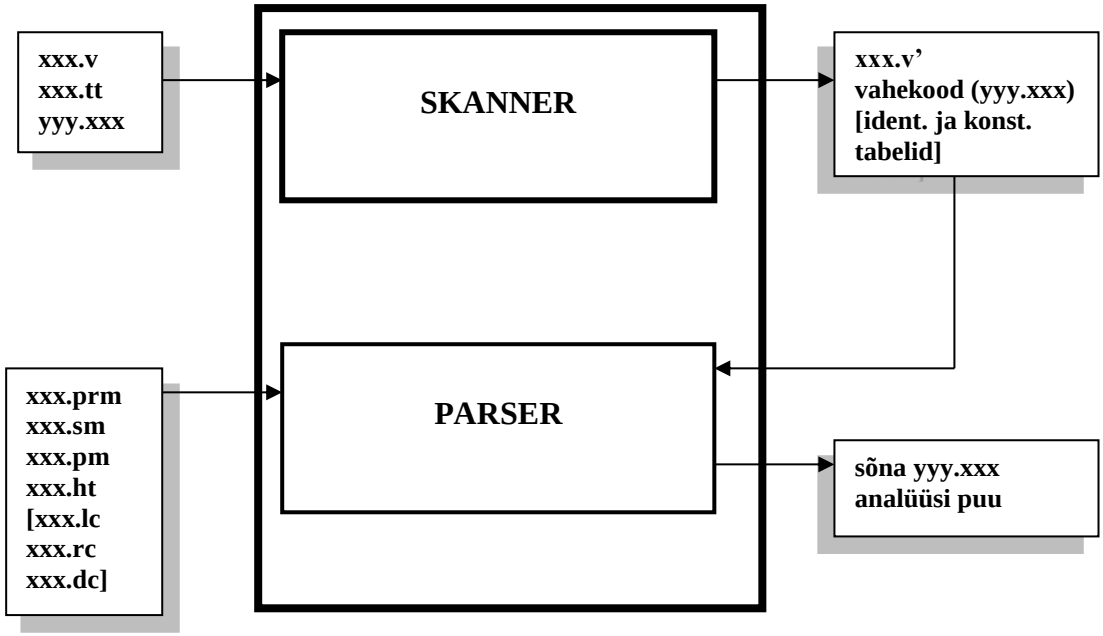
See vektor on saadud tekstifaili *G8.sem* transleerimise resultaadina, fail *G8.sem* ise[1] on järgmine:

```
1=1    $ #
2=2    $ a
3=3    $ b
4=4    $ 1
P1=5   $   P 1: `T' -> # `S' #
P2=6   $   P 2: `S' -> a `A' a
P3=7   $   P 3: `S' -> b `A' b
P4=8   $   P 4: `S' -> a `B' b
P5=9   $   P 5: `S' -> b `B' a
P6=10  $   P 6: `A' -> 1
P7=11  $   P 7: `B' -> 1
```

[1] Fail on genereeritud (koos kommentaaridega).

6.3. TTSi teine plokk: Analüsaator

Analüsaator on programm, mis on võimeline genereerima iga *Konstruktori* poolt aktsepteeritud keele *sõna* (*resp.* süntaktiliselt korrektse programmi[1]) analüüsi puu (ja vajadusel ka muud andmestruktuurid[2]).



Joonis 6.3.a. *Analüsaatori* plokid.

Põhimõtteliselt teame analüsaatori algoritmi juba varem esitatust; siin keskendume (põgusalt) realisatsioonile ja nendele aspektidele, mis teooriale huvi ei paku – näiteks, kuidas käituda, kui analüsaator saab vea ja keel on *programmeerimiskeel*.

Analüsaator kui programm koosneb kahest järjestikku töötavast plokist: need on *Skanner* ja *Parser*[3] (vt. joonis 6.3.a). Ent – juhatagem see teema sisse meie näitegrammatikaks valitud *G8* poolt defineeritud keele sõna *ff.g8* analüüsi logiga.

```
Start of G8 Parser for a Word ff.G8 at Wed Jun 06 12:31:08 2001

Scanner started
Input program:
# b 1 b #
Scanned program:
1 3 4 3 1
Scanner ended
Parser started
```

[1] Juhul, kui keel on programmeerimiskeel.
[2] Programmeerimiskeele puhul kindlasti identifikaatorite ja konstantide tabelid.
[3] Meil pole nende terminite jaoks välja kujunenud üldaktsepteeritud terminoloogiat. Võiksime *Skannerit* nimetada *lekseemide tuvastajaks*, ent mis omakeelne termin too *lekseem*gi on. *Parser* võiks olla *puumoodustaja*, aga see ei avaks kõiki tolle ploki tegevusi.

Stack & Word	#	<•b	1	b	#

Stack & Word	#	<•b	<•1	b	#

Stack & Word	#	<•b	=•A	b	#

Parsing tree

A
1

Stack & Word	#	<•b	=•A	=•b	#

Stack & Word	#	=•S	#

Parsing tree

S		
b	A	b
	1	

the parsing is completed

Parsing tree

S		
b	A	b
	1	

Result tables

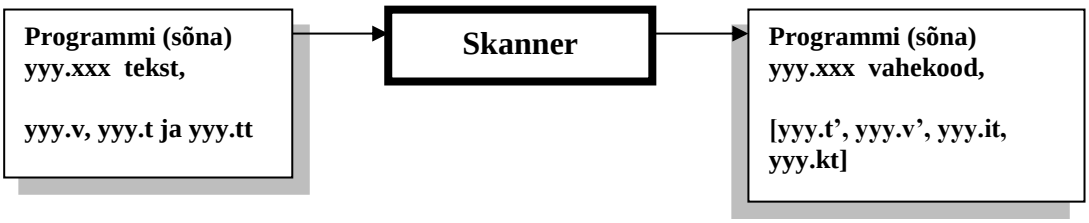
File	Size
ff.prm	28
ff.t	200
ff.it	0
ff.kt	0
ff.pt	180

Parser ended at Wed Jun 06 12:31:08 2001

6.3.1. Skanner

Transleerimiseks antud programmi (*sõna*) teksti võime vaadelda kui *lekseemide* (terminaalse tähestiku elementide e. terminalide) jada, kusjuures nood lekseemid on kas *reserv-* või *võtmesõnad*, *eraldajad* või *lekseemiklasside* (identifikaatorid, konstandid, stringid ja kommentaarid) esindajad; lisandub *müra* − tühikud, reavahetused, tabulatsioon[1] jmt.

Skanner on programm, mis peab esiteks tuvastama lekseemid ja teiseks väljastama vektori, kus iga lekseem (algselt *tekst*) on asendatud *koodiga* (loe: kümnendarvuga), millega opereerivad kõik järgnevad algoritmid. Lisaks peab *Skanner* reaalse (programmeerimiskeele) translaatori koosseisus tuvastama lekseemiklasside objektid ning moodustama nende jaoks asjakohased andmestruktuurid (nt. identifikaatorite ja konstantide *tabelid*)[2].



Joonis 6.3.1.a. *Skanneri* töö.

Ühearvulised koodid omistab *Konstruktor* reserv- ja võtmesõnadele ning eraldajatele, ent mitte *lekseemiklassidele*[3]. Terminaalses tähestikus on fikseeritud *lekseemiklassi* kood (nt. *Trigol*is, et suvalise *identifikaatori* kood on 04) ja relatsioonid on fikseeritud *klassi* jaoks. Samas, translaatorit huvitab, milline on konkreetse antud klassi kuuluva elemendi kirjeldus. Sestap lisatakse klassi koodile "uue terminali" kood *k* (see omistatakse alates esimesest vabast koodist, s.o. *k* = viimase mitteterminali kood + 1)[4]. Järgmised lekseemiklasside esindajad kodeeritakse kui ++*k*. Näiteks, identifikaator *nimi* võib olla kodeeritud kui 04 89.

Niisiis, *Skanner* peab tegema kaht kardinaalselt erinevat tööd: esiteks, tuvastama sisendtekstis (lähteprogrammis) terminaalse tähestiku konkreetseid sümboleid, ja teiseks – kui järjekordne lekseem pole vahetult äratuntav sümbol, siis peab *Skanner* otsustama, millisesse lekseemiklassi ta kuulub ning menetlema teda programmiga dikteeritud korras.

Klassikas ([Aho, Ullman], [Lewis jt]) soovitatakse *Skanner* alati häälestada kui automaat, mis interpreteerib *regulaarsete grammatikate* poolt defineeritud keeltes kirjutatud *regulaaravaldisi.* Toome siinkohal primitiivse näite (vt. [Lebedev], lk. 224 jj.):

---

[1] Mõneti kurioossena tundub tõik, et *Python*i loojad otsustasid tabulatsioonist teha operaatorsulgude paari(de) asendaja. Aga − keelte autorid on oma valikutes vabad ning hea süntaksi kriteeriumid (vt. nt. [Isotamm, 2007], lk. 230 jj.) pole kuigivõrd siduvad.

[2] Võime Skannerit käsitleda nii, et iga tuvastatav lekseem on *regulaaravaldis*, mida saab kas aktsepteerida või ei, ja kui saab, siis tuleb ta *kodeerida vahekeelde.*

[3] Mõistagi, *kommentaarid* kuuluvad lihtsalt eiramisele − olles translaatori jaoks müra.

[4] Seega on sellised koodid kahearvulised: esiteks, klassi kood ja teiseks kood *k*.

```
<Пр> → КОТ <С>        //<lause> →KASS <midateeb>
<Пр> → ПЕС <С>        //<lause> →KOER <midateeb>
<Пр> → ОН <С>         //<lause> →TEMA <midateeb>
<С> → ИДЕТ            //<midateeb> →KÕNNIB
<С> → ЛЕЖИТ          //<midateeb> →LESIB
```

Automaat, mis on võimeline tuvastama selle grammatika poolt genereeritud sõnu, on lihtne[1] (vt. [Lebedev], lk. 225)[2]:

sisendsümbol/ /olek	КОТ	ПЕС	ОН	ИДЕТ	ЛЕЖИТ
<Пр>	<С>	<С>	<С>		
<С>				ОК	ОК

Joonis 6.3.1.b. Lihtne automaat. Tühi lahter on viga.

Tundub, et „ilmutatud" terminalide (mis esinevad oma ainuvõimalikul kujul produktsioonide paremates pooltes) tuvastamiseks pole vaja neid „üle" kirjeldada regulaaravaldistega, kuivõrd nad on juba defineeritud kontekstivaba grammatika produktsioonide hulgas. Ent identifikaatorid, konstandid, stringid ja kommentaarid on mõttekas defineerida (kas omaette antud tekstiga või kontekstivaba grammatika produktsioonide hulga koosseisus (*Konstruktorile* arusaadaval kujul)) *Chomsky* 3. klassi grammatikaga ning genereerida *Konstruktorisse* vastavaid regulaaravaldisi töötlevad automaadid[3].

Meie *TTS* suudab produktsioonide hulgaga antud terminale tuvastada automaatselt, ent identifikaatorite ja konstantide töötlus on „sisse programmeeritud": kui järjekordne lekseem pole ühene terminal (s.o. *pole* ei reservsõna ega eraldaja, aga *on Trigoli* jaoks kas klasside *#i#* (identifikaatorid) või *#c#* (konstandid) vastuvõetav esindaja[4]), siis *Skanner* peab identifikaatoriks tähega algavat (ja tähtedest või numbritest koosnevat ja tähestikus *V* fikseerimata) lekseemi ning konstandiks numbriga algavat ning numbritest koosnevat lekseemi.

*Skanner*i realiseerimisel tuleb lahendada üks (pisi)probleem, mida on keeruline programmeerida ka regulaaravaldisi töötleva automaadi abil. Kasutagem näitena *TTS*i õppe- ja näitekeelt *Trigol*, sissejuhatavalt toome järgnevas asjakohaseid fragmente *Konstruktori* logi-failist (*tri.htm*) ja ühe *Trigol*-programmi analüsaatori logist (P6p.htm). Esiteks, *Trigol*i terminaalne tähestik koos iga terminali koodiga:

```
Terminal alphabet
# 1 = #
# 2 = ;
# 3 = :
# 4 = #i#
# 5 = :=
```

[1] Juhime tähelepanu tõigale, et automaadi *olekute* arv seondub grammatika produktsioonide arvuga (lisanduda võib stardiolek) ning sisendsümbolite arv on määratud terminaalse tähestikuga.

[2] Nt., kui olekus <Пр> loeti sümbol *KOT*, siis minnakse üle olekusse <*C*>.

[3] Sellega sai hakkama *Nikita Šipilov* oma bakalaureusetöös [Šipilov], kavatsuste järgi pidanuks nende kaante vahel olema asjakohased peatükid (ja lisad) temalt, ent paraku jäime ajahätta. Nonde generaatorite programmid kirjutas ja silus *Nikita* ära.

[4] *Trigol*is pole ei stringe ega ka kommentaare. Terminaalsesse tähestikku kuuluvad sümbolid *#i#* ja *#c#* on mõeldud kasutamiseks kõikides antud *TTS*i abil realiseeritavate keelte grammatikates.

```
# 6 = GOTO
# 7 = +
# 8 = -
# 9 = *
#10 = /
#11 = #c#
#12 = (
#13 = )
#14 = <
#15 = >
#16 = <=
#17 = >=
#18 = /=
#19 = =
#20 = IF
#21 = THEN
#22 = READ
#23 = WRITE
```

Niisiis, ühesed koodid on kõigil terminalidel peale identifikaatorite klassi (*#i#*, klassi objektide koodi prefiksiks saab bait "4") ja konstantide klassi liikmetel (*#c#*, prefiks "11"). Viimase terminali kood on 23 ning mitteterminalide koodid algavad 24-st:

Nonterminal alphabet

```
#24 = `programm'
#25 = `programm12'
#26 = `operaatorid'
. . .

#49 = `loogilav9'
#50 = `loogilav10'
#51 = `loogilav11'
```

Konstruktor tuvastas ja fikseeris 51 koodi; esimene vaba kood (52) omistatakse transleeritava programmi esimesele identifikaatorile või konstandile. Seejuures lisatakse tolle koodiga objekt nii (seansiaegsetesse) tabelitesse *.v* (tähestik *V*) kui ka *.t* (terminalid), ent samuti skaneerimise ajal moodustatavatesse tabelitesse (*IT* või *KT*) järgmiste kirjeldustega[1]:

```
struct D *IT[50];
struct D *KT[50];
```

Nagu näha, kirjutab *Skanner* neisse tabelitesse järjekordse objekti (uus identifikaator või konstandi tekst) aadressi tähestiku *V* puus. *Analüsaatori* töö lõpus – kui süntaksivigu ei leitud – moodustatakse *IT* ja *KT* baasil uued tabelid (nende kirjete struktuure tutvustame osas *Translaator*), mis kirjutatakse ka kettale, nimelaienditega vastavalt *.it* ja *.kt.*

Milles on *Skanneri* probleem? Ta peab ilma "sisseprogrammeerimiseta" tuvastama eksimatult suvalise "ühese" terminali (kui ei õnnestu, siis tuleb valida klasside *#i#* ja *#c#* või vea vahel – aga see selgub hiljem, et kas oli viga). Niisiis, välistada tuleb "sisseprogrammeeritud ettevaatamine: näiteks, kui *Algol*-tüüpi keele *Skanner* leiab sümboli ":", siis oleks programmeeritud kontroll, et kas järgmine sümbol on "=" või ei. *TTS*i *Skanner* peab selle situatsiooni lahendama ilma programse sekkumiseta. Mis on ka loomulik, kuivõrd *Skanner* ei „tea" midagi ei süntaksist ega ka semantikast, tema "*teab* ainult leksikat.

[1] Struktuur *D* kirjeldab grammatika *V* puu tippu, vt. eestpoolt.

Skaneerimise saaks lihtsaks teha programmeerimiskeele süntaksi abil, kehtestades ranged kitsendused – näiteks nii, et *kõik* lekseemid evivad erieraldajaid; lihtsaimal juhul, iga kahe lekseemi vahel peab olema vähemalt üks tühik (erijuhul reavahetus). Näiteks märgendatud omistamisoperaator *Algol*-tüüpi keeles võiks välja näha nii:

```
M  :  A  :=   B  +  7  ×  C  ;
```

Veelgi jõulisem variant oli kasutusel näiteks *Malgol*is, kus võtmesõnad (mis kõik olid üksiti reservsõnad) lõppesid markeriga "apostroof", näiteks:

```
FOR'I:=1 STEP'1 UNTIL'N DO'BEGIN'
```

Paraku pole ei eraldajate ega ka markeritega variandid programmeerijasõbralikud – programmeerijad püüavad vältida keeli, kus nad peavad kirjutama rohkem sümboleid kui hädapärast vaja[1].

Erikitsendusteta skaneerimise teevad keeruliseks liiteraldajad (nt. lihteraldajad : ja < ühelt ja liiteraldajad := ja <= teiselt poolt) ning näiteks *C*-keele konstruktsioonid, nagu *a*++ või – –*b*.

Toogem näiteks taas *Trigol*i, sedakorda *Trigol*-programmi *P6.tri* logifaili (*P6p.htm*) baasil; allpool esitame asjakohaseid fragmente tollest failist.

Scanner started

Input program:
#READ n;F:=1;I:=0;M1:I:=I+1;IF I>n THEN GOTO M2; F:=F*I;GOTO M1;M2: WRITE F#

Scanned program:
1 22 4 52 2 4 53 5 11 54 2 4 55 5 11 56 2 4 57 3 4 55 5 4 55 7 11 54 2 20 4 55 15 4 52 21 6 4 58 2 4 53 5 4 53 9 4 55 2 6 4 57 2 4 58 3 23 4 53 1

Identifiers: n, F, I, M1, M2;

Constants: 1, 0;

Nagu näeme, on omistamine `I:=0;` skaneeritud kui

```
4 55 5 11 56 2
```

ning `M1:I:=I+1;`

kui `4 57 3 4 55 5 4 55 7 11 54 2`.

Seega, eraldaja ***:*** on kodeeritud kui *3* ning *:=* kui *5* – just nii nagu vaja. Meie *TTS*i *Skanner* kasutab järgmist võtet: terminaalse tähestiku V_T vektori *tri.t* baasil on moodustatud skaneerimisvektor *tri.tt*, mis kujutab endast terminaalse tähestiku sorteeritud esitust elementide pikkuste mittekasvavas järjekorras[2] – nii, nagu esitatud järgmises logifaili *trird.htm* fragmendis:

---

[1] Sellest aspektist on „laisale kirjutajale" ideaalne keel *FORTRAN*, kus eraldajaid pole *üldse* vaja kirjutada, näiteks nii: IF(IF.EQ.3)GOTO3. Mõistagi, nii v*õib* kirjutada, aga ei pea, kui see hakkab segama kirjutatu mõttest aru saamist.

[2] Mõistagi, seegi vektor sisaldab *viitasid* lekseeme esitavatele stringidele.

Scanner Table

File TRI.tt

```
WRITE THEN READ GOTO #c# #i# /= IF := <= >= ( ) < > ; : + = - * / #
```

Iga järjekordse lekseemi tuvastamist alustatakse sisendprogrammi läbivaatamata osa algusest alamstringi otsimisega; noiks alamstringideks on *.tt*-vektori elemendid (vasakult paremale) – nii on garanteeritud, et näiteks ":=" leitakse varem kui ":". Kui järjekordne lekseem ei ole ükski vektori *tt* element, siis peetakse seda võimalusel kas identifikaatoriks või konstandiks. Skanneri täpne algoritm (aga selleks saab olla ainult *programm*) on esitatud lisas 7.

Rõhutagem veel kord, et meie *TTS* (sh. nii *Analüsaator* kui ka selle esimene plokk – *Skanner* – töötavad samamoodi kõigi *kontekstivabade BRC-redutseeritavate eelnevusgrammatikate* puhul, mööndusega, et *Skannerisse* on sisse programmeeritud lekseemiklasside "Identifikaatorid" ja "Konstandid" tuvastamine (häälestudes õppekeele *Trigol* kirjeldusele) ning katteta on klassid "Stringid" ja "Kommentaarid" – seega, kui meie *TTS*i tahab kasutada keegi, kelle realiseeritav keel tingib kahe viimase klassi lisamist[1] ja/või kahe esimese klassi modifitseerimist, siis tuleb tal ka *Skanneri C*-koodi modifitseerida.

Nende lugejate jaoks, keda huvitab täpselt, kuidas *Analüsaator* on realiseeritud ja tutvuvad *Skanneri* koodiga, ütleme, et *Skanneri* viimase tööna lisatud terminal *DUMMY* (selle sõna üks tähendusi on "reaalse objekti imitatsioon", *tont*) on vajalik meie *TTS*i *Parseri* töö jätkamiseks veasituatsioonis.

6.3.2. Parser

Parser (sisuldasa *süntaktiline analüsaator*) on meie *Analüsaatori* teise ploki tinglik nimetus. Esimene plokk, *Skanner*, on leksika-analüsaator, mis teisendab põhitööna sisendprogrammi vahekoodi –, viimane on *Parser*i[2] sisendkeeleks. *Parseri* algoritm on plokkskeemi kujul esitatud ülalpool, joonisel 3.3.b. Seal on jäetud konkretiseerimata plokk, mis saab juhtimise siis, kui magasini tipmise sümboli ja vahekoodi aktiivse sümboli vahel puudub relatsioon, s.o.

$M \ \Re \ T_i = \odot,$

seal lihtsalt katkestati töö.

Õppegrammatikal baseeruva keele sõna analüüsi puhul tulebki nii toimida. Kui aga analüüsitakse programmeerimiskeele sõna (programmi), siis pole see tee vastuvõetav[3] ning tuleb valida mingi sisseprogrammeeritud strateegia töö jätkamiseks, et Analüsaator vaataks läbi kogu sisendteksti ning püüaks avastada võimalikult palju olemasolevatest süntaksivigadest.

---

[1] *Trigoli* jaoks on seda teinud õppetöö käigus paljud üliõpilased, ent neid täiendusi pole süsteemi lülitatud. Põhjused on lihtsad: esiteks, *Trigol* on *väga* lihtne õppevahend, mille tabelid ja realisatsioon peavad olema minimaalse mahuga ja teiseks, kõigile täiendamishuvilistele tuleb luua võrdsed võimalused. Edukat laiendamisnäidet vt. Lisas 11.

[2] Seega, kui keegi tahab reaalselt läbi mängida mõne õppegrammatika sõna analüüsi, peaks ta seda tegema vahekoodi kasutades.

[3] Omal ajal ringles (usutavasti pahatahtlik) folkloor, et just nii käituvad varajased *COBOL*-translaatorid: väljastavad teate: „*ERROR IN INPUT-PROGRAM*" ilma täpsustuseta, ning lõpetavad töö.

Ideaalne oleks, kui *Analüsaator parandaks* avastatud vead nii, et resultaadiks oleks nii süntaktiliselt kui ka semantiliselt korrektne programm, mis *teeks* seda, mida programmeerija *mõtles*, ja *mitte seda*, mida ta ei soovinud (mis võib juhtuda, kui jätta süntaksivigade *parandamine* translaatori pädevusse). Erandiks on võib-olla ortograafiavead reservsõnades: näiteks, kui *BEGIN* asemele on kirjutatud *PEGIN* või *BEGINN*, siis oleks mõeldav vea parandamine. *David Gries* [Gries, lk. 354–356] refereerib *D. Freeman*i artiklit "*Error correction in CORC: the Cornell computing language. Thesis, Cornell University, 1963*", kus on osutatud neljale tüüpsituatsioonile:

1) üks täht sõnas on valesti kirjutatud (või perforeeritud);
2) üks täht on vahele jäänud;
3) üks täht on liigne;
4) kaks kõrvutiolevat tähte on vahetuses.

Gries möönab, et ortograafiavigade parandamine on tõenäosem reservsõnade ja eraldajate puhul ning kahtlasem identifikaatorite puhul. Tegelikult on ortograafiavigade võimaliku avastamise koht *Skanneris* ja konkreetsele keelele orienteeritud *Analüsaatoril* on seal kahtlemata võimalusi. Süntaksorienteeritud *Analüsaatoril* need paraku puuduvad[1].

Niisiis, *Parseri* ülesanne on fikseerida viga ja püüda vältida analüüsi jätkamisel sekundaarseid vigu – need väljenduvad tegelikult *olematute* süntaksivigade leidmises programmist ajal, mil *Parser* püüab neutraliseerida *tegelikku* viga.

Alt-üles-analüüsi (aga seda meetodit järgib meie *TTS*) jaoks esitab *David Gries* [Gries, lk. 360], tuginedes tolleks ajaks levinud lahendustele, neli võimalust (situatsioon on *xTt*, kus *x* on magasinis olev koodide vektori tipmine sümbol, *T* on vahekoodi "aktiivne sümbol" ja *t* on vahekoodi vektori ülejäänud osa, need kõik on terminalid):

1) eemaldada *T* ja püüda jätkata;
2) kirjutada *x* ja *T* vahele terminaalsete vahekoodide vektor *q* – resultaat on *xqTt* – ja alustada kohalt *q.Tt.* Nii saame töödelda lõpuni vektori *q.T* – enne uue vea avastamist;
3) teha sama, mis eelmises variandis, ent alustada kohalt *.T*;
4) kustutada *x* lõpust ükshaaval sümboleid (uus *x* on taas magasini tipus).

Gries nendib, et kaks viimast varianti pole vastuvõetavad, kuivõrd nende puhul eliminеeritakse varasema analüüsi (võimalik, et korrektsed) resultaadid, ja osutab, et kaks esimest varianti on kasutatavad, ning et mõlemal on omad plussid.

Meie *TTS* valis variandi, kus eeldatatakse, et kui programmis on tehtud viga, siis on see tehtud varem kui järgmine veakandidaat, seega on meie lähenemine pigem (2) kui muud. Kodeeritud on algoritm, kus vaadatakse eelnevusmaatriksis läbi relatsioonid magasini tipmise elemendi ja *terminaalse tähestiku* sümbolite vahel ning valitakse neist esimene mittetühja relatsiooniga terminal täitma sisendrea aktiivse sümboli rolli. Sel moel on magasinis aktsepteeritav lause vormi baasi prefiks (või lause vormi baas ise, kui lisatud terminal järgneb magasini tipusümbolile).

Kui lisatud fiktiivne terminal juhtus olema klasside *#i#* või *#c#* esindaja, siis moodustatakse analüüsi puu uus tipp koodiga *DUMMY* (vt. eelmise jaotise 6.3.1 viimast lõiku). Toome näiteks vigase *Trigol*-programmi *Pv1.tri* ja fragmente selle analüüsist.

[1] Nt oletagem, et *Algol*-tüüpi keele skanner leiab sõna *begun* ja parandab selle *begin*iks – ent programmeerija mõtles kasutada venekeelset muutujat tõlkenimega "jooksja". Segadust kui palju…

```
#F:=;:=0;
::=+;
IF I>5 THEN GOTO GOTO M2;
F:=F*I;
GOTO M1;
M2: F:=F#
```

Tuletagem meelde, et *Skanner* ei tea süntaksist midagi ja kodeerib vasakult paremale kõik lekseemid (vajadusel tuvastades identifikaatoreid või konstante). Allpool toome tolle meelega vigasena kirjutatud programmi analüüsi logi.

Start of TRI Parser for a Program PV1.tri at Thu Sep 07 17:52:14 2006

Scanner started

Input program:

F := ; := 0 ; : := + ; IF I > 5 THEN GOTO GOTO M2 ; F := F * I ; GOTO M1 ; M2 : F := F

Scanned program:

1 4 52 5 2 5 11 53 2 3 5 7 2 20 4 54 15 11 55 21 6 6 4 56 2 4 52 5 4 52 9 4 54 2 6 4 57 2 4 56 3 4 52 5 4 52 1

Identifiers:F, I, M2, M1;

Constants:0, 5;

Scanner ended

Nagu näeme, ei reageerinud *Skanner* mingilgi moel ühelegi süntaksiveale. Küll aga teeb seda *Parser*, käesoleval juhul nii:

Stack & Word	#	<•muutuja	=•:=	;	:=	#c#	0	;	:	:=	+	;	IF	#i#	I	>	#c#	5	THEN

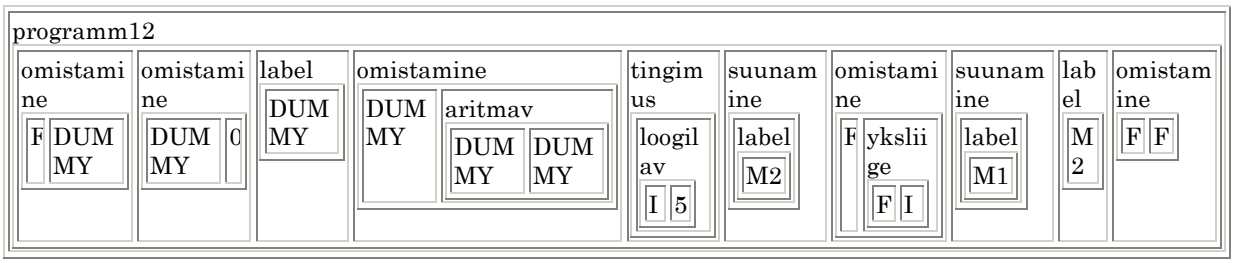
no relationship between the symbols := and ;

The relationships of :=
:= < #i#
:= < #c#
:= < (
:= = omistamine1
:= = loogilav
:= < aritmav
:= < yksliige
:= < tegur

I'll try to correct..

Stack & Word	#	<•muutuja	=•:=	<•#i#	;	:=	#c#	0	;	:	:=	+

Nagu näeme, kirjutas *Parser* magasini tippu terminali *#i#* (koodiga *DUMMY*, seda pole küll näha, aga me teame) ja programmi *Pv1* analüüsi puu on lõpuks järgmine:



Puu vastab järgmisele programmile:

```
F:=DUMMY; DUMMY:=0; DUMMY: DUMMY:=DUMMY+DUMMY; IF I>5 THEN GOTO M2;
F:=F*I; GOTO M1; M2: F:=F;
```

Juhime tähelepanu kahele seigale:

- *Analüsaator* ei püüagi vigu parandada (siiski, liigne *GOTO* on eemaldatud);
- ühegi vea mõjuulatus pole ülemäära suur, need jäävad igal juhul ühe operaatori raamidesse.

Üpris lihtne oleks teha "jätkusümboli" valimine ja kirjutamine magasini tippu ratsionaalsemaks: võiks võtta magasinist lause vormi baasi algus ja vaadata selles kontekstis, mis sümbol oleks aktsepteeritav, lähtudes tolle baasi jätkamise aspektist. Teisisõnu, me ei valiks tolleks sümboliks esimest ettejuhtuvat, vaid esimese sisuliselt sobiva. Üliõpilastele: siin on võimalus *TTS*i laiendamiseks. Miks ka mitte teadustööks.

Aga – peale veasituatsioonis jätkamise – pole *Analüsaatoril* (sh. süntaksorienteeritul) probleeme, ja kui vigu ei leita, siis väljundiks on (hõre) analüüsi puu ja (programmeerimiskeele puhul) identifikaatorite ja konstantide (vajadusel ka stringide) tabelid.

7. Translaator

Translaator on programm, mis teeb sisendprogrammi *lahendatavaks.* Meie *TTS*i kontseptsiooni kohaselt saab *translaator* ette *Analüsaatori* väljundi (analüüsi puu ja tabelid) ning interpreteerib noid andmestruktuure; meie *TTS*i kontekstis tähendab see, et edasine tuleb programmeerida "käsitsi"[1] ning keele realiseerijatel on valida kahe variandi vahel: kas programmeerida *interpretaator* või *kompilaator*. Et neid teid on eelnevalt juba korduvalt kirjeldatud ja võrreldud, siis tutvustame allpool variante meie *TTS*i näitekeele *Trigol* baasil.

Niisiis, translaator *interpreteerib* sisendprogrammi analüüsi puud, mille "ehitas" *süntaktiline Analüsaator*, mis garanteerib tolle programmi *süntaktilise* õigsuse, s.o. vastavuse produktsioonidega antud süntaksireeglitele. Ent peale nende reeglite on keele mitteformaalse kirjeldusega fikseeritud tavaliselt palju kitsendusi, mida süntaksiga ei saa esitada. Näiteks: märgendid peavad olema unikaalsed, suunata saab ainult olemasolevale märgendile või et muutujatele ei ole mõtet omistada väärtusi, mida programmis ei kasutata. Sellised seigad seonduvad mõistega *semantiline analüüs* ja sellega tegeleb *translaator*.

7.1. Andmestruktuurid

Täpsustagem *translaatori* (*interpretaatori* või *kompilaatori*) andmestruktuure. Analüüsi puu tipu (tegelik) kirjeldus on järgmine:

```
struct top{
      int kood;    /* vahekeelne kood V_T või V_N */
      int leks;    /* kui tipp=ident/const, siis selle jrk-nr. */
      int sem;     /* semantikakood */
      int label;   /* kui tipp on märgendatud operaator – märgendi nr */
      int truel;         /* kompilaator: go to true */
      int falsel;        /* kompilaator: go to false */
      struct top *up;    /* puuviidad: üles, */
      struct top *right; /* naabrile ja */
      struct top *down;  /* alluvale */
      };
```

Identifikaatorite tabel *IDT* ja konstantide tabel *CT* koosnevad järgmiste kirjeldustega kirjetest:

```
/* identifikaatorite tabeli kirje */
struct itr{
      int nr;          /* jrk-nr tähestikus V */
      int loc;         /* reservis[2] */
      int value;       /* interpretaator: muutuja väärtus */
      struct top *t;   /* kui id=märgend: viit operaatorile */
      int io;          /* 0, kui pole, 1: read, 2: write, 3: r&w[3] */
      };

/* konstantide tabeli kirje */
struct ctr{
      int nr;          /* jrk-nr tähestikus V */
      int loc;         /* reservis */
      int value;       /* konstandi väärtus */
      };
```

[1] Alternatiivi näiteks sobib translaator *Modula-2 → FORTH.*

[2] Loe: realisatsiooni kavandades tundus see väli mõttekana, ent programmeerides enam mitte. Mis aga ei tähenda, et süsteemi kasutajad ei võiks seda välja ikkagi oma huvides mõtestada. Sama kehtib välja „ctr->loc" puhul.

[3] See on oluline, kui muutuja väärtus tuleb .exe-faili täitmisel klaviatuurilt sisestada või displeile trükkida.

*Trigol*i programmide analüüsi puu interpreteerimisel juhindutakse järgmisest puusemantikast:

```
#define muut 1      /* #i# - identifikaator */
#define konst 2     /* #c# - konstant */
#define pisem 3     /* < */
#define suurem 4    /* > */
#define piv 5       /* <= TTpisem-võrdne*/
#define suv 6       /* >= suurem-võrdne*/
#define pov 7       /* /= pole võrdne*/
#define vord 8      /* = võrdne*/
#define omist 10    /* omistamisoperaator */
#define jag 11      /* jagamistehe */
#define korrut 12   /* korrutamistehe */
#define lahut 13    /* lahutustehe */
#define liit 14     /* liitmistehe */
#define labl 15     /* märgend */
#define suunam 16   /* suunamisoperaator */
#define kuisiis 18  /* if-lause */
#define tingop 19   /* tingimusoperaator */
#define lugem 20    /* lugemisoperaator */
#define kirjut 21   /* kirjutamisoperaator */
```

Nii interpretaator kui ka kompilaator ei vaja analüüsi puu läbimiseks puuviitade magasini (seda võimaldab *viit üles*), ent sisuline töö käib magasini abil (sinna salvestatakse rippuvate tippude andmeid ja tehete resultaate). Magasini elemendi formaat on järgmine:

```
/* interpretaatori või kompilaatori magasinielement */
struct item{
      int liik;          /* 0 töömuutuja, 1 id, 2 c, 3 top */
      int index;         /* töömuutuja väärtus  */
      struct itr *id;    /* liik = 1 (muutuja)  */
      struct ctr *c;     /* liik = 2 (konstant) */
      struct top *t;     /* liik = 3 (märgend)  */
      };

 struct item *stack[20]; /* interpretaatori või kompilaatori magasin */
```

Töömuutujaid vajab translaator aritmeetika- ja loogikatehete vahetulemuste hoidmiseks.

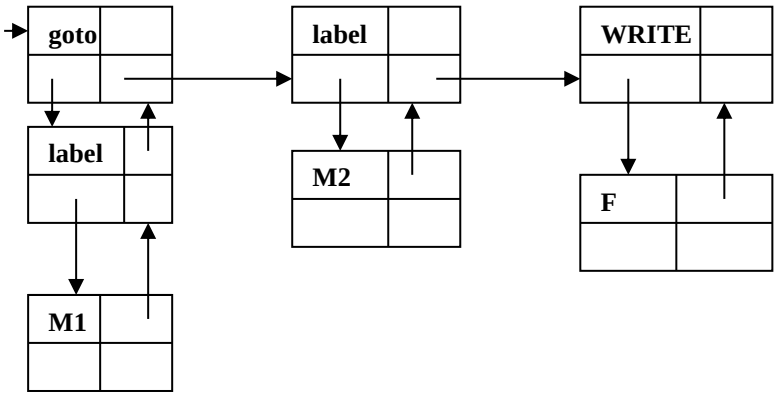
7.2. Interpretaator

*Trigol*i interpretaatori põhitöö on sisendprogrammi analüüsi puu läbimine ja selle käigus resultaatide arvutamine. Ent sellele põhitööle eelneb märgendite ja suunamiste töötlus, mille käigus kontrollitakse märgendite unikaalsust ja seda, kas kõik suunamised on olemasolevatele märgenditele, modifitseeritakse identifikaatorite tabelit ning teisendatakse analüüsi puud.

Eirame järgnevatel joonistel analüüsi puu tippude tegelikke formaate ning kasutame juba varasemast tuttavat lihtsustust: tipus on nimi ja kolm puuviita. Võtkem näiteks programm *P6.tri*:

```
#READ n; F:=1; I:=0;
M1: I:=I+1;
IF I>n THEN GOTO M2;
F:=F*I;
GOTO M1;
M2: WRITE F#
```

Selle programmi kahe viimase rea puu on kujutatud joonisel 7.2.a.



Joonis 7.2.a. Suunamine ja märgendid analüüsi puu fragmendis.

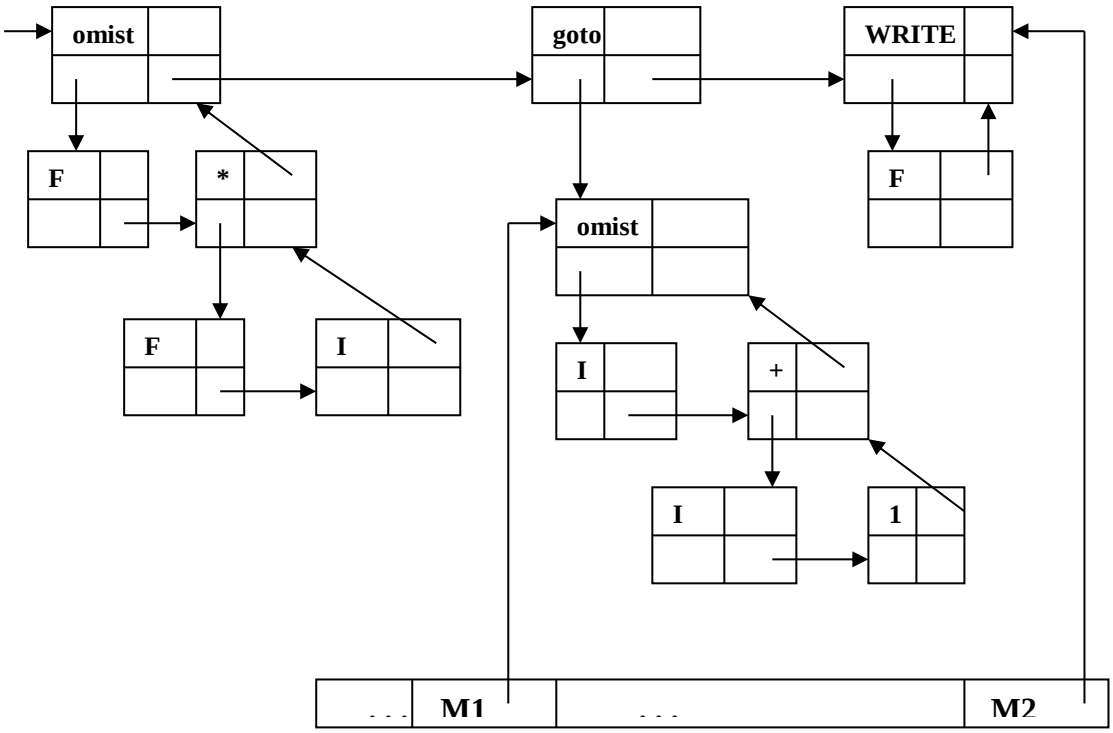
Loodetavasti lugeja nõustub, et interpretaatori jaoks pole sel joonisel esitatud puu kuigi ratsionaalne: esiteks, "goto" alampuus pole midagi mõistlikku teha vahetasemel "label" ning tipus "M1" tuleks hakata tegelema märgendatud operaatori otsimisega, ja teiseks, "labeli" alampuu raiskab ainult interpretaatori aega[1]. Sestap teisendatakse analüüsi puud ning info märgendatud objektide kohta salvestatakse identifikaatorite tabelisse (vt. joonis 7.2.b): puu (fragmendid) on joonise ülaosas ning identifikaatorite tabeli asjassepuutuv fragment on joonise alumises servas.

Näeme, et "goto"-tipu alluv-viit osundab märgendiga *M1* tähistatud omistamisoperaatorile, operaatoriga "WRITE" seonduvat "label"-alampuud enam ei ole ning identifikaatorite tabeli elemendid *M1* ja *M2* on saanud *väärtused* – neiks on märgendatud operaatorite aadressid (füüsilised viidad).

Sellise tulemuse saavutamiseks tuleb programmi analüüsi puust esmalt leida märgendatud operaatorid ja kanda nende aadressid identifikaatorite tabelisse (kui selgub, et märgendil juba on väärtus, väljastatakse veateade ja modifitseeritakse vigade indikaatorit), seejärel – kui identifikaatorite tabelis on kõikide märgendite väärtused – otsitakse puust kõik suunamisoperaatorid ja kontrollitakse, kas on kasutatud väärtustatud märgendit (kui ei, fikseeritakse viga) ja seejärel – kui eelmistel sammudel vigu ei leitud – teisendatakse analüüsi puu interpreteerimiseks mugavale kujule.

Märkida tuleks vist sedagi, et meie *puu* pole rangelt võttes kasutatava andmestruktuuri jaoks kohaldatav termin – meie struktuur on *tsükliline graaf*: enne teisendamisi rikkusid puustruktuuri *üles*-viidad, teisendused lisasid aga lisaviidad suunamisoperaatoritega määratud tippudele.

[1] Tegemist pole analüüsi puu moodustamise juhtimise (fail *tri.sem*) veaga: *Analüsaator* peab tuvastama *märgendi* nii operaatori nime kui ka suunamisoperaatori argumendi rollis, seetõttu on paratamatu, et mitteterminali *label* definitsioonil on semantika.



Joonis 7.2.b. Modifitseeritud puu fragment.

Trigol-programmide interpretaator on vormistatud protseduurina *trigol_Int(struct top *root*), kus **root* on viit analüüsi (modifitseeritud) puu tipule. Tähtsamatest andmestruktuuridest kasutab Interpretaator järgmisi:

```
struct top *t;     //Analüüsi puu tipp
struct ctr *c;     //Konstantide tabeli CT kirje
struct itr *id;    //Identifikaatorite tabeli IDT kirje
struct D *leks;    //Tähestiku V puu tipp
struct item *s;    //Magasinielement
```

Interpretaatori magasini elementidele võetakse (ja tagastatakse) mälu dünaamiliselt (võetakse protseduuriga `make_item()`,võrdlus- ja aritmeetikatehete sooritamiseks kasutatakse protseduure *loogik* ja *aritm* ning omistamiseks väärtuse lugemise ja kirjutamise protseduure. Allpool esitame *Trigol-interpretaatori* põhiprogrammi koodi[1] – lootes, et see on selle leheküljeni jõudnud lugejale loetav (keel on tavaline *C*[2]). Interpretaatori täistekst on toodud lisas 8.

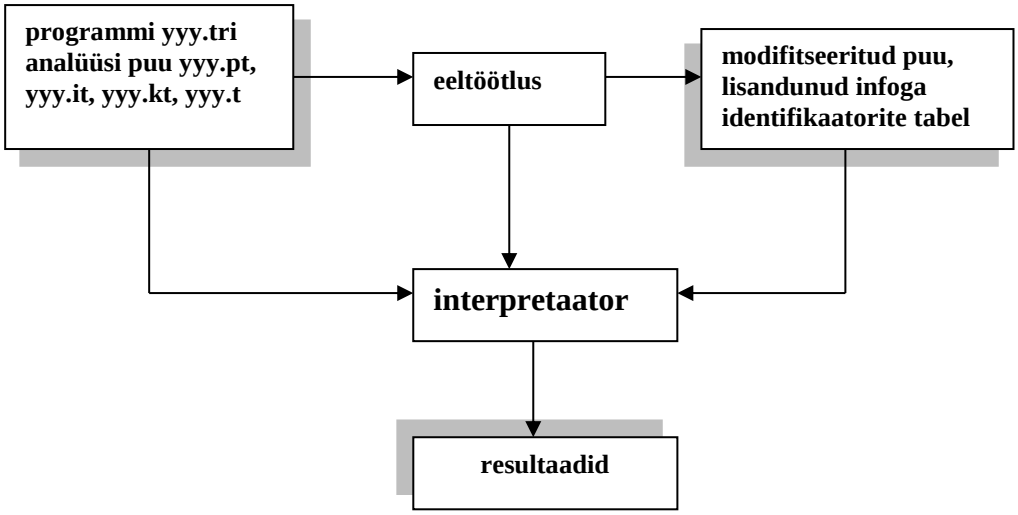
```
/* TRIGOL-keelsete programmide interpretaator  */
void trigol_Int(struct top *root){
      struct item *s;
      struct top *t;
      struct ctr *c;
      struct itr *id;
      int op;
      int i=0;
      int j,k,x;
```

[1] Nende ridade autorile ei meeldi termini *kood* kasutamine sisendkeelse teksti tähenduses, ent mis parata: just samamoodi nagu mõiste *õpetaja* uus sisu sundis pastoreid kasutama ametinime *kirikuõpetaja*, peame tänapäeval kirjutama *masinkood*, kui mõtleme *koodi*.
[2] Seega, eelis on neil, kes on selle kursuse läbinud. Ja neile peaks see raamat pakkuma lisamaterjali.

```
t=root->down;
      for(j=0;j<20;j++) stack[j]=(struct item *)NULL;
      fprintf(Logi,"<HR>");  /* interpretaator genereerib HTML-
                  formaadis logi-faili teksti. */
down: p_label(t);
      if(t->down!=(struct top *)NULL){
            t=t->down;
            goto down;
            }

/* leht => stack  */
      s=make_item( );
      op=t->sem;
      switch(op){
            case muut:{
                  s->liik=1;                          /* muutuja */
                  for(k=0;k<itl;k++){
                        if(IT[k]==t->leks){
                              id=IDT[k];
                              goto iok;
                              }
                        }
                  iok: s->id=id;
                  if(id->t!=(struct top *)NULL){
                        s->liik=3;                /* märgend */
                        s->t=id->t;
                        }
                  break;
                  }
            case konst:{
                  for(k=0;k<ktl;k++){
                        if(KT[k]==t->leks){
                              c=CT[k];
                              goto cok;
                              }
                        }
                  cok: s->c=c;
                  s->liik=2;                          /* konstant */
                  break;
                  }
            default:{
                  fprintf(Logi,"parsing tree is incorrect<BR>");
                  goto jokk;
                  }
            }
      stack[i]=s;
      i++;
      prist(I,0);   //magasini trükk logifaili
naaber:
      if(t->up==(struct top *)NULL){
            t=t->right;
            goto down;
            }
      t=t->up;
      if(t==root) goto ok;
      fprintf(Logi,"</FONT>interpreting the operator  ");
      print_Op(t);  //tehte trükk logifaili
      pp2html(t);   //alampuu trükk logifaili, autorid Jesmin ja Lehes
      op=t->sem;
      if(op>=pisem && op<=vord){
            loogik(I,op);
                  i-=1;
                  goto pimm;
                  }
      if(op>=jag && op<= liit){
            aritm(I,op);
                  i-=1; goto pimm;
                  }
```

```
      switch(op){
            case omist:{              /* omistamine */
                  x=get_x(I,1);
                  write_x(x,i);
                  freestack(I,2); //magasinielementide mälu vabastamine
                  i-=2;
                  break;
                  }
            case suunam:{                /* suunamine */
                  s=stack[i-1];
                  t=s->t;
                  id=s->id;
                  if(logi==1) fprintf(Logi,"goto %s<BR>",T[id->nr]);
                  freestack(I,1);
                  i-=1;
                  goto down;
                  }
            case kuisiis:{
                  x=get_x(I,1);
                  freestack(I,1);
                  i-=1;
                  if(x==0){
                        t=t->right;
                        goto naaber;
                        }
                  break;
                  }
            case tingop: break;  //semantikafailis seda koodi pole
            case lugem:{
                  s=stack[i-1];
                  id=s->id;
                  printf("\aInput %s= ",T[id->nr]);
                  scanf("%d",&x);
                  if(logi==1)fprintf(Logi,"Input%s=%d<BR>",T[id-nr],x);
                  id->value=x;
                  freestack(I,1);
                  i-=1;
                  break;
                  }
            case kirjut:{
                  s=stack[i-1];
                  id=s->id;
                  fprintf(Logi,"Output %s=%d<BR>",T[id->nr],id->value);
                  freestack(I,1);
                  i-=1;
                  break;
                  }
            }

pimm:
      prist(I,0);
      goto naaber;
ok:
      fprintf(Logi,"program %s is completed<BR>",pr_name);
jokk:
      print_variables();
}
```

Trigol-interpretaatori töö võtab kokku joonis 7.2.c.



Joonis 7.2.c. Interpretaatori plokid ja andmevood.

Allpool toome ära meie tuttava näiteprogrammi *P6.tri* lahendamise ekraanitõmmise (joonisel 7.2.d) ning interpretaatori logi alguse ja lõpu.

Joonis 7.2.d. *P6.tri* interpretaatori (int32) väljakutse ning tulemused.

Start of Interpreter for program p6.tri at Sun Oct 09 18:04:36 2011
Program

```
# READ n;
F:=1;
I:=0;
M1: I:=(I+1);
IF (I>n) THEN GOTO M2;
F:=(F*I);
GOTO M1;
M2: WRITE F#
```

Parsing tree

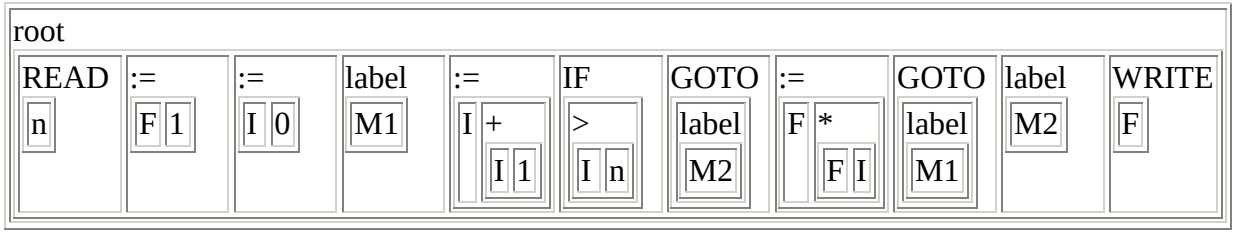


Table of constants

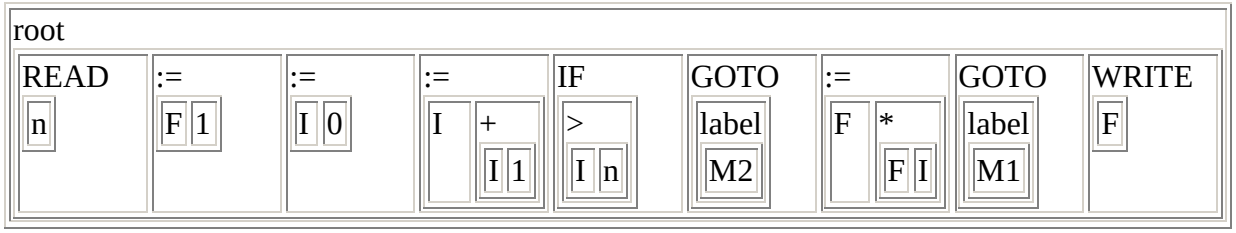
c1=1	c2=0

Table of identifiers

i1=n	i2=F	i3=I	i4=M1	i5=M2

label 'M1' is address of the operator {omistamine} (00915698)
label 'M2' is address of the operator {kirjutamine} (009157B8)

Modified tree



Stack	id n=0

interpreting the operator READ n

Input n=6

Stack

Stack | id F=0

Stack | id F=0 | const=1

interpreting the operator F:=1

F:=1

Stack

Stack | id I=0

Stack | id I=0 | const=0

interpreting the operator I:=0

I:=0

Stack

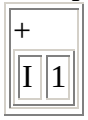
M1:

Stack | id I=0

Stack | id I=0 | id I=0

Stack | id I=0 | id I=0 | const=1

interpreting the operator (I+1)



1 = 0 + 1

Stack	id I=0	tm=1

interpreting the operator I:=(I+1)

:=	IF	GOTO	:=	GOTO	WRITE
I +	>	label	F *	label	F
I 1	I n	M2	F I	M1	

I:=1

Stack

Stack	id I=1

Stack	id I=1	id n=6

interpreting the operator (I>n)

>
I n

1 > 6 ?

Stack	tm=0

interpreting the operator IF (I>n) THEN

IF	GOTO	:=	GOTO	WRITE
>	label	F *	label	F
I n	M2	F I	M1	

Stack	id F=1

Stack	id F=1	id F=1

Stack	id F=1	id F=1	id I=1

interpreting the operator (F*I)

*
F I

1 = 1 * 1

Stack	id F=1	tm=1

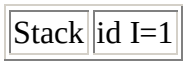
interpreting the operator F:=(F*I)

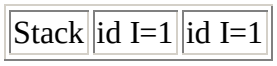
F:=1

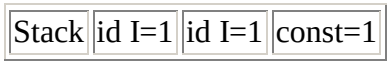
interpreting the operator M1:

interpreting the operator GOTO M1

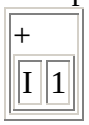
goto M1
M1:



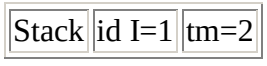




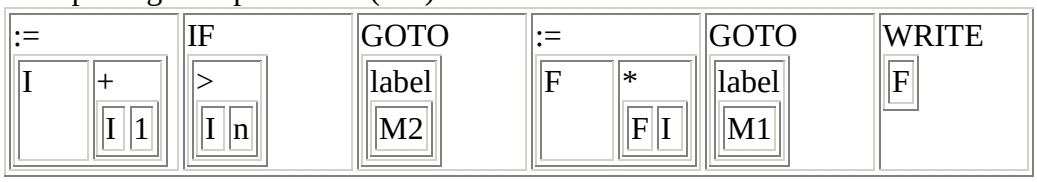
interpreting the operator (I+1)



2 = 1 + 1



interpreting the operator I:=(I+1)



I:=2

Stack

Stack	id I=2

Stack	id I=2	id n=6

interpreting the operator (I>n)

>
I n

2 > 6 ?

Stack	tm=0

interpreting the operator IF (I>n) THEN

IF	GOTO	:=	GOTO	WRITE
>	label	F *	label	F
I n	M2	F I	M1	

Stack	id F=1

Stack	id F=1	id F=1

Stack	id F=1	id F=1	id I=2

interpreting the operator (F*I)

*
F I

2 = 1 * 2

Stack	id F=1	tm=2

interpreting the operator F:=(F*I)

:=	GOTO	WRITE
F *	label	F
F I	M1	

F:=2

(*Jätame vahele sammud I=3..6 ning jätkame interpreteerimise lõpuga*)

Stack	id F=120	id F=120	id I=6

interpreting the operator (F*I)

*
F I

720 = 120 * 6

Stack	id F=120	tm=720

interpreting the operator F:=(F*I)

:=		GOTO	WRITE
F	*	label	F
	F I	M1	

F:=720

Stack

Stack	M1:

interpreting the operator M1:

label
M1

Stack	M1:

interpreting the operator GOTO M1

GOTO	WRITE
label	F
M1	

goto M1
M1:

Stack	id I=6

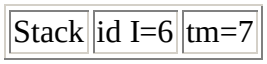
Stack	id I=6	id I=6

Stack	id I=6	id I=6	const=1

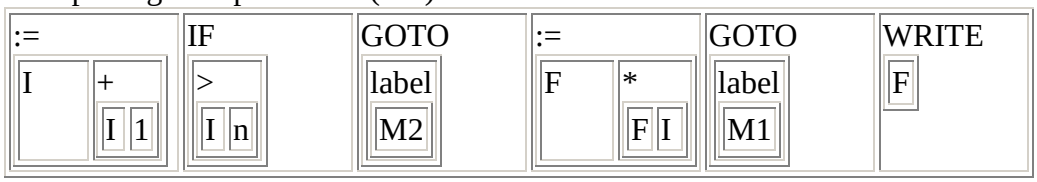
interpreting the operator (I+1)

+

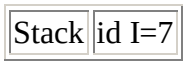
7 = 6 + 1

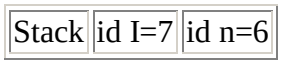


interpreting the operator I:=(I+1)

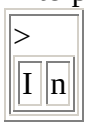


I:=7



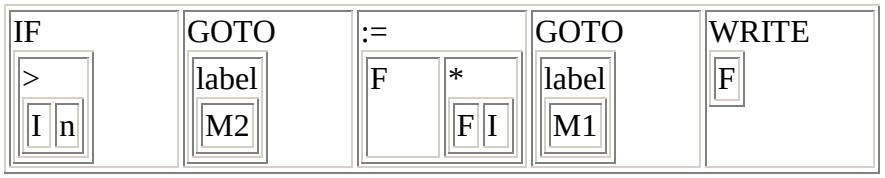


interpreting the operator (I>n)



7 > 6 ?

interpreting the operator IF (I>n) THEN



interpreting the operator M2:

interpreting the operator GOTO M2

goto M2
M2:

interpreting the operator WRITE F

Output F=720

program p6.tri is completed

THE VARIABLES:
n=6
F=720
I=7

Interpreter ended at Sun Oct 09 18:04:41 2011

7.3. Kompilaator

Tuletagem meelde: interpretaator liigub lähteprogrammi analüüsi puus üldjuhul tsükliliselt ja lõpetab oma töö interpreteeritava programmi resultaatide fikseerimisega (väljastamise või salvestamisega). Kompilaator seevastu läbib – võimalik, et pärast eeltöötlust – analüüsi puu ühekordselt ja genereerib selle läbimise käigus lähteprogrammiga adekvaatse koodi mingis teises keeles. Kuni objektmasinate masinkoodid olid lihtsad ja läbipaistvad (nagu *Minsk-32* või *IBM/360* puhul), siis kompilaatorite väljundiks oli reeglina *masinkood*. Tänapäeval (masinkood on ülekattega nagu *Intel*il) on kompilaatori objektkoodi rollis tavaliselt *assembler*. Nii on see ka meie *TTS*i *Trigol*-kompilaatori puhul: objektkood on mikroprotsessori assembler[1].

[1] 2011. aasta novembrini oli see meie *TTS*is *TASM*i (*Borland*i *Turbo Assembler*) 16-bitine variant, mida suutis kasutada ka 32-bitine arhitektuur (nt. *Windows XP*), sealt alates aga *Microsoft*i *MASM32*, mis on ühildatud 64-bitise arhitektuuriga (mida toetab nt. *Windows7*). Pakett *MASM32* on vabavaraline ja allalaaditav aadressilt
http://www.masm32.com/masmdl.htm
*TTS*i lülitamata (ent realiseeritud) on kolm baitkoodi, *Java*, *.NET* ja *MONO*, tegijateks *Henri Lakk* ja *Einar Pius*. Kevadel 2012 lisandub loodetavasti veel üks väljund, *C*, mis on *Anderei Letošnev*i bakalaureusetöö teemaks.

7.3.1. Ettevalmistustööd

Interpretaatorit realiseerides pidime tegelema mõningate (ainult mõningate, sest *Trigol* on triviaalne keel) semantiliste probleemidega – ent *kompilaator* võib eirata üsnagi paljusid interpretaatori probleeme: sisuldasa need jäävad, ent nende lahendamine on mugav jätta assembler-translaatori hooleks (kui *TTS*i *kompilaatori* objektkood on assembler, siis genereeritud assemblertekst antakse üle assembler-translaatorile ning semantikavigade[1] avastamine jääb juba viimase kompetentsi).

Sellest hoolimata on vaja teha mõningane transleeritava programmi *Analüsaatorilt* saadud väljundandmestruktuuride kompileerimiseelne modifitseerimine. Minimaalselt tuleks teha järgmist.

- Teha kindlaks[2] vajalike töömuutujate arv – töömuutujad on vajalikud avaldiste vahetulemuste säilitamiseks (meie *kompilaatoris* omistatakse neile nimed *dTv*i, kus i=0,…,*n* (*n* = vajalike töömuutujate arv – 1)). Assembleri reeglid nõuavad muutujate deklareerimist andmesektsioonis, enne koodisektsiooni algust. Töömuutujad on lokaliseeritud *avaldiste* koosseisus – iga avaldise mõjupiirkonna lõppedes on "ammendatud" kõik töömuutujad. Seega, iga töömuutuja on kasutatav järgmises operaatoris.
- Luurata[3] puus tingimusoperaatoreid ning teisendada analüüsi puud, hõlbustamaks järgnevat kompileerimist nii, et puu ühekordse kompileerimisaegse läbimisega saaks genereerida kogu objektkoodi.

Tuletagem meelde tipu tegeliku formaadi:

```
/* analyysi puu tipp */

struct top{
      int kood;     /* vahekeelne kood */
      int leks;     /* kui tipp=ident/const, siis selle jrk-nr. */
      int sem;      /* semantikakood */
      int label;    /* kui tipp on märgendatud operaator -
                       märgendi number */
      int truel;          /* kompilaator: go to true */
      int falsel;         /* kompilaator: go to false */
      struct top *up;     /* puuviidad: üles, */
      struct top *right; /* naabrile ja */
      struct top *down;  /* alluvale */
      };
```

Väljade *top→ truel* ja *top→ falsel* algväärtuseks on Ø. Nende otstarbe selgitamiseks toome lihtsa (abstraktse) näite:

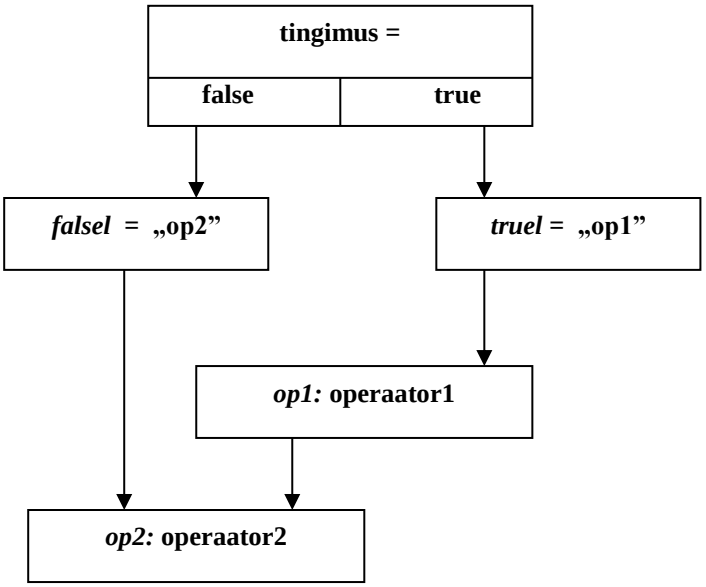
```
IF tingimus THEN operaator1; operaator2; ...
```

Niisiis, kui *tingimus* on täidetud, siis tuleb täita *operaator1* ja seejärel *operaator2*, vastasel korral aga kohe *operaator2*. Situatsiooni illustreerib joonis 7.3.a: seal kujutatud märgendid

[1] Nt. korduvad märgendid, suunamine deklareerimata märgendile jmt.
[2] Algoritm on lihtne: tuleb läbida analüüsi puu ning imiteerida magasini abil kompileerimisaegset tööd, otsides töömuutuja maksimaalset numbrit *n*. Seega, *n+1* ongi vajalike töömuutujate arv.
[3] Luurest kui efektiivsest programmeerimisvõttest vt. [Isotamm 2009], lk. 92 jj.

truel ja *falsel* on kas tegelikud, s.o. programmis defineeritud, või *töömärgendid*, mis genereeritakse *kompilaatori* poolt – sel juhul kasutab *Trigol*-kompilaator nimesid *MEx*i (kus sufiks i = 1, 2,..)[1].

Pragmaatilistel kaalutlustel eristagem kaht varianti: *operaator1* on kas *suunamine* või ei ole; suunata saab ilmutatud kujul esitatud märgendile, muudel juhtudel tuleb kasutada nn. *töömärgendit*, s.o. *kompilaatori* poolt genereeritud märgendit.



Joonis 7.3.a. Tingimuse kompileerimine.

Analüüsi puu eeltöötluse ja (vajadusel) teisendamisega tegeleb meie *Trigol*-kompilaatoris protseduur *det_nrlv*, mille teksti esitamine allpool tundub otstarbekamana kui algoritmi verbaalne või skemaatiline seletamine. Töömuutujate arvu luuramiseks kasutatakse *int*-vektorit (sisuliselt *LIFO*-magasini), *st.* järgmiste võimalike elementide väärtustega:

`st[i]=1,` kui element on muutuja;
`st[i]=2,` kui element on konstant;
`st[i]=3,` kui element on märgend;
`st[i]=4,` kui element on töömuutuja.

Iga (järjekordse) avaldise puu läbimisel kasutatakse muutujat *tm* (väärtused 1, 2, …) ja vajalike töömuutujate arvu (maksimaalne *tm*) säilitatakse muutuja *N* väärtusena.

Tingimuste ja nendega seonduvate märgendite, suunamiste ja puu teisendamistega seotud puu tippude viitade jaoks on protseduuris *det_nrlv* kasutusel muutujad **t, *t1,...,*t4* ning töömärgendite hilisemaks genereerimiseks kasutatakse muutujat *IX*.

[1] *Trigol*-kompilaator kasutab töömuutujate tähistamiseks nime *dTv*i ja töömärgendi tähistamiseks – *MEx*i. Miks nii: mõlemate objektide nimed peavad olema sellised, mille peale programmeerija juhuslikult ei tohiks tulla, ja kuivõrd nende ridade autor sai TÜ meiliserverilt endale esialgseks parooliks *dTvMExi* – mis näib olevat täiesti juhuslik – siis tunduski, et tolle juhusliku sõna erinevad pooled võiksid sobida tööobjektide üsna kindlasti originaalsete nimede prefiksiteks.

Lisaülesandeks on tollel protseduuril luurata, kas programmis kirjeldatud muutujate väärtusi tuleb konsoolilt sisestada ja/või konsoolile väljastada, s.o. kas objektprogrammi päisessse tuleb genereerida sisestamis-/väljastamisoperatsioonide alamprogrammide kirjeldused (seda *TASM*-versiooni jaoks).

Protseduur (funktsioon) *det_nrlv* on järgmine:

```
/* luurab töömuutujate arvu (tmarv), töötleb if-lause puud */
int det_nrlv(struct top *root){
      struct top *t,*t1,*t2,*t3,*t4;
      struct itr *id;  //identifikaatorite tabeli kirje
      int k;
      char st[20];     //luuremagasin; 1=i, 2=c, 3=label, 4=tm
      int op;          //operatsiooni semantika kood
      int tm=0;        //töömuutuja number
      int N=0;  /* maks. töömuutuja number */
      int i=0;  /* magasiniindeks */
      IX=1;     /* töömärgendi nr */
      t=root->down;
      memset(st,'\0',20);   //magasini puhastamine

down:
      if(t->down!=(struct top *)NULL){
            t=t->down;
            goto down;
            }
/* leht => stack  */
leht:
      op=t->sem;
      switch(op){
            case muut:{                                  /* muutuja */
                  st[i]=1;
                  for(k=0;k<itl;k++){
                        if(IT[k]==t->leks){
                              id=IDT[k];
                              goto iok;
                              }
                        }
                  iok:
                  if(id->t!=(struct top *)NULL){
                        st[i]=3;              /* märgend */
                        }
                  break;
                  }
            case konst:{
                  st[i]=2;                    /* konstant */
                  break;
                  }
            default: red("error in the parsing tree");
                   return(-1);
            }
      i++;
naaber:
      if(t->up==(struct top *)NULL){
            t=t->right;
            if(t->down!=(struct top *)NULL) goto down;
            else goto leht;
            }
      t=t->up;
      if(t==root) goto ok;
      op=t->sem;
      if((op>=pisem)&&(op<=vord)){              /* võrdlustehe */
            if(st[i-2]==4) tm--;
            if(st[i-1]==4) tm--;
            i-=2;
```

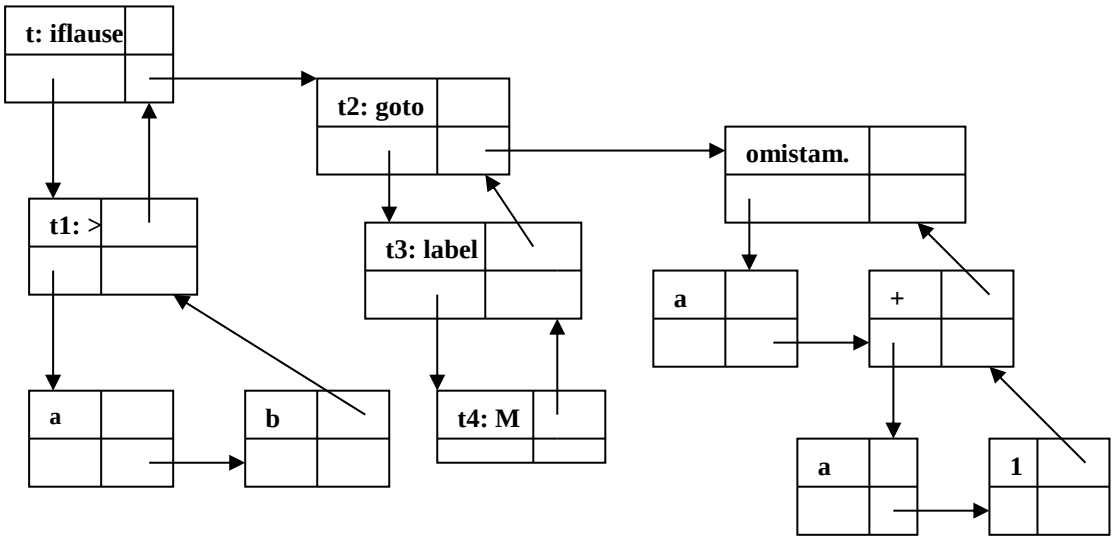
```
            goto naaber;
            }
      if((op>=jag)&&(op<=liit)){            /* aritmeetika */
                  if(st[i-2]==4) tm--;
                  if(st[i-1]==4) tm--;
                  st[i-2]=4;     /* töömuutuja */
                  tm++;
                  if(tm>N) N=tm;
                  i-=1;
                  goto naaber;
                  }
      switch(op){
            case omist:{              /* omistamine */
                  if(st[i-2]==4) tm--;
                  if(st[i-1]==4) tm--;
                  i-=2;
                  break;
                  }
            case suunam:{               /* suunamine */
                  i-=1;
                  break;
                  }

//if-lause töötlus: puu teisendamine või töömärgendite ettevalmistamine

            case kuisiis:{        /* if-lause */
                  t1=t->down;     /* võrdlemine */
                  t2=t->right;    /* IFi naaber */
                  if(t2->sem==suunam){
                        t3=t2->down;
                        t4=t3->down;
                        t1->truel=t4->leks;
                        free(t3);  //kustutatakse tipp t3
                        free(t4);  //kustutatakse tipp t4
                        t->up=t2->up;   //korrigeeritakse viitu t→up ja
                        t->right=t2->right;   //t→right
                        free(t2);   /* eemaldan 'goto' */
                        }
                  else{
                        t3=t2->right;
                        t1->truel=IX;
                        t2->label=IX;
                        IX++;
                        t1->falsel=IX;
                        t3->label=IX;
                        IX++;
                        }
                  break;
                  }
            case tingop: break;

//Rd>0, kui programm peab konsoolilt lugema, ja Wr>0, kui kirjutama.
//identifikaatorite tabeli kirjes id→io=0, kui muutujat ei kasutata

//infovahetuseks konsooliga, id→io=1, kui ta väärtus sisestatakse,
// id→io=2, kui väljastatakse, ja id→io=3, kui nii sisestatakse kui
//ka väljastatakse.

            case lugem:{
                  t1=t->down;
                  for(k=0;k<itl;k++){
                        if(IT[k]==t1->leks){
                              id=IDT[k];
                              goto lok;
                              }
                        }
                  lok:
                  id->io=id->io||1;
```

```
                  Rd=1;
                  i-=1;
                  break;
                  }
            case kirjut:{
                  t1=t->down;
                  for(k=0;k<it1;k++){
                        if(IT[k]==t1->leks){
                              id=IDT[k];
                              goto kok;
                              }
                        }
                  kok:
                  id->io=id->io||2;
                  Wr=1;
                  i-=1;
                  break;
                  }
            }
      goto naaber;
ok:   return(N);
}
```

Usutavasti pole vaja tuua näidet ei töömuutujate ega ka sisend-/väljundoperaatorite luuramisest, küll aga tingimusoperaatorite (*if*-lause) töötlusest. Vaadelgem kaht näidet, esimene neist kasutab *ilmutatud* suunamist:

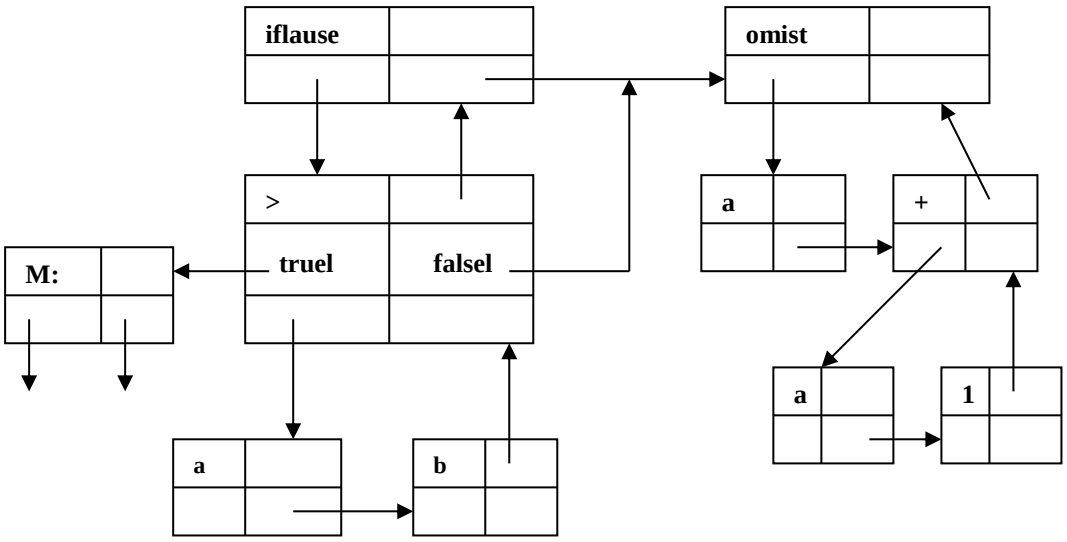
```
IF a>b THEN GOTO M; a:=a+1;
```

Harjumuspärasel kujul on ülaltoodud programmifragmendi puu (lisatud on ülaltoodud protseduuris kasutatud märgendid t_i) toodud joonisel 7.3.b.



Joonis 7.3.b. Puu *goto*-fragment.

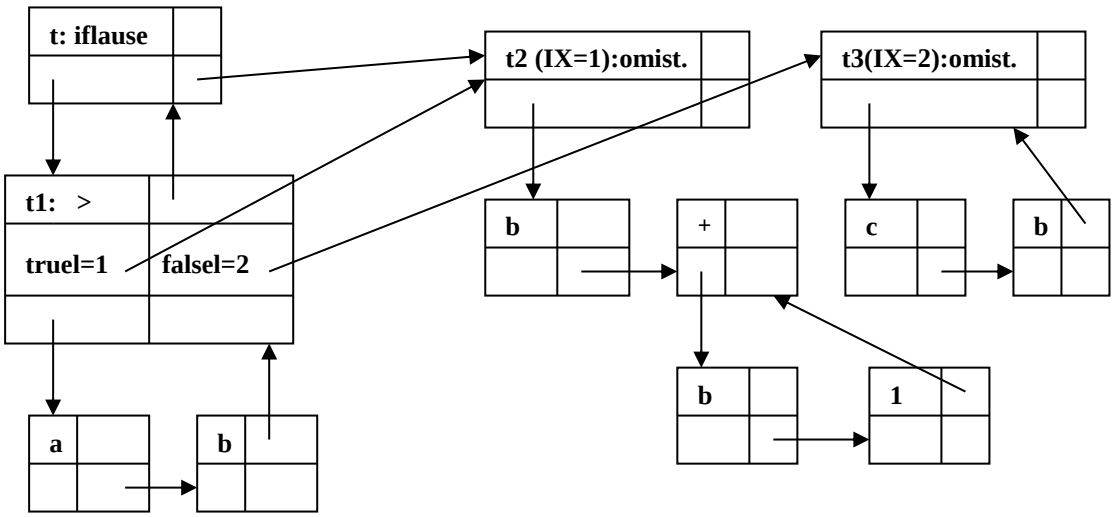
Joonisel 7.3.c on kujutatud ülaltoodud puu teisendatud versioon.



Joonis 7.3.c. Kompileerimiseks kohandatud *if-lause goto*-fragment.

Teine näide on selline, kus tingimus *pole* seotud suunamisega (vt. joonis 7.3.d):

```
IF a>b THEN b:=b+1; c:=b;
```



Joonis 7.3.d. Kompileerimiseks kohendatud *if-lause* suunamiseta fragment.

7.3.2. Näited

Allpool toome näidetena kahe *Trigol*-programmi (*P4* ja *P6*) transleerimislogi asjakohaseid fragmente ning nende *Assembler*-tekstid (objektkeel on 16-bitine assembler).

7.3.2.1. P6.tri

Programm P6.tri. *Kompilaatori* logi algus.

```
# READ n; F:=1; I:=0;
M1: I:=I+1;
IF I>n THEN GOTO M2;
F:=F*I;
GOTO M1
M2: WRITE F#
```

Parsing tree[1]

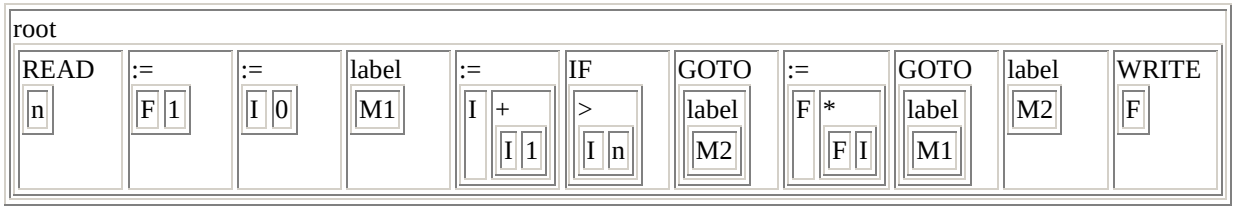


Table of constants

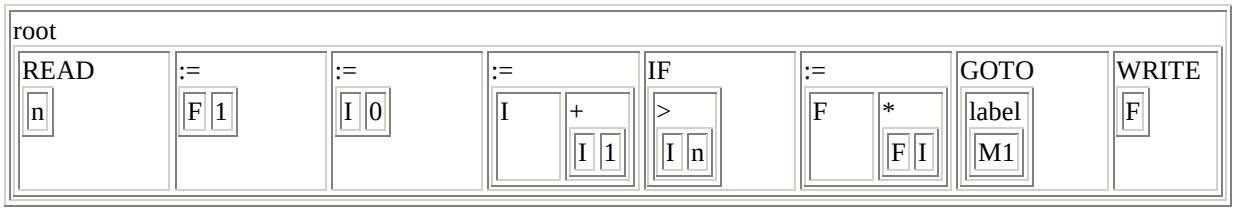
c1=1	c2=0

Table of identifiers

i1=n	i2=F	i3=I	i4=M1	i5=M2

label 'M1' is address of the operator {omistamine} (00481E40)
label 'M2' is address of the operator {kirjutamine} (00481C60)

Modified tree:



Compiler to Assembler started;

Programm P6.tri. Väljundprogramm P6.asm

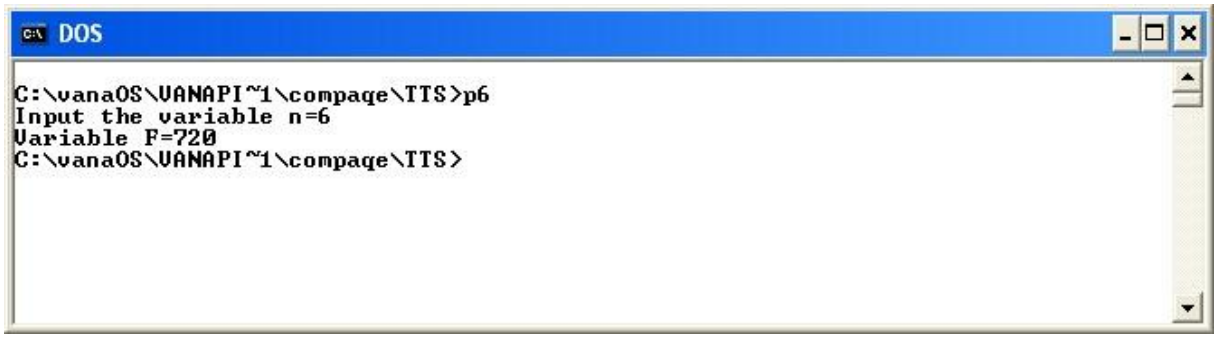
```
#READ n; F:=1; I:=0;
; M1: I:=I+1;
; IF I>n THEN GOTO M2;
; F:=F*I;
; GOTO M1;
; M2: WRITE F#
;
; Program P6.asm
```

[1]Analüüsi puu *HTML*-formaadis esitamise ideem algoritmi ja realisatsiooni autorid on tolleaegsed üliõpilased *Lauri Jesmin* ja *Leho Lehes.*

```
      .MODEL small
      .STACK 100h
      EXTRN readint:PROC
      EXTRN bin2dec:PROC
      .DATA
n     DW    0
F     DW    0
I     DW    0
dTv0  DW    0
Sisse DB    'Input the variable ','$'
Trykk DB    'Variable ','$'
Reavahetus  DB    13,10,'$'
n_S   DB    'n=','$'
F_S   DB    'F=','$'
      .CODE
ProgramStart:
      mov   ax,@data
      mov   ds,ax
      mov   ah,9h
      mov   bx,1
      mov   cx,17
      mov   dx,OFFSET Sisse
      int   21h
      mov   ah,9h
      mov   bx,1
      mov   cx,2
      mov   dx,OFFSET n_S
      int   21h
      call  readint
      mov   n,ax
      mov   ax,1
      mov   F,ax
      mov   ax,0
      mov   I,ax
M1:   mov   ax,I
      add   ax,1
      mov   dTv0,ax
      mov   ax,dTv0
      mov   I,ax
      mov   ax,I
      cmp   ax,n
      jg    M2
      mov   ax,F
      mov   dx,I
      mul   dx
      mov   dTv0,ax
      mov   ax,dTv0
      mov   F,ax
      jmp   M1
M2:   mov   ah,9h
      mov   bx,1
      mov   cx,8
      mov   dx,OFFSET Trykk
      int   21h
      mov   ah,9h
      mov   bx,1
      mov   cx,2
      mov   dx,OFFSET F_S
      int   21h
```

```
      mov   ax,F
      mov   dx,0
      cmp   ax,0
      jg    kirj0
      mov   dx,1
kirj0:
      mov   ch,1
      call  bin2dec
      mov   dx,OFFSET Reavahetus
      mov   ah,9h
      int   21h
      mov   ah,4ch
      int   21h
      END   ProgramStart
```

Genereeritud assemblertekst antakse ette 32-bitisele *Intel*i assemblerile ja linkerile, nood teevad meie *Trigol*-programmiga *P6.tri* semantiliselt ekvivalentse programmi *P6.exe*, mille lahendamist illustreerib joonisel 7.3.e toodud ekraanitõmmis.



Joonis 7.3.e. Programmi *P6.exe* lahendamine.

7.3.2.2. P4.tri

Pöörakem tähelepanu selle programmi tekstile ja mõtelgem, kui ratsionaalne see on. Kui leiame, et mitte eriti, siis ongi ta sobiv sisse juhatama *optimeerimise* teemat.

Programm P4.tri. *Kompilaatori* logi algus.

Program

IF (1=1) THEN F:=(7*(3+(2*5)))
F:=100#

Parsing tree

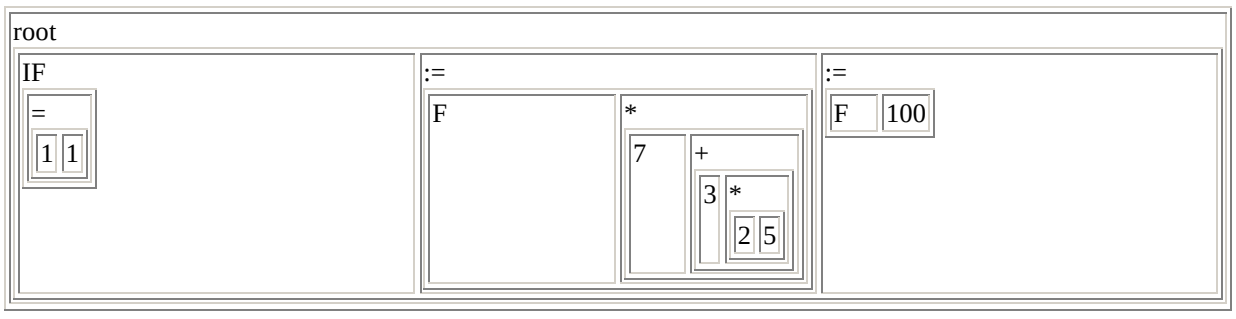


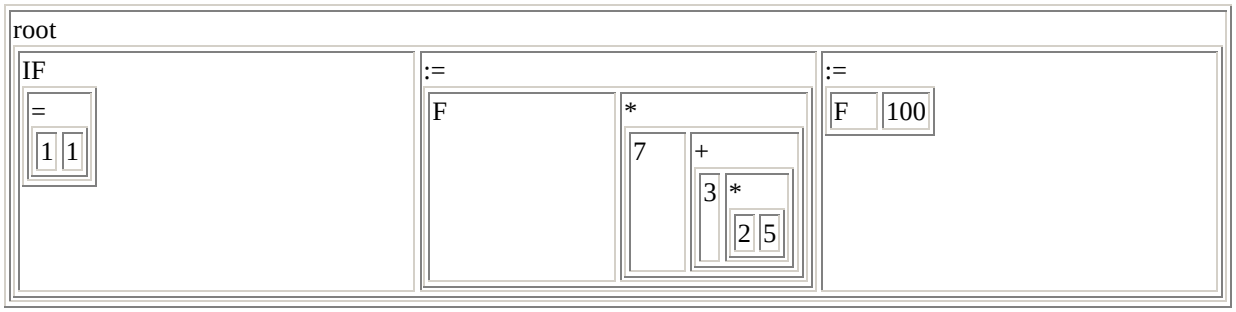
Table of constants

c1=1	c2=7	c3=3	c4=2	c5=5	c6=100

Table of identifiers

i1=F

Modified tree:



Compiler to Assembler started

…

Logi lõpeb järgmiselt:

```
programm P4.asm is compiled

 I'll start compiler from assembler, and linker


tasm P4 >>P4c.htm

Turbo Assembler  Version 3.1  Copyright I 1988, 1992 Borland International

Assembling file:   P4.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  420k


tlink P4+teek >>P4c.htm

Turbo Link  Version 2.0  Copyright I 1987, 1988 Borland International

Compiler ended at  Thu Sep 21 16:08:10 2006
```

Programm P4.tri. Väljundprogramm P4.asm.

```
; # IF 1 = 1 THEN F := 7 * (3 + (2 * 5 ) ) ; F := 100  #
;
; Program P4.asm
      .MODEL small
      .STACK 100h
      .DATA
F     DW    0
dTv0  DW    0
      .CODE
ProgramStart:
      mov   ax,@data
      mov   ds,ax
      mov   ax,1
      cmp   ax,1
      je    MExi1
      jmp   MExi2
MExi1:
      mov   ax,2
      mov   dx,5
      mul   dx
      mov   dTv0,ax
      mov   ax,3
      add   ax,dTv0
      mov   dTv0,ax
      mov   ax,7
      mov   dx,dTv0
      mul   dx
      mov   dTv0,ax
      mov   ax,dTv0
      mov   F,ax
MExi2:
      mov   ax,100
      mov   F,ax
      mov   ah,4ch
      int   21h
      END   ProgramStart
```

7.3.3. Optimeerimine

„Eesti entsüklopeedia ([EE 7], lk.79) annab märksõna *optimaalne* vasteks ’(mingist seisukohast) parim, sobivaim, soodsaim[1]”. Termin pärineb ladina keelest: *optimum* – ’parim’. Selles võtmes pole mõtet rääkida *optimaalsest koodist* (transleerimise väljundist), kuivõrd pole kriteeriumi, mis fikseeriks mingi algoritmiga adekvaatse *optimaalse koodi.* Samas võime käsitleda *optimeerimist* kui parima lahendi suunas kulgevat *protsessi*, mille resultaat on loodetavasti parem kood, ent ärgem unustagem: me ei saa tõestada, et ta on *optimaalne.*

Ajalooliselt oli optimeerimine varajaste masinate puhul ülioluline, kuivõrd mälu oli vähe, protsessorid olid aeglased ja masina tööaeg kallis. Optimeerimiseks tehti reeglina suur arv vahekeelse koodi läbivaatusi. *V. N. Lebedevi* ([Lebedev], lk. 207 jj.) andmetel tegi *Alfa*-trans-

[1] Siit johtub tõik, et sõnal *optimaalne* puuduvad võrded; me ei saa rääkida, et miski lahendus on „optimaalsem“ või „kõige optimaalsem“. Nende ridade autor on oma üliõpilastele püüdnud selgitada, et selletaolisi keelendeid tohivad kasutada ainult ajakirjanikud ja poliitikud, ent mitte muud (liht)inimesed. Täpselt sama lugu on mõistega *ideaalne*. Miski kas on seda või ei, muid võimalusi pole.

laator (*Algol-60* → masinkood, Novosibirsk 1964) kokku 24 läbivaatust, enamik neist just optimeerimiseks.

Optimeerivad translaatorid töötavad reeglina oluliselt aeglasemad nn. kiiretest, ent annavad tunduvalt lühema ja kiirema objektkoodi. Varasematel aegadel realiseeritigi keeled tavaliselt kahes variandis: kiire translaator koos silumisvahenditega ja täitmisaegsete kontrollidega (võimaldamaks silumist) ning optimeeriv variant, mille väljundist olid eemaldatud nii silumisplokk kui ka kontrollid[1].

Tänapäeval on optimeerimine säilitanud oma tähtsuse reaalajas töötavate programmide ja suuremahuliste teadusarvutuste puhul (sh. paralleelprotsessingut ja/või *GRID*-tehnoloogiat kasutades). Tavakasutaja jaoks pole vahet, kas tema käivitatud rakendusprogramm töötab terve sekundi või (kõigest!) pool sekundit. Optimeerida tasub seal, kus võidetakse kiirushinnangu[2] suurusjärk(e) ja mitte testülesande lahendamisaja protsente.

7.3.3.1. Trigol

*Trigol*i optimeeriv kompilaator kasutab teadlikult ainult koodi tasemel[3] optimeerimist ja ainult üht kõrgema taseme võtet; põhjuseks on anda võimalus kursuse "Automaadid, keeled ja translaatorid" kuulajatele kompilaatori täiendamiseks.

Madalama taseme lihtsaim optimeerimisvõte on liigsete infovahetuskäskude vältimine: püütakse eemaldada ülearused *register*→ *mälu*- ja *mälu*→ *register*-käsud. Vaatame näiteks programmi *P6.asm*:

```
M1:   mov   ax,I      ;(I) → ax
      add   ax,1      ;(ax)++
      mov   dTv0,ax   ;dTv0=ax on liigne
      mov   ax,dTv0   ;ax=dTv0 on liigne
      mov   I,ax      ;I=ax
      mov   ax,I      ;ax=I
      cmp   ax,n      ;ax=n?
      jg    M2        ;if > then goto M2
      mov   ax,F      ;ax=F
      mov   dx,I      ;dx=I
      mul   dx        ;F×I
      mov   dTv0,ax   ;dTv0=ax on liigne
      mov   ax,dTv0   ;ax=dTv0 on liigne
      mov   F,ax      ;F=ax
```

Normaalsem kood on järgmine (see on saadud *Trigol*i optimeeriva kompilaatoriga):

```
M1:   mov   ax,I
      add   ax,1
      mov   I,ax
      mov   ax,I
      cmp   ax,n
```

[1] *Mati Tombak* (TTÜ informaatika instituut) tegi 1.12.11 seminariettekande *bootstrapping*ust (vt. ka [Isotamm 2009, lk. 245]), millest meie raamatu jaoks olulised slaidid on toodud *M.T.* loal lisas 13.

[2] Meenutagem loengukursusi „Algoritmid ja andmestruktuurid" ning „Keerukusteooria".

[3] Koodi tase kasutab ainult analüüsi puu võimalusi. Kõrgem tase on üldjuhul süntaksorienteeritud translaatorile kättesaamatu, see püüab optimeerida programmi üldist struktuuri, sh. näiteks aritmeetilise avaldise tehete täitmise järjekorda. *Trigol*i puhul on see vahetegemine tinglik, ka siin kirjeldatud kõrgem tase on tegelikult analüüsi puu andmetele tuginev.

```
      jg    M2
      mov   ax,F
      mov   dx,I
      mul   dx
      mov   F,ax
```

Kõrgema taseme (e. semantikataseme) optimeerimise vahendiks on *Trigol*is konstantavaldiste avastamine[1] ja nende väärtuste arvutamine juba kompileerimise ja mitte lahendamise ajal. Konstant(alam)avaldise all mõeldakse aritmeetilist või loogilist (alam)avaldist, kus binaarse operatsiooni mõlemad operandid on konstandid (ja mitte muutujad). Näiteks programmi *P4.tri* kompileerimisresultaat on esitatud allpool. Mõtlemiskoht lugejale: ons see minimaalne kood või saaks veel midagi kokku hoida?

```
; # IF 1 = 1 THEN F := 7 * (3 + (2 * 5 ) ) ; F := 100  #
;
; Program P4.asm
      .MODEL      small
      .STACK      100h
      .DATA
F     DW    0
dTv0  DW    0
      .CODE
ProgramStart:
      mov   ax,@data
      mov   ds,ax
MExi1:      mov   ax,91
      mov   F,ax
MExi2:      mov   ax,100
      mov   F,ax
      mov   ah,4ch
      int   21h
      END   ProgramStart
```

Usutavasti vajab ülaltoodud tekst kommentaari: töömuutujat *dTv0* ja töömärgendeid *MExi1* ja *MExi2* ei kasutata. Põhjus on lihtne: *Trigol*i optimeeriv kompilaator "näeb" töö ajal ainult magasini kahte (või ühte) tipmist elementi; ta ei tegele globaalse optimeerimisega, vaid ainult lokaalsega.

Toome allpool veel ühe näitena programmi *P3.tri* optimeeriva kompileerimise logi:

Start of TRIGOL Optimizing Compiler for a Program P3.tri at Mon Aug 01 13:56:24 2011

Program

F:=(7*(3+(2*5)))#

Parsing tree

root
:=

[1] Sisuliselt on seegi madal tase, kuivõrd see on tehtav analüüsi puu pealt.

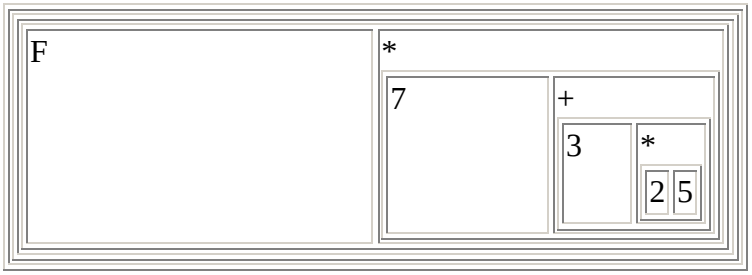


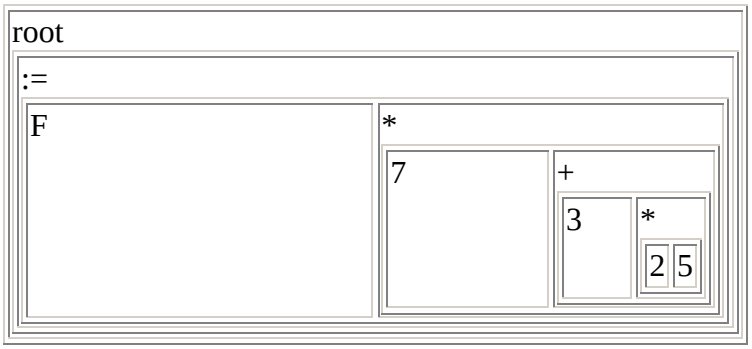
Table of constants

c1=7	c2=3	c3=2	c4=5

Table of identifiers

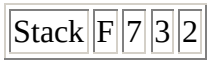
i1=F

Modified tree:

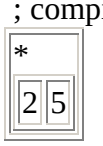


Compiler to Assembler started

```
; gen_header: source text
; #F:=7*(3+(2*5))#
;
; Program P3.asm
        .MODEL      small
        .STACK      100h
; gen_header: Rd=0 Wr=0
; gen_header: # of identifiers=1
        .DATA
F       DW    0
dTv0    DW    0
; gen_header: code segment'll start
        .CODE
ProgramStart:
        mov   ax,@data
        mov   ds,ax
```

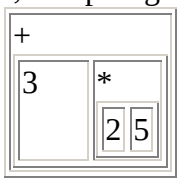


; compiling the operator (2*5)



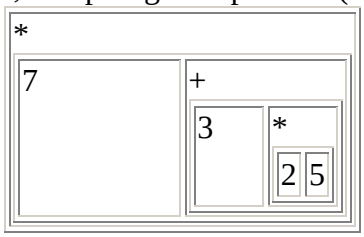
constant expression, I'll optimize..
10 = 2 * 5

; compiling the operator (3+(2*5))



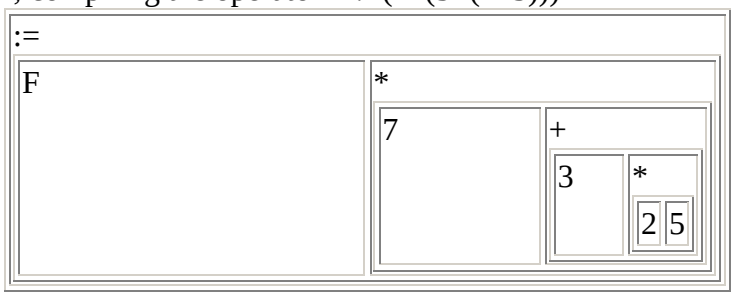
constant expression, I'll optimize..
13 = 3 + 10

; compiling the operator (7*(3+(2*5)))



constant expression, I'll optimize..
91 = 7 * 13

; compiling the operator F:=(7*(3+(2*5)))




```
        mov     ax,91
        mov     F,ax
```

Stack

```
        mov     ah,4ch
        int     21h
        END     ProgramStart
```

programm P3.asm is compiled

I'll start compiler from assembler, and linker

tasm P3 >>P3oc.htm
tlink P3+teek >>P3oc.htm
Compiler ended at Mon Aug 01 13:56:54 2011

7.3.3.2. Muud lihtsamad võimalused

Selles jaotises on toetutud *David Gries*i ([Gries], lk. 420 jj.), *Philip M. Lewis*i, *Daniel J. Rosenkrantz*i ja *Richard E. Stearns*i ([Lewis jt.], lk. 571 jj.), *Alfred V. Aho* ja *Jeffrey D. Ullman*i ([Aho, Ullman], lk. 327 jj.) raamatutele ning N.Liidu selle valdkonna üldtunnustatud eksperdi, *Igor Pottossin*i 1979. aasta loengusarjale TÜ AK ja Eesti Teaduste Akadeemia Küberneetika Instituudi suvekoolis (Elbis). Tuntumad võimalused on järgmised.

Lineaarsed lõigud. Intuitiivselt, lineaarne lõik on operaatorite jada, mis ei sisalda märgendeid ega suunamist. Näiteks:

```
... a:=2; b:=3; c:=a+b; a:=b:=44; ...
```

Ilmselt saame selle lõigu asendada tekstiga

```
... c:=5; a:=b:=44; ...
```

Ja kui *Trigol*-kompilaator töötleks lineaarseid lõike, siis *P4.tri* sisaldaks ainult üht operaatorit:

```
F:=100;
```

Aritmeetilised avaldised. Lihtsaim võte on ajaliselt “kallite” tehete asendamine lihtsamate ja kiirematega. Näiteks,

*y:=a*2;* võib asendada avaldisega *y:=a+a* ning
y:=(a+b)↑2 (kus ↑ tähistab astendamist) avaldistega *temp:=a+b; y:=temp*temp;*

Lewis jt.([Lewis jt.], lk. 574 jj.) toovad näite, kus avaldise transleerimiseks saab kasutada ainult registreid 1 ja 2. Avaldiseks on

```
A*B+(C+D)*(E+F)).
```

*IBM*i-sarnases assembleris on selle kodeerimiseks vaja 9 käsku:

```
L       1,A
M       1,B
L       2,C
A       2,D
ST      1,TEMP
```

```
L     1,E
ADD   1,F
MR    1,2
A     1,TEMP
```

Tehete järjekorra muutmisega saame kokku hoida ühe direktiivi:

```
L     1,C
A     1,D
L     2,E
A     2,F
MR    1,2
L     2,A
M     2,B
AR    1,2
```

Seega, kompileeriti avaldis *(C+D)*(E+F)+(A*B).* Siiski, tehete järjekorra muutmine pole alati ohutu. Näiteks *X–1+1* ei anna ületäitumist ja *X+1–1* annab, kui *X* väärtus on *int*-tüüpi muutuja maksimaalselt võimalik väärtus. *Igor Pottossin* rääkis, et nende *Alfa*-translaatori (*Algol-60* → masinkood, Novosibirsk 1964) võimas optimeerimisplokk tuli füüsikute tungival nõudmisel muuta väljalülitatavaks: kõrgeltkvalifitseeritud kasutajad mängisid keeruliste valemite programmeerimisel tihti vea piiril (kas vahetulemus põhjustab liiga suur või väike olles ületäitumise või ei)[1] ning programmeerisid seetõttu valemeid kaugeltki mitte parimal (kiiruse ja käskude arvu mõttes) kujul. Optimeeriv plokk nullis nende mängu ära ja füüsikud olid leiutanud isegi termini „pessimeeriv režiim“[2] .

Levinud (ja ohutu) võte on avaldistest ühiste alamavaldiste eraldamine, näiteks ([Lewis jt.], lk. 575 jj.) avaldis

```
(A+B)*C+D/(A+B)
```

tuleks kompileerida kui

```
TEMP:=A+B; TEMP*C+D/TEMP;
```

Tsüklite optimeerimine võib evida märgatavat efekti. Peamised võtted on järgmised.

- Tsükli puhastamine: invariantse koodi viimine tsükli ette. Näiteks programmilõik
  FOR I:= 1 STEP 1 until N DO A[I]:=C [I]+E*F;
  on mõttekas asendada koodiga
  TEMP:=E*F; FOR I:= 1 STEP 1 until N DO A[I]:=C [I]+TEMP;
- Tsüklite liitmine, näiteks:
  FOR I:= 1 STEP 1 UNTIL N DO A[I]:=0;
  FOR I:= 1 STEP 1 UNTIL N DO B[I]:=0;
  asemel on parem täita operaator
  FOR I:= 1 STEP 1 UNTIL N DO BEGIN A[I]:=0; B[I]=0; END
- Tsüklite lahutamine – kui meil on kasutada kaks protsessorit, siis eelmise näite tsüklid on mõttekas jaotada: vektori A „nullimine“ esimese ja B – teise protsessori tööks.
- Tsüklioperaatori täitmiskordade vähendamine, näiteks tsükli (N on paarisarv)
  FOR I:= 1 STEP 1 UNTIL N DO A[I]:=0;
  asemel on kiirem variant
  FOR I:= 1 STEP 2 UNTIL N DO BEGIN A[I]:=0; A[I+1]:=0; END

[1] Tuletagem meelde tolleaegsete arvutite arhitektuurist tingitud piiranguid.
[2] Vrd. *optimist* ja *pessimist.*

- “kallite” tehete asendamine “odavamatega”: – laename näite ([Lewis jt.], lk. 578 jj.):
  FOR I:= 1 STEP 1 UNTIL N DO A[I]:=I*5;
  asemel on “odavam” kirjutada
  J:=5; FOR I:= 1 STEP 1 UNTIL N DO BEGIN A[I]:=J; J:=J+5; END
- Indeksmuutujate asendamine lihtmuutujatega, näiteks (eriti, kui masinal pole indeksregistreid), siis programm

```
FOR I:=1 STEP 1 UNTIL N DO
        FOR J:=I+1 STEP 1 UNTIL N–1 DO BEGIN
                IF A[I,J]>A[I,J+1] THEN BEGIN
                        TEMP:=A[I,J];
                        A[I,J]:=A[I,J+1],
                        A[I,J+1]:=TEMP;
                        END
                END
```

töötab kiiremini, kui ta kodeerida nii:

```
FOR I:=1 STEP 1 UNTIL N DO
        BEGIN TEMP:=A[I,J];
        FOR J:=I+1 STEP 1 UNTIL N–1 DO BEGIN
                IF TEMP>A[I,J+1] THEN BEGIN
                        A[I,J]:=A[I,J+1];
                        A[I,J+1]:=TEMP;
                        END
                END
        END
```

Kasutamata objektide elimineerimine. Sellisteks on deklareeritud, ent avaldistes kasutamata konstandid, muutujad, mida kas ei väärtustata üldse või väärtustatakse, kuid ei kasutata; märgendid, millele ei suunata, ning programmilõigud, kuhu ei anta kunagi juhtimist. Mõistagi, neil juhtudel tuleb alati väljastada hoiatusteated: tegu võib olla vajalike asjadega, mis on “kasutud” programmeerija juhuslike vigade tõttu.

Keerulised seigad seonduvad (plokkstruktuuriga keelte) plokkide, eraldi transleeritavate alamprogrammide, rekursiooni ja paralleelprotsessinguga (kui loetleda neist enamtuntuid). *Eero Vainikko* [Vainikko, lk. 12] kirjutab: “Kuigi on olemas teatud reeglid, mida optimaalsel programmeerimisel arvestada, jääb suur osa programmikoodi optimeerimistööst kompilaatori kanda. Ilmne reegel on: mida keerulisem keel, seda raskem on kompilaatoril teha õigeid optimeerimisotsuseid. Näiteks keeles *Fortran77* kirjutatud programmi on tunduvalt lihtsam optimeerida kui keeles *C++*, põhjuseks *Fortran77* staatiline mäluhaldus.

Fortran95 täiendab *Fortran77* standardit moodsate keeleliste vahenditega, arvestades seejuures, et optimeeritavus säiliks niipalju kui võimalik. Objektorienteeritud kontseptsioonist rakendatakse vaid teatud lihtsam osa, mis ei kahjusta programmikoodi optimeeritavust arvutuskiiruse mõttes.”

Kokkuvõtteks

Sissejuhatuses märkisime, et käesolev raamat on mõeldud õppevahendiks kursuse „Automaadid, keeled ja translaatorid“ („AKT“) kuulajaile. Tuleb tõdeda, et osalt objektiivsete, osalt subjektiivsete asjaolude tõttu pole nende kaante vahel kaetud tolle aine kogu materjal, käsitlemata on regulaarsete grammatikate ja keelte ning nendele vastavate automaatide genereerimisega seonduv. Sestap pole ka raamatu pealkiri mitte *AKT*, vaid *TTS − translaatorite tegemise süsteem.*

Selliseid süsteeme on programmeeritud üpris palju (huviline võiks guugeldada märksõnu *Compiler Compiler* või *Parser Generator*), sj. üsna varakult ka Eestis; *Mati Tombak*u juhtimisel tehtud süsteemidest oli ülalpool juttu, neid tegi ta alates *Minsk-32*st üle *Apple*'i ja *Iskra* kuni *PC XT*ni. Samas, Tombak polnud Eestis ainus, kes tegeles süntaksorienteeritud transleerimisega, mainigem tema kolleegidest kas või *Aare Vooglaid*u tollasest TPIst või *Merik Meriste*t meie ülikoolist[1].

Me pole võrrelnud oma TTSi kiirust teiste analoogiliste süsteemidega, ent kuivõrd meie programmi ajalise keerukuse hinnang on lineaarne, siis pole noil võrdlustel (kui neid saakski reaalselt teha) erilist kaalu.

Kõik programmid, mis on nende kaante vahel kirjeldatud, on realiseeritud, sj. kõik programmitekstid on mahalaaditavad (aadressil http://www.cs.ut.ee/~isotamm/) ja laadija jaoks vabalt muudetavad. Ja nood programmid on silutud ja töötavad.

Siinkohal (kokkuvõte ju!) „kriipsutan alla“ *Mati Tombak*u rolli selle raamatu ainese sisus ja vormis. Kui me koos seda raamatut kavandasime, siis minu osaks jäi pelgalt *C*-programmeerimine ja kogu teoreetiline tekst kuulus loomulikult *M.T*-le. Paraku, *Mati* kolis Tallinna. Tema õpilane *Ahti Peder* kasutas meie *TTS*i nii oma magistri- kui ka doktoritöö kirjutamisel ja minu õnneks nõustus ta hakkama käesoleva raamatu toimetajaks.

Ahti Peder, informaatikadoktor ja 2010. a. Eesti odaviskemeister. („Sakala“, 27.07.2010)

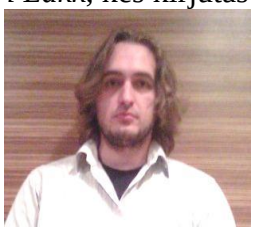
Ent lisaks neile kahele kuulub minu tänu mu (kaasaegsetele) noortele kolleegidele ja need on järgmised.

- *Gunnar Kudrjavets* oli esimene *Trigol*i laiendaja[2] ja ta veenis mind tegema *TTS*i *Windowsi*-liidese. Üsna noorelt sai ta tööle Microsofti peakontorisse ja minu andmetel on ta tänini seal.

[1] Siiski, kui nad ka polnud *Mati* õpilased, olid nad teda aktsepteerivad kolleegid.

[2] Selle õppekeele võimaluste laiendamine on üliõpilastele variant asendamaks eksami teise poole − teooria − sooritamist. Lisas 10 on toodud seni viimase laiendaja *Viktor Karabut*i eeskujuliku töö seletuskiri.

- *Nikita Šipilov*, kes kirjutas regulaaravaldisi interpreteeriva üldise generaatori ja oli kuni viimase ajani selle raamatu kaasautor; ja ehkki ta ennast sellest rollist taandas, on koos temaga kavandatud raamatu struktuur loodetavasti vastuvõetav. Paraku, tema teema pole selles raamatus esitatud.
- *Henri Lakk*, kes kirjutas kompilaatori *Trigol* → *Java baitkood*, sj. raskeim osa oli programmeerida *Java*-koodi päisfaili generaator (mille resultaati aktsepeeriks *Java* interpretaator).



- *Einar Pius*, kes jätkas Henri avatud teed ning võttis realiseerida kompilaatori .*NET*i ning leidis, et *MONO* realiseerimine on suhteliselt triviaalne − kui tehtud on kompilaator dot-netti (.*NET*), ja tegi sellegi translaatori ära [Pius]. Selle raamatu kirjutamise ajal oli *Einar* Edinburghi (Šoti-maa) Ülikooli doktorant, spetsialiseerus kvantarvutustele ja -arvutitele.

- *Jaak Ristioja*, kes lõi õpitarkvara kursuse „Automaadid, keeled ja translaatorid“ omandamise hõlbustamiseks (vt. [Ristioja]).
- Siia nimekirja võiksid kuuluda kõik *Trigol*i arvukad laiendajad ja miks ka mitte kõik kursuse aktiivsed kuulajad ja ka kõik kursuse läbinud. Aga nii saaks see nimekiri ilmselgelt liiga pikk. Ent: aitäh kõigile!

Allpool toodud lisadest on mahukaimad *TTS*i *C*-programmid. Leidsime, et nende äratoomist õigustab seik, et nad esitavad *täpseid* algoritme, sj. kontrollituid. Nende esitamiseks valisime alul variandi, kus elimineerisime kõik logi genereerimisega seonduva, ent selle esitamisest loobusime kahel põhjusel: esiteks, seda pügatud varianti osutus võimatuks napi ajaga testida nii, et võiksime seda kindlustundega publitseerida, ja teiseks − näidates, kuidas genereerida hüperteksti, loodame seda raamatut huvitavamaks ja kasulikumaks teha. Ning võimaldada omavahel kokku panna logi ja programmi: mis kuskil toimub.

Lisad

Lisa 1. Oluliselt mitmene keel

Mati Tombak saatis 13.09.2010 meie raamatu jaoks *oluliselt mitmese* keele $\mathscr{L}$ näite:

$\mathscr{L}$= $\{a^n b^n c^k : n,k>0\} \cup \{ a^n b^k c^k : n,k>0\}$.

Suvaline grammatika, mis genereerib selle keele, on oluliselt mitmene. Näiteks

```
S →AB
S →CD
A →aAb
A →ab
B →Bc
B →c
C →Ca
C →a
D →bDc
D →bc
```

Intuitiivselt on põhjus selles, et keele $\mathscr{L}$ kahe komponendi ühisosa

$\{a^n b^n c^k : n,k>0\} \cap \{ a^n b^k c^k : n,k>0\} = \{ a^n b^n c^n : n>0\}$

ei ole kontekstivaba keel. Selle tõestamine on aga üsna keeruline, selleks tuleb enne tõestada *Ogden*i lemma. Täpsemalt on *Aho*, *Ullman*i 1. köites p. 2.6 [Aho, Ullman].

Märgime omalt poolt, et a^n tähistab sümboli *a n*-kordset konkatenatsiooni, näiteks a^3b^2 on `aaabb`. Võrdle $2^3=8 \equiv 2\times2\times2$.

Lisa 2. Valik grammatikaid

See lisa on kopeeritud *TTS*-süsteemi *Help*-lingiga kättesaadavast manuaalist.

Grammatikaid on süsteemis kolme liiki: õppegrammatikad prefiksiga "G", triviaalse interpreteeritava ja kompileeritava keele "Tri" ning mitte-tavaprogrammeerimiskeele grammatika "form8". Allpool näitame tabelis, millised on nende grammatikate omadused *Konstruktori* vaatevinklist.

Grammatika	eelnevusgrammatika?	pööratav?	analüüsitavus	kommentaar
G1	jah	jah	on	
G2	jah	jah	on	
G3	ei	jah	on	
G4	ei	ei	pole	
G5	jah	jah	on	
G6	jah	ei	pole	
G7	jah	ei	BRC(1|1)	
G8	jah	ei	BRC(1,1)	
G9	ei	ei	BRC(1,1)	
G10	ei	ei	BRC(1,1)	
G11	jah	ei	pole	
G12	jah	ei	BRC(1,1)	
G13	ei	ei	pole	
G15	ei	jah	on	
G17	jah	ei	pole	
G26	jah	ei	BRC(1,1)	
G41	ei	ei	BRC(1|1)	G4 juurdetoodud kontekstiga
G100	jah	ei	pole	
G101	jah	ei	BRC(1|1)	
Ga33	jah	ei	BRC(1|1)	LL(k)-grammatika
Form8	jah	ei	BRC(1,1)	autorid Mati Tombak ja Ahti Peder
Tri	jah	ei	BRC(1|1)	

G1.GRM

```
`S'->#`A'#
`A'->`B'`C'
`B'->a
`B'->`B'a
`C'->b
```

G2.GRM

```
`T'->#`S'#
`S'->`U'`V'
`U'->a`U'b
`U'->a b
`V'->c`V'
`V'->c
```

G3.GRM

```
`T'->#`S'#
`S'->`U'`V'
`U'->a`U'b
`U'->a b
`V'->`V'c
`V'->c
```

G4.GRM

```
`T'->#`S'#
`S'->a`A'
`S'->b`B'
`A'->0`A'1
`A'->0 1
`B'->0`B'1 1
`B'->0 1 1
```

G41.GRM

```
`T'->#`S'#
`S'->a`A'
`S'->b`B'
`A'->`C'`A'1
`A'->0 1
`B'->`D'`B'1 1
`B'->0 1 1
`C'->0
`D'->0
```

G5.GRM

```
`T'->#`S'#
`S'->`U'`V'
`U'->a b
`V'->c`V'
`V'->c
```

G6.GRM

```
`T'->#`S'#
`S'->a`A'
`S'->b`B'
`A'->0`A'1
`A'->0 1
`B'->`C'1
`B'->`D'1
`C'->0`B'1
`D'->0 1
```

G7.GRM

```
`T'->#`S'#
`S'->a`A'
`S'->b`B'
`A'->0`A'1
`A'->`C'1
`B'->`C'1
`C'->1
```

G8.GRM

```
`T'->#`S'#
`S'->a`A'a
`S'->b`A'b
`S'->a`B'b
`S'->b`B'a
`A'->1
`B'->1
```

G9.GRM

```
`T'->#`S'#
`S'->a`A'a
`S'->b`A'b
`S'->a`B'b
`S'->b`B'a
`A'->1
`B'->1
`A'->`A'c
`B'->d`B'
```

G10.GRM

```
`T'->#`S'#
`S'->a`A'a
`S'->b`A'b
`S'->a`B'b
`S'->b`B'a
`A'->1
`B'->1
`A'->`B'c
```

G11.GRM

```
`T'->#`S'#
`S'->`B'`C'
`A'->`B'`D'
`A'->`A'a
`A'->a c
`B'->a
`C'->`A'b
`D'->a
```

G12.GRM

```
`T'->#`S'#
`S'->a`A'a
`S'->b`A'b
`S'->a`B'b
```

```
`S'->b`B'a
`A'->1
`B'->1
`A'->c`B'
```

G13.GRM

```
`T'->#`S'#
`S'->a`A'a
`S'->b`A'b
`S'->a`B'b
`S'->b`B'a
`A'->1
`B'->1
`A'->`B'c
`B'->d`A'
```

G15.GRM

```
`S'->#`N'#
`N'->`A'
`N'->`A'`N'
`A'->x
`A'->a
`A'->b
`A'->c
`A'->d
`A'->e
`A'->f
```

G17.GRM

```
`S'-># a `A' a #
`S'-># b `A' b #
`S'-># a `B' b #
`S'-># b `B' a #
`A'->1
`B'->1
```

G26.GRM

```
`T'-># `S' #
`S'->a `A' a
`S'->b `A' b
`S'->a `B' b
`S'->b `B' a
`A'->1
`B'->1
`A'->c `B' c
```

GA33.GRM

```
`X1'-># a `X2' #
`X2'->a`X2'`X3'
`X2'->b
`X3'->b
`X1'-># a`X4' #
`X4'->a`X4'`X5'
`X4'->c
`X5'->c
```

TRI.GRM

```
`programm'      ->      `programm12'#
`programm12'    ->      #`operaatorid'
`operaatorid'   ->      `operaator'
                ->      `operaatorid13';`operaatorid'
`operaatorid13'->       `operaator'
`operaator'     ->      `label':`operaator'
                ->      `omistamine'
                ->      `iflause'
                ->      `suunamine'
                ->      `lugemine'
                ->      `kirjutamine'
`label'         ->      #i#
`omistamine'    ->      `muutuja':=`omistamine1'
                ->      `muutuja':=`loogilav'
`omistamine1'   ->      `aritmav'
`muutuja'       ->      #i#
`iflause'       ->      `tingimus'`operaator'
`suunamine'     ->      GOTO`label'
`aritmav'       ->      `yksliige'
                ->      `aritmav'+`aritmav2'
                ->      `aritmav'-`aritmav3'
`aritmav2'      ->      `yksliige'
`aritmav3'      ->      `yksliige'
`yksliige'      ->      `tegur'
                ->      `yksliige'*`yksliige4'
                ->      `yksliige'/`tegur'
`yksliige4'     ->      `tegur'
`tegur'         ->      #i#
                ->      #c#
                ->      (`tegur5'
`tegur5'        ->      `aritmav')
`loogilav'      ->      `aritmav'<`loogilav6'
                ->      `aritmav'>`loogilav7'
                ->      `aritmav'<=`loogilav8'
                ->      `aritmav'>=`loogilav9'
                ->      `aritmav'/=`loogilav10'
                ->      `aritmav'=`loogilav11'
`loogilav6'     ->      `aritmav'
`loogilav7'     ->      `aritmav'
`loogilav8'     ->      `aritmav'
`loogilav9'     ->      `aritmav'
`loogilav10'    ->      `aritmav'
`loogilav11'    ->      `aritmav'
`tingimus'      ->      IF`loogilav'THEN
`lugemine'      ->      READ #i#
`kirjutamine'   ->      WRITE #i#
```

FORM8.GRM (vt. ka Lisa 12)

```
`program'        -> #`family'#
`family'          -> `metaformula'
`metaformula'     -> `input' `bigformula'
                  -> `bigformula'
`input'           -> INPUT (graph `graphnames' )
                  -> INPUT (var `varnames' )
                  -> INPUT (var `varnames' ; graph `graphnames' )
`graphnames'      -> `graphnames1'
`graphnames1'     -> `graphnames1' `grkoma' `graphname'
```

```
                -> `graphname'
`graphname'     -> #i#
`varnames'      -> `varnames1'
`varnames1'     -> `varnames1' `varkoma' `variable'
                -> `variable'
`variable'      -> #i#
`bigformula'    -> \bigvee_{ `limits' `lbra' `bigformula'
                -> \bigwedge_{ `limits' `lbra' `bigformula'
                -> \bigoplus_{ `limits' `lbra' `bigformula'
                -> odd `prefpar' `bigformula1' :`limits' )
                -> even `prefpar' `bigformula1' :`limits' )
                -> exactlyone `prefpar' `bigformula1' :`limits' )
                -> atmostone `prefpar' `bigformula1' :`limits' )
                -> atleastone `prefpar' `bigformula1' :`limits' )
                -> none `prefpar' `bigformula1' :`limits' )
                -> eq `prefpar' `bigformula1' :`limits' )
                -> `atmost' `prefpar' `bigformula1' :`limits' )
                -> `atleast' `prefpar' `bigformula1' :`limits' )
                -> `level' `prefpar' `bigformula1' :`limits' )
                -> `formula'
`bigformula1'   -> `bigformula'
`atmost'        -> atmost_{ `rate' `rbra'
`atleast'       -> atleast_{ `rate' `rbra'
`level'         -> level_{ `rate' `rbra'
`rate'          -> `expression'
`limits'        -> `limit' ;`constraints1'
                -> `limit'
`limit'         -> `metavariables' \in`set'
                -> `metavariables' \in V(`graphname1' )
                -> `metavariables' \in E(`graphname1' )
                -> `metavariable' \leftarrow `arc' \in E(`graphname1' )
                -> `arc' \in E(`graphname1' )
                -> `limexpression' `lessrel' `metavariables' `lessrel'
`expression'
`limexpression'  -> `expression'
`graphname1'    -> `graphname'
`arc'           -> `graphbra' `node' `nodekoma' `node' \}
`metavariables'  -> `metavariables1'
`metavariables1' -> `metavariables1' `metakoma' `metavariable'
                -> `metavariable'
`metavariable'  -> #i#
`set'           -> B^{ `length' }
`length'        -> `expression'
`node'          -> `metavariable'
`lessrel'       -> <
                -> \leq
                -> \prec
                -> \prec^+
                -> \prec^*
`constraints1'  -> `constraints'
`constraints'   -> `constraints' `constrkoma' `constraint'
                -> `constraint'
`constraint'    -> `conexpression' `relation' `expression'
`conexpression'  -> `expression'
`relation'      -> <
                -> \leq
                -> \prec
                -> \prec^+
                -> \prec^*
                -> =
                -> \ne
```

```
                -> '
                -> \geq
`expression'    -> `expression1'
`expression1'   -> `expression1' +`term'
                -> `expression1' -`term'
                -> `term'
`term'          -> `factor' *`term'
                -> `factor' /`term'
                -> `factor'
`factor'        -> `factor1' }
                -> `primary'
`factor1'       -> `factor' ^{ `primary'
`primary'       -> #c#
                -> `metavariable'
                -> |V( `graphname' )|
                -> |E( `graphname' )|
                -> | `metavariable' |
                -> min `mmbra' `pexpression' `maxminkoma' `expression' \}
                -> max `mmbra' `pexpression' `maxminkoma' `expression' \}
                -> `aritmpar' `pexpression' )
`pexpression'   -> `expression'
`formula'       -> `leftpart' \sim `formula'
                -> `leftpart' \oplus `formula'
                -> `leftpart' \to `formula'
                -> `leftpart'
`leftpart'      -> `leftpart' \vee `clause'
                -> `clause'
`clause'        -> `logterm' \& `clause'
                -> `logterm'
`logterm'       -> \neg `literal'
                -> `literal'
`literal'       -> `logvar'
                -> `logvar' _{ `indexes' }
                -> `prefixformula'
                -> `logicpar' `bigformula1' )
`logvar'        -> #i#
`indexes'       -> `indexes1'
`indexes1'      -> `index' `indexkoma' `indexes1'
                -> `index'
`index'         -> `expression'
`prefixformula' -> odd `prefpar' `formlist' )
                -> even `prefpar' `formlist' )
                ->  exactlyone `prefpar' `formlist' )
                ->  atmostone `prefpar' `formlist' )
                ->  atleastone `prefpar' `formlist' )
                ->  none `prefpar' `formlist' )
                ->  eq `prefpar' `formlist' )
                -> `atmost' `prefpar' `formlist' )
                -> `atleast' `prefpar' `formlist' )
                -> `level' `prefpar' `formlist' )
`formlist'      -> `formlist1'
`formlist1'     -> `formlist1' `formkoma'  `bigformula'
                -> `bigformula'
`grkoma'        -> ,
`varkoma'       -> ,
`metakoma'      -> ,
`nodekoma'      -> ,
`constrkoma'    -> ,
`maxminkoma'    -> ,
`indexkoma'     -> ,
`formkoma'      -> ,
```

```
`graphbra'        -> \{
`mmbra'           -> \{
`logicpar'        -> (
`aritmpar'        -> (
`prefpar'         -> (
`rbra'            -> }
`lbra'            -> }
```

Lisa 3. Trigoli semantikafail

```
4=1    $ #i#
11=2   $ #c#
p32=3  $ <
p33=4  $ >
p34=5  $ <=
p35=6  $ >=
p36=7  $ /=
p37=8  $ =
p13=10 $ omistamine->muutuja:=omistamine1
p26=11 $ yksliige->yksliige/tegur
p25=12 $ yksliige->yksliige*yksliige4
p21=13 $ aritmav->aritmav-aritmav3
p20=14 $ aritmav->aritmav+aritmav2
p12=15 $ label->#i#
p18=16 $ suunamine->GOTO label
p44=18 $ tingimus->IF loogilav THEN
$  p17=19  iflause->tingimus operaator
p45=20 $ lugemine->READ #i#
p46=21 $ kirjutamine->WRITE #i#
```

Lisa 4. TTSi päisfail

Siin on defineeritud *#define-makroga* konstant *HTL*[1]. Seda kasutatakse paljude (sisuliselt dünaamiliste) andmestruktuuride defineerimisel, seal, kus peakski dünaamikat kasutama. Esimeses lähenduses kasutasime seda konstanti ajutise parameetrina, ent süsteemi arendades selgus, et selle konstandi kasutamine teeb paljud asjad oluliselt lihtsamaks, sj. kui mingi keele jaoks jääb tollest konstandist väheks, on seda globaalselt triviaalne muuta. Milles on asi: kontekstivabade eelnevusgrammatikate klassi kuuluvate keelte konstrueerimisel pole ei produktsioonide arv ega mitteterminaalse tähestiku pikkus piiratud, ning oluliselt mugavam oli noid andmestruktuure programmeerida staatilistena, nii et võimalike indeksite ülempiir on fikseeritud ja vajalikust piisavalt kaugel. See *HTL* on kasutusel paljude andmestruktuuride mälueraldamistel ning nonde (pool)staatiliste struktuuride mahtude muutmiseks piisab vaid tolle makromäärangu *HTL*i muutmisest.

```
//two32.h  :: 32-bitise  TTSi  päisfail
#define HTL 256    //siin see HTL ongi defineeritud.
#define sk '\xBA'  // "||" kood
#define ap '\xFA'   /* xFA on relatsiooni tryki "punkt" */
#define MAX_FAILINIME_PIKKUS 256 //tee juurkataloogist failini

  FILE *of;        /* w_bin & r_bin */
  FILE *rdf;       /* real data structures */
  int X;           /* 0 - Gx.grm,  1 - tri.grm  */
  char *Nimi;
  char rida[44];
  char word[20];   // terminaalse või mitteterminaalse   tähestiku  sümbol
  int def[20];     /* produktsiooni parem pool vahekeeles */
  int dl;                  /* "def" pikkus */
  char sat[44];         /* ={"trigol.txt"};  */
  char *fname;     //grammatika teksti nimi (võib koos teega)
  char RD[256];    /* Filename: Real Data Structures */
  char Rn[256];
  FILE *rules;     /* grammatikafaili handle */
  char *Buf;       /* sisendpuhver */
  int PF_length;   /* sisestatud teksti pikkus (Milam) */
  int P_length;    /* sisestatud teksti pikkus */
  char *GBuf;      /* sisendpuhver: grammar */
  int Glen;        /* sisestatud teksti pikkus: grammar */
  int Pnr;         /* produktsiooni jrk-nr */
  int olek;            /* automaadi jaoks */
  char *P;           /* sisendpuhvri järg */
  char T[2*HTL][20];  /* tähestiku V sümbolite nimed */
  int left;
  int right;
  int V;
  int semflag;
  int *semantika;     /* semantikavektor */
  char *p_M;          /* puutrüki indeksite vektor */
  int p_ML;            /* indeksite vektori pikkus */
  char *PM;          /* eelnevusmaatriks: output */
  char Pm[HTL][HTL];  /* konstruktori eelnevusmaatriks */
  char Lm[HTL][HTL];  /* leftmost-maatriks */
  char Rm[HTL][HTL];  /* rightmost-maatriks */
  char *LC;          /* vasak sõltumatu kontekst */
  char *RC;          /* parem sõltumatu kontekst */
```

[1] Esialgne tähendus oli *HashTableLength* – 'paisktabeli pikkus'.

```
  int Index;          // eelnevusteisenduste uute produktsioonide sufiks
  int dep;            /* dep>0; analüsaator kasutab BRC(1,1)-konteksti */
  int dc;  //dc=1, kui sõltuvat konteksti ei saa arvutada või see ei eristu
  int si;                  /* stack index */
  char *DepV;     /* kui P[i]-l tegelik sõltuv kontekst: DepV[i]=1 */

/* Produktsiooni parem pool */
struct R{
      int P;               /* produktsiooni jrk-nr */
      int sem;           /* semantikakood */
      int n;               /* NT definitsiooni pikkus */
      int *d;             /* NT definitsioon */
      struct R *alt; /* NT alternatiivne definitsioon */
      };

/* Tähestiku V sümboli (NT või T) kirje */
struct D{
      int tunnus;          /* 0: NT, 1: T */
      int code;             /* vahekeelne kood */
      int L;                  /* NT või id/const nime pikkus  */
      int loc;                /* ident | const : nr */
      int icx;                /* itl või ktl - IT või KT index */
      struct D *left;    /* kahendpuu viidad (võti on s) */
      struct D *right;
      struct R *def;    /* NT: vahekeelne definitsioon */
      };

struct parm{
      int nr;           //tähestiku V pikkus (V=V_T∪V_N)
      int tnr;         // tähestiku V_T pikkus
      int BRC;    //0: G on pööratav
      int Pnr;      //Produktsioonide arv |P|
      int dep;     //1: sõltuv kontekst
      int itl;       //identifikaatorite arv (parser)
      int ktl;      //konstantide arv (parser)
};

/* Produktsioonide paremate poolte järgi paisatud tabeli kirje */
struct h{
      int P;          /* produktsiooni jrk-nr */
      int n;          /* definitsiooni pikkus  */
      int NT;       /* defineeritav nonterminal  */
      int *d;       / * definitsioon */
      int nc;                /* ´=1 - sõltuv kontekst */
      char *c;             /* reservis */
      struct h *same; /* sama parem pool */
      struct h *col;    /* põrkeviit */
      };

//Andmeviidad V-puu ehitamiseks
  struct D *DT;      /* first   */
  struct D *DC;      /* current */
  struct D *DL;      /* last    */
  struct D *DN;      /* new     */

  int nr;     /* tähestiku V jooksev pikkus */
  int tnr;    /* terminaalse tähestiku jooksev pikkus */
  int ttl;    /* tähestiku V_T pikkus */
  int NR;     // eelnevusteisenduste 'uute produktsioonide' jooksev jrk-nr
  int Sn;     /* 'nr' enne eelnevusteisendusi */
  int BRC;      /* □ühesuguste paremate pooltega produktsioonide arv */
```

```
  int context;  /* 0 kui G on pööratav, 1 kui pole */
  struct parm *PARM;      /* parameetrid */
  struct h *hc;           /* BRC-analüüsil detekteeritud kirje viit */
  struct h *HT[HTL];      /* hashtabel */
  struct h *prod[HTL];
  char ***DEP;    /* sõltuv kontekst */
  FILE *Logi;
  char *L_name;
/* ------------------------------PARSER32-------------------------*/

#define idtl 50  //identifikaatorite ja konstantide tabelite pikkused
#define ctl 50

/* identifikaatorite tabeli kirje */
struct itr{
      int nr;             /* jrk-nr tähestikus V */
      int loc;           //"reservis"
      int value;       /* interpretaator: muutuja väärtus */
      struct top *t;  /* kui id=märgend: viit operaatorile */
      int io;            /* 0, kui pole, 1: read, 2: write, 3: r&w */
      };

/* konstantide tabeli kirje */
struct ctr{
      int nr;         /* jrk-nr tähestikus V */
      int loc;       //"reservis"
      int value;   /* konstandi väärtus */
      };

/* analüüsi puu tipp */
struct top{
      int kood;          /* vahekeelne kood */
      int leks;          /* kui tipp=ident/const, siis selle jrk-nr. */
      int sem;           /* semantikakood */
      int label;         // kui tipp on märgendatud operaator – märgendinr
      int truel;         /* kompilaator: go to true */
      int falsel;        /* kompilaator: go to false */
      struct top *up;    /* puuviidad: üles, */
      struct top *right; /* naabrile ja */
      struct top *down;  /* alluvale */
      };

  int KOOD;
  char *PBuf=NULL;     /* sisendpuhver: programm */
  int Plen=0;                   /* sisestatud teksti pikkus: programm */
  char *gr_name;
  char *pr_name;
  char *Pr_name;          /* ilma laiendita */
  char TT[HTL][20];    /* pikkuse järgi järjestatud eraldajad VT-st */
  int vkl;                       /* VK pikkus */
  int *VK=NULL;       /* vahekeelne programm pikkusega vkl*/
  int VEAD=0;            /* programmi vigade loendaja */
  int VKL=0;               /* vahekeelse programmi symbolite arv */
  int bok;      /* tell_about( ): 0 - ok, 1 - paha */
  int k_k;      /* konstandi vahekeelne kood */
  int m_k;    /* muutuja (identifikaatori) vahekeelne kood */
  int itl;       /* identifikaatorite tabeli index/pikkus */
  int ktl;      /* konstantide tabeli indeks/pikkus */
  int IT[50];     /* identifikaatorite tabel */
  int KT[50];    /* konstantide tabel */
  struct ctr *CT[ctl];
```

```
  struct top *t;   /* top of the parsing tree */
  struct itr *IDT[idtl];  //identifikaatorite tabel
  int logi=0;                 /* kui 1, siis teeb kettale  xxx.log  */
  struct h *t_;
  int Stack[200];
  char rela[200];
  int lausevorm[20];
  struct top *puu[200];
  struct top *p_=NULL;     /*analüüsi puu tipp*/
  struct top *cp=NULL;
  int rel,I,J;
  int n_,l_,s_,N_,x_;
  int sem,marker;
  int dummy;
  int flag;
  int Rd;
  int Wr;

void ExiT(char *t);
int constr(void);            /* constr.c */
void ctree(void);            /* constr.c */
void print_h(struct h *t);              /* hash.c */
int hfunc(int keylen,int *key);        /* hash.c */
struct h *ReadHash(int n,int *key);    /* hash.c */
int HashRule(struct D *t,struct R *r);  /* hash.c */
void HashNT(struct D *t);               /* hash.c */
void HashRules(struct D *t);            /* hash.c */
void print_L(struct h *CP);             /* hash.c */
void print_HT(void);                    /* hash.c */
void brc(int NT);                              /* indep.c */
void print_indepC(void);                       /* indep.c */
void print_cxt(char **L);                      /* dep.c */
void print_all_cxt(void);                      /* dep.c */
void print_all_cxtp(void);                     /* dep.c */
int g1(char **L,int X,int D);                  /* dep.c */
int g2(char **L,struct h *t,int X);            /* dep.c */
int g3(char **L,struct h *t,int D);            /* dep.c */
int g4(char **L,struct h *t);                  /* dep.c */
int search_springs(struct h *t,int A);         /* dep.c */
int test_dep_con(struct h *x,struct h *y);     /* dep.c */
int brctest(struct h *t);                      /* indep.c */
int make_BRC(void);                            /* indep.c */
void pT(int v);                                 /* dict.c */
struct D *newD(void);                          /* dict.c */
struct R *newR(void);                          /* dict.c */
void free_R(struct R *r);                      /* dict.c */
void free_tree(struct D *t);                   /* dict.c */
void printR(struct R *r);                      /* dict.c */
int print_rule(struct D *t,struct R *r);       /* dict.c */
int put_semant(void);                          /* sem.c */
struct D *search(int x);                       /* dict.c */
struct D *getV(char *key);                     /* dict.c */
void viga(int i);                              /* dict.c */
int p_viga(int i);                             /* scanner.c */
int print_rules(struct D *t);                  /* dict.c */
struct D *leftside(int l);                     /* dict.c */
void nonterm(int l);                           /* dict.c */
void term(int l,int flag);                     /* dict.c */
void makerul(void);                            /* dict.c */
int Terms(void);                               /* dict.c */
int Rules(void);                               /* dict.c */
```

```
void first_LR(struct D *t);                     /* lrm.c */
void sec_LR(int lrf,int nt);                    /* lrm.c */
void make_LRB(void);                            /* lrm.c */
void print_def(struct R *r);                    /* dict.c */
int relac(struct D *t);                         /* lrm.c */
int make_PM(void);                              /* lrm.c */
void print_rela(void);                          /* lrm.c */
void s_to_right(int lim, struct D *t,struct R *d);  /* conf.c */
void P2(int x,int y);                           /* conf.c */
void s_to_left(int lim, struct D *t,struct R *d);   /* conf.c */
void P1(int x,int y);                           /* conf.c */
void tell_conf(int p,int i,int j,int c);        /* conf.c */
int conflicts(void);                            /* conf.c */
void print_LRM(char M[HTL][HTL]);               /* lrm.c */
void print_PM(void);       /* lrm.c */
void fprint_PM(void);      /* lrm.c */
int w_tabs(void);                       /* w_bin.c */

/* ----------------------PARSER32--------------------------------*/

char *jarray(char *pealkiri);
void ExIT(void);
char *gM(int p);                  /* w_bin.c */
struct top *r_ptree(void);        /* r_bin.c */
/* void set_up(struct top *root);     r_bin.c */
struct top *analyzer(void);       /* parser.c */
void print_VK(int k);      /* scanner.c */
int Ident(int k);          /* scanner.c */
int Const(int k);          /* scanner.c */
int scanner(void);         /* scanner.c */
int reduce(struct h *t,int x, int y);                       /* parser.c
*/
int p_stack(char *name,int *v,char *rel,int a,int n,int k);  /* parser.c */
struct top *newtop(int x,int i,int sem);                    /* parser.c
*/
void red_tree(struct top *puu[],struct top *p,int s,int n);   /* parser.c
*/
int pp2html(struct top *p);                   /* parser.c */
void sentential_form(void);                   /* parser.c */
int correct(void);                            /* parser.c */
struct top *parser(void);                     /* parser.c */
struct ctr *make_ctr(void);                   /* parser.c */
int make_CT(void);                            /* parser.c */
struct itr *make_itr(void);                   /* parser.c */
void free_CT(void);                           /* parser.c */
int make_IDT(void);                           /* parser.c */
void free_IDT(void);                          /* parser.c */
void p_itr(struct itr *id);                   /* parser.c */
void print_variables(void);                   /* parser.c */
void p_ctr(struct ctr *id);                   /* parser.c */
struct item *make_item(void);          /* interp.c */
void freestack(int i,int k);           /* interp.c */
void prist(int i,int ic);              /* interp.c */
int get_x(int i,int k);                /* interp.c */
void write_x(int x,int i);             /* interp.c */
void aritm(int i,int r);               /* interp.c */
void loogik(int i,int r);              /* interp.c */
int print_Op(struct top *t);           /* interp.c */
void p_prog(struct top *root);         /* interp.c */
void trigol_Int(struct top *root);     /* interp.c */
void prop(int s);                  /* parser.c */
```

```
void tprop(int s);                  /* parser.c */
FILE *opr(void);                      /* r_bin.c */
struct D *r_V(void);                  /* r_bin.c */
struct h *grec(void);                 /* r_bin.c */
struct h *r_hr(void);                 /* r_bin.c */
void r_HT(void);                      /* r_bin.c */
int r_tabs(void);                     /* r_bin.c */
void ctree(void);
void print_TREEK(void);
void print_data(int x,int y);
void print_mx(char *M,int x);
void print_sem(void);
void p_top(char *t,struct top *p);
void pf(const char *format,...);
void print_cxt(char **L);                        /* dep.c */
int itr(void);
```

Lisa 5. TTSi juhtprogramm

```
//two.cpp  ::  32-bitise TTSi juhtprogramm, VisualC++, Dev-C++
//aktsepteeritud 64-bitise masina poolt; asm-väljund on MASM32.
#include <afxwin.h>
#include <afxext.h>
#include <afxcmn.h>
#include "windows.h"
#include "resource.h"
#include "two32.h"

char Pf[20];
char G[20];
char W[20];
HINSTANCE rc;
//int G_lines;
//int W_lines;
char FailiNimi[256];
char *Name;
char *GrammarName;
char *ProgramName;
char gn[20];
char pn[20];
char Ln[256];
int lines=0;
char MB[256];
char Title[100];
char *filter;
BOOL gr,wd,pr;
char pp[256];
FILE *f=NULL;

RECT r={0,0,800,510};
HWND hwndSB;
OPENFILENAME a;
HMENU hMenu;
HMENU hLang;
HMENU hLear;
HMENU hForm;

LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
char szWinName[]="MyWin";

int WINAPI WinMain(HINSTANCE hThisInst,HINSTANCE hPrevInst,
      LPSTR lpszArgs,int WinMode){
MSG msg;
HWND hwnd;
WNDCLASSEX wcl;
      wcl.hInstance=hThisInst;
      wcl.lpszClassName=szWinName;
      wcl.lpfnWndProc=WndProc;
      wcl.style=CS_HREDRAW|CS_VREDRAW;
      wcl.cbSize=sizeof(WNDCLASSEX);
      wcl.hIcon=LoadIcon(NULL,IDI_APPLICATION);
      wcl.hIconSm=LoadIcon(NULL,IDI_WINLOGO);
      wcl.hCursor=LoadCursor(NULL,IDC_ARROW);
      wcl.lpszMenuName=MAKEINTRESOURCE(IDR_MENU1);
      wcl.cbClsExtra=0;
      wcl.cbWndExtra=0;
      wcl.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
```

```
	if(!RegisterClassEx(&wcl)) return 0;
	hMenu=LoadMenu(hThisInst,MAKEINTRESOURCE(IDR_MENU1));
	lines=0;
	GrammarName=(char *)malloc(256);
	memset(GrammarName,'\0',256);
	ProgramName=(char *)malloc(256);
	memset(ProgramName,'\0',256);
	hwnd=CreateWindow(
		szWinName,
		"Learning Compiler Compiler",
		WS_OVERLAPPEDWINDOW|WS_HSCROLL|WS_VSCROLL,
		0,0,800,572,
		HWND_DESKTOP,
		hMenu,
		hThisInst,
		NULL);
	EnableMenuItem(hMenu,1,MF_BYPOSITION | MF_GRAYED);
	EnableMenuItem(hMenu,2,MF_BYPOSITION | MF_GRAYED);
	EnableMenuItem(hMenu,3,MF_BYPOSITION | MF_GRAYED);
	EnableMenuItem(hMenu,4,MF_BYPOSITION | MF_GRAYED);
	EnableMenuItem(hMenu,5,MF_BYPOSITION | MF_GRAYED);
//	EnableMenuItem(hMenu,6,MF_BYPOSITION | MF_ENABLED);
//	EnableMenuItem(hMenu,6,MF_BYPOSITION | MF_GRAYED);
	hLang=GetSubMenu(hMenu,3);
	hLear=GetSubMenu(hMenu,2);
	hForm=GetSubMenu(hMenu,4);
	EnableMenuItem(hLear,1,MF_BYPOSITION | MF_GRAYED);
	EnableMenuItem(hLang,1,MF_BYPOSITION | MF_GRAYED);
	EnableMenuItem(hLang,2,MF_BYPOSITION | MF_GRAYED);
	EnableMenuItem(hLang,3,MF_BYPOSITION | MF_GRAYED);
	EnableMenuItem(hLang,4,MF_BYPOSITION | MF_GRAYED);
	EnableMenuItem(hForm,1,MF_BYPOSITION | MF_GRAYED);
	EnableMenuItem(hForm,2,MF_BYPOSITION | MF_GRAYED);
	ShowWindow(hwnd,SW_SHOW);
	UpdateWindow(hwnd);
	gr=wd=pr=0;
	memset(Ln,'\0',256);
	Name=(char *)malloc(12);
	filter=(char *)malloc(256);
	while(GetMessage(&msg,NULL,0,0)){
		TranslateMessage(&msg);
		DispatchMessage(&msg);
		}
	return msg.wParam;
}

void GetName(void){
	int i,j,k;
	k=0;
	memset(Name,'\0',12);
	for(i=0;i<256;i++){
		if(GrammarName[i]=='\\') k=i+1;
		if(GrammarName[i]=='0') goto makename;
		}
	makename: for(j=0;j<12;j++){
				if(GrammarName[k]=='.') break;
				Name[j]=GrammarName[k];
				k++;
				}
	strcpy(gn,Name);
}
```

```
BOOL Get_FileName(HWND owner){
      ZeroMemory(&FailiNimi,256);
      a.lStructSize=sizeof(OPENFILENAME);
      a.hwndOwner=owner;
      char ff[256]="";
      a.lpstrFile=ff;
      a.nMaxFile=256;
      a.lpstrTitle=Title;
      a.nFilterIndex=1;
      a.lpstrCustomFilter=NULL;
      a.Flags=OFN_PATHMUSTEXIST|OFN_FILEMUSTEXIST;
      a.lpstrFileTitle=NULL;
//    a.lpstrInitialDir="E:\\Program Files\\Microsoft Visual Studio\\TTS";
      a.nMaxFileTitle=0;
      if(GetOpenFileName(&a)) strcpy(FailiNimi,a.lpstrFile);
      else return FALSE;
      return TRUE;
}

void Clean(HWND hwnd){
HDC hdc;
HBRUSH hbr;
      hdc=GetDC(hwnd);
      hbr=(HBRUSH)GetStockObject(WHITE_BRUSH);
      FillRect(hdc,&r,hbr);
      ReleaseDC(hwnd,hdc);
}

BOOL SelectGrammar(HWND hwnd){
      BOOL ret;
      Clean(hwnd);
     ZeroMemory(&a,sizeof(OPENFILENAME));
      pn[0]='\0';
      sprintf(Title,"Select a Grammar");
      a.lpstrFilter="Grammar\0*.grm\0\0";
      if(!Get_FileName(hwnd)){
            ret=FALSE;
            goto p;
            }
      ret=TRUE;
      SetWindowText(hwnd,FailiNimi);
      memset(GrammarName,'\0',256);
      strcpy(GrammarName,FailiNimi);
      GetName();
      strcpy(G,gn);
      strcat(G,".grm");
      ShellExecute(hwnd,"open","notepad.exe",G,NULL,SW_SHOWNORMAL);
      EnableMenuItem(hMenu,0,MF_BYPOSITION | MF_ENABLED);
      EnableMenuItem(hMenu,1,MF_BYPOSITION | MF_ENABLED);

      if(Name[0]=='T'){
            EnableMenuItem(hMenu,3,MF_BYPOSITION | MF_ENABLED);
            EnableMenuItem(hMenu,2,MF_BYPOSITION | MF_GRAYED);
            EnableMenuItem(hMenu,4,MF_BYPOSITION | MF_GRAYED);
            goto p;
            }
      if((Name[0]=='F')||(Name[0]=='f')){
            EnableMenuItem(hMenu,4,MF_BYPOSITION | MF_ENABLED);
            EnableMenuItem(hMenu,2,MF_BYPOSITION | MF_GRAYED);
            EnableMenuItem(hMenu,3,MF_BYPOSITION | MF_GRAYED);
```

```
            goto p;
            }
      if(Name[0]=='G'){
            EnableMenuItem(hMenu,2,MF_BYPOSITION | MF_ENABLED);
            EnableMenuItem(hMenu,3,MF_BYPOSITION | MF_GRAYED);
            EnableMenuItem(hMenu,4,MF_BYPOSITION | MF_GRAYED);
            }
p:    if(ret==TRUE) EnableMenuItem(hMenu,5,MF_BYPOSITION | MF_ENABLED);
//    EnableMenuItem(hMenu,6,MF_BYPOSITION | MF_ENABLED);
      DrawMenuBar(hwnd);
      WndProc(hwnd,WM_SIZE,SIZE_RESTORED,MAKELPARAM(776,510));
      InvalidateRect(hwnd,&r,TRUE);
      return ret;
};

BOOL Word(HWND hwnd){
char *p;
char s[20];
      ZeroMemory(&s,20);
      Clean(hwnd);
     ZeroMemory(&a,sizeof(OPENFILENAME));
      sprintf(Title,"Select a Word");
      memset(filter,'\0',256);
      sprintf(filter,"Word%c*.%s",'\0',gn);
      a.lpstrFilter=filter;
      if(!Get_FileName(hwnd)) return FALSE;
      memset(ProgramName,'\0',256);
      strcpy(ProgramName,FailiNimi);
      p=strrchr(ProgramName,'\\');
      strcpy(pn,p+1);
      p=strrchr(pn,'.');
      strcpy(Pf,pn);
      p[0]='\0';
      ShellExecute(hwnd,"open","notepad.exe",Pf,NULL,SW_SHOWNORMAL);
      EnableMenuItem(hLear,1,MF_BYPOSITION | MF_ENABLED);
      DrawMenuBar(hwnd);
      WndProc(hwnd,WM_SIZE,SIZE_RESTORED,MAKELPARAM(776,510));
      InvalidateRect(hwnd,&r,TRUE);
      return TRUE;
    };

BOOL TriProg(HWND hwnd){
char *p;
      Clean(hwnd);
     ZeroMemory(&a,sizeof(OPENFILENAME));
      sprintf(Title,"Select a Program");
      a.lpstrFilter="Program\0*.tri\0\0";
      if(!Get_FileName(hwnd)) return FALSE;
      memset(ProgramName,'\0',256);
      strcpy(ProgramName,FailiNimi);
      p=strrchr(ProgramName,'\\');
      strcpy(pn,p+1);
      strcpy(Pf,pn);
      p=strrchr(pn,'.');
      p[0]='\0';
      ShellExecute(hwnd,"open","notepad.exe",Pf,NULL,SW_SHOWNORMAL);
      strcpy(pp,pn);
      p=strrchr(pp,'.');
      if(p!=NULL) p[0]='\0';
      strcat(pp,".pt");
```

```
      EnableMenuItem(hLang,1,MF_BYPOSITION | MF_ENABLED);
      EnableMenuItem(hLang,2,MF_BYPOSITION | MF_ENABLED);
      EnableMenuItem(hLang,3,MF_BYPOSITION | MF_ENABLED);
      EnableMenuItem(hLang,4,MF_BYPOSITION | MF_ENABLED);
      WndProc(hwnd,WM_SIZE,SIZE_RESTORED,MAKELPARAM(776,510));
      InvalidateRect(hwnd,&r,TRUE);
      return TRUE;
};

BOOL Formel(HWND hwnd){
      Clean(hwnd);
      char *p;
     ZeroMemory(&a,sizeof(OPENFILENAME));
      sprintf(Title,"Select a Formula");
      a.lpstrFilter="Word\0*.frm\0\0";
      if(!Get_FileName(hwnd)) return FALSE;
      SetWindowText(hwnd,FailiNimi);
      memset(ProgramName,'\0',256);
      strcpy(ProgramName,FailiNimi);
      p=strrchr(ProgramName,'\\');
      strcpy(pn,p+1);
      strcpy(Pf,pn);
      //AfxMessageBox(pn);
      p=strrchr(pn,'.');
      p[0]='\0';
      ShellExecute(hwnd,"open","notepad.exe",Pf,NULL,SW_SHOWNORMAL);
      EnableMenuItem(hForm,1,MF_BYPOSITION | MF_ENABLED);
      EnableMenuItem(hForm,2,MF_BYPOSITION | MF_ENABLED);
      WndProc(hwnd,WM_SIZE,SIZE_RESTORED,MAKELPARAM(776,510));
      InvalidateRect(hwnd,&r,TRUE);
      return TRUE;
    };

BOOL Files(HWND hwnd){
char *n,*p;
int i;
char ext[6][4]={"grm","htm","sem","tri","frm","asm"};
      Clean(hwnd);
select:
     ZeroMemory(&a,sizeof(OPENFILENAME));
      sprintf(Title,"Select a File");
      memset(filter,'\0',256);
      if(pn[0]=='\0'){
            sprintf(filter,"%s%c%s.grm;%s.htm;%s.sem;%srd.htm",
                  gn,'\0',gn,gn,gn,gn);
            goto go;
            }
      sprintf(filter,
"%s%c%s.grm;%s.htm;%s.sem;%srd.htm;%s.%s;%s.asm;%sp.htm;%si.htm;%sc.htm;%so
c.htm;%srd.htm",
gn,'\0',gn,gn,gn,gn,pn,gn,pn,pn,pn,pn,pn,pn);
go:   a.lpstrFilter=filter;
      if(!Get_FileName(hwnd)) return FALSE;
      n=strrchr(FailiNimi,'\\');
      n++;
      p=strrchr(n,'.');
      for(i=0;i<19;i++){
            if(strcmp(p+1,ext[i])==0) goto fork;
            }
      return FALSE;
fork:
```

```
      switch(i){
            case 0:   //grm

      ShellExecute(hwnd,"open","notepad.exe",n,NULL,SW_SHOWNORMAL);
                  break;
            case 1:   //htm
                  ShellExecute(hwnd,"open",FailiNimi,NULL,NULL,SW_SHOW);
                  break;
            case 2:    //sem

      ShellExecute(hwnd,"open","notepad.exe",n,NULL,SW_SHOWNORMAL);
                  break;
            case 3:    //tri

      ShellExecute(hwnd,"open","notepad.exe",n,NULL,SW_SHOWNORMAL);
                  break;
            case 4:    //frm

      ShellExecute(hwnd,"open","notepad.exe",n,NULL,SW_SHOWNORMAL);
                  break;
            case 5:    //asm

      ShellExecute(hwnd,"open","notepad.exe",n,NULL,SW_SHOWNORMAL);
                  break;
            }
      goto select;
      return TRUE;
    };

LRESULT CALLBACK WndProc(HWND hwnd,UINT message,WPARAM wParam,LPARAM
lParam){
   switch(message){
      case WM_DESTROY:
         PostQuitMessage(0);
         return 0;
      case WM_COMMAND:
         switch(LOWORD(wParam)){
            case ID_GRAMMAR:
               Clean(hwnd);
               gr=SelectGrammar(hwnd);
               break;
            case ID_CONSTRUCTOR:
               Clean(hwnd);
               sprintf(MB,"wcr32 %s",Name);
               system(MB);
               sprintf(Ln,"%s.htm",gn);
               ShellExecute(hwnd,"open",Ln,NULL,NULL,SW_SHOW);
               break;
            case ID_WORD:
               Clean(hwnd);
               if(Word(hwnd)==FALSE) break;
               break;
            case ID_PARSER:
               Clean(hwnd);
               sprintf(MB,"parser32 %s %s %s",gn,pn,pn);
               // AfxMessageBox(MB);
               system(MB);
               sprintf(Ln,"%sp.htm",pn);
               //AfxMessageBox(Ln);
               rc=ShellExecute(hwnd,"open",Ln,NULL,NULL,SW_SHOW);
               //sprintf(Ln,"code=%d",rc);
```

```
   //AfxMessageBox(Ln);
   break;
case ID_PROGRAM:
   Clean(hwnd);
   if(TriProg(hwnd)==FALSE) break;
   break;
case ID_INTERP:
   f=fopen(pp,"rb");
   if(f==NULL){
      sprintf(MB,"parser32 %s %s %s",gn,pn,pn);
      system(MB);
      }
   Clean(hwnd);
   sprintf(MB,"int32 %s",pn);
   system(MB);
   sprintf(Ln,"%si.htm",pn);
   ShellExecute(hwnd,"open",Ln,NULL,NULL,SW_SHOW);
   break;
case ID_SIMPLE:
   f=fopen(pp,"rb");
   if(f==NULL){
      sprintf(MB,"parser32 %s %s %s",gn,pn,pn);
      system(MB);
      }
   Clean(hwnd);
   sprintf(MB,"cmp32 %s",pn);
   system(MB);
   sprintf(Ln,"%sc.htm",pn);
   ShellExecute(hwnd,"open",Ln,NULL,NULL,SW_SHOW);
   sprintf(MB,"%s.asm",pn);
   ShellExecute(hwnd,"open","notepad.exe",MB,NULL,
      SW_SHOWMAXIMIZED);
   break;
case ID_OPT:
   f=fopen(pp,"rb");
   if(f==NULL){
      sprintf(MB,"parser32 %s %s %s",gn,pn,pn);
      system(MB);
      }
   Clean(hwnd);
   sprintf(MB,"cmp32 %s o",pn);
   system(MB);
   sprintf(Ln,"%soc.htm",pn);
   ShellExecute(hwnd,"open",Ln,NULL,NULL,SW_SHOW);
   sprintf(MB,"%s.asm",pn);
   ShellExecute(hwnd,"open","notepad.exe",MB,NULL,
      SW_SHOWMAXIMIZED);
   break;
case ID_FORM:
   Clean(hwnd);
   Formel(hwnd);
   break;
case ID_TRANS:
   Clean(hwnd);
   sprintf(MB,"log32 %s %s %s",gn,pn,pn);
   //AfxMessageBox(MB);
   system(MB);
   sprintf(Ln,"%s.htm",pn);
   //AfxMessageBox(Ln);

      ShellExecute(hwnd,"open",Ln,NULL,NULL,SW_SHOW);
```

```
                  break;
            case ID_FILES:
                  Clean(hwnd);
                  Files(hwnd);
                  break;
            case ID_CR:
                  ShellExecute(hwnd,"open","notepad.exe","wcr32.c",NULL,
                        SW_SHOWMAXIMIZED);
                        break;
            case ID_P:
                  ShellExecute(hwnd,"open","notepad.exe","parser32.c",
                        NULL,SW_SHOWMAXIMIZED);
                  break;
            case ID_TRII:
                  ShellExecute(hwnd,"open","notepad.exe","int32.c",
                        NULL,SW_SHOWMAXIMIZED);
                  break;
            case ID_TRIC:
                  ShellExecute(hwnd,"open","notepad.exe","cmp32.c",
                        NULL,SW_SHOWMAXIMIZED);
                  break;
            case ID_TWO:
                  ShellExecute(hwnd,"open","notepad.exe","two.cpp",NULL,
                        SW_SHOWMAXIMIZED);
                  break;
            case ID_HEADER:
                  ShellExecute(hwnd,"open","notepad.exe","two32.h",
                        NULL,SW_SHOWMAXIMIZED);
                  break;
            case ID_G:
                  ShellExecute(hwnd,"open","grm.htm",NULL,NULL,SW_SHOW);
                  break;
            case ID_HELP:
                  //Clean(hwnd);
                  //AfxMessageBox("Help?");
                  ShellExecute(hwnd,"open","help.htm",
                        NULL,NULL,SW_SHOW);
                  break;
            }
      }
      return DefWindowProc(hwnd,message,wParam,lParam);
}
```

Lisa 6. Konstruktor

```
/* wcr32.c  :: Constructor for VisualC++ 6.0 (HTML-logi). December 2000  */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
#include <sys/stat.h>
#include "two32.h"

struct D *SD[HTL];
struct R *SR[HTL];

FILE *Sem=NULL;

struct U{
      int x;
      int y;
      struct U *next;
};

struct U *UP=NULL;

char s[8][60]={"",
      "<FONT COLOR=\"#0000FF\">=&#149</FONT>",
      "<FONT COLOR=\"#0000FF\">&lt&#149</FONT>",
      "<FONT COLOR=\"#FF0000\">&lt&#149=&#149</FONT>",
      "<FONT COLOR=\"#0000FF\">&#149&gt</FONT>",
      "<FONT COLOR=\"#0000FF\">=&#149&#149&gt</FONT>",
      "<FONT COLOR=\"#FF0000\">&lt&#149&#149&gt</FONT>",
      "<FONT COLOR=\"#FF0000\">&lt&#149=&#149&#149&gt</FONT>"
      };

/* char s[8][30]={"",
      "<IMG SRC=\"a.gif\">",
      "<IMG SRC=\"e.gif\">",
      "<IMG SRC=\"ea.gif\">",
      "<IMG SRC=\"j.gif\">",
      "<IMG SRC=\"ja.gif\">",
      "<IMG SRC=\"ej.gif\">",
      "<IMG SRC=\"eaj.gif\">"
      };    */


void printD(struct D *t);
void make_rtf(void);


/* tekstimassiivi sisestamine kettalt */
char *r_text(char *pealkiri){
      FILE *tekst=NULL;
      char *B;
      char c;
      int  i;
      int  k;
      struct stat *buf;
      buf=(struct stat *)malloc(sizeof(struct stat));
      if(buf==NULL){
```

```
            sprintf(rida,"r_text: I haven't %d bytes of memory..",
                  sizeof(struct stat));
            ExiT(rida);
            }
      tekst = fopen(pealkiri, "r");
      if (tekst==NULL){
            sprintf(rida,"cannot find the file  %s",pealkiri);
            ExiT(rida);
            }
      if(stat(pealkiri,buf)==-1){
            sprintf(rida,"r_text: stat failed");
            ExiT(rida);
            }
      k=buf->st_size;
      B = (char *)malloc(k+10);
      if(B == NULL){
            sprintf(rida,"r_text: I have't %d bytes of memory..",k+10);
            ExiT(rida);
            }
      memset(B,'\0',k+10);
/* fill buffer */
      rewind(tekst);
      i=0;
      while (!feof(tekst)){
            c=fgetc(tekst);
            B[i]=c;
            i++;
      };
      P_length=i;
      for(i=P_length;i>0;i--){
            c=B[i]; if(isgraph(c)){
                  i++;
                  B[i]='\n';
                  i++;
                  B[i]='\0';
                  P_length=i;
                  goto out;
                  }
            }
out:  fclose(tekst);
      return(B);
}

int gramm(void){
      GBuf=r_text(fname);
      if(GBuf!=NULL){
            Glen=P_length;
            return(1);
            }
      return(0);
}

/* adresseerib maatrikseid  süntaksi analüüsi ajal. nr=V pikkus
i=0..ridade arv - 1 j=0..veergude arv -1 */
int addr(int i, int j){
      int k;
      if(NR>0) goto indeks;
      if((i<1)||(j<1)||(i>nr)||(j>nr)){
            sprintf(rida,"addr: erroneus index nr=%d tnr=%d i=%d j=%d",
                  nr,tnr,i,j);
            ExiT(rida);
```

```
            }
      indeks:
      k=(i-1)*nr+j;
      if(NR>0) return(k);
      if(k>(nr+1)*(nr+1)){
            sprintf(rida,"addr: index %d □le piiri. i=%d j=%d
nr=%d",k,i,j,nr);
            ExiT(rida);
            }
      return(k);
}

//mälueraldus d × d maatriksile
char *DefArray(int d){
      int n,i;
      char *M;
      n=(d*d);
      M=(char *)malloc(n);
      if(M==(char *)NULL){
            sprintf(rida,"DefArray: I don't have memory enough..");
            ExiT(rida);
            }
      for(i=0;i<n;i++) M[i]='\0';
      return(M);
}

/* vektoritele mälu eraldamine */
char *DefVect(int n){
      int i;
      char *M;
      M=(char *)malloc(n);
      if(M==(char *)NULL){
            sprintf(rida,"DefVect: I don't have memory enough..");
            ExiT(rida);
            }
      for(i=0;i<n;i++) M[i]='\0';
      return(M);
}

/* sõltuva konteksti puu Ülemine tase: mõisted */
char ***defDEP(void){
      int i,n;
      n=(nr+1)*sizeof(char ***);
      DEP=(char ***)malloc(n);
      if(DEP==(char ***)NULL){
            sprintf(rida,"defDEP: I don't have memory enough..");
            ExiT(rida);
            }
      for(i=0;i<nr+1;i++) DEP[i]=(char **)NULL;
      return(DEP);
}

/* sõltuva konteksti puu vahetase: vasakud kontekstid */
char **defL(void){
      int i,n;
      char **L;
      n=(nr+1)*sizeof(char **);
      L=(char **)malloc(n);
      if(L==(char **)NULL){
            sprintf(rida,"defL: I don't have memory enough..");
            ExiT(rida);
```

```
            }
      for(i=0;i<nr+1;i++) L[i]=(char *)NULL;
      return(L);
}

void ExiT(char *t){
      fprintf(Logi,"%s&lt;BR&gt;",t);
      fprintf(Logi,"&lt;"/BODY&gt;&lt;"/HTML&gt;");
      fflush(Logi);
      fclose(Logi);
      printf("%s",t);
      getchar();
      abort();
}

/* produktsioonide paisktabeli lüli trükk */
void print_h(struct h *t){
      int i,j;
      char *r;
      fprintf(Logi,"P=%d `%s' -&gt ",t->P,T[t->NT]);
      for(i=0;i<t->n;i++){
            j=t->d[i];
            r=T[j];
            if(j<=tnr) fprintf(Logi,"%s ",r);
            else fprintf(Logi,"`%s' ",r);
            }
      fprintf(Logi,"<BR>");
}

/* paiskfunktsioon (võti on prod. vahekeelne parem pool) */
int hfunc(int keylen,int *key){
      int h=0,g;
      int i;
      for(i=0;i<keylen;i++) h=h^key[i];
      g=h;
      h=h/HTL;
      h=g-(h*HTL);
      if((h<0)||(h>HTL)) ExiT("HT error!");
      return(h);
}

/* produktsiooni otsimine paisktabelist */
struct h *ReadHash(int n,int *key){
      struct h *t;
      int i,a;
      a=hfunc(n,key);
      t=HT[a];
otsi:
      if(t!=(struct h *)NULL){
            if(n==t->n){
                  for(i=0;i<n;i++){
                        if(key[i]!=t->d[i]) goto next;
                        }
                  return(t);
                  }
            next: t=t->col;
            goto otsi;
            }
      return(NULL);
}
```

```
/* produktsiooni paiskamine */
int HashRule(struct D *t,struct R *r){
      struct h *L,*TP,*S;
      int a,i;
      L=(struct h *)malloc(sizeof(struct h));
      if(L==NULL){
            sprintf(rida,"HashRule: I don't have memory enaugh..");
            ExiT(rida);
            }
      memset(L,'\0',sizeof(struct h));
      L->col=(struct h *)NULL;
      L->same=(struct h *)NULL;
      L->d=malloc(r->n*sizeof(int));
      if(L->d==NULL){
            sprintf(rida,"HashRule: I don't have memory enaugh..");
            ExiT(rida);
            }
      memcpy(L->d,r->d,r->n*sizeof(int));
      L->n=r->n;
      L->NT=t->code;
      L->P=r->P;
      a=hfunc(L->n,L->d);
      if(HT[a] == (struct h *)NULL){
            HT[a]=L;
            return(1);
            }
      TP=HT[a];
      while(TP!=(struct h *)NULL){
            if(L->n==TP->n){
                  for(i=0;i<L->n;i++){
                        if(L->d[i]!=TP->d[i]) goto col;

                        }
                  S=TP->same;
                  TP->same=L;
                  L->same=S;
                  BRC++;
                  return(1);
                  }
            col: TP=TP->col;
            }
      TP=HT[a];
      HT[a]=L;
      L->col=TP;
      return(0);
}

/* Mitteterminali definitsioonide salvestamine */
void HashNT(struct D *t){
      struct R *r;
      r=t->def;
      while(r!=(struct R *)NULL){
            HashRule(t,r);
            r=r->alt;
            }
}

/* kõikide mitteterminalide definitsioonide salvestamine */
void HashRules(struct D *t){
      if(t!=(struct D *)NULL){
            if(t->tunnus==0) HashNT(t);
```

```
            HashRules(t->left);
            HashRules(t->right);
            }
}

/* produktsioonide definitsitsioonide ahela  lüli trükk */
void print_L(struct h *CP){
int i,j;
char *r;
      fprintf(Logi,"P%2d `%s' -&gr ",CP->P,T[CP->NT]);
      for(j=0;j<CP->n;j++){
            i=CP->d[j];
            r=T[i];
            if(i<tnr+1) fprintf(Logi,"%s ",r);
            else fprintf(Logi,"`%s' ",r);
            }
      fprintf(Logi,"<BR>");
}

/* produktsioonide paisktabeli trükk */
void print_HT(void){
      struct h *CP,*s;
      int i;
            fprintf(Logi,"<H4>Productions (HT):</H4>");
            for(i=0; i<HTL; i++){
                  CP=HT[i];
                  while(CP != (struct h *)NULL){
                        print_L(CP);
                        s=CP->same;
                        while(s != (struct h *)NULL){
                              print_L(s);
                              s=s->same;
                              }
                        CP=CP->col;
                        }
                  }
}


/* leftmost ja rightmost: esimene lähendus */
void first_LR(struct D *t){
      struct R *r;
      int x;
      r=t->def;
ring: if(r != (struct R *)NULL){
            if(t->tunnus==0){
                  x=r->d[0];
                  Lm[t->code][x]=1;
                  x=r->d[r->n-1];
                  Rm[t->code][x]=1;
                  }
            r=r->alt;
            goto ring;
            }
}

/* leftmost: lisatakse L(A), kui A kuulub L */
void sec_LRL(int nt){
      int i,j;
      for(j=1; j<nr+1; j++){
            if(Lm[j][nt]==1){
```

```
                  for(i=1;i<nr+1;i++){
                        if(Lm[nt][i]==1) Lm[j][i]=1;
                        }
                  }
            }
}

/* rightmost: lisatakse R(A), kui A kuulub R */
void sec_LRR(int nt){
      int i,j;
      for(j=1; j<nr+1; j++){
            if(Rm[j][nt]==1){
                  for(i=1;i<nr+1;i++){
                        if(Rm[nt][i]==1) Rm[j][i]=1;
                        }
                  }
            }
}

/* mitteterminalide left- ja rightmost-hulkade moodustamine */
void make_LRB(void){
      int i;
      struct D *t;
      for(i=1; i<nr+1; i++){
            t=getV(T[i]);
/*          if(logi==1) fprintf(Logi,"LRM NT=%s<BR>",T[t->code]);      */
            first_LR(t);
            }
      for(i=nr; i>tnr; i--){
            sec_LRL(i);
            sec_LRR(i);
            }
}

/* eelnevusmaatriksi koostamine: NT[i] relatsioonid maatriksisse */
int relac(struct D *t){
      struct R *r;
      int i,j,k,x,y;
      int ret=1;
      r=t->def;
ring: if(r!=(struct R *)NULL){
                  if(r->n==1) goto next;
                  for(i=0;i<=r->n-2;i++){         /* def. paar x,y */
                        x=r->d[i];
                        y=r->d[i+1];
                        if(Pm[x][y]!=0 && Pm[x][y]!=1) ret=0;
                        Pm[x][y]=Pm[x][y]|1;      /* ajastub */
                        if(y>tnr){         /* parem naaber on NT */
                              for(k=1;k<nr+1;k++){
                                    if(Lm[y][k]==1){
                                       if(Pm[x][k]!=0 && Pm[x][k]!=2) ret=0;
                                       Pm[x][k]=Pm[x][k]|2;  //iga eelneb
                                       }
                                    }
                              }
                        if(x>tnr){           /* X on NT */
                              for(k=1;k<nr+1;k++){
                                    if(Rm[x][k]==1){
                                       if(Pm[k][y]!=0 && Pm[k][y]!=4) ret=0;
                                       Pm[k][y]=Pm[k][y]|4; // iga järgneb
                                       }
```

```
                                    }
                              }
                        if((x>tnr)&&(y>tnr)){   /* X ja Y on NT */
                              /* R(X) > L(Y) */
                              for(j=1;j<nr+1;j++){
                                    if(Rm[x][j]==1){
                                     for(k=1;k<nr+1;k++){
                                        if(Lm[y][k]==1){
                                          if(Pm[j][k]!=0&&Pm[j][k]!=4) ret=0;
                                          Pm[j][k]=Pm[j][k]|4;
                                          }
                                        }
                                     }
                              }
                        }
                  }
            next: r=r->alt;
            goto ring;
            }
      return(ret);
}

/* eelnevusmaatriksi koostamine: anna järgmine NT */
int make_PM(void){
      int i;
      int ret=1;
      struct D *t;
      for(i=tnr+1; i<nr+1; i++){
            t=getV(T[i]);
/*          fprintf(Logi,"make_PM: i=%d NT=%s<BR>",i,T[i]);
            printD(t);  */
            ret=ret & relac(t);
            }
      return(ret);
}

/* left- ja rightmost-hulkade trükk */
void print_LRM(char M[HTL][HTL]){
      int i,j;
      fprintf(Logi,"<TABLE BORDER=1>");
      fprintf(Logi,"<TR><TD><FONT COLOR=\"#0000FF\">Symbol</TD>");
      for(i=1;i<nr+1;i++){
         if(Nimi[0]=='G'||Nimi[0]=='g')
          fprintf(Logi,"<TD><FONT COLOR=\"#0000FF\">%s</FONT></TD>",T[i]);
         else fprintf(Logi,"<TD><FONT COLOR=\"#0000FF\">%d</FONT></TD>",i);
       }
      fprintf(Logi,"</TR>");
      for(i=tnr+1; i<nr+1; i++){
       fprintf(Logi,"<TR><TD>%d.<FONTCOLOR=\"#0000FF\">%s</FONT></TD>",
               i,T[i]);
        for(j=1; j<nr+1; j++){
            if(M[i][j]>0)  fprintf(Logi,"<TD><FONT COLOR=\"#0000FF\">
                                                  *</FONT></TD>");
           else fprintf(Logi,"<TD>0</TD>");
          }
      fprintf(Logi,"</TR>");
      }
      fprintf(Logi,"</TABLE>");
}

void cf(char *c){
```

```
     fprintf(Logi,"<TD><FONT COLOR=\"#0000FF\"><STRONG>%s</FONT></TD>",c);
    }

void p_r(char *R){
      cf(R);
      }

void p_cr(char *R){
      cf(R);
      }

void print_rela(void){
int i,j,r;

      for(i=1;i<nr+1;i++){
            fprintf(Logi,
            "The relationships of symbol #<FONT COLOR=\"#0000FF\">%d ",i);
            if(i<=tnr) fprintf(Logi," %s</FONT>:<BR>",T[i]);
            else fprintf(Logi,"`%s'</FONT>:<BR>",T[i]);
            fprintf(Logi,"<TABLE BORDER=1><TR>");
            for(j=1;j<nr+1;j++){
                  r=Pm[i][j];
                  if(r>0){
                        fprintf(Logi,"<TD>%s ",s[r]);
                        if(j<=tnr) fprintf(Logi,"%s</TD>",T[j]);
                        else fprintf(Logi,"`%s'</TD>",T[j]);
                        }
                  }
            fprintf(Logi,"</TR></TABLE><BR>");
            }
}

/* eelnevusmaatriksi trükk */
void print_PM(void){
      int i,j;
      fprintf(Logi,"<TABLE BORDER=1><STRONG>");
      fprintf(Logi,"<TR><TD><FONT COLOR=\"#0000FF\">Symbol</FONT></TD>");
      for(i=1;i<nr+1;i++){
         if(Nimi[0]=='G'||Nimi[0]=='g')
            fprintf(Logi,"<TD><FONT
COLOR=\"#0000FF\">%s</FONT></TD>",T[i]);
          else fprintf(Logi,"<TD><FONT COLOR=\"#0000FF\">%d</FONT></TD>",i);
        }
      fprintf(Logi,"</TR>");
      for(i=1; i<nr+1; i++){
          fprintf(Logi,
                    "<TR><TD>%d.<FONT COLOR=\"#0000FF\">%s</FONT></TD>",
                    i,T[i]);
            for(j=1; j<nr+1; j++){
                switch(Pm[i][j]){
                  case 0: fprintf(Logi,"<TD>0</TD>"); break;
                  case 1: cf("="); break;
                  case 2: cf("&lt"); break;
                  case 4: cf("&gt"); break;
                  default:
                        fprintf(Logi, <TD><FONTCOLOR=\"#FF0000\"><STRONG>
                        %d</FONT></TD>",  Pm[i][j]);
                        break;
                  }
            }
            fprintf(Logi,"</TR>");
```

```
	}
	fprintf(Logi,"</TABLE>");
}

/* eelnevusmaatriksi trükk */
void print_rPM(void){
	int i,j;
	fprintf(rdf,"<TABLE BORDER=1><STRONG>");
	fprintf(rdf,"<TR><TD><FONT COLOR=\"#0000FF\">Nr</FONT></TD>");
	for(i=1;i<nr+1;i++)
		fprintf(rdf,"<TD><FONT COLOR=\"#0000FF\">%d</FONT></TD>",i);
	fprintf(rdf,"</TR>");
	for(i=1; i<nr+1; i++){
		fprintf(rdf,
			"<TR><TD><FONT COLOR=\"#0000FF\">%d</FONT></TD>",i);
		for(j=1; j<nr+1; j++) fprintf(rdf,"<TD>%d</TD>",Pm[i][j]);
		fprintf(rdf,"</TR>");
		}
	fprintf(rdf,"</TABLE>");
}

/* stratifikatsioon paremale (P2-konflikt) */
void s_to_right(int lim, struct D *t,struct R *d){
	int k,r;
	int *sec;
	struct D *n;
	struct R *b;
	fprintf(Logi,"The source is the production ");
	print_rule(t,d);
	memset(word,'\0',20);
	sprintf(word,"%s%d",T[t->code],Index);
	NR++;
	if(NR>HTL){
		sprintf(rida,"too many symbols");
		ExiT(rida);
		}
	memcpy(T[NR],word,20);
	Index++;
	DC=newD();
	b=newR();
	r=d->n-lim;
	b->d=(int *)malloc(r*sizeof(int));
	memset(b->d,'\0',r*sizeof(int));
	r=0;
	for(k=lim;k<d->n;k++){
		b->d[r]=d->d[k];
		r++;
		}
	b->n=r;
	DC->def=b;
	DC->code=NR;
	DC->tunnus=0;
	DC->L=t->L+1;
	b->P=Pnr;
	Pnr++;
	n=search(1);
	fprintf(Logi," I'll add a new NT ");
	print_rule(DC,b);

/* uus reegel on valmis. */
	r=lim+1;
```

```
      sec=(int *)malloc(r*sizeof(int));
      memset(sec,'\0',r*sizeof(int));
      for(k=0;k<lim;k++) sec[k]=d->d[k];
      sec[lim]=NR;
      d->n=lim+1;
/*    free(d->d);  */
      d->d=sec;
      fprintf(Logi,"  I'll change the production ");
      print_rule(t,d);
      }

/* P2_konflikti likvideerimine:
      A->xXYy asendada k6ikjal A->xXD, kus D->Yy  */
void P2(int x,int y){
      int i,j;
      struct D *t;
      struct R *d;
    /* otsin konflikti allikaid: definitsioone, kus on paar XY */
      for(i=tnr+1; i<nr+1; i++){
            t=getV(T[i]);
            d=t->def;
ring: if(d!=(struct R *)NULL){
            for(j=0;j<d->n-1;j++){
                  if(d->d[j]==x){
                        if(d->d[j+1]==y) s_to_right(j+1,t,d);
                        }
                  }
            d=d->alt;
            goto ring;
            }
      }
}

/* stratifikatsioon vasakule (P1-konflikt) */
void s_to_left(int lim, struct D *t,struct R *d){
      int k,r;
      int *sec;
      struct D *n;
      struct R *b;
      fprintf(Logi,"The source is the production ");
      print_rule(t,d);
      r=t->L;
      memset(word,'\0',20);
      sprintf(word,"%s%d",T[t->code],Index);
      NR++;
      if(NR>HTL){
            sprintf(rida,"too many symbols");
            ExiT(rida);
            }
      memcpy(T[NR],word,20);
      Index++;
      DC=newD();
      b=newR();
      b->d=(int *)malloc(lim*sizeof(int));
      memset(b->d,'\0',lim*sizeof(int));
      for(k=0;k<lim;k++) b->d[k]=d->d[k];
      b->n=lim;
      DC->def=b;
      DC->code=NR;
      DC->tunnus=0;
      DC->L=r+1;
```

```
      b->P=Pnr;
      Pnr++;
      n=search(1);
      fprintf(Logi," I'll add a new NT ");
      print_rule(DC,b);
/* uus reegel on valmis. */
      r=d->n-lim;
      sec=(int *)malloc(r*sizeof(int));
      memset(sec,'\0',r*sizeof(int));
      sec[0]=n->code;
      r=1;
      for(k=lim;k<d->n;k++){
            sec[r]=d->d[k];
            r++;
            }
      d->n=r;
/*    free(d->d); */
      d->d=sec;
      fprintf(Logi,"  I'll change the production ");
      print_rule(t,d);
      }

/* P1-konflikti likvideerimine:
   A->xXYy asendada k6ikjal A->DYy, kus D->xX  */
void P1(int x,int y){
      int i,j,u;
      struct D *t;
      struct R *d;
/* otsin konflikti allikaid: definitsioone, kus on paar XY
   või XZ ja Y kuulub LM[Z]*/
for(i=tnr+1; i<nr+1; i++){
      t=getV(T[i]);
      d=t->def;
ring: if(d!=(struct R *)NULL){
            for(j=0;j<d->n-1;j++){
                  if(d->d[j]==x){
                        if(d->d[j+1]==y){
                              s_to_left(j+1,t,d);
                              }
                        else{
                              if(d->d[j+1]>tnr){
                                    u=d->d[j+1];
                                    if(Lm[u][y]==1)   s_to_left(j+1,t,d);
                                    }
                              }
                        }
                  }
            d=d->alt;
            goto ring;
            }
      }
}

/* teade eelnevuskonfliktist */
void tell_conf(int p,int i,int j,int c){
      fprintf(Logi,"<FONT COLOR=\"#FF0000\">");
      fprintf(Logi,"<BR>P%d-conflict: </FONT>",p);
      if(i<=tnr) fprintf(Logi,"%s <FONT COLOR=\"#FF0000\">%s
</FONT>",T[i],s[c]);
      else fprintf(Logi,"`%s' <FONT COLOR=\"#FF0000\">%s
</FONT>",T[i],s[c]);
```

```
      if(j<=tnr) fprintf(Logi,"%s<BR>",T[j]);
      else fprintf(Logi,"`%s'<BR>",T[j]);
      fprintf(Logi,"</FONT>");
}

/* eelnevuskonfliktide detekteerimine ja likvideerimine */
int conflicts(void){
      int i,j;
      int res=0;
      int oldN;
      NR=nr;
      for(i=1;i<nr+1;i++){
            for(j=1;j<nr+1;j++){
                  if(Pm[i][j]==3){
                        res=1;
                        tell_conf(2,i,j,3);
                        oldN=NR;
                        P2(i,j);
                        if(NR==oldN){
                              fprintf(Logi,"Cannot find any source");
                              }
                        }
                  }
            }
      if(res==0) goto p1;
      memset(Lm,'\0',HTL*HTL);
      memset(Rm,'\0',HTL*HTL);
      nr=NR;
      make_LRB();
      fprintf(Logi,"<H4>Leftmost-set</H4>");
      print_LRM(Lm);
p1:   for(i=1;i<nr+1;i++){
            for(j=1;j<nr+1;j++){
                  if(Pm[i][j]>4){
                        res=1;
                        tell_conf(1,i,j,Pm[i][j]);
                        oldN=NR;
                        P1(i,j);
                        if(NR==oldN){
                              fprintf(Logi,"Cannot find any source");
                              goto out;
                              }
                        }
                  }
            }
out:
      nr=NR;
      return(res);
}

/* vasaku konteksti maatriksi trükk */
void print_LC(void){
      int i,j;
      fprintf(Logi,"<TABLE BORDER=1>");
      fprintf(Logi,"<TR><TD><FONT COLOR=\"#0000FF\">Symbol</TD>");
      for(i=1;i<nr+1;i++){
            if(Nimi[0]=='G'||Nimi[0]=='g')
                  fprintf(Logi,"<TD><FONT COLOR=\"#0000FF\">%s</TD>",T[i]);
            else fprintf(Logi,"<TD><FONT COLOR=\"#0000FF\">%d</TD>",i);
            }
      fprintf(Logi,"</TR></FONT>");
```

```
      for(i=tnr+1; i<nr+1; i++){
            fprintf(Logi,
                  "<TR><TD>%d.<FONT COLOR=\"#0000FF\">%s</FONT></TD>",
                  i,T[i]);
            for(j=1; j<nr+1; j++){
               if(LC[addr(i,j)]>0)
                  fprintf(Logi,"<TD><FONT
COLOR=\"#0000FF\">*</FONT></TD>");
              else fprintf(Logi,"<TD>0</TD>");
                        }
            fprintf(Logi,"</TR>");
            }
      fprintf(Logi,"</TABLE>");
}

/* parema konteksti maatriksi trükk */
void print_RC(void){
      int i,j;
      fprintf(Logi,"<TABLE BORDER=1>");
      fprintf(Logi,"<TR><TD><FONT COLOR=\"#0000FF\">Symbol</TD>");
      for(i=1;i<tnr+1;i++){
            if(Nimi[0]=='G'||Nimi[0]=='g')
                  fprintf(Logi,"<TD><FONT COLOR=\"#0000FF\">%s</TD>",T[i]);
            else fprintf(Logi,"<TD><FONT COLOR=\"#0000FF\">%d</TD>",i);
            }
      fprintf(Logi,"</TR></FONT>");
      for(i=tnr+1; i<nr+1; i++){
          fprintf(Logi,"<TR><TD>%d.<FONT COLOR=\"#0000FF\">%s</FONT></TD>",
                  i,T[i]);
         for(j=1; j<tnr+1; j++){
            if(RC[addr(i,j)]>0)
            fprintf(Logi,"<TD><FONT COLOR=\"#0000FF\">*</FONT></TD>");
           else fprintf(Logi,"<TD>0</TD>");
          }
      fprintf(Logi,"</TR>");
      }
   fprintf(Logi,"</TABLE><BR><BR>");
}


/* vasaku konteksti maatriksi trükk */
void print_rLC(void){
      int i,j;
      fprintf(rdf,"<TABLE BORDER=1>");
      fprintf(rdf,"<TR><TD><FONT COLOR=\"#0000FF\">Nr</TD>");
      for(i=1;i<nr+1;i++){
            fprintf(rdf,"<TD><FONT COLOR=\"#0000FF\">%d</TD>",i);
            }
      fprintf(rdf,"</TR></FONT>");
      for(i=tnr+1; i<nr+1; i++){
          fprintf(rdf,"<TR><TD><FONT COLOR=\"#0000FF\">%d</FONT></TD>",i);
            for(j=1; j<nr+1; j++){
                  fprintf(rdf,"<TD>%d</TD>",LC[addr(i,j)]);
                  }
            fprintf(rdf,"</TR>");
            }
      fprintf(rdf,"</TABLE>");
}

/* parema konteksti maatriksi trükk */
void print_rRC(void){
```

```
      int i,j;
      fprintf(rdf,"<TABLE BORDER=1>");
      fprintf(rdf,"<TR><TD><FONT COLOR=\"#0000FF\">Nr</TD>");
      for(i=1;i<tnr+1;i++){
            fprintf(rdf,"<TD><FONT COLOR=\"#0000FF\">%d</TD>",i);
            }
      fprintf(rdf,"</TR></FONT>");
      for(i=tnr+1; i<nr+1; i++){
            fprintf(rdf,
                  "<TR><TD><FONT COLOR=\"#0000FF\">%d</FONT></TD>",i);
            for(j=1; j<tnr+1; j++){
                  fprintf(rdf,"<TD>%d</TD>",RC[addr(i,j)]);
                  }
            fprintf(rdf,"</TR>");
            }
      fprintf(rdf,"</TABLE><BR><BR>");
}

/* mitteterminali sõltumatu konteksti hulga moodustamine */
void brc(int NT){
      int A,i;
      A=NT;
      for(i=1;i<nr+1;i++){
            if((Pm[i][A]==1)||(Pm[i][A]==2)) LC[addr(A,i)]=1;
            }
      for(i=1;i<tnr+1;i++){
            if(Pm[A][i]>0) RC[addr(A,i)]=1;
            }
}

/* mitteterminali sõltumatu konteksti hulga trükk */
void p_brc(int NT){
      int A,i,f;
      A=NT;
      f=0;
      fprintf(Logi,"<FONT COLOR=\"#0000FF\">`%s' left context:
</FONT>",T[NT]);
      for(i=1;i<nr+1;i++){
           if(LC[addr(A,i)]==1){
            if(f==0) f++;
            else fprintf(Logi,"<FONT COLOR=\"#FF4500\"><STRONG> ,
</FONT>");
            if(i<=tnr) fprintf(Logi," %s ",T[i]);
            else fprintf(Logi," `%s' ",T[i]);
            }
      }
      fprintf(Logi,"<BR></FONT>");
      f=0;
      fprintf(Logi,"<FONT COLOR=\"#0000FF\">`%s' right context:
</FONT>",T[NT]);
      for(i=1;i<tnr+1;i++){
           if(RC[addr(A,i)]==1){
            if(f==0) f++;
            else fprintf(Logi,"<FONT COLOR=\"#FF4500\"><STRONG> ,
</FONT>");
            fprintf(Logi,"%s ",T[i]);
            }
      }
      fprintf(Logi,"<BR><BR>");
      fflush(Logi);
}
```

```
void p_all_brc(void){
      int i;
      fprintf(Logi,
            "<H4><FONT COLOR=\"#0000FF\">Independent context</H4></FONT>");
      for(i=tnr+1;i<=nr;i++) p_brc(i);
}

void p_lr(int NT){
      int A,i,f;
      A=NT;
      f=0;
      fprintf(Logi,"<FONT COLOR=\"#0000FF\">`%s' leftmost set:
</FONT>",T[NT]);
      for(i=1;i<nr+1;i++){
          if(Lm[A][i]==1){
            if(f==0) f++;
            else fprintf(Logi,"<FONT COLOR=\"#FF4500\"><STRONG> ,
</FONT>");
            fprintf(Logi," `%s' ",T[i]);
            }
      }
      fprintf(Logi,"<BR></FONT>");
      f=0;
      fprintf(Logi,"<FONT COLOR=\"#0000FF\">`%s' rightmost set:
</FONT>",T[NT]);
      for(i=1;i<nr+1;i++){
          if(Rm[A][i]==1){
            if(f==0) f++;
            else fprintf(Logi,"<FONT COLOR=\"#FF4500\"><STRONG> ,
</FONT>");
            fprintf(Logi,"%s ",T[i]);
            }
      }
      fprintf(Logi,"<BR><BR>");
      fflush(Logi);
}

void p_all_lr(void){
      int i;
      fprintf(Logi,
            "<H4><FONT COLOR=\"#0000FF\">Leftmost & rightmost
sets</H4></FONT>");
      for(i=tnr+1;i<=nr;i++) p_lr(i);
}

void make_all_indep(void){
int i;
for(i=tnr+1;i<nr+1;i++) brc(i);
fprintf(Logi,"<FONT COLOR=\"#0000FF\"><H4>Left Context</H4></FONT><BR>");
print_LC();
fprintf(Logi,"<BR><FONT COLOR=\"#0000FF\"><H4>Right
Context</H4></FONT><BR>");
print_RC();
}

int deptest(struct h *x,struct h *y){
      BRC++;
      fprintf(Logi,"independent context didn't help us. ");
     fprintf(Logi,"I'll try to use the dependent one.<BR>");
      fflush(Logi);
```

```
     if(DEP==NULL) DEP=defDEP();
     dc=1;
      memset(word,'\0',20);
      si=0;
      DepV[x->P]=1;
      DepV[y->P]=1;
      dc=search_springs(x,x->NT);
      if(dc==0) return(0);
      memset(word,'\0',20);
      si=0;
      dc=search_springs(y,y->NT);
      fflush(Logi);
      if(dc==1){
            dc=test_dep_con(x,y);
            if(dc==1) dep++;
            }
      fflush(Logi);
      return(dc);
}

int testpaar(int A,int B){
      int i,s;
      s=0;
      for(i=1;i<nr+1;i++){
            if(LC[addr(A,i)]==1){
                  if(LC[addr(B,i)]==1) goto rc;
                  }
            }
      fprintf(Logi,"<FONT COLOR=\"#008000\">The left context of </FONT>");
      fprintf(Logi,"`%s' ",T[A]);
      fprintf(Logi,"<FONT COLOR=\"#008000\">and </FONT> `%s'",T[B]);
      fprintf(Logi,"<FONT COLOR=\"#008000\"> is different<BR></FONT>");
      return(0);
rc:   for(i=1;i<tnr+1;i++){
            if(RC[addr(A,i)]==1){
                if(RC[addr(B,i)]==1){
                   fprintf(Logi,"<FONT COLOR=\"#FF0000\">");
                   fprintf(Logi,"The independent context of </FONT>`%s'
                   <FONT COLOR=\"#FF0000\">and </FONT>`%s' <FONT
                   COLOR=\"#FF00000\">is not different<BR>",
                        T[A],T[B]);
                  fprintf(Logi,"</FONT><HR>");
                  return(1);
                  }
            }
      }
      fprintf(Logi,"<FONT COLOR=\"#008000\">The right context of </FONT>");
      fprintf(Logi,"`%s' ",T[A]);
      fprintf(Logi,"<FONT COLOR=\"#008000\">and </FONT> `%s'",T[B]);
      fprintf(Logi,"<FONT COLOR=\"#008000\"> is different<BR></FONT>");
      return(0);
}

/*  mitteterminalide konteksti eristumise test */
int brctest(struct h *t){
      struct h *x,*y;
      struct h *SAM[40];
      int ns;
      int j,k,A,B;
      for(j=0;j<40;j++) SAM[j]=(struct h *)NULL;
      x=t;
```

```
      ns=0;
fill: SAM[ns]=x;
      ns++;
      x=x->same;
      if(x!=(struct h *)NULL) goto fill;
      fflush(Logi);
      for(j=0;j<ns-1;j++){
            x=SAM[j];
            A=x->NT;
            for(k=j+1;k<ns;k++){
                  y=SAM[k];
                  B=y->NT;
                  if(testpaar(A,B)!=0) deptest(x,y);
                  fflush(Logi);
                  }
            }
      return(1);
}

/* otsib ühesuguse parema poolega produktsioone, teeb C1|1-hulgad */
int make_BRC(void){
      struct h *CP,*S;
      int i,j,k;
      char *r;
      BRC=0;
fprintf(Logi,"<HR>");
      for(i=0; i<HTL; i++){
            CP=HT[i];
            while(CP != (struct h *)NULL){
                  if(CP->same!=(struct h *)NULL){
                        fprintf(Logi,"<FONT COLOR=\"#FF0000\">");
                        fprintf(Logi,"Equivalent definitions:</FONT><BR>");
                        S=CP;
                        p: fprintf(Logi,"`%s' &#151&gt ",T[S->NT]);
                        fflush(Logi);
                        for(j=0;j<S->n;j++){
                              k=S->d[j];
                              r=T[k];
                              if(k<=tnr) fprintf(Logi,"%s ",r);
                              else fprintf(Logi,"`%s' ",r);
                              fflush(Logi);
                              }
                        S=S->same;
                        if(S==(struct h *)NULL) goto B;
                        fprintf(Logi,"<FONT COLOR=\"#FF0000\"> &amp
</FONT>");
                        goto p;
                        B: fprintf(Logi,"<BR>");
                        fflush(Logi);
                        p_brc(CP->NT);
                        }
                  S=CP->same;
                  while(S != (struct h *)NULL){
                        p_brc(S->NT);
                        S=S->same;
                        fflush(Logi);
                        }
                  if(CP->same!=(struct h *)NULL){
                        fprintf(Logi,"<BR>");
                        if(brctest(CP)==0) return(0);
                        fprintf(Logi,"<BR>");
```

```
                        fflush(Logi);
                        }
                  CP=CP->col;
                  }
      }
      return(1);
}

/* eemaldab pseudo-sõltuvad kontekstid */
void del_dep(void){
      struct h *CP,*S;
      int i,k;
      for(i=0; i<HTL; i++){
            CP=HT[i];
            while(CP != (struct h *)NULL){
                  if(CP->same!=(struct h *)NULL){
                        k=CP->P;
                        if(DepV[k]==0) CP->nc=0;
                        }
                  S=CP->same;
                  while(S != (struct h *)NULL){
                        k=S->P;
                        if(DepV[k]==0) S->nc=0;
                        S=S->same;
                        }
                  CP=CP->col;
                  }
      }
}

void pset(int x,int y){
      fprintf(Logi," {");
      fprintf(Logi,"<FONT COLOR=\"#0000FF\">");
      if(x<=tnr) fprintf(Logi,"%s ",T[x]);
      else fprintf(Logi,"`%s' ",T[x]);
      fprintf(Logi,"<FONT COLOR=\"#FF4500\"><STRONG> , </FONT>");
      if(y<=tnr) fprintf(Logi,"%s",T[y]);
      else fprintf(Logi,"`%s'",T[y]);
      fprintf(Logi,"</FONT>");
      fprintf(Logi,"} ");
}

void priv(char *v){
      int i;
      for(i=0;i<tnr+1;i++) fprintf(Logi,"%d ",v[i]);
      fprintf(Logi,"<BR>");
}

void print_DEP(void){
char **L;
char *rc;
int i,j,k;
      for(i=0;i<nr+1;i++){
            L=DEP[i];
            if(L!=(char **)NULL){
                  fprintf(rdf,"<BR>DEP[%d]=%p<BR>",i,L);
                  for(j=0;j<nr+1;j++){
                        rc=L[j];
                        if(rc!=(char *)NULL){
                              fprintf(rdf,"L[%d]=%p<BR>",j,rc);
                              for(k=0;k<tnr+1;k++)
```

```
                                    if(rc[k]>0)
                                       fprintf(rdf,"rc[%d]=%d ",k,rc[k]);
                                 fprintf(rdf,"<BR>");
                                 }
                           }
                     }
               }
}

void join_cxt(char **D,char **S){
      int i,j;
      char *rc,*src;
      for(i=1;i<nr+1;i++){
            if(S[i]!=(char *)NULL){
                  src=S[i];
                  rc=D[i];
                  if(rc==(char *)NULL){
                        rc=DefVect(tnr+1);
                        D[i]=rc;
                        }
                  for(j=1;j<tnr+1;j++){
                        if(src[j]==1) rc[j]=src[j];
                        }
                  }
            }
}

int test_cxt(char **D,char **S){
      int i,j,x;
      int flag=1;
      char *rc,*src;
      for(i=1;i<nr+1;i++){
            src=S[i];
            if(src!=(char *)NULL){
                  rc=D[i];
                  if(rc!=(char *)NULL){
                        for(j=1;j<tnr+1;j++){
                          x=rc[j];
                            if(x==1){
                              if(src[j]==1){
                                flag=0;
                                fprintf(Logi,
                                  "<FONTCOLOR=\"red\"><BR>common:</FONT>");
                                pset(i,j);
                                fprintf(Logi,"<BR>");
                                }
                              }
                           }
                  }
              }
            }
      return(flag);
}

void print_cxt(char **L){
      int i,j;
      char *rc;
      for(i=1;i<nr+1;i++){
            rc=L[i];
            if(rc!=(char *)NULL){
                  for(j=1;j<tnr+1;j++){
```

```
                        if(rc[j]==1) pset(i,j);
                        }
                  }
            }
      fflush(Logi);
}

void print_cxtp(char **L){
int i;
      char *rc;
      for(i=1;i<nr+1;i++){
            rc=L[i];
            if(rc!=(char *)NULL) fprintf(Logi,"L[%d]=%p<BR>",i,rc);
            }
}

void print_all_cxt(void){
char **L;
int i;
      fprintf(Logi,
            "<H4><FONT COLOR=\"#0000FF\">Dependent context</H4></FONT>");
      for(i=tnr+1;i<nr+1;i++){
            L=DEP[i];
            if(L!=(char **)NULL){
                  fprintf(Logi,"<BR>dependent context of `%s':<BR>",T[i]);
                  print_cxt(L);
                  fprintf(Logi,"<BR>");
                  }
            }
/*    print_DEP();      */
}

void print_all_cxtp(void){
char **L;
int i;
      for(i=tnr+1;i<nr+1;i++){
            L=DEP[i];
            if(L!=(char **)NULL){
                  fprintf(Logi,"DEP[%d]=%p<BR>",i,L);
                  print_cxtp(L);
                  }
            }
}

/* sõltuv kontekst. g1(A)={(X,T) | B->uXADv & [T=D or T in L(D)]}  */
int g1(char **L,int X,int D){
      int i;
      char *rc;
      rc=L[X];
      if(rc==(char *)NULL){
            rc=DefVect(tnr+1);
            L[X]=rc;
            }
      if(D>tnr){
            for(i=1;i<tnr+1;i++){
                  if(Lm[D][i]==1){
                        rc[i]=1;
                        pset(X,i);
                        }
                  }
            }
```

```
      else{
            rc[D]=1;
            pset(X,D);
            }
      fprintf(Logi,"<BR>");
      return(1);
}

/* sõltuv kontekst. g2(A)={(X,T) | B->uXA & T in RC(B)]}  */
int g2(char **L,struct h *t,int X){
      int i;
      char *rc;
      rc=L[X];
      if(rc==(char *)NULL){
            rc=DefVect(tnr+1);
            L[X]=rc;
            }
      for(i=1;i<tnr+1;i++){
            if(RC[addr(t->NT,i)]==1){
                  rc[i]=1;
                  pset(X,i);
                  }
            }
      fprintf(Logi,"<BR>");
      return(1);
}

/* sõltuv kontekst. g3(A)={(X,T) | B->ADv & X in LC(B) & [T=D or T in
L(D)]} */
int g3(char **L,struct h *t,int D){
int i,j;
char *rc;
for(i=1;i<=nr;i++){
      if(LC[addr(t->NT,i)]==1){
            if(D<=tnr){
                  rc=L[i];
                  if(rc==(char *)NULL){
                        rc=DefVect(tnr+1);
                        L[i]=rc;
                        }
                  rc[D]=1;
                  pset(i,D);
                  }
            else{
                  for(j=1;j<tnr+1;j++){
                        if(Lm[D][j]==1){
                              rc=L[i];
                              if(rc==(char *)NULL){
                                    rc=DefVect(tnr+1);
                                    L[i]=rc;
                                    }
                              rc[j]=1;
                              pset(i,j);
                              }
                        }
                  }
            }
      }
      fprintf(Logi,"<BR>");
      return(1);
}
```

```
/* sõltuv kontekst. g4(A)={(X,T) | B->A & (X,T) in C1,1(B)}  */
int g4(char **L,struct h *t){
      char **S;
      if(t->nc==0) search_springs(t,t->NT);
      if(dc==0) return(0);
      S=DEP[t->NT];
      join_cxt(L,S);
      fflush(Logi);
      return(1);
}

/* otsib mitteterminali A sõltuva konteksti allikaid, moodustab C1,1(A) */
int search_springs(struct h *t,int A){
char **L;
struct h *C;
int i,j,k;
int Word[40];
L=DEP[A];
if(L==(char **)NULL){
      L=defL();
      DEP[A]=L;
      }
fprintf(Logi,"<BR><BR>I'll find the subsets of dependent context of
`%s'<BR>",
      T[t->NT]);
fflush(Logi);
for(i=0;i<si+1;i++){
      if(Word[i]==A){
            fprintf(Logi,"<BR>deadlock, the next will be `%s'<BR>",T[A]);
            for(j=0;j<i+1;j++){
                  k=Word[j];
                  fprintf(Logi,"  %s ",T[k]);
                  }
            fprintf(Logi,"<BR>");
            fflush(Logi);
            dc=0;
            goto out;
            }
      }
Word[si]=A;
si++;
for(i=0;i<Pnr;i++){
C=prod[i];
if(C!=(struct h *)NULL){
      for(j=0;j<C->n;j++){
            if(C->d[j]==A){
              if((j==0)&&(C->n==1)){
                fprintf(Logi,"<FONT COLOR=\"#008000\">");
                fprintf(Logi,
                   "gamma4:</FONT> the source is the production     <BR>");
                print_h(C);
                if(g4(L,C)==0) goto out;
                fflush(Logi);
                goto nextj;
                }
             if(j==0){
               fprintf(Logi,"<FONT COLOR=\"#008000\">");
               fprintf(Logi,
                  "gamma3: </FONT> the source is the production <BR>");
               print_h(C);
```

```
              g3(L,C,C->d[1]);
              goto nextj;
              }
            if((j>0)&&(j!=C->n-1)){
              fprintf(Logi,"<FONT COLOR=\"#008000\">");
              fprintf(Logi,
                  "gamma1: </FONT> the source is the production <BR>");
              print_h(C);
              g1(L,C->d[j-1],C->d[j+1]);
              goto nextj;
              }
            if(j==C->n-1){
              fprintf(Logi,"<FONT COLOR=\"#008000\">");
              fprintf(Logi,
                  "gamma2:</FONT> the source is the production <BR>");
              print_h(C);
              g2(L,C,C->d[C->n-2]);
              goto nextj;
              }
            }
            nextj: fflush(Logi);
        }
      }
}
Word[si]=0;
si--;
t->nc=1;
fprintf(Logi,"<BR>The set of dependent context of `%s':<BR>",T[A]);
print_cxt(L);
fprintf(Logi,"<BR>");
fflush(Logi);
out:  return(dc);
}

void p_nlist(){
struct U *u;
u=UP;
fprintf(Logi,"<FONT COLOR=\"#FF0000\">");
fprintf(Logi,
 "<BR>The following pairs of nonterminals are having nondifferent
context:<BR>");
fprintf(Logi,"</FONT>");
while(u!=(struct U *)NULL){
      fprintf(Logi,"`%s'",T[u->x]);
      fprintf(Logi,"<FONT COLOR=\"#FF0000\">and </FONT>");
      fprintf(Logi,"`%s'<BR>",T[u->y]);
      u=u->next;
      }
fprintf(Logi,"<BR>");
fflush(Logi);
}

void add_not(int x,int y){
      struct U *u;
      u=(struct U *)malloc(sizeof(struct U));
      u->x=x;
      u->y=y;
      u->next=UP;
      UP=u;
}
```

```
/* C1,1(A) ja C1,1(B) eristumise test */
int test_dep_con(struct h *x,struct h *y){
      int flag;
      char **A,**B;

      A=DEP[x->NT];
      B=DEP[y->NT];
      fprintf(Logi,"<BR>test_dep_con ");
      fprintf(Logi,"<FONT COLOR=\"#0000FF\"> %s </FONT> and ",T[x->NT]);
      fprintf(Logi,"<FONT COLOR=\"#0000FF\"> %s </FONT><BR>",T[y->NT]);
      flag=test_cxt(A,B);
      if(flag==0){
        fprintf(Logi,"<FONT COLOR=\"#FF0000\">");
        fprintf(Logi,
            "<BR>dependent context is not different: </FONT>%s ",T[x->NT]);
        fprintf(Logi,"<FONT COLOR=\"#FF0000\">and </FONT>");
        fprintf(Logi," %s<BR>",T[y->NT]);
        add_not(x->NT,y->NT);
        fprintf(Logi,"<HR>");
        return(0);
        }
      BRC--;
      fprintf(Logi,"<FONT COLOR=\"#008000\">");
      fprintf(Logi,"<BR>dependent context of `%s' and `%s' is
      different<BR></FONT>",T[x->NT],T[y->NT]);
      fprintf(Logi,"<HR>");
      fflush(Logi);
      return(1);
}

void print_h_lyli(struct h *s){
int i;
      fprintf(rdf," h %p: P=%d n=%d NT=%d d=%p nc=%d same=%p col=%p",
       s,s->P,s->n,s->NT,s->d,s->nc,s->same,s->col);
      fprintf(rdf,"<FONT COLOR=\"#0000FF\">  (%s)</FONT><BR>",T[s->NT]);
      fprintf(rdf," d %p: ",s->d);
      for(i=0;i<s->n;i++) fprintf(rdf,"%d ",s->d[i]);
      fprintf(rdf,"<BR>");
      fflush(rdf);
      if(s->same!=(struct h *)NULL) print_h_lyli(s->same);
      if(s->col!=(struct h *)NULL) print_h_lyli(s->col);
}

void print_h_tabel(void){
struct h *t;
int i;
      for(i=0;i<HTL;i++){
            t=HT[i];
            if(t!=(struct h *)NULL) print_h_lyli(t);
            }
}

/* tähestiku V puu tipu trükk */
void printD(struct D *t){
struct R *r;
int i;
fprintf(rdf," D %p: tunnus=%d code=%d L=%d loc=%d icx=%d left=%p right=%p
def=%p   ",
t,t->tunnus,t->code,t->L,t->loc,t->icx,t->left,t->right,t->def);
fprintf(rdf,"<FONT COLOR=\"#0000FF\">(%s)</FONT><BR>",T[t->code]);
r=t->def;
```

```
ring: if(r != (struct R *)NULL){
            fprintf(rdf," R %p: P=%d sem=%d n=%d d=%p alt=%p<BR>",
            r,r->P,r->sem,r->n,r->d,r->alt);
            fprintf(rdf," d %p: ",r->d);
            for(i=0;i<r->n;i++){
                  fprintf(rdf,"%d ",r->d[i]);
                  }
            fprintf(rdf,"<BR>");
            r=r->alt;
            goto ring;
            }
}

/* tähestiku V väljatrükk */
void print_tree(struct D *t){
      if(t->left != (struct D *)NULL) print_tree(t->left);
      printD(t);
      if(t->right != (struct D *)NULL) print_tree(t->right);
}

FILE *opw(void){
      FILE *of=NULL;
      of=fopen(rida,"wb");
      if (of==NULL){
            fclose(of);
            fprintf(Logi,"<BR>cannot create the file %s<BR>",rida);
            }
      return(of);
}

void w_V(FILE *of,struct D *t){
      struct R *d;
      struct R *prev;
      int q;
      if(t!=(struct D *)NULL){
            q=fwrite(t,sizeof(struct D),1,of);
            if(q!=1){
                  sprintf(rida,"write error w_V()<BR>");
                  ExiT(rida);
                  }
            if(t->tunnus==0){
                  d=t->def;
                  while(d!=(struct R *)NULL){
                        q=fwrite(d,sizeof(struct R),1,of);
                        if(q!=1){
                              sprintf(rida,"write error w_V()<BR>");
                              ExiT(rida);
                              }
                        q=fwrite(d->d,d->n*sizeof(int),1,of);
                        if(q!=1){
                              sprintf(rida,"write error w_V()<BR>");
                              ExiT(rida);
                              }
                        prev=d;
                        d=d->alt;
                        }
                  }
            w_V(of,t->left);
            w_V(of,t->right);
            }
}
```

```
void w_DEP(FILE *of){
      int i,j,q;
      char **L;
      char *rc;
      q=fwrite(DEP,sizeof(char ***),nr+1,of);
      if(q!=nr+1){
            sprintf(rida,"write error w_DEP()<BR>");
            ExiT(rida);
            }
      for(i=0;i<(nr+1);i++){
            if(DEP[i]!=(char **)NULL){
                  L=DEP[i];
                  q=fwrite(L,sizeof(char **),nr+1,of);
                  if(q!=nr+1){
                        sprintf(rida,"write error w_DEP()<BR>");
                        ExiT(rida);
                        }
                  for(j=0;j<nr+1;j++){
                        rc=L[j];
                        if(rc!=(char *)NULL){
                              q=fwrite(rc,tnr+1,1,of);
                              if(q!=1){
                                    sprintf(rida,"write error w_DEP()<BR>");
                                    ExiT(rida);
                                    }
                              }
                        }
                  }
            }
/*    print_all_cxt();  */
}


void w_hr(FILE *of,struct h *r){
      int q;
      q=fwrite(r,sizeof(struct h),1,of);
      if(q!=1){
            sprintf(rida,"write error w_hr()<BR>");
            ExiT(rida);
            }
      q=fwrite(r->d,r->n*sizeof(int),1,of);
      if(q!=1){
            sprintf(rida,"write error w_hr()<BR>");
            ExiT(rida);
            }
      if(r->same!=(struct h *)NULL) w_hr(of,r->same);
      if(r->col!=(struct h *)NULL) w_hr(of,r->col);
}

void w_HT(FILE *of){
      int i,q;
      q=fwrite(HT,sizeof(struct h *),HTL,of);
      if(q!=HTL){
            sprintf(rida,"write error w_HT()<BR>");
            ExiT(rida);
            }
      for(i=0;i<HTL;i++){
            if(HT[i]!=(struct h *)NULL) w_hr(of,HT[i]);
            }
}
```

```
 void w_PM(void){
      int i,j,q;
      sprintf(rida,"%s.pm",Nimi);
      of=opw();
      if(of==NULL){
            sprintf(rida,"create error w_PM()<BR>");
            ExiT(rida);
            }
      PM=DefArray(nr+1);
      for(i=1;i<=nr;i++){
            for(j=1;j<=nr;j++){
                  PM[addr(i,j)]=Pm[i][j];
                  }
            }
      q=fwrite(PM,(nr+1)*(nr+1),1,of);
      if(q!=1){
            sprintf(rida,"write error w_PM()<BR>");
            ExiT(rida);
            }
      fclose(of);
}

void p_w(struct stat *b){
      int p;
      if(stat(rida,b)==-1){
            sprintf(rida,"w_tabs: stat failed<BR>");
            ExiT(rida);
            }
      p=b->st_size;
      fprintf(Logi,"<TR><TD>%s</TD><TD
ALIGN=\"right\">%d</TD></TR>",rida,p);
}

/* TT: sorted by length terminal alphabet */
void make_TT(void){
      int i,j;
      for(i=0;i<tnr+1;i++) memcpy(TT[i],T[i],20);
      for(i=0;i<tnr;i++){
            for(j=i+1;j<tnr+1;j++){
                  if(strlen(TT[i])<strlen(TT[j])){
                        memcpy(word,TT[i],20);
                        memcpy(TT[i],TT[j],20);
                        memcpy(TT[j],word,20);
                        }
                  }
            }
}

int w_tabs(void){
      int q;
      struct stat *buf;
      fprintf(Logi,"<H3><FONT COLOR=\"#0000FF\">Result tables</H3><BR>");
      fprintf(Logi,"<TABLE BORDER=1><TR>");
      fprintf(Logi,"<TD><FONT COLOR=\"#0000FF\">File</TD><TD><FONT
COLOR=\"#0000FF\">Size</TD></FONT>");
      buf=(struct stat *)malloc(sizeof(struct stat));
      if(buf==NULL){
            sprintf(rida,"w_tabs: I haven't %d bytes of memory..<BR>",
                  sizeof(struct stat));
            ExiT(rida);
```

```
      }
PARM=(struct parm *)malloc(sizeof(struct parm));
if(PARM==(struct parm *)NULL){
      sprintf(rida,"cannot create PARM<CR>");
      ExiT(rida);
      }
memset(PARM,'\0',sizeof(struct parm));
PARM->nr=nr;
PARM->tnr=tnr;
PARM->Pnr=Pnr;
PARM->BRC=context;
PARM->dep=dc;
sprintf(rida,"%s.prm",Nimi);
of=opw();
if(of==NULL) goto bad;
q=fwrite(PARM,sizeof(struct parm),1,of);
if(q!=1){
      sprintf(rida,"write error PARM<BR>");
      ExiT(rida);
      }
fclose(of);
p_w(buf);

w_PM();
p_w(buf);

sprintf(rida,"%s.t",Nimi);
of=opw();
if(of==NULL) goto bad;
q=fwrite(&T,20,nr+1,of);
if(q!=nr+1){
      sprintf(rida,"write error w_T)<BR>");
      ExiT(rida);
      }
fclose(of);
p_w(buf);

make_TT();
sprintf(rida,"%s.tt",Nimi);
of=opw();
if(of==NULL) goto bad;
q=fwrite(&TT,20,tnr+1,of);
if(q!=tnr+1){
      sprintf(rida,"write error w_TT<BR>");
      ExiT(rida);
      }
fclose(of);
p_w(buf);

sprintf(rida,"%s.ht",Nimi);
of=opw();
if(of==NULL) goto bad;
w_HT(of);
fclose(of);
p_w(buf);

if(semantika!=(int *)NULL){
      sprintf(rida,"%s.sm",Nimi);
      of=opw();
      if(of==NULL) goto bad;
      q=fwrite(semantika,(tnr+Pnr+1)*sizeof(int),1,of);
```

```
            if(q!=1){
                  sprintf(rida,"write error w_semantics<BR>");
                  ExiT(rida);
                  }
            fclose(of);
            p_w(buf);
            }

      if(DT!=(struct D *)NULL){
            sprintf(rida,"%s.v",Nimi);
            of=opw();
            if(of==NULL) goto bad;
            w_V(of,DT);
            fclose(of);
            p_w(buf);
            }

      if(context==1){
            sprintf(rida,"%s.lc",Nimi);
            of=opw();
            if(of==NULL) goto bad;
            q=fwrite(LC,(nr+1)*(nr+1),1,of);
            if(q!=1){
                  sprintf(rida,"write error w_LC<BR>");
                  ExiT(rida);
                  }
            fclose(of);
            p_w(buf);

            sprintf(rida,"%s.rc",Nimi);
            of=opw();
            if(of==NULL) goto bad;
            q=fwrite(RC,(nr+1)*(nr+1),1,of);
            if(q!=1){
                  sprintf(rida,"write error w_RC<BR>");
                  ExiT(rida);
                  }
            fclose(of);
            p_w(buf);
            }
      if(dc==1){
            sprintf(rida,"%s.dc",Nimi);
            of=opw();
            if(of==NULL) goto bad;
            w_DEP(of);
            fclose(of);
            p_w(buf);
            }
      goto out;
bad:  sprintf(rida,"<BR>cannot write all the tables of %s.grm<BR>",Nimi);
      ExiT(rida);
out:
      fprintf(Logi,"</TABLE>");
      fflush(Logi);
      return(1);
}

char *gM(int p){
      char *ptr=NULL;
      ptr=malloc(p);
      if(ptr==NULL){
```

```
            sprintf(rida,"gM: I haven't %d bytes of memory..",p);
            ExiT(rida);
            }
      return(ptr);
}

/* tähestiku väljatrükk */
void pT(int v){
int j,i,p;
j=(v==0) ? 1 : tnr+1;
p=(v==0) ? tnr+1 : nr+1;
      for(i=j;i<p;i++){
            fprintf(Logi,"<FONT COLOR=\"008000\"> #%2d = </FONT>",i);
            if(v==0) fprintf(Logi,"%s<BR>",T[i]);
            else fprintf(Logi,"`%s'<BR>",T[i]);
            }
      fprintf(Logi,"<BR>");
      fflush(Logi);
}

/* eraldab mälu uuele tipule tähestiku V puusse */
struct D *newD(void){
      struct D *ptr;
      ptr=(struct D *)malloc(sizeof(struct D));
      if(ptr==NULL){
            sprintf(rida,"newD: I don't have memory enaugh..");
            ExiT(rida);
            }
      memset(ptr,'\0',sizeof(struct D));
      ptr->left=(struct D *)NULL;
      ptr->right=(struct D *)NULL;
      ptr->def=(struct R *)NULL;
      return(ptr);
}

/* eraldab mälu uuele NT-definitsioonile */
struct R *newR(void){
      struct R *ptr;
      ptr=(struct R *)malloc(sizeof(struct R));
      if(ptr==NULL){
            sprintf(rida,"newR: I don't have memory enaugh..");
            ExiT(rida);
            }
      memset(ptr,'\0',sizeof(struct R));
      ptr->alt=(struct R *)NULL;
      return(ptr);
}


/* vabastab NT-definitsiooni mälu */
void free_R(struct R *r){
      if(r != (struct R *)NULL){
             free_R(r->alt);
             free(r->d);
             free(r);
             }
}

/* vabastab tähestiku V puuga hõivatud mälu */
void free_tree(struct D *t){
      if(t!=(struct D *)NULL){
```

```
            free_tree(t->left);
            free_tree(t->right);
            free_R(t->def);
            free(t);
            }
}

/* mitteterminali definitsiooni(de) trükk */
void printR(struct R *r){
int i,j;
char *s;
ring: if(r != (struct R *)NULL){
            fprintf(Logi,"dl=%d<BR>",r->n);
            for(i=0;i<r->n;i++){
                  j=r->d[i];
                  s=T[j];
                  fprintf(Logi,"%s ",s);
                  }
            fprintf(Logi,"<BR>");
            r=r->alt;
            goto ring;
            }
}

/* tähestiku V elemendi otsimine/lisamine (x=1), otsimine (x=0) */
struct D *search(int x){
      struct D *t;
      int res;
      t=DT;
      if(t==(struct D *)NULL){
            if(x==0) goto exit;
            DT=DC;
            t=DC;
            goto out;
            }
ring: if(t->loc==0) res=strcmp(word,T[t->code]);
       else res=strcmp(word,T[t->loc]);
       if(res==0){
/*          free(DC);  */
            goto out;
            }
       if(res<0){
            if(t->left != (struct D *)NULL){
                  t=t->left;
                  goto ring;
                  }
             else{
                  if(x==0) goto exit;
                  t->left=DC;
                  t=DC;
                  goto out;
                  }
            }
            if(res>0){
                  if(t->right != (struct D *)NULL){
                        t=t->right;
                        goto ring;
                        }
                  else{
                        if(x==0) goto exit;
                        t->right=DC;
```

```
                        t=DC;
                        goto out;
                        }
                  }
      goto out;
exit: t=(struct D *)NULL;
//    free(DC);
out:  return(t);
}


/* tähestiku V elemendi otsimine */
struct D *getV(char *key){
      struct D *t;
      int res;
      t=DT;
      if(t==(struct D *)NULL) goto exit;
ring: if(t->loc==0) res=strcmp(key,T[t->code]);
            else res=strcmp(key,T[t->loc]);
            if(res==0) goto out;
            if(res<0){
                  if(t->left != (struct D *)NULL){
                        t=t->left;
                        goto ring;
                        }
                  else goto exit;
                  }
            if(res>0){
                  if(t->right != (struct D *)NULL){
                        t=t->right;
                        goto ring;
                        }
                  else goto exit;
                  }
exit: t=(struct D *)NULL;
out:  return(t);
}

/* tähestiku V moodustamine: veateade */
void viga(int i){
int j;
      fprintf(Logi,"error in  grammar: nr of symbol=%d
GrammarLength=%d<BR>",
            i,Glen);
      for(j=0;j<i;j++) fprintf(Logi,"%c",GBuf[j]);
      fprintf(Logi,"%c",sk);
      while(i<Glen){
            fprintf(Logi,"%c",GBuf[i]);
            i++;
            }
      fprintf(Logi,"<BR>");
}

/* produktsioonide otseadresseeriv järjestamine väljatrükiks */
void sort_rules(struct D *t,struct D *SD[],struct R *SR[]){
struct R *r;
      if(t!=(struct D *)NULL){
            if(t->tunnus==0){
                  r=t->def;
                  while(r!=(struct R *)NULL){
                        if(r->P<0||r->P>Pnr){
```

```
                                fprintf(Logi,"\ar->P=%d<BR>",r->P);
                                fflush(Logi);
                                }
                        SD[r->P]=t;
                        SR[r->P]=r;
                        r=r->alt;
                        }
                }
            sort_rules(t->left,SD,SR);
            sort_rules(t->right,SD,SR);
            }
}

/* mitteterminali ja tema definitsiooni(de) trükk - conf.c */
int print_rule(struct D *t,struct R *r){
int i,j;
char *s;
fprintf(Logi,"<FONT COLOR=\"#0000FF\">P%2d: `%s' -> ",r->P,T[t->code]);
for(i=0;i<r->n;i++){
      j=r->d[i];
      s=T[j];
      if(r->d[i]<=tnr) fprintf(Logi,"%s ",s);
      else fprintf(Logi,"`%s' ",s);
      }
fprintf(Logi,"</FONT><BR>");
return(1);
}

/* produktsiooni trükk */
void Print_rule(struct D *t,struct R *r){
int i,j;
if(r != (struct R *)NULL){
      fprintf(Logi,"<FONT COLOR=\"#008000\">P%2d: </FONT>",r->P);
      fprintf(Logi,"`%s'",T[t->code]);
      fprintf(Logi,"<FONT COLOR=\"#FF0000\"> -> </FONT>");
      for(i=0;i<r->n;i++){
         j=r->d[i];
         if(r->d[i]<=tnr)fprintf(Logi,"<FONT COLOR=\"#0000FF\">%s ",T[j]);
         else fprintf(Logi,"`%s' ",T[j]);
                  fprintf(Logi,"</FONT>");
                  }
            fprintf(Logi,"<BR>");
            }
}

/* produktsiooni trükk */
void Print_rule_comm(struct D *t,struct R *r){
int i,j;
      if(r != (struct R *)NULL){
            fprintf(Logi,"P%2d: `%s' -> ",r->P,T[t->code]);
            for(i=0;i<r->n;i++){
                  j=r->d[i];
                  if(r->d[i]<=tnr)fprintf(Logi,"%s ",T[j]);
                  else fprintf(Logi,"`%s' ",T[j]);
                  }
            fprintf(Logi,"<BR>");
            }
}

/* produktsiooni trükk faili*/
void Print_rule_f(struct D *t,struct R *r){
```

```
int i,j;
      if(r != (struct R *)NULL){
            fprintf(Sem,"  P%2d: `%s' -> ",r->P,T[t->code]);
            for(i=0;i<r->n;i++){
                  j=r->d[i];
                  if(r->d[i]<=tnr)fprintf(Sem,"%s ",T[j]);
                  else fprintf(Sem,"`%s' ",T[j]);
                  }
            fprintf(Sem,"\n");
            }
}

/* produktsioonide hulga P trükk */
int print_rules(struct D *t){
      int i;
/*    fprintf(Logi,"->print_rules<BR>");  */
      for(i=1;i<=Pnr;i++){
            SD[i]=(struct D *)NULL;
            SR[i]=(struct R *)NULL;
            }
      sort_rules(t,SD,SR);
/*    print_rules(t);  */
      for(i=1;i<=Pnr;i++){
            if(SR[i]!=NULL) Print_rule(SD[i],SR[i]);
            }
/*    fprintf(Logi,"<-print_rules<BR>");  */
      return(1);
}


/* produktsiooni 'vasaku poole' salvestamine */
struct D *leftside(int l){
      struct D *t;
      dl=0;
      DC=newD();
      DC->L=l;
      DL=DC;
      t=search(1);
      memset(def,'\0',20);
      if(t->code==0){
            nr++;
            if(nr>HTL){
                  sprintf(rida,"too many symbols: nr=%d
limes=%d<BR>",nr,HTL);
                  ExiT(rida);
                  }
            t->code=nr;
            memcpy(T[nr],word,l+1);
            }
      return(t);
}

/* mitteterminal -> definitsioon */
void nonterm(int l){
      struct D *t;
      DC=newD();
      DC->L=l;
      t=search(1);
      if(t->code==0){
            nr++;
            if(nr>HTL){
```

```
                  sprintf(rida,"too many symbols: nr=%d limes=%d<BR>",
                        nr,HTL);
                  ExiT(rida);
                  }
            t->code=nr;
            memcpy(T[nr],word,l+1);
            }
      def[dl]=t->code;
      dl++;
}

/* terminal tähestikku V (flag=0) või definitsiooni (flag=1) */
void term(int l,int flag){
      struct D *t;
      DC=newD();
      DC->tunnus=1;
      DC->L=l;
      t=search(1);
      if(t->code==0){
            tnr++;
            t->code=tnr;
            memcpy(T[tnr],word,20);
            }
      if(flag==1){
            def[dl]=t->code;
            dl++;
            }
}

/* produktsiooni moodustamine ja salvestamine */
void makerul(void){
      struct R *r;
      int i;
      r=newR();
      r->P=Pnr;
      Pnr++;
      r->n=dl;
      r->d=(int *)malloc(dl*sizeof(int));
      if(r->d==NULL){
            sprintf(rida,"makerul: I don't have memory<BR>");
            ExiT(rida);
            }
      for(i=0;i<dl;i++) r->d[i]=def[i];
      for(i=0;i<20;i++) def[i]=0;
      r->alt=DL->def;
      DL->def=r;
      dl=0;
}

/* terminaalse tähestiku koostamine */
int Terms(void){
      int i,l,mid,left;  char k;
      int res=0;
      i=0;
      left=0;
      olek=0;
      mid=0;
nstate:     memset(word,'\0',20);
      l=0;
loop: if(i>=Glen) goto out;
      if(GBuf[i]=='\t'){
```

```
            i++;
            goto loop;
            }
      switch(olek){
/* vasak */ case 0:{
                  switch(GBuf[i]){
                        case '-':{
                              olek=1;
                              mid=0;
                              goto nstate;
                              }
                        case ' ':i++;
                               goto loop;
                        case '`':{
                              if(left==1){
                                    viga(i);
                                    res=1;
                                    }
                              left=1;
                              i++;
                              goto loop;
                              }
                        case '\'':{
                              mid=0;
                              olek=1;   /* middle */
                              i++;
                              goto nstate;
                              }
                        default :{
                              i++;
                              goto loop;
                              }
                        }
                  }   /* case0 */
/* middle */      case 1:     {
                  switch(GBuf[i]){
                        case ' ': i++;
                                goto loop;
                        case '-': if(mid==0){
                                    mid=1;
                                    i++;
                                    }
                                  else{
                                    viga(i);
                                    i++;
                                    res=1;
                                    }
                                  goto loop;
                        case '>':  if(mid==0){
                                    i++;
                                    viga(i);
                                    i++;
                                    res=1;
                                    goto loop;
                                    }
                                  else{
                                    i++;
                                    olek=2;
                                    goto nstate;
                                    }
                        }
```

```
                }   /* case1 */
/* right */ case 2: {
                switch(GBuf[i]){
                      case ' ': i++;
                              goto loop;
                      case '`': {
                                    olek=3;
                                    i++;
                                    goto nstate;
                                    }
                      case '\n': left=0;
                               olek=0;
                               i++;
                               goto nstate;
                      default: k=GBuf[i];  if(isgraph(k)){
                                  olek=4;
                                  goto nstate;
                                  }
                               else{
                                  viga(i);
                                  i++;
                                  res=1;
                                  }
                               i++;
                               goto loop;
                      }
                }  /* case2 */
/* NT */    case 3: {
                k=GBuf[i]; if(isalnum(k)){
                      i++;
                      goto loop;
                      }
                else{
                      switch(GBuf[i]){
                            case '`':
                                  viga(i);
                                  i++;
                                  res=1;
                                  goto loop;
                            case '\'':
                                  i++;
                                  left=0;
                                  olek=2;
                                  goto nstate;
                            case ' ': i++;
                                    goto loop;
                            default: viga(i);
                                  i++;
                                  res=1;
                                  goto loop;
                            }
                      }
                }   /* case3 */
/* T */           case 4:{
                if(GBuf[i]=='`'){
                      term(l,0);
                      olek=2;
                      goto nstate;
                      }
                k=GBuf[i]; if(isgraph(k)){
                      word[l]=GBuf[i];
```

```
                        i++;
                        l++;
                        goto loop;
                        }
                  if(GBuf[i]==' '){
                        i++;
                        term(l,0);
                        olek=2;
                        goto nstate;
                        }
                  if(GBuf[i]=='\n'){
                        i++;
                        term(l,0);
                        olek=0;
                        left=0;
                        goto nstate;
                        }
                  }    /* case4 */
            }   /* switch olek */
out:  return(res);
}

/* mitteterminaalse tähestiku ja produktsioonide hulga koostamine */
int Rules(void){
      int i,l,mid,left,res=0;
      char k;
      i=0;
      left=0;
      mid=0;

nstate:     memset(word,'\0',20);
      l=0;
loop: if(i>=Glen) return(0);
      if(GBuf[i]=='\t'){
            i++;
            goto loop;
            }
      switch(olek){
/* vasak */ case 0:{
                  switch(GBuf[i]){
                        case '-':{
                              olek=1;
                              mid=0;
                              goto nstate;
                              }
                        case ' ':i++;
                               goto loop;
                        case '`':{
                              if(left==1){
                                    viga(i);
                                    i++;
                                    res=1;
                                    }
                              left=1;
                              i++;
                              goto loop;
                              }
                        case '\'':{
                              mid=0;
                              DL=leftside(l);
                              olek=1;   /* middle */
```

```
                              left=0;
                              i++;
                              goto nstate;
                              }
                        default :{
                              k=GBuf[i]; if(isalnum(k)){
                                    word[l]=GBuf[i];
                                    l++;
                                    i++;
                                    }
                              else{
                                    viga(i);
                                    i++;
                                    res=1;
                                    }
                              goto loop;
                              }
                        }
                  }   /* case0 */
/* middle */      case 1:     {
                  switch(GBuf[i]){
                        case ' ': i++;
                                  goto loop;
                        case '-': if(mid==0){
                                    mid=1;
                                    i++;
                                    }
                                  else{
                                    viga(i);
                                    i++;
                                    res=1;
                                    }
                                  goto loop;
                        case '>':  if(mid==0){
                                    i++;
                                    viga(i);
                                    res=1;
                                    goto loop;
                                    }
                                  else{
                                    i++;
                                    if(GBuf[i]=='\t') i++;
                                    olek=2;
                                    goto nstate;
                                    }
                        }
                  }   /* case1 */
/* right */ case 2: {
                  switch(GBuf[i]){
                        case ' ': i++;
                                  goto loop;
                        case '`': {
                                    olek=3;
                                    i++;
                                    goto nstate;
                                    }
                        case '\n': makerul();
                                  left=0;
                                  olek=0;
                                  i++;
                                  goto nstate;
```

```
                        default: k=GBuf[i];  if(isgraph(k)){
                                    olek=4;
                                    goto nstate;
                                    }
                                 else{
                                    viga(i);
                                    i++;
                                    res=1;
                                    }
                                 i++;
                                 goto loop;
                        }
                  }  /* case2 */
/* NT */    case 3: {
                  k=GBuf[i]; if(isalnum(k)){
                        word[l]=GBuf[i];
                        i++;
                        l++;
                        goto loop;
                        }
                  else{
                        switch(GBuf[i]){
                              case '`':
                                    viga(i);
                                    i++;
                                    res=1;
                                    goto loop;
                              case '\'':
                                    nonterm(l);
                                    i++;
                                    left=0;
                                    olek=2;
                                    goto nstate;
                              case ' ': i++;
                                      goto loop;
                              default: viga(i);
                                    i++;
                                    res=1;
                                    goto loop;
                              }
                        }
                  }   /* case3 */
/* T */           case 4:{
                  if(GBuf[i]=='`'){
                        term(l,1);
                        olek=2;
                        goto nstate;
                        }
                  k=GBuf[i]; if(isgraph(k)){
                        word[l]=GBuf[i];
                        i++;
                        l++;
                        goto loop;
                        }
                  if(GBuf[i]==' '){
                        i++;
                        term(l,1);
                        olek=2;
                        goto nstate;
                        }
                  if(GBuf[i]=='\n'){
```

```
                        i++;
                        term(l,1);
                        makerul();
                        olek=0;
                        left=0;
                        goto nstate;
                        }
                  }    /* case4 */
            }   /* switch olek */
      return(res);
}

void print_def(struct R *r){
int i,j;
      for(i=0;i<r->n;i++){
            j=r->d[i];
            if(j<=tnr) fprintf(Logi,"%s ",T[j]);
            else fprintf(Logi,"`%s' ",T[j]);
            }
      fprintf(Logi,"<BR>");
}

int make_sem(char *B){
      int i=0;
      int p,ix,kood,piir;
      piir=tnr+Pnr+1;
      i=0;
      p=0;
      if(B[i]=='$'){
            i++;
            end: while(B[i]!='\n') i++;
            i++;
            return(i);
            }
      if((B[i]=='P')||(B[i]=='p')){
            i++;
            p=tnr;
            }
      i+=sscanf(B+i,"%d",&ix);
      p=p+ix;
      while(B[i]!='=') i++;
      i++;
      i+=sscanf(B+i,"%d",&kood);
      if(p>=piir){
            fprintf(Logi,"p=%d piir=%d<BR>",p,piir);
            goto end;
            }
/*    fprintf(Logi,"p=%d kood=%d<BR>",p,kood);  fflush(Logi);  */
      semantika[p]=kood;
      goto end;
}

/* semantikamassiivi täitmine */
int put_semant(void){
      char *B;
      int i,piir;
      char finame[256];
      piir=tnr+Pnr+1;
      semantika=(int *)malloc((piir)*sizeof(int));
      if(semantika==NULL){
            sprintf(rida,"No memory for semantics: l=%d",piir);
```

```
            ExiT(rida);
            }
      for(i=0;i<piir;i++) semantika[i]=0;
      memset(finame,'\0',256);
      for(i=0;i<256;i++){
            if((fname[i]!='.')&&(fname[i]!='\0')) finame[i]=fname[i];
            }
      strcat(finame,".sem");
      fprintf(Logi,"Semantics file is %s<BR>",finame); fflush(Logi);
      Sem=fopen(finame, "r");
      if(Sem!=NULL) goto readit;

      fprintf(Logi,"semantics file %s is lack<BR>",finame);
      fprintf(Logi,"I'll make it myself<BR>");
      fflush(Logi);
      Sem=fopen(finame,"w");
      if(Sem==NULL){
            sprintf(rida,"cannot create file %s",finame);
            ExiT(rida);
            }
      for(i=1;i<piir-1;i++){
            semantika[i]=i;
            if(i<=tnr) fprintf(Sem,"%d=%d <SPACE>$ %s\n",i,i,T[i]);
            else{
                  fprintf(Sem,"P%d=%d <SPACE>$ ",i-tnr,i);
                  Print_rule_f(SD[i-tnr],SR[i-tnr]);
                  }
            }
      fclose(Sem);
      goto showit;

readit:
      fclose(Sem);
      B=r_text(finame);
      i=0;
      while(i<P_length){
            i+=make_sem(B+i);
            }
      fflush(Logi);

showit:
      sort_rules(DT,SD,SR);
      for(i=1;i<piir;i++){
            if(semantika[i]>0){
                  if(i<=tnr) fprintf(Logi,"%s=%d<BR>",T[i],semantika[i]);
                  else{
                        fprintf(Logi,"P%d=%d<SPACE> <SPACE> ",i-
tnr,semantika[i]);
                        fprintf(Logi,"<FONT COLOR=\"#008000\"> $");
                        Print_rule_comm(SD[i-tnr],SR[i-tnr]);
                        fprintf(Logi,"</FONT>");
                        }
                  fflush(Logi);
                  }
            }
      return(1);
}

void make_prod(void){
      int i,j;
      struct h *C,*D;
```

```
      j=0;
      for(i=0;i<HTL;i++){
            if(HT[i]!=(struct h *)NULL){
                  C=HT[i];
                  coll: D=C;
                  prod[j]=C;
                  j++;
                  same: if(D->same!=(struct h *)NULL){
                        D=D->same;
                        prod[j]=D;
                        j++;
                        goto same;
                        }
                  if(C->col!=(struct h *)NULL){
                        C=C->col;
                        goto coll;
                        }
                  }
            }
}

int constr(void){
      int pre,i;
      char *Ti;
/* konstruktori initsialiseerimine */
      int ret=1;
      olek=0;
      Pnr=1;
      Index=1;      /* eelnevusteisenduste uute produktsioonide sufiks */
      DT=(struct D *)NULL;     /* first   */
      DC=(struct D *)NULL;     /* current */
      DL=(struct D *)NULL;     /* last    */
      DN=(struct D *)NULL;     /* new     */
      context=0;               /* independent & */
      dc=0;                    /* dependent context */
      tnr=0;  /* terminaalse tähestiku jooksev pikkus */
      nr=0;   /* tähestiku V jooksev pikkus */
      NR=0;   /* nr eelnevusteisenduste ajal */
      BRC=0;  /* ühesuguste paremate pooltega produktsioonide arv */
      rules=NULL;
      for(i=0;i<(2*HTL);i++){
            Ti=T[i];
            memset(Ti,'\0',20);
            }
/* terminaalse tähestiku moodustamine */
      if(Terms()==1){
            ret=0;
            goto bye;
            }
      nr=tnr;
      fprintf(Logi,"<H4><FONT COLOR=\"#0000FF\">Terminal
alphabet</H4></FONT>");
      pT(0);
      fprintf(Logi,"<HR>");
/* mitteterminaalse tähestiku ja produktsioonide hulga moodustamine */
      if(Rules()==1){
            ret=0;
            goto bye;
            }
      fprintf(Logi,"<H4><FONT COLOR=\"#0000FF\">Nonterminal
alphabet</H4></FONT>");
```

```
      pT(1);
      fprintf(Logi,"<HR>");
      fprintf(Logi,"<H4><FONT COLOR=\"#0000FF\">Productions</H4></FONT>");
      print_rules(DT);
      fprintf(Logi,"<HR>");
/* left- ja rightmost-hulkade moodustamine, eelnevusmaatriksi koostamine */
new:
      memset(Pm,'\0',HTL*HTL);
      memset(Lm,'\0',HTL*HTL);
      memset(Rm,'\0',HTL*HTL);
      make_LRB();
      fprintf(Logi,"<H4><FONT COLOR=\"#0000FF\">Leftmost-set</H4></FONT>");
      print_LRM(Lm);
      fprintf(Logi,"<HR>");
      fprintf(Logi,"<H4><FONT COLOR=\"#0000FF\">Rightmost-
set</H4></FONT>");
      print_LRM(Rm);
      fprintf(Logi,"<HR>");
      p_all_lr();
      fprintf(Logi,"<HR>");
      pre=make_PM();
      fprintf(Logi,"<H4><FONT COLOR=\"#0000FF\">Precedence
matrix</H4></FONT>");
      print_PM();
      fprintf(Logi,"<HR>");
      print_rela();
      fprintf(Logi,"<HR>");
      fflush(Logi);
/*    if(Nimi[0]=='G') print_PM( );
      else print_rela( ); */
      if(pre==1) goto on;
/* eelnevustest ja eelnevusteisendused */
      fprintf(Logi,"<H4><FONT COLOR=\"#0000FF\">Precedence
varies</H4></FONT>");
      if(conflicts()>0){
            fprintf(Logi,"<H4><FONT COLOR=\"#0000FF\">New
grammar</H4></FONT>");
            print_rules(DT);
            fprintf(Logi,"<HR>");
            goto new;
            }
on:
      fprintf(Logi,"<H4><FONT COLOR=\"#008000\">");
      fprintf(Logi,"<H3>Grammar %s is a precedence
grammar</H3></FONT>",fname);
/* produktsioonide 'paremate poolte' paisktabeli koostamine, pööratavuse
test */
      HashRules(DT);
      DepV=DefVect(Pnr);
      if(BRC==0){
            context=0;
            fprintf(Logi,"<H4><FONT COLOR=\"#008000\">");
            fprintf(Logi,"<H4>Grammar %s is invertible</H4></FONT>",fname);
            goto semm;
            }
      fprintf(Logi,"<H4><FONT COLOR=\"#FF0000\">");
      fprintf(Logi,"<H4>Grammar %s is not invertible</H4></FONT>",fname);
      context=1;
/* sõltumatu konteksti hulkade moodustamine, BRC(1|1)-test */
      fprintf(Logi,"<HR>");
      if(LC!=(char *)NULL) free(LC);
```

```
      LC=DefArray(nr+1);
      if(RC!=(char *)NULL) free(RC);
      RC=DefArray(nr+1);
      make_all_indep();
      p_all_brc();
      fflush(Logi);
      for(i=0;i<Pnr;i++) prod[i]=(struct h *)NULL;
      make_prod();
      dc=0;
      if(make_BRC()==0) goto sos;
      if(BRC==0&&dep==0){
            fprintf(Logi,"<H4><FONT COLOR=\"#008000\">");
            fprintf(Logi,"<H4>Grammar %s is BRC(1|1)-
reducible</H4></FONT>",fname);
            goto semm;
            }
      fflush(Logi);
      if(dc==1){
            fprintf(Logi,"<H4><FONT COLOR=\"#008000\">");
            fprintf(Logi,"<H4>Grammar %s is BRC(1,1)-
reducible</H4></FONT>",fname);
            fflush(Logi);
            del_dep();
            print_all_cxt();
            goto semm;
            }
sos:
      fprintf(Logi,"<H4><FONT COLOR=\"#FF0000\">");
      fprintf(Logi,"Grammar %s is not BRC-reducible</H4></FONT>",fname);
      p_nlist( );
/*    print_all_cxt( );  */
      ret=0;
      goto bye;
/* semantika omistamine */
semm:
      fprintf(Logi,"<HR>");
      fprintf(Logi,"<H4><FONT COLOR=\"#0000FF\">");
      fprintf(Logi,"<H4>Semantics</H4></FONT>");
      fflush(Logi);
      if(semantika!=(int *)NULL){
            free(semantika);
            semantika=(int *)NULL;
            }
      put_semant( );
      Sn=nr;
      fprintf(Logi,"<HR>");
      ret=w_tabs( );
      fprintf(Logi,"<HR>");
      if(rdf!=NULL) make_rtf();
      fprintf(Logi,"Look at result <A HREF=\"%s\">tables</A>",RD);
bye:
      return(ret);
}

void make_rtf(void){
int i;
      fprintf(rdf,"<HTML><HEAD><TITLE>");
      fprintf(rdf,"TABLES %s",fname);
      fprintf(rdf,
            "</TITLE></HEAD><BODY><H1><FONT COLOR=\"#0000FF\">CONSTRUCTOR
            TABLES of the Grammar %s</H1>",fname);
```

```
      fprintf(rdf,"<BODY LINK=\"blue\"><BODY VLINK=\"blue\"><BODY
      ALINK=\"red\"></FONT><B>");
      fflush(rdf);
      fprintf(rdf,"<UL><LI><A HREF=\"#parm\">Parameters</A></LI>");
      fprintf(rdf,"<LI><A HREF=\"#t\">Alphabet V</A></LI>");
      fprintf(rdf,"<LI><A HREF=\"#tt\">Scanner Table</A></LI>");
      fprintf(rdf,"<LI><A HREF=\"#v\">Tree of the Alphabet V</A></LI>");
      fprintf(rdf,"<LI><A HREF=\"#ht\">Reduce Table</A></LI>");
      fprintf(rdf,"<LI><A HREF=\"#pm\">Precedence Matrix</A></LI>");
      if(PARM->BRC>0){
            fprintf(rdf,"<LI><A HREF=\"#lc\">Left Context</A></LI>");
            fprintf(rdf,"<LI><A HREF=\"#rc\">Right Context</A></LI>");
            }
      if(PARM->dep>0)
            fprintf(rdf,"<LI><A HREF=\"#dc\">Dependent Context</A></LI>");

      fprintf(rdf,"<LI><A HREF=\"#sm\">Semantics</A></LI>");
      fprintf(rdf,"</UL>");
      fprintf(rdf,"<HR>");
      fprintf(rdf,"<P><A NAME=\"parm\"></A>");
      fprintf(rdf,"<H2>Parameters</H2>");
      fprintf(rdf,"File %s.prm<BR>",Nimi);
      fprintf(rdf,"<PRE>");
      fprintf(rdf,"struct parm{\n");
      fprintf(rdf,"\tint nr   //tähestiku V pikkus\n");
      fprintf(rdf,"\tint tnr  //tähestiku V<SUB>T</SUB> pikkus\n");
      fprintf(rdf,"\tint BRC  //0: G on pööratav\n");
      fprintf(rdf,"\tint Pnr  //Produktsioonide arv\n");
      fprintf(rdf,"\tint dep  //1: sõltuv kontekst\n");
      fprintf(rdf,"\tint itl  //identifikaatorite arv (Parser)\n");
      fprintf(rdf,"\tint ktl  //konstantide arv (Parser)\n");
      fprintf(rdf,"}\n\n");
      fprintf(rdf,"nr=%d tnr=%d BRC=%d Pnr=%d dep=%d itl=%d ktl=%d\n\n",
            PARM->nr,PARM->tnr,PARM->BRC,PARM->Pnr,PARM->dep,PARM->itl,
            PARM->ktl);
      fprintf(rdf,"</P></PRE><A NAME=\"t\"></A><HR>");
      fprintf(rdf,"<H2>Alphabet V</H2>");
      fprintf(rdf,"File %s.t<BR>",Nimi);
      fprintf(rdf,"<PRE>");
      for(i=1;i<=nr;i++) fprintf(rdf,"%s ",T[i]);
      fprintf(rdf,"</PRE><BR><BR><HR>");
      fprintf(rdf,"</P></PRE><A NAME=\"tt\"></A><HR>");
      fprintf(rdf,"<H2>Scanner Table</H2>");
      fprintf(rdf,"File %s.tt<BR>",Nimi);
      fprintf(rdf,"<PRE>");
      for(i=0;i<=tnr;i++) fprintf(rdf,"%s ",TT[i]);
      fprintf(rdf,"</PRE><BR><BR><HR>");
      fprintf(rdf,"<A NAME=\"v\"></A>");
      fprintf(rdf,"<H2>Tree of the Alphabet V</H2>");
      fprintf(rdf,"File %s.v<BR>",Nimi);
/* Produktsiooni parem pool */
      fprintf(rdf,"<PRE>");
      fprintf(rdf,"struct R{<BR>");
      fprintf(rdf,"\tint P;         /* produktsiooni jrk-nr */<BR>");
      fprintf(rdf,"\tint sem;       /* semantikakood */<BR>");
      fprintf(rdf,"\tint n;         /* NT definitsiooni pikkus */<BR>");
      fprintf(rdf,"\tint *d;        /* NT definitsioon */<BR>");
      fprintf(rdf,"\tstruct R *alt; /* NT alternatiivne definitsioon */
            <BR>");
      fprintf(rdf,"}<BR><BR>");
```

```
/* Tähestiku V sümboli (NT või T) kirje */
      fprintf(rdf,"struct D{<BR>");
      fprintf(rdf,"\tint tunnus;  /* 0: NT, 1: T */<BR>");
      fprintf(rdf,"\tint code;    /* vahekeelne kood */\n");
      fprintf(rdf,"\tint L;       /* NT või id/const nime pikkus  */\n");
      fprintf(rdf,"\tint loc;     /* ident | const : nr */\n");
      fprintf(rdf,"\tint icx;     /* itl või ktl - IT või KT index */\n");
      fprintf(rdf,"\tstruct D *left;   /* kahendpuu viidad (võti on
T[code]) */\n");
      fprintf(rdf,"\tstruct D *right;\n");
      fprintf(rdf,"\tstruct R *def;\n");
      fprintf(rdf,"}\n\n");
      print_tree(DT);
      fprintf(rdf,"</PRE><HR>");
      fprintf(rdf,"<A NAME=\"ht\"></A>");
      fprintf(rdf,"<H2>Reduce Table</H2>");
      fprintf(rdf,"File %s.ht<BR>",Nimi);
      fprintf(rdf,"<PRE>");
      fprintf(rdf,"struct h{\n");
      fprintf(rdf,"\tint P;          /* produktsiooni jrk-nr */\n");
      fprintf(rdf,"\tint n;          /* definitsiooni pikkus  */\n");
      fprintf(rdf,"\tint NT;         /* defineeritav nonterminal  */\n");
      fprintf(rdf,"\tint *d;         /* definitsioon */\n");
      fprintf(rdf,"\tint nc;         /* 1 - sõltuv kontekst */\n");
      fprintf(rdf,"\tchar *c;        /* reservis */\n");
      fprintf(rdf,"\tstruct h *same; /* sama parem pool */\n");
      fprintf(rdf,"\tstruct h *col;  /* põrkeviit */\n");
      fprintf(rdf,"}\n\n");
      print_h_tabel( );
      fprintf(rdf,"</PRE><HR>");
      fprintf(rdf,"<A NAME=\"pm\"></A>");
      fprintf(rdf,"<H2>Precedence Matrix</H2>");
      fprintf(rdf,"File %s.pm<BR>",Nimi);
      print_rPM( );
      fprintf(rdf,"<BR><BR><HR>");
if(PARM->BRC>0){
      fprintf(rdf,"<A NAME=\"lc\"></A>");
      fprintf(rdf,"<H2>Left Context</H2>");
      fprintf(rdf,"File %s.lc<BR>",Nimi);
      print_rLC( );
      fprintf(rdf,"<BR><BR><HR>");
      fprintf(rdf,"<A NAME=\"rc\"></A>");
      fprintf(rdf,"<H2>Right Context</H2>");
      fprintf(rdf,"File %s.rc<BR>",Nimi);
      print_rRC( );
      fprintf(rdf,"<BR><BR><HR>");
      }
if(PARM->dep>0){
      fprintf(rdf,"<A NAME=\"dc\"></A>");
      fprintf(rdf,"<H2>Dependent Context</H2>");
      fprintf(rdf,"File %s.dc<BR>",Nimi);
      print_DEP( );
      fprintf(rdf,"<BR><BR><HR>");
      }
      fprintf(rdf,"<A NAME=\"sm\"></A>");
      fprintf(rdf,"<H2>Semantics</H2>");
      fprintf(rdf,"File %s.sm<BR><BR>",Nimi);
      for(i=1;i<=tnr+Pnr;i++) fprintf(rdf,"%d ",semantika[i]);
      fprintf(rdf,"</BODY></HTML>");
      fflush(rdf);
      fclose(rdf);
```

```
}

int construc(void){
      int ret=0;
      time_t t0,t1;
      GBuf=NULL;
      Logi=fopen(L_name,"w");
      if(Logi==NULL){
            printf("Cannot open log-book<BR>");
            return(0);
            }
      fprintf(Logi,"<HTML><HEAD><TITLE>");
      time(&t0);
      fprintf(Logi,"CONSTRUCTOR %s",fname);
      fprintf(Logi,
        "</TITLE></HEAD><BODY><H2>Start of CONSTRUCTOR for the Grammar %s
        %s</H2>",fname,asctime(localtime(&t0)));
      fprintf(Logi,"</FONT><B>");
      if(gramm( )==0) goto jokk;
      ret=constr( );

jokk:
      time(&t1);
      fprintf(Logi,"<FONT COLOR=\"#0000FF\">");
      fprintf(Logi,"<H4>Finish of CONSTRUCTOR
%s</H4>",asctime(localtime(&t1)));
      fprintf(Logi,"</FONT></BODY></HTML>");
      fflush(Logi);
      fclose(Logi);
      return(ret);
}

int main(int argc,char **argv){
      if(argc!=2){
            printf("I do need only the name of a grammar<BR>"); abort();
            }
      fname=(char *)malloc(256);
      Nimi=(char *)malloc(9);
      L_name=(char *)malloc(256);
      memset(RD,'\0',256);
      memset(fname,'\0',256);
      memset(Nimi,'\0',9);
      memset(L_name,'\0',256);
      sprintf(fname,"%s",argv[1]);
      sprintf(L_name,"%s",argv[1]);
      sprintf(RD,"%srd.htm",argv[1]);
      rdf=fopen(RD,"w");
      strcpy(Nimi,fname);
      strcat(fname,".grm");
      strcat(L_name,".htm");
      printf("fname=%s Nimi=%s L_name=%s\n",fname,Nimi,L_name);
      construc( );
      getchar( );
      return(1);
}
```

Lisa 7. Analüsaator

```
/* parser32.c  May 2001 */
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
#include <sys/stat.h>
#include "two32.h"

int NR;
FILE *rdf=NULL;

void ExiT(char *t){
      fprintf(Logi,"%s<BR>",t);
      fprintf(Logi,"</BODY></HTML>");
      fflush(Logi);
      fclose(Logi);
      printf("%s",t);
      getchar( );
      abort( );
}

void blue(char *t){
      fprintf(Logi,"<FONT COLOR=\"0000FF\">%s</FONT>",t);
}

void green(char *t){
      fprintf(Logi,"<FONT COLOR=\"008000\">%s</FONT>",t);
}

void red(char *t){
      fprintf(Logi,"<FONT COLOR=\"FF0000\">%s</FONT>",t);
}

void pset(int x,int y){
      blue("{");
      fprintf(Logi,"%s",T[x]);
      blue(",");
      fprintf(Logi,"%s",T[y]);
      blue("}");
}

void print_cxt(char **L){
      int i,j;
      char *rc;
      fprintf(Logi,"cxt: L=%p<BR>",L); fflush(Logi);
      for(i=tnr+1;i<nr+1;i++){
            rc=L[i];
            if(rc!=(char *)NULL){
                  fprintf(Logi,"cxt: nr=%d i=%d rc=%p<BR>",nr,i,rc);
                  fflush(Logi);
                  for(j=0;j<tnr+1;j++){
                        if(rc[j]==1) pset(i,j);
                        }
                  }
            }
```

```
      fprintf(Logi,"<BR>"); fflush(Logi);
}

void print_cxtp(char **L){
int i;
      char *rc;
      for(i=1;i<nr+1;i++){
            rc=L[i];
            if(rc!=(char *)NULL) fprintf(Logi,"L[%d]=%p<BR>",i,rc);
            }
}

void print_DEP(void){
char **L;
char *rc;
int i,j,k;
      for(i=tnr+1;i<nr+1;i++){
            L=DEP[i];
            if(L!=(char **)NULL){
                  fprintf(Logi,"<BR>DEP[%d]=%p<BR>",i,L);
                  for(j=0;j<tnr+1;j++){
                        rc=L[j];
                        if(rc!=(char *)NULL){
                              fprintf(Logi,"L[%d]=%p<BR>",j,rc);
                              for(k=0;k<tnr+1;k++)
                                 fprintf(Logi,"rc[%d]=%d ",k,rc[k]);
                              fprintf(Logi,"<BR>");
                              }
                        }
                  }
            }
}

void print_all_cxt(void){
char **L;
int i;
      for(i=tnr+1;i<nr+1;i++){
            L=DEP[i];
            if(L!=(char **)NULL){
                  fprintf(Logi,"<BR>dependent context of %s:<BR>",T[i]);
                  print_cxt(L);
                  fprintf(Logi,"<BR>");
                  }
            }
}

void print_all_cxtp(void){
char **L;
int i;
      for(i=tnr+1;i<nr+1;i++){
            L=DEP[i];
            if(L!=(char **)NULL){
                  fprintf(Logi,"DEP[%d]=%p<BR>",i,L);
                  print_cxtp(L);
                  }
            }
}

void print_L(struct h *CP){
int i,j;
char *r;
```

```
	fprintf(Logi,"<BR>L: P%2d `%s' -&gt ",CP->P,T[CP->NT]);
	for(j=0;j<CP->n;j++){
		i=CP->d[j];
		r=T[i];
		if(i<tnr+1) fprintf(Logi,"%s ",r);
		else fprintf(Logi,"`%s' ",r);
		}
	fprintf(Logi,"<BR>");
	fflush(Logi);
}

void print_h(struct h *t){
	int i,j;
	char *r;
	fprintf(Logi,"<BR>P=%d `%s' -&gt ",t->P,T[t->NT]);
	for(i=0;i<t->n;i++){
		j=t->d[i];
		r=T[j];
		if(j<=tnr) fprintf(Logi,"%s ",r);
		else fprintf(Logi,"`%s' ",r);
		}
	fprintf(Logi,"<BR>");
	fflush(Logi);
}

void print_top(struct top *t){
fprintf(Logi,
"top %p: kood=%d leks=%d sem=%d label=%d truel=%d falsel=%d up=%p right=%p
down=%p",t,t->kood,t->leks,t->sem,t->label,t->truel,t->falsel,t->up,
t->right,t->down);
fprintf(Logi,"  (%s)<BR>",(t->leks==0) ? T[t->kood] : T[t->leks]);
if(t->right!=(struct top *)NULL) print_top(t->right);
if(t->down!=(struct top *)NULL) print_top(t->down);
}

void print_Itr(struct itr *t){
fprintf(Logi," itr %p: nr=%d loc=%d value=%d top=%p io=%d",
		t,t->nr,t->loc,t->value,t->t,t->io);
fprintf(Logi,"   (%s)<BR>",T[t->nr]);
}

void print_Ctr(struct ctr *t){
fprintf(Logi," ctr %p: nr=%d loc=%d value=%d",   t,t->nr,t->loc,t->value);
fprintf(Logi,"   (%s)<BR>",T[t->nr]);
}

void print_one_top(struct top *t){
fprintf(Logi," top %p: kood=%d leks=%d sem=%d label=%d truel=%d falsel=%d
up=%p right=%p down=%p", t,t->kood,t->leks,t->sem,t->label,t->truel,
t->falsel,t->up,t->right, t->down);
fprintf(Logi,"  (%s)<BR>",(t->leks==0) ? T[t->kood] : T[t->leks]);
}

/* analüüsi puu tipu trükk */
void p_top(char *t,struct top *p){
if(logi==1){
	fprintf(Logi,"%s top:<BR>",t);
	print_one_top(p);
	}
}
```

```
// adresseerib maatrikseid programmi analüüsi ajal. Sn=nr-(ident & con arv)
/* scanner lisab id. ja c-d V-sse, iga puhul nr++ */
int adr(int i, int j){
      int k;
      k=(i-1)*Sn+j;
      if(k>(Sn+1)*(Sn+1)){
            fprintf(Logi,"adr: indeks %d  üle piiri!<BR>",k);
            k=0;
            }
      return(k);
}

void set_show(char *title){
      fprintf(Logi,"<H4><FONT COLOR=\"#0000FF\">%s",title);
      fprintf(Logi,"</FONT></H4>");
}

char *DefArray(int d){
      int n,i;
      char *M;
      n=(d*d);
      M=(char *)malloc(n);
      if(M==(char *)NULL){
            fprintf(Logi,"DefArray: I don't have memory enough..<BR>");
            ExIT();
            }
      for(i=0;i<n;i++) M[i]='\0';
      return(M);
}

/* vektoritele mälu eraldamine */
char *DefVect(int n){
      int i;
      char *M;
      M=(char *)malloc(n);
      if(M==(char *)NULL){
            fprintf(Logi,"I don't have memory enough..");
            ExIT();
            }
      for(i=0;i<n;i++) M[i]='\0';
      return(M);
}

/* sõltuva konteksti puu ülemine tase: mõisted */
char ***defDEP(void){
      int i,n;
      n=(nr+1)*sizeof(char ***);
      DEP=(char ***)malloc(n);
      if(DEP==(char ***)NULL){
            fprintf(Logi,"I don't have memory enough..");
            ExIT( );
            }
      for(i=0;i<nr+1;i++) DEP[i]=(char **)NULL;
      return(DEP);
}

/* sõltuva konteksti puu vahetase: vasakud kontekstid */
char **defL(void){
      int i,n;
      char **L;
      n=(nr+1)*sizeof(char **);
```

```
      L=(char **)malloc(n);
      if(L==(char **)NULL){
            fprintf(Logi,"I don't have memory enough..");
            ExIT( );
            }
      for(i=0;i<nr+1;i++) L[i]=(char *)NULL;
      return(L);
}

void ps(void){
      fprintf(Logi,"<BR>");
}

void ExIT(void){
      if(logi==1){
            fprintf(Logi,"Abnormal end<BR><BR>");
            fflush(Logi);
            fclose(Logi);
            }
      abort( );
}

/* paiskfunktsioon (võti on prod. vahekeelne parem pool) */
int hfunc(int keylen,int *key){
      int h=0,g;
      int i;
      for(i=0;i<keylen;i++) h=h^key[i];
      g=h;
      h=h/HTL;
      h=g-(h*HTL);
      if((h<0)||(h>HTL)){
            fprintf(Logi,"HT error!<BR>");
            ExIT( );
            }
      return(h);
}

/* produktsiooni otsimine paisktabelist */
struct h *ReadHash(int n,int *key){
      struct h *t;
      int i,a;
      a=hfunc(n,key);
      t=HT[a];
otsi:
      if(t!=(struct h *)NULL){
            if(n==t->n){
                  for(i=0;i<n;i++){
                        if(key[i]!=t->d[i]) goto next;
                        }
                  return(t);
                  }
            next: t=t->col;
            goto otsi;
            }
      return(NULL);
}

int p_viga(int i){
      int j;
      VEAD++;
      fprintf(Logi,"<BR>\rerror at symbol #%d %s<BR>",i,word);
```

```
      for(j=0;j<i;j++) fprintf(Logi,"%c",PBuf[j]);
      if(logi==1) fprintf(Logi,"%c",sk);
      while(i<Plen){
            fprintf(Logi,"%c",PBuf[i]);
            i++;
            }
      ps( );
      return(1);
}

/* parser: lähte- ja vahekeelse programmi trükk */
void print_VK(int k){
int t,i,j;
      set_show("Input program:");
      for(i=0;i<k;i++){
            j=VK[i];
            if(j!=m_k){
                  if(j!=k_k) fprintf(Logi,"%s ",T[j]);
                  }
            }
      fprintf(Logi,"<BR>");
      set_show("Scanned program:");
      for(i=0;i<k;i++) fprintf(Logi,"%d ",VK[i]);
      fprintf(Logi,"<BR>");
      if(itl>0){
            fprintf(Logi,"<BR><FONT
COLOR=\"#0000FF\">Identifiers:</FONT>");
            for(i=0;i<itl;i++){
                  t=IT[i];
                  fprintf(Logi,"%s%s",T[t],(i<itl-1) ? ", " : ";");
                  }
            ps( );
            }
      if(ktl>0){
            fprintf(Logi,"<BR><FONT COLOR=\"#0000FF\">Constants:</FONT>");
            for(i=0;i<ktl;i++){
                  t=KT[i];
                  fprintf(Logi,"%s%s",T[t],(i<ktl-1) ? ", " : ";");
                  }
            ps( );
            }
}

/* eraldab mälu uuele tipule tähestiku V puusse */
struct D *newD(void){
      struct D *ptr;
      ptr=(struct D *)malloc(sizeof(struct D));
      if(ptr==NULL) ExIT();
      memset(ptr,'\0',sizeof(struct D));
      ptr->left=(struct D *)NULL;
      ptr->right=(struct D *)NULL;
      ptr->def=(struct R *)NULL;
      return(ptr);
}

/* tähestiku V elemendi otsimine */
struct D *getV(char *key){
      struct D *t;
      int res;
      t=DT;
      if(t==(struct D *)NULL) goto exit;
```

```
ring: if(t->loc==0) res=strcmp(key,T[t->code]);
       else{
            res=strcmp(key,T[t->loc]);
            fprintf(Logi,"key=%s t->loc=%d %s<BR>",key,t->code,T[t->loc]);
            }
       if(res==0) goto out;
       if(res<0){
                  if(t->left != (struct D *)NULL){
                        t=t->left;
                        goto ring;
                        }
                  else goto exit;
                  }
            if(res>0){
                  if(t->right != (struct D *)NULL){
                        t=t->right;
                        goto ring;
                        }
                  else goto exit;
                  }
exit: t=(struct D *)NULL;
out:
      return(t);
}

/* scanner: identifikaatori töötlemine */
int Ident(int k){
      int t,i;
      char *id;
      for(i=0;i<itl;i++){
            t=IT[i];
            id=T[t];
            if(strcmp(id,word)==0) goto itis;
            }
      nr++;
      memcpy(T[nr],word,20);
      VK[k]=m_k;
      IT[itl]=nr;
      itl++;
      k++;
      VK[k]=nr;
      k++;
      goto ok;
itis:
      VK[k]=m_k;
      k++;
      VK[k]=t;
      k++;
ok:   memset(word,'\0',20);
      return(k);
}

/* scanner: konstandi töötlemine */
int Const(int k){
      int t,i;
      char *con;
      for(i=0;i<ktl;i++){
            t=KT[i];
            con=T[t];
            if(strcmp(con,word)==0) goto itis;
            }
```

```
      nr++;
      memcpy(T[nr],word,20);
      VK[k]=k_k;
      KT[ktl]=nr;
      ktl++;
      k++;
      VK[k]=nr;
      k++;
      goto ok;
itis:
      VK[k]=k_k;
      k++;
      VK[k]=t;
      k++;
ok:   memset(word,'\0',20);
      return(k);
}

int lekseem(int i){
      char *b,*g;
      struct D *t;
      int a,j,s,n;
      b=PBuf+i;
      for(s=0;s<tnr+1;s++){
            g=TT[s];
            n=strlen(g);
            if(n==0) return(0);
            j=0;
            memset(word,'\0',20);
            for(a=0;a<n;a++){
                  if(b[a]!=g[a]) goto next;
                  word[j]=b[j];
                  j++;
                  }
            i=i+j;
            t=getV(word);
            if(t==NULL) ExIT();
            KOOD=t->code;
            return(j);
            next: j=0;
            }
      return(0);
}

/* teisendab lähteprogrammi vahekeelseks */
int scanner(void){
      struct D *t;
      int i,j,k,r;
      int olek;
      j=0;
      itl=ktl=0;
      m_k=k_k=0;
      VEAD=0;
      t=getV("#i#");
      if(t!=(struct D *)NULL) m_k=t->code;
      t=getV("#c#");
      if(t!=(struct D *)NULL) k_k=t->code;
      vkl=Plen*8;
      VK=(int *)malloc(vkl);
      if(VK==NULL){
            fprintf(Logi,"I don't have memory enaugh..");
```

```
      return(0);
      }
memset(VK,'\0',vkl);
if(PBuf==NULL){
      fprintf(Logi,"scanner: pole faili %s",pr_name);
      return(0);
      }
i=0;
olek=0;
marker=0;
t=getV("#");
if(t!=(struct D *)NULL) marker=t->code;
if(marker==0){
  fprintf(Logi,"aksioomi definitsioonis peavad olema markerid '#'");
  return(0);
  }
VK[0]=marker;
k=1;
if(PBuf[i]=='#') i=1;
memset(word,'\0',20);
while(i<Plen){
      switch(olek){
            case 0:{
                  if(isspace(PBuf[i])){
                        i++;
                        break;
                        }
                  if(k_k>0){
                        if(isdigit(PBuf[i])){
                              j=0;
                              olek=2;
                              break;
                              }
                        }
                  r=lekseem(i);
                  if(r>0){
                        VK[k]=KOOD;
                        k++;
                        memset(word,'\0',20);
                        i=i+r;
                        j=0;
                        break;
                        }
                  j=0;
                  memset(word,'\0',20);
                  if(isalpha(PBuf[i])){
                        j=0;
                        olek=1;
                        break;
                        }
                  if(ispunct(PBuf[i])){
                        p_viga(i);
                        i++;
                        j=0;
                        memset(word,'\0',20);
                        break;
                        }
                  }
            case 1:{
                  if(isalpha(PBuf[i])){
                        word[j]=PBuf[i];
```

```
                        i++;
                        j++;
                        break;
                        }
                  if(isdigit(PBuf[i])){
                        word[j]=PBuf[i];
                        i++;
                        j++;
                        break;
                        }
                  if(ispunct(PBuf[i])){
                        k=Ident(k);
                        j=0;
                        olek=0;
                        break;
                        }
                  if(isspace(PBuf[i])){
                        k=Ident(k);
                        j=0;
                        i++;
                        olek=0;
                        break;
                        }
                  }
            case 2:{
                  if(isalpha(PBuf[i])){
                        p_viga(i);
                        i++;
                        j=0;
                        memset(word,'\0',20);
                        break;
                        }
                  if(isdigit(PBuf[i])){
                        word[j]=PBuf[i];
                        i++;
                        j++;
                        break;
                        }
                  if(ispunct(PBuf[i])){
                        k=Const(k);
                        j=0;
                        olek=0;
                        break;
                        }
                  if(isspace(PBuf[i])){
                        k=Const(k);
                        i++;
                        j=0;
                        olek=0;
                        break;
                        }
                  }
            }
}
if(VK[k-1]!=marker){
      VK[k]=marker;
      k++;
      }
print_VK(k);
VKL=k;
memset(word,'\0',20);
```

```
      memcpy(word,"DUMMY",5);
      Ident(k);
      dummy=nr;
      set_show("Scanner ended");
      if(VEAD>0) return(0);
      NR=nr;
      nr=Sn;
      return(1);
}

FILE *opr(void){
      of=fopen(rida,"rb");
      if (of==NULL){
            fprintf(Logi,"cannot open %s",rida);
            return(of);
            }
      return(of);
}

FILE *opw(void){
      of=fopen(rida,"wb");
      if (of==NULL){
            fprintf(Logi,"cannot open %s",rida);
            return(of);
            }
      return(of);
}

/* eraldab mälu uuele NT-definitsioonile: id / const */
struct R *newR(void){
      struct R *ptr;
      ptr=(struct R *)malloc(sizeof(struct R));
      if(ptr==NULL) ExIT( );
      memset(ptr,'\0',sizeof(struct R));
      ptr->alt=(struct R *)NULL;
      return(ptr);
}

struct D *r_V(void){
      struct D *t;
      struct R *od,*d;
      int f;
      t=newD( );
      fread(t,sizeof(struct D),1,of);
      if(feof(of)){
            fclose(of);
            return((struct D *)NULL);
            }
      f=ferror(of);
      if(f>0){
            fprintf(Logi,"file error %d",f);
            fclose(of);
            return((struct D *)NULL);
            }
      if(t->tunnus==0){
            t->def=newR();
            fread(t->def,sizeof(struct R),1,of);
            d=t->def;
            if(d->d!=(int *)NULL&&d->n>0){
                        d->d=(int *)malloc(d->n*sizeof(int));
                        if(d->d==NULL) ExIT();
```

```
                        fread(d->d,d->n*sizeof(int),1,of);
                        }
                od=d->alt;
                while(od!=(struct R *)NULL){
                        d->alt=newR( );
                        d=d->alt;
                        fread(d,sizeof(struct R),1,of);
                        if(d->d!=(int *)NULL&&d->n>0){
                                d->d=(int *)malloc(d->n*sizeof(int));
                                if(d->d==NULL) ExIT();
                                fread(d->d,d->n*sizeof(int),1,of);
                                }
                        od=d->alt;
                        }
                }
        if(t->left!=(struct D *)NULL) t->left=r_V();
        if(t->right!=(struct D *)NULL) t->right=r_V();
        return(t);
}

/* tabelite lugemine */
void r_DEP(void){
        int i,j;
        char **L;
        char *rc;
        DEP=defDEP();
        fread(DEP,sizeof(char ***),nr+1,of);
        for(i=0;i<nr+1;i++){
                if(DEP[i]!=(char **)NULL){
                        L=defL();
                        fread(L,sizeof(char **),nr+1,of);
                        DEP[i]=L;
                        for(j=0;j<nr+1;j++){
                                if(L[j]!=(char *)NULL){
                                        rc=DefVect(tnr+1);
                                        fread(rc,tnr+1,1,of);
                                        L[j]=rc;
                                        }
                                }
                        }
                }
}

struct h *grec(void){
        struct h *L;
        L=(struct h *)malloc(sizeof(struct h));
        if(L==(struct h *)NULL){
                fprintf(Logi,"I haven't memory enough..");
                ExIT( );
                }
        return(L);
}

char *gM(int p){
        char *ptr=NULL;
        ptr=(char *)malloc(p);
        if(ptr==NULL){
                fprintf(Logi,"I haven't memory enough..");
                ExIT( );
                }
        return(ptr);
```

```
}

struct h *r_hr(void){
      struct h *r;
      r=grec( );
      fread(r,sizeof(struct h),1,of);
      r->d=(int *)malloc(r->n*sizeof(int));
      fread(r->d,r->n*sizeof(int),1,of);
      if(r->same!=(struct h *)NULL) r->same=r_hr();
      if(r->col!=(struct h *)NULL) r->col=r_hr();
      return(r);
}

void r_HT(void){
      int i;
      fread(HT,sizeof(struct h *),HTL,of);
      for(i=0;i<HTL;i++){
            if(HT[i]!=(struct h *)NULL) HT[i]=r_hr();
            }
}

struct top *r_ptree(void){
      struct top *p;
      p=(struct top *)malloc(sizeof(struct top));
      if(p==(struct top *)NULL){
            fprintf(Logi,"I haven't memory enough..");
            ExIT( );
            }
      fread(p,sizeof(struct top),1,of);
      if(p->right!=(struct top *)NULL) p->right=r_ptree();
      if(p->down!=(struct top *)NULL) p->down=r_ptree();
      return(p);
}

void w_ptree(struct top *p){
      fwrite(p,sizeof(struct top),1,of);
      if(p->right!=(struct top *)NULL) w_ptree(p->right);
      if(p->down!=(struct top *)NULL) w_ptree(p->down);
}

int w_tree(void){
      sprintf(rida,"%s.pt",Pr_name);
      if(opw( )!=NULL){
            w_ptree(p_);
            return(1);
            }
      return(0);
}

struct top *set_up(struct top *root){
      struct top *p=NULL;
      if(root!=(struct top *)NULL){
            if(root->down!=(struct top *)NULL) p=set_up(root->down);
            if(root->right!=(struct top *)NULL) p=set_up(root->right);
            if(p->up!=(struct top *)NULL) p->up=root;
            }
      return(p);
}

int r_t(void){
      sprintf(rida,"%s.t",Nimi);
```

```
      if(opr()!=NULL){
            fread(&T,20,nr+1,of);
            fclose(of);
            return(1);
            }
      else return(0);
}

int r_tt(void){
      sprintf(rida,"%s.tt",Nimi);
      if(opr()!=NULL){
            fread(&TT,20,tnr+1,of);
            fclose(of);
            return(1);
            }
      else return(0);
}

int w_t(void){
      sprintf(rida,"%s.t",Pr_name);
      if(opw()!=NULL){
            fwrite(&T,20,NR+1,of);
            fclose(of);
            return(1);
            }
      else return(0);
}

int r_parm(void){
      PARM=(struct parm *)malloc(sizeof(struct parm));
      if(PARM==(struct parm *)NULL) ExIT();
      memset(PARM,'\0',sizeof(struct parm));
      sprintf(rida,"%s.prm",Nimi);
      if(opr()!=NULL){
            fread(PARM,sizeof(struct parm),1,of);
            fclose(of);
            nr=PARM->nr;
            tnr=PARM->tnr;
            Sn=nr;
            context=PARM->BRC;
            Pnr=PARM->Pnr;
            dc=PARM->dep;
            return(1);
            }
      return(0);
}


int w_parm(void){
      sprintf(rida,"%s.prm",Pr_name);
      if(opw()!=NULL){
            PARM->nr=NR;
            PARM->ktl=ktl;
            PARM->itl=itl;
            fwrite(PARM,sizeof(struct parm),1,of);
            fclose(of);
            return(1);
            }
      return(0);
}
```

```
int r_sm(void){
      sprintf(rida,"%s.sm",Nimi);
      if(opr()!=NULL){
            semantika=(int *)malloc(sizeof(int)*(tnr+Pnr+1));
            fread(semantika,sizeof(int)*(tnr+Pnr+1),1,of);
            fclose(of);
            semflag=1;
            return(1);
            }
      return(0);
}

int r_v(void){
      sprintf(rida,"%s.v",Nimi);
      if(opr()!=NULL){
            DT=r_V();
            fclose(of);
            return(1);
            }
      else return(0);
}

int r_ht(void){
      sprintf(rida,"%s.ht",Nimi);
      if(opr()!=NULL){
            r_HT();
            fclose(of);
            return(1);
            }
      else return(0);
}

int r_pm(void){
      sprintf(rida,"%s.pm",Nimi);
      if(opr()!=NULL){
            PM=DefArray(nr+1);
            fread(PM,(nr+1)*(nr+1),1,of);
            fclose(of);
            return(1);
            }
      else return(0);
}

int inxt(void){
      if(context==1){
            sprintf(rida,"%s.lc",Nimi);
            if(opr()!=NULL){
                  LC=DefArray(nr+1);
                  fread(LC,(nr+1)*(nr+1),1,of);
                  fclose(of);
                  }
            else return(0);
            sprintf(rida,"%s.rc",Nimi);
            if(opr( )!=NULL){
                  RC=DefArray(nr+1);
                  fread(RC,(nr+1)*(nr+1),1,of);
                  fclose(of);
                  }
            else return(0);
      }
      return(1);
```

```
}

int depc(void){
      if(dc==1){
            sprintf(rida,"%s.dc",Nimi);
            if(opr( )!=NULL){
                  r_DEP();
                  fclose(of);
                  }
            else return(0);
            }
      return(1);
}

int w_it(void){
      sprintf(rida,"%s.it",Pr_name);
      if(opw()!=NULL){
            fwrite(IT,itl*sizeof(int),1,of);
            fclose(of);
            return(1);
            }
      return(0);
}

int w_kt(void){
      sprintf(rida,"%s.kt",Pr_name);
      if(opw()!=NULL){
            fwrite(KT,ktl*sizeof(int),1,of);
            fclose(of);
            return(1);
            }
      return(0);
}

int r_tabs(void){
      int flag;
      flag=1;
      flag=r_parm();
      if(flag==0) goto out;
      flag=r_t( );
      if(flag==0) goto out;
      flag=r_tt( );
      if(flag==0) goto out;
      flag=r_sm( );
      if(flag==0) goto out;
      flag=r_v( );
      if(flag==0) goto out;
      flag=r_ht( );
      if(flag==0) goto out;
      PM=DefArray(nr+1);
      flag=r_pm( );
      if(flag==0) goto out;
      if(context>0){
            LC=DefArray(nr+1);
            RC=DefArray(nr+1);
            flag=inxt( );
            if(flag==0) goto out;
            }
if(dc>0){
      flag=depc( );
      }
```

```
out:    if(flag==0){
            fprintf(Logi,"cannot read all the tables..");
            }
      return(flag);
}

void p_w(struct stat *b){
      int p;
      if(stat(rida,b)==-1){
            sprintf(rida,"w_tabs: stat failed<BR>");
            ExiT(rida);
            }
      p=b->st_size;
      fprintf(Logi,"<TR><TD>%s</TD><TD
ALIGN=\"right\">%d</TD></TR>",rida,p);
}

int w_p_tabs(void){
      struct stat *buf;
      fprintf(Logi,"<H3><FONT COLOR=\"#0000FF\">Result tables</H3><BR>");
      fprintf(Logi,"<TABLE BORDER=1><TR>");
      fprintf(Logi,"<TD><FONT COLOR=\"#0000FF\">File</TD><TD><FONT
      COLOR=\"#0000FF\">Size</TD></FONT>");
      buf=(struct stat *)malloc(sizeof(struct stat));
      if(buf==NULL){
            sprintf(rida,"w_tabs: I haven't %d bytes of memory..<BR>",
                  sizeof(struct stat));
            ExiT(rida);
            }
      w_parm( );
      fclose(of);
      p_w(buf);
      w_t( );
      fclose(of);
      p_w(buf);
      w_it( );
      fclose(of);
      p_w(buf);
      w_kt( );
      fclose(of);
      p_w(buf);
      w_tree( );
      fclose(of);
      p_w(buf);
      fprintf(Logi,"</TABLE>");
      fflush(Logi);
      return(1);
}

int p_top_rdf(struct top *t){
if(t==(struct top *)NULL) return(0);
fprintf(rdf,
   "top %p: kood=%d leks=%d sem=%d label=%d truel=%d falsel=%d up=%p
   right=%p down=%p<BR>",
   t,t->kood,t->leks,t->sem,t->label,t->truel,t->falsel,t->up,t->right,
   t->down);
p_top_rdf(t->down);
p_top_rdf(t->right);
return(1);
}
```

```
int make_rtf(void){
int i;
      sprintf(rida,"%srd.htm",Pr_name);
      rdf=fopen(rida,"w");
      if(rdf==NULL) return(0);
      fprintf(rdf,"<HTML><HEAD><TITLE>");
      fprintf(rdf,"TABLES %s",pr_name);
      fprintf(rdf,
      "</TITLE></HEAD><BODY><H4><FONT COLOR=\"#0000FF\">PARSER TABLES of
the Word %s</H4>",
            pr_name);
      fprintf(rdf,"</FONT><B>");
      fflush(rdf);
      fprintf(rdf,"<UL><LI><A HREF=\"#parm\">Parameters</A></LI>");
      fprintf(rdf,"<LI><A HREF=\"#t\">Dictionary T</A></LI>");
      if(PARM->itl>0)
            fprintf(rdf,"<LI><A HREF=\"#it\">Identifiers</A></LI>");
      if(PARM->ktl>0)
            fprintf(rdf,"<LI><A HREF=\"#kt\">Constants</A></LI>");
      fprintf(rdf,"<LI><A HREF=\"#pt\">Parsing tree</A></LI>");
      fprintf(rdf,"</UL>");
      fprintf(rdf,"<HR>");

      fprintf(rdf,"<A NAME=\"parm\"></A>");
      fprintf(rdf,"<H4><FONT COLOR=\"#0000FF\">Parameters</H4></FONT>");
      fprintf(rdf,"File %s.prm<BR>",Pr_name);
      fprintf(rdf,"<PRE>");
      fprintf(rdf,"struct parm{\n");
      fprintf(rdf,"\tint nr   //t&auml;hestiku V pikkus\n");
      fprintf(rdf,"\tint tnr  //t&auml;hestiku V<SUB>T</SUB> pikkus\n");
      fprintf(rdf,"\tint BRC  //0: G on p&ouml;&ouml;ratav\n");
      fprintf(rdf,"\tint Pnr  //Produktsioonide arv\n");
      fprintf(rdf,"\tint dep  //1: s&otilde;ltuv kontekst\n");
      fprintf(rdf,"\tint itl  //identifikaatorite arv\n");
      fprintf(rdf,"\tint ktl  //konstantide arv\n");
      fprintf(rdf,"}\n\n");
      fprintf(rdf,"nr=%d tnr=%d BRC=%d Pnr=%d dep=%d itl=%d ktl=%d\n\n",
      PARM->nr,PARM->tnr,PARM->BRC,PARM->Pnr,PARM->dep,PARM->itl,
      PARM->ktl);
      fprintf(rdf,"</PRE><A NAME=\"t\"></A><HR>");
      fprintf(rdf,"<H4><FONT COLOR=\"#0000FF\">Dictionary T</H4></FONT>");
      fprintf(rdf,"File %s.t<BR>",Pr_name);
      fprintf(rdf,"<PRE>");
      for(i=1;i<=NR;i++) fprintf(rdf,"%s ",T[i]);
      fprintf(rdf,"</PRE><BR><BR><HR>");

if(itl>0){
      fprintf(rdf,"<A NAME=\"it\"></A>");
      fprintf(rdf,"<H4><FONT COLOR=\"#0000FF\">Identifiers</H4></FONT>");
      fprintf(rdf,"File %s.it<BR>",Pr_name);
      fprintf(rdf,"<PRE>");
      for(i=0;i<itl;i++) fprintf(rdf,"%d ",IT[i]);
      fprintf(rdf,"</PRE><BR><BR><HR>");
      }
if(ktl>0){
      fprintf(rdf,"<A NAME=\"kt\"></A>");
      fprintf(rdf,"<H4><FONT COLOR=\"#0000FF\">Constants</H4></FONT>");
      fprintf(rdf,"File %s.kt<BR>",Pr_name);
      fprintf(rdf,"<PRE>");
      for(i=0;i<ktl;i++) fprintf(rdf,"%d ",KT[i]);
```

```
      fprintf(rdf,"</PRE><BR><BR><HR>");
      }
      fprintf(rdf,"<A NAME=\"pt\"></A>");
      fprintf(rdf,"<H4><FONT COLOR=\"#0000FF\">Parsing tree</H4></FONT>");
      fprintf(rdf,"File %s.pt<BR>",Pr_name);
      fprintf(rdf,"<PRE>");
      fprintf(rdf,"struct top{<BR>");
      fprintf(rdf,"\tint kood;\t/* vahekeelne kood */<BR>");
      fprintf(rdf,"\tint leks;\t/* kui tipp=ident/const, siis selle
      jrk-nr.*/<BR>");
      fprintf(rdf,"\tint sem;\t/* semantikakood */<BR>");
      fprintf(rdf,"\tint label;\t/* kui tipp on m&auml;rgendatud operaator
      - m&auml;rgendi nr */<BR>");
      fprintf(rdf,"\tint truel;\t/* kompilaator: go to true */<BR>");
      fprintf(rdf,"\tint falsel;\t/* kompilaator: go to false */<BR>");
      fprintf(rdf,"\tstruct top *up;\t/* puuviidad: &uuml;les, */<BR>");
      fprintf(rdf,"\tstruct top *right;\t/* naabrile ja */<BR>");
      fprintf(rdf,"\tstruct top *down;\t/* alluvale */<BR>");
      fprintf(rdf,"}<BR><BR><BR>");
      p_top_rdf(p_);
      fprintf(rdf,"</PRE>");

      fprintf(rdf,"</BODY></HTML>");
      fflush(rdf);
      fclose(rdf);
      return(1);
}

struct top *analyzer(void){
      if(r_tabs()==0) return(0);
      Sn=nr;
      P=PBuf;
      set_show("Scanner started");
      if(scanner()==0||VEAD>0) return(0);
      set_show("Parser started");
      p_=parser( );
      if(p_==(struct top *)NULL) return(0);
      set_show("Parsing tree");
      pp2html(p_);
      ctree( );
      return(p_);
}

void ctree(void){
      if(ktl>0){
            set_show("Table of constants");
            make_CT();
            }
      if(itl>0){
            set_show("Table of identifiers");
            make_IDT();
            fprintf(Logi,"<BR>");
            }
}

int red_dep(struct h *S,int x,int y){
      char **L;
      char *rc;
      L=DEP[S->NT];
      if(L!=(char **)NULL){
            rc=L[x];
```

```
            if(rc==(char *)NULL) return(0);
            if(rc[y]==1){
                  hc=S;
                  return(S->NT);
                  }
            }
      return(0);
}

int red_indep(struct h *t,int x,int y){
      int ret=0;
      int r=0;
      struct h *S;
      S=t;
      while(S!=(struct h *)NULL){
            if(S->nc==0){
                  if(LC[adr(S->NT,x)]>0){
                        r++;
                        ret=S->NT;
                        hc=S;
                        }
                  }
            S=S->same;
            }
      if(r==0){
            fprintf(Logi,"\n%s is not in any LCs\n",T[x]);
            hc=(struct h *)NULL;
            return(0);
            }
      if(r==1) return(ret);
      S=t;
      while(S!=(struct h *)NULL){
            if(S->nc==0){
                  if(RC[adr(S->NT,y)]>0){
                        hc=S;
                        return(S->NT);
                        }
                  }
            S=S->same;
            }
      fprintf(Logi,"\n%s is not in any RCs\n",T[y]);
      hc=(struct h *)NULL;
      return(0);
}

/* redutseerib lause vormi baasi mitteterminali koodiks */
int reduce(struct h *t,int x, int y){
      int ret;
      struct h *S;
      ret=0;
      hc=(struct h *)NULL;
      if(t->same==(struct h *)NULL){
            hc=t;
            return(t->NT);
            }
      S=t;
      while(S!=(struct h *)NULL){
            if(S->nc>0){
                  ret=red_dep(S,x,y);
                  if(ret>0) return(ret);
                  }
```

```
            S=S->same;
            }
      ret=red_indep(t,x,y);
      if(ret>0) return(ret);
      return(0);
}

char s[7][60]={
      " ",
      "<FONT COLOR=\"#FF4500\">=&#149</FONT>",
      "<FONT COLOR=\"#FF4500\">&lt&#149</FONT>",
      " ",
      "<FONT COLOR=\"#FF4500\">&#149&gt</FONT>",
      };

/* analüüs: magasini ja analüüsimata teksti väljatrükk */
int p_stack(char *name,int *v,char *rel,int a,int n,int k){
      int i,j,lv;
      fprintf(Logi,"<BR><TABLE BORDER=1><B><TR><TD NOWRAP>Stack &
Word</TD>");
      for(lv=n;lv>0;lv--){
            if(rel[lv]==2) break;
            }
      for(i=a;i<n;i++){
            j=rel[i];
            fprintf(Logi,"<TD NOWRAP> %s",s[j]);
            j=v[i];
            if(i<lv)
              fprintf(Logi,"<FONT
COLOR=\"#0000FF\"><STRONG>%s</FONT></TD>",T[j]);
            else
              fprintf(Logi,"<FONT
COLOR=\"#FF0000\"><STRONG>%s</FONT></TD>",T[j]);
            }
      for(i=k; i<VKL; i++){
            j=VK[i];
            fprintf(Logi,"<TD>%s </TD>",T[j]);
            }
      fprintf(Logi,"</TR></TABLE>");
      return(0);
}

/* analüüsi puu moodustamine: uus tipp */
struct top *newtop(int x,int i,int sem){
      struct top *ptr;
      ptr=(struct top *)NULL;
      if(x<tnr+1 && semantika[sem]==0) goto out;
      ptr=(struct top *)malloc(sizeof(struct top));
      if(ptr==(struct top *)NULL){
            fprintf(Logi,"newtop: lack of memory of %d bytes<BR>",
            sizeof(struct top));
            ExIT( );
            }
      memset(ptr,'\0',sizeof(struct top));
      ptr->up=(struct top *)NULL;
      ptr->right=(struct top *)NULL;
      ptr->down=(struct top *)NULL;
      ptr->kood=x;
      ptr->sem=semantika[sem];
      if((x==m_k)||(x==k_k)){
            ptr->leks=(i==0) ? dummy : VK[i];
```

```
            }
out:    return(ptr);
}

/* analüüsi puu moodustamine: uue tipu/alampuu lisamine */
void red_tree(struct top *puu[ ],struct top *p,int s,int n){
      int i;
      struct top *cp=(struct top *)NULL;
      struct top *first=(struct top *)NULL;
      struct top *last=(struct top *)NULL;
      for(i=0;i<n;i++){
            if(puu[i+s]!=(struct top *)NULL){
                  cp=puu[i+s];
                  if(cp->sem==0){
                        if(cp->down==(struct top *)NULL) goto ilja;
                        if(cp->down!=(struct top *)NULL){
                              cp=cp->down;
                              if(first==(struct top *)NULL) first=cp;
                              if(last!=(struct top *)NULL) last->right=cp;
                              while(cp->right!=(struct top *)NULL){
                                    cp=cp->right;
                                    }
                              cp->up=(struct top *)NULL;
                              goto next;
                              }
                       }
                  if(first==(struct top *)NULL) first=cp;
                  if(last!=(struct top *)NULL) last->right=cp;
                  }
            next: last=cp;
            ilja: cp=cp;
            }
      p->down=first;
      if(last!=(struct top *)NULL) last->up=p;
}

/* analüüsi puu trükk */
int pp2html(struct top *p){
      double pr;
      int n=1;
      struct top *t;
      char s[20];
      t=p;
naaber:
      if(t->right!=NULL){
            n++;
            t=t->right;
            goto naaber;
            }
      pr=100.0/n;
      sprintf(s,"\"%2.0f&#037\"",pr);
      fprintf(Logi,"<table border=1><tr>");
next:
      fprintf(Logi,
      "<td width=%s valign=\"top\"><STRONG>",s);
      if((p->kood==m_k)||(p->kood==k_k)) fprintf(Logi,"%s",T[p->leks]);
      else fprintf(Logi,"%s",T[p->kood]);
      if(p->down!=NULL) pp2html(p->down);
      fprintf(Logi,"</td>");
      if(p->right!=NULL){
            p=p->right;
```

```
            goto next;
            }
      fprintf(Logi,"</tr></table>");
      return 1;
}

void p_sf(void){
      int i,j;
      fprintf(Logi,"<BR>Sentential form: ");
      for(i=0; i<n_; i++){
            j=lausevorm[i];
            fprintf(Logi,"%s ",T[j]);
            }
      fprintf(Logi,"<BR>");
      fflush(Logi);
}

void sentential_form(void){
      for(s_=J;s_>0;s_--){
            if(rela[s_]==2) break;
            }
      memset(lausevorm,'\0',sizeof(lausevorm));
      n_=J-s_;
      for(l_=0;l_<n_;l_++){
      lausevorm[l_]=Stack[s_+l_];
      }
}


int correct(void){
      int i,r;
      if(x_==VK[I]) return(0);
      for(i=1;i<tnr;i++){
            r=PM[adr(x_,i)];
            if(r!=0){
                  Stack[J]=i;
                  rela[J]=r;
                  x_=i;
                  p_stack("\n",Stack,rela,0,J+1,I);
                  if((x_==m_k)||(x_==k_k)) puu[J]=newtop(x_,0,x_);
                  J++;
                  return(1);
                  }
            }
      return(0);
}


/* analüsaator. Ette vahekeelne programm, välja (hõre) analüüsi puu */
struct top *parser(void){
      int i,j,r;
      char c=' ';
      memset(Stack,'\0',sizeof(Stack));
      memset(rela,'\0',200);
      for(I=0;I<200;I++) puu[I]=(struct top *)NULL;
      Stack[0]=marker;
      x_=marker;
      J=1;
      I=1;
while(I<VKL){
      rel=PM[adr(x_,VK[I])];
```

```
      j=VK[I];
      rela[J]=rel;
      switch(rel){
            case 0:
                  j=VK[I];
                  fprintf(Logi,
                        "<BR>no relationship between the symbols %s and
%s<BR>",
                        T[x_],T[j]);
                  fprintf(Logi,"The relationships of %s<BR>",T[x_]);
                  for(i=0;i<nr+1;i++){
                        r=PM[adr(x_,i)];
                        if(r>0){
                              if(r==1) c='=';
                              if(r==2) c='<';
                              if(r==4) c='>';
                              fprintf(Logi,"%s %c %s<BR>",T[x_],c,T[i]);
                              }
                        }
                  fprintf(Logi,"<BR>");
                  if(X==0){
                        p_=(struct top *)NULL;
                        return(p_);
                        }
                  fprintf(Logi,"I'll try to correct..\n");
                  flag=1;
                  if(correct()==0) I++;
                  break;
            case 1: if(VK[I]==marker){
                        set_show("the parsing is completed");
                        return(p_);
                        }
                  Stack[J]=VK[I];
                  x_=Stack[J];
                  if((x_==m_k)||(x_==k_k)) I++;
                  puu[J]=newtop(x_,I,x_);
                  p_stack("\n",Stack,rela,0,J+1,I+1);
                  J++;
                  I++;
                  break;
            case 2: Stack[J]=VK[I];
                  x_=Stack[J];
                  if((x_==m_k)||(x_==k_k)) I++;
                  puu[J]=newtop(x_,I,x_);
                  p_stack("\n",Stack,rela,0,J+1,I+1);
                  J++;
                  I++;
                  break;
            case 4:
                  sentential_form( );
                  ps( );
                  t_=ReadHash(n_,lausevorm);
                  if(t_==(struct h *)NULL){
                        p_=(struct top *)NULL;
                        return(p_);
                        }
                  N_=reduce(t_,Stack[s_-1],VK[I]);
                  fflush(Logi);
                  t_=hc;
                  if(N_==0){
                        set_show("reduce error");
```

```
                        p_=(struct top *)NULL;
                        return(p_);
                        }
                  rela[s_]=PM[adr(Stack[s_-1],N_)];
                  Stack[s_]=N_;
                  x_=N_;
                  p_=newtop(N_,I,t_->P+tnr);
                  red_tree(puu,p_,s_,n_);
                  puu[s_]=p_;
                  p_stack("<BR>",Stack,rela,0,s_+1,I);
                  if(semantika[t_->P+tnr]>0){
                        set_show("Parsing tree");
                        pp2html(p_);
                        }
                  fprintf(Logi,"<BR>");
                  J=s_+1;
                  break;
            }
      }
      return(p_);
}

/* mälueraldus konstantide tabeli kirjele */
struct ctr *make_ctr(void){
      struct ctr *c;
      c=(struct ctr *)malloc(sizeof(struct ctr));
      if(c==(struct ctr *)NULL) ExIT();
      memset(c,'\0',sizeof(struct ctr));
      return(c);
}

/* konstantide tabeli moodustamine */
int make_CT(void){
      int t;
      struct ctr *c;
      int i;
      if(ktl==0) return(0);
      fprintf(Logi,"<TABLE BORDER=1><TR>");
      for(i=0;i<ktl;i++){
            t=KT[i];
            c=make_ctr();
            c->nr=t;
            c->value=atoi(T[c->nr]);
            CT[i]=c;
            fprintf(Logi,"<TD>c%d=%d</TD>",i+1,c->value);
            }
      fprintf(Logi,"</TR></TABLE>");
      return(ktl);
}

/* mälueraldus identifikaatorite tabeli kirjele */
struct itr *make_itr(void){
      struct itr *c;
      c=(struct itr *)malloc(sizeof(struct itr));
      if(c==(struct itr *)NULL) ExIT();
      memset(c,'\0',sizeof(struct itr));
      c->t=(struct top *)NULL;
      return(c);
}
```

```
/* identifikaatorite tabeli moodustamine */
int make_IDT(void){
      int t;
      struct itr *c;
      int i;
      if(itl==0) return(0);
      itl=(flag==0) ? itl-1 : itl;
      fprintf(Logi,"<TABLE BORDER=1><TR>");
      for(i=0;i<itl;i++){
            t=IT[i];
            c=make_itr();
            c->nr=t;
            IDT[i]=c;
            fprintf(Logi,"<TD>i%d=%s</TD>",i+1,T[c->nr]);
            }
      fprintf(Logi,"</TR></TABLE>");
      return(itl);
}

/* identifikaatorite tabeli kirje trükk */
void p_itr(struct itr *id){
      fprintf(Logi,"ITR nr=%d V=%s loc=%d value=%d t=%p<BR>",
            id->nr,T[id->nr],id->loc,id->value,id->t);
}

/* muutujate väljatrükk */
void print_variables(void){
      struct itr *id;
      int i,k;
      set_show("THE VARIABLES:");
      k=(flag==0) ? itl-1 : itl;
      for(i=0;i<k;i++){
            id=IDT[i];
            if(id->t==(struct top *)NULL)
            fprintf(Logi,"%s=%d<BR>",T[id->nr],id->value);
            }
}

/* konstantide tabeli kirje trükk */
void p_ctr(struct ctr *id){
      fprintf(Logi,"CTR nr=%d V=%s loc=%d value=%d<BR>",
            id->nr,T[id->nr],id->loc,id->value);
}

/* tekstimassiivi sisestamine kettalt */
char *jarray(char *pealkiri){
      FILE *tekst=NULL;
      char *B;
      char c;
      int  i;
      int  k;
      struct stat *buf;
      buf=(struct stat *)malloc(sizeof(struct stat));
      if(buf==NULL){
            fprintf(Logi,"r_text: I haven't %d bytes of memory..<BR>",
                  sizeof(struct stat));
            ExIT();
            }
      tekst = fopen(pealkiri, "r");
      if (tekst==NULL){
            fprintf(Logi,"cannot find the file  %s <BR>",pealkiri);
```

```
            ExIT();
            }
      if(stat(pealkiri,buf)==-1){
            fprintf(Logi,"r_text: stat failed<BR>");
            ExIT();
            }
      k=buf->st_size;
      B = (char *)malloc(k+10);
      if(B == NULL){
            fprintf(Logi,"I don't have memory enaugh..");
            ExIT();
            }
      memset(B,'\0',k+10);
/* fill buffer */
      rewind(tekst);
      i=0;
      while (!feof(tekst)){
            c=fgetc(tekst);
            B[i]=c;
            i++;
            }
      P_length=i;
      for(i=P_length;i>0;i--){
            if(isgraph(B[i])){
                  i++;
                  B[i]='\n';
                  i++;
                  B[i]='\0';
                  P_length=i;
                  goto out;
                  }
            }
out:    fclose(tekst);
      return(B);
}

int prog(void){
      PBuf=jarray(pr_name);
      if(PBuf!=NULL){
            Plen=P_length;
            return(1);
            }
      return(0);
}

int itr(void){
      int ret=0;
      time_t t0;
      char c;
      GBuf=NULL;
      Logi=fopen(L_name,"w");
      if(Logi==NULL){
            printf("Cannot open log-book\n");
            return(0);
            }
      logi=1;
      time(&t0);
      fprintf(Logi,"<HTML><HEAD><TITLE>Parser</TITLE></HEAD><BODY><B>");
      fprintf(Logi,
      "<FONT COLOR=\"#0000FF\"><H3>Start of %s Parser for a ",Nimi);
      c=Nimi[0];  c=toupper(c);
```

```
	switch(c){
		case 'G': fprintf(Logi,"Word "); break;
		case 'T': fprintf(Logi,"Program "); break;
		case 'F': fprintf(Logi,"Formula "); break;
		}
	fprintf(Logi,"</FONT>%s ",pr_name);
	fprintf(Logi,"<FONT COLOR=\"#0000FF\"> at </FONT>");
	fprintf(Logi,"%s</H3><BR>",asctime(localtime(&t0)));
	if(prog()==0){
		fflush(Logi);
		fclose(Logi);
		return(ret);
		}
	if(analyzer()!=NULL){
		w_p_tabs();
		make_rtf();
		}
	time(&t0);
	fprintf(Logi,"<FONT COLOR=\"#0000FF\"><H4>Parser ended at </FONT> ");
	fprintf(Logi,"%s</H4>",asctime(localtime(&t0)));
	fprintf(Logi,"</BODY></HTML>");
	return(ret);
}

struct top *main(int argc,char **argv){
char *p, c;
char wrd[20];
	pr_name=(char *)malloc(256);
	Pr_name=(char *)malloc(256);
	L_name=(char *)malloc(256);
	Nimi=(char *)malloc(256);
	memset(pr_name,'\0',256);
	memset(Pr_name,'\0',256);
	memset(L_name,'\0',256);
	memset(Nimi,'\0',256);
	strcpy(Nimi,argv[1]);
	sprintf(pr_name,"%s",argv[2]);
	strcpy(L_name,pr_name);
	strcpy(Pr_name,pr_name);
	p=strrchr(L_name,'.');
	strcat(L_name,"p.htm");
	memset(wrd,'\0',20);
	X=0;
	c=tolower(Nimi[0]);
	switch(c){
		case 'g':
			sprintf(wrd,".%s",Nimi);
			strcat(pr_name,wrd);
			break;
		case 't': strcat(pr_name,".tri");
				X=1;
				break;
		case 'f': strcat(pr_name,".frm"); break;
		}
	printf("gr_name=%s.grm pr_name=%s L_name=%s\n",Nimi,pr_name,L_name);
	itr( );
	fflush(Logi);
	return(p_);
}
```

Lisa 8. Trigoli interpretaator

```
/* int32.c  May 2001 */
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
#include <sys/stat.h>
#include "two32.h"

/*        S E M A N T I C S                          */

#define muut 1      /* #i# - identifikaator */
#define konst 2     /* #c# - konstant */
#define pisem 3     /* < */
#define suurem 4    /* > */
#define piv 5       /* <= */
#define suv 6       /* >= */
#define pov 7       /* /= */
#define vord 8      /* = */
#define omist 10    /* omistamisoperaator */
#define jag 11      /* jagamistehe */
#define korrut 12   /* korrutamistehe */
#define lahut 13    /* lahutustehe */
#define liit 14     /* liitmistehe */
#define labl 15     /* märgend */
#define suunam 16   /* suunamisoperaator */
#define kuisiis 18  /* if-lause */
#define tingop 19   /* tingimusoperaator */
#define lugem 20    /* lugemisoperaator */
#define kirjut 21   /* kirjutamisoperaator */

/* interpretaatori magasinielement */
struct item{
      int liik;         /* 0 töömuutuja, 1 id, 2 c, 3 top */
      int index;       /* töömuutuja väärtus  */
      struct itr *id;   /* liik = 1 (muutuja)  */
      struct ctr *c;    /* liik = 2 (konstant) */
      struct top *t;    /* liik = 3 (märgend)  */
      };

  struct item *stack[20];   /* interpretaatori magasin */

void blue(char *t){
      fprintf(Logi,"<FONT COLOR=\"0000FF\">%s</FONT>",t);
}

void green(char *t){
      fprintf(Logi,"<FONT COLOR=\"008000\">%s</FONT>",t);
}

void red(char *t){
      fprintf(Logi,"<FONT COLOR=\"FF0000\">%s</FONT>",t);
}

void pset(int x,int y){
      blue("{");
      fprintf(Logi,"%s",T[x]);
```

```
      blue(",");
      fprintf(Logi,"%s",T[y]);
      blue("}");
}

void print_top(struct top *t){
fprintf(Logi,
"top %p: kood=%d leks=%d sem=%d label=%d truel=%d falsel=%d up=%p right=%p
down=%p",t,t->kood,t->leks,t->sem,t->label,t->truel,t->falsel,t->up,
t->right,t->down);
fprintf(Logi,"  (%s)<BR>",(t->leks==0) ? T[t->kood] : T[t->leks]);
if(t->right!=(struct top *)NULL) print_top(t->right);
if(t->down!=(struct top *)NULL) print_top(t->down);
}

void print_Itr(struct itr *t){
fprintf(Logi," itr %p: nr=%d loc=%d value=%d top=%p io=%d",
   t,t->nr,t->loc,t->value,t->t,t->io);
fprintf(Logi,"   (%s)<BR>",T[t->nr]);
}

void print_Ctr(struct ctr *t){
fprintf(Logi," ctr %p: nr=%d loc=%d value=%d",
   t,t->nr,t->loc,t->value);
fprintf(Logi,"   (%s)<BR>",T[t->nr]);
}

void print_one_top(struct top *t){
fprintf(Logi," top %p: kood=%d leks=%d sem=%d label=%d truel=%d falsel=%d
up=%p right=%p down=%p",
   t,t->kood,t->leks,t->sem,t->label,t->truel,t->falsel,t->up,t->right,
   t->down);
fprintf(Logi,"  (%s)<BR>",(t->leks==0) ? T[t->kood] : T[t->leks]);
}

/* analüüsi puu tipu trükk */
void p_top(char *t,struct top *p){
if(logi==1){
      fprintf(Logi,"%s top:<BR>",t);
      print_one_top(p);
      }
}

void set_show(char *title){
      fprintf(Logi,"<H4><FONT COLOR=\"#0000FF\">%s",title);
      fprintf(Logi,"</FONT></H4>");
}

void ps(void){
      fprintf(Logi,"<BR>");
}

void ExIT(void){
      if(logi==1){
            fprintf(Logi,"Abnormal end<BR><BR>");
            fflush(Logi);
            fclose(Logi);
            }
      abort( );
}
```

```
FILE *opr(void){
      of=fopen(rida,"rb");
      if (of==NULL){
            fprintf(Logi,"cannot open %s",rida);
            return(of);
            }
      return(of);
}

struct top *r_ptree(void){
      struct top *p;
      p=(struct top *)malloc(sizeof(struct top));
      if(p==(struct top *)NULL){
            fprintf(Logi,"I haven't memory enough..");
            ExIT();
            }
      fread(p,sizeof(struct top),1,of);
      if(p->right!=(struct top *)NULL) p->right=r_ptree();
      if(p->down!=(struct top *)NULL) p->down=r_ptree();
      return(p);
}

//kirjutab kettalt loetud analüüsi puusse up-viidad
void set_up(struct top *root){
      struct top *p;
      p=root;
      if(p->down!=(struct top *)NULL){
            p=p->down;
            while(p->right!=(struct top *)NULL){
                  p=p->right;
                  }
            p->up=root;
            }
      if(root->down!=(struct top *)NULL) set_up(root->down);
      if(root->right!=(struct top *)NULL) set_up(root->right);
}

int r_tree(void){
      sprintf(rida,"%s.pt",Pr_name);
      if(opr()!=NULL){
            p_=r_ptree();
            set_up(p_);
            return(1);
            }
      return(0);
}

int r_t(void){
      sprintf(rida,"%s.t",Pr_name);
      if(opr()!=NULL){
            fread(&T,20,nr+1,of);
            fclose(of);
            return(1);
            }
      else return(0);
}

int r_parm(void){
      PARM=(struct parm *)malloc(sizeof(struct parm));
      if(PARM==(struct parm *)NULL) ExIT();
      memset(PARM,'\0',sizeof(struct parm));
```

```
      sprintf(rida,"%s.prm",Pr_name);
      if(opr( )!=NULL){
            fread(PARM,sizeof(struct parm),1,of);
            fclose(of);
            nr=PARM->nr;
            itl=PARM->itl;
            ktl=PARM->ktl;
            return(1);
            }
      return(0);
}

int r_it(void){
      sprintf(rida,"%s.it",Pr_name);
      if(opr( )!=NULL){
            fread(&IT,itl*sizeof(int),1,of);
            fclose(of);
            return(1);
            }
      return(0);
}

int r_kt(void){
      sprintf(rida,"%s.kt",Pr_name);
      if(opr()!=NULL){
            fread(&KT,ktl*sizeof(int),1,of);
            fclose(of);
            return(1);
            }
      return(0);
}

int r_tabs(void){
      int flag;
      flag=1;
      flag=r_parm( );
      if(flag==0) goto out;
      flag=r_t( );
      if(flag==0) goto out;
      flag=r_it( );
      if(flag==0) goto out;
      flag=r_kt( );
      if(flag==0) goto out;
      flag=r_tree( );
      fflush(Logi);
out:    if(flag==0){
            fprintf(Logi,"cannot read all the tables..");
            fflush(Logi);
            }
      return(flag);
}

/* analüüsi puu trükk */
int pp2html(struct top *p){
      double pr;
      int n=1;
      struct top *t;
      char s[20];
      t=p;
naaber:
      if(t->right!=NULL){
```

```
            n++;
            t=t->right;
            goto naaber;
            }
      pr=100.0/n;
      sprintf(s,"\"%2.0f&#037\"",pr);
      fprintf(Logi,"<table border=1><tr>");
next:
      fprintf(Logi,
      "<td width=%s valign=\"top\"><STRONG>",s);
      if((p->kood==m_k)||(p->kood==k_k)) fprintf(Logi,"%s",T[p->leks]);
      else tprop(p->sem);
      if(p->down!=NULL) pp2html(p->down);
      fprintf(Logi,"</td>");
      if(p->right!=NULL){
            p=p->right;
            goto next;
            }
      fprintf(Logi,"</tr></table>");
      return 1;
}

/* mälueraldus konstantide tabeli kirjele */
struct ctr *make_ctr(void){
      struct ctr *c;
      c=(struct ctr *)malloc(sizeof(struct ctr));
      if(c==(struct ctr *)NULL) ExIT();
      memset(c,'\0',sizeof(struct ctr));
      return(c);
}

/* konstantide tabeli moodustamine */
int make_CT(void){
      int t;
      struct ctr *c;
      int i;
      if(ktl==0) return(0);
      fprintf(Logi,"<TABLE BORDER=1><TR>"); fflush(Logi);
      for(i=0;i<ktl;i++){
            t=KT[i];
            c=make_ctr();
            c->nr=t;
            c->value=atoi(T[c->nr]);
            c->loc=i;
            CT[i]=c;
            fprintf(Logi,"<TD>c%d=%d</TD>",i+1,c->value); fflush(Logi);
            }
      fprintf(Logi,"</TR></TABLE>"); fflush(Logi);
      return(ktl);
}

/* mälueraldus identifikaatorite tabeli kirjele */
struct itr *make_itr(void){
      struct itr *c;
      c=(struct itr *)malloc(sizeof(struct itr));
      if(c==(struct itr *)NULL) ExIT();
      memset(c,'\0',sizeof(struct itr));
      c->t=(struct top *)NULL;
      return(c);
}
```

```
/* identifikaatorite tabeli moodustamine */
int make_IDT(void){
      int t;
      struct itr *c;
      int i;
      if(itl==0) return(0);
      fprintf(Logi,"<TABLE BORDER=1><TR>"); fflush(Logi);
      for(i=0;i<itl;i++){
            t=IT[i];
            c=make_itr();
            c->nr=t;
            c->loc=i;
            IDT[i]=c;
            fprintf(Logi,"<TD>i%d=%s</TD>",i+1,T[c->nr]); fflush(Logi);
            }
      fprintf(Logi,"</TR></TABLE>");
      return(itl);
}

/* identifikaatorite tabeli kirje trükk */
void p_itr(struct itr *id){
      fprintf(Logi,"ITR nr=%d V=%s loc=%d value=%d t=%p<BR>",
            id->nr,T[id->nr],id->loc,id->value,id->t);
}

/* muutujate väljatrükk */
void print_variables(void){
      struct itr *id;
      int i;
      set_show("THE VARIABLES:");
      for(i=0;i<itl;i++){
            id=IDT[i];
            if(id->t==(struct top *)NULL)
            fprintf(Logi,"%s=%d<BR>",T[id->nr],id->value);
            }
}

/* konstantide tabeli kirje trükk */
void p_ctr(struct ctr *id){
      fprintf(Logi,"CTR nr=%d V=%s loc=%d value=%d<BR>",
            id->nr,T[id->nr],id->loc,id->value);
}

/* märgendi trükk */
void p_label(struct top *p){
      if(p->label>0) fprintf(Logi,"%s: ",T[p->label]);
}

/* märgendatud operaatorite otsimine */
int det_label(struct top *P){
      struct top *t,*op,*lab;
      struct itr *id;
      int k,flag=0;
      t=P;
ring:
      if(t==(struct top *)NULL) goto out;
      if(t->sem==labl){
            op=t->right;
            lab=t->down;
            for(k=0;k<itl;k++){
                  if(IT[k]==lab->leks){
```

```
                        id=IDT[k];
                        goto iok;
                        }
                  }
            iok:
            if(id->t!=(struct top *)NULL){
                  fprintf(Logi,"label '%s' repeats<BR>",T[lab->leks]);
                  flag++;
                  goto next;
                  }
            id->t=op;
            fprintf(Logi,"label'%s'is address of the operator
            {%s}(%p)<BR>",T[lab->leks],T[op->kood],op);
            t=op;
            goto ring;
            }
next: t=t->right;
      goto ring;
out:  return(flag);
}

/* suunamiste otsimine */
int det_goto(struct top *t,int flag){
      struct top *down;
      struct itr *id;
      int k;
      if(t!=(struct top *)NULL){
            if(t->sem==suunam){
                  down=t->down;
                  down=down->down;
                  for(k=0;k<itl;k++){
                        if(IT[k]==down->leks){
                              id=IDT[k];
                              goto dok;
                              }
                        }
dok:              if(id->t==(struct top *)NULL){
                        if(logi==1)
                        fprintf(Logi,"label '%s' is lack<BR>",T[id->nr]);
                        flag++;
                        }
                  }
            flag+=det_goto(t->down,flag);
            flag+=det_goto(t->right,flag);
            }
      return(flag);
}

/* märgendatud operaatorite otsimine, märgendi-alampuu eemaldamine */
void set_label(struct top *P){
      struct top *t,*op,*lab,*prev;
      struct itr *id;
      int k;
      t=prev=P;
ring: if(t!=(struct top *)NULL){
            if(t->sem==labl){
                  op=t->right;
                  lab=t->down;
                  for(k=0;k<itl;k++){
                        if(IT[k]==lab->leks){
                              id=IDT[k];
```

```
                              goto iok;
                              }
                        }
                  iok: id->t=op;
                  prev->right=op;
                  op->label=lab->leks;
                  free(t);
                  free(lab);
                  t=op;
                  goto ring;
                  }
            prev=t;
            t=t->right;
            goto ring;
      }
}

void ctree(void){
      int f;
      if(ktl>0){
            set_show("Table of constants"); fflush(Logi);
            make_CT();
            }
      if(itl>0){
            set_show("Table of identifiers"); fflush(Logi);
            make_IDT();
            fprintf(Logi,"<BR>");
            f=0;
            f=det_label(p_->down);
            det_goto(p_->down,f);
            if(f==0){
                  set_label(p_->down);
                  set_show("Modified tree");
                  pp2html(p_);
                  }
            else{
                  flag=1;
                  p_=(struct top *)NULL;
                  }
            }
}

/* magasini elemendi mälueraldus */
struct item *make_item(void){
      struct item *c;
      c=(struct item *)malloc(sizeof(struct item));
      if(c==NULL) ExIT();
      memset(c,'\0',sizeof(struct item));
      return(c);
}

/* magasini mälu vabastamine */
void freestack(int i,int k){
      struct item *c;
      while(k>0){
            c=stack[i-k];
            free(c);
            stack[i-k]=(struct item *)NULL;
            k--;
            }
}
```

```
/* magasini trükk */
void prist(int i,int ic){
      struct item *s;
      struct ctr *c;
      struct itr *id;
      int k;
      fprintf(Logi,"<BR><TABLE BORDER=1><B><TR>");
      fprintf(Logi,"<TD><FONT COLOR=\"#0000FF\"><STRONG>Stack<FONT></TD>");
      for(k=0;k<i;k++){
        s=stack[k];
          switch(s->liik){
            case 0: fprintf(Logi,"<TD>tm=%d</TD>",s->index); break;
            case 1: id=s->id;
              if(ic==0){
                fprintf(Logi,"<TD>id %s=%d</TD>",T[id->nr],id->value);
                }
               else{
                fprintf(Logi,"<TD>id %s</TD>",T[id->nr]);
                }
            break;
          case 2: c=s->c;
            fprintf(Logi,"<TD>const=%d</TD>",c->value);
            break;
          case 3: id=s->id;
            fprintf(Logi,"<TD>%s:</TD>",T[id->nr]);
            break;
            }
      }
      fprintf(Logi,"</TR></TABLE>");
}

/* anna muutuja, konstandi või töömuutuja väärtus  */
int get_x(int i,int k){
      struct item *s;
      struct ctr *c;
      struct itr *id;
      int x=0;
      s=stack[i-k];
      switch(s->liik){
            case 0:{
                  x=s->index;
                  break;
                  }
            case 1:{
                  id=s->id;
                  x=id->value;
                  break;
                  }
            case 2:{
                  c=s->c;
                  x=c->value;
                  break;
                  }
            }
      return(x);
}

/* salvesta muutuja väärtus */
void write_x(int x,int i){
      struct item *s;
```

```
      struct itr *id;
      s=stack[i-2];
      id=s->id;
      id->value=x;
      fprintf(Logi,"<BR>%s:=%d<BR>",T[id->nr],id->value);
}

/* aritmeetiliste tehete täitja */
void aritm(int i,int r){
struct item *s;
int x,y,res=0;
x=get_x(i,2);
y=get_x(i,1);
s=make_item( );
s->liik=0;
fprintf(Logi,"<BR>");
switch(r){
      case jag:    res=x/y; fprintf(Logi,"%d = %d / %d",res,x,y); break;
      case korrut: res=x*y; fprintf(Logi,"%d = %d * %d",res,x,y); break;
      case lahut:  res=x-y; fprintf(Logi,"%d = %d - %d",res,x,y); break;
      case liit:   res=x+y; fprintf(Logi,"%d = %d + %d",res,x,y); break;
      }
fprintf(Logi,"<BR>");
s->index=res;
freestack(i,2);
stack[i-2]=s;
}

/* võrdlustehete täitja */
void loogik(int i,int r){
struct item *s;
int x,y;
int res=0;       /* false */
x=get_x(i,2);
y=get_x(i,1);
s=make_item( );
s->liik=0;
fprintf(Logi,"<BR>");
switch(r){
      case pisem:  if(x<y)  res=1; fprintf(Logi,"%d < %d ?",x,y); break;
      case suurem: if(x>y)  res=1; fprintf(Logi,"%d > %d ?",x,y); break;
      case piv:    if(x<=y) res=1; fprintf(Logi,"%d <= %d ?",x,y); break;
      case suv:    if(x>=y) res=1; fprintf(Logi,"%d >= %d ?",x,y); break;
      case pov:    if(x!=y) res=1; fprintf(Logi,"%d /= %d ?",x,y); break;
      case vord:   if(x==y) res=1; fprintf(Logi,"%d = %d ?",x,y); break;
      }
fprintf(Logi,"<BR>");
s->index=res;
freestack(i,2);
stack[i-2]=s;
}

void tprop(int s){
      switch(s){
            case 0:       fprintf(Logi,"root"); break;
            case pisem:   fprintf(Logi,"<"); break;
            case suurem:  fprintf(Logi,">"); break;
            case piv:     fprintf(Logi,"<="); break;
            case suv:     fprintf(Logi,">="); break;
            case pov:     fprintf(Logi,"/="); break;
            case vord:    fprintf(Logi,"="); break;
```

```
            case omist:   fprintf(Logi,":="); break;
            case jag:     fprintf(Logi,"/"); break;
            case korrut:  fprintf(Logi,"*"); break;
            case lahut:   fprintf(Logi,"-"); break;
            case liit:    fprintf(Logi,"+"); break;
            case suunam:  fprintf(Logi,"GOTO "); break;
            case kuisiis: fprintf(Logi,"IF "); break;
            case tingop:  fprintf(Logi," "); break;
            case lugem:   fprintf(Logi,"READ "); break;
            case kirjut:  fprintf(Logi,"WRITE "); break;
            case labl:    fprintf(Logi,"label"); break;
            default:      fprintf(Logi,"%d",s); break;
            }
}

int print_Op(struct top *t){
      int fg=0;
      if(t!=(struct top *)NULL){
            if(t->sem==labl){
                  t=t->down;
                  fprintf(Logi,"%s: ",T[t->leks]);
                  return(0);
                  }
            if(t->sem==lugem||t->sem==kirjut){
                  tprop(t->sem);
                  t=t->down;
                  fprintf(Logi,"%s",T[t->leks]);
                  return(0);
                  }
            if(t->sem==suunam){
                  tprop(t->sem);
                  t=t->down;
                  t=t->down;
                  fprintf(Logi," %s",T[t->leks]);
                  return(0);
                  }
            if(t->sem==kuisiis){
                  fprintf(Logi,"IF ");
                  print_Op(t->down);
                  fprintf(Logi," THEN ");
                  return(0);
                  }
            if(t->down!=(struct top *)NULL&&t->sem!=omist){
                  fprintf(Logi,"( ");
                  fg=1;
                  }
            print_Op(t->down);
            if(t->leks!=0) fprintf(Logi,"%s",T[t->leks]);
            else tprop(t->sem);
            t=t->down;
            if(t!=(struct top *)NULL){
                  print_Op(t->right);
                  if(fg==1) fprintf(Logi,")");
                  }
            }
      return(0);
}


/* TRIGOL-keelsete programmide trükk puu järgi  */
void p_prog(struct top *root){
```

```
      struct top *t;
      t=root->down;
      fprintf(Logi,"</FONT># ");
naaber:     print_Op(t);
      if(t->sem!=kuisiis&&t->sem!=labl&&t->up!=root) ps();
      t=t->right;
      if(t==(struct top *)NULL) goto ok;
      goto naaber;
ok:   fprintf(Logi,"#<BR>");
}

/* TRIGOL-keelsete programmide interpretaator  */
void trigol_Int(struct top *root){
      struct item *s;
      struct top *t;
      struct ctr *c;
      struct itr *id;
      int op;
      int i=0;
      int j,k,x;
      t=root->down;
      for(j=0;j<20;j++) stack[j]=(struct item *)NULL;
      fprintf(Logi,"<HR>");
down: p_label(t);
      if(t->down!=(struct top *)NULL){
            t=t->down;
            goto down;
            }
/* leht => stack  */
      s=make_item();
      op=t->sem;
      switch(op){
            case muut:{
                  s->liik=1;                        /* muutuja */
                  for(k=0;k<itl;k++){
                        if(IT[k]==t->leks){
                              id=IDT[k];
                              goto iok;
                              }
                        }
                  iok: s->id=id;
                  if(id->t!=(struct top *)NULL){
                        s->liik=3;                /* märgend */
                        s->t=id->t;
                        }
                  break;
                  }
            case konst:{
                  for(k=0;k<ktl;k++){
                        if(KT[k]==t->leks){
                              c=CT[k];
                              goto cok;
                              }
                        }
                  cok: s->c=c;
                  s->liik=2;                        /* konstant */
                  break;
                  }
            default:{
                  fprintf(Logi,"parsing tree is incorrect<BR>");
                  goto jokk;
```

```
                }
            }
      stack[i]=s;
      i++;
      prist(i,0);
naaber:     if(t->up==(struct top *)NULL){
            t=t->right;
            goto down;
            }
      t=t->up;
/*    p_top("up",t);  */
      if(t==root) goto ok;
      fprintf(Logi,"</FONT>interpreting the operator  ");
      print_Op(t);
      pp2html(t);
      op=t->sem;
      switch(op){
            case  pisem:{              /* < */
                  loogik(i,op);
                  i-=1;
                  break;
                  }
            case  suurem:{              /* > */
                  loogik(i,op);
                  i-=1;
                  break;
                  }
            case  piv:{               /* <= */
                  loogik(i,op);
                  i-=1;
                  break;
                  }
            case  suv:{               /* >= */
                  loogik(i,op);
                  i-=1;
                  break;
                  }
            case  pov:{               /* /= */
                  loogik(i,op);
                  i-=1;
                  break;
                  }
            case  vord:{               /* = */
                  loogik(i,op);
                  i-=1;
                  break;
                  }
            case omist:{              /* omistamine */
                  x=get_x(i,1);
                  write_x(x,i);
                  freestack(i,2);
                  i-=2;
                  break;
                  }
            case jag:{              /* jagamine */
                  aritm(i,op);
                  i-=1;
                  break;
                  }
            case korrut:{              /* korrutamine */
                  aritm(i,op);
```

```
                  i-=1;
                  break;
                  }
            case lahut:{              /* lahutamine */
                  aritm(i,op);
                  i-=1;
                  break;
                  }
            case liit:{              /* liitmine */
                  aritm(i,op);
                  i-=1;
                  break;
                  }
            case suunam:{               /* suunamine */
                  s=stack[i-1];
                  t=s->t;
                  id=s->id;
                  if(logi==1) fprintf(Logi,"goto %s<BR>",T[id->nr]);
                  freestack(i,1);
                  i-=1;
                  goto down;
                  }
            case kuisiis:{
                  x=get_x(i,1);
                  freestack(i,1);
                  i-=1;
                  if(x==0){
                        t=t->right;
                        goto naaber;
                        }
                  break;
                  }
            case tingop: break;
            case lugem:{
                  s=stack[i-1];
                  id=s->id;
                  printf("\aInput %s= ",T[id->nr]);
                  scanf("%d",&x);
                  if(logi==1) fprintf(Logi,"Input %s=%d<BR>",T[id->nr],x);
                  id->value=x;
                  freestack(i,1);
                  i-=1;
                  break;
                  }
            case kirjut:{
                  s=stack[i-1];
                  id=s->id;
                  printf("Output %s=%d<BR>",T[id->nr],id->value);
                  fprintf(Logi,"Output %s=%d<BR>",T[id->nr],id->value);
                  freestack(i,1);
                  i-=1;
                  break;
                  }
            }
      prist(i,0);
      goto naaber;
ok:   fprintf(Logi,"program %s is completed<BR>",pr_name);
jokk:
      print_variables();
}
```

```
struct top *analyzer(void){
      if(r_tabs()==0) return(0);
      set_show("Program");
      p_prog(p_);
      set_show("Parsing tree");
      pp2html(p_);
      ctree( );
      return(p_);
}

int itr(void){
      int ret=0; time_t t0;
      GBuf=NULL;
      Logi=fopen(L_name,"w");
      if(Logi==NULL){
            printf("Cannot open log-book<BR>");
            return(0);
            }
      m_k=4; k_k=11;
      logi=1; time(&t0);
      fprintf(Logi,
            "<HTML><HEAD><TITLE>Interpreter</TITLE></HEAD><BODY><B>");
      fprintf(Logi,
      "<FONT COLOR=\"#0000FF\"><H3>Start of Interpreter for program %s",
            pr_name);
      fprintf(Logi," at %s<BR>",asctime(localtime(&t0)));
      if(analyzer()!=NULL) trigol_Int(p_);
      time(&t0);
      fprintf(Logi,
            "<FONT COLOR=\"#0000FF\"><H4>Interpreter ended at/FONT> ");
      fprintf(Logi,"%s</H4>",asctime(localtime(&t0)));
      fprintf(Logi,"</BODY></HTML>");
      return(1);
}

int main(int argc,char **argv){
      if(argc!=2){
            printf("I do need only the name of a tri-program\n"); abort();
            }
      pr_name=(char *)malloc(256);
      Pr_name=(char *)malloc(256);
      L_name=(char *)malloc(256);
      Nimi=(char *)malloc(256);
      memset(pr_name,'\0',256);
      memset(Pr_name,'\0',256);
      memset(L_name,'\0',256);
      memset(Nimi,'\0',256);
      strcpy(Nimi,".tri");
      sprintf(pr_name,"%s",argv[1]);
      strcpy(L_name,pr_name);
      strcpy(Pr_name,pr_name);
/*    p=strrchr(L_name,'.');
      p[1]='\0';  */
      strcat(L_name,"i.htm");
      strcat(pr_name,".tri");
      printf("pr_name=%s L_name=%s\n",pr_name,L_name);
      itr( );
      return(1);
}
```

Lisa 9. Trigoli kompilaator

```
//cmp32.c :: 32-bitine masm32 64-bitisele masinale.  02.11.11
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
#include <sys/stat.h>
#include "two32.h"

FILE *maketri=NULL;

//        S E M A N T I C S

#define muut 1      /* #i# - identifikaator */
#define konst 2     /* #c# - konstant */
#define pisem 3     /* < */
#define suurem 4    /* > */
#define piv 5       /* <= */
#define suv 6       /* >= */
#define pov 7       /* /= */
#define vord 8      /* = */
#define omist 10    /* omistamisoperaator */
#define jag 11      /* jagamistehe */
#define korrut 12   /* korrutamistehe */
#define lahut 13    /* lahutustehe */
#define liit 14     /* liitmistehe */
#define labl 15     /* märgend */
#define suunam 16   /* suunamisoperaator */
#define kuisiis 18  /* if-lause */
#define tingop 19   /* tingimusoperaator */
#define lugem 20    /* lugemisoperaator */
#define kirjut 21   /* kirjutamisoperaator */

/* interpretaatori magasinielement */
struct item{
      int liik;         /* 0 töömuutuja, 1 id, 2 c, 3 top */
      int index;       /* töömuutuja väärtus  */
      struct itr *id;   /* liik = 1 (muutuja)  */
      struct ctr *c;    /* liik = 2 (konstant) */
      struct top *t;    /* liik = 3 (märgend)  */
      };

  struct item *stack[20];   /* kompilaatori magasin */
  int IX;
  int tmarv;  //töömuutujate arv
  int Label;
  int opt;
  int nado=0;     //omistamine: 0: laadi value -> eax  1: value ON eax-s
                //sama aritmeetilistes avaldistes, tipa a+2+b/tsee

void blue(char *t){
      fprintf(Logi,"<FONT COLOR=\"0000FF\">%s</FONT>",t);
}


void green(char *t){
      fprintf(Logi,"<FONT COLOR=\"008000\">%s</FONT>",t);
```

```
}

void red(char *t){
      fprintf(Logi,"<FONT COLOR=\"FF0000\">%s</FONT>",t);
}

void pset(int x,int y){
      blue("{");
      fprintf(Logi,"%s",T[x]);
      blue(",");
      fprintf(Logi,"%s",T[y]);
      blue("}");
}

void print_top(struct top *t){
fprintf(Logi," top %p: kood=%d leks=%d sem=%d label=%d truel=%d falsel=%d
up=%p right=%p down=%p",
   t,t->kood,t->leks,t->sem,t->label,t->truel,t->falsel,t->up,t->right,
   t->down);
fprintf(Logi,"  (%s)<BR>",(t->leks==0) ? T[t->kood] : T[t->leks]);
if(t->right!=(struct top *)NULL) print_top(t->right);
if(t->down!=(struct top *)NULL) print_top(t->down);
}

void print_Itr(struct itr *t){
fprintf(Logi," itr %p: nr=%d loc=%d value=%d top=%p io=%d tio=%d<BR>",
   t,t->nr,t->loc,t->value,t->t,t->io,t->tio);
fprintf(Logi,"   (%s)<BR>",T[t->nr]);
}

void print_Ctr(struct ctr *t){
fprintf(Logi," ctr %p: nr=%d loc=%d value=%d",
   t,t->nr,t->loc,t->value);
fprintf(Logi,"   (%s)<BR>",T[t->nr]);
}

void print_one_top(struct top *t){
fprintf(Logi," top %p: kood=%d leks=%d sem=%d label=%d truel=%d falsel=%d
up=%p right=%p down=%p",
   t,t->kood,t->leks,t->sem,t->label,t->truel,t->falsel,t->up,t->right,
   t->down);
fprintf(Logi,"  (%s)<BR>",(t->leks==0) ? T[t->kood] : T[t->leks]);
}

/* analyysipuu tipu trükk */
void p_top(char *t,struct top *p){
      fprintf(Logi,"%s top:<BR>",t);
      print_one_top(p);
}

void print_item(int k,struct item *s){
fprintf(Logi," k=%d) item %p: liik=%d index=%d id=%p c=%p t=%p<BR>",
   k,s,s->liik,s->index,s->id,s->c,s->t);
}

void set_show(char *title){
      fprintf(Logi,"<H4><FONT COLOR=\"#0000FF\">%s",title);
      fprintf(Logi,"</FONT></H4>");
}

void ps(void){
```

```
      fprintf(Logi,"<BR>");
}

void ExIT(void){
      if(logi==1){
            fprintf(Logi,"Abnormal end<BR><BR>");
            fflush(Logi);
            fclose(Logi);
            }
      abort();
}

FILE *opr(void){
      of=fopen(rida,"rb");
      if (of==NULL){
            fprintf(Logi,"cannot open %s. Constructor?",rida);
            return(of);
            }
      return(of);
}

struct top *r_ptree(void){
      struct top *p;
      p=(struct top *)malloc(sizeof(struct top));
      if(p==(struct top *)NULL){
            fprintf(Logi,"I haven't memory enough..");
            ExIT();
            }
      fread(p,sizeof(struct top),1,of);
      if(p->right!=(struct top *)NULL) p->right=r_ptree();
      if(p->down!=(struct top *)NULL) p->down=r_ptree();
      return(p);
}

void set_up(struct top *root){
      struct top *p;
      p=root;
      if(p->down!=(struct top *)NULL){
            p=p->down;
            while(p->right!=(struct top *)NULL){
                  p=p->right;
                  }
            p->up=root;
            }
      if(root->down!=(struct top *)NULL) set_up(root->down);
      if(root->right!=(struct top *)NULL) set_up(root->right);
}

int r_tree(void){
      sprintf(rida,"%s.pt",Pr_name);
      if(opr()!=NULL){
            p_=r_ptree();
            set_up(p_);
            return(1);
            }
      return(0);
}


int r_t(void){
      sprintf(rida,"%s.t",Pr_name);
```

```
      if(opr()!=NULL){
            fread(&T,20,nr+1,of);
            fclose(of);
            return(1);
            }
      else return(0);
}

int r_parm(void){
      PARM=(struct parm *)malloc(sizeof(struct parm));
      if(PARM==(struct parm *)NULL) ExIT();
      memset(PARM,'\0',sizeof(struct parm));
      sprintf(rida,"%s.prm",Pr_name);
      if(opr()!=NULL){
            fread(PARM,sizeof(struct parm),1,of);
            fclose(of);
            nr=PARM->nr;
        tnr=PARM->tnr;
            itl=PARM->itl;
            ktl=PARM->ktl;
            return(1);
            }
      return(0);
}

int r_it(void){
      sprintf(rida,"%s.it",Pr_name);
      if(opr()!=NULL){
            fread(&IT,itl*sizeof(int),1,of);
            fclose(of);
            return(1);
            }
      return(0);
}

int r_kt(void){
      sprintf(rida,"%s.kt",Pr_name);
      if(opr()!=NULL){
            fread(&KT,ktl*sizeof(int),1,of);
            fclose(of);
            return(1);
            }
      return(0);
}

int r_tabs(void){
      int flag;
      flag=1;
      flag=r_parm();
      if(flag==0) goto out;
      flag=r_t();
      if(flag==0) goto out;
      flag=r_it();
      if(flag==0) goto out;
      flag=r_kt();
      if(flag==0) goto out;
      flag=r_tree();
      fflush(Logi);
out:    if(flag==0){
            fprintf(Logi,"cannot read all the tables..");
            fflush(Logi);
```

```
            }
      return(flag);
}

int print_Op(struct top *t){
      int fg=0;
      if(t!=(struct top *)NULL){
            if(t->sem==labl){
                  t=t->down;
                  fprintf(Logi,"%s: ",T[t->leks]);
                  return(0);
                  }
            if(t->sem==lugem||t->sem==kirjut){
                  tprop(t->sem);
                  t=t->down;
                  fprintf(Logi,"%s",T[t->leks]);
                  return(0);
                  }
            if(t->sem==suunam){
                  tprop(t->sem);
                  t=t->down;
                  t=t->down;
                  fprintf(Logi," %s",T[t->leks]);
                  return(0);
                  }
            if(t->sem==kuisiis){
                  fprintf(Logi,"IF ");
                  print_Op(t->down);
                  fprintf(Logi," THEN ");
                  return(0);
                  }
            if(t->down!=(struct top *)NULL&&t->sem!=omist){
                  fprintf(Logi,"(");
                  fg=1;
                  }
            print_Op(t->down);
            if(t->leks!=0) fprintf(Logi,"%s",T[t->leks]);
            else tprop(t->sem);
            t=t->down;
            if(t!=(struct top *)NULL){
                  print_Op(t->right);
                  if(fg==1) fprintf(Logi,")");
                  }
            }
      return(0);
}


/* TRIGOL-keelsete programmide trükk puu järgi  */
void p_prog(struct top *root){
      struct top *t;
      t=root->down;
      fprintf(Logi,"</FONT># ");
naaber:     print_Op(t);
      if(t->sem!=kuisiis&&t->sem!=labl&&t->up!=root) ps();
      t=t->right;
      if(t==(struct top *)NULL) goto ok;
      goto naaber;
ok:   fprintf(Logi,"#<BR>");
}
```

```
/* märgendatud operaatorite otsimine */
int det_label(struct top *P){
      struct top *t,*op,*lab;
      struct itr *id;
      int k,flag=0;
      t=P;
ring:
      if(t==(struct top *)NULL) goto out;
      if(t->sem==labl){
            op=t->right;
            lab=t->down;
            for(k=0;k<itl;k++){
                  if(IT[k]==lab->leks){
                        id=IDT[k];
                        goto iok;
                        }
                  }
            iok:
            if(id->t!=(struct top *)NULL){
                  fprintf(Logi,"label '%s' repeats<BR>",T[lab->leks]);
                  flag++;
                  goto next;
                  }
            id->t=op;
            fprintf(Logi,"label '%s' is address of the operator {%s}
(%p)<BR>",
                    T[lab->leks],T[op->kood],op);
            t=op;
            goto ring;
            }
next: t=t->right;
      goto ring;
out:  return(flag);
}

/* märgendatud operaatorite otsimine, märgendi töötlus */
void set_label(struct top *P){
      struct top *t,*op,*lab,*prev;
      struct itr *id;
      int k;
      t=prev=P;
ring: if(t!=(struct top *)NULL){
            if(t->sem==labl){
                  op=t->right;
                  lab=t->down;
                  for(k=0;k<itl;k++){
                        if(IT[k]==lab->leks){
                              id=IDT[k];
                              goto iok;
                              }
                        }
                  iok: id->t=op;
                  prev->right=op;
                  op->label=lab->leks;
                  free(t);
                  free(lab);
                  t=op;
                  goto ring;
                  }
            prev=t;
            t=t->right;
```

```
            goto ring;
      }
}

void ctree(void){
      int f;
      if(ktl>0){
            blue("<BR>Table of constants<BR>");
            make_CT();
            }
      if(itl>0){
            blue("<BR>Table of identifiers<BR>");
            make_IDT();
            f=0;
            f=det_label(p_->down);
            if(f==0) set_label(p_->down);
            }
}

void tprop(int s){
      switch(s){
            case 0:       fprintf(Logi,"root"); break;
            case pisem:   fprintf(Logi,"<"); break;
            case suurem:  fprintf(Logi,">"); break;
            case piv:     fprintf(Logi,"<="); break;
            case suv:     fprintf(Logi,">="); break;
            case pov:     fprintf(Logi,"/="); break;
            case vord:    fprintf(Logi,"="); break;
            case omist:   fprintf(Logi,":="); break;
            case jag:     fprintf(Logi,"/"); break;
            case korrut:  fprintf(Logi,"*"); break;
            case lahut:   fprintf(Logi,"-"); break;
            case liit:    fprintf(Logi,"+"); break;
            case suunam:  fprintf(Logi,"GOTO "); break;
            case kuisiis: fprintf(Logi,"IF "); break;
            case tingop:  fprintf(Logi," "); break;
            case lugem:   fprintf(Logi,"READ "); break;
            case kirjut:  fprintf(Logi,"WRITE "); break;
            case labl:    fprintf(Logi,"label"); break;
            default:      fprintf(Logi,"%d",s); break;
            }
}

int pp2html(struct top *p) {
      double pr;
      int n=1;
      struct top *t;
      char s[20];
      t=p;
naaber:
      if(t->right!=NULL){
            n++;
            t=t->right;
            goto naaber;
            }
      pr=100.0/n;
      sprintf(s,"\"%2.0f&#037\"",pr);
      fprintf(Logi,"<table border=1><tr>");
next:
      fprintf(Logi,
      "<td width=%s valign=\"top\"><STRONG>",s);
```

```
      if((p->kood==m_k)||(p->kood==k_k)) fprintf(Logi,"%s",T[p->leks]);
      else tprop(p->sem);
      if(p->down!=NULL) pp2html(p->down);
      fprintf(Logi,"</td>");
      if(p->right!=NULL){
            p=p->right;
            goto next;
            }
      fprintf(Logi,"</tr></table>");
      return 1;
}

/* mälueraldus konstantide tabeli kirjele */
struct ctr *make_ctr(void){
      struct ctr *c;
      c=(struct ctr *)malloc(sizeof(struct ctr));
      if(c==(struct ctr *)NULL) ExIT();
      memset(c,'\0',sizeof(struct ctr));
      return(c);
}

/* konstantide tabeli moodustamine */
int make_CT(void){
      int t;
      struct ctr *c;
      int i;
      if(ktl==0) return(0);
      fprintf(Logi,"<TABLE BORDER=1><TR>");
      for(i=0;i<ktl;i++){
            t=KT[i];
            c=make_ctr();
            c->nr=t;
            c->loc=i;
            c->value=atoi(T[c->nr]);
            CT[i]=c;
            fprintf(Logi,"<TD>c%d=%d</TD>",i+1,c->value);
            }
      fprintf(Logi,"</TR></TABLE>");
      return(ktl);
}

/* mälueraldus identifikaatorite tabeli kirjele */
struct itr *make_itr(void){
      struct itr *c;
      c=(struct itr *)malloc(sizeof(struct itr));
      if(c==(struct itr *)NULL) ExIT();
      memset(c,'\0',sizeof(struct itr));
      c->t=(struct top *)NULL;
      return(c);
}

/* identifikaatorite tabeli moodustamine */
int make_IDT(void){
      int t;
      struct itr *c;
      int i;
      if(itl==0) return(0);
      fprintf(Logi,"<TABLE BORDER=1><TR>");
      for(i=0;i<itl;i++){
            t=IT[i];
            c=make_itr();
```

```
            c->nr=t;
            c->loc=i;
            IDT[i]=c;
            fprintf(Logi,"<TD>i%d=%s</TD>",i+1,T[c->nr]);
            }
      fprintf(Logi,"</TR></TABLE>");
      return(itl);
}

/* identifikaatorite tabeli kirje trükk */
void p_itr(struct itr *id){
      fprintf(Logi,"ITR nr=%d V=%s loc=%d value=%d t=%p<BR>",
            id->nr,T[id->nr],id->loc,id->value,id->t);
}

/* muutujate väljatrükk */
void print_variables(void){
      struct itr *id;
      int i;
      set_show("THE VARIABLES:");
      for(i=0;i<itl;i++){
            id=IDT[i];
            if(id->t==(struct top *)NULL)
                  fprintf(Logi,"%s=%d<BR>",T[id->nr],id->value);
            }
}

/* konstantide tabeli kirje trükk */
void p_ctr(struct ctr *id){
      fprintf(Logi,"CTR nr=%d V=%s loc=%d value=%d<BR>",
            id->nr,T[id->nr],id->loc,id->value);
}

/* magasini elemendi mälueraldus */
struct item *make_item(void){
      struct item *c;
      c=(struct item *)malloc(sizeof(struct item));
      if(c==NULL) ExIT();
      memset(c,'\0',sizeof(struct item));
      return(c);
}

/* magasini mälu vabastamine */
void freestack(int i,int k){
      struct item *c;
      while(k>0){
            c=stack[i-k];
            free(c);
            stack[i-k]=(struct item *)NULL;
            k--;
            }
}

/* luurab töömuutujate arvu (tmarv), töötleb if-lause puud */
int det_nrlv(struct top *root){
      struct top *t,*t1,*t2,*t3,*t4;
      struct itr *id;
      int k;
      char st[20];
      int op;
      int tm=0;
```

```
      int N=0;  /* maks. töömuutuja number */
      int i=0;  /* magasiniindeks */
      IX=1;     /* töömärgendi nr */
      t=root->down;
      memset(st,'\0',20);
down:
      if(t->down!=(struct top *)NULL){
            t=t->down;
            goto down;
            }
/* leht => stack  */
leht:
      op=t->sem;
      switch(op){
            case muut:{                              /* muutuja */
                  st[i]=1;
                  for(k=0;k<itl;k++){
                        if(IT[k]==t->leks){
                              id=IDT[k];
                              goto iok;
                              }
                        }
                  iok:
                  if(id->t!=(struct top *)NULL){
                        st[i]=3;              /* märgend */
                        }
                  break;
                  }
            case konst:{
                  st[i]=2;                        /* konstant */
                  break;
                  }
            default: red("error in the parsing tree");
                   return(-1);
            }
      i++;
naaber:
      if(t->up==(struct top *)NULL){
            t=t->right;
            if(t->down!=(struct top *)NULL)    goto down;
            else goto leht;
            }
      t=t->up;
      if(t==root) goto ok;
      op=t->sem;
      if((op>=pisem)&&(op<=vord)){              /* võrdlustehe */
            if(st[i-2]==4) tm--;
            if(st[i-1]==4) tm--;
            i-=2;
            goto naaber;
            }
      if((op>=jag)&&(op<=liit)){            /* aritmeetika */
                  if(st[i-2]==4) tm--;
                  if(st[i-1]==4) tm--;
                  st[i-2]=4;     /* tmuutuja */
                  tm++;
                  if(tm>N) N=tm;
                  i-=1;
                  goto naaber;
                  }
      switch(op){
```

```
case omist:{              /* omistamine */
      if(st[i-2]==4) tm--;
      if(st[i-1]==4) tm--;
      i-=2;
      break;
      }
case suunam:{               /* suunamine */
      i-=1;
      break;
      }
case kuisiis:{          /* if-lause */
      t1=t->down;     /* ]rdlemine */
      t2=t->right;    /* IFi naaber */
      if(t2->sem==suunam){
            t3=t2->down;
            t4=t3->down;
            t1->truel=t4->leks;
            free(t3);
            free(t4);
            t->up=t2->up;
            t->right=t2->right;
            free(t2);   /* eemaldan 'goto' */
            }
      else{
            t3=t2->right;
            t1->truel=IX;
            t2->label=IX;
            IX++;
            t1->falsel=IX;
            t3->label=IX;
            IX++;
            }
      break;
      }
case tingop: break;
case lugem:{
      t1=t->down;
      for(k=0;k<itl;k++){
            if(IT[k]==t1->leks){
                  id=IDT[k];
                  goto lok;
                  }
            }
      lok:
      id->io=id->io||1;
      Rd=1;
      i-=1;
      break;
      }
case kirjut:{
      t1=t->down;
      for(k=0;k<itl;k++){
            if(IT[k]==t1->leks){
                  id=IDT[k];
                  goto kok;
                  }
            }
      kok:
      id->io=id->io||2;
      Wr=1;
      i-=1;
```

```
                  break;
                  }
            }
      goto naaber;
ok:   return(N);
}

/* tekstimassiivi sisestamine kettalt */
char *jarray(char *pealkiri){
      FILE *tekst=NULL;
      char *B;
      char c;
      int  i;
      int  k;
      struct stat *buf;
      buf=(struct stat *)malloc(sizeof(struct stat));
      if(buf==NULL){
            fprintf(Logi,"r_text: I haven't %d bytes of memory..<BR>",
                  sizeof(struct stat));
            ExIT();
            }
      tekst = fopen(pealkiri, "r");
      if (tekst==NULL){
            fprintf(Logi,"cannot find the file  %s <BR>",pealkiri);
            ExIT();
            }
      if(stat(pealkiri,buf)==-1){
            fprintf(Logi,"r_text: stat failed<BR>");
            ExIT();
            }
      k=buf->st_size;
      B = (char *)malloc(k+10);
      if(B == NULL){
            fprintf(Logi,"I don't have memory enaugh..");
            ExIT();
            }
      memset(B,'\0',k+10);
/* fill buffer */
      rewind(tekst);
      i=0;
      while (!feof(tekst)){
            c=fgetc(tekst);
            B[i]=c;
            i++;
            }
      P_length=i;
      for(i=P_length;i>0;i--){
            if(isgraph(B[i])){
                  i++;
                  B[i]='\n';
                  i++;
                  B[i]='\0';
                  P_length=i;
                  goto out;
                  }
            }
out:  fclose(tekst);
      return(B);
}

int prog(void){
```

```
      PBuf=jarray(pr_name);
      if(PBuf!=NULL){
            Plen=P_length;
            return(1);
            }
      return(0);
}

void Put(char *c){
      fprintf(rules,"%s",c);
      if(c[0]==';') fprintf(Logi,"<FONT COLOR=\"008000\">%s</FONT>",c);
      else fprintf(Logi,"%s",c);
}

void PreOn(void){
      fprintf(Logi,"<CODE><PRE><BIG>");
}

void PreOff(void){
      fprintf(Logi,"</PRE>");
}

void gen_bat(void){
    FILE *maketri=NULL;
    maketri=fopen("maketri.bat","w");
    fprintf(maketri,"@echo off\n");
fprintf(maketri,"if not exist rsrc.rc goto over1 \\masm32\\bin\\rc /v
rsrc.rc \\masm32\\bin\\cvtres /machine:ix86 rsrc.res\n");
    fprintf(maketri,"\n :over1\n");
    fprintf(maketri,"if exist \"%s.obj\" del
\"%s.obj\"\n",Pr_name,Pr_name);
    fprintf(maketri,"if exist \"%s.exe\" del \"%s.exe\"\n",
Pr_name,Pr_name);
    fprintf(maketri,"\\masm32\\bin\\ml /c /coff %s.asm\n", Pr_name);
    fprintf(maketri,"if errorlevel 1 goto errasm\n");
    fprintf(maketri,"if not exist rsrc.obj goto nores\n");
    fprintf(maketri,"\\masm32\\bin\\Link /SUBSYSTEM:CONSOLE \"%s.obj\"
rsrc.res\n",Pr_name);
    fprintf(maketri,"if errorlevel 1 goto errlink\n");
    fprintf(maketri,"goto TheEnd\n");
    fprintf(maketri,":nores\n");
    fprintf(maketri,"\\masm32\\bin\\Link /SUBSYSTEM:CONSOLE
\"%s.obj\"\n",Pr_name);
    fprintf(maketri,"if errorlevel 1 goto errlink\n");
    fprintf(maketri,"dir \"%s.*\"\n",Pr_name);
    fprintf(maketri,"goto TheEnd\n");
    fprintf(maketri,":errlink\n");
    fprintf(maketri," echo _\n");
    fprintf(maketri,"echo Link error\n");
    fprintf(maketri,"goto TheEnd\n");
    fprintf(maketri,":errasm\n");
    fprintf(maketri," echo _\n");
    fprintf(maketri,"echo Assembly Error\n");
    fprintf(maketri,"goto TheEnd\n");
    fprintf(maketri,":TheEnd\n");
    fflush(maketri);
    fclose(maketri);
}

int gen_header(void){
      struct itr *id;
```

```
      int i,j,io=0;
      int id_nr=0;
    gen_bat();
      memset(rida,'\0',80);
      sprintf(rida,"%s.asm",Pr_name);
      rules=fopen(rida, "w");
      if (rules==NULL){
            fprintf(Logi,"cannot create the file %s<BR>",rida);
            fclose(rules);
            return(0);
            };
      blue("<BR><BR>Compiler to Assembler started<BR><BR>");
      if(prog()==0) goto yle;
      green("; gen_header: source text<BR>");
      memset(rida,'\0',80);
      j=0;
      for(i=0;i<Plen;i++){
            rida[j]=PBuf[i];
            j++;
            if(PBuf[i]=='\n'){
                  fprintf(Logi,
                        "<FONT COLOR=\"008000\">; %s<BR></FONT>",rida);
                  fprintf(rules,"; %s",rida);
                  memset(rida,'\0',80);
                  j=0;
                  }
            }
      fprintf(rules,";\n");
      green(";<BR>");
yle:
      PreOn();
      sprintf(rida,"; Program %s.asm\n",Pr_name);
      Put(rida);
      Put("include \\masm32\\include\\masm32rt.inc\n");
    Put(".data\n");
      id_nr=0;
      for(i=0;i<itl;i++){
            id=IDT[i];
        switch(id->io){
            case 0:break;
            case 1:
                io++;
                fprintf(rules,"io%d\tdb\t\"%s=\",0\n",io,T[id->nr]);
                fprintf(Logi,"io%d\tdb\t\"%s=\",0\n",io,T[id->nr]);
                id->tio=io;
                break;
            case 2:
                io++;
                fprintf(rules,"io%d\tdb\t\"%s=\",0\n",io,T[id->nr]);
                fprintf(Logi,"io%d\tdb\t\"%s=\",0\n",io,T[id->nr]);
                id->tio=io;
                break;
            case 3:
                io++;
                fprintf(rules,"io%d\tdb\t\"%s=\",0\n",io,T[id->nr]);
                fprintf(Logi,"io%d\tdb\t\"%s=\",0\n",io,T[id->nr]);
                id->tio=io;
                io++;
                fprintf(rules,"io%d\tdb\t\"%s=\",0\n",io,T[id->nr]);
                fprintf(Logi,"io%d\tdb\t\"%s=\",0\n",io,T[id->nr]);
                id->tio=io;
```

```
                break;
            }
        if(id->t==(struct top *)NULL) id_nr++;
        }
    Put("format1\tdb \"%d\",0\n");

    if(id_nr>0){
            Put(".data?\n");
            sprintf(rida,"; gen_header: # of identifiers=%d<BR>",id_nr);
            green(rida);
            for(i=0;i<itl;i++){
                  id=IDT[i];
                  if(id->t==(struct top *)NULL){
                        sprintf(rida,"%s\tdd\t?\n",T[id->nr]);
                        Put(rida);
                        }
                  }
            }
      if(tmarv>1){
            sprintf(rida,"; gen_header: # of workvariables=%d<BR>",tmarv);
            for(i=0;i<tmarv;i++){
                  sprintf(rida,"dTv%d\tdd\t?\n",i);
                  Put(rida);
                  }
      }
code: green("; gen_header: code segment'll start<BR>");
      Put(".code\nstart:\n");
      return(1);
}

void w_label(int label){
      if(label>0){
            if(label>tnr) sprintf(rida,"%s:",T[label]);
            else sprintf(rida,"MExi%d:",label);
            Put(rida);
            Label=0;
            }
}

/* genereeri aadress */
void w_addr(int i,int k){
      struct item *s;
      struct ctr *c;
      struct itr *id;
      s=stack[i-k];
      switch(s->liik){
            case 0:{
                  sprintf(rida,"dTv%d",s->index);
                  Put(rida);
                  if(IX>0) IX--;
                  break;
                 }
            case 1:{
                  id=s->id;
                  sprintf(rida,"%s",T[id->nr]);
                  Put(rida);
                  break;
                  }
            case 2:{
                  c=s->c;
                  sprintf(rida,"%s",T[c->nr]);
```

```
                  Put(rida);
                  break;
                  }
            case 3:{
                  id=s->id;
                  sprintf(rida,"%s",T[id->nr]);
                  Put(rida);
                  break;
                  }
            case 4:{
                  sprintf(rida,"%d",s->index);
                  Put(rida);
                  if(IX>0) IX--;
                  break;
                  }
            }
}

/* magasini trükk */
void prist(int i,int ic){
      struct item *s;
      struct ctr *c;
      struct itr *id;
      int k;
      fprintf(Logi,"<BR><TABLE BORDER=1><B><TR>");
      fprintf(Logi,"<TD><FONT COLOR=\"#0000FF\"><STRONG>Stack<FONT></TD>");
      for(k=0;k<i;k++){
            s=stack[k];
            switch(s->liik){
                  case 0: fprintf(Logi,"<TD>tm</TD>"); break;
                  case 1: id=s->id;
                        if(ic==0){
                            fprintf(Logi,"<TD>%s</TD>",T[id->nr]);
                            }
                         else{
                            fprintf(Logi,"<TD>%s</TD>",T[id->nr]);
                            }
                        break;
                  case 2: c=s->c;
                        fprintf(Logi,"<TD>%d</TD>",c->value);
                        break;
                  case 3: id=s->id;
                        fprintf(Logi,"<TD>%s:</TD>",T[id->nr]);
                        break;
                  }
            }
      fprintf(Logi,"</TR></TABLE>");
}

struct item *leaf(struct top *t){
      int k,op;
      struct item *s;
      struct itr *id;
      struct ctr *c;
      s=make_item();
      op=t->sem;
      switch(op){
            case muut:{
                  s->liik=1;                        /* muutuja */
                  for(k=0;k<itl;k++){
                        if(IT[k]==t->leks){
```

```
                        id=IDT[k];
                        goto iok;
                        }
                  }
            iok:
            s->id=id;
            if(id->t!=(struct top *)NULL){
                  s->liik=3;                /* märgend */
                  s->t=id->t;
                  }
            break;
            }
      case konst:{
            for(k=0;k<ktl;k++){
                  if(KT[k]==t->leks){
                        c=CT[k];
                        goto cok;
                        }
                  }
            cok:
            s->c=c;
            s->liik=2;                        /* konstant */
            break;
            }
      default: {
            memset(rida,'\0',80);
            sprintf(rida,"del %s.asm",pr_name);
            system(rida);
            fclose(rules);
            fprintf(Logi,"cannot compile the program %s.asm<BR>",
                  pr_name);
            ExIT();
            }
      }
   return(s);
}

int logic(int i, struct top *t){
   int op;
   op=t->sem;
   Put("\tmov\teax,");
   w_addr(i,2);
   Put("\n\tcmp\teax,");
   w_addr(i,1);
   Put("\n");
   switch(op){
      case  pisem:{              /* < */
            Put("\tjb\t");
            break;
            }
      case  suurem:{              /* > */
            Put("\tjg\t");
            break;
            }
      case  piv:{              /* <= */
            Put("\tjbe\t");
            break;
            }
      case  suv:{              /* >= */
            Put("\tjge\t");
            break;
```

```
                }
            case  pov:{              /* /= */
                  Put("\tjne\t");
                  break;
                  }
            case  vord:{              /* = */
                  Put("\tje\t");
                  break;
                  }
            }
      freestack(i,2);
      i-=2;
      if(t->truel>0){
            if(t->truel>tnr) sprintf(rida,"%s\n",T[t->truel]);
            else sprintf(rida,"MExi%d\n",t->truel);
            Put(rida);
            }
      if(t->falsel>0){
            sprintf(rida,"\tjmp\tMExi%d\n",t->falsel);
            Put(rida);
            }
      return(i);
}

int Assign(int i){
//1. variant: lae value->eax ja kirjuta eax->addr
      if(nado==0){
       Put("\tmov\teax,");
         w_addr(i,1);
         Put("\n\tmov\t");
         w_addr(i,2);
         Put(",eax\n");
         freestack(i,2);
         i-=2;
         return(i);
       }
//2. variant: value ON eax-s, kirjuta dest-addr stack-tipust
   Put("\tmov\t");
   w_addr(i,1);
   Put(",eax\n");
   freestack(i,1);
   i-=1;
   return i;
}

//Aritm:  lipp nado=0: omist -- tipa i:=0, aritm -- a+1
//nado=1: omist -- value is in eax, "pikk aritm": 1. operand on eax-s

int Aritm(int i,struct top *t){
      int op;
      struct item *s;
      struct top *t1;
      op=t->sem;
      switch(op){
            case jag:
             if(nado==0){
                Put("\txor\tedx,edx\n");
                Put("\tmov\teax,");
                w_addr(i,2);
                Put("\n\tmov\tebx,");
                w_addr(i,1);
```

```
                Put("\n");
                Put("\tdiv\tebx\n");
                }
            else{
                Put("\txor\tedx,edx\n");
                Put("\tmov\tebx,");
                w_addr(i,1);
                Put("\n");
                Put("\tdiv\tebx\n");
               }
              break;
            case korrut:
             if(nado==0){
                Put("\tmov\teax,");
                  w_addr(i,2);
                  Put("\n\tmov\tedx,");
                  w_addr(i,1);
                  Put("\n");
                  Put("\tmul\tedx\n");
                }
            else{
                Put("\tmov\tedx,");
                  w_addr(i,1);
                  Put("\n");
                  Put("\tmul\tedx\n");
                }
                  break;
            case lahut:              /* lahutamine */
             if(nado==0){
                Put("\tmov\teax,");
                w_addr(i,2);
                Put("\n\tsub\teax,");
                w_addr(i,1);
                Put("\n");
                }
            else{
                Put("\tsub\teax,");
                w_addr(i,1);
                Put("\n");
                }
              break;
            case liit:               /* liitmine */
              if(nado==0){
                Put("\tmov\teax,");
                w_addr(i,2);
                Put("\n\tadd\teax,");
                w_addr(i,1);
                Put("\n");
                }
            else{
                Put("\tadd\teax,");
                w_addr(i,1);
                Put("\n");
                }
            break;
            }
    if(nado==0)freestack(i,2);   //2 tipmist tyhjaks, stackindex i ei muutu
    else freestack(i,1);         //tipmine vabaks
//tmarv: >1 teeb: res->tm
    if(tmarv>1){
         s=make_item();
```

```
            s->liik=0;
            s->index=IX;
            stack[i-2]=s;  //1. operandi asemele töömuutuja
            sprintf(rida,"\tmov\tdTv%d,eax\n",IX);
            Put(rida);
            IX++;
            i-=1;   //uus kirjutamiskoht tippu 2.operandi asemele
          return(i);
         }
     if(nado==0){
       nado=1;   // ar-av value jääb eax-i,
       i-=2;     //kaks operandi magasinist minema
       }
     else i-=1;  //jrk operand magasini tipust minema, jooxev res=eax
     return(i);
}

int InOut(int i,struct top *t){
   struct item *s;
   struct itr *id;
   int op,x;
   s=stack[i-1];
   id=s->id;
   op=t->sem;
   switch(op){
    case lugem:
      sprintf(rida,"\tinvoke\tcrt_printf,ADDR io%d\n",id->tio);
      Put(rida);
      sprintf(rida,"\tinvoke\tcrt_scanf,ADDR format1,ADDR %s\n",T[id->nr]);
      Put(rida);
      break;
   case kirjut:
      sprintf(rida,"\tinvoke\tcrt_printf,ADDR io%d\n",id->tio);
      Put(rida);
      sprintf(rida,"\tinvoke\tcrt_printf,ADDR format1,%s\n",T[id->nr]);
      Put(rida);
      break;
}
freestack(i,1);
i-=1;
return(i);
}

void trigol_Asm(struct top *root){
      struct top *t;
      int op;
      int i=0;
      int j;
      IX=0;       /* t””muutuja index */
      Label=0;
      t=root->down;
      for(j=0;j<20;j++) stack[j]=(struct item *)NULL;
down:
      if(t->label>0) Label=t->label;
      if(t->down!=(struct top *)NULL){
            t=t->down;
            goto down;
            }
      stack[i]=leaf(t);
      i++;
      prist(i,0);
```

```
naaber:     if(t->up==(struct top *)NULL){
            t=t->right;
            goto down;
            }
      t=t->up;
      if(t==root) goto ok;
      green(";compiling the operator  ");
      print_Op(t);
      ps();
      pp2html(t);
      if(Label!=0) w_label(Label);
      op=t->sem;
      if(op>=pisem&&op<=vord){
            i=logic(i,t);
            goto naaber;
            }
      if(op>=jag&&op<=liit){
            i=Aritm(i,t);
            goto naaber;
            }
      switch(op){
            case omist:             /* omistamine */
                  i=Assign(i);
            nado=0;
                  break;
            case suunam:               /* suunamine */
                  Put("\tjmp\t");
                  w_addr(i,1);
                  Put("\n");
                  freestack(i,1);
                  i-=1;
                  break;
            case kuisiis:     break;
            case tingop: break;
            case lugem: i=InOut(i,t); break;
            case kirjut: i=InOut(i,t); break;
            }
      prist(i,0);
      goto naaber;
ok:   Put("\tinvoke\tExitProcess,0\n");
      Put("end start\n");
      fflush(rules);
      fclose(rules);
      fprintf(Logi,
        "<FONT COLOR=\"008000\">programm %s.asm is compiled<BR></FONT>",
        Pr_name);
}

FILE *op(void){
      FILE *inf=NULL;
      inf=fopen(rida,"rb");
      if (inf==NULL){
            fprintf(Logi,"cannot open the file %s ",rida);
            fclose(inf);
            return(NULL);
            }
      rewind(inf);
      return(inf);
}

int LogicO(int i,struct top *t){
```

```
struct item *s1,*s2;
struct ctr *c;
int op,r,x,y;
s1=stack[i-2];
s2=stack[i-1];
if((s1->liik==2||s1->liik==4)&&(s2->liik==2||s2->liik==4)){
      green("constant expression, I'll optimize..<BR>");
      r=0;
      if(s1->liik==2){
            c=s1->c;
            x=c->value;
            }
      else x=s1->index;
      if(s2->liik==2){
            c=s2->c;
            y=c->value;
            }
      else y=s2->index;
      op=t->sem;
      switch(op){
            case pisem:
                  if(x<y)  r=1;
                  sprintf(rida,"%d < %d ?<BR>",x,y);
                  green(rida);
                  break;
            case suurem:
                 if(x>y)r=1;
                 sprintf(Logi,"<FONT COLOR=\"008000\">%d > %d
      ?<BR></FONT>",x,y);
                 break;
            case piv:
                  if(x<=y) r=1;
                  printf(Logi,"<FONT COLOR=\"008000\">%d <= %d
            ?<BR></FONT>",x,y); break;
            case suv:
                  if(x>=y) r=1;
                  fprintf(Logi,
                        "<FONT COLOR=\"008000\">%d >= %d
                        ?<BR></FONT>",x,y);
                  break;
            case pov:
                  if(x!=y) r=1;
                  fprintf(Logi,"<FONT COLOR=\"008000\">%d /= %d
            ?<BR></FONT>",x,y); break;
            case vord:
                  if(x==y) r=1;
                  fprintf(Logi,
                        "<FONT COLOR=\"008000\">%d = %d
                  ?<BR></FONT>",x,y);
                  break;
                  }
            freestack(i,2);
            i-=2;
            if(r==1){
             if(t->truel>tnr)sprintf(rida,"\tjmp\t%s\n",T[t->truel]);
             else sprintf(rida,"\tjmp\tMExi%d\n",t->truel);
             Put(rida);
             }
            else{
             if(t->truel>tnr)
```

```
                        sprintf(rida,"\tjmp\t%s\n",T[t->falsel]);
                  else sprintf(rida,"\tjmp\tMExi%d\n",t->falsel);
                   Put(rida);
                 t=t->up;
                 t=t->right;
                 }
      }
      return(i);
}

int AritmO(int i,struct top *t){
      struct item *s1,*s2;
      struct ctr *c;
      int op,r,x,y;
      s1=stack[i-2];
      s2=stack[i-1];
      if((s1->liik==2||s1->liik==4)&&(s2->liik==2||s2->liik==4)){
            green("constant expression, I'll optimize..<BR>");
            fflush(Logi);
            r=0;
            if(s1->liik==2){
                  c=s1->c;
                  x=c->value;
                  }
            else x=s1->index;
            if(s2->liik==2){
                  c=s2->c;
                  y=c->value;
                  }
            else y=s2->index;
            op=t->sem;
            switch(op){
                  case jag: r=x/y;
                        fprintf(Logi,
                        "<FONT COLOR=\"008000\">%d = %d /
%d<BR></FONT>",r,x,y);
                              break;
                  case korrut: r=x*y;
                        fprintf(Logi,
                        "<FONT COLOR=\"008000\">%d = %d *
%d<BR></FONT>",r,x,y);
                              break;
                  case lahut: r=x-y;
                        fprintf(Logi,
                        "<FONT COLOR=\"008000\">%d = %d -
%d<BR></FONT>",r,x,y);
                              break;
                  case liit: r=x+y;
                        fprintf(Logi,
                        "<FONT COLOR=\"008000\">%d = %d +
%d<BR></FONT>",r,x,y);
                              break;
                        }
                  s1->liik=4;
                  s1->index=r;
                  freestack(i,1);
                  i-=1;
            }
      fflush(Logi);
      return(i);
}
```

```
void trigol_Asm_O(struct top *root){
      int op;
      int i=0;
      int j;
      IX=0;       /* töömuutuja index */
      Label=0;
      t=root->down;
      for(j=0;j<20;j++) stack[j]=(struct item *)NULL;
down:
      if(t->label>0) Label=t->label;
      if(t->down!=(struct top *)NULL){
            t=t->down;
            goto down;
            }
      stack[i]=leaf(t);
      i++;
      prist(i,0);
naaber:     if(t->up==(struct top *)NULL){
            t=t->right;
            goto down;
            }
      t=t->up;
      if(t==root) goto ok;
      green("; compiling the operator  ");
      print_Op(t);
      ps();
      pp2html(t);
      if(Label!=0) w_label(Label);
      fflush(Logi);
      op=t->sem;
      if(op>=pisem&&op<=vord){
            j=i;
            i=LogicO(i,t);
            if(i==j) i=logic(i,t);
            goto naaber;
            }
      if(op>=jag&&op<=liit){
            j=i;
            i=AritmO(i,t);
            fflush(Logi);
            if(i==j) i=Aritm(i,t);
            fflush(Logi);
            goto naaber;
            }
      switch(op){
            case omist: i=Assign(i);
            nado=0;
                  break;
            case suunam:
                  Put("\tjmp\t");
                  w_addr(i,1);
                  Put("\n");
                  freestack(i,1);
                  i-=1;
                  break;
            case kuisiis:
                  break;
            case tingop: break;
            case lugem: i=InOut(i,t); break;
            fflush(Logi);
```

```
            case kirjut: i=InOut(i,t); break;
            fflush(Logi);
            }
      //printf("i=%d\n",i); getchar();
    prist(i,0);
      goto naaber;
ok:   Put("\tinvoke\tExitProcess,0\n");
    Put("end start\n");
      fflush(rules);
      fclose(rules);
      fprintf(Logi,
        "<FONT COLOR=\"008000\">programm %s.asm is compiled<BR></FONT>",
        Pr_name);
}

void to_asm(struct top *p){
      Rd=0;
      Wr=0;
      tmarv=det_nrlv(p);
      set_show("Modified tree:");
      pp2html(p_);
    if(gen_header()==0) goto ots;
      (opt==0) ? trigol_Asm(p_) : trigol_Asm_O(p);
    fprintf(Logi,"<BR>I'll start MASM32 without any logfile and start
%s.exe<BR>",Pr_name);
    gen_bat();
    system("maketri");
    sprintf(rida,"%s\n",Pr_name);
    system(rida);
ots:
    fflush(Logi);
    fflush(rules);
    fclose(rules);
    printf("\npress any key..\n"); getchar();
}


struct top *analyzer(void){
      if(r_tabs()==0) return(0);
      set_show("Program");
      p_prog(p_);
      set_show("Parsing tree");
      pp2html(p_);
      ctree();
      fflush(Logi);
      return(p_);
}

int itr(void){
      int ret=0;
      time_t t0;
      GBuf=NULL;
      Logi=fopen(L_name,"w");
      if(Logi==NULL){
            printf("Cannot open log-book\n");
            return(0);
            }
      m_k=4; k_k=11;
      logi=1;
      time(&t0);
      fprintf(Logi,"<HTML><HEAD><TITLE>Compiler</TITLE></HEAD><BODY><B>");
```

```
      fprintf(Logi,"<FONT COLOR=\"#0000FF\"><H3>Start of TRIGOL ");
      if(opt==1) fprintf(Logi,"Optimizing ");
      fprintf(Logi,"Compiler for a Program ");
      fprintf(Logi,"</FONT>%s ",pr_name);
      fprintf(Logi,"<FONT COLOR=\"#0000FF\"> at </FONT>");
      fprintf(Logi,"%s</H3><BR>",asctime(localtime(&t0)));
      if(analyzer()!=0) to_asm(p_);
      time(&t0);
      fprintf(Logi,
            "<FONT COLOR=\"#0000FF\"><H4>Compiler ended at </FONT> ");
      fprintf(Logi,"%s</H4>",asctime(localtime(&t0)));
      fprintf(Logi,"</BODY></HTML>");
    fflush(Logi);
    fclose(Logi);
      return(ret);
}

int main(int argc,char **argv){
      opt=0;
      if(argc<2){
            printf("arguments: p-name [o]\n"); abort();
            }
      if(argc==3) opt=1;

      pr_name=(char *)malloc(256);
      L_name=(char *)malloc(256);
      Nimi=(char *)malloc(8);
      Pr_name=(char *)malloc(256);
      memset(pr_name,'\0',256);
      memset(L_name,'\0',256);
      memset(Nimi,'\0',8);
      strcpy(Nimi,"tri");
      memset(Pr_name,'\0',256);
      sprintf(pr_name,"%s",argv[1]);
      strcpy(Pr_name,pr_name);
      strcpy(L_name,pr_name);
      strcat(pr_name,".tri");
      if(opt==0) strcat(L_name,"c.htm");
      else strcat(L_name,"oc.htm");
      printf("pr_name=%s L_name=%s\n",pr_name,L_name);
      itr();
      return(1);
}
```

Lisa 10. Trigoli kompilaatori abivahendid ja väljundid

1. TASMi versiooni sisend- ja väljundprogramm teek.asm (bin2dec[1] ja readint)

```
; The Waite Group's MS-DOS Developer's Guide, Second Edition,
; John Angermeyer jt,Howard W. Sams & Company, 1989, pp 724-725
;
; bin2dec
; INPUT: AX - number to be displayed
;      CH - minimum number of digits to be displayed
;      DX = 0, if number is unsigned, 1, if signed
; OUTPUT None
;
;
      .MODEL      small
      .STACK      100h
      .DATA
      .CODE
      PUBLIC bin2dec
      PUBLIC readint
;
bin2dec     PROC  NEAR
      push  ax
      push  bx
      push  cx
      push  dx
      mov   cl,0
      mov   bx,10
      cmp   dx,0
      je    more_dec
      or    ax,ax
      jnl   more_dec
      neg   ax
;     @DisChr     '-'
      push  ax
      push  dx
      mov   dl,'-'
      mov   ah,02h
      int   21h
      pop   dx
      pop   ax
more_dec:
      xor   dx,dx
      div   bx
      push  dx
      inc   cl
      or    ax,ax
      jnz   more_dec
; Main Digit Print Loop - Reverse Order
      sub   ch,cl
      jle   morechr
      xor   dx,dx
morezero:
      push  dx
      inc   cl
      dec   ch
      jnz   morezero
```

[1] Vt. [TWG].

```
morechr:
      pop   dx
      add   dl,30h
; DisChr
      push  ax
      push  dx
      mov   ah,02h
      int   21h
      pop   dx
      pop   ax
      dec   cl
      jnz   morechr
      pop   dx
      pop   cx
      pop   bx
      pop   ax
      ret
bin2dec     ENDP
;
;
; vt. ka
; Turbo Assembler. Users Guide, Version 1.0, Borland International, 1989
; p 55[1]; Angermayer et al, p.560
;
; readint
; INPUT: none
; OUTPUT AX - sisestatud ja teisendatud arv
;
;
readint     PROC  NEAR
      push  bx
      push  cx    ;save
      push  dx    ;registers; res=ax
      xor   ax,ax   ;arv
ring: push  ax    ;save
      mov   ah,1  ;DOS keyboard input
      int   21h   ;get the next symbol
      mov   cl,al ;new digit in cl
      sub   cl,30h      ;symbol->10nd-nr
      cmp   al,13 ;Enter?
      jz    oki   ;valmis
      pop   ax    ;senine arv
      mov   dx,10
      mul   dx    ;senine arv * 10
      xor   ch,ch ;prepare for 16-bit add
      add   ax,cx ;new digit is added in
      jmp   ring
oki:  mov   dl,10 ;linefeed
      mov   ah,2  ;DOS display output
      int   21h
      pop   ax
      pop   dx
      pop   cx
      pop   bx
      ret
readint ENDP
      END
```

[1] Vt. [Borland].

2. Programm P6.asm, TASMi versioon

```
; # READ n ; F := 1 ; I := 0 ;
; M1 : I := I + 1 ;
; IF I > n THEN GOTO M2 ;
; F := F * I ;
; GOTO M1 ;
; M2 : WRITE F #
;
; Program P6.asm
      .MODEL      small
      .STACK      100h
      EXTRN readint:PROC
      EXTRN bin2dec:PROC
      .DATA
n     DW    0
F     DW    0
I     DW    0
dTv0  DW    0
Sisse DB    'Input the variable ','$'
Trykk DB    'Variable ','$'
n_S   DB    'n=','$'
F_S   DB    'F=','$'
      .CODE
ProgramStart:
      mov   ax,@data
      mov   ds,ax
      mov   ah,9h
      mov   bx,1
      mov   cx,17
      mov   dx,OFFSET Sisse
      int   21h
      mov   ah,9h
      mov   bx,1
      mov   cx,2
      mov   dx,OFFSET n_S
      int   21h
      call  readint
      mov   n,ax
      mov   ax,1
      mov   F,ax
      mov   ax,0
      mov   I,ax
M1:   mov   ax,I
      add   ax,1
      mov   dTv0,ax
      mov   ax,dTv0
      mov   I,ax
      mov   ax,I
      cmp   ax,n
      jg    M2
      mov   ax,F
      mov   dx,I
      mul   dx
      mov   dTv0,ax
      mov   ax,dTv0
      mov   F,ax
      jmp   M1
M2:   mov   ah,9h
      mov   bx,1
      mov   cx,8
```

```
      mov   dx,OFFSET Trykk
      int   21h
      mov   ah,9h
      mov   bx,1
      mov   cx,2
      mov   dx,OFFSET F_S
      int   21h
      mov   ax,F
      mov   dx,0
      cmp   ax,0
      jg    s1h2o3w
      mov   dx,1
s1h2o3w:
      mov   ch,1
      call  bin2dec
      mov   ah,4ch
      int   21h
      END   ProgramStart
```

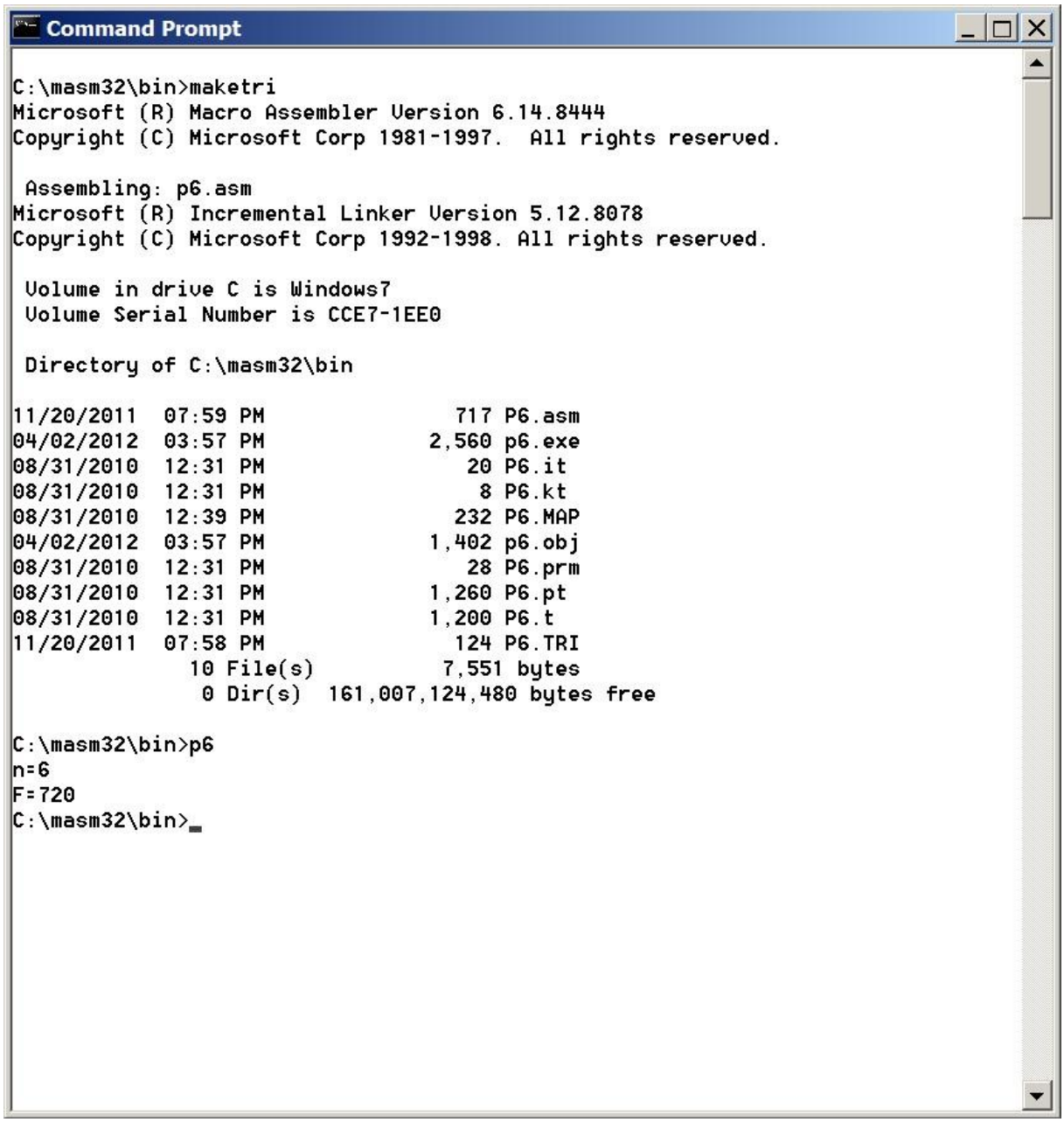
3. Programm maketri.bat, MASM32 versioon (genereeritud MASMi editori poolt) P6.tri jaoks

```
@echo off

if not exist rsrc.rc goto over1
\masm32\bin\rc /v rsrc.rc
\masm32\bin\cvtres /machine:ix86 rsrc.res
 :over1
f exist "p6.obj" del "p6.obj"
if exist "p6.exe" del "p6.exe"

\masm32\bin\ml /c /coff "p6.asm"
if errorlevel 1 goto errasm

if not exist rsrc.obj goto nores

\masm32\bin\Link /SUBSYSTEM:CONSOLE "p6.obj" rsrc.res
 if errorlevel 1 goto errlink

dir "p6.*"
goto TheEnd

:nores
 \masm32\bin\Link /SUBSYSTEM:CONSOLE "p6.obj"
 if errorlevel 1 goto errlink
dir "p6.*"
goto TheEnd

:errlink
 echo _
echo Link error
goto TheEnd
:errasm
 echo _
echo Assembly Error
goto TheEnd

:TheEnd

pause
```

4. Programm P6.tri, MASM32 versioon

```
; # READ n ; F := 1 ; I := 0 ;
; M1 : I := I + 1 ;
; IF I > n THEN GOTO M2 ;
; F := F * I ;
; GOTO M1 ;
; M2 : WRITE F #
;
; Program p6.asm
include \masm32\include\masm32rt.inc
.data
Sisse   db "Enter n=",0
format1 db "%d",0
Trykk db "F=%d",0
.data?
n     dd ?
F     dd ?
I     dd ?
dTv0  dd ?
.code
start:
    invoke  crt_printf,ADDR Sisse
    invoke  crt_scanf,ADDR format1,ADDR n
      mov   eax,1
      mov   F,eax
      mov   eax,0
      mov   I,eax
M1:   mov   eax,I
      add   eax,1
      mov   I,eax
      mov   eax,I
      cmp   eax,n
      jg    M2
      mov   eax,F
      mov   edx,I
      mul   edx
      mov   F,eax
      jmp   M1
M2:   invoke  crt_printf,ADDR Trykk,F
      invoke  ExitProcess,0
end start
```

5. Programmi P6.tri MASM32-versiooni kompileerimine ja lahendamine



Lisa 11. Trigoli laiendamine: Viktor Karabuti versioon

Viktor Karabut.

Trigoli laiendamine (12.12.2010)

Ülesanne

Lisada keelele ühemõõtmelised massiivid, sealjuures:

1. massiivi deklareerimine ja initsialiseerimine:
2. `ARRAY a[<length_expression>];`
3. massiivi elemendile omistamine;
4. `a[<index_expression>] := <value_expression>;`
5. lugemine massiivist;
6. `<variable> := a[<index_expression>];`

Programmide näited

ext1.tri

```
#
ARRAY a[4];
a[0]:=1;
a[1]:=2;
a[2]:=3;
a[3]:=a[0]+a[1]+a[2]
#
```

ext2.tri

Maksimaalse elemendi otsimine

```
#
alen := 5;
ARRAY a[alen];
a[0] := 23;
a[1] := 85;
a[2] := 10;
a[3] := 47;
a[4] := 62;
max := 0;
i := 1;
WHILE:
        IF a[max] < a[i] THEN max := i;
        i := i + 1;
IF i < alen THEN GOTO WHILE;
maxvalue := a[max]#
```

ext3.tri

Massiivi sorteerimine

```
#
alen := 5;
ARRAY a[alen];
a[0] := 23;
a[1] := 85;
a[2] := 10;
a[3] := 47;
a[4] := 62;
i := 0;
FORi:
        j := i + 1;
        FORj:
                IF a[i] <= a[j] THEN GOTO NOSWAP;
                        t := a[i];
                        a[i] := a[j];
                        a[j] := t;
                NOSWAP:
                j := j + 1;
        IF j < alen THEN GOTO FORj;
        i := i + 1;
IF i < alen-1 THEN GOTO FORi
#
```

Grammatika täiendus

```
`masindeks'          ->  [`aritmav']
`masdeklaratrsion'   ->  ARRAY`massiiv'
`operaator'          ->  `masdeklaratrsion'
`operaator'          ->  `masomistamine'
`massiiv'            ->  #i#`masindeks'
`masomistamine'      ->  `massiiv':=`omistamine1'
`masvaartus'         ->  `massiiv'
`tegur'              ->  `masvaartus'
```

Semantika täiendus

```
p48=22 $ masdeklaratrsion' ->  ARRAY`masnimi'`masindeks'
p52=23 $ masomistamine'    ->  `massiv':=`omistamine1'
p53=24 $ masvaartus'       ->  #i#`masindeks'
```

Interpretaatori täiendus

Fail two32.h

1. Uus struktuur, kuhu salvestatakse trigoli massiiv:
2. #define MAX_ARRAY_LEN 1000
3. struct tri_array{
4. int len;   /* massiivi pikkus */
5. int *data;   /* massiivi sisend */
6. };
7. Identifikaatorite tabeli kirjesse on lisatud uus väli struct tri_array *arr:
8. struct itr{

```
 9.  ...
 10.         struct tri_array *arr; /* interpretaaator: massiiv */
 11.         ...
     };
```

Fail int32.c

```
1. Semantika koodid:
2. ...
3. #define massdekl  22    /* massiivi deklareerimine */
4. #define massomist 23    /* massiivi omistamine */
5. #define mvaartus  24    /* massiivi väärtus */
6. ...
```

7. On muudetud operaatori teksti trükk analüüsi puu järgi

```
8. int print_Op(struct top *t){
9.   int fg=0;
10.         if(t!=(struct top *)NULL){
11.                 ...
12.                 if(t->sem==massdekl) {
13.                         fprintf(Logi,"ARRAY ");
14.                         print_Op(t->down);
15.                         fprintf(Logi,"[");
16.                         print_Op(t->down->right);
17.                         fprintf(Logi,"]");
18.                         return(0);
19.                         }
20.                 if(t->sem==massomist) {
21.                         print_Op(t->down);
22.                         fprintf(Logi,"[");
23.                         print_Op(t->down->right);
24.                         fprintf(Logi,"]:=");
25.                         print_Op(t->down->right->right);
26.                         return(0);
27.                         }
28.                 if(t->sem==mvaartus) {
29.                         print_Op(t->down);
30.                         fprintf(Logi,"[");
31.                         print_Op(t->down->right);
32.                         fprintf(Logi,"]");
33.                         return(0);
34.                         }
35.                 ...
36.         }
37.         ...
38. }
39.
40. void tprop(int s){
41.         switch(s){
42.                 ...
43.                 case massdekl:fprintf(Logi,"ARRAY"); break;
44.                 case massomist:fprintf(Logi,"[]:="); break;
45.                 case mvaartus:fprintf(Logi, "->"); break;
46.                 ...
47.                 }
   }
```

48. Funktsioonid struktuuriga struct tri_array töötamiseks:

```
49.  /* uue massiivi mälueraldus */
50.  struct tri_array *make_tri_array(int len){
51.         struct tri_array *c;
```

```
52.         c=(struct tri_array *)malloc(sizeof(struct tri_array));
53.         if(c==NULL) ExIT();
54.         c->data = (int *)malloc(sizeof(int)*len);
55.         if(c->data == NULL) {
56.                 free(c);
57.                 ExIT();
58.         }
59.         memset(c->data,'\0',sizeof(int)*len);
60.         c->len = len;
61.         return(c);
62. }
63.
64. /* trigoli massiivi kustutamine ja mälu vabastamine */
65. void free_tri_array(struct tri_array *arr) {
66.         if (arr->data != NULL) {
67.                 free(arr->data);
68.         }
69.         free(arr);
    }
```

70. Ja interpretaatori täiendus.

```
71. /* Luua uus massiiv */
72. void create_tri_array(int len, int i) {
73.         struct item *s;
74.         struct itr *id;
75.         if(len <= 0) {
76.                 red("Error, array length should be posititive
    integer.");
77.                 ExIT();
78.         }
79.         if(len > MAX_ARRAY_LEN) {
80.                 red("Error, so big array.");
81.                 ExIT();
82.         }
83.         s=stack[i-2];
84.         id=s->id;
85.         if(id->arr != NULL) {
86.                 free_tri_array(id->arr);
87.         }
88.         id->arr = make_tri_array(len);
89.         fprintf(Logi,"<BR>Created new array %s<BR>",T[id->nr]);
90.         p_tri_array(T[id->nr], id->arr);
91. }
92.
93. /* Kontrollida massiivi indeksit */
94. void check_index(struct tri_array *arr, int index) {
95.         if(arr == NULL) {
96.                 red("Error, array is not initialized");
97.                 ExIT();
98.         }
99.         if (index < 0) {
100.                red("Error, array index should be greater or equal
    than 0");
101.                ExIT();
102.        }
103.        if (index >= arr->len) {
104.                red("Error, too big index");
105.                ExIT();
106.        }
107. }
108.
```

```
109. /* Kirjuta massiivi */
110. void write_to_cell(int value, int index, int i) {
111.         struct item *s;
112.         struct itr *id;
113.         s=stack[i-3];
114.         id=s->id;
115.         check_index(id->arr, index);
116.         id->arr->data[index] = value;
117.         fprintf(Logi,"<BR>Writed value %d to %s[%d]<BR>", value,
   T[id->nr], index);
118.         p_tri_array(T[id->nr], id->arr);
119. }
120.
121. /* Lugeda massiivist  */
122. void tri_array_value(int index, int i) {
123.         struct item *s1, *s2;
124.         struct itr *id;
125.         int res;
126.         s1=stack[i-2];
127.         id=s1->id;
128.         check_index(id->arr, index);
129.         res = id->arr->data[index];
130.         fprintf(Logi,"%s[%d]=%d",T[id->nr], index, res);
131.         freestack(i,2);
132.         s2=make_item();
133.         s2->liik=0;
134.         s2->index=res;
135.         stack[i-2]=s2;
136. }
137.
138. /* TRIGOL-keelsete programmide interpretaator  */
139. void trigol_Int(struct top *root){
140.         ...
141.         switch(op){
142.                 ...
143.                 case massdekl:{
144.                         x=get_x(i,1);
145.                         create_tri_array(x,i);
146.                         freestack(i,2);
147.                         i-=2;
148.                         break;
149.                         }
150.                 case massomist:{
151.                         x=get_x(i,1);
152.                         y=get_x(i,2);
153.                         write_to_cell(x, y, i);
154.                         freestack(i,3);
155.                         i-=3;
156.                         break;
157.                         }
158.                 case mvaartus:{
159.                         x=get_x(i,1);
160.                         tri_array_value(x, i);
161.                         i-=1;
162.                         break;
163.                         }
164.                 }...}
```

Lisa 12. Näide sõnast keeles Form8[1]

Paljud arvutiteaduses (ka diskreetses matemaatikas) ettetulevad probleemid on kirjeldatud parametriseeritud lausearvutusvalemite perede abil. Selline vajadus tekib tõsiasjast, et paljudele kombinatoorikaülesannetele pole leitud analüütilisel kujul lahendivalemit. Üldine idee on siis esitada ülesanne lausearvutusvalemi abil ja leida spetsiaalse programmiga selle lahend – kehtestavate väärtustuste arv.

Samas tekib probleem, kuidas transleerida tekkinud (üldiselt parameetritest sõltuv) lausearvutusvalem selliseks lausearvutusvalemiks, milles pole parameetreid. Klassikalist viisi jälgides tuleb iga kord koostada programm, mis etteantud parameetrite väärtustusi arvestades sooritab teisenduse. Taolised programmid võivad olla küllalt pikad ja keerulised. Luues aga kõikide selliste valemite kirjelduse (teatavate mööndustega muidugi; vt. grammatika *form8*), saame kirjutada translaatori (vt. [Peder, Tombak, Isotamm]), mis selles keeles lubatud sõna (ehk valemi) analüüsi puust lähtuvalt vajaliku teisenduse sooritab. Saadud tulemusele saame rakendada olemasolevaid kehtestajaid.

Seega, grammatikaga *form8* saadavad keele sõnad on tegelikult mingid parameetritega lausearvutusvalemid ja nimetatud translaatorit (koos grammatikaga) võime vaadelda kui programmeerimiskeele[2] realisatsiooni.

Näide. Malelaual paiknevate kuningate probleem kuulub kombinatoorikaülesannete klassikasse ja on näide probleemi kirjeldusest lausearvutusvalemiga. Järgnevalt esitame *2m×2n* malelaual *m×n* kuninga paarikaupa tules olemist kirjeldava lausearvutusvalemite pere (rohkem kuningaid pole võimalik sellisele lauale asetada, ilma et ükski kuningatepaar poleks vastastikku tules).

Ülesande lahendusidee on jagada kogu suur ala *2×2* ruudukesteks, igasse neist asetada täpselt üks kuningas. Lausemuutujate paari $(x_{i,j}, y_{i,j})$ väärtustus kirjeldab konkreetse kuninga asukohta enda *2×2* ruudus. Sellise valemite pere parameetriteks on arvud *m* ja *n*. Saame valemi

$$F = F_1 \& F_2 \& F_3 \& F_4,$$

kus F_i-id peame asendama järgmiselt[3]:

$$F_1 = \bigwedge_{1 \le i \le m-1} \bigwedge_{1 \le j \le n} (\overline{x_{i,j}} \vee x_{i+1,j}),$$

$$F_2 = \bigwedge_{1 \le i \le m} \bigwedge_{1 \le j \le n-1} (\overline{y_{i,j}} \vee y_{i,j+1}),$$

$$F_3 = \bigwedge_{1 \le i \le m-1} \bigwedge_{1 \le j \le n-1} (\overline{x_{i,j}} \vee \overline{y_{i,j}} \vee x_{i+1,j+1} \vee y_{i+1,j+1}),$$

[1] Selle lisa kirjutas toimetaja *Ahti Peder*, üks keele *Form8* kahest autorist.

[2] A. I., selle raamatu autor: keel ei pea olema ei protseduur- ega ka probleemorienteeritud (nagu me oleme harjunud mõtlema) *programmeerimis*keel, vaid võib olla tekstide teisendaja ühelt kujult teisele, sellisele, mille jaoks on olemas programne realisatsioon. *Form8* väljundandmestruktuuri jaoks oli see programne tugi olemas (programm #sat).

[3] Anname vihje: siintoodu on loodetavasti mõistetev inimesele, kes on tuttav tekstide generaatori T_EX-iga.

$$F_4 = \bigwedge_{1\le i\le m-1} \bigwedge_{2\le j\le n} (\overline{x_{i,j}} \vee y_{i,j} \vee x_{i+1,j-1} \vee \overline{y_{i+1,j-1}}).$$

Grammatika *form8* (vt. Lisa 2) abil saab seda samaväärselt esitada järgmiselt:

```
#(\bigwedge_{1\leqi\leqm1}\bigwedge_{1\leqj\leqn}(\negx_{i,j}
\veex_{i+1,j}))\&(\bigwedge_{1\leqi\leqm}
\bigwedge_{1\leqj\leqn1}(\negy_{i,j}\veey_{i,j+1}))\&
(\bigwedge_{1\leqi\leqm1}\bigwedge_{1\leqj\leqn1}(\negx_{i,j}
\vee\negy_{i,j}
\veex_{i+1,j+1}\veey_{i+1,j+1}))\&(\bigwedge_{1\leqi\leqm1}
\bigwedge_{2\leqj\leqn}
(\negx_{i,j}\veey_{i,j}\veex_{i+1,j-1}\vee\negy_{i+1,j-1}))#.
```

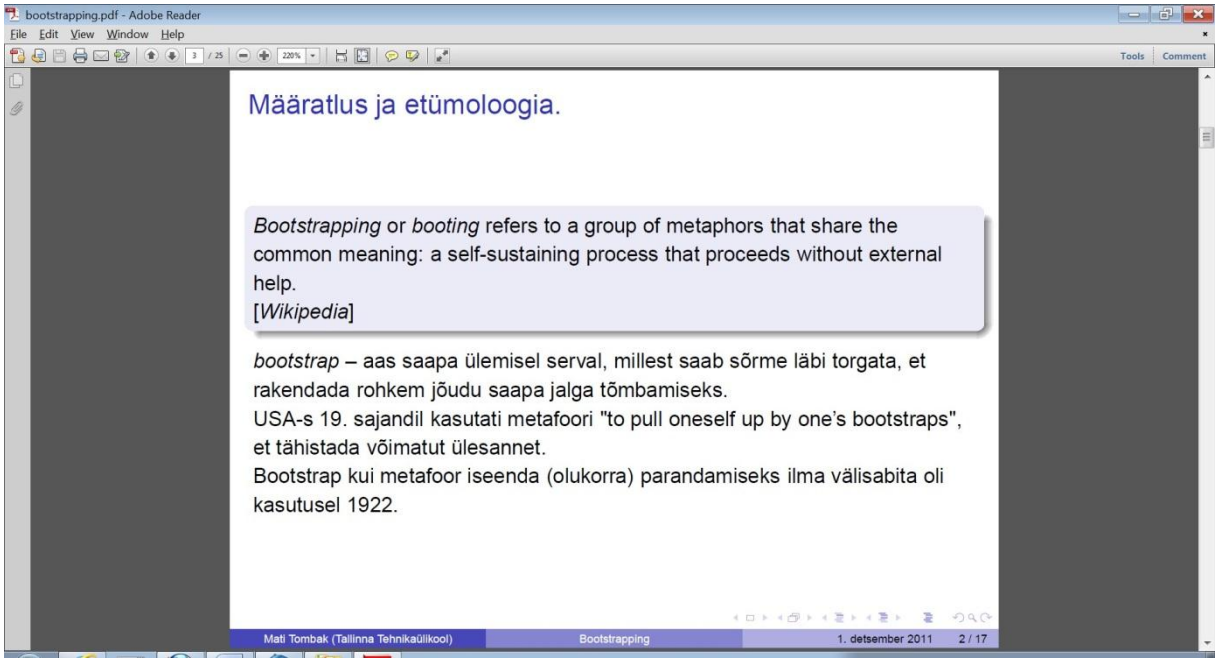
Rakendades eelnimetatud translaatorit ja väärtustades selle töö käigus parameetrid *m=4* ja *n=4* (klassikaline malelaud), saame tavalise prefiksikujul lausearvutusvalemi (ühtlustamiseks on muutujad $x_{i,j}$ ja $y_{i,j}$ asendatud *xi*-dega):

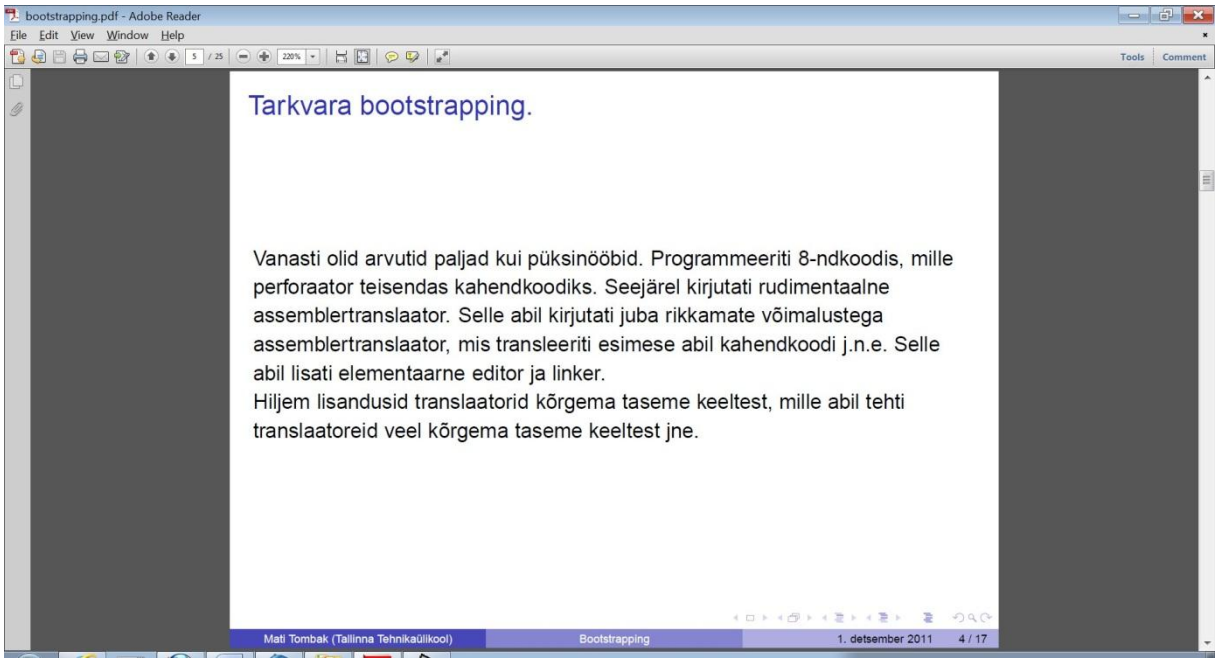
```
and((and(and((or(not(x1),x2)),(or(not(x3),x4)),(or(not(x5),x6)),
(or(not(x7),x8))),and((or(not(x2),x9)),(or(not(x4),x10)),
(or(not(x6),x11)),(or(not(x8),x12))),and((or(not(x9),x13)),
(or(not(x10),x14)),(or(not(x11),x15)),(or(not(x12),x16))))),
and((and(and((or(not(x17),x18)),(or(not(x18),x19)),
(or(not(x19),x20))),and((or(not(x21),x22)),(or(not(x22),x23)),
(or(not(x23),x24))),and((or(not(x25),x26)),(or(not(x26),x27)),
(or(not(x27),x28))),and((or(not(x29),x30)),(or(not(x30),x31)),
(or(not(x31),x32))))),and((and(and((or(or(or(not(x1),not(x17)),
x4),x22)),(or(or(or(not(x3),not(x18)),x6),x23)),
(or(or(or(not(x5),not(x19)),x8),x24))),and((or(or(or(not(x2),
not(x21)),x10),x26)),(or(or(or(not(x4),not(x22)),x11),x27)),
(or(or(or(not(x6),not(x23)),x12),x28))),and((or(or(or(not(x9),
not(x25)),x14),x30)),(or(or(or(not(x10),not(x26)),x15),x31)),
(or(or(or(not(x11),not(x27)),x16),x32))))),(and(and((or(or(or
(not(x3),x18),x2),not(x21))),(or(or(or(not(x5),x19),x4),
not(x22))),(or(or(or(not(x7),x20),x6),not(x23)))),and((or(or
(or(not(x4),x22),x9),not(x25))),(or(or(or(not(x6),x23),x10),
not(x26))),(or(or(or(not(x8),x24),x11),not(x27)))),
and((or(or(or(not(x10),x26),x13),not(x29))),
(or(or(or(not(x11),x27),x14),not(x30))),
(or(or(or(not(x12),x28),x15),not(x31)))))))).
```

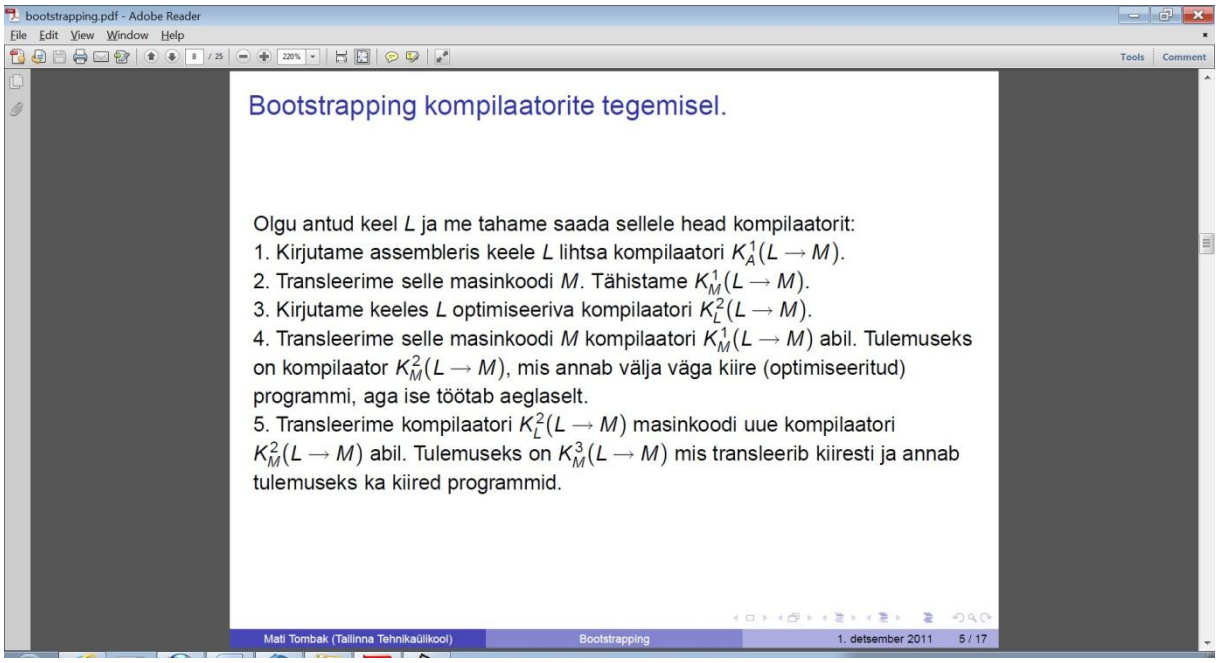
Nüüd saab leida selle kehtestavate väärtustuste arvu, tulemuseks on *281571*. Esialgse ülesande kontekstis tähendab see võimaluste arvu *4×4=16* kuninga selliseks paigutuseks tavalisele malelauale.

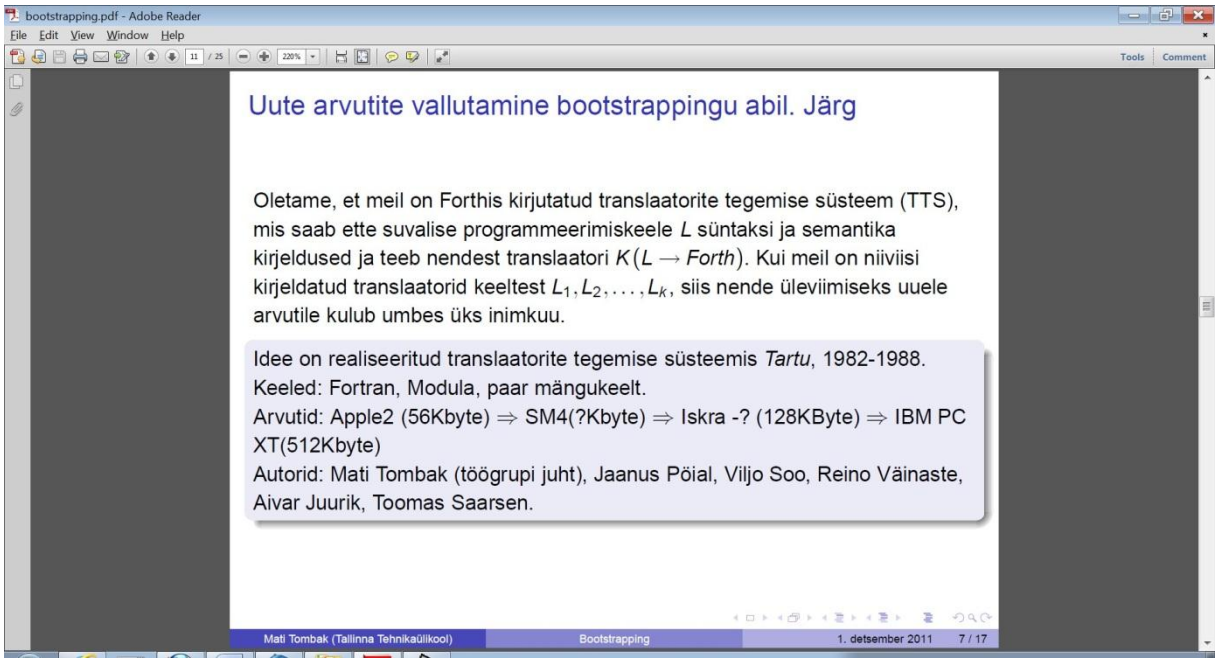
Lisa 13. Kompilaatori optimeerimine:

Bootstrapping (Mati Tombaku slaidid)









Märkigem, et siinkirjutaja ei osalenud ei kõnesoleva *TTS*i projekteerimisel ega realiseerimisel, vaid oli translaatori *Modula2* $\to$ *Forth* tegemisel kaasatud *switch*-operaatorit tegema ja pisut hiljem sama *TTS*i abil paralleelprotsessingu süsteemi *MF32* tegijaks olema.

Kasutatud materjalid

[Aho, Ullman] А. Ахо, Дж. Ульман, Теория синтаксического анализа, перевода и компиляции, том 1, Издательство «Мир», Москва, 1978.

[Backus] J. W. Backus, The syntax and semantics of the proposed international Algebraic language of the Zurich ACM-GAMM conference, Proceedings of the International Conference on Information Processing (1959), UNESCO, 125–132.

[Borland] Turbo Assembler. Users Guide, Version 1.0, Borland International, 1989.

[Chomsky] N. Chomsky, Three models for the description of languages, IRE Transactions on Information Theory 2:3 (1956), 113–124.

[c-list] http://en.wikipedia.org/wiki/Compiler_compiler (21.08.06),
http://en.wikipedia.org/wiki/List_of_compiler-compilers (21.08.06),
http://www.kosmix.com/topic/parser_generator#ixzz1NBUbDxTb (28.05.11).

[EE, 7] Eesti entsüklopeedia 7, Eesti Entsüklopeediakirjastus, Tallinn, 1994.

[Galles] David Galles, Compiler Design, Scott/Jones Inc. Publishers, 2005.

[Gries] Д. Грис, Конструирование компиляторов для цифровых вычислительных машин, Издательство «Мир», Москва, 1975.

[Henno] Jaak Henno, Formaalsed keeled, grammatikad ja translaatorid, TTÜ Kirjastus, 2006.

[Isotamm, 2007] Ain Isotamm, Programmeerimiskeeled, TÜ matemaatika-informaatikateaduskond, arvutiteaduse instituut, TÜ Kirjastus, Tartu, 2007.

[Isotamm, 2009] Ain Isotamm, Programmeerimine C-keeles *Algoritmide ja andmestruktuuride näidetel*, TÜ matemaatika-informaatikateaduskond, arvutiteaduse instituut, TÜ Kirjastus, Tartu, 2009.

[Koit&Roosmaa] Mare Koit, Tiit Roosmaa, Tehisintellekt, Tartu Ülikool, arvutiteaduse instituut, TÜ Kirjastus, Tartu, 2011.

[Lakk] Henri Lakk, Õppekeele Trigol programmist Java baitkoodi kompilaator, bakalaureusetöö, TÜ matemaatika-informaatikateaduskond,Tartu, 2008 (käsikiri).

[Lebedev] Лебедев В. Н., Введение в системы программирования, Издательство «Статистика», Москва, 1975.

[Lewis jt.] Ф. Льюис, Д. Розенкранц, Р.Стирнз, Теоретические основы проектирования компиляторов, Издательство «Мир», Москва, 1979.

[Naur] P. Naur et. al., "Report of the algorithmic language ALGOL 60," Communications of the Association for Computing Machinery 3:5 (1960), 299–314. Revised in 6:1
(1963), 1–17.

[Peder, Tombak, Isotamm] Ahti Peder, Mati Tombak, Ain Isotamm, A Meta-Compiler for Propositional Formulae, Proc. of Seventh Symposium on Programming Languages and Software Tools, Szeged, Hungary, 250–261, 2001.

[Peder, Tombak] Ahti Peder, Mati Tombak, Finding the description of structure by counting method: a case study, SOFSEM 2011: Theory and Practice of Computer Science, Lecture Notes in Computer Science 6543, Springer-Verlag, London, 455–461, 2011.

[Pius] Einar Pius, Õppekeele Trigoli kompilaator Mono baitkoodi, bakalaureusetöö, TÜ matemaatika-informaatikateaduskond, Tartu, 2009 (käsikiri).

[Power] James Power, Notes on Formal Language Theory and Parsing, 2002. http://www.cs.nuim.ie/~jpower/Courses/parsing (21.05.2007)

[Ristioja] Jaak Ristioja, Õpitarkvara ainele „Automaadid, keeled ja translaatorid“, bakalaureusetöö, TÜ matemaatika-informaatikateaduskond, Tartu, 2008 (käsikiri).

[Steele] Oliver Steele. *Visualizing Regular Expressions*, 2006. http://osteele.com/archives/2006/02/reanimator/ (15.05.2007)

[Šipilov] Nikita Šipilov, Automaatide genereerimine, bakalaureusetöö, TÜ matemaatika-informaatikateaduskond, arvutiteaduse instituut, Tartu, 2007 (käsikiri).

[Tombak, 1976] Томбак М., Об устранении конфликтов предшествования, Труды ВЦ ТГУ, Тарту, 1976, вып. 37, 60–91.

[Tombak, 1980] Томбак Мати Оскарович, Об использовании метода предшествования в синтаксическом анализе, Автореферат диссертации на соискание ученой степени кандидата физико-математических наук, Ленинград, 1980.

[TWG] The Waite Group's MS-DOS Developer's Guide, Second Edition, Howard W. Sams & Company, 1989.

[Vainikko] Vainikko Eero, Fortran95 ja MPI, Tartu Ülikool, Arvutiteaduse instituut, Tartu, 2004.

[Webster's] Webster's II, New Riverside University Dictionary, The Riverside Publishing Company, 1984.

[Wirth] Wirth Niklaus, Algorithm 265: Find Precedence Functions, CACM, 8, 10(1965), 604-605.

[W: AT] Automata Theory, 2007 http://en.wikipedia.org/wiki/Automata_theory (5.05.2007)

[W: Chomsky] http://en.wikipedia.org/wiki/Noam_Chomsky (25.01.11)

[W:kanooniline] http://et.w3dictionary.org/index.php?q=kanooniline (3.01.12)

[W: FLT] Formal Language Theory, 2003.
http://everything2.com/index.pl?node_id=113762/ (01.05.2007)

[wCPG] http://en.wikipedia.org/wiki/Comparison_of_parser_generators (16.08.08)

[wGl] http://en.wikipedia.org/wiki/Glossary (12.04.11)

[wRD] http://en.wikipedia.org/wiki/Ren%C3%A9_Descartes#Mathematical_legacy (20.05.11)

[wTB] http://en.wikipedia.org/wiki/Tony_Brooker (22.05.11)

[wCC] http://en.wikipedia.org/wiki/Compiler_Compiler (23.05.11)

[TB] http://www.essex.ac.uk/csee/people/emeritus/brooker.aspx (23.05.11)

[W: Man] http://www.cs.man.ac.uk/CCS/res/res45.htm#e (23.05.11)

[ParGen] http://www.kosmix.com/topic/parser_generator (23.05.11)

Indeks

#

#c# ..16, 17, 18, 20, 25, 48, 50, 51, 77, 85, 86, 88, 89, 90, 93, 132, 134, 136, 207, 228
#i# ..16, 17, 18, 20, 25, 48, 50, 51, 77, 85, 86, 88, 89, 90, 93, 132, 133, 134, 136, 207, 228, 276

.

.exe-fail 3
.NET 4, 14, 65, 126

1

1,1-redutseeritav eelnevusgrammatika 13, 60, 61
1|1-redutseeritav eelnevusgrammatika 13, 58

6

64-bitine arhitektuur 14

A

ahel 17, 48, 51
ajaline keerukus 31
ajalise keerukuse hinnang 33, 125
ajastub 12, 33, 35, 43, 45, 157
aksioom 11, 27, 28
alamahel 51
Algol 15, 16, 86, 87, 89, 123
  Alfa-translaator 118
alluvahel 47
alt-üles-analüüs 89
alt-üles-strateegia 23
alt-üles-tehnika 65
a^n 127
Analüsaator . 14, 43, 44, 61, 64, 66, 69, 70, 78, 82, 88, 89, 91, 92, 94, 200
analüsaatorite generaator 65
analüüs 28, 30, 33, 42, 43, 45, 50, 52, 57, 59, 61, 69
analüüsi puu 11, 14, 28, 30, 42, 43, 44, 46, 47, 48, 50, 51, 64, 66, 82, 89, 90, 91, 92, 93, 94, 95, 106, 107, 108, 118, 119, 139, 140, 202, 221, 222, 229, 231, 277
analüüsi puu teisendamine
  interpretaator 94
  kompilaator 107
andmestruktuurid 4, 5, 15, 18, 22, 23, 69, 70, 82, 84, 118
Archimedes 23
assembler 3, 15, 16, 17, 65, 66, 106, 107, 116
assemblerkood 15
Automaadid, keeled ja translaatorid 3, 6, 66, 126, 285
automaat 85

B

Backus,John W. 16, 65, 284
baitkood 4, 14, 65, 126
BNF 16, 65
BRC(1,1) 13, 57, 66, 75, 128, 138, 196
BRC(1|1) 63
Brooker, Anthony (Tony) 65, 286

C

C 6, 14, 15, 17, 18, 27, 39, 58, 65, 69, 87, 88, 95, 125, 126, 173, 284
C++ 17, 65, 124
$C_{1,1}(A)$ 13, 59, 60, 61, 64, 173, 175
$C_{1|1}(A)$ 13, 57, 58, 59, 61, 63, 64, 79
Chomsky, Noam 6, 10, 16, 26, 27, 85, 284, 285
CNF 16, 26
COBOL 88
Compiler Compiler 50, 65, 125, 144

D

derivatsioon 11, 27, 28, 30, 31, 33, 45, 57
derivatsiooni puu 11, 28, 30, 46, 47
derivatsioonist sõltuv kontekst 13, 59, 60, 61, 66
Descartes, René 57
detekteerimine 43
Dev-C++ 143
DUMMY 88, 89, 90, 91, 210

E

eelneb 12, 33, 34, 35, 36, 43, 45, 57, 157
eelnevusgrammatika 13, 37, 40, 42, 57, 58, 62, 63, 64, 66, 128
eelnevuskonflikt 37
eelnevuskonfliktide likvideerimine 38, 40, 61
eelnevusmaatriks 34, 36, 41, 52, 70, 79, 137
eelnevusmeetod 3
eelnevusrelatsioonid 12, 34
eelnevusrelatsioonide koodid 79

F

faktoriseerimine 62
fiktiivne terminal 89
Formula 3, 147, 227
Forth 3, 15, 50
FORTRAN 16, 65, 87

G

G1 27, 29, 33, 34, 37, 42, 44, 45, 47, 52, 68, 128
G3 39, 40, 42, 128, 129
G41 ..... 62, 63, 128, 129
G7 57, 58, 59, 128, 130
G8 59, 60, 61, 69, 70, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 128, 130
Galles, David 5, 284
glossaarium 6
GOTO ..10, 16, 17, 21, 25, 77, 86, 87, 88, 90, 91, 93, 110, 111, 113, 132, 136, 238, 275, 276
Grammar 66, 72, 73, 74, 75, 76, 145, 195, 196, 199
Gries, David 89, 122, 284

H

hash-võti ... 78
Help ... 128, 150
Henno, Jaak ... 3, 284
HTML ... 5, 39, 96, 151, 154, 196, 198, 199, 200, 217, 218, 226, 227, 242
hõre puu ... 49, 51, 56, 69, 91

I

identifikaatori vahekeelne kood ... 139
identifikaatorid ... 5, 10, 16, 50, 84, 85
identifikaatorite ja konstantide tabelid ... 69
identifikaatorite tabel ... 92, 139, 140
Intel ... 3
*Intel*i assembler ... 106
interaktiivne programm ... 24
interpretaator ... 14, 15, 19, 23, 24, 50, 69, 92, 93, 95, 96, 139, 239, 279
interpretaatori või kompilaatori magasin ... 93
interpreteerimine ... 15, 50

J

Java ... 4, 14, 15, 65, 66, 284
Java baitkood ... 4
Jesmin, Lauri ... 96
järgneb ... 12, 13, 33, 34, 35, 36, 43, 45, 57, 89, 157

K

kahendpuu ... 6, 18, 19, 20, 138, 198
kanooniline ... 11
kanooniline derivatsioon ... 28
Karabut, Viktor ... 275
keel *L(G)* ... 28
Kelder, Tõnis ... 4
kiire translaator ... 118
Knuth, Donald ... 16
kolmetraktiline magasin ... 43, 52
kompilaator ... 4, 14, 19, 23, 24, 50, 69, 92, 93, 107, 108, 118, 119, 122, 126, 139, 218, 243, 284, 285
konstandi vahekeelne kood ... 139
konstandid ... 5, 10, 16, 18, 25, 50, 77, 84, 85, 119, 124
konstantavaldis ... 18
konstantide tabel ... 92, 139
Konstruktor ... 14, 37, 42, 61, 64, 66, 69, 70, 84, 86, 151
Konstruktori algoritm ... 71
kontekstitundlikud keeled ... 10, 26
kontekstivaba ... 11, 16, 23, 28, 40, 42, 62, 70, 85, 127
kontekstivaba grammatika ... 11, 27, 28, 64
kontekstivabad *BRC*-analüüsitavad eelnevusgrammatikad ... 23
kontekstivabad keeled ... 10, 26
korrektne programm ... 26, 89
Kudrjavets, Gunnar ... 5, 125
KVG ... 11, 12, 27, 28, 31, 34, 37, 39, 42, 64, 70

L

L(*A*) ... 12, 34, 36, 39, 156
L(G) ... 11, 12, 28, 37, 42, 47, 58, 64
label (märgend) ... 16, 17, 19, 20, 25, 91, 92, 94, 96, 107, 109, 110, 113, 132, 136, 139, 202, 216, 218, 229, 233, 234, 235, 238, 239
Lakk, Henri ... 4, 66, 126, 284
lause vorm ... 11, 28, 42, 59, 61
lause vormi baas 11, 12, 28, 31, 33, 42, 43, 45, 47, 48, 51, 57, 59, 61, 89
LC(*A*) ... 13, 57, 58, 59, 61, 63, 80
$LC_{1|1}(A)$ ... 80
lekseem ... 10, 20, 28, 43, 84, 85, 88, 207, 208
lekseemiklassid ... 5, 10, 50
leksika ... 86, 88
LIFO ... 6, 18, 19, 22, 31, 43, 108
lihteraldajad ... 87
liiteraldajad ... 87
LISP ... 15
logi .70, 85, 90, 96, 97, 113, 115, 126, 140, 151, 157, 202, 204, 205, 226, 229, 234, 241, 242
look at ... 69

M

maketri.bat ... 272
Malgol ... 87
marker ... 17, 45, 51, 140, 208, 209, 222, 223
masinkood ... 15, 95, 106, 118, 123
masinkoodiprogramm ... 65
MASM32 ... 14, 272, 273, 274
Meriste, Merik ... 125
Milam, Stan ... 5
MIT (*Massachusetts Institute of Technology*) ... 16
mitteterminaalne (mõistete) tähestik V_N ... 25
mitteterminal ... 11, 28, 45, 47, 48, 60, 78, 185
Modula-2 ... 3, 50, 92
Modula-2 → *Forth* ... 3
MONO ... 4, 14, 65, 66, 126
MS-DOS ... 5
mõiste ... 16, 26, 27, 40, 43, 47, 60, 62, 78, 95
müra ... 21, 50, 68, 84

N

Naur, Peter ... 16, 284
Nemenman, Mark ... 4

O

objektkeel ... 14
*Ogden*i lemma ... 127
oluliselt mitmene keel ... 31, 127
operaator ... 10, 16, 17, 20, 21, 25, 48, 92, 107, 123, 132, 136, 139, 218, 276
operatsioonisüsteem ... 4
optimaalne ... 117
optimeerimine ... 117, 118, 123
ortograafiavigade parandamine ... 89

P

P1-konflikt ... 37, 41, 161
P2-konflikt ... 37, 40, 160
P4.asm ... 116, 117, 119
P4.tri ... 115, 117, 119, 122

P6 85
P6.asm 113
P6.tri 17, 23, 26, 87, 93, 113
paisktabel 78
parem kontekst 13, 57
parser ... 65, 70, 76, 82, 83, 88, 89, 90, 197, 218, 226, 227
Parser Generator 125
Peder, Ahti 1, 3, 125, 128
Pius, Einar 4, 66, 126, 285
pop 31, 43, 45
produktsiooni parem pool 11, 33, 43, 45, 137
produktsioonide hulk 11, 27, 31, 39
produktsioonide keel 65
programmeerimiskeele mõisted 10
programmi puu 22, 23, 24, 26
push 19, 31, 43
Pv1.tri 89
Pöial, Jaanus 3, 50
pööratav eelnevusgrammatika 42, 44, 58, 64, 66, 76, 128, 138, 139, 197
pööratav EG 12
pügatud puu 46

R

R(*A*) 12, 34, 36, 39, 157
Razdan-3 4
RC(*A*) 13, 57, 58, 61, 63, 80
redutseerimine 43, 51
redutseerimisprobleem 12
redutseerimistabel 78
regulaarsed keeled 10, 27
reservsõnad 10, 87

S

semantika 14, 50, 51, 52, 81, 94, 109, 137, 179, 192, 193, 196, 198, 214, 220, 224
semantikafail 66, 136
semantikakood 92, 107, 138, 139, 197, 218
semantiline analüüs 92
sisendkeel 14
skanner 70, 77, 84, 85, 86, 88, 89, 90
Soo, Viljo 3, 50
string 10, 25, 26, 44, 45, 200, 228
sõltumatu kanooniline kontekst 13, 57
sõltuv kontekst ei eristu 13, 61, 62, 63, 66, 70
sõna (programm) 11, 14, 26, 27, 28, 30, 34, 42, 44, 45, 46, 47, 50, 52, 58, 59, 60, 61, 64, 66, 82, 84, 88, 89, 108
sümbol 10, 25, 45, 61
süntaksi analüüsi ülesanne 11, 28
süntaksorienteeritud 23, 33, 50, 65, 118, 125
süntaksorienteeritud transleerimine 30
süntaktiline analüsaator 88
süsteemprogrammeerimine 5
Šipilov, Nikita 3, 5, 85, 126, 285

T

tabelid 5, 14, 23, 66, 76, 79, 82, 84, 86, 88, 91
TASM 14, 269, 271
teek 116, 122, 269
terminaalne tähestik 11, 25, 27, 85
terminal 13, 25, 33, 48, 57, 60, 63, 85, 88, 89, 178, 186
T_EX 280
TLINK 14
Tombak, Mati 3, 37, 50, 59, 62, 66, 69, 125, 127, 285
translaator 4, 5, 15, 20, 22, 50, 64, 65, 86, 92, 93
tri.t 87
tri.tt 87
Trigol 14, 15, 16, 17, 20, 23, 25, 26, 48, 51, 65, 85, 87, 88, 89, 92, 95, 106, 107, 108, 112, 118, 122, 126, 284
*Trigol*i interpretaator 95, 98
tsükkel 31
tsükliline graaf 94
TTS.. 3, 5, 14, 23, 24, 27, 33, 50, 64, 65, 66, 69, 85, 88, 89, 125, 128, 145
tts.exe 66
*TTS*i logi 62
Help 66
tuletatav 11, 27
Turing, Alan 65
tuvastaja (skanner) 5, 23, 25
tähestik 10, 25
tähestik *V* 10
täispuu 46, 51, 52, 54, 66
töömuutujad 107
tühisõna 10, 25, 26, 31

V

V 10, 25, 64
V^* 10, 11, 26, 27, 57
V^+ 10, 26
vahekeel 65
`vahekeelne kood` 92, 107, 138, 139, 198, 218
`vahekeelne programm` 139, 222
vahetult tuletatav 11, 27
vasak faktoriseerimine 62
vasak kontekst 13, 57, 60, 61, 62
vasaku ja parema konteksti hulgad 13, 57
vea mõjuulatus 91
vea parandamine 89
viga 85, 86, 89, 94, 140, 183, 187, 188, 189, 190, 191, 204, 208, 209
`VisualC++` 143, 151
V_N 10, 11, 12, 25, 26, 27, 28, 33, 34, 37, 38, 43, 47, 64, 70, 77, 79, 80, 138
Vooglaid, Aare 125
V_T 10, 11, 12, 13, 25, 26, 27, 28, 30, 33, 34, 37, 42, 43, 47, 48, 51, 57, 60, 76, 77, 79, 80, 87, 138, 139
V_T^* 10, 11, 13, 26, 27, 28, 30, 60
Võhandu, Leo 33

W

Windows 5, 33
Wirth, Niklaus 33, 66

Ü

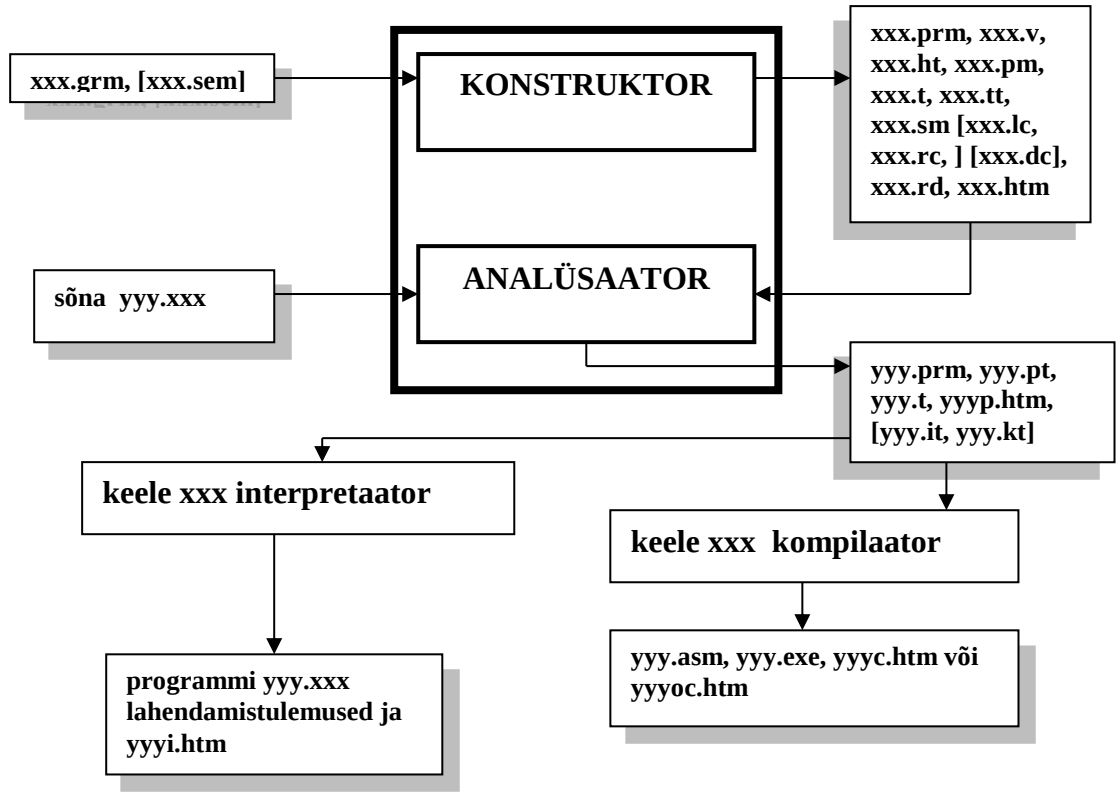
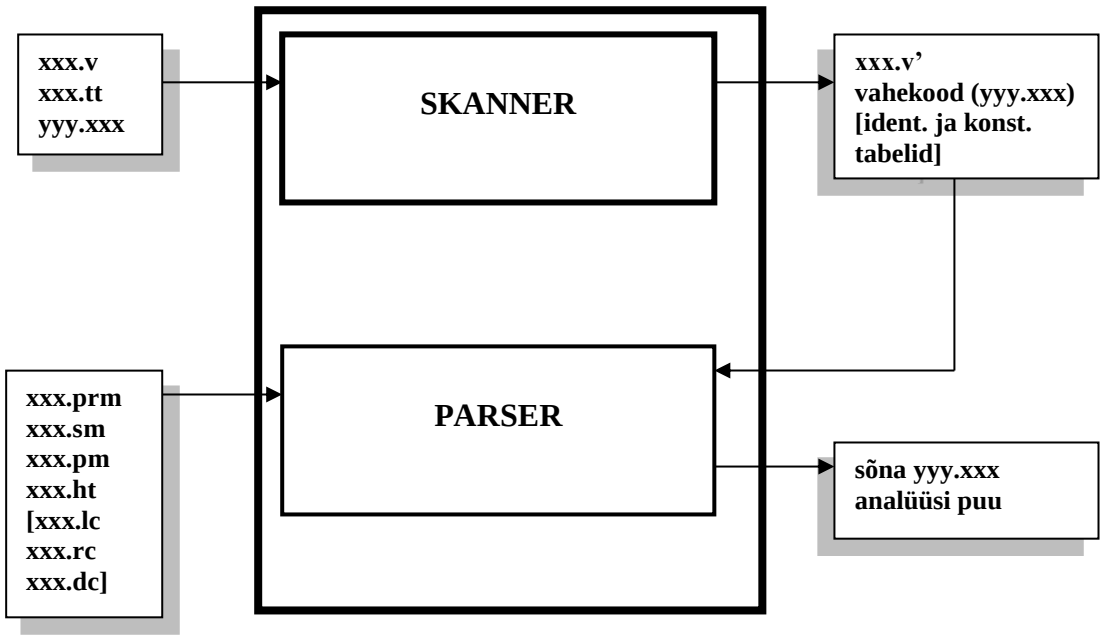
ühene grammatika 31, 33, 85
ühesed koodid 84

ülalt-alla-strateegia 22
ülesviit 47, 51

X

xxx.grm 27

TTS saab ette KVG G ning genereerib Analüsaatori, mis teeb ∀ x∈L(G) analüüsi puu. TTS ei genereeri translaatorit (interpretaatorit või kompilaatorit). **TTSi toimimine**:



Analüsaatori toimimine: