

TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Informaatika eriala

Pille Pullonen

**Tsüklite eemaldamine krüptograafiliste
protokollide analüsaatori protokolliesitusest**
Bakalaureusetöö (6 EAP)

Juhendaja: PhD Peeter Laud

Autor:..... “...” juuni 2011

Juhendaja:..... “...” juuni 2011

Lubada kaitsmisele

Professor:..... “...” juuni 2011

TARTU 2011

Sisukord

Sissejuhatus	4
1 Analüsaatori tutvustus	5
1.1 Protokollialalüüs	5
1.2 Sõltuvusgraafid	6
1.3 Mängude jada	7
1.4 Arendusvahendid	8
1.4.1 Objective Caml	8
1.4.2 OMake	8
1.4.3 uDraw(Graph)	8
2 Ülesanne	10
2.1 Tsüklite olemus	10
2.2 Kasutatavad teisendused	11
2.2.1 Tippude ühendamine	11
2.2.2 Serva eemaldamine graafist	11
2.2.3 <i>MUX</i> tipu liigutamine	11
2.2.4 <i>Or</i> tipu liigutamine	12
2.2.5 Üle <i>Id</i> tipu liigutamine	13
2.3 Tööhüpotees	13
3 Lahendus	14
3.1 Tsüklite leidmine	14
3.2 Tsükli ja teisenduse valimine	15
3.3 Tippude liigutamine üle <i>Or</i> tipu	16
3.4 Puu korrastamine pärast teisendust	17
3.4.1 Serva eemaldamine graafist	17
3.4.2 Tippude ühendamine	18
3.4.3 <i>MUX</i> tipu liigutamine	18
3.4.4 <i>Or</i> tipu liigutamine	20
3.4.5 Üle <i>Id</i> tipu liigutamine	21
3.4.6 Üle <i>Id</i> tipu liigutamine ei tööta	22
3.5 Lahenduse analüüs	22
3.6 Edasised ülesanded	24
Kokkuvõte	25
Summary	26
Viited	27
Lisad	28
Lisa 1 - Lähtekoodiga CD	28
Lisa 2 - Serva eemaldamine graafist	29
Lisa 3 - Tippude ühendamine üle puuserva	30
Lisa 4 - Tippude ühendamine üle tagasiulatuva serva	31
Lisa 5 - <i>MUX</i> tipu liigutamine üle puuserva	32

Lisa 6 - <i>MUX</i> tipu liigutamine üle tagasiulatuva serva	33
Lisa 7 - <i>Or</i> tipu liigutamine puuserval, kui sihttipp ei koonda dimensioone	34
Lisa 8 - <i>Or</i> tipu liigutamine puuserval, kui sihttipp koondab dimensioone	35
Lisa 9 - <i>Or</i> tipu liigutamine tagasiulatuval serval, kui sihttipp ei koonda dimensioone	36
Lisa 10 - <i>Or</i> tipu liigutamine tagasiulatuval serval, kui sihttipp koondab dimensioone	37
Lisa 11 - Üle <i>Id</i> tipu liigutamine puuserval	38
Lisa 12 - Üle <i>Id</i> tipu liigutamine puuserval, mille sihttipp on tõeväärtusväljundiga	39
Lisa 13 - Üle <i>Id</i> tipu liigutamine tagasiulatuval serval	40
Lisa 14 - Üle <i>Id</i> tipu liigutamine ei tööta	41
Lisa 15 - Tsüklite eemaldamine ei tööta	42

Sissejuhatus

Järjest enam tõstatatakse tehnoloogia valdkonnas turvalisust puudutavaid küsimusi. Nende lahendamiseks disainitakse erinevaid krüptograafilisi protokolle, mis lubaksid turvalist andmevahetust. Keeruline on aga kindlaks teha, kas kirjeldatud eeskiri vastab siiski turvalisusnõuetele. Efektiivne turvalisuskontroll vajaks automaattõestajaid või muud tarkvara, mis lihtsustaks tõestuste läbiviimist ja aitaks vältida vigu. PhD Peeter Laud ja PhD Ilja Tšahhirov on loonud krüptograafiliste protokollide analüsaatori, mis võimaldab kontrollida, kas protokoll säilitab andmete konfidentsiaalsust ja terviklikkust. Analüsaatoriga saab interaktiivselt protokollit muuta kuni eeskirja turvalisus on ilmne ning seejuures on tagatud transformatsioonide korrektsus. Interaktiivne teisendamine on kasulik, kui analüüsija teab protokollitugevusi ning valib teisendusi, mis viivad kiiresti turvalise lõpptulemuseni, mitte ei proovi lihtsalt juhuslikke variante.

Käsitleva analüsaatori peamiseks omapäraks on protokollit esitamine sõltuvusgraafide abil, mis võimaldavad üheselt määrata järgneva muutuse kohta. Sõltuvusgraaf on suunatud graaf, mille igas tipus on funktsioon ning servad kannavad tulemusi ühest tipust teise. Neisse võib aga teisenduste tagajärjel tekkida suunatud tsükleid. Käesoleva töö eesmärgiks on kirjeldada võimalikke tsükleid, nende arvutuslikku tähendust ning tsüklitest vabanemise viise. Transformatsioonid, mida tsüklite eemaldamine vajab, on suuremas osas analüsaatoris juba realiseeritud. Käsitletavaks ülesandeks on uurida viise nende kombineerimiseks ja järjestamiseks, et edukalt kõikidest tsüklitest vabaneda. Seega peaks analüsaatori kasutamine antud töö tulemusel lihtsustuma.

Töö algab analüsaatori põhimõtete ning osade põhjalikuma kirjeldusega, keskendudes ülesande ning lahenduse mõistmiseks vajalikule. Edasi on täpsemalt selgitatud tsüklite olemust ja nendest vabanemise olulisust ning võimalikkust. Ühtlasi on teises peatükis kirjeldatud tsüklite eemaldamiseks vajalikke transformatsioone. Viimane osa kajastab uuritavate tsüklite eemaldamise algoritmi põhilisi aspekte ja analüüsib saadud tulemusi. Lisadena on toodud CD plaadil analüsaatori lähtekood keeles OCaml koos dokumentatsiooni ning käivitamiseks olulise infoga ning teisenduste joonised.

1 Analüsaatori tutvustus

Mängude jadadel ning sõltuvusgraafidel põhineva krüptograafiliste protokollide analüsaatori idee on kirjeldatud Ilja Tšahhirovi doktoritöös [1], mis on ka antud peatüki peamiseks allikaks. Peeter Laua ja Ilja Tšahhirovi koostöös valmis analüsaatori esimene prototüüp, mis oli automaattõestaja. Et automaattõestaja vajab efektiivseks töötamiseks head transformatsioonide järjekorra valimise süsteemi, mille fikseerimine on keeruline, valmis samu põhimõtteid kasutav interaktiivne tõestaja [2]. Allikast [2] leiab lisaks veel näite analüsaatori kasutamisest protokollianalüüsis. Analüsaatori täiendamisele keskendus ka Erik Jaaniso bakalaureusetöö [3]. Käesolev peatükk tutvustab analüsaatorist eelkõige neid osi, mis on vajalikud tsüklite eemaldamise ülesande ning lahenduse mõistmiseks. Põhjalikuma ning täpsema kirjelduse leiab Ilja Tšahhirovi doktoritööst [1], kus on ühtlasi ka põhjendatud, miks analüsaatori töö on korrektne.

1.1 Protokollianalüüs

Krüptograafiliste protokollide staatilise analüüsi eesmärk on protokollide käivitamata kindlaks teha, kas protokoll vastab mingitele fikseeritud turvanõuetele. Turvaliseks loetakse protokoll, mille puhul on võimalik tõestada, et ei leidu rünnet, mis huvipakkuvaid turvaomadusi muraks. Protokollide turvalisuse hindamisel vaadatakse üldjuhul kolme aspekti: konfidentsiaalsus, terviklikkus ja käideldavus. Analüsaator keskendub neist kahele esimesele, mis väljendavad eelkõige ebasoovitava funktsionaalsuse puudumist. Vastanduvalt tähendab käideldavus soovitava funktsionaalsuse eksisteerimist. Seega võimaldab analüsaator kontrollida, kas informatsiooni saavad näha ning muuta ainult need isikud, kellel on selleks õigus. Protokoll rahuldab konfidentsiaalsuse nõuet, kui tema käitamisel saadud tulemus ei anna informatsiooni salajase väärtuse kohta.

Analüüsis kasutatakse aktiivse vastase mudelit, mis eeldab, et kogu võrgusuhtlus ja protokollis osalevate osapoolte töö järjestamine on vastase kontrolli all. See mudel vastab üldisele reaalsele võrgule, millest vähemalt mingid osad on sageli mõne kolmanda osapoole kontrolli all. Ühtlasi on aktiivse vastase mudel tugevam passiivse vastase mudelist, kus vastane ainult kuulab informatsiooni, mitte ei mõjuta seda. Kui protokoll on turvaline analüsaatoris kasutatavas aktiivse vastase mudelis, on ta seda ka nõrgemates mudelites.

Analüsaator kasutab kahest võistlevast krüptograafia turvalisuse mudelist Goldwasser-Micali 84 [4] ehk arvutuslikku mudelit. See tähendab, et protokollis kasutatavate krüptograafiliste funktsioonide kohta eeldatakse, et nad on turvalised arvutuslikus mudelis. Vastavalt sellele võib krüpteeritud teksti ilma võtmeta dekrüpteerimine olla küll võimalik, aga dekrüpteerimise õnnestumise tõenäosus on väga väike. Ebaoluliseks loetakse tõenäosust, mis saadakse kasutatud võtme pikkusest rakendades sellele kaduvväikest funktsiooni (*negligible function*). Teine konkureeriv turvalisushinnang põhineb Dolev-Yao 83 [5] ehk formaalsel mudelil, mis eeldab, et võtit teadmata on dekrüpteerimine võimatu. Turvalisust on lihtsam tõestada Dolev-Yao mudelis, ent arvutuslik mudel on üldiselt reaalsele elule lähemal, sest krüptograafilised primitiivid pole üldjuhul ideaalsed mustad kastid. Erinevalt vastast puudutavatest mudelitest ei ole erinevates krüptograafia mudelites saavutatud tulemused üksteisele kergelt ülekantavad. Protokoll saab turvaline olla ainult siis, kui kasutatud krüptograafilised funktsioonid on turvalised. Analüsaator lubab protokollis kasutada avaliku võtme krüptosüsteeme, salajase võtme krüptosüsteeme ning digitaalallkirja, mis on arvutuslikus mudelis tugevad.

Analüsaatori töö võtmepunktideks on protokollide eristamatus ning transformatsioonid.

Kaks protokoll on eristamatud, kui neile oma soovi järgi sisendeid andes ei ole väljundi järgi võimalik kindlaks teha, kumma protokolliga töötati. Ühtlasi, kui üks neist protokollidest on turvaline, on seda ka teine. Transformatsioonid lubavad protokoll muuta ainult nii, et tulemus oleks algsest eristamatu. Analüüsi alustatakse meid huvitavast protokollist, mis sisaldab mõnda salajast väärtust ning algse protokoll saab lugeda nimetatud mudelites konfidentsiaalsuse omadust säilitavaks, kui transformatsioonidega jõutakse protokollini, mis salajasest väärtusest enam ei sõltu. Nimelt järeldeb viimasest protokollist, et ühegi protokollijas esinenud protokoll väljund ei anna informatsiooni salajase väärtuse kohta.

Protokoll säilitab terviklikkust, kui iga protokoll käitamine vastab mingitele fikseeritud nõuetele. Seega tõenäosus, et protokoll käitamine nõuetele ei vasta, on tühine. Terviklikkuse hindamiseks uuritakse sõnumite vahetamise ajastumist (*Correspondence assertion*). Selleks fikseeritakse erinevate protokoll osade algused ja lõpud ning tulemus on kooskõlaline, kui ühe osa lõpule eelnes tema algus. Vahetust nimetatakse injektiivseks, kui igale lõpule vastab eraldi algus.[6]

Tulenevalt arvutusliku mudeli kasutamisest on õnnestunud tõestuse tulemust õige sõnas-
tada viisil "protokoll on turvaline (välja arvatud juhtudel, mis esinevad ebaolulise tõenäo-
susega), kui kasutatud krüptograafilised meetodid on arvutuslikus mudelis turvalised". Kui
valitud transformatsioonide komplekti abil ei õnnestunud salajast väärtust eemaldada, ei jä-
reldu sellest, et protokoll on ebaturvaline, sest mõni teine transformatsioonide järjestus võib
viia positiivse tulemuseni.

1.2 Sõltuvusgraafid

Protokollid on analüsaatoris esitatud sõltuvusgraafidena. Sõltuvusgraaf on suunatud graaf, mille igas tipus on funktsioon, mis tagastab väärtuse. Funktsiooni väärtused võivad olla bitijadad või tõeväärtused. Kuni bitijada tagastav tipp pole oma väärtust arvutanud, on selleks \perp ehk viga. Tõeväärtust tagastava tipu algväärtus on väär (*false*). Sisendservad näitavad, millistes tippudes arvutatud väärtusi tipp oma arvutusteks vajab, ning väljundservad, millised teised tipud vajavad selles tipus arvutatud tulemust. Sisendservad on seotud portidega, mille järgi tipp teab, millise argumendi väärtuse ta sellelt servalt saab. Lisaks võib tipul olla tõeväärtusega kontrollsisend, mille olemasolul arvutab tipp väärtuse ainult siis, kui kontrollsisend on tõene. Väär tõeväärtusega kontrollsisendi puhul tagastatakse \perp . Andmete graafi sisse toomiseks on olemas erilised sisendtipud, millel ei ole sisendservi, ning analoogiliselt on olemas väljundservadeta väljundtipud, mis kujutavad andmete kaitsmata võrku saatmist.

Protokolliesitusest tingitult on tippudel kasutajaliideses veel andmeid lisaks. Joonisel lõplikud sõltuvusgraafid kujutavad üldjuhul lõpmatut sõltuvusgraafi ning vajavad seetõttu dimensioone. Tipp, mille dimensioon $k \geq 1$, on lõpmatus graafis tegelikult loenduv arv tippe. Graafiesituses olev serv tähendab lõpmatus graafis servi samade koordinaatidega tippude vahel. Kui serva lähtetipul on vähem dimensioone kui sihttipul, siis on lõpmatus graafis igal lähtetipul servi lõpmatul arvul sihttippudesse. Kui serva sihttipul on vähem dimensioone kui lähtetipul, siis on sihttipul lõpmatu kogus sisendeid - üks igast lähtetipu kordusest. Dimensioone muutvatel tippudel on alati info selle kohta, milliseid dimensioone nad muudavad. Lisaks on mõnedel eritippudel info selle kohta, milliseid koordinaate nad võrdlevad. Üheks oluliseks dimensioone muutvaks tipuks on *LongOr*.

Püstitatava ülesande seisukohast on oluline, et graafi tipud jagunevad funktsioonide pool-
lest kahte klassi. Ranged tipud arvutavad oma väärtuse ainult siis, kui nad on saanud kõik
sisendid. Mitteranged tipud (esindajateks *MUX* ja *Or*) võivad väärtuse tagastada ka varem.
MUX tähiseiga tipud on ühendavad (*multiplexor*) tipud, mis annavad välja ainsa oma sisen-

ditest, mis ei ole \perp . Kui mitu *MUX* tipu sisendit erinevad veast, tähendab see, et graafis on ebakõla ning väärtustamine lõpetatakse. Töö veaga lõpetamine on vastasele näha, aga korrektsete transformatsioonide korral ei juhtu seda sagedamini kui ebaolulise tõenäosusega. *Or* tipud tähistavad loogilist disjunktsiooni. *MUX* ja *Or* on ainsad tipud, mille dimensioon võib olla väiksem kui nende argumentide dimensioonid. Tervikluse kontrolliks on veel ka eritipud *begin* ja *end*, mis võimaldavad uurida operatsioonide järjekorda.

Protokollide sõltuvusgraafideks tõlkimise algoritm, mis säilitab konfidentsiaalsuse ja terviklikkuse kohta käiva info, on kirjeldatud Ilja Tšahhirovi töös [1]. Sõltuvusgraafide kasutamine analüsaatoris on võimalik tänu sellele, et pärast analüüsi lõppu ei ole vaja lõppgraafi uuesti protokollile kujule tõlkida. Tagasi tõlkimine osutuks tõenäoliselt keeruliseks, sest graaf võib sisaldada osi, mida on raske järjestada.

1.3 Mängude jada

Antud kontekstis on mänguks ühe konkreetse protokollile käitamine ning mängude jadaks analüüsija valitud transformatsioonide tulemusel tekkiv protokollide järjend. Edukas tõestuses väheneb vastase võiduvõimalus ühelt mängult teisele liikudes tühiselt, kuid jada viimases mängus ei ole tal enam võiduvõimalust. Järjestikuste mängude protokollid on vastase jaoks arvutuslikult eristamatud. Aktiivse vastase jaoks eristamatu tähendab, et ta ei suuda protokollile tööd mõjutades olulise tõenäosusega öelda, kumba protokollile jooksutatakse.

Üheks mänguks on ühe sõltuvusgraafi käivitamine, mis algab juhuslike väärtuste ning salajase väärtuse fikseerimisega. Alguses on kõikide bitijadasid arvutavate tippude väärtus \perp ning tõeväärtust arvutavate tippude väärtus väär. Edasi korratakse järgnevat samm, kuni vastane tsükli katkestab:

1. Vastane sätib mõne *req* (tipp määrab, millist osa graafist hakatakse arvutama) tipu tõseks ja/või fikseerib (aga ei muuda) mõne *receive* (tipp, mis kujutab võrgust info saamist) tipu väärtuse.
2. Sisenditest lähtuvalt arvutatakse graafi tippude väärtusi nii kaua kui võimalik.
3. Vastasele antakse kõik *send* (tipp, mis kujutab info võrku saatmist) tippude väärtused, mis on arvutatud (erinevad väärtusest \perp).

Vastane võib mängu, kui tal õnnestub saada salajasest väärtusest mittetühisel määral sõltuv väljund. Analüüsi käigus graafi ei käivitata, aga käivitamise olemus on oluline, et fikseerida sobivad teisendused.

Mängude jada järgmine graaf saadakse, kui vaadeldavale graafile rakendatakse mõnda transformatsiooni. Transformatsiooni käigus asendatakse vaadeldava graafi mõni alamgraaf arvutuslikult eristamatu alamgraafiga. Transformatsioonid on disainitud selliselt, et nad vähendavad graafi keerukust, muudavad ilmseks mõne operatsiooni varjatud tähenduse või kõrvaldavad andmevoogusid. Seega toovad transformatsioonid selgemalt välja, millised on protokollile avatud andmevood ning mida kaitsevad krüptograafilised operatsioonid [6]. Kõik transformatsioonid on realiseeritud selliselt, et ühele graafile transformatsiooni rakendamisel on alati tulemuseks üks graaf, mis on võimalik, sest üks graaf suudab hoida infot protokollile erinevate käivitamiste kohta. Ühtlasi säilitavad kõik defineeritud transformatsioonid protokollide eristamatuse. Vastavad tõestused on olemas Ilja Tšahhirovi doktoritöös [1].

1.4 Arendusvahendid

Käesoleva peatüki eesmärk on lühidalt tutvustada keeli ja rakendusi, mida on vaja analüsaatori lähtekoodi kompileerimiseks või rakenduse kasutamiseks. Kompileeritud analüsaatori kood suhtleb oma graafilise väljundi `uDraw(Graph)`'i ga kasutades UNIX-i torusid.

1.4.1 Objective Caml

Objective Caml ehk OCaml on üldotstarbelise funktsionaalse programmeerimiskeele Caml peamine realisatsioon. Camli arendab Prantsusmaa riiklik arvutiteaduse ja kontrolli uurimiskeskus INRIA (*Institut National de Recherche en Informatique et en Automatique*). Camli esimesed kasutusvaldkonnad olid automaattõestajad, kompilaatorid, interpretaatorid ning programmide analüsaatorid. Tänapäevaks on ta eelkõige kasutuses akadeemilistes ringkondades, kuid ka mõningates suurkorporatsioonides. Camli disainimisel üritati eelkõige tagada programmide töökindlus ning turvalisus. Camli erinevad versioonid ja INRIA loodud kompilaatorid on vabavaralised [7, teema "About Caml"].

Objective Caml lisab Camlile objektorienteeritud kihi ning laialdase moodulite süsteemi. Analüsaator on kirjutatud keeles OCaml, kuid selle objektorienteeritud võimalusi tegelikult ei kasutata. OCaml'i distributsioonis on olulisel kohal masinkoodi kompilaator `ocamlopt`, baitkoodi kompilaator `ocamlc`, interaktiivne keskkond `ocaml`, parser ja printer `camlp4` ning dokumentatsiooni generaator `ocamldoc`. [7, teema "Objective Caml"] Analüsaatori kompileerimiseks on lisaks kompilaatorile vaja ka `ocamlfind` [8] tööriista, mis lihtsustab kompileerimisel teekide ning skriptide lisamist, ning OCaml'i graafide teeki `ocamlgraph` [9], mis pakub erinevaid graafide implementatsioone ning algoritme.

Kasutajaliidese täiendamiseks on kasutatud `lablTk` teeki, mis lisab OCamlile võimaluse rakendada Tcl/Tk graafilise liidese loomise võimalusi. `LablTk` lubab kirjutada rakendusele lisaaknaid, mis on olulised näiteks kasutajalt lisainformatsiooni küsimiseks. `LablTk` kasutamine on loomulik ka seetõttu, et graafide kuvamiseks kasutatud programmile `uDraw(Graph)` saab saata Tcl/Tk käsked või infot koodifailide kohta. `uDraw` interpreteerib seda koodi sisseehitatud interpretaatoris ning nii on võimalik luua lisaaknaid. Analüsaatoris on realiseeritud küll variant, kus rakendus loob ja kuvab ise oma aknaid, sest sellel viisil saab Tcl/Tk ka ligipääsu rakendusele, mitte ainult graafilisele keskkonnale. `LablTk` kasutamiseks peab olema installeeritud Tcl/Tk, milline versioon täpselt, sõltub süsteemist. `LablTk` teek kuulub OCaml'i uuematesse versioonidesse.

1.4.2 OMake

Analüsaatori lähtekoodil on kompileerimiseks ja käivitatava faili moodustamiseks olemas ehitusskript `OMakefile`, mida käitab programm `OMake` [10]. `OMake` on eelkõige loodud suurte projektide ehitamiseks, toetades projekte, mis koosnevad paljudest alamkaustadest. Tema peamisteks tugevusteks on töökindel sõltuvusanalüüs ning skriptitavus, kusjuures on olemas teek, mis toetab erinevaid ülesandeid C-s, C++-s, OCamlis ja LaTeXis. Lisaks on olemas `OMake`'i interaktiivne käsuriida `osh`. `OMake` toetab kõiki platvorme, mida toetab ka OCaml.

1.4.3 uDraw(Graph)

`uDraw(Graph)` [11] on Bremeni ülikoolis loodud vahend graafide, voodiagrammide ja muude jooniste loomiseks, mis oskab diagramme automaatselt kujundada. Lisaks on `uDraw(Graph)`'i

võimalik ühendada omaloodud programmidega ning kasutada andmete visualiseerimiseks. Analüsaator kasutab uDraw(Graph)'iga suhtlemiseks UNIX-i torusid, aga uDraw(Graph) toetab ka TCP/IP ühendusi. Eraldiseisva rakendusega koos töötades tegeleb uDraw(Graph) talle saadetud graafi kuvamisega ning kasutaja tehtud valikute rakendusele edastamisega. Lisaks suudab uDraw(Graph) kuvada rakendusespetsiifilisi menüüsid, mille abil saab alus-rakendusega suhelda. Analüsaator võimaldab uDraw(Graph)'i abil teisendatavaid graafe salvestada ning töötada varasemalt salvestatud failidega. Peamiseks kasutajaliidese piiranguks on, et uDraw(Graph) lubab korraga valida mitu tippu või ühe serva, mis tähendab, et analüsaatoris realiseeritud transformatsioonid saavad samuti sisendiks kas ühe serva või palju tippe.

Graafe kuvatakse hierarhiliselt nii, et samal tasemel olevad tipud paiknevad teineteise suhtes horisontaalselt ning kaared on vaikimisi suunatud alla. Kaarte ning tasemete suunda on soovi korral võimalik muuta, kuid kui analüsaator uDraw(Graph)'le uuendatud graafi saadab, kasutatakse jälle vaikesätteid. Kui valitakse serv, mis läbib mitut tasandit, siis kuvatakse iga läbitava tasandi kohta vastaval kõrgusel täpik. Algselt kuvatakse graaf joonistades enne tipu esmalt leitud järglased, kuid menüüvalikute abil on võimalik seda muuta. On olemas algoritm, et vähendada lõikuvate servade arvu, kuid see võib olla ajamahukas ülesanne ning tuleb alati eraldi välja kutsuda. Graafi uuendamisel üritab uDraw(Graph) võimaluse piires säilitada eelnevat kujundust, mis tähendab, et reeglina ei pea pärast transformatsiooni hakkama uuesti otsima, millise graafi osaga töötati.

2 Ülesanne

Analüsaatorit on lihtsam kasutada, kui õnnestub kiiresti eemaldada selgelt ebaoluline informatsioon. Tsüklid teevad keerulisemaks nii graafide kujutamise kui ka nendega opereerimise. Seega on huvipakkuv uurida, mida tsüklid protokollis käitumise mõttes tähendavad ja kas sama tulemust on võimalik saada protsesse tsükli asemel lineaarselt järjestades. Ühtlasi tutvustab see peatükk ülesande lahendamiseks vajaminevaid analüsaatoris olemasolevaid transformatsioone. Esitatud on nende toimimise kirjeldus ja korrektsuse tunnetuslik põhjendus. Korrektsuse tõestused on olemas Ilja Tšahhrovi doktoritöös [1].

2.1 Tsüklite olemus

Protokolliesituseks kasutatud sõltuvusgraafidesse võib transformatsioonide tulemusel tekkida suunatud tsükleid. Algsetes graafides üldiselt tsükleid ei ole, aga mõningad teisendused, näiteks früptograafilised transformatsioonid, võivad neid tekitada. Tsükklisse ei saa kuuluda sisend- või väljundtipud, sest igal tsükli tipul peab olema vähemalt üks sisend- ja üks väljundserv. Seega on kõik tsükli tipud sõltuvusgraafi mõttes harilikud ning jagunevad rangeteks ja mitterangeteks tippudeks.

Tsüklites, mis sisaldavad ainult rangeid tippe, ei saa ükski sinna kuuluv tipp kunagi oma väärtust arvutada. Nimelt vajab iga range tipp arvutamiseks kõiki oma sisendeid, seega kui valida tsüklist üks konkreetne tipp, siis see vajab arvutamiseks kindlasti ka talle tsükli eelneva tipu väärtust. See eelnev tipp vajab omakorda talle eelneva tipu väärtust. Niimoodi mööda tsükli tagasi liikudes jõuab paradoksini, kus valitud tipp vajab enda väärtuse arvutamiseks iseenda väärtuse teadmist. Seega ei saa see tsükkel muuta protokollis käitumise tulemusi ning tsükli saab eemaldada kasutades olemasolevat teisendust “*Cut edge with strict edges from target to source*” (“Eemalda serv, mille sihttipust lähtetipuni leidub rangetest servadest tee”). Rangeteks loetakse servi, mille sihttipp on range.

Tsüklid, mis sisaldavad ainult ühte mitteranget tippu, arvutavad oma väärtuse ainult siis, kui mitterange tipp saab kuskilt tsüklist väljast väärtuse, mille ta saab välja anda. Nimelt saaks mitterange tipp anda välja tsüklist tuleva väärtuse, kui kõik ranged tipud tsükli on väärtustatud, sealhulgas peab arvutatud olema ka range tipp, mis vajab vaadeldava mitterange tipu väärtust. *Or* tipp on sellel hetkel juba tsükklisse andnud tõese tõeväärtuse ja tsüklist tulev tõeväärtus seda ei mõjuta. *MUX* tipp peab andma välja esimese veast erineva tulemuse ja ei tohiks kunagi saada teist, järelikult ei tohiks ta ka tsüklist tulemust saada. *Or* ja *MUX* on hetkel analüsaatori ainsad mitteranged tipud. Seega ei anna üks mitterange tipp kunagi välja tulemust, mille ta võiks tsüklist saada ja on võimalik eemaldada tsükkel, kustutades teisendusega “*Cut edge with strict edges from target to source*” mitterangesse tippu tsüklist tuleva sisendserva.

Tsüklites, mis sisaldavad mitut *MUX* tippu, on võimalik teisendustega “*Move Over MUX*” (“Liiguta üle *MUX* tipu”) need tipud liigutada niiviisi, et nende vahel on tsükklisse kuuluv serv. Servaga ühendatud *MUX* tipud on võimalik ühendada kasutades teisendust “*Fuse two Ands / Ors / LongOrs / MUXes*” (“Ühenda kaks *And / Or / LongOr / MUX* tippu”). Teisenduse tulemusel jääb tsükklisse kahe tipu asemel ainult üks ja säilivate tippudega saab teisendust korrata. Seega on need tsüklid “*Move Over MUX*” ja “*Fuse two Ands / Ors / LongOrs / MUXes*” transformatsioonid kasutades teisendatavad eelnevalt kirjeldatud ühe *MUX* tipuga tsükliteks. *Or* tippude jaoks on siinkohal tarvilik defineerida analoogiline transformatsioon “*Move Over Or*” (“Liiguta üle *Or* tipu”), mis võimaldaks vahetada *Or* tipu ja tema mõne järglase järjekorra nii, et *Or* saaks oma endise järglase järglaseks. Selle teisenduse olemasolul

saab mitme *Or* tipuga tsükli eemaldada kasutades “*Move Over Or*” ja “*Fuse two Ands / Ors / LongOrs / MUXes*” teisendusi analoogiliselt *MUX* tippe sisaldava tsükliga.

Kui tsükliks on mõlemat tüüpi mitterangeid tippe, on neid võimalik tsükliks liigutada kasutades defineeritavat “*Move Over Or*” teisendust ja olemasolevat “*Move Over MUX*” teisendust, aga et ühe sisendid ja väljundid on bitijadad ning teise omad tõeväärtused, ei saa nad kunagi olla servaga ühendatud. *MUX* tipp muutub enne olukorda, kus tema järglaseks oleks *Or* tipp, ka ise *Or* tipuks, seega on *MUX* tippe liigutades võimalik saada erinevaid mitterangeid tippe sisaldavast tsüklist ainult *Or* tippe sisaldavad tsükliks.

Keeruliseks teeb ülesande tõsiasi, et *MUX* ja *Or* tipu liigutamise teisendused tekitavad üldjuhul graafi tippe ja servi juurde. Seega võib suureneka ka tsükli arv graafis ning võib juhtuda, et edukalt õnnestub eemaldada vaid osa tsükke. Lisaks mõjutab tsükli eemaldamist see, kuidas täpselt defineerida teisendus *Or* tipu jaoks.

2.2 Kasutatavad teisendused

Tsükli eemaldamise kirjelduse järgi tuleb tsüklist vabanemiseks kasutada mitmeid eeldefineeritud transformatsioone. Käesoleva peatüki eesmärk on selgitada, kuidas need teisendused töötavad ning kuidas neid kasutada. Lisaks kirjeldatakse, kuidas teisendus “*Move Over Or*” võiks töötada.

2.2.1 Tippude ühendamine

Teisendus menüünimega “*Fuse two Ands / Ors / LongOrs / MUXes*” võimaldab ühendada kahte sama tüüpi tippu, kui nende vahel on olemas serv. Teisendus rakendatakse servale, mis on ühendatavate tippude vahel. Transformatsiooni tulemusel kaob nende vahel olev serv ning alluv saab endale ka kõik ülemtipu sisendid. Alles jääb ka ülemine tipp endiste sisendite ning ülejäänud väljunditega. Tsüklist, mis enne läbis mõlemat ühendatavat tippu, saab nüüd mõnest ülemise tipu eellasest minna otse alumisse tippu, seega jääb tsükliks seda tüüpi tippe vähemaks. Üldjuhul tekitab antud teisendus servi juurde. Lisas 1 on teisendus kirjeldatud failides *GrbTrFuseTwoAndOr.ml* ja *GrbTrFuseTwoLongOr.ml*, millest *Or* ja *MUX* tippe puudutab esimene fail. Lisas 3 on toodud joonis teisenduse tööst, kui seda rakendada algseisus märgitud servale. Graafis ei ole joonisel kujutatud servade liikidel tähendust.

2.2.2 Serva eemaldamine graafist

Teisendus menüünimega “*Cut edge with strict edges from target to source*” rakendatakse servale, mille sihttipust lähtetippu leidub graafi servadest suunatud ahel nii, et iga ahelale jääva serva sihttipp on range. Seega valitud serva sihttipp võib olla mitterange, aga teised tipud ahelal ei tohi seda olla. Teisendus eemaldab valitud serva ning serva sihttipp saab eemaldatud serva asemel sisendserva tipust *Error* (või *False*), mis tähendab vigast sisendit. See vastab tsükli eemaldamise üldisele ideele, sest valitud serva sihttipp ei saa sellelt servalt saada kasutatavat sisendit. Lisas 1 on teisendus kirjeldatud failis *GrbTrCutStrictCycle.ml* ning lisas 2 on joonis teisenduse tööst.

2.2.3 MUX tipu liigutamine

Teisendus menüünimega “*Move over MUX*” rakendub servale, mille lähtetipp on *MUX* ning liigutab sihttipu üle *MUX* tipu nii, et sihttipu sisendiks on *MUX* tipu sisendid. Sõltuvalt sellest, kas sihttipp oli tõeväärtustüüpi või bitijada tüüpi väljundiga, tekib graafi uus *Or* või

MUX tipp. Täpsemalt tekib sihttipust nii palju eksemplare kui palju *MUX* tipul on sisendeid ning neil kõigil on kõik algse sihttipu sisendid ning üks paarikaupa erinev *MUX* tipu sisend. Igal uuel tipul on üks väljund, mis läheb uude *MUX* või *Or* tippu, millel on omakorda kõik algse sihttipu väljundid. *MUX* tipule, millest algavale servale teisendus rakendati, jäävad alles kõik sisendid ning kõik väljundid peale sihttipu suubuva.

Sihttipu alluvate jaoks jääb sisendväärtus samaks. Nimelt, kui varem sai sihttipp ühe sobiva argumendi *MUX* tipult, siis nüüd arvutab väärtuse esimene, mis mõnest *MUX* tipu sisendist tulemuse saab, kusjuures see esimene sisend on sama, mille *MUX* oleks sihttipule andnud. Seejärel annab alumine *MUX* või *Or* selle sihttipu järglastele edasi. Kui teisenduse alguses läbis tsükel mõnda *MUX* tipu sisendit, edasi *MUX* tippu ja teisenduse sihttipu, siis pärast saab vaadata tsükli, mis läbib seda *MUX* tipu sisendit, sisendile vastavat teisenduse sihttipu eksemplari ning lõpuks tekkinud *Or* või *MUX* tippu. Tekkinud mitterange tipp on nüüd tsüklis vaadatud sihttipu järel, seega liikunud madalamale.

Teisendus on lisas 1 kirjeldatud failis *GrbTrMoveOverMux.ml*, mis vastab bitijada väljundiga tipu liigutamisele ja failis *GrbTrMoveDtoCMux.ml*, mis vastab tõeväärtustüüpi väljundiga tipule. Lisas 5 on toodud ka transformatsiooni joonis tõeväärtustüüpi sihttipu kohta, bitijada tüüpi väljundi puhul oleks joonisel ainsaks erinevuseks *MUX* tipp *Or* tipu asemel. Graafis ei ole joonisel märgitud servade tüüpidel tähtsust ja kõigi märgitud servade kohal on graafis servad.

2.2.4 *Or* tipu liigutamine

Uus defineeritav teisendus menüünimega “*Move over Or*” peab lubama liigutada *Or* tipu järglase tema eellaseks. Teisendus peaks rakenduma *Or* tipust väljuvale servale. Kui *Or* tipu järglane on *And* tipp (loogiline konjunktsioon), saab kasutada loogilise disjunktsiooni ja konjunktsiooni distributiivsust $A_1 \& \dots \& A_n \& (B_1 \vee \dots \vee B_m) \equiv A_1 \& \dots \& A_n \& B_1 \vee \dots \vee A_1 \& \dots \& A_n \& B_m$.

Kui *Or* tipule järgneb mõni bitijada välja andev tipp *OP*, on võimalik sisse tuua *Id* tipp, millel on sisendiks *OP* ja kontrollsisendiks *Or* ning väljundiks *OP* tipu kõik väljundid. Sealjuures jääb *Id* tipp *OP* tipu ainsaks järglaseks. See asendus ei muuda sisuliselt midagi, sest *Id* tipp annab välja *OP* tipult saadud sisendi ning tema kaudu jõuab see kõigi *OP* tipu järglasteni. Samas on võimalik kasutada defineeritud teisendust “*Pass over an Id-node*” (“Tõsta üle *Id* tipu”), et sobiv *Id* tipu järglane liigutada *Or* ja *OP* tipu järglaseks. Lisa 7 kujutab selle teisenduse põhijuhtu.

Kui *Or* tipu järglane on *LongOr*, võib *LongOr* tipu asetada *Or* tipu ja iga tema eellase vahele. Täpsemalt on see sarnane *MUX* tipu jaoks kirjeldatud teisendusega: tekib uus *Or* tipp, mille sisenditeks on *LongOr* tipud ning väljunditeks algse *LongOr* tipu väljundid. *LongOr* tipul on alati ainult üks sisend, algsel *LongOr* tipul tuleb sisendserv *Or* tipust, loodavatel mõnest *Or* tipu sisendist. Loogiliselt jääb see graafi fragment samaväärseks algsega, sest *Or* ja *LongOr* annavad alati välja esimese tõese väärtuse. Seega varem jõudis tulemus tippu *LongOr* siis, kui *Or* tippu oli tulnud mõne tõese tõeväärtusega sisend ning sealt said selle *LongOr* tipu järglased. Nüüd saavad nii *Or* kui eelmise *LongOr* tipu järglased tõese väärtuse kohe, kui *Or* tippu jõuab tõene väärtus.

Lahendust on täpsemalt kirjeldatud peatükis “Tippude liigutamine üle *Or* tipu”. Transformatsiooni sellise defineerimise peamiseks plussiks on see, et maksimaalselt saab kasutada analüsaatori olemasolevat. Vastanduvalt oleks võimalik defineerida *Or* tipu liigutamine üle *OP* tipu sarnaselt *MUX* teisendusega üle tõeväärtustipu, kus *Or* tipu asemele tuleks alamiseks tipuks uut tüüpi tipp.

2.2.5 Üle *Id* tipu liigutamine

Teisendus menüünimega “*Pass over an Id-node*” rakendub mõnele *Id* tipu väljundservale, mille sihttipp liigutatakse üle *Id* tipu. Kui sihttipp on tõeväärtustüüpi väljundiga, siis *Id* kontrollsisend ja sihttipu väljund ühendatakse *And* tipuga, mille järglased on sihttipu järglased ning sihttipp saab *Id* tipust tuleva sisendi asemel sisendi *Id* tippu tulevast bitijada sisendist. Kui sihttipp on bitijada väljundiga ja tal on kontrollsisend, siis ühendatakse tema kontrollsisend ja *Id* tipu kontrollsisend *And* tipuga, mille tulemus on sihttipu kontrollsisend. *Id* tipust tuleva sisendi asemel on sisendiks *Id* tippu tulev bitijada tüüpi sisend. Lisas 1 on teisendus kirjeldatud failides *GrbTrPassOverId.ml* ja *GrbTrDtoCOverId.ml*, millest esimene puudutab bitijada väljundiga tippe ning teine tõeväärtustüüpi väljundiga tippe. Lisades 11 ja 12 on teisenduse skeem juhul, kui seda rakendatakse pärast “*Move over Or*” teisendust.

Üle *Id* tipu liigutamise teisendus ei ole defineeritud juhul, kui sihttipul pole tõeväärtus-sisendit ega -väljundit. Tsüklis tähendab see, et olukorras, kus on järjest *Or* tipp, *Id* tipp ja *MUX* tipp, lõpetab “*Pass over an Id-node*” teisendus veateatega “*Expansion failed: The target node has no PortSingleB port*” (“Töö ebaõnnestus: Sisendtipul ei ole PortSingleB porti (ehk tõeväärtussisendit)”). Seega vajab see situatsioon eraldi lähenemist.

Tsüklite eemaldamise kontekstis on see teisendus oluline ainult siis, kui *Id* tipu kontrollsisend tuleb *Or* tipust ning tsükel läbib seda *Or* tippu, *Id* tippu ning seda *Id* tipu järglast, mille jaoks teisendust rakendatakse. Pärast teisendust on *Or* tipu järglaseks *And* tipp. Edasi vaadatakse tsüklit, mis läbib *Or* ja *And* tippe ning siis kas sihttippu või tsüklisse kuuluvat sihttipu järglast.

2.3 Tööhüpotees

Tsüklite eemaldamine peaks olema võimalik kasutades graafist sügavutiläbimisel saadud aluspuud. Graafis ei ole ühtegi tsüklit siis, kui ei leidu aluspuusse mittekuuluvaid servi, mille lähtetipp on puus sügavamal kui sihttipp. Kirjeldatud teisendusi saab teha mööda puuservi kuni mitteranged tipud on puus tagasiulatuva serva lähtetipuks. Viimasena saaks rakendada teisenduse tagasiulatuvale servale ja seejärel eemaldada serva tsüklist.

3 Lahendus

Uuritava lahenduse üldine idee on vaadelda graafi aluspuud ning eemaldada suunatud tsükleid, mis tekivad aluspuusse kuuluvatest servadest ning ühest graafi servast, mis puusse ei kuulu. Iga puusse mitte kuuluv serv ei pruugi tekitada suunatud tsükleid, seega tasub graafi servad jagada klassidesse nii, et oleks selge, millised tekitavad tsükleid. Tsüklit tekitava graafi serva eemaldamisel kaoks sellisel juhul kindlasti ka vähemalt üks tsükel. Ideaalis peaks sellise lahenduse korral eemaldama ainult nii palju tsükleid, kui algses graafis on tsükleid tekitavaid servi. Tegelikult võib teatud tingimustel neid servi juurde tekkida. Lisaks on vaja pärast iga sooritatud teisendust transformatsiooni tulemusel saadud graaf teisendada aluspuuks nii, et vaadeldava puu struktuur säiliks võimalikult suures ulatuses. Lisas 1 on lahenduse kood failis *GrbTrRemoveMuxStrictCycles.ml*.

3.1 Tsüklite leidmine

Tsüklite leidmiseks saab vaadelda graafi sügavutiläbimisel saadud aluspuud. Graafi servad, mis puusse ei kuulu, võib jagada kolme klassi: tagasiulatuvad (*backward*) servad, edasi suunatud (*forward*) servad ja ristervad (*cross*) [12]. Tagasiulatuvad servad viivad puu tipust mõne selle tipu eellase juurde ehk puus kõrgemale, ristervad viivad mõnda puu kõrvalharu ning edasi suunatud servad viivad mõnda tipu alampuusse. Graafi servade klassid ning sügavutiläbimispuu saadakse järgmise algoritmiga kasutades sügavutiotsingut ning eelisjärjekorras nummerdamist.

1. Kõik sisendservadeta tipud on vaadeldavate tippude nimekirjas.
2. Kui vaadeldavate tippude nimekiri on tühi, siis tagasta saadud puu ja lisainfo, muidu jätkata.
3. Vaadata nimekirja esimest vaadeldavat tippu x .
Kui tipul x on läbimata väljundservi, valida neist esimene ja jätkata punktist 4.
Kui tipul x pole läbimata väljundservi, märkida tipp x läbituks ja kustutada nimekirjast, jätkata punktist 2.
4. Tipust x algava serva, mille lõpptipp on y läbimisel:
Kui y on nummerdamata, siis lisada see serv puusse ja nummerdada y . Lisada y vaadeldavate tippude nimekirja algusesse.
Kui y on suurema järjekorranumbriga kui x , siis on see edasi suunatud serv.
Kui y on väiksema või võrdse järjekorranumbriga kui x ja y pole veel läbitud, siis on see tagasiulatuva serv.
Kui y on väiksema või võrdse järjekorranumbriga kui x ja y on läbitud, siis on see risterv.
Jätkata punktist 2.

Vaadeldakse ainult puuservadest ja ühest tagasiulatuvast servast koosnevaid tsükleid. Eesmärk on jõuda vastavasse tsüklist kuuluva tagasiulatuva serva graafist eemaldamiseni. Kui järjest tsükleid valides õnnestub eemaldada kõik algselt leitud tagasiulatuvad servad, pole graafis enam tsükleid. Puu sügavutiläbimisega tegeleb meetod *depth* ja tema sisemine meetod *work*, millest esimene vastutab kõigi tippude rekursiivse läbimise eest ning teine otsustab, mida teha ühe läbitava serva ja selle sihttipuga. Et puu struktuur edasistes teisendustes võimalikult üheselt säiliks, jäädvustatakse iga tipu juurde ka see, millises järjekorras tema

väljudservi esmakordselt läbiti. Lisaks fikseeritakse sisendtipptide läbimise järjekord, mis kokku tähendavad, et uuesti samal puul sügaviläbimist välja kutsudes on tulemuseks sama struktuuriga puu. Ühtlasi fikseeritakse uus järglaste järjekord iga teisenduse juures lähtdes ideest, et järglased, mis vastavad tekkivatele puuservadele, on esimesed ja teiste servadega tekkivad järglased lisatakse järglaste järjekorra lõppu.

Tegelikkuses moodustub eraldi puu iga algse graafi sisendtipu kohta ning puude vahel on teistsugust liiki servad. Seega oleks korrektne saadud struktuuri nimetada metsaks, kuid antud töö piires kasutatakse siiski terminit puu. Sellest võib mõelda ka nii, et tsükkel puu servadest ning ühest tagasiulatuvast servast kuulub korraga ikkagi ainult ühte puusse.

3.2 Tsükli ja teisenduse valimine

Pärast graafist kõigi tsüklite leidmist valitakse üks tsükkel, millel hakatakse teisendusi sooritama. Kui leidub ainult rangetest tippudest tsükkel, vaadatakse kõigepealt seda, sest selle saab eemaldada ühe teisendusega, mis ei muuda puu struktuuri. Kui ranget tsükli ei leidu, kasutatakse üldist valimise algoritmi. Töötlemiseks sobivas tsükli ei ole ühelgi tipul peale kõige ülemise sisendservi, mis oleksid tagasiulatuvat tüüpi. Kõige ülemine tsükli tipp on kõigi tsükliks kuuluvate tippude vanem ning temasse tuleb sisse tsükli moodustav tagasiulatuv serv. Lisaks võib ülemisel tipul olla teisi tagasiulatuvaid sisendeid. Kõige ülemise tsükli tipu alampuudes ei ole teiste tsüklite ülemisi tippe. Alati saab valida sellise tsükli, sest igas puu harus saab tagasiulatuvate servade sihttippudest valida alumise ning vaadelda temasse suubuva tagasiulatuva serva moodustatud tsükli. Kui nende kriteeriumite järgi on palju sobivaid tsükleid, eelistatakse kõigepealt ühe mitterange tipuga tsükleid ning seejärel tsükleid, milles ei ole *MUX* tippu. Ühe tsükliga töötatakse seni kuni on eemaldatud teda moodustanud serv. Seejärel valitakse uus alumine tsükkel. Tsükli valimisega tegeleb meetod *find_next_cycle*, mille argumentideks on kõigi tsüklite järjend ning vaadeldav puu.

Pärast tsükli valimist korrastatakse puud niimoodi, et tsükli tipud oleksid ülemise tipu esimesena läbitud (vasakus) alampuus. Vastavalt asetatakse iga tsükli tipu esimeseks järglaseks talle vaadeldavas tsükli järgnev tipp. Alluvate järjekorra muutmise tegeleb lähtekoodis *rearrange* meetod. Alluvate järjekord on oluline, et pärast teisenduse sooritamist puud korrastades hinnata, millised servad eksisteerivad ja kuidas nad muutuda võivad. Alluvate ümberjärjestamine ei tekita juurde tagasiulatuvaid servi, mis siseneksid tsükli tippudesse, seega on tsükkel ka pärast uude kohta tõstmist teisendamiseks sobilik. Tsükli tippudest algavad alampuud võivad selles protsessis suurened, kui mõnel alampuul oli risterv mõnesse haru, mis oli enne tsüklist vasakul. Nendest vasakul olevatest harudest ei leidu serva tagasi tsükliks, sest muidu oleks see osa tsüklist läbitud mõnda vasakut haru läbides. Algses puus nende tsükli alampuusse tulevate servade puudumine välistab alampuu korrastamisel uute tagasiulatuvate tsüklite tekkimise.

Tsükli töötlemise sammul kontrollitakse kõigepealt, kas tsükliks on mitterangeid tippe. Kui neid ei ole, eemaldatakse teisendusega “*Cut edge with strict edges from target to source*” tsükli moodustav tagasiulatuv serv. Kui leidub üks mitterange tipp ja see on tsükli kõige ülemine tipp, siis eemaldatakse sama teisendusega temasse tulev tagasiulatuv serv. Kui tsükliks on mitu mitteranget tippu või paikneb ainus mitterange tipp tsükli kuskil madalamal, alustatakse teisendamist kõrgeimast ülemise alla jäävast mitterangest tipust. Kui valitud tipp on *Or*, siis sooritatakse teisendus “*Move Over Or*”. Kui see teisendus vajab lisaks ka “*Pass Over an Id-node*” teisendust, kutsutakse see teisendus välja osana puu korrastamise protsessist.

Kui valitud tipp on *MUX*, siis kontrollitakse teisendusele antava serva sihttippu. Kui see on samuti *MUX*, sooritatakse teisendus “*Fuse two Ands / Ors / LongOrs / MUXes*” ja

ühendatakse need *MUX* tipud tsüklil. Kui sihttipp pole *MUX*, siis sooritatakse teisendus “*Move Over MUX*” vastavalt siis kas tõeväärtustüüpi või bitijada tüüpi väljundiga sihttipu jaoks. Teisenduste valik toimub funktsioonis *choose_move*, mille argumendid on vaadeldav tsükel ja vaadeldav puu ning tulemuseks on paar uuest puust ja uuest tsüklist.

Kirjeldatud tsüklis teisenduse valimise ja tsükliga töötamise halvimaks juhuks on olukord, kus tsüklis on üks mitterange tipp, mis on tsükli ülemise tipu alluv. Serva kustutamise teisendus “*Cut edge with strict edges from target to source*” lubaks sellisel juhul kustutada temasse tuleva sisendserva, kuid see ei säilitaks puustruktuuri ning seetõttu liigutatakse tippu mööda tsükli kuni ta muutub ülemiseks tipuks ning tema sisendserva saab kustutada. Ühe teisenduse asemel on teisenduste arv võrdeline tsükklisse kuuluvate tippude arvuga.

Pärast valitud teisenduse sooritamist kutsutakse välja funktsioon puu korrastamiseks, mis ühtlasi ka fikseerib muutused vaadeldavas tsüklis. Kui uus vaadeldav tsükel on tühi, valitakse uus alumine tsükel, vastasel juhul jätkatakse tulemuseks saadud tsükliga. Üldjuhul on vaadeldav tsükel tühi siis, kui teisendust tehti üle tagasiulatuva serva, sest see on iga tsükli viimane teisendus.

3.3 Tippude liigutamine üle *Or* tipu

Teisendus “*Move Over Or*” töötab erinevalt sõltuvalt sellest, kas üle *Or* tipu liigutatakse teine *Or* tipp, *And* tipp, *LongOr* tipp või mõni ülejäänud tippudest. Kui argumendiks saadud serva sihttipp on *Or*, kutsutakse välja eelnevalt defineeritud tippude ühendamise teisendus “*Fuse two Ands / Ors / LongOrs / MUXes*”.

Kui sihttipp on *And* või *LongOr*, on teisenduse idee võrdlemisi sarnane *MUX* tipu liigutamisele üle bitijada väljundiga tipu. Seda sarnasust on üritatud ka implementatsioonis rõhutada, sest see võimaldab neil teisendustel kasutada sama puu korrastamise meetodit. Sarnaselt “*Move Over MUX*” teisendusele jääb ülemise *Or* tipu identifikaator (id) samaks ning juurde tekib uus *Or* tipp ja üks *And* või *LongOr* tipp iga algse *Or* tipu sisendi kohta. Kustutatakse algne *And* või *LongOr* tipp. Igal uuel *And* tipul on kõik algsed sisendid ning üks algse *Or* tipu sisend ja üks väljundserv uude *Or* tippu. Igal *LongOr* tipul on vastavalt üks *Or* tipu sisend, sest *LongOr* tipul on ainult üks sisend. Lisaks on *LongOr* tipul samuti väljund uude *Or* tippu. Uuel *Or* tipul on kõik algse *And* või *LongOr* tipu väljundid ning kõik tema sisendid tulevad mõnest uuest tipust. Teisendus vastab lisas 5 esitatud joonisele, kui *OP* tipu asemel on *And* või *LongOr* ja *MUX* tipu asemel *Or*.

Dimensioonid jäävad *And* ja *LongOr* tippudel samaks, mis algsel *And* või *LongOr* tipul, kusjuures ei muutu ka *LongOr* tipu poolt kokkutõmmatavad dimensioonid. Uue *Or* tipu sisemised dimensioonid on samad, mis *And* või *LongOr* tipul, seega on väljundservadel samad dimensioone muutvad funktsioonid. Olgu algselt ülemise *Or* tipu sisendserval dimensioone muutev funktsioon *a*, *Or* tipust *LongOr* või *And* tippu mineval serval funktsioon *b* ning *And* või *LongOr* tipu väljundserval funktsioon *c*, siis pärast on *LongOr* või *And* sisendserval funktsioonide *a* ja *b* kompositsioon, *LongOr* või *And* tipust *Or* tippu minev serv dimensioone ei muuda ning uue *Or* tipu väljundserval on algne funktsioon *c*.

Teisendusel üle mõne muu graafi tipu *OP*, mille jaoks *Or* tipust tulev serv on kontrollsisend, luuakse uus *Id* tipp. Sõltuvalt sellest, kas *OP* tipp koondab dimensioone või mitte, töötab teisendus veidi erinevalt. Kui *OP* tipp ei koonda dimensioone, siis on *Id* tipul kõik *OP* tipu väljundid ning sisendiks on algsed *OP* ja *Or*. Serv *Id* tippu jääb *OP* tipu ainsaks väljundiks, *Or* tipule jäävad alles kõik algsed väljundid ning tekib üks uus. Algsete tippude sisendid jäävad samaks. *Id* tipu sisemised dimensioonid on samad, mis *OP* tipul. Kõigi *Id* tipu väljundservade funktsioonid jäävad samaks, mis vastaval serval olid *OP* tipu väljundina.

Or tipust *Id* tippu mineva serva funktsioon on sama, mis *Or* tipust *OP* tippu mineval serval, ning *Id* sisendserv *OP* tipust dimensioone ei muuda.

Kui *OP* tipp koondab dimensioone, on teisendus võrdlemisi eelneva sarnane, kuid *Id* ja *Or* tipu vahele tekib *LongOr* tipp. *LongOr* tipu ainus sisend tuleb *Or* tipust ning tal on ainult üks väljund, mis läheb *Id* tippu. *LongOr* tipp koondab samu dimensioone, mida koondab *OP* tipp, ning tema enda dimensioonid on *OP* tipuga samad. Serv *Or* tipust *LongOr* tippu on sama funktsiooniga, kui serv *Or* tipust *OP* tippu ja serv *LongOr* tipust *Id* tippu dimensioone ei muuda. Lisad 7 ja 8 kujutavad teisenduse tööd, kui sihttipuks on *OP* tipp, teisenduse üldise töö seisukohalt ei ole märgitud servatüüpidel tähendust.

Selleks, et teisendus *OP* tipul täidaks oma eesmärgi, on edasi vaja välja kutsuda “*Pass Over an Id-node*” transformatsioon, et huvipakkuv tipp satuks *Or* tipu otseseks järglaseks. *Id* tipust lähtuval servale teisendust rakendades muutub *Or* tipu otseseks järglaseks uus *And* tipp, millest saab *Or* tippu üle liigutada. *And* tipu järglaseks on edasi kas “*Pass Over an Id-node*” sihttipp või sihttipu järglased. Seega üle *Id* tipu liigutamisega kombineeritult õnnestub algne *OP* tipp liigutada tsüklis välja nii, et pärast *Or* tippu saab tsüklis liikuda *And* tipu kaudu mõnesse *OP* tipu järglasesse.

Võimalik, et tulevikus oleks kasulik implementeerida teisendusest ka versioon, mis tunneb ära, et on vaja rakendada “*Pass Over an Id-node*” teisendust. Seega saaks kasutajalt kohe küsida lisisisendit, selle kohta, millist tippu soovitakse *Or* tipu edasiseks järglaseks ning teisendus vastaks paremini oma nimele. Tsüklite eemaldamise protsessis pole see aga vajalik, sest tsüklis on fikseeritud ka järgmine tipp ning “*Pass Over an Id-node*” teisenduse saab eraldi algoritmi seest välja kutsuda.

3.4 Puu korrastamine pärast teisendust

Puu korrastamine pärast teisendust on protsess, mis võrdleb juba moodustatud puud ning transformatsiooni tulemusel saadud graafi ja teostab ka puus selle transformatsiooni. Puu servade liigid fikseeritakse selle järgi, kust sügavutiläbimise algoritm vastava tipu leiab. Teist liiki servade liigid tulenevad sellest, kuidas nende siht- ja väljundtipp puus paiknevad. Ühtlasi muudetakse teisendusele vastavalt ka tsüklit, millega tööd jätkatakse.

Käesoleva peatüki eesmärk on kirjeldada, mis oleks kõige loomulikum viis graafi puuks muutmiseks ning kuidas seda programmis rakendada. Ilmneb, et mõningatel juhtudel ei sobi üldine lähenemine, kus eemaldatakse alati tagasiulatuva serv. Ühtlasi on näha, mis tingimustel puusse tekib uusi tagasiulatuvaid servi, mis moodustavad uusi tsükleid. Joonistel on üldiselt igat tüüpi sisendi või väljundi jaoks üks näide, tegelikkuses võib neid tipul olla rohkem, kuid mõttekäik nendega toimimiseks jääb samaks. Samalaadselt ei pruugi tipul olla servi kõikidest tüüpidest.

3.4.1 Serva eemaldamine graafist

Puust eemaldatakse ainult tagasiulatuvaid servi. Lisa 2 kujutab kahte võimalikku olukorda serva eemaldamiseks. Serva võib eemaldada ainult rangetest tippudest koosnevast tsüklis või ühe mitterange tipuga tsüklis, kus mitterange tipp on kõige kõrgem tsükli tippudest. Tagasi ulatuva serva eemaldamine ei muuda puud rohkem, kui selle serva kustutamine. Tekib uus sisenditipp *Error*, mis tähistab sisendit, mida tsüklis ei ole võimalik saada. Kui puustruktuuri võimalikult suures ulatuses säilitada, siis jääb ülemisele tipule alles sisend, mis kuulub puusse ning *Error* tipu läbimise ajaks on ülemine tipp juba lisatud. Seega peaks serv *Error* tipust ülemisse tippu olema ristserv, sest ta suundub juba läbitud alampuusse. *Error* moodustab ühetipulise puu.

Serva eemaldamine on alati ühe tsükliga töötamise lõpetamine. Uut tsükli ei moodustata, vaid uus tsükkel valitakse järgmisel sammul teisendamiseks sobiva tsükli leidmise algoritmi järgi. Lähtekoodis tegeleb kirjeldatud puu korrastamisega funktsioon *rearrange_treeCut*.

3.4.2 Tippude ühendamine

Tippude ühendamine võib tsükli toimuda kahes erinevas kohas: üle puuserva või üle tagasiulatava serva. Teisendust üle puuserva kujutab lisa 3, joonisel on kujutatud *Or* tippude ühendamine, *MUX* tippude korral toimuks see samamoodi. Edasine kirjeldus on *MUX* tippude ühendamisele üheselt ülekantav. Pärast teisenduse toimumist on alumisel *Or* tipul lisaks enda sisenditele ka kõik ülemise tipu *Or1* sisendid. Et alumine tipp jääks puus võimalikult samasse kohta, peaks puuserv temasse tulema samast tipust, kust tuleb *Or1* tipu sisendserv. *Or* tipp ja temast algav alampuu liiguvad teisenduse tulemusel küll puus kõrgemale, kuid kõik tema eellased peale *Or1* tipu jäävad samaks. Tipp 2 on joonisel *Or1* tipu ja tipu 1 eellane, seega jääb ta ka *Or* tipu eellaseks ja vastav serv on edasi minev. Tipp 3 paikneb tipu 1 või mõne kõrgema tipu kõrvalharus, seega on serv tipust 3 *Or* tippu risterv. *Or* tipu endised sisendid ja väljundid jäävad sama tüüpi. Tipp 0 võib olla sama, mis tipp 1, sellisel juhul on olulisem säilitada infot puuserva kohta. Kui tipp 1 kuulub tsükliksse ja ei ole tsükli kõrgeim tipp, siis ta ei saa olla sama, mis tipp 11 ja tipp 11 on kuskil tipu 1 eellaste hulgas. Mõlemal juhul jääb serv tippu 11 tagasiulatavaks servaks. Tipp 11 ei saa olla sama, mis *Or1*, sest ainus tipp tsükliks, millel on tagasiulatavaid sisendeid, on tsükli ülemine tipp, aga kunagi ei valita teisenduseks ülemisest tipust algavat serva. Et kogu tsükkel on alates ülemisest tipust oma alampuu vasakpoolseim (esimesena läbitud) haru, kuulub tipp 13 mõnda tsüklist kõrgemal hargnevasse vasakusse alampuusse ja temasse minev serv on endiselt risterv.

Lisa 4 kujutab teisendust üle tagasiulatava serva. Kui seda tehakse, on see tsükli viimane teisendus, seega on kahe märgitud *Or* tipu vahel puus ainult ranged tipud. *Or* tippudest algavad alampuud teisenduses ei muutu ja nad jäävad puus samasse kohta, seega nende endised väljundid ja sisendid on kõik endist tüüpi. Ülemise *Or* tipu uued sisendservad tippudest 9 ja 10 on tagasiulatavad servad, sest 9 ja 10 on *7Or* tipu alluvad. Need tagasiulatavad servad võib ka kohe eemaldada, sest nad kuuluvad tsükliksse, milles on üks mitterange tipp, mis on tagasiulatava serva sihttipuks. Tipp 1 paikneb kuskil tipu *7Or* kohal, seega on tipust 1 algav uus serv edasi minev. Tipust 11 tuleva *7Or* sisendserva liik sõltub tipu 11 asukohast. Kui puus leidub suunatud ahel tipust *7Or* tippu 11, siis on tipp 11 tipu *7Or* alluv ning temast tulev serv on tagasiulatav. Selliselt tekkinud tagasiulatuvatest servadest tsükleid tuleb eraldi vaadata, sest tippe 11 ja *7Or* ühendaval suunatud ahelal võib olla ka teisi mitterangeid tippe. Kui tipp 11 ei ole tipu *7Or* alluv, siis paikenb ta mõnes kõrgemal hargnevas paremas alampuus ja *7Or* sisendserv tipust 11 on risterv.

Teisendusel üle puu serva on järgmine tsükkel sarnane algsega, ainult ta ei läbi enam ülemist *Or1* tippu vaid läheb tipust 1 otse tippu *Or*. Üle tagasiulatava serva teisenduse korral lõpetatakse tsükliga töötamine. Need tekkinud tagasiulatavad servad, mida võib eemaldada, kustutatakse ning uut tsükli ei fikseerita. Lähtekoodis on vastav puu korrastamine funktsioonis *rearrange_treeFuse*.

3.4.3 MUX tipu liigutamine

MUX tipu liigutamise teisendusest on kaks erinevat versiooni sõltuvalt sellest, kas sihttipp on tõeväärtustüüpi või bitijada tüüpi väljundiga. Graafi ja puud muudavad nad üsna sarnaselt, seetõttu pole neist eraldi jooniseid. Tõeväärtusväljundiga tipu jaoks on toodud liigutamine

üle puuserva ning bitijada väljundiga tipu jaoks üle tagasiulatava serva. Teisenduste erinevuseks on see, kas uus tipp on *MUX* või *Or*, aga servade analüüs on puuduvatel joonistel analoogiline olemasoleva joonisega, mis vastab sama serva teisendusele. Lisas 1 on olemas ka puuduvad joonised.

Lisa 5 kujutab *MUX* tipu liigutamist üle tõeväärtustüüpi väljundiga tipu. Iga *MUX* tipu sisendi kohta tekib üks uus tipp, mis on sama tüüpi kui tipp 1. Uuel tipul on kõik tipu 1 sisendid peale *MUX* tipust tuleva ja igal uuel tipul on üks sisend *MUX* tipu sisendtipust. Uute tipule 1 vastavate tippude järglaseks saab *Or* tipp, mis kujutab asukohta vahetanud *MUX* tippu. *Or* tipule jäävad kõik tipu 1 endised väljundid ja sealt algab samasugune alampuu, seega on kõik väljundid endist liiki. Uus puu servade ahel algab tipust 2, mis on *MUX* tipu vanem puus ning läbib sellele tipule vastava tipu 1 koopia, kust see liigub *Or* tippu ning jätkab tsüklis fikseeritud teed alampuu. Vastavasse tippu 1 tulevad teised sisendid jäävad samaks, sest tema asukoht puus ei muutu eriti võrreldes algse tipu 1 asukohaga.

Tippude puhul, mille sisend tuleb tippudest 3 või 4, otsitakse kõige madalama lähtetipuga endine edasi minev serv. Kui sellel tipul 1 leidub sisendservi, mille endine tüüp on ristserv, siis kontrollitakse, kas mõne ristserva lähtetipp on valitud madalaima edasi mineva lähtetipu alluvuses. Kui neid ei ole, saab madalaima lähtetipuga edasi minev serv puuservaks ja teised jäävad endiselt edasi minevateks servadeks, kõik sisendid, mis on ristservad, jäävad ristserveks. Kui neid on, saab puuservaks kõige madalamalt hargneva kõrvalharu kõige ülemine ristserv. Kõik teised servad jäävad endist liiki. Kui edasi minevaid sisendeid ei ole, saab samuti puuservaks kõige madalamalt hargneva kõrvalharu kõige kõrgemalt algav ristserv. Kõrvalharude hargnemiskohaks loetakse tipp tipu 2 kohal olevas puuharus, kust saab alguse haru, milles on vaadeldav serva lähtetipp. Võib juhtuda, et kõik ristservad tulevad mõnest teisest graafi sisendtipust algavast puust ja madalaimat ühendust ei saa leida. Sellisel juhul tuleb puuservaks muuta serv tipust, mis graafis esimesena leiti ning ülejäänud jääksid ristserveks. Ka eelnevalt kirjeldatud puuserva fikseerimine lähtub ristserve korral sellest, milline ristserva lähtetipp kõige esimesena leitakse.

Lisa 6 kujutab *MUX* tipu liigutamist üle bitijada väljundiga tipu, kui valitud serv on tagasiulatav. Joonisel kujutatud olukorras saab kindel olla selles, et tipul 7 on sisendid *MUX* tipust ja tipust 2 ning *MUX* tipul tipust 10. Seega, et kujutatud alampuu jääks võimalikult samasse kohta, peaks uue *MUX* tipu puuserv tulema uuest tipust 7, mille sisendiks on tipp 10 ning tema puuserv tuleb nagu ennegi tipust 2. Sellega on paigas, kus uued tipud puus asuvad.

Tipust 9 sisendit saav tipp 7 võiks paikneda tipu 9 alluvuses. Algoritmi järgi võiks vastav tipp 7 olla tipu 9 esimesena leitav alluv, mis küll rikub reeglit, et tsükel on kõige vasakpoolseim alampuu, kuid et tegemist on käesoleva tsükli viimase teisendusega, siis ei tekita see probleemi. Vastav tipp 7 paikneb siis uue *MUX* tipu alluvuses ning serv temast *MUX* tipu on tagasiulatavat tüüpi. Selle tagasiulatava serva võib eemaldada, sest teisenduse ajal on tsüklis algse *MUX* tipu kohal vaid ranged tipud. Vastava tipu 7 algsed sisendid, mis on edasi minevad või ristservad, jäävad samaks. Puuservast saab edasi minev serv, sest uus tipu 7 koopia on tipu 2 kaugem alluv. Kui tipp 5 paikneb ahelal tipust *OP* tipuni 9, siis on ta edasi minev serv, muidu ristserv.

Tipu 7 koopiate puhul, mille sisendid tulevad tippudest 1 ja 5, tuleb kontrollida, kas tipul 7 oli tagasiulatavaid sisendeid peale selle, mis tuleb *MUX* tipust. Kui neid on, tuleb valida tagasiulatavate servade lähtetippudest vasakpoolne kõrgeim, millest algav sisendserv saab puuservaks. Tema mujalt tulevad endised tagasiulatavad servad muutuvad ristserveks. Tipu 7 puuserv sisend muutub edasi minevaks servaks, sest vastav tipu 7 koopia paikneb sügavamal tipu 2 alluvuses. Sisendid tippudest 3 ja 4 jäävad samaks. Sisend tipust 1 jääb

edasi minevaks, sest tipp 1 paikeneb alguses tipu 7 kohal ning tipu 7 vastav koopia on kuskil algse tipu 7 asukoha alluvuses. Tipust 11 tulev sisend jääb ristservaks, sest tipu 7 vastav koopia muutub kuskil üksikuks omaette haruks. Serv tipust 7 tippu *MUX* on tagasiulatuv serv, mida ei saa kohe eemaldada. Kui tipul 7 ei ole tagasiulatuvat sisendit, siis jäävad tipu 7 sisendid koopial sama liiki, mis alguses. Serv tipust 1 tippu 7 on edasi minev, sest tipp 1 oli kuskil tipu 2 kohal. Serv tipust 11 tippu 7 on tagasiulatuv, kui 11 on algses puus tipu 7 alluv, ja ristserv, kui ta seda pole. Kui tekib tagasiulatuv serv, peab seda tsüklit eraldi vaatama.

Kirjeldatud olukord ei vii oluliselt tsüklite eemaldamisele lähemale, sest kui uus *MUX* tipp liiutada alla kohale, kus on tipp 10, siis tuleks uuesti eemaldada samasugust tsüklit. Et *MUX* on tsükli, mille tagasiulatuv serv läheb tipust 10 tippu 7, ainus mitterange tipp, võib eemaldada *MUX* tipu vastavat tipust 7 tuleva sisendserva. See on erakorraline teisendus, mis lõhub ära puustruktuuri, sest kogu alampuu võib liikuda kuskile mujale, kui pole selliseid tipu 7 koopiaid, millest *MUX* tippu minev serv säiliks. Kui säilib mõni *MUX* tipu sisend, siis jääb alampuu endiselt tipu 7 kaudu tipu 1 alluvusse, kui kõik *MUX* tipu sisendservad kustutatakse on vaja puu uuesti moodutada, sest on raske kontrollida, kuhu paiknevad alampuusse kuulunud tipud.

Üle puuserva teisendusel moodustatakse uus tsükel, mis tsüklis olnud *MUX* tipu ja talle järgneva tipu asemel läbib kõigepealt *OP* tipu selle koopia, millest lähtub puuserv ja siis tekkinud *MUX* või *Or* tipu. Lähtekoodis tegeleb *MUX* tipul tehtud teisenduse järel puu korrastamisega funktsioon *rearrange_treeMUX*.

3.4.4 *Or* tipu liigutamine

Teisendus *Or* tipul on sihttippude *And* ja *LongOr* puhul analoogiline *MUX* tipu liigutamisele. Joonised on sarnased lisades 5 ja 6 toodud joonistega, erinevuseks on see, et uue tipuna tekib alati *Or* tipp ja sihttipp on *And* või *LongOr*, kusjuures *LongOr* puhul ei pea vaatama tagasiulatava serva juhtu, sest *LongOr* tipu ainus sisend on alati puuserv. Teisendus, kui sihttipuks on teine *Or* tipp, on samaväärne tippude ühendamise, mida illustreerivad lisad 3 ja 4.

Lisa 7 kujutab *Or* tipu teisendust üle *Op* tipu, kui *Op* tipp dimensioone ei koonda. Sellisel juhul tekib juurde üks *Id* tipp, mis puus ja tsüklis võtab endale *Op* tipu koha. Selleks, et ta kuuluks tsükklisse, peab temasse *Or* tipust tulev serv olema puuserv ning ta peaks olema esimene *Or* tipu alluv. Seega jääb *Op* tipp *Id* tipu suhtes kõrvalharu ning nende vahele tekib ristserv. Lisa 8 kujutab sama teisendust juhul, kui *Op* tipp koondab dimensioone. Tsükklisse tekib *Or* tipu järele ka *LongOr* tipp, ahel *Or* tipust *LongOr* tippu ja edasi *Id* tippu kuulub tsükklisse ja asendab tsüklis olnud puuserva tippu *Op*, seega on vahepealsed servad puuservad. Serv *Op* tipust *Id* tippu jääb eelnevaga samal põhjusel ristservaks.

Lisad 9 ja 10 kujutavad *Or* teisendust üle tagasiulatava serva. Kõige loomulikum puu struktuuri säilitamiseks oleks teha *Id* tipp *Op* tipu järglaseks puus, sest siis saab tema kaudu puus liikuda tippu 8. Kui üritada *Id* tippu sobitada *Or* tipu otseseks järglaseks või alluvaks *LongOr* tipu kaudu, siis asetuksid ka kõik *Op* tipu järglased puus teise kohta ning tekiks küsimus, kust tuleb tsüklit puuga ühendav puuserv. Loomuliku puu ehitamise juures on aga see puudus, et teisendus tekitab juurde tagasiulatava serva ning see tagasiulatuv serv tipust *Or* tippu *Op*, millest tsüklis vabaneda tahetakse, säilib samuti. Võib teha teisenduse, mis kustutab *Or* tipu tsüklis tulevad sisendid, sest *Or* tipp on vaadeldava tsükli ainus mitterange tipp. See vahesamm eemaldab kindlasti joonisel oleva tsükli, kuid muudab ka puu struktuuri, sest *Or* tipp ja tema alluvad paiknevad puus uude kohta.

Kui teisendus toimub üle puuserva, siis moodustatakse uus tsükel, kus *Op* tipu asemel

on *LongOr*, kui ta tekkis, ning *Id* tipp. Järgmisena rakendatakse tsüklis teisendust “*Pass Over an Id-node*”, et tsüklis sobiv *Id* tipu järglane liigutada *Or* tipu järglaseks. Lähtekoodis tegeleb selle korrastamisega *rearrange_treeOr*.

3.4.5 Üle *Id* tipu liigutamine

Üle *Id* tipu liigutamise juhul saab *Id* tipule eelneda kaks võimalikku seisu sõltuvalt sellest, kas tipp, mille kohale *Id* tekkis, koondas dimensioone või mitte. *Id* tipu teisenduse jaoks on need juhud sarnased, oluline on, et tekkiva *And* tipu sisend tuleb *Id* tipule tõeväärtustüüpi sisendit andvast tipust. Lisades toodud joonised kujutavad ainult juhte, kus *Id* tipule eelneb *Or*, sest kui tema asemel on *LongOr*, ei muutu midagi olulist. Lisas 1 on siiski olemas ka eraldi joonised *LongOr* tipu kohta. Ühtlasi on lisas 1 joonised selle kohta, kui *Id* tippu tuleb serv on tagasiulatuvat tüüpi, kuid rakendus kasutab sellel puhul *Or* tipu sisendite kustutamise võimalust, mis on kirjeldatud *Or* tipu liigutamise peatükis.

Lisas 11 on joonis puu muutumisest, kui teisendust rakendatakse servale, mille sihttipp on bitijada tüüpi väljundiga ning millel on tõeväärtustüüpi sisend. Tsükel läbib edaspidi tippu *Or*, *And* ja *Op*, seega tekivad nende vahele puuservad. Alumine *Op* tipp on ülemise suhtes kõrvalharus, seega tekib sinna ristserv. Alumine *Op* tipp jääb puus samale kohale, lihtsalt tema eellane on nüüd *Id* tipu asemel *And*, seega jäävad samaks ka tema säilivad sisendtipud. *And* tipu sisend tipust 5 jääb sama tüüpi, mis ta oli *Op* tipu sisendina, sest *And* tipp on puus *Op* tipuga sarnases kohas, kõik *Op* tipu eellased on ka tema eellased ja tipud, mis on *Op* tipu jaoks kõrvalharus, on seda ka *And* tipu jaoks. *Op* tipust algav alampuu jääb endiseks. *Id* tipu ühe väljundi eemaldamine ei muuda tema sisendeid ega väljundeid.

Lisa 12 kujutab puu muutumist, kui üle *Id* tipu liigutamist rakendatakse servale, mille sihttipp on tõeväärtustüüpi. Tsüklis sihttipu alluvateni jõudmiseks on vaja, et serv *Or* tipust *And* tippu oleks puuserv. Et *Op* tipp jääks tsüklis võimalikult samale kohale, on serv ülemisest *Op* tipust temasse puuserv. Ülemine *Op* tipp saab nüüd *Or* tipu vasakpoolsemaks alluvaks, mistõttu muutuvad *Id* tipu sisendservad. Selline tippude järjestus on oluline, et *Op* tipu sisendid jääksid sama tüüpi, mis enne. Eelkõige on oluline, et *Id* tipu alluvusest tulevad sisendid on endiselt ristservad. *Id* tipu väljudservad selles teisenduses ei muutu, sest nad olid varem ülemise *Op* tipu väljundid ning selline asukoha vahetus puus neid ei mõjuta. *And* tipp on puus enam-vähem samas kohas kui *Op* tipp ning tema väljundite tüüp ei muutu.

Lisa 13 kujutab *Id* teisendust üle tagasiulatuva serva juhul, kui sihttipp on tõeväärtustüüpi väljundiga. Valitud *Id* tipust lähtuva serva sihttipp *Op* paikneb puus kuskil *Or* tipu kohal. Kui üritada säilitada üleval puu struktuuri, peaks *And* tipp olema *Op* tipu alluv. Sellest järeldub, et servad *Or* tipust *And* tippu ja *Op* tipust *Op* tippu on mõlemad tagasiulatuvad servad. Joonisel kujutatud olukord vastab aga samale tsüklile, mis oli enne *Id* tipu tekkimist, kus oli tagasiulatuv serv *Op* tipust *Op* tippu ja tehti teisendus *Or* tipu liigutamiseks üle esimese *Op* tipu. Seega sellises situatsioonis *Id* teisendus lahenduseni ei vii ning on võimalik kustutada *Or* tipu tsüklis tulevaid sisendeid, sest ta on tsükli ainus mitterange tipp. Juhul kui sihttipp on bitijada tüüpi väljundiga, tekib samasugune probleem, kus algne tsükel ei kao ning seetõttu peaks proovima ebareeglipärasest lahendust.

Tsüklite korrastamisel tuleb üle puuserva teisendusel tsüklis *Id* tipp asendada *And* tipuga, kui tegemist oli teisendusega üle bitijada tüüpi väljundiga tipu. Tõeväärtusväljundiga tipu puhul tuleb tsüklis *Id* ja talle järgnenud *Op* tipp asendada tekkinud *And* tipuga. Teisendusel üle tagasiulatuva serva tsüklit ei fikseerita. Puu korrastamisega üle *Id* tipu liigutamisel tegeleb funktsioon *rearrange_treeId*.

3.4.6 Üle *Id* tipu liigutamine ei tööta

“*Pass Over an Id-node*” teisendus ei ole defineeritud, kui valitud serva sihttipuks on *MUX* tipp. Kui tsüklik teisendusi tehes selline olukord tekib, kus transformatsioon ei tööta, siis on kuskil tsüklik järjest tipud *Or*, *Id* ja *MUX* (või *Or*, *LongOr*, *Id*, *MUX*, kui enne *MUX* ja *Or* tipu vahel olnud tipp muutis dimensioone). Kui kuskil tsüklik on veel mõni mitterange tipp, saab seda liigutada ning puu uuesti moodustada. Selle tulemusena võib juhtuda, et leitakse uus tsükkel, millega osatakse teha reeglipäraseid teisendusi või et lõpuks muutub tsükkel sellisel viisil, et seal teisi mitterangeid tippe ei ole.

Kui tsüklik ei ole teisi mitterangeid tippe peale *MUX* ja *Or* tipu, mille vahel on *Id* tipp, võib teatud tingimustel kustutada *Or* tipu tsüklist tuleva sisendi. Nimelt, kui *Or* ja *Id* tipul ei ole väljundservi mujale kui *MUX* tippu. *Or* tipul on selles situatsioonis vähemalt kaks väljundit *Id* tippu ja *Op* tippu, millele teisendust rakendades *Id* tipp tekkis. Samas, *Op* tipul on ainult üks väljund, mis läheb *Id* tippu. Lisaks võib *Or* ja *Id* tipul olla teisi väljundeid. Erik Jaaniso bakalaureusetöö [3] kirjeldab kasutajapoolse lissisendiga teisendust, mis de-kombineerib tippe. *Id* ja *Or* tipule võib siinkohal rakendada sarnast teisendust.

Lisa 14 kujutab sellisel hetkel graafis tehtavaid muudatusi. Kõigepealt peaks *Or* tipu asemele tekitama kaks tippu, millest ühel on väljundid *Id* ja *Op* tippu ning teisel teistesse *Or* tipu väljunditesse. *Id* tipu võib jagada kaheks nii, et ühel on ainult väljundserv *MUX* tippu ja teisel ülejäänud väljundid. Kusjuures teise *Id* tipu võib tõsta selle *Or* tipu järglaseks, millel on *Or* tipu ülejäänud väljundid, sest need *Or* tipud on teineteisega samaväärsed. *OP* tipust jääb kaks samaväärset eksemplari, mis erinevad ainult selle poolest, kust tuleb tõeväärtustüüpi sisend. Nüüd on graafis fragment, kus on kahe väljundservaga *Or* tipp, mille järglastel *Op* ja *Id* tippudel on kummalgi üks väljund. Fragmendi ainus väljundserv on tippu *MUX*. Vaadeldav tsükkel läbib seda graafifragmenti ning kui nüüd eemaldada *Or* tipu tsüklist tulev sisendserv, katkeb ka tsükkel sellisel kujul. Tunnetuslikult on selle serva kustutamine korrektne, sest hetkeks, mil tsüklist tuleb *Or* tippu tõene tõeväärtus, on *MUX* tipp oma väärtuse juba arvutanud, sest muidu poleks seda saanud arvutada ükski tsükli tipp. Seega, kui *MUX* tipp peaks andma välja vaadeldavast *Id* tipust tuleva väärtuse, peaks *Or* tipp saama tõese väärtuse mõnest muust sisendist.

Joonisel ei ole kujutatud servade liike, üldiselt jäävad dubleeritud servad sama liiki, mis enne. Tsüklist *Or* tippu tulnud serv oli puuserv, seega liigub see *Or* tipp, mille sisend kustutati mõne teise sisendi järglaseks. Sõltuvalt *OP* tipu teistest sisendservadest võivad puud uuesti läbides muutuda ka selle *Or* tipu alampuusse jäävate servade liigid.

3.5 Lahenduse analüüs

Puude korrastamisel selgus, et erinevatel teisendustel üle tagasiulatuva serva võib graaf niimoodi muutuda, et tekib juurde uusi tsükleid. Selliselt tekkivate tsüklite keerukus, näiteks mitterangete tippude arv neis, või üldse tekkivate tsüklite arv, ei ole graafiga tööd alustades hästi hinnatav. Igal sammul on küll tagasiulatuvate servade arvu järgi lihtne hinnata, kui palju tsükleid on hetkel teada, kuid see ei fikseeri, kui paljudel tsüklitel veel teisendusi teha tuleb.

Tõsisemaks probleemiks pakutud lahenduses on aga defineeritud teisendus “*Move Over Or*”. Puude korrastamise juures tuli välja, et lisaks olukorrale, kus “*Pass Over an Id-node*” teisendus ei tööta, on veel mitmeid eriolukordi, kus tuleb kustutada servi, mis ei ole tagasiulatuvat tüüpi. Erinevalt *MUX* tipu jaoks kirjeldatud teisendusest, kus samuti kustutati vajadusel puuservi, ei säili *Or* tipu jaoks loogilist kohta, mille alluvusse ta saaks jääda. Seega ei luba selle teisenduse olemus pidada kinni eesmärgist hoida puu struktuuri teisenduste

jooksul võimalikult ühesena. Siiski peaksid defineeritud erijuhud viima mõne serva ja sellega ka mõnel kindlal kujul tsükli eemaldamiseni.

Tegelikult illustreervad lisas 15 toodud joonised, et defineeritud teisendus *Or* tipu üle bitijada tüüpi tipu viimiseks koos kirjeldatud erijuhuga, millal “*Pass Over an Id-node*” teisendus ei tööta, ei jõua tegelikult tsükli eemaldamiseni. Joonis kujutab olukorda, kus tsükli ülemiseks mitterangeks tipuks on *Or* tipp, millele järgnevad *OP* tipud, mille väljundid on bitijadad, ja lõpuks *MUX* tipp. Joonis on korrektne, kui tsükli ei ole teisi mitterangeid tippe peale kujutatute, sest lõpuks peab olema võimalik kasutada teisendust üle *Id* tipu liigutamise erijuhuks. Nummerdatud joonised kujutavad järjest tsükli tekkivaid situatsioone vastavalt sellele, milliseid teisendusi algoritm valib, kusjuures kasutatud teisenduse nimi on toodud iga alamjoonisega. Joonistel toimuv vastab üksikute teisenduste joonistele, kuid kujutatud ei ole kõiki servi, näiteks on 3. jaotises tekkival *And* tipul alati veel üks sisend. Joonise viimasel sammul tehakse lisas 14 kujutatud erijuhu teisendus, mille tulemusel kaob ühe *Or* tipu sisendserv ja lõigatakse läbi tsükkel, mis läbi parempoolse *And* tipu jõudis *MUX* tipu. Teoreetiliselt peaks sellega olema eemaldatud tsükkel, millel teisendamist alustati, sest vaadeldavast tsüklist teisendustega saadud tsüklist on kustutatud serv. Joonisel on näha, et säilivad servad algsest *Or* tipust *OP1* tippu ja edasi *OP2* tippu ning sealt *Id* tipu kaudu *MUX* tippu ja seega säilib algse sarnane ühe *Id* tipu võrra pikenenud tsükkel. Järgmiseks tsükliks graafis leitakse tõenäoliselt see säilinud tsükkel. Analoogiline olukord tekib ka siis, kui valitud tsükliks jääb *Or* ja *MUX* tipu vahele rohkem bitijada tüüpi väljundiga *OP* tippe. “*Move Over Or*” teisenduses tekkiv teede duleerimine (*Or* tipust saab *Id* tippu otse ja läbi *OP* tipu) võib ka teistes situatsioonides põhjustada seda, et tsükli ei õnnestu eemaldada.

Erinevate analüsaatoris defineeritud näitegraafide korral ilmneb, et kirjeldatud olukord esineb küllalt sageli. Implementeeritud lahenduse logide järgi selgub, et kui valida tsükleid nii, et eelisjärjekorras vaadatakse alumistest tsüklitest neid, millel on ainult üks mitterange tipp ja seejärel neid, milles *MUX* tippe ei ole, õnnestub üldiselt esimesena leitud tsüklid eemaldada. Esimeste tsüklite eemaldamisele on iseloomulik ka see, et tsükliga töötamise lõpus tagasiulatuvate servade arv väheneb, seega ei tekkinud uusi tagasiulatuvaid servi. Keerulisemate tsüklite puhul, milles on mõlemat tüüpi mitterangeid tippe, satub algoritm aga kergesti olukorda, kus tsükli päriselt ei eemaldata.

Lahendust võib vaadata ka ilma järjestikuste *Or*, *Id* ja *MUX* tippude kohta käiva erijuhuta. Sellisel juhul liigutatakse probleemsesse olukorda sattudes tsükli *MUX* tippu edasi sarnaselt sellele, kuidas käitutakse, kui satutakse olukorda, kus *Id* tipu teisendus ei tööta, kuid tsükli on veel teisigi mitterangeid tippe. Ühel hetkel peaks sellisel juhul *MUX* tipp liikuma üle tõeväärtustüüpi tipu ja tema asemel tekiks tsükli *Or* tipp, lõpuks saaks tegeleda tsükliga, kus on ainult *Or* tipud ja probleemseid situatsioone ei teki. Selline *MUX* tipu liigutamine on vastuolus ideega, et teisendus üle tagasiulatuva serva on tsükli viimane, sest on väga võimalik, et *MUX* tipp peab liikuma tsükli *Or* tipu kohale enne kui ta ise *Or* tipuga asendub. Ühtlasi tähendaks see seda, et tuleb liigutada tsükli ülemist tippu ja kui teisendatava serva lähtetipul on tagasiulatuvaid sisendeid, on uute servade tüüpide fikseerimine keeruline. Antud implementatsioonis saab ka seda versiooni testida, kui kommenteerida funktsioonis *rearrange_treeId* välja märgitud koodilõik, kuid tulemuseks saadud puu struktuur ei ole täpselt fikseeritud.

Üldiselt, kui erakorraliseks teisenduseks valitud *MUX* tipp jõuab tsükli kohta, kust tsükli töötlemise alguses mitterange tipp allapoole liikuma hakkas (või kuskile madalamale), on tema alampuudes suure tõenäosusega sama tüüpi tippudest ahelad. Analüüsides üle *MUX* tipu liigutamise teisendust (või ka analoogilisi teisendusi *Or* tipu jaoks, kui sihttipp on *And* või *LongOr*), on selge, et kui näiteks lisas 5 tekkinud *Or* tipule rakendada üle tõeväärtus-

tüüpi tipu viimise teisendust, on pärast seda teisendust tekkinud uue *Or* tipu igaks sisendiks ahel, kus on üks lisas 5 tekkinud *Op* tipp ja üks eelmises teisenduses tekkinud tõeväärtustüüpi tipp. Need ahelad ei koosne kõik kunagi puu servadest, kuid graafis eksisteerivad nad ikkagi. Kui *MUX* tippu mööda ühte ahelat liigutada alla *Or* tipuni ja siis saadud mitterange tipu sisendserv kustutada, jäävad allest ahelate alguse kohale jäänud *MUX* tipu ja ülejäänud ahelate poolt moodustatud tsüklid ning lõpuks tuleb *MUX* tippu liigutada alla mööda kõiki neid ahelaid. Seega ei anna ka erijuhuta versioon häid tulemusi.

“*Move Over Or*“ teisendusest tekkinud probleemid ei võimalda anda ühest hinnangut sellele, kas hüpotees, et sügavutiläbimispuud saab tsüklite eemaldamiseks kasutada, peab paika. Hetkel ei ole põhjust selles meetodis kahelda, kuid edasiseks uurimiseks tuleb *Or* tipu jaoks loodud teisendus asendada uuega.

3.6 Edasised ülesanded

Peamisi probleeme põhjustas üle *Or* tipu liigutamise teisendus *OP* tipu puhul. See teisendus oleks võimalik defineerida sarnaselt *MUX* tipu üle tõeväärtustüüpi tipu liigutamise teisendusele. Tekkivaks mitterangeks tipuks peaks olema uut tüüpi *MUX* tipu laadne tipp, mis annab välja esimese sisendväärtuse, kuid ei anna järgnevate sisendite korral viga, kui need on võrdsed esimese sisendiga. Saades teistsuguse sisendi peaks ta lõpetama veaga nagu *MUX* tipp, kui ta saab rohkem kui ühe korrektse sisendi. Puu korrastamiseks oleks võimalik kasutada *MUX* tipuga sama teisendust, kuid eraldi vajaks läbimõtlemist, kuidas uus tipp graafis edasi käitub.

Võimalik oleks uurida ka juhtu, kus igas tsüklis tehakse esmalt teisendusi *MUX* tippudel ning *Or* tippe vaadeldakse alles siis, kui tsüklis *MUX* tippe ei ole. Sellisel juhul tuleks paika panna, kuidas puu muutub, kui teisendatakse tsükli ülemist tippu või loobuda pidevalt võimalikult ühte moodi fikseeritud aluspuu ideest.

“*Move Over MUX*“ teisenduse asemel võiks tagasiulatuva serva korral dubleerida alumist *MUX* tippu. Ühele tipule jäävad kõik sisendid ja väljundid peale tagasiulatuva serva ja teisele tagasiulatuvale servale vastav väljund ja kõik sisendid peale tsüklilist tuleva puu-serva. Sellise teisendusega ei tohiks tagasiulatuvate servade arv tsüklis kasvada, kuid algse tagasiulatuva serva asemele võib tekkida üks uus.

Kokkuvõte

Käesolev töö käsitles ühte võimalust Peeter Laua ja Ilja Tšahhirovi loodud krüptograafiliste protokollide analüsaatori protokolliesitusest tsüklite eemaldamiseks. Protokolliesituseks kasutatakse analüsaatoris sõltuvusgraafe, millesse transformatsioonide tulemusel võivad tekkida tsüklid. Analüsaatoris on olemas teisendused, mis võimaldavad eemaldada tsükleid, kus on maksimaalselt üks mitterange tipp. Lahendada oli vaja rohkemaid mitterangeid tippe sisaldavate tsüklite eemaldamise ülesanne.

Lahenduse üldidee seisnes graafi sügavuti läbimisel saadud aluspuu fikseerimises koos infoga nende servade kohta, mis aluspuusse ei kuulu. Vastavalt sellele peaks puusse kuuluvatel servadel tegema transformatsioone kuni õnnestub eemaldada kõik tsükleid moodustavad tagasiulatuva servad. Suurem osa teisendusi, mida nendest servadest vabanemiseks vaja läheb, oli analüsaatoris juba olemas, nii et eelkõige oli küsimus nende kasutamise järjekorras.

Esmajärjekorras eemaldati tsüklid, milles ei olnud mitterangeid tippe. Edasi analüüsi ti tsükleid, mille alampuudes ei olnud teisi tsükleid. Tsükli piires valiti teisendus nii, et mitteranged tipud liiguksid tsükli järjest madalamale ja viimaseks jääks teisendus üle tagasiulatuva serva. Üldine idee oli selles, et tsükklisse jääb üks mitterange tipp, mis on tsükli tippudest puus kõige kõrgemal ning olemasoleva teisendusega saab kustutada temasse suubuva tagasiulatuva serva ja seega ka katkestada tsükli. Puu struktuuri muutumist analüüsides ilmnes, et kohati tuleks siiski kustutada ka teisi servi.

Puuduva teisenduse implementatsioonis kasutati loogilisi samaväärsusi ja ühte olemasolevat teisendust, et liigutada mõnda *Or* tipu alluvuses olevat tippu *Or* tipu vahetuks järglaseks. Bitijada tüüpi väljundiga tippude puhul tekitas transformatsioon aga dubleeritud ahela *Or* tipust tipuni, mis ta järglaseks muutus, ning tsüklite eemaldamise kontekstis ei osutunud selline teisendus kasulikuks. Teisenduse tõttu tekkis olukord, kus ühe serva eemaldamine küll eemaldas ühe tsükli, aga selle tsükli teisendamisel oli säilinud paralleelne tee, mis säilitas algsega analoogilise tsükli.

Kokkuvõttes ei õnnestunud tsüklite eemaldamise ülesannet lahendada, kuid siiski ei saa väita, et algne lahendusidee oleks olnud ekslik. Negatiivne tulemus tuleneb teadaolevalt *Or* tipu jaoks defineeritud teisendusest ja mitte üldisest algoritmist tsüklite eemaldamiseks. Tulevikus peaks jätkama tööd sama algoritmiga, kuid uue teisendusega *Or* tipu jaoks.

Removing Cycles from the Protocol Representation of the Cryptographic Protocol Analyser

Bachelor thesis
Pille Pullonen

Abstract

The aim of this paper was to describe a cryptographic protocol analyser, specially its protocol representation, and implement a transformation to simplify protocol representation. The analyser was first described in the doctoral thesis of Ilja Tšahhirov [1] and was afterwards implemented by Peeter Laud and Ilja Tšahhirov. The analyser is designed for static protocol analysis to check confidentiality and integrity of secret messages. All cryptographic primitives used in the protocols have to be strong in computational model in order for the analysis to be correct.

Protocols are represented by dependency flow graphs that indicate possible data and control flows in protocol execution. Like any directed graphs, they may contain directed cycles that make understanding and drawing graphs more difficult. In general, all nodes in graph can be divided to two groups - strict and non-strict nodes. Strict nodes only compute their value if they have received all their inputs, on the other hand, non-strict nodes compute their value as soon as they have sufficient inputs to do so. All cycles in graph can be divided to groups depending on which non-strict nodes they contain and how many of them are present. The analyser already has transformations to cut cycles with none or just one non-strict nodes.

This paper described a way based on depth-first search trees to find all cycles and transform them so that they contain only one non-strict node and can therefore be removed by cutting an edge in the cycle. Most transformations needed were already present in the analyser, but transformation to move *Or* nodes (nodes that represent disjunction) over their children was not implemented. This transformation was implemented using nodes and transformations already present in the analyser, so that it would not have any side effects for the analyser.

The proposed algorithm for cycle removing is probably correct, at least there are no known reasons why it should not work, but the transformation for *Or* node should be reconsidered. Current implementation causes problems because other transformations needed as part of this are not universal. Situations where needed transformations are not defined result in different special cases that can be solved locally, but together lead to general cycles that remain connected although some edges are cut out from the graph.

Following research should consider alternative transformation to move some nodes over *Or* node, possibly something similar to transformation *Move Over MUX* that is already implemented. This would lead to adding a new node to general graph description and problems about how this node should function in transformations. Other possible developments include choosing a different order for transformations in a cycle and describing new special versions of some transformations.

Viited

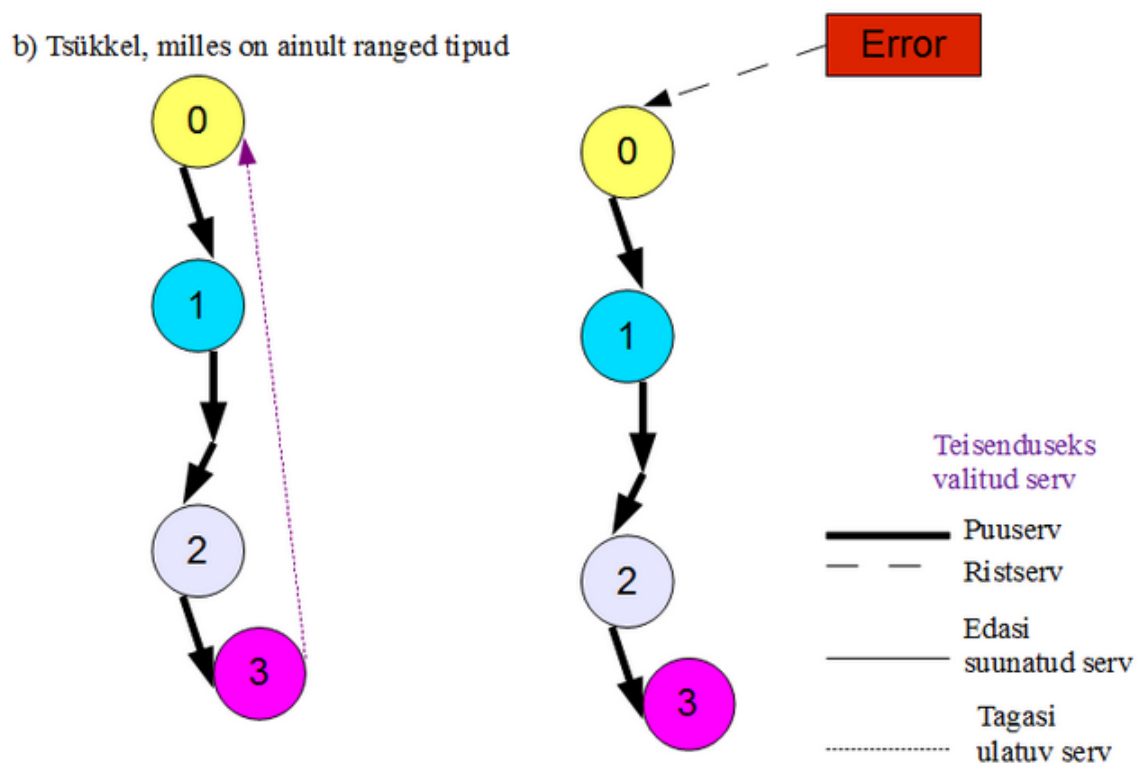
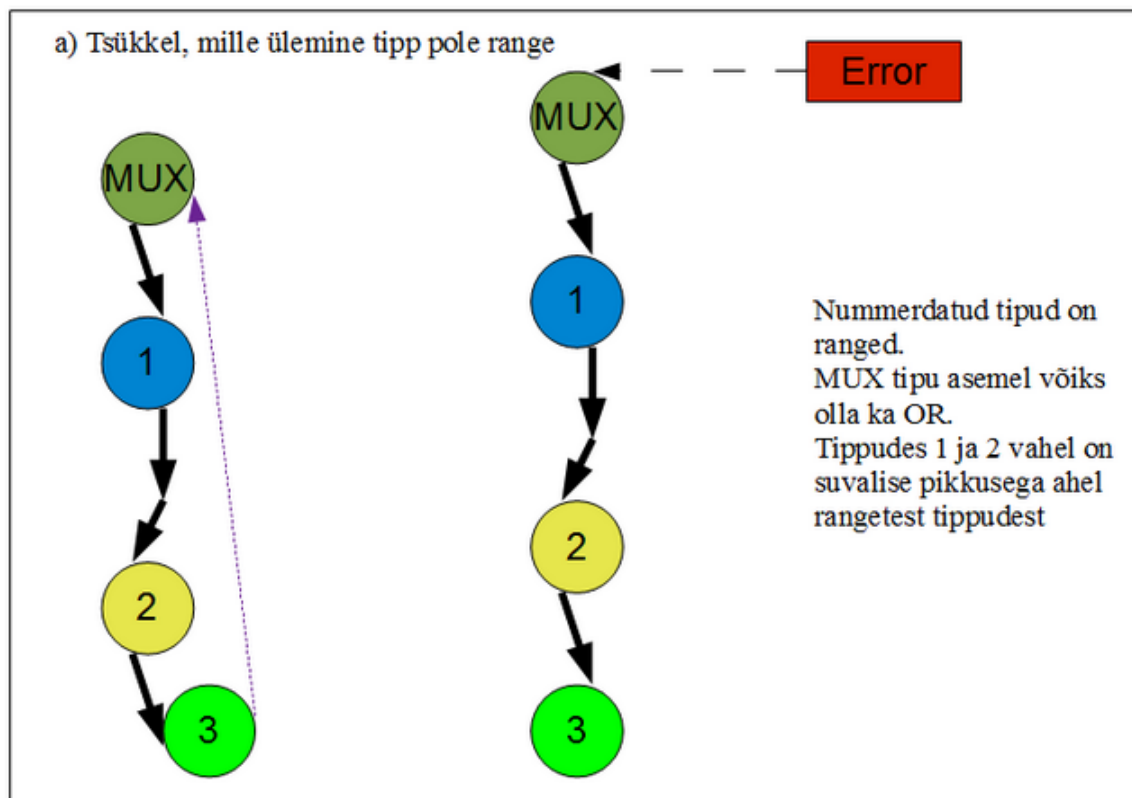
- [1] Tšahhirov, Ilja, *Security Protocols Analysis in the Computational Model - Dependency Flow Graphs-Based Approach*. Tallinna Tehnikaülikool, Doktoritöö, 2008.
- [2] Laud, Peeter; Tšahhirov, Ilja, *A user interface for a game-based protocol verification tool*. In proceedings of the sixth International Workshop on Formal Aspects in Security and Trust (FAST2009), Madalmaad, Springer-Verlag November 2009
- [3] Jaaniso, Erik, “Mängude jadal põhineva krüptograafiliste protokollide analüsaatori täiendamise”. Tartu Ülikool, Bakalaureusetöö, 2009.
- [4] Goldwasser, Shafi; Micali, Silvio, *Probabilistic encryption*. Journal of Computer and System Sciences, Volume 28, Issue 2, Aprill 1984
- [5] Dolev, Danny; Yao, Andrew C. *On the security of public key protocols*. IEEE Transactions on Information Theory, Volume 29, Issue 2, Märts 1983
- [6] Laud, Peeter; Tšahhirov, Ilja, *Application of Dependency Graphs to Security Protocol Analysis*. In the 3rd Symposium on Trustworthy Global Computing (TGC 2007), Prantusmaa, Springer-Verlag 2008
- [7] INRIA, *The Caml language*.
<http://caml.inria.fr/> (30.05.2011)
- [8] Camlcity, *OCaml library manager*.
<http://projects.camlcity.org/projects/findlib.html> (30.05.2011)
- [9] Conchon, Sylvain; Filliâtre, Jean-Christophe; Signoles, Julien, *OCamlgraph*.
<http://ocamlgraph.lri.fr/> (30.05.2011)
- [10] OMake, *Omake build system*.
<http://omake.metaprl.org/index.html> (30.05.2011)
- [11] Universität Bremen, Bernd Krieg-Brückneri uurimisgrupp, *uDraw(Graph)*.
<http://www.informatik.uni-bremen.de/uDrawGraph/en/index.html> (30.05.2011)
- [12] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford, *Introduction to Algorithms*. Peatükk *Elementary graph algorithms*, esmatrükk MIT Press and McGraw-Hill, 1990

Lisad

Lisa 1 - Lähtekoodiga CD

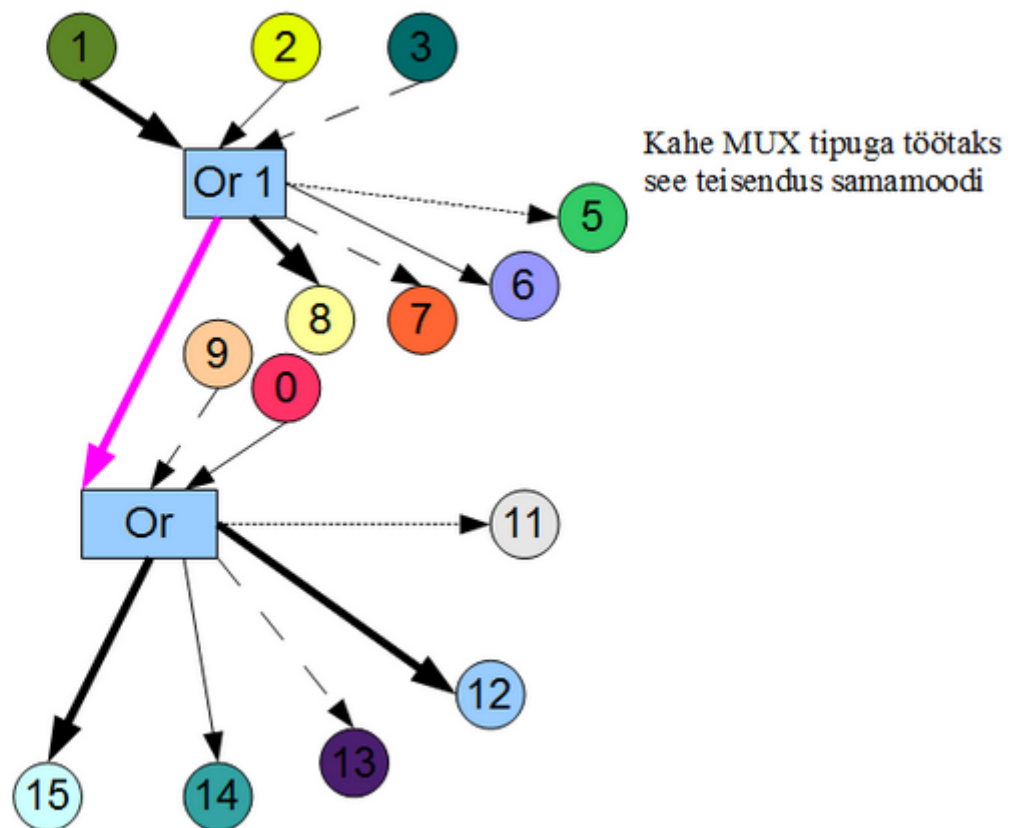
Lisatud plaadil on töös kirjeldatud analüsaatori lähtekood koos programmeeritud uuendustega. Lisaks on seal käesolev töö, uue koodi dokumentatsioon ning lühike abifail, mis loetleb kompileerimiseks ja käivitamiseks vajaliku tarkvara. Olemas on töös puuduvad joonised ning täpsemad juhised kirjeldatud lahenduse testimiseks ja logifaili lugemiseks.

Lisa 2 - Serva eemaldamine graafist

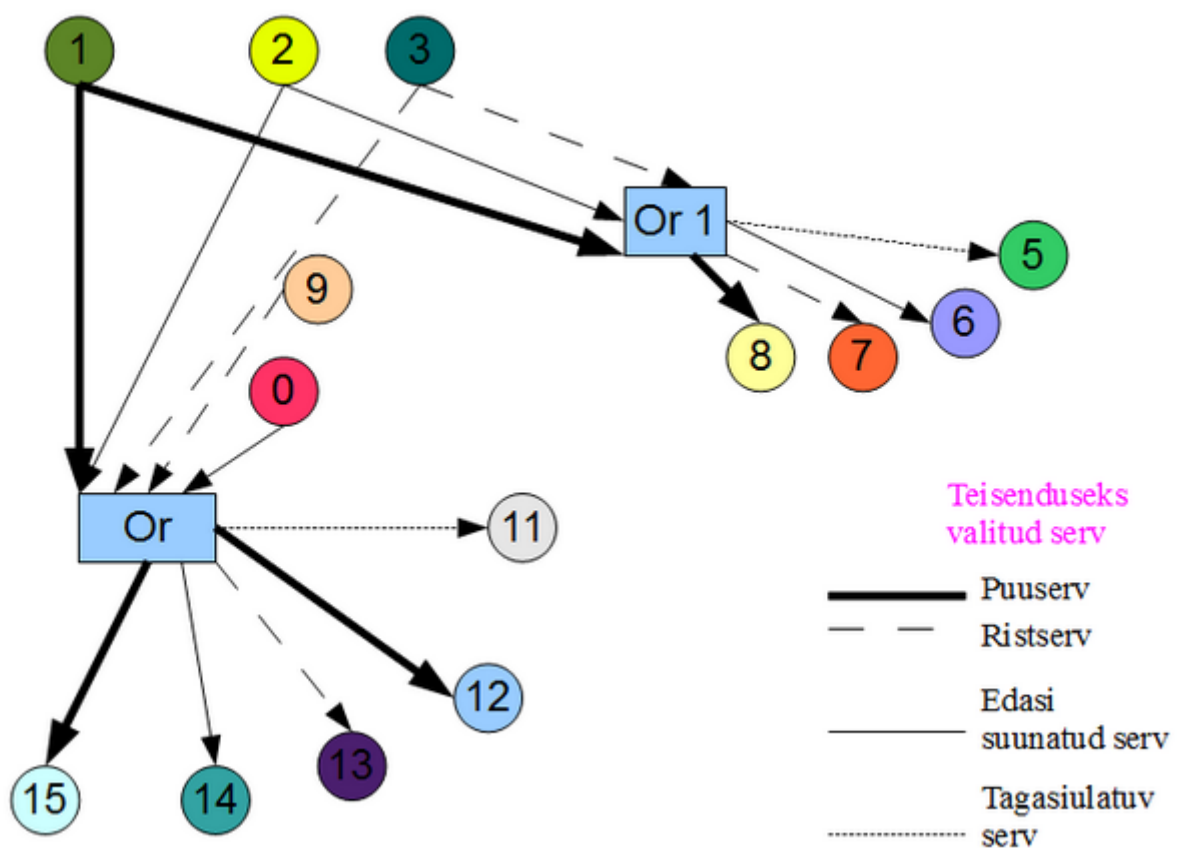


Lisa 3 - Tippude ühendamine üle puuserva

Enne teisendust

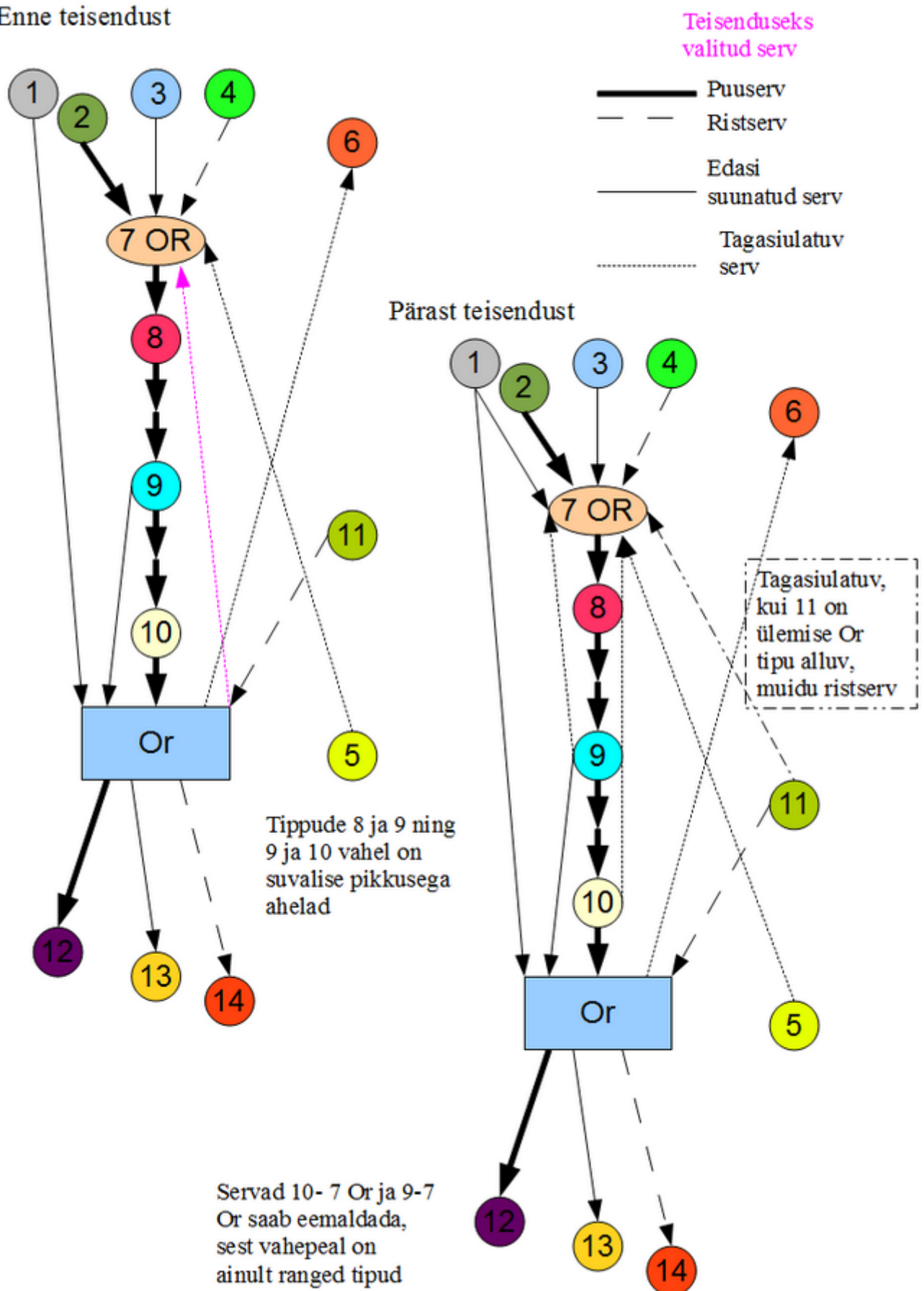


Pärast teisendust



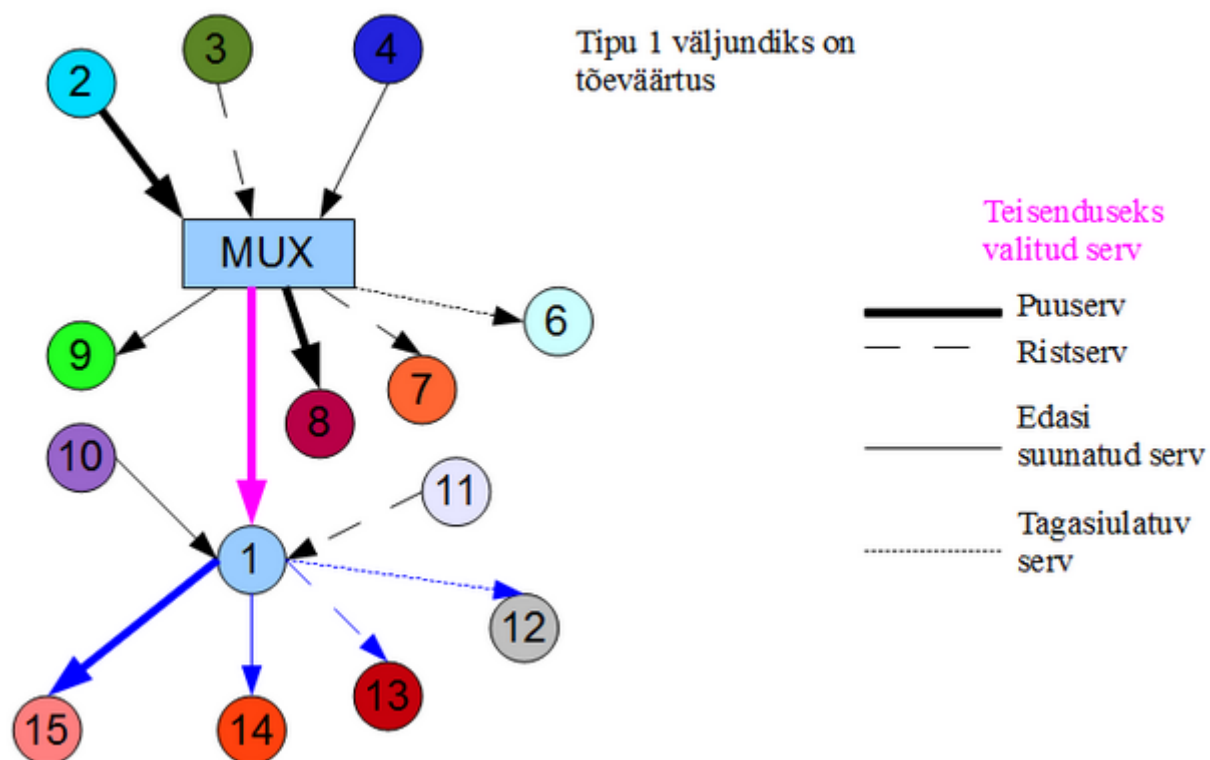
Lisa 4 - Tippude ühendamise üle tagasiulatuva serva

Enne teisendust

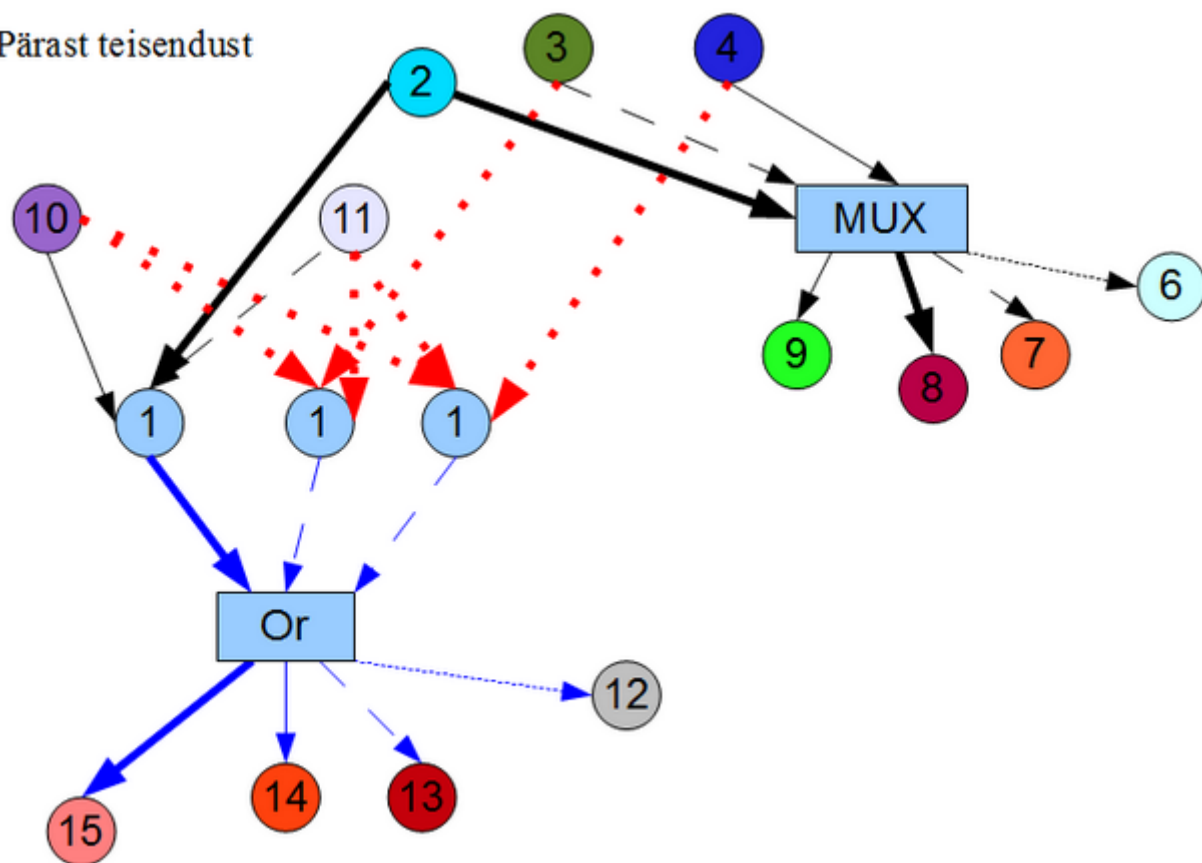


Lisa 5 - MUX tüüpi liigutamine üle puuserva

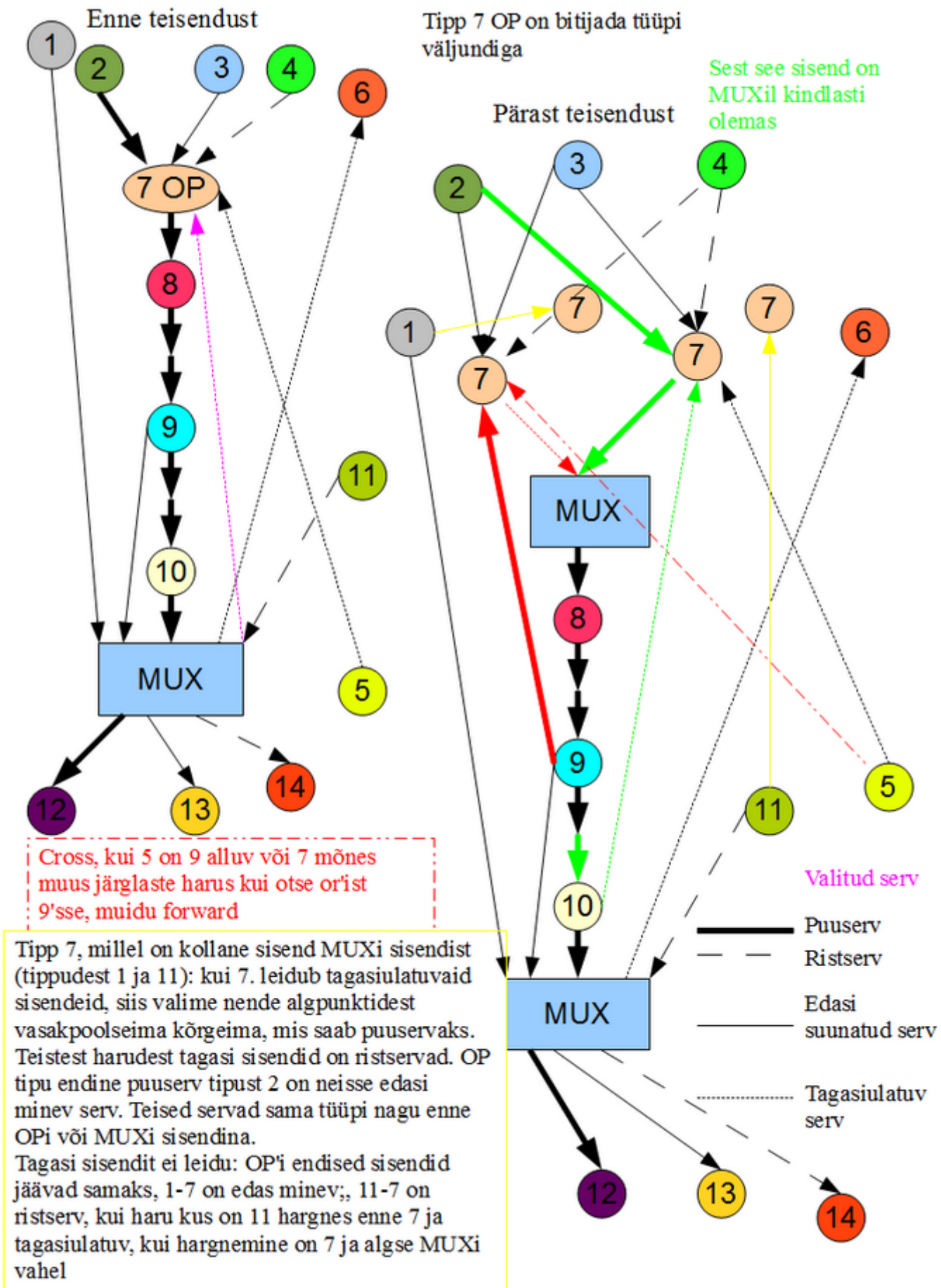
Enne teisendust



Pärast teisendust

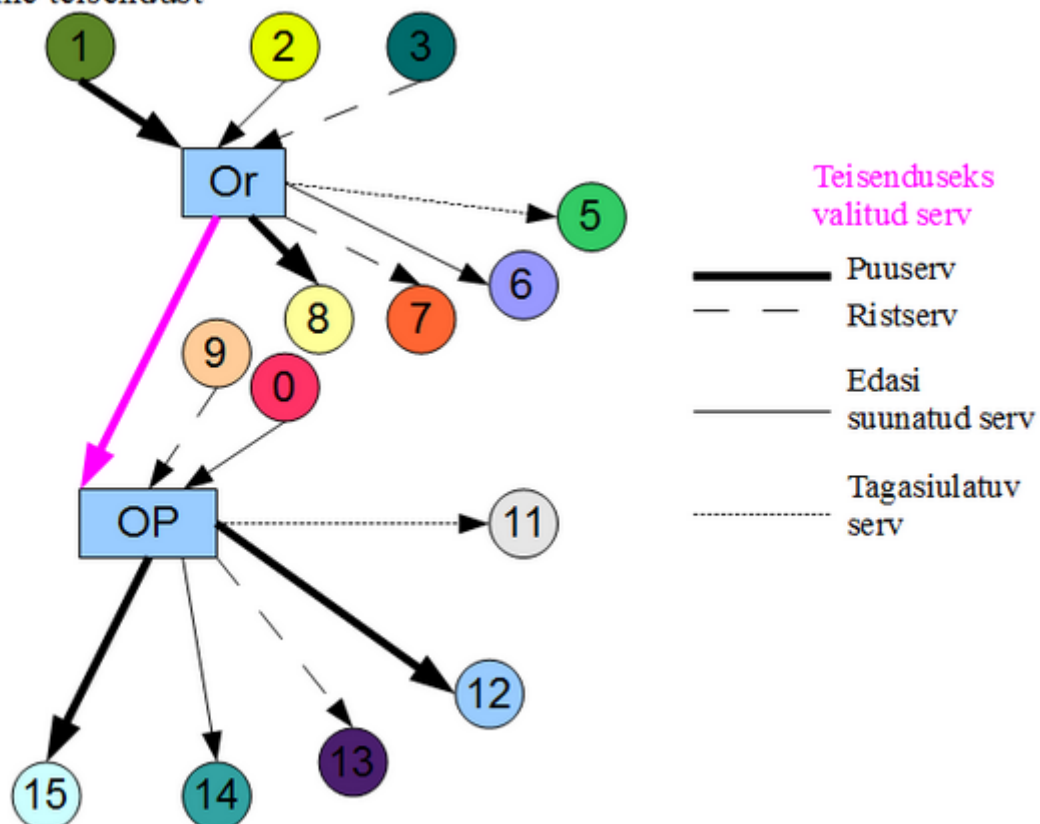


Lisa 6 - MUX tipu liigutamine üle tagasiulatava serva

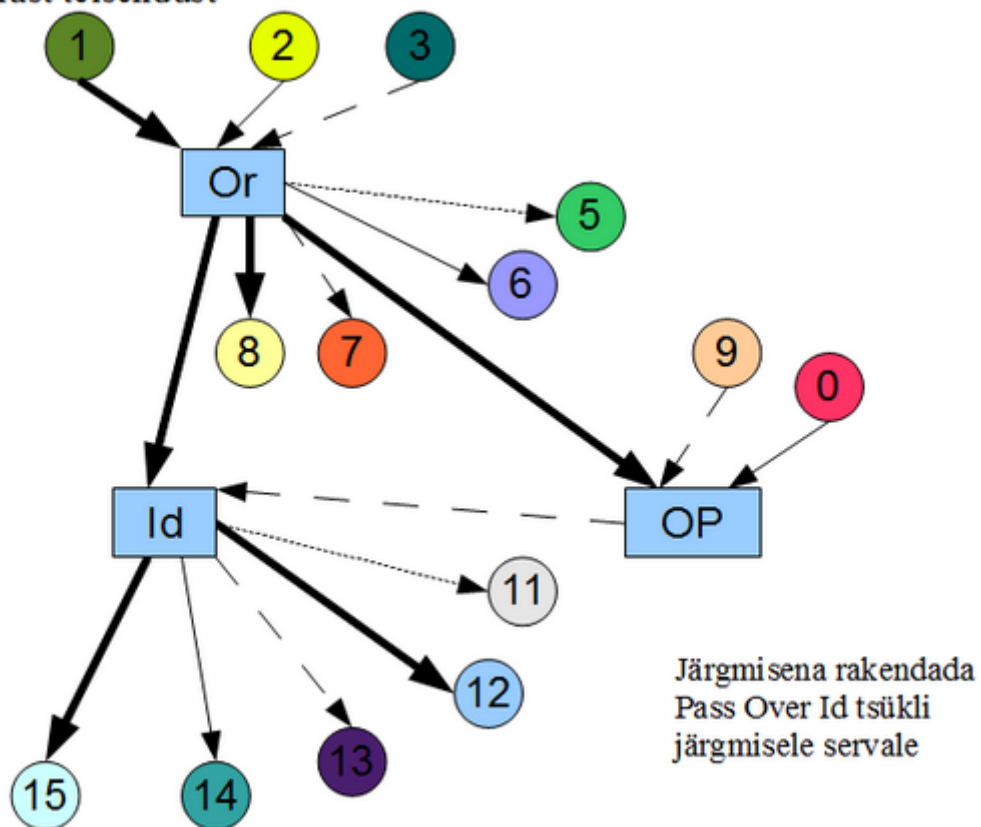


Lisa 7 - Or tipu liigutamine puuserval, kui sihttipp ei koonda dimensioone

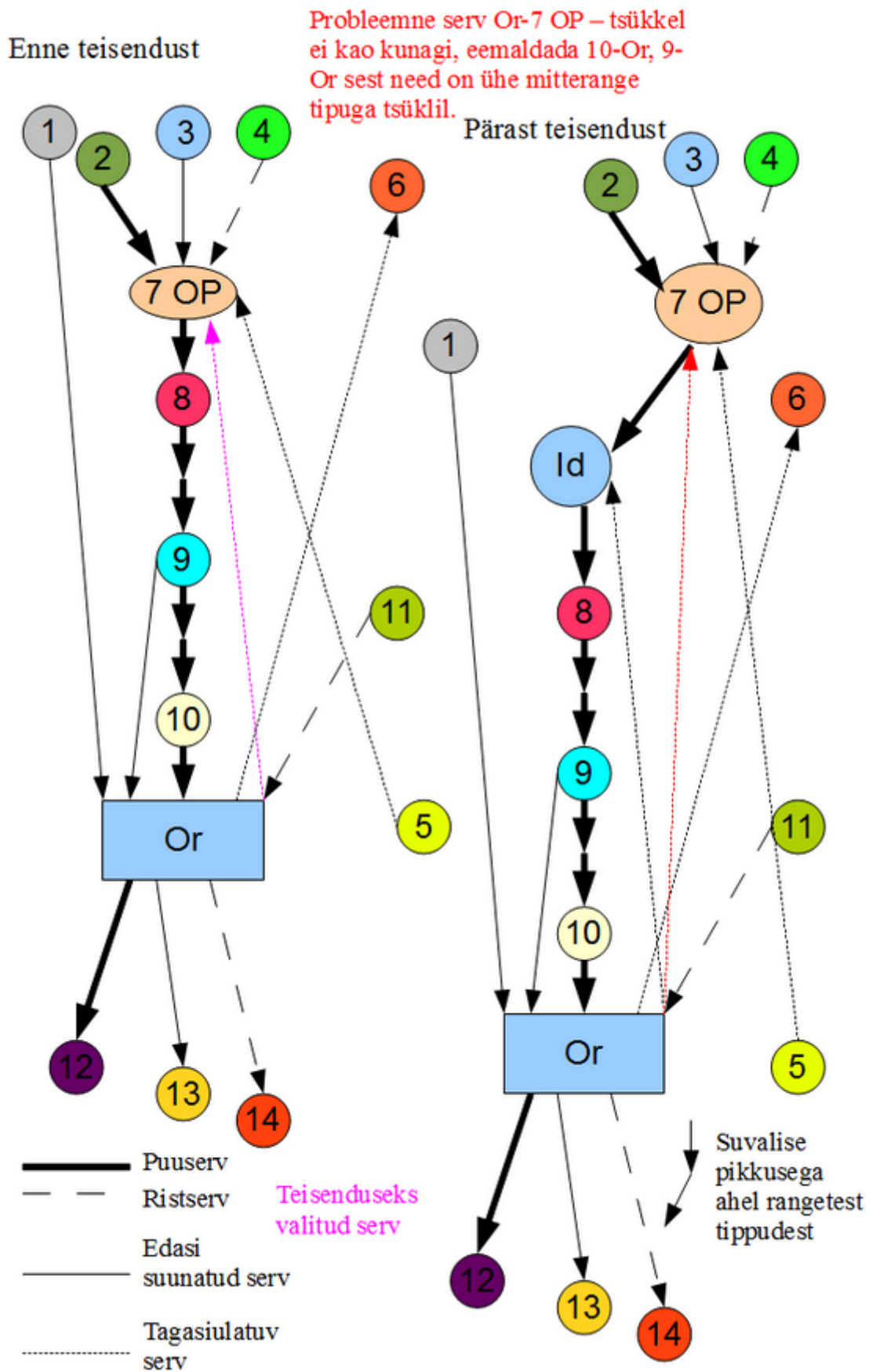
Enne teisendust



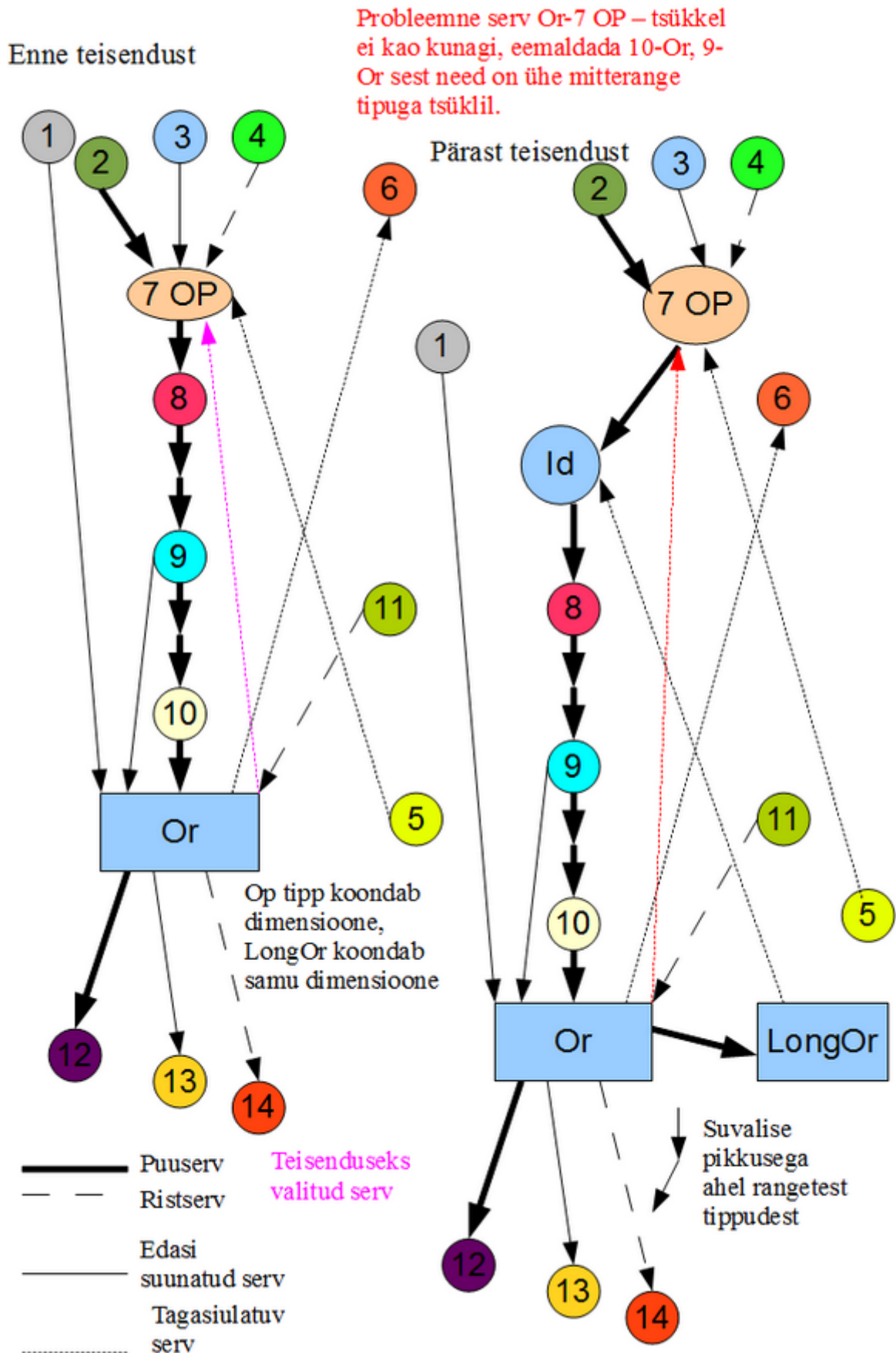
Pärast teisendust



Lisa 9 - Or tipu liigutamine tagasiulatuval serval, kui sihttipp ei koonda dimensioone

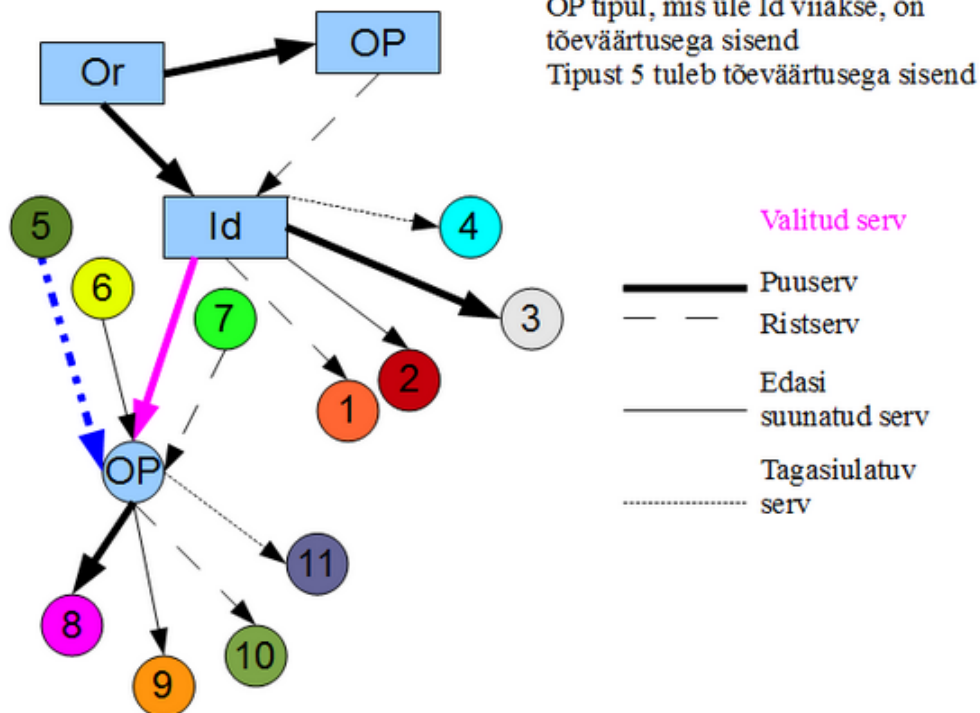


Lisa 10 - Or tipu liigutamine tagasiulatuval serval, kui sihttipp koondab dimensioone

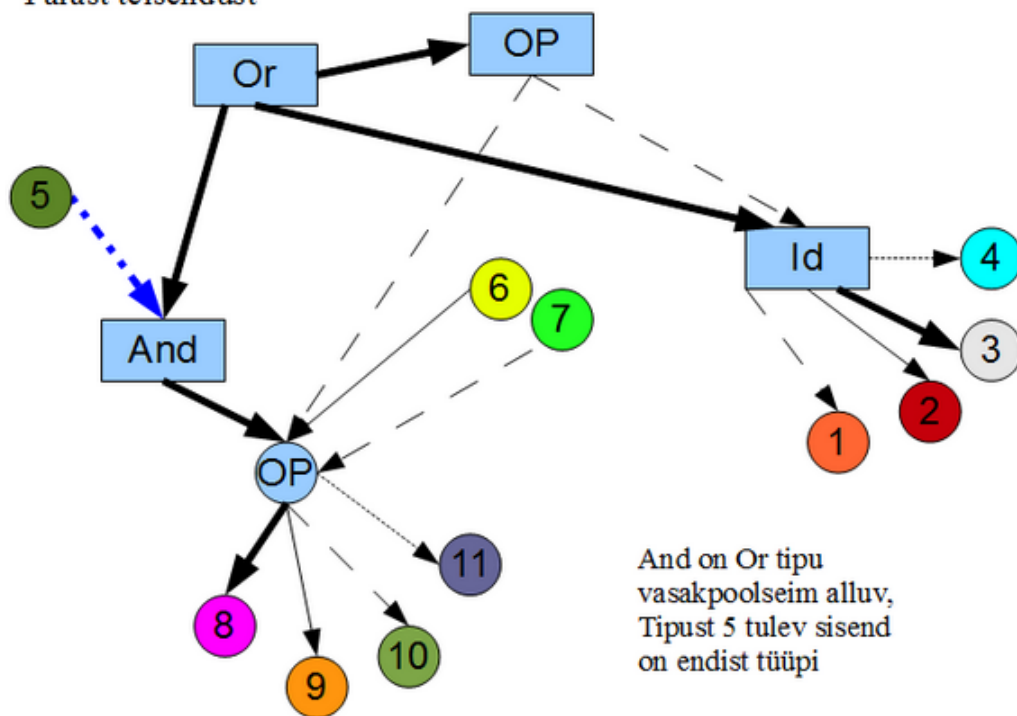


Lisa 11 - Üle Id tipu liigutamine puuserval

Enne teisendust

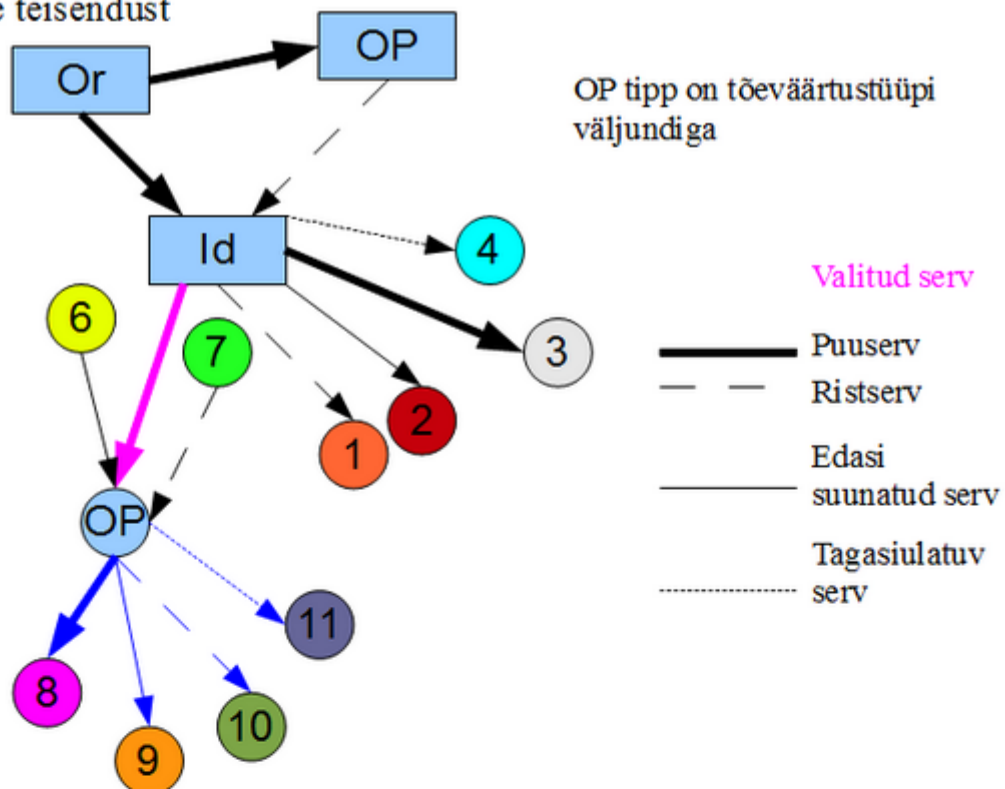


Pärast teisendust

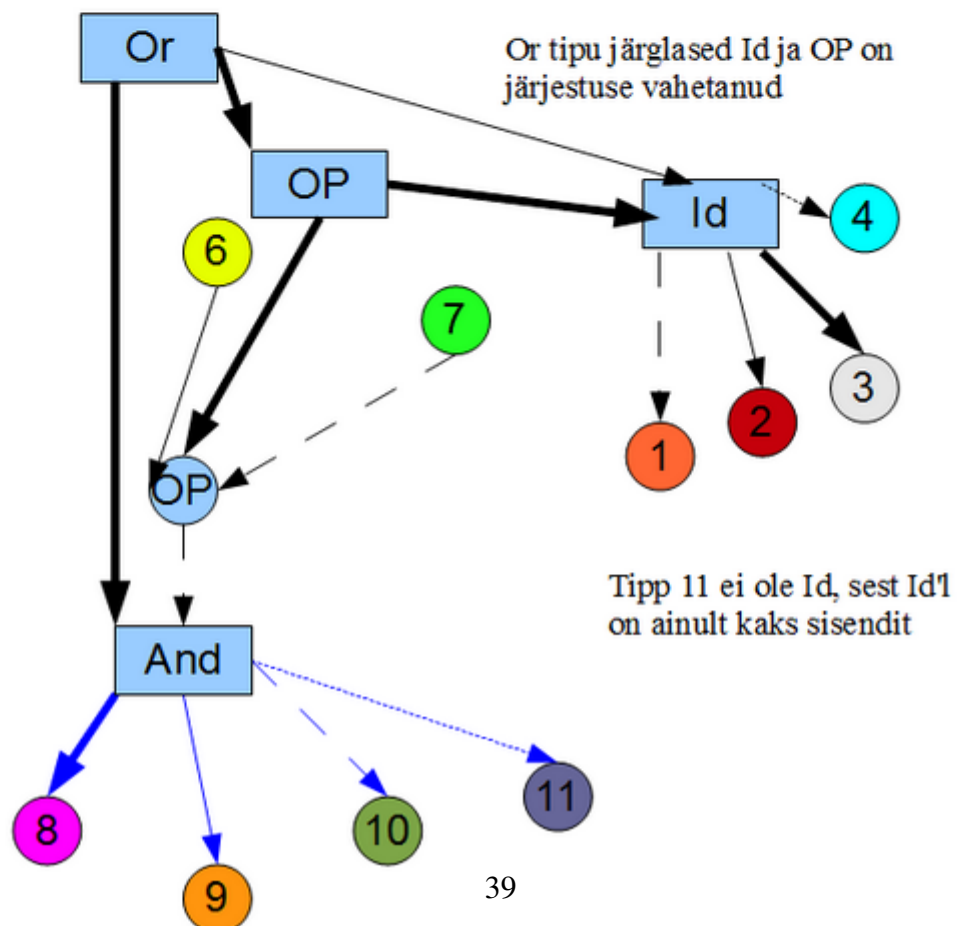


Lisa 12 - Üle Id tipu liigutamine puuserval, mille sihttipp on tõeväärtusväljundiga

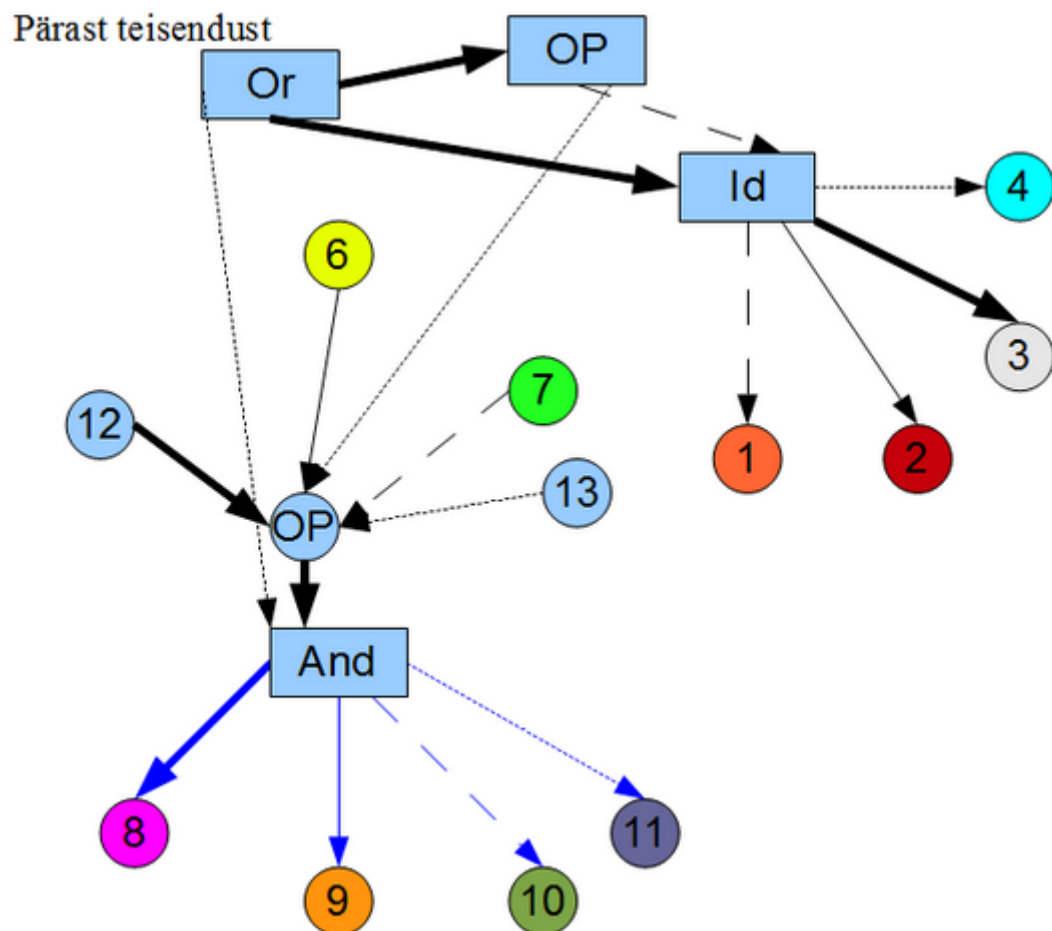
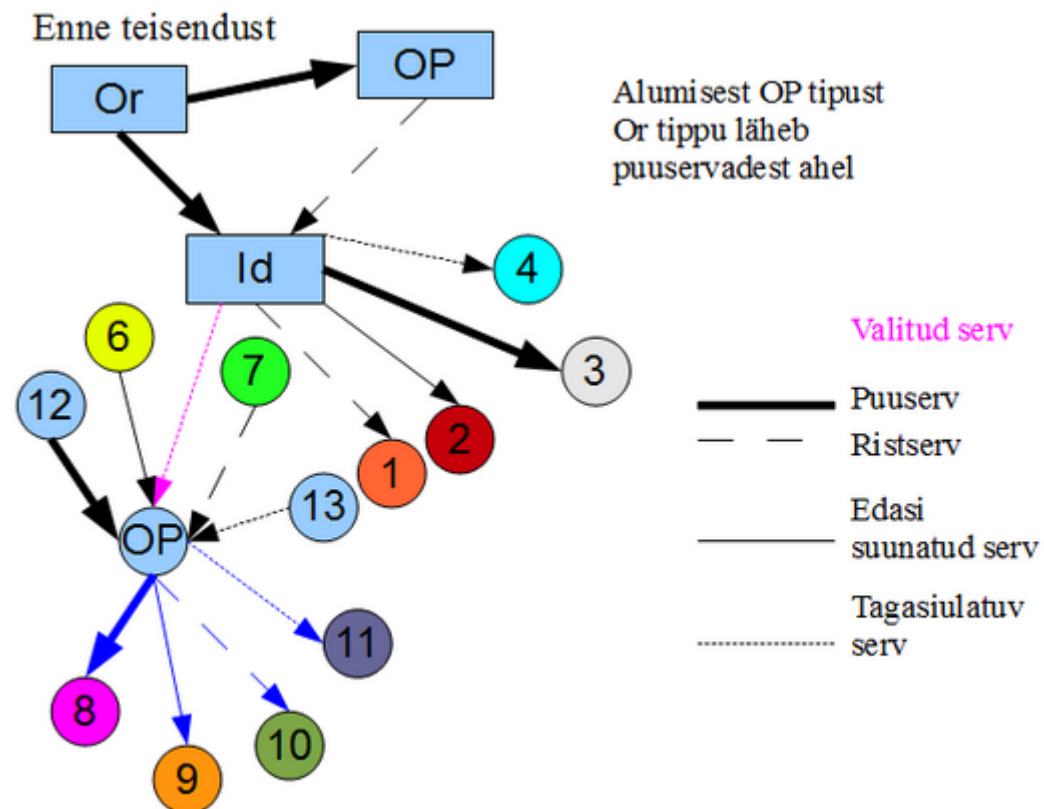
Enne teisendust



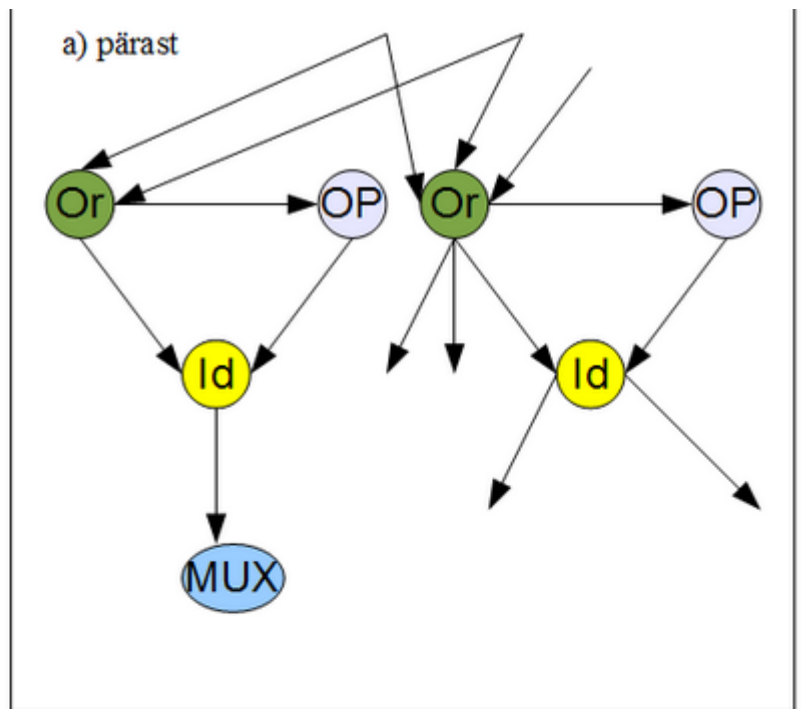
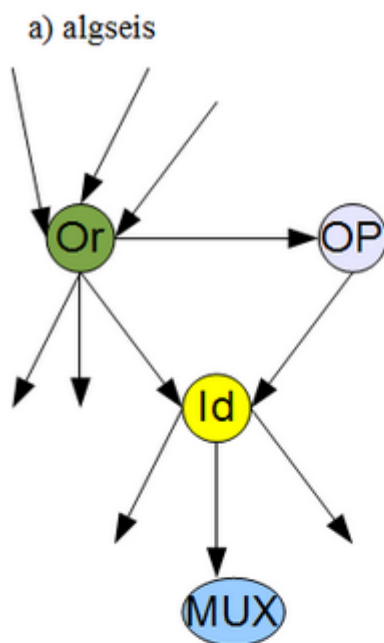
Pärast teisendust



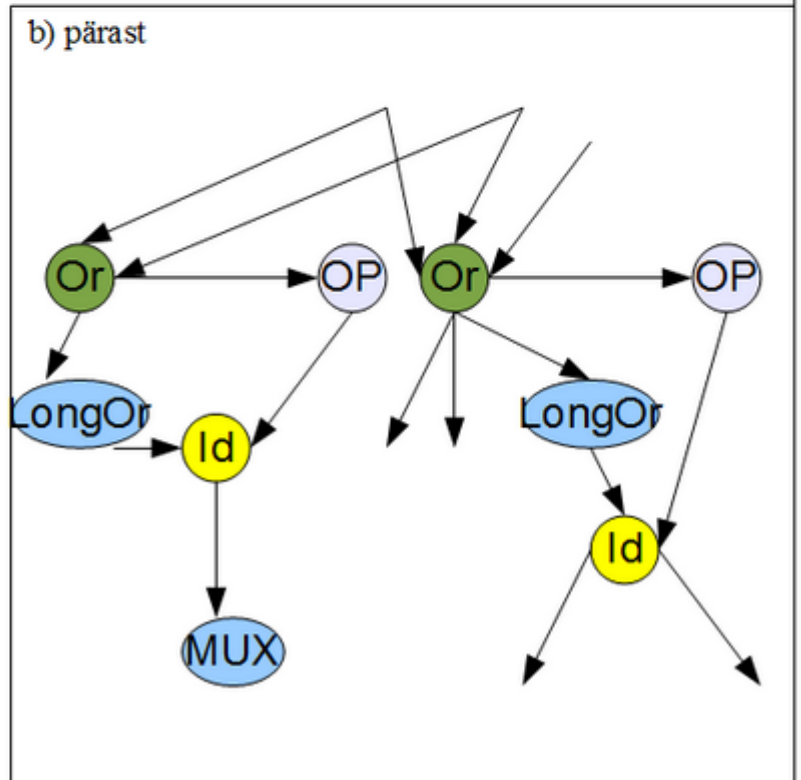
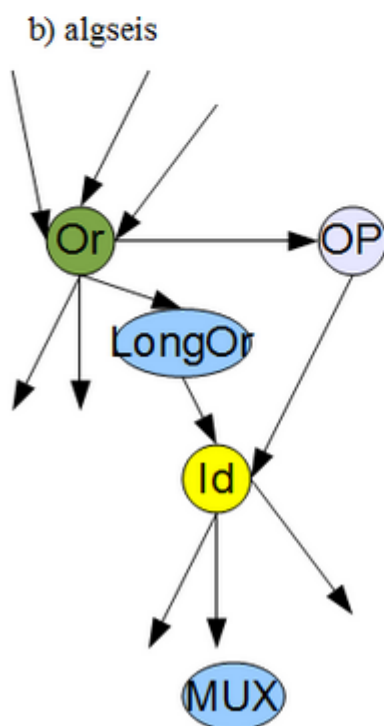
Lisa 13 - Üle *Id* tipu liigutamine tagasiulatuval serval



Lisa 14 - Üle *Id* tipu liigutamine ei tööta

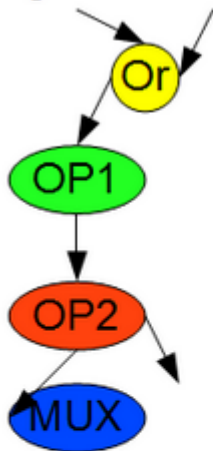


Dimensioone koondades

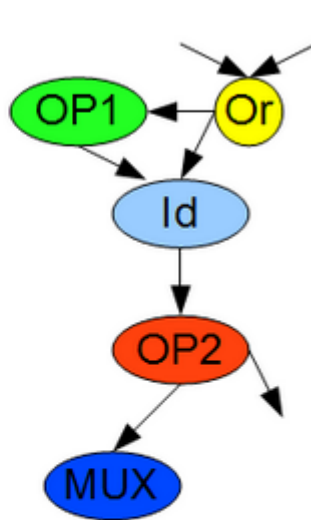


Lisa 15 - Tsüklite eemaldamine ei tööta

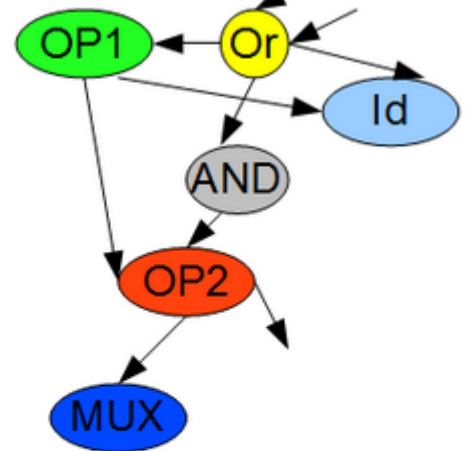
1. tsükli algne fragment



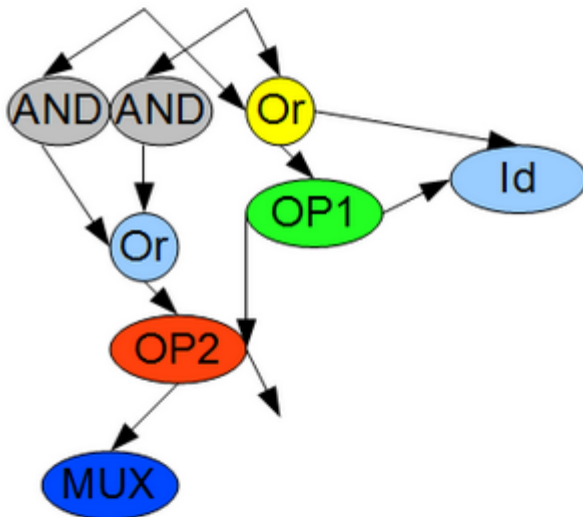
2. pärast „Move Over Or“ teisendust



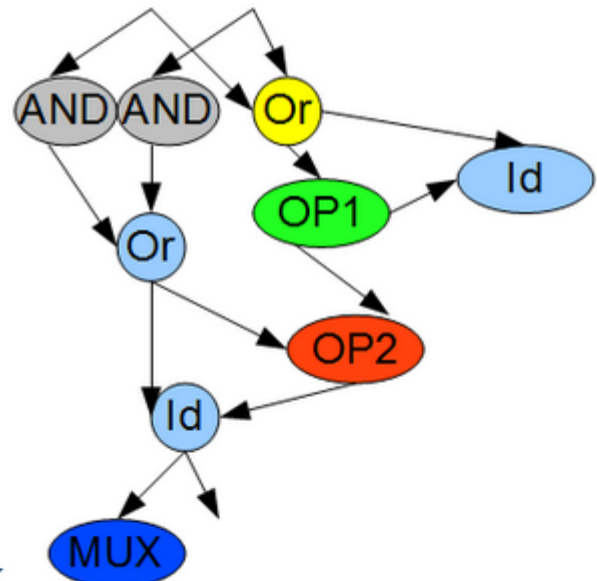
3. pärast „Pass over id“ teisendust



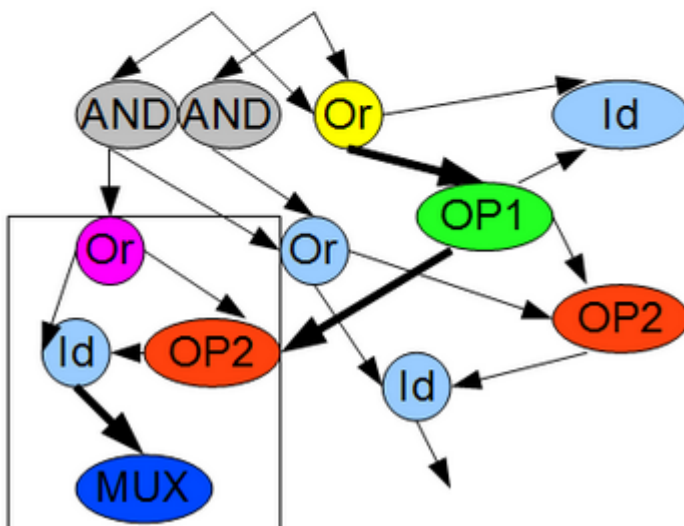
4. pärast „Move Over Or“ teisendust And tipu jaoks



5. pärast „Move Over Or“ teisendust OP2 tipu jaoks



6. pärast erijuhtu olukorrale Or->Id->MUX



Kuigi uuel Or tipul (uued ühe väljundservaga tipud on kastis) pole enam tsüklilist tulnud sisendit ja selles osas on tsükel katkenud, säilib ka joonisel 1 vaadatud ahel, kuhu on lihtsalt tekkinud üks Id tipp ja tegelik tsükel ei ole katkenud.