TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Infotehnoloogia eriala

Roman Jagomägis

Programmeerimiskeel privaatsust säilitavate rakenduste loomiseks

Bakalaureusetöö (4 AP)

Juhendaja: Dan Bogdanov

Autor: ..... "....." mai 2008

Juhendaja: ..... "....." mai 2008

Lubada kaitsmisele

Professor .... "....." mai 2008

TARTU 2008

Sisukord

1 Sissejuhatus **4**

2 Privaatsust säilitavad arvutused **5**

2.1 Ülesandepüstitus 5

2.2 Arvutusraamistik Sharemind 6

3 Lahenduse kirjeldus **7**

3.1 Keele struktuur 7

3.1.1 Registrimasin 7

3.1.2 Kommentaarid 7

3.1.3 Identifikaatorid 8

3.1.4 Instruktsioonid 8

3.1.5 Andmetüübid 8

3.1.6 Avaldised 9

3.1.7 Muutujad 13

3.1.8 Vektoritega töötamine 14

3.1.9 Sisend ja väljund 15

3.1.10 Vookontroll 16

3.1.11 Matemaatilised käsud 17

3.2 Privaatsust säilitavad instruktsioonid 17

3.2.1 Pinumasin 17

3.2.2 Pinu modifitseerimine 18

3.2.3 Kuhja käsud 19

3.2.4 Andmebaasi käsud 20

3.2.5 Aritmeetika 21

3.3 Keele ja interpretaatori ehitamine 21

3.3.1 Interpretaator, selle struktuur ja tööpõhimõte 21

3.3.2 Leksikaanalüsaator 23

3.3.3 Süntaksianalüsaator 25

3.3.4 Andmed 26

3.3.5 Avaldised 28

3.3.6 Instruktsioonid 29

3.3.7 Käsuprotsessor 30

3.3.8 Vookontroll 31

4 Tulemused **32**

5 Kokkuvõte **33**

6 A programming language for creating privacy-preserving applications. 34

7 Kasutatud kirjandus 35

Lisa 1 36

Lisa 2 38

Lisa 3 39

1 Sissejuhatus

Privaatsuse säilitamine on oluline, seda eriti meditsiini, finantsi ja ühiskonna valdkonnas, kus tihtipeale soovitakse koguda ja analüüsida tundlikke andmeid. Andmete kogumine osutub raskeks privaatsuse rikkumise tõttu. Inimesed ei usalda teineteist ning seega jääb arengut toov andmete statistiline analüüs tegemata või satuvad andmed valesse kätesse. Tagajärjena on raskendatud erinevate eluvaldkondade parendamine, andmeid kuritarvitatakse ning ühiskonna areng pidurdub.

On loodud lahendusi, mis lubavad analüüsida andmeid privaatsust rikkumata. Need ei ole kahjuks eriti paindlikud või on nende abil privaatsust säilitavate süsteemide ehitamine liiga keeruline. Seega oleks vaja luua lihtne viis selliste süsteemide ehitamiseks.

Käesolevas töös luuakse suhteliselt madalatasemeline, ent piisavalt paindlik keel ja selle interpretaator, mis põhineb privaatsust säilitavaid arvutusi teha võimaldaval raamistikul Sharemind. Loodava keele abil saab võimalikuks lihtne ja kiire privaatsust säilitavaid arvutusi teostavate rakenduste loomine. Kogu keerukus peidetakse keele interpretaatori sisse.

Lisaks privaatsete süsteemide ehitamise lihtsustamisele on motivatsiooniks ka Sharemindi raamistiku areng. Loodav keel ja interpretaator saavad raamistiku osaks ning pikemas perspektiivis hakkavad toetama selle koosseisus töötava virtuaalmasina autonoomset juhtimist.

Töös kirjeldatakse lühidalt Sharemindi raamistikku, loodud keelt, selle grammatikat, käsustikku ja keele interpretaatorit. Tööga koos esitatakse töötava interpretaatori lähtekood. Lisaks tuuakse ka mõned näidisprogrammid, mis on kirjutatud loodud keeles ja mida saab interpretaatori abil käivitada.

2 Privaatsust säilitavad arvutused

2.1 Ülesandepüstitus

Dan Bogdanov ja Sven Laur on kirjeldanud Sharemindi-nimelise raamistiku [1], mis kujutab endast privaatsust säilitavate operatsioonidega virtuaalmasinat. Selle abil on võimalik ehitada süsteeme, mis tagavad andmete privaatsuse säilimise. Raamistikus on kõik operatsioonid realiseeritud eraldi funktsioonidena, kuid nende abil programmide tegemine on ebamugav. Programmeerija peab kogu aeg kontrollima, kas operatsioonide tegemisel ei tekkinud vahepeal vigu. Vigade kontrollimine viib aga tähelepanu tegelikust lahendusest eemale ning teeb programmikoodi kohmakaks ja raskemini arusaadavaks. Antud probleem on lahendatav sellise interpretaatori ehitamisega, mis kasutab virtuaalmasinat ning oskaks programmide täitmisel ise vigu kontrollida. Lisaks on tarvis luua lihtne keel, milles programme kirja panna.

Teiseks, privaatsust säilitav virtuaalmasin töötab eraldi autonoomse süsteemina, mida peaks olema võimalik mingil määral kaugjuhtida. Seega on pikemas perspektiivis vaja süsteemile lisada saadetavate käskude või koguni suuremate programmide täitmise tugi. Virtuaalmasina juhtimine ühe käsu haaval, kasutades juba sisseehitatud funktsioone, ei luba süsteemile piisavalt keerulisi ja paindlikke ülesandeid anda. Samuti on tavalises kompileeritavas keeles kirjutatud programme keeruline teises süsteemis käivitada. Need probleemid on lahendatavad uue keele loomise ning virtuaalmasinasse keele interpretaatori sisseehitamise abil. Süsteemi osana töötav interpretaator laseb käivitada programme, ilma et süsteemi peaks iga kord taaskäivitama.

Arvestades tekkinud probleeme ja vajadusi koostatakse käesolevas töös lihtne keel, mis võimaldab kasutada Sharemindi privaatseid funktsioone. Loodava keele süntaks on assembleri stiilis, kuid sisaldab ka mõningaid kõrgemate keelte elemente, näiteks matemaatilisi, loogilisi ja sõneavaldisi. Kuna Sharemindi raamistiku operatsioonid töötavad näiteks vektoriseeritud ja sõneandmetega, siis tuleb keeles realiseerida kõik vajalikud andmetüübid. Samuti peab keel mingil määral võimaldama programmivoo kontrolli, mis on vajalik keerulisemate algoritmide koostamiseks. Lisaks luuakse interpretaatori osana lihtne virtuaalmasin avalike funktsioonide ja andmete kasutamiseks. Interpretaatori üheks ülesandeks on ka veakontroll.

Interpretaatorit peab olema võimalik lihtsal moel integreerida ka rakenduse sisse, kus see saaks ükshaaval täita sealt saadud instruktsioone. Samuti peab saama lugeda ja täita suuremaid programme otse failist. Seega on mõistlik luua klass, mis pakuks lihtsat liidest interpretaatori kasutamiseks. Seda klassi kasutades kirjutatakse ka väike rakendus, mis töötaks inimese ja interpretaatori vahelülina, täites inimese poolt sisestatud instruktsioone.

2.2 Arvutusraamistik Sharemind

Sharemind on privaatsust säilitavaid arvutusi võimaldav raamistik, mille alusideeks on osakutega arvutamine. Raamistikus kasutatakse andmete salvestamiseks aditiivset ühissalastusskeemi, mis põhimõtteliselt tähendab andmete osakuteks jagamist ning viimaste erinevate osapoolte vahel ära jaotamist. Iga osapool saab andmetest ühe osaku, millest ei ole võimalik originaalseid andmeid taastada. Nüüd saavad osapooled nende andmetega arvutusi teha, kasutades raamistikus kirjeldatud turvalisi ühisarvutuse protokolle. Hetkel toetab Sharemind kolme osapoolega konfiguratsiooni. Sharemindi protokollid on informatsiooni-teoreetiliselt turvalised ausas-kuid-uudishimulikus mudelis. Kuigi see ei kaitse andmeid pahatahtlike osapoolte eest, annab see rohkem kui mõistliku privaatsusgarantii.

Raamistik kujutab endast pinumasina põhimõttel töötavat virtuaalmasinat, millel on andmetega opereerimiseks sisse ehitatud vajalikud pinud, kuhi, andmebaas ja hulk erinevaid operatsioone, mida saab mainitud andmestruktuurides sisalduvate andmetega teostada. Virtuaalmasin koosneb andmekaevandusserveritest, mida edaspidi nimetame *mineriteks*. Iga osapoolel jookseb üks *mineri* instants, millel on ligipääs arvutivõrku. *Minerid* omakorda kujutavad endast autonoomseid virtuaalmasinaid — igal *mineril* on oma pinu ja muud vajalikud meetmed osakute töötlemiseks ja säilitamiseks ning käskude täitmiseks. Samuti on olemas üks keskne server — kontroller, mis juhib *minerite* tööd.

Sharemind töötab nii, et kontrollerist sisestatud andmed jaotatakse osakuteks ning jaotatakse seejärel *minerite* vahel ära. Iga *miner* saab ühe osaku ning salvestab selle mällu. Siis annab kontroller *mineritele* käske, mida nende andmetega teha. *Minerid* suhtlevad omavahel üle arvutivõrgu, kasutades turvalisi protokolle, mis säilitavad andmete privaatsuse. Tulemusena muudetakse *minerite* mälus olevad osakud nii, et nendes kajastuvad tehtud operatsioonid, kuid andmete privaatsus ei ole rikutud.

Raamistiku põhiliseks disainieesmärgiks on efektiivsus ja kasutuslihtsus. Kasutatavad protokollid ei tugine aeganõudvatel krüptograafilistel primitiividel ning seega ei vaja nende täitmine suurt arvutusjõudlust. Raamistiku eeliseks on võime opereerida suurte andmehulkadega. Implementeeritud protokollid on komponeeritavad, seega saab neid kasutada järjestikuliselt ja paralleelselt. See lubab teha andmetega järjest erinevaid operatsioone — sisuliselt kirjutada programme. Sharemindi kontroller pakub liidest *mineritele* käskude saatmiseks ning võimaldab seega ehitada privaatsust säilitavaid arvutusi teostavaid rakendusi.

3 Lahenduse kirjeldus

3.1 Keele struktuur

Keele instruktsioonid jagunevaks kaheks: Sharemindi raamistikul põhinevad privaatsust säilitavad instruktsioonid ja avalikud instruktsioonid. Selles osas käsitleme erinevaid keele osi, mis on seotud avalike andmete ja nendega opereerimisega.

3.1.1 Registrimasin

Registrimasin on seade, millel on lõplik arv registreid $x_1, ..., x_m$. Iga register on võimeline hoidma mistahes suurusega täisarvu. Registrimasina programm on lõplik instruktsioonide loetelu, mida programmi täitmisel järjest läbitakse. Seda tüüpi masinale on iseloomulik, et instruktsioonidele antakse operandidena ette konstantsed andmed ja/või registrid.

Masinas on ka käsuloendaja register, milles hoitakse hetkel täidetava käsu aadressi. Täitmist alustatakse esimesest instruktsioonist ning iga käsu täitmisega suurendatakse käsuloendajat ühe võrra. Käsuloendajat muutes on võimalik programmi täitmist suunata. Nii suunavad vookontrolli käsud programmi täitmist loendaja sisu muutmise teel.

Antud töös interpretaatori osana ehitatav avalik virtuaalmasin kujutab endast registrimasina varianti. Programmi täitmisel hoitakse mälus mingi hulk muutujaid, mida võib pidada registriteks. Neid saab käskudele operandidena ette anda. Samas saab loodavas masinas instruktsioonidele ette anda ka keerulisemaid andmeid nagu avaldised. Käskude täitmine on realiseeritud käesolevas peatükis kirjeldatuga sarnasel viisil.

3.1.2 Kommentaarid

Kommentaarid on lähtekoodi osa, mida interpretaator ignoreerib. Seega ei ole neil programmivooga mingit seost. Kommentaaride ülesandeks on lubada programmeerijal lisada oma lähtekoodi märkusi või seletusi.

Sharemindi keel toetab kahte tüüpi kommentaare: ühe- ja mitmerealised ehk plokk-kommentaarid.

```
// üherealine kommentaar
/* mitmerealine ehk
plokk-kommentaar */
```

Üherealine kommentaar sisaldab endas kõike, mis asub pärast kaldkriipsude paari `//` kuni sama rea lõpuni. Mitmerealine kommentaar aga sisaldab kõike, mis on märkide `/*` ja esimese esineva `*/` märkide kombinatsiooni vahel, ning lubab seega sisaldada

endas rohkem kui ühte rida teksti. Ülejäänud programmifaili sisu, mille puhul ei rakendata märke `//`, `/*` ja `*/`, käsitletakse instruktsioonide ja avaldistena ning täidetakse interpretaatori poolt.

3.1.3 Identifikaatorid

Identifikaatoriks peetakse ühest või rohkemast tähest, numbrist või alakriipsust "_" koosnevat jada, mis algab kas tähe või alakriipsuga. Teised märgid ei või selles jadas esineda. Samuti ei või identifikaator alata numbriga.

Identifikaatoreid kasutatakse keeles muutujate nimede ja pealdiste üles leidmisel. Ka keele reserveeritud sõnad on olemuselt identifikaatorid ning neid ei või kasutada muutujate või pealdiste nimedena. Reserveeritud sõnade hulka kuuluvad keeles kasutusel olevate instruktsioonide nimed, tüübid ning sõnalised väärtused "`true`" ja "`false`".

3.1.4 Instruktsioonid

Keele põhilisteks komponentideks on käsud ehk instruktsioonid. Iga instruktsioon koosneb operatsiooni mnemoonilisest koodist, millele võib järgneda null või rohkem operande. Operatsiooni kood viitab avaliku või privaatse virtuaalmasina instruktsioonile.

Selline käsuformaat sarnaneb tavalise assemblerkeele omaga. Samas on Sharemindi keele oluline erinevus operandide väljendusvõimsuses — iga operand on mingit tüüpi avaldis. See lubab tavalisest assemblerist paindlikumat ja mugavamat programmeerimist, kuna nüüd võib operandi väärtus sõltuda korraga mitmest teisest väärtusest. Samuti võimaldab see instruktsioonidel kontrollida etteantavate andmete tüüpe ning käituda tüübist lähtudes erinevalt.

Programm koosneb instruktsioonide jadast, mida täidetakse üks teise järel. Täitmise ajal võib interpretaator vookontrolli instruktsioonide abil hüpata jada mõnda märgistatud kohta, võimaldades nii keerulisemate algoritmide loomist ja käivitamist.

3.1.5 Andmetüübid

Sharemindi raamistik kasutab funktsioonide sisendiks ja väljundiks erinevat tüüpi andmeid. Viimaseid tuleb vahendada läbi instruktsioonide operandide nii, et instruktsioonid oskaks nendega midagi peale hakata. Andmeid on tarvis kuidagi töötleda, teha nendega erinevaid operatsioone ja teostada ka veakontrolli. Selleks peab keele interpretaator teadma, missuguste andmetega on tegu, ning kuidas nende puhul käituda. Lähtudes Sharemindi raamistikust ning tüüpide vajadusest on antud keeles loodud tugi järgmiste andmetüüpide jaoks: 32-bitine märgita täisarv (`int`), 32-bitiste täisarvude massiiv ehk vektor (`int[]`), sõne (`string`), tõeväärtus (`bool`) ja märgend.

Täisarvud. Sharemindi raamistiku ja keele põhiliseks andmetüübiks on 32-bitine märgita täisarv, mille väärtused saavad olla vahemikus 0 kuni $2^{32} - 1$. Maksimaalsest järgmine arv loetakse uuesti 0-ks, sellest järgmine 1-ks jne. Sisuliselt käsitletakse täisarve modulo 2^{32}.

Vektorid. Täisarvudest on võimalik luua vektor. Vektor kujutab endast dünaamilist massiivi, kuhu saab lisada uusi elemente ning lugeda, muuta ja eemaldada olemasolevaid elemente. Viimaseid saab vektorist kätte indeksite kaudu. Põhimõtteliselt pole vektori pikkusel teisi piiranguid kui arvuti vaba operatiivmälu maht.

Sõned. Sõne tüüp kujutab endast märkide jada ning võimaldab käsitleda ja töödelda teksti. Sõnede tugi Sharemindi keeles on lisatud selleks, et oleks võimalik väljastada teateid ja informatsiooni.

Keeles on olemas ka sõnede konkatenatsiooni tugi, mis lubab olemasoleva sõne otsa liita mistahes teist tüüpi avaldise väärtuse tekstilise esituse ning pärast näiteks väljastada see ekraanile. Tänu konkatenatsiooni lihtsusele on väljundi konstrueerimisel võimalik palju aega kokku hoida. Selle tüübi pikkust piirab samuti ainult arvuti vaba operatiivmälu maht.

Tõeväärtused. Kuna keeles on loogiliste avaldiste tugi, siis on vajalik ka tõeväärtuse tüüp. Tõeväärtus iseloomustab loogilisi avaldisi ja nende vahetulemusi ning saab olla kas tõene või väär (`true` või `false`).

Märgendid. Märgend on andmetüüp, mida käsitletakse teistest tüüpidest erineval viisil. Keeles ei ole võimalik deklareerida märgendi tüüpi muutujat, samas on võimalik deklareerida märgendit, mis määrab programmis ära koha, kuhu interpretaator saab vajadusel programmi täitmisel liikuda. Samuti saab märgendit anda operandina vookontrolli teostavatele instruktsioonidele. Selle tõttu tuleb märgendeid osaliselt käsitleda kui andmetüüpi.

3.1.6 Avaldised

Kõik Sharemindi keele instruktsioonide operandidena käsitletavad andmed on avaldised. Avaldised jagunevad kaheks: triviaalsed ja mittetriviaalsed. Triviaalseid avaldisi iseloomustavad andmete tüüp ja tüübile vastav väärtus. Triviaalsed avaldised on näiteks konstant ja muutuja. Mittetriviaalseteks loeme aga selliseid avaldisi, millel on keerulisem struktuur. Avaldise väärtusel on alati kindel andmetüüp.

Sharemindi keeles on võimalik teha avaldistega tehteid. Tehted on mittetriviaalsed avaldised. Neil peab olema määratud tüüp, tehtekood ja tehte argumendid, mille peal tehet rakendatakse. Tehte argumendid on samuti avaldised ning seega võivad nad

omakorda olla kas andmed või tehted. Nii tekib ettekujutus mittetriviaalsest avaldisest kui puustruktuurist ning selle tüübi määrab puu juure tehtekood ja esimese argumendi tüüp. Tehte määrab keeles tehtemärk ehk operaator.

Avaldisi võib keele andmetüüpidele toetudes klassifitseerida järgmiselt: matemaatilised, loogilised, sõne- ja märgendiavaldised. Kõigi klasside jaoks välja arvatud märgendid, on defineeritud teatud hulk võimalikke tehteid. Järgnevalt vaatleme avaldiseklasse eraldi.

Matemaatilised avaldised. Seda liiki triviaalseteks avaldiseks võivad olla täisarvu tüüpi konstandid, muutujad või vektori elemendid. Samasse hulka võib lugeda ka vektori tüüpi avaldisi, kuid nende jaoks pole kirjeldatavas keeles tehteid defineeritud.

Täisarvulise tüübiga töötamiseks on Sharemind keeles defineeritud järgmised tehted: liitmine ($+$), lahutamine ($-$), korrutamine ($*$), jagamine ($/$), jäägi leidmine ($\%$), astendamine ($\hat{}$) ja vastandväärtustamine (unaarne $-$). Kõik need tehted moodustavad koos oma argumentidega täisarvu tüüpi avaldised ning selle tõttu saavad nende väärtusteks olla ainult täisarvud. Jagamisel saab avaldise väärtuseks jagatise täisosa. Matemaatiliste avaldiste puhul peavad konsistentsuse tagamiseks nii tehte, kui kõigi selle argumentide (avaldiste) tüübid olema täisarvud.

Mainitud tehted (v.a. unaarne miinus) on vasakassotsiatiivsed. See tähendab, et tehte argumentidest väärtustatakse esimesena vasakpoolne argument ning siis parempoolne argument. Seejärel kasutatakse tehte rakendamisel vasakut argumenti esimese parameetrina ja paremat argumenti teise parameetrina. Näiteks avaldise $A - B$ puhul lahutatakse väärtusest A väärtus B mitte vastupidi.

Toome välja ka matemaatiliste avaldiste grammatika:

```
AVALDIS :
| KONSTANT
| VEKTORI_ELEMENT
| MUUTUJA
| AVALDIS + AVALDIS
| AVALDIS - AVALDIS
| AVALDIS * AVALDIS
| AVALDIS / AVALDIS
| AVALDIS % AVALDIS
| AVALDIS ^ AVALDIS
| - AVALDIS
| (AVALDIS)
```

Võib täheldada, et grammatika on rekursiivse ülesehitusega ning toetab lõpmatult

sügavaid avaldisi.

Loogilised avaldised. Antud liiki avaldised on alati loogilise tõeväärtuse tüüpi `bool` ning nende võimalikeks väärtusteks on kas tõene või väär (`true` või `false`). Triviaalseteks loogilisteks avaldisteks võivad Sharemindi keeles olla ainult konstandid ja muutujad.

Seoses tõeväärtuse tüübiga on keeles defineeritud hulk tehteid (operaatoreid), mis ise on küll loogilised avaldised, kuid argumentide poolest võib neid jagada võrdlusoperaatoriteks ja loogilisteks operaatoriteks: esimesed võtavad argumentideks ainult matemaatilisi või sõneavaldisi ja teised ainult loogilisi avaldisi. Võrdlusoperaatorid on mõeldud matemaatiliste ja sõneavaldiste väärtuste võrdlemiseks. Keeles on implementeeritud järgmised vasakassotsiatiivsed võrdlusoperaatorid: väiksem-või-võrdne ($\leq$), suurem-või-võrdne ($\leq$), väiksem-kui ($<$), suurem-kui ($>$), võrdne ($==$), mittevõrdne ($!=$). Loogilisi operaatoreid on kolm: konjunktsioon (&&), disjunktsioon (||) ja loogiline eitus (!).

Konjunktsioon ja disjunktsioon on samuti vasakassotsiatiivsed, kuid nende argumentide väärtustamine käib teistest operaatoritest erinevalt. Esimesena väärtustatakse vasakpoolne argument. Seejärel sõltuvalt tehtest käitutakse taas erineval viisil. Konjunktsiooni puhul vaadatakse esimese argumendi väärtust ning kui see on "väär", siis parempoolset argumenti enam ei väärtustata, vaid tagastatakse kogu avaldise väärtuseks "väär". Disjunktsiooni puhul väärtustatakse kogu avaldis kohe tõeseks, kui vasakpoolse argumendi väärtuseks on "tõene".

Loogiline eitus on paremassotsiatiivne tehe. See tähendab, et esimesena väärtustatakse tehte parempoolne argument (tehet rakendatakse paremalt vasakule). Loogiline eitus muudab oma parempoolse argumendi tõeväärtuse vastupidiseks: tõene muudetakse vääraks ning väär tõeseks.

Matemaatiliste avaldiste puhul töötavad võrdlusoperaatorid intuitiivselt - sisuliselt nad võrdlevad omavahel täisarve. Sõneavaldiste puhul on võrdlustehe keerulisem: sõne A on suurem sõnest B, kui esimene samal positsioonil olev mittevõrdne sümbol on sõne A puhul tähestikus sõne B omast kaugemal - seda nimetatakse ka leksikograafiliseks võrdluseks.

Toome välja ka loogiliste avaldiste grammatika, kus prefiks "LOOG" viitab loogilist tüüpi avaldisele ning "MS" matemaatilise või sõne tüüpi avaldisele:

LOOG_AVALDIS :
| KONSTANT
| MUUTUJA
| MS_AVALDIS $\leq$ MS_AVALDIS

```
| MS_AVALDIS ≥ MS_AVALDIS
| MS_AVALDIS < MS_AVALDIS
| MS_AVALDIS > MS_AVALDIS
| MS_AVALDIS == MS_AVALDIS
| MS_AVALDIS != MS_AVALDIS
| LOOG_AVALDIS && LOOG_AVALDIS
| LOOG_AVALDIS || LOOG_AVALDIS
| (LOOG_AVALDIS)
| ! LOOG_AVALDIS
```

Sõneavaldised. Ka triviaalseks sõneavaldiseks saab olla kas konstant või muutuja. Seda tüüpi avaldiste puhul on defineeritud lisaks loogilistele tehetele ka üks vasakassotsiatiivne sõnede liitmistehe ehk konkatenatsioon.

Sõnede liitmise võimalus on realiseeritud (+) operaatori abil ning sarnaneb C++ või Java keeles oleva sõnede liitmisega. Eksisteerib üks kitsendus: konkateneerida saab ainult siis, kui tehte esimene argument on sõne (SÕNE+AVALDIS) - see määrab tulemusena saadud avaldise tüübi, milleks peab olema samuti sõne. Kui näiteks liita vektorile sõne, siis ei ole tehte semantika enam selge. Samas liites sõnele ükskõik mis teist tüüpi avaldis konverteeritakse viimase väärtus samuti sõneks ning tulemusena liidetakse üks sõne teise otsa, mis ongi olemuselt konkatenatsioon.

Märgendid. Nagu varem mainitud, saab märgendeid anda operandina ette vookontrolli instruktsioonidele. Kuna operandid on avaldised, siis võimaldamaks märgendit instruktsioonidele ette anda peab lugema märgendid erilist tüüpi avaldisteks. Sisuliselt antakse sellise avaldisega instruktsioonile üle märgendi nimi, mille järgi saab instruktsioon vastavast tabelist üles leida märgendile vastava koha programmis. Muud funktsiooni seda tüüpi avaldistel ei ole.

Tehete prioriteedid. Antud alampeatükis on kirjeldatud hulk erinevaid tehteid kuid jäid käsitletamata nende prioriteedid. Tehete prioriteedid määravad, millises järjekorras neid täidetakse - kõrgem prioriteet esimesena. Sharemindi keeles on tehete prioriteedid järgmised (üleval kõrgemad prioriteedid):

1. astendamine (^)
2. loogiline eitus (!)
3. unaarne miinus (−)
4. korrutamine (∗), jagamine (/), jäägi leidmine (%)

5. liitmine (+), lahutamine (−)
6. väiksem-või-võrdne (≤), suurem-või-võrdne (≥), väiksem-kui (<), suurem-kui (>)
7. võrdlus (==)
8. konjunktsioon (&&)
9. disjunktsioon (||)

Avaldisi saab tehete järjekorra muutmiseks grupeerida. Nii on näiteks avaldiste $4 + 5 * 2 + 3$ ja $4 + 5 * (2 + 3)$ väärtusteks vastavalt 17 ja 29.

3.1.7 Muutujad

Algoritmide realiseerimiseks on tarvis käsitletavaid andmeid töötleda ja säilitada. Sharemindi keeles täidavad avalike andmete meeles pidamise ülesannet muutujad. Iga muutuja on avaldis, seega on tal ka tüüp. Tüüp määratakse muutuja deklareerimisel ning hiljem ei ole võimalik seda enam muuta. Muutujale antakse ka unikaalne tõstutundlik nimi.

Muutuja deklareerimine on lihtne — uuel real tuleb kirjutada muutuja tüüp ning kohe pärast seda muutuja nime tähistav identifikaator ilma jutumärkideta:

```
muutuja_tüüp muutuja_nimi
```

Deklareerimisel väärtustatakse muutuja sõltuvalt tema tüübist teatud vaikeväärtusega. Tabelis 1 toome välja nimekirja andmetüüpidest ja nendele vastavatest vaikeväärtustest.

Tüüp	**Vaikeväärtuse tekstiline esitus**	**Kommentaar**
`int`	0	null
`int[]`	{}	tühi vektor
`bool`	`true`	tõene
`string`	""	tühi sõne

Tabel 1: Andmetüübid ja nende vaikeväärtused

Pärast muutuja deklareerimist saab selle väärtust ümber defineerida (muuta). Selleks on Sharemindi keelde konstrukeeritud `mov`-nimeline instruktsioon. See võtab kaks operandi, millest esimene peab olema muutuja, kuhu uus väärtus salvestatakse ning teine peab olema avaldis, kust see väärtus võetakse. Programmikoodi tuleb kirjutada:

```
mov muutuja avaldis
```

Avaldis peab üldjuhul andmete konsistentsuse tagamiseks olema muutujaga sama tüüpi. Kui aga avaldis on sõne tüüpi, siis üritatakse sealt vastavalt muutuja tüübile välja lugeda seda tüüpi väärtus. Selleks peab sõne sisaldama muutuja tüübile vastava väärtuse korrektset tekstilist esitust. Antud implementatsioonis eeldatakse, et sisestatakse korrektseid andmeid, seega sõnest väärtuse välja lugemise ebaõnnestumisel lõpetatakse programmi töö.

3.1.8 Vektoritega töötamine

Vektor kujutab endast dünaamilist massiivi, kus igal elemendil on järjekorranumber ehk indeks. Sharemindi keeles on võimalik luua täisarvude vektorit. Seda tehakse järgmise deklaratsiooniga:

```
int[] muutuja_nimi
```

Deklaratsiooni tulemusena luuakse tühi täisarvu tüüpi vektor. Edasi on sellega võimalik teha mitmeid operatsioone, mis on realiseeritud instruktsioonidena. Vektoritega töötamiseks mõeldud instruktsioonide esimese operandina võetakse muutuja, sest vektori tüüpi konstantidega opereerimine ei oma antud keeles mõtet. Näiteks konstantse vektori puhul läheks konstant pärast elemendi lisamist kaduma ning sellega ei saaks edasi midagi teha. Muutuja väärtus aga jääb mällu.

Vektori lõppu elemendi lisamine. Instruktsioon `vecinsert` võtab esimese operandina eelnevalt deklareeritud vektori muutuja ja teise operandina täisarvu tüüpi avaldise. Siis lisab `vecinsert` täisarvu väärtuse vektori lõppu. Vektori pikkus suureneb ühe võrra.

```
vecinsert vektori_muutuja täisarv
```

Vektori lõpust elemendi eemaldamine. Etteantud vektori lõpust eemaldatakse üks element ning vektori pikkus kahaneb ühe võrra.

```
vecremove vektori_muutuja
```

Vektori suuruse lugemine. Teine operand on täisarvu tüüpi muutuja, kuhu salvestatakse esimese operandina antud vektori pikkus.

```
   vecsize vektori_muutuja täisarvuline_muutuja
```

Vektori tühjendamine. Etteantud vektor tühjendatakse - kustutatakse kõik tema elemendid ning vektori pikkuseks saab 0.

```
   vecclear vektori_muutuja
```

3.1.9 Sisend ja väljund

Sisendi ja väljundi instruktsioonide kaudu lubab Sharemindi keel programmi ja kasutaja vahelist suhtlust - programmis on võimalik väljastada andmed ekraanile ja küsida kasutajalt läbi käsurea sisendit.

Andmete ekraanile väljastamiseks on kaks instruktsiooni: `print` ja `println`. Nende ainus vahe on väljundi lõpus uuele reale minek - `print` ei lisa väljundi lõppu uue rea märki, `println` aga lisab, viies kursori uuele reale. Mõlemad käsud võtavad ühe operandi, mille tüübiks võib olla suvaline keeles deklareeritatavatest andmetüüpidest. Operandiks võib olla nii konstant, muutuja, kui nendest mingis kombinatsioonis koosnev keerulisem legaalne avaldis. Ekraanile väljastatakse operandi väärtuse tekstilise esituse. Toome mõned näited andmete ekraanile väljastamisest:

```
   int a // defineerime täisarvulise muutuja a
   mov a,7 // paneme muutuja a väärtuseks 7
   println a // trükime ekraanile 7 ja viime kursori uuele reale
   println ''muutuja a väärus on ''+(a+1) // trükime ekraanile sõne
ning selle järel 8
   println 3+4*(5+a) // trükime ekraanile avaldise väärtuse, milleks
on 51
```

Andmeid saab programm kasutajalt küsida käsu `input` abil. Sõltuvalt vajalike andmete iseloomust peab käsu operandiks olema vastavat tüüpi muutuja. Kui sisestatud andmete tekstiline esitus on teisendatav tegelikuks andmestruktuuriks, siis salvestatakse tulemus etteantud muutujasse. Vastasel juhul lõpetab programm töö. Käsu `input muutuja` väljakutsel tekib käsureale ”>” märk, mille järel saab kasutaja andmed sisestada.

3.1.10 Vookontroll

Programmile üheks iseloomulikuks omaduseks on täidetavate instruktsioonide järjekord. See järjekord ei pea alati olema lineaarne. Programmi täitmise protsessis on tihtipeale tarvis teha hargnemisi või kordumisi - tuleb otsustada, mida, millal ja mis tingimustel teha. Selleks on Sharemindi keeles loodud vookontrolli meetmed, mis põhinevad märgendi ja "märgendile hüppe" ideel.

Deklareerides märgendeid on võimalik märgistada programmis ära teatud kohad, kuhu hiljem samu märgendeid teatud instruktsioonidele ette andes saab programmi täitmist suunata. Märgendi deklareerimiseks tuleb programmi lähtekoodi eraldi real kirjutada märgendi nime tähistav identifikaator, mille järgi lisatakse koolon:

```
märgendi_nimi:
```

Hüppe tegemiseks on kaks viisi: tingimuslik ja tingimuseta hüpe. Tingimuseta hüppe tegemiseks on instruktsioon `jmp`. See võtab ühe märgendi tüüpi argumendi, ning jätkab programmi täitmist sealt, kus antud märgend oli deklareeritud. Märgendi operandina instruktsioonile ette andmiseks tuleb operandi kohale kirjutada koolon, millele järgneb vajaliku märgendi nime tähistav identifikaator, näiteks:

```
jmp :märgendi_nimi
```

Toome välja lihtsa programmi, mis lõpmatus tsüklis trükib ekraanile arve:

```
int a
tsykk:
println a
mov a,a+1
jmp :tsykk
```

Tingimuslik hüpe tähendab seda, et enne tegelikku hüpet tehakse otsus, kas hüpata või mitte. Otsuse tegemiseks on keeles olemas käsk `cmp`, mis võtab operandiks loogilise avaldise. Avaldise väärtus salvestatakse `cmp` poolt interpretaatori võrdlustulemuste pinusse, kust hiljeb saavad selle väärtuse kätte käsud `jt` ja `jf`. Käsu nimi `jt` tuleneb inglisekeelsest fraasist *jump if true* (hüppa, kui tõene) ning `jf` vastavalt fraasist *jump if false* (hüppa, kui väär). Need kaks käsku saavad operandina ette märgendi ning teevad täpselt seda, mida käskude nimed ütlevad: hüppavad märgendi poolt osutatud kohta programmis, kui eelnevalt võrdlustulemuste pinusse salvestatud väärtus on vastavalt kas tõene või väär. Järgnev näidisprogramm trükib ekraanile arvud nullist üheksani ning kõige lõppu trükib sõne "Lõpp!".

```
int a
tsykk:
println a
mov a,a+1
cmp a<10
jt :tsykk
println ''Lõpp!''
```

3.1.11 Matemaatilised käsud

Matemaatiliste andmetega töötamiseks on keelde lisatud mõningad kasulikud instruktsioonid.

Käsk `inc` võtab operandina täisarvu tüüpi muutuja ning suurendab selle väärtust ühe võrra. Kui muutuja `a` väärtus on 7, siis pärast `inc a` käivitamist on `a` väärtuseks 8. Sarnase tööpõhimõttega on käsk `dec`, mis muutuja väärtuse suurendamise asemel vähendab seda.

Juhuslike arvude genereerimiseks on instruktsioon `random`, mis võtab operandina täisarvu tüüpi muutuja ja paneb selle väärtuseks krüptograafiliselt turvalise juhusliku täisarvu.

3.2 Privaatsust säilitavad instruktsioonid

Sharemindi raamistikul põhinevad instruktsioonid võimaldavad teha operatsioone andmetega privaatsust säilitaval viisil.

Raamistik töötab privaatse virtuaalmasinana, ning selle sees täidetavad käsud töötavad juba turvaliselt osadeks jaotatud andmetega - nimetame viimaseid privaatseteks andmeteks. Keele intepretaator töötab aga avaliku virtuaalmasinana ning selle muutujates olevad andmed on avalikud ning kõigile nähtavad. Seega kõik andmed, mis saadetakse avalikust virtuaalmasinast privaatsesse, muutuvad privaatseteks. Antud osas käsitleme instruktsioone, mida täidetakse privaatses virtuaalmasinas.

3.2.1 Pinumasin

Kuna privaatne virtuaalmasin on oma olemuselt pinumasin, siis käsitleme lühidalt pinumasina mõistet ja tööpõhimõtet.

Pinumasin kujutab endast arvutusmudelit, milles arvutimälu esitatakse ühe või rohkema pinuna. Pinumasina käsud töötavad pinu tipus olevate väärtustega. Pinust võetakse arvutusteks vajalikud väärtused, teostatakse nendega teatud arvutus ning tulemus pannakse tagasi pinusse.

Tavaliselt on sellistel masinatel ka suvalise mäluaadressi poole pöördumise käsud. Sharemindi virtuaalmasinas on realiseeritud ka kuhja ehk mälupesaga töötamise käsud. Nii saab nende abil andmed mälust pinusse ja vastupidi laadida.

3.2.2 Pinu modifitseerimine

Järgmised käsud on mõeldud privaatse virtuaalmasina pinuga töötamiseks. Need on realiseeritud Sharemindi kontrollerliidest kasutades. Tabelis 2 kirjeldame lühidalt iga käsu ülesannet. Operandi nime järel olev küsimärk tähistab seda, et antud operand võib olemata jääda ja sellisel juhul loetakse tema väärtus üheks.

Käsk	**Kirjeldus**
`dup n?`	Loeb pinust n-elemendilise vektori ning paneb pinusse tagasi vektori koopia. Sellega duplitseerib käsk pinu ülemised n elementi. Kui pinus on 10 elementi, siis pärast käsu `dup 5` käivitamist on pinus 15 elementi. Kaks ülemist 5-elemendilist vektorit on seejärel võrdsed ning sama elementide järjekorraga.
`swap n?`	Vahetab omavahel ära pinu kaks ülemist n-elemendilist vektorit. Antud suurusega vektorid loetakse pinust maha ning pannakse tagasi vastupidises järjekorras.
`stacksize size`	Võtab operandiks muutuja ning paneb selle väärtuseks pinus olevate elementide arvu.
`clearstack`	Käseb privaatsel virtuaalmasinal oma pinu tühjendada.
`push src`	Paneb etteantud vektori pinusse muutes andmed privaatseks.
`pop dest, n?`	Võtab pinust n-elemendilise vektori ning salvestab selle käsu esimese operandina oleva vektori tüüpi muutujasse. Kui pinust võetakse ainult üks element, siis tagastatakse ühe-elemendiline vektor.
`pushgte n?`	Võtab pinust kaks n-elemendilist vektorit ning paneb tagasi vektori, mis koosneb kahe vektori paariti suurematest elementidest.
`pushgteres n?`	Võrdleb pinu kahe ülemise n-elemendilise vektori elemente ning paneb tagasi vektori, mille iga bitt tähistab vastavast paarist suuremat elementi. Kui tõeväärtuseks on ”tõene”, siis esimese vektori vastav väärtus oli suurem.
`pusheqres n?`	Võrdleb pinu kahe ülemise n-elemendilise vektori elemente ning paneb tagasi vektori, mille iga bitt tähistab, kas vektorite vastavad väärtused olid võrdsed või mitte.

Tabel 2: Käsud privaatse virtuaalmasina pinuga töötamiseks

Oluline on tähele panna, et need võrdlused on privaatsust säilitavad.

3.2.3 Kuhja käsud

Tabel 3 kirjeldab privaatse virtuaalmasina kuhjaga töötavaid instruktsioone.

Käsk	**Kirjeldus**
`hpush addr`	Loeb antud aadressilt kuhjast väärtuse ning paneb selle pinu tippu
`hpop addr`	Eemaldab pinu tipust ühe väärtuse ning salvestab selle ette antud aadressile kuhja.
`hread addr, dest`	Loeb antud aadressil kuhjas oleva väärtuse ning salvestab selle teise operandina ette antud täisarvu tüüpi muutujasse.
`hwrite addr, src`	Salvestab teise operandina ette antud täisarvulise väärtuse esimese operandina ette antud aadressile kuhja.
`hwritebit addr, src`	Salvestab teise operandina ette antud biti väärtuse esimese operandina ette antud aadressil kuhja. Biti väärtust käsitletakse modulo 2

Tabel 3: Käsud privaatse virtuaalmasina kuhjaga töötamiseks

3.2.4 Andmebaasi käsud

Privaatne virtuaalmasin võimaldab töötada lihtsustatud relatsioonilise andmebaasiga. Järgnevas tabelis 4 kirjeldame andmebaasiga töötamiseks mõeldud instruktsioonid.

Käsk	Kirjeldus
`dbload name`	Valmistutakse tööks antud nimega andmebaasiga.
`dbaddcol name`	Lisab andmebaasi ette antud nimega tabelisse uue veeru. Tühja tabeli nime puhul kasutatakse parajasti valitud tabelit.
`dbcreatetbl name, cols`	Loob andmebaasi antud nime ja veergude arvuga uue tabeli.
`dbtblexists table, result`	Kontrollib, kas avatud andmebaasis eksisteerib ette antud nimega olev tabel. Tulemus salvestatakse teise operandina antud tõeväärtuse tüüpi muutujasse.
`dbdroptbl name`	Kustutab andmebaasist antud nime kandva tabeli.
`dbusetbl name`	Valib aktiivse andmebaasi tabeli. Kui on valitud vaikimisi aktiivne tabel, siis teistes andmebaasiga seotud käskudes võib `table` operandiks panna tühja sõne. Sellisel juhul kasutatakse seal vaikimi aktiivset tabelit.
`dbcolcount colcount, table`	Tagastab andmebaasi tabeli veergude arvu. Tulemus salvestatakse esimese operandina ette antavasse täisarvu tüüpi muutujasse. Tühja tabeli nime puhul kasutatakse parajasti aktiivset tabelit.
`dbrowcount rowcount, table`	Tagastab andmebaasi tabeli ridade arvu.
`dbpushcol colnum, table`	Paneb andmebaasi tabeli veeru pinusse. Esimene operand tähistab veeru indeksit.
`dbpushrow rownum, table`	Paneb andmebaasi tabeli rea pinusse. Esimene operand tähistab rea indeksit.
`dbpushval row, col, table`	Paneb andmebaasi tabeli rea ja veeru indeksite poolt määratud väärtuse pinu tippu.
`dbsavevec vec, table`	Salvestab ette antud vektori andmebaasi tabelisse uue reana.
`dbdelrow row, table`	Kustutab andmebaasi tabelist ette antud indeksit omava rida.

Tabel 4: Privaatse virutaalmasina andmebaasiga töötavad käsud

3.2.5 Aritmeetika

Tabelis 5 kirjeldatud käsud töötavad samuti pinu andmetega, kuid nende tööülesanded on rohkem seotud aritmeetikaga.

Käsk	Kirjeldus
`add n?`	Võtab pinust kaks n-elemendilist vektorit ning liidab nende elemendid paarikaupa kokku. Tulemusvektor pannakse pinu tippu.
`sum n?`	Võtab pinust ühe n-elemendilise vektori, liidab tema elemendid kokku ning paneb saadud summa pinusse tagasi.
`mul n?`	Võtab pinust kaks n-elemendilist vektorit ning korrutab nende elemendid paarikaupa. Tulemusvektor pannakse pinusse.
`mulc c, n?`	Korrutab pinu tipus oleva n-elemendilise vektori elemendid konstandiga c
`extractbits n?`	Võtab pinust n-elemendilise vektori ning iga tema elemendi väärtuse kohta tekitab tulemusvektorisse väärtuse bitiesituse (32 elementi, iga element on üks bitt). Seejärel pannakse pinusse $32 * n$ pikkune tulemusvektor.
`convz2 n?`	Antud käsk on tähtis privaatse virtuaalmasina sisemiste protokollide jaoks. See võtab pinult n-elemendilise vektori ning konverteerib tema osakute väärtused $Z_2 \rightarrow Z_{2^{32}}$. Seejärel pannakse konverteeritud vektor tagasi pinusse.
`bitwiseadd m, n`	Võtab pinult m n-elemendilist vektori paari. Seejärel teostatakse nende bitihaaval liitmine ja tulemusvektor pannakse koos ülekandebittidega pinusse tagasi.

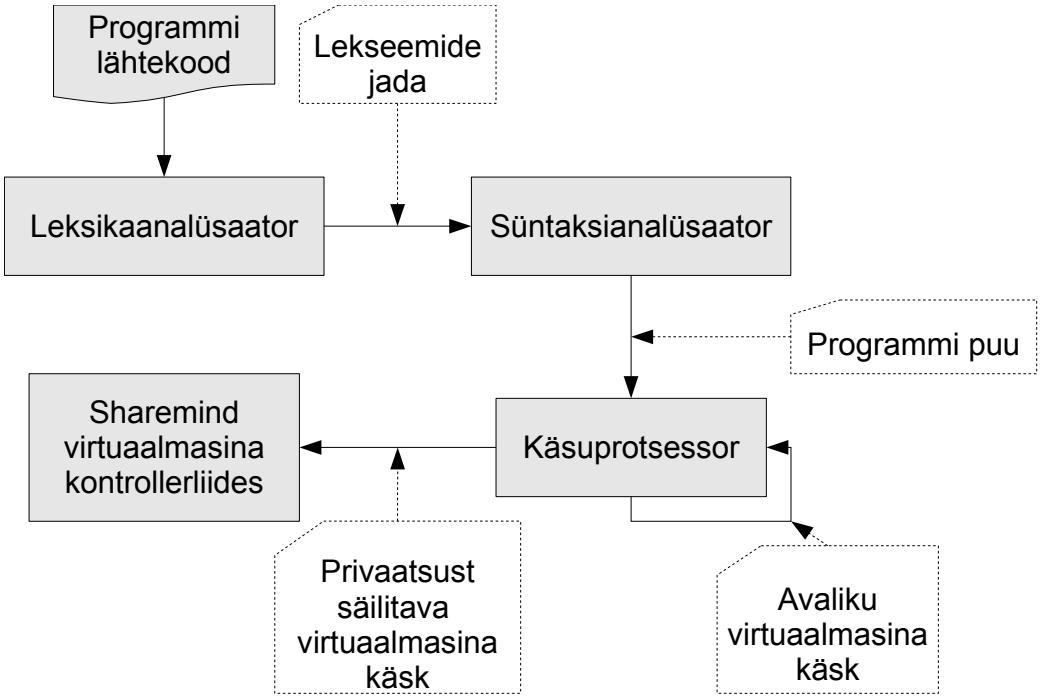
Tabel 5: Privaatse virutaalmasina pinuga töötavaid aritmeetilisi käske

3.3 Keele ja interpretaatori ehitamine

3.3.1 Interpretaator, selle struktuur ja tööpõhimõte

Intepretaator on tarkvararakendus, mis oskab lugeda ja käivitada loodavas keeles kirjutatud programme. Selleks peab interpretaator oskama programmi osi selle tekstist välja lugeda.

Lähtekoodi lugemisest käskude täitmiseni jõudmine on keerukas protsess - selleks tuleb läbida mitu etappi. Iga etapi eest vastutab interpretaatori teatud plokk, millel on olemas sisend ja tulem. Ülevaate koostatavast intepretaatorist annab joonis 1.



Joonis 1: Interpretaatori struktuur ja tööpõhimõte

Esimene suurem plokk on leksikaanalüsaator. Selle sisendiks on Sharemindi keeles kirjutatud programmi lähtekood. Leksikaanalüsaatori ülesandeks on leida lähtekoodis lekseemid, kõige väiksemad iseseisva tähendusega lähtekoodi tekstijupid: kommentaarid, sõned, toetatavate andmete tekstilised esitused, lähtekeele võtmesõnad, identifikaatorid, operaatorid, eraldajad. Samuti eemaldatakse ülearused tühikud, mis lubab lähtekoodi kokku suruda ja seega kiirendada järgnevaid etappe. Leksikaanalüsaatori tulemusena muutub lähtekoodi sümbolite jada lekseemide jadaks.

Järgmine plokk on süntaksianalüsaator, mille sisendiks on eelmise ploki väljund ehk lekseemide jada. Antud ploki ülesandeks on lekseemide jada programmikäskudeks ja avaldisteks teisendamine ning nendest programmi puu ehitamine.

Kolmas on käsuprotsessori plokk — see kujutab endast avalikku virtuaalmasinat, mille sisendiks on süntaksianalüsaatorilt saadud programmi puu. Seda interpreteeritakse ja täidetakse käskhaaval. Programmi puu koosneb instruktsioonide jadast ning nende operandidest. Viimased väärtustatakse jooksvalt programmi täitmise ajal. Käsuprotsessori ülesannete hulka kuuluvad ka muutujate ja märgendite haldus, tüübi- ja veakontroll, avaldiste konsistentsuse ja legaalsuse kontroll ning nende väärtustamine. Käsuprotsessoris hallatavad andmed on avalikud andmed, mis on kõigile nähtavad. Seega nimetame käsuprotsessorit ka avalikuks virtuaalmasinaks.

Neljas plokk on Sharemindi virtuaalmasina kontrollerliides, mille kaudu saadetakse privaatsete andmetega töötavad käsud privaatsele virtuaalmasinale. Viimases töödeldavad andmed on hästi kaitstud — seal tehtavad operatsioonid teostatakse privaatsust säilitaval viisil sisuliselt reaalseid andmeid teadmata.

Kaks esimest plokki moodustavad keele translaatori ning kaks viimast plokki moodustavad loogikaprotsessori, kus lähtekoodist transleeritud programmi täidetakse. Kuna interpretaator peab programmi täitmiseks läbima mitu plokki ning iga neist peidab endas suurt hulka loogikat, siis mõistagi on interpreteeritud programmid kompileeritud programmidest aeglasemad — neis tehtakse palju jooksvaid teisendusi selle asemel, et käske otse masinkoodis täita. Küll aga pole antud keele interpretaatori kiirus kõige suurem prioriteet, sest käskude täitmisel on privaatne virtuaalmasin oma ehituse tõttu avalikust virtuaalmasinast oluliselt aeglasem ning interpretaatori kiirus mõjutab kogukiirust vähe.

3.3.2 Leksikaanalüsaator

Leksikaanalüsaatori ehk skanneri genereerimiseks kasutati antud töös `Flex`-nimelist vahendit (*Fast Lexical Analyzer*). `Flex` teisendab talle antud translaatori leksikat sisaldava faili `Scanner.ll` C või C++ programmiks, mida hiljem kasutatakse interpretaatori koodi osana.

Failis `Scanner.ll` toodud leksikaanalüsaatori kirjeldus koosneb kolmest üksteisest `%%` sümbolitega eraldatud osast: `definitsioonid`, `reeglid` ja `kasutaja funktsioonid`.

`Flex`i abil kirjeldatud skanner on tegelikult lihtsalt sisendteksti alamlõikude teisendaja: ta asendab `definitsioonid`-osas kirjeldatud regulaarsetele avaldistele vastavad lõigud skaneeritavas programmis (skanneri sisendis) vastavalt osas `reeglid` esitatud reeglitele. Sisendi need osad, mis ei vasta ühelegi seal kirjeldatud regulaarsele avaldisele, kopeeritakse väljundisse muutumatult.

Asendatud lõigud nimetame lekseemideks — need on kõige väiksemad iseseisva tähendusega lähtekoodi tekstitükid. Lekseemide kirjeldamiseks kasutatakse terminale ja mitteterminale. Terminalid on defineeritavates jadades kasutatavad märgid (näiteks täisarvude defineerimisel numbrid $0, 1, 2, ..., 9$). Mitteterminalid on definitsiooni käigus kasutatavad abimõisted (näiteks `NUMBER`, `INT`).

Osas `definitsioonid` defineeritud lekseemide kirjeldustes kasutatavate mitteterminalide regulaaravaldised on toodud tabelis 6. Regulaaravaldisi kasutatakse lekseemide äratundmisel.

Mitteterminal	Regulaaravaldis
CHARACTER	[a-zA-Z_]
DIGIT	[0-9]
NUMBER	{DIGIT}+
IDENT	{CHARACTER}({CHARACTER}|{DIGIT})*
STRING	\"[^"\n]*["\n]
SINGLE_LINE_COMMENT	\/\/.*
MULTIPLE_LINE_COMMENT	\/*([^*]|[\r\n]|(*+([^*/]|[\r\n])))**+\/
INTVECTOR	\{([0-9]+,)*([0-9]+)*\}

Tabel 6: Lekseemide kirjeldustes kasutatavad mitteterminalid

Lekseemi definitsioon võib põhineda ka eelnevalt defineeritud lekseemide definitsioonidel. Näiteks koosneb identifikaatori lekseem `IDENT` varem defineeritud `CHARACTER` ja `DIGIT` lekseemide kombinatsioonist. Samuti võib lekseeme kirjeldavaid regulaaravaldisi kasutada otse `reeglid`-osas — suurem hulk lekseeme ongi antud keele leksikaanalüsaatori puhul sellisel viisil kirjeldatud. Kõik loodava keele leksikaanalüsaatori poolt otsitavad ja tagastatavad avaliku virtuaalmasinaga seotud lekseemid on toodud tabelis 8.

Otsingu regulaaravaldis	Tagastatav lekseem	Väärtus, kui olemas
{SINGLE_LINE_COMMENT}	ei tagastata	
{MULTIPLE_LINE_COMMENT}	ei tagastata	
{STRING}	STRING	jutumärkidevaheline sõne
true	BOOLEAN	true
false	BOOLEAN	false
{NUMBER}	INTEGER	arv
{INTVECTOR}	INTVECTOR	täisarvuline vektor
int	T_INTEGER	
int\[\]	T_INTVECTOR	
bool	T_BOOLEAN	
string	T_STRING	
mov	MOV	
input	INPUT	
print	PRINT	
println	PRINTLN	
cmp	CMP	
jt	JT	
jf	JF	
jmp	JMP	
inc	INC	
random	RANDOM	
vecinsert	VECINSERT	
vecremove	VECREMOVE	
vecsize	VECSIZE	
vecclear	VECCLEAR	
{IDENT}	IDENT	sõne
[\t]+	ei tagastata	
[\n\r]	EOL	
\<=	LE	
\>=	GE	
==	EQ	
!=	NE	
&&	AND	
||	OR	
.	muutumatul kujul	

Tabel 8: Kõik otsitavad ja tagastatavad avaliku virtuaalmasinaga seotud lekseemid

Lähtetekstis leitud kommentaarid eemaldatakse, nagu ka üleliigsed tühikud. Sõned, identifikaatorid, ja muud lekseemid edastatakse süntaksianalüsaatorile.

3.3.3 Süntaksianalüsaator

Süntaksianalüsaator rakendab talle edastatud lekseemide jadale grammatikareegleid, teisendab selle programmikäskudeks ja avaldisteks ning ehitab siis viimastest keele

grammatikale vastava programmi puu.

Käesolevas töös kasutati süntaksianalüsaatori genereerimiseks programmi `Bison`. `Bison`i sisendiks on fail `Parser.yy`, mis sisaldab transleeritava programmeerimiskeele süntaksit kirjeldavat kontekstivaba grammatikat. Ka sellest failist genereeritakse hiljem intepretaatori osana kasutatav C++ klass - süntaksianalüsaator. Grammatika sisaldab reegleid, terminale ja mitteterminale.

`Bison`ile esitatava grammatika terminalid on leksikaanalüsaatori poolt leitud lekseemid. Need esitatakse `Bison`ile võtmesõna `%token` järel ning kirjutatakse suurte tähtedega. Terminalidena võib kasutada ka tekstikonstante, näiteks `'+'` või `'*'`. Grammatika mitteterminalid kirjutatakse väiketähtedega. Neid eraldi ei loeta, `Bison` tunneb nad ära grammatikareeglitest.

Programmilõikude semantika saab leida ainult selle programmilõigu struktuuri abil. See struktuur kirjeldatakse grammatikaga ja programmilõigu tähenduse esitab selle programmilõigu süntaksipuu ehk derivatsioonipuu, mille põhjal see programmilõik grammatika algussümbolist saadakse. Seega peab programmi üheseltmõistetavuse tagamiseks ühele grammatikale vastata üks derivatsioonipuu - grammatika peab olema ühene.

`Flex` ja `Bison` on kaks eraldi vahendit, mis genereerivad C++ koodi, mistõttu on tarvis seda koodi kasutades leksika- ja süntaksianalüsaatorit koostööks seadistada. Selleks on loodud `Flex`i ja `Bison`i genereeritud koodi kapseldav klass `Driver`, mis loob liidese analüsaatorite lihtsaks kasutamiseks. Samuti on `Driver` klassi kapseldatud veateadete edastamise eest vastutav klass ning keelekonteksti klass, kuhu süntaksianalüsaatori poolt ehitatakse programmi puu.

Programmi puu ehitamise põhimõte on lühidalt kokku võttes järgmine: iga instruktsiooni puhul ehitatakse vajalike avaldiste puud ning lisatakse need vastava instruktsiooni operandide vektorisse. Seejärel lisatakse valmis instruktsioon keelekontekstis olevasse käskude vektorisse. Tsüklit korratakse kuni kõik programmi käsud on puusse lisatud. Avaldiste ja instruktsioonide puude struktuure vaatleme lähemalt järgnevates peatükkides.

3.3.4 Andmed

Konstandid. Pärast kommentaaride eemaldamist filtreerib leksikaanalüsaator välja erinevat tüüpi andmed: sõned, tõeväärtused, arvud ja vektorite tekstilised esitused.

Tabelis 8 nägime, kuidas viia otsitava tekstiga vastavusse osa lekseemidest. Konstantsete andmete puhul (sõne, tõeväärtus, arv, täisarvuline vektor või identifikaator) tagastab leksikaanalüsaator vastava lekseemi ning edastab lekseemile vastava väärtuse. Väärtuse tüübi määrab lekseem: kui lekseem oli `INTEGER`, siis edastatakse leitud täisarv. Kui aga lekseemiks oli vektori tekstiline esitus, siis edastatakse koos lekseemiga tekstilisest esitusest rekonstrueeritud vektori sisemise vormingu versioon.

Seejärel saab süntaksianalüsaator sisendiks lekseemide jada ning leiab, et lekseemid

INTEGER, BOOLEAN, STRING, INTVECTOR või kooloniga algav märgendit kujutav identifikaator rahuldavad mitteterminaliga constant algavat reeglit. Seega iga variandi puhul konstrueeritakse vastava tüübi ja väärtusega avaldis, mille aadress seotakse mitteterminali constant uue instantsiga. Pärast seda saab seda mitteterminali kasutada näiteks avaldise grammatikareeglis.

Muutujad. Muutujad leitakse süntaksianalüsaatori poolt lekseemi IDENT järgi. Nagu ka konstantide puhul, seotakse muutujate puhul mitteterminaliga variable muutujat kujutava avaldise aadress. Avaldis sisaldab muutuja nime, milleks on lekseemiga IDENT edastatud sõne tüüpi väärtus, ning spetsiaalset tehtekoodi, mis tähistab, et avaldis on muutuja. Mitteterminali variable saab seejärel kasutada teistes grammatikareeglites.

Muutujate deklareerimisel on vaja neile anda tüübid. Mitteterminali variable_type väärtuseks pannakse andmetüüp vastavalt leksikaanalüsaatorist tulnud lekseemi, milleks võib olla T_INTEGER, T_INTVECTOR, T_BOOLEAN või T_STRING. Nüüd saab luua grammatikareegli, mis vastab muutuja deklareerimise süntaksile:

```
i_variable_declaration : variable_type IDENT {...}
```

See reegel läheb kasutusse, kui skaneeritavas programmikoodis leidub paar, kus esimene lekseem tähistab andmetüüpi ja teine on identifikaator, näiteks: int a.

Vektori elemendid. Konstantide ja muutujate abil saab töötada tervete vektoritega. Tihti on aga tarvis pääseda ligi vektori elementidele. Selleks peab keeles olema kindel süntaks. Loodavas keeles kasutatakse selleks C ja Java keelest tuntud nurksulge: vektori_nimi[indeks]. Sellise süntaksi lubamiseks on keele grammatikas defineeritud järgmine reegel:

```
vector_element : variable '[' exp ']'
{
vector<Expression> ve;
ve.push_back(*$3);
$$ = new Expression(INT, _VARIABLE, ve);
$$->name = $1->name;
}
```

See reegel ütleb, et kui lekseemide jadas leidub reegli paremal pool toodud alamjada, siis mitteterminali vector_element väärtuseks pannakse sellise muutujat kujutava avaldise aadress, millel on täisarvuline tüüp, variable muutujaga sarnane nimi ning millel on ka üks argument, milleks on indeksit tähistav avaldis. Interpretaator

oskab selle info järgi hiljem õigest vektorist õiget elementi üles leida. Mitteterminali `vector_element` saab nüüd teistes reeglites kasutada ning see tähistab seal mingi vektori mingit elementi.

3.3.5 Avaldised

Loodavas keelel on avaldiste tugi. Toetatakse triviaalseid ja mittetriviaalseid avaldisi ning nende realiseerimiseks on loodud alamavaldist kirjeldav klass. Klassi struktuur ilma funktsioonideta on järgmine:

```
class Expression
{
string name;
expression_types type;
exp_value_t value;
exp_op_codes op;
vector<Expression> params;
};
```

Avaldise struktuurist on näha, et sellel on nimi, tüüp, väärtus, tehtekood ja parameetrid. Nimi on tavaline sõne. Tüübiks võib olla üks viiest väärtusest: `INT`, `INTV`, `BOOL`, `STRING`, `LABEL`. Avaldise väärtus on liittüüp, mis võib olla kas tõeväärtus, 32-bitine täisarv, või viit sõnele, täisarvulisele vektorile või täisarvule mingis vektoris. Tehtekood `op` võib osutada sellele, kas avaldis on konstant, muutuja või vektori element või on ta mõni matemaatiline, loogiline või sõnetehe. Väli `params` võib sisaldada teisi avaldisi, mis lubab ehitada keeruliste avaldiste puid. Sellese välja lisatakse avaldisi siis, kui tegemist on tehet kujutava avaldisega või vektori elemendiga, mil üheks parameetriks on vektori elemendile osutav indeks.

Meil on struktuur, mis võimaldab koostada keerulisi avaldisi. Avaldisi on aga vaja kuidagi programmi lähtekoodist välja lugeda. Selleks kasutame eelnevalt käsitletud konstantide, muutujate ja vektori elementide jaoks loodud grammatikareegleid ning lisaks mingit hulka lekseemidest. Saame defineerida uue grammatikareegli, mis sätestab avaldiste ehitamise loogika:

```
exp : constant { $$ = $1; }
| vector_element { $$ = $1; }
| variable { $$ = $1; }
| exp '+' exp { ... }
| exp '-' exp { ... }
```

```
| exp '*' exp { ... }
| exp '/' exp { ... }
| exp '^' exp { ... }
| exp '%' exp { ... }
| '-' exp %prec UMINUS { ... }
| exp LE exp { ... }
| exp GE exp { ... }
| exp '<' exp { ... }
| exp '>' exp { ... }
| exp EQ exp { ... }
| exp NE exp { ... }
| exp AND exp { ... }
| exp OR exp { ... }
| '(' exp ')' { $$ = $2; }
| '!' exp { ... }
```

Antud grammatikast näeme, et avaldiseks loetakse loodava keele süntaksianalüsaatori poolt kas konstanti, vektori elementi, muutujat, mõnda tehet avaldistega või sulgude sees olevat teist avaldist. Kui tegemist on juba valmis avaldisega, siis kasutatakse selle viita, vastasel juhul luuakse uus avaldis, vajadusel lisatakse sellele argumendid ning kasutatakse loodud avaldise viita.

Antud grammatika implementatsioonis ei tehta vahet andmetüüpidel — ehitatakse kokku ka mittekokkusobivatest avaldistest koosnevaid avaldisi. Andmetüüpe kontrollitakse hiljem käsuinterpretaatori poolt.

3.3.6 Instruktsioonid

Programm koosneb instruktsioonidest ehk käskudest. Kuna käske täidetakse üksteise järel, siis on tarvis hoida need mingis kindlat tüüpi järjekorras või jadas. Selleks loome käsu abstratsiooni - individuaalset käsku kirjeldava klassi:

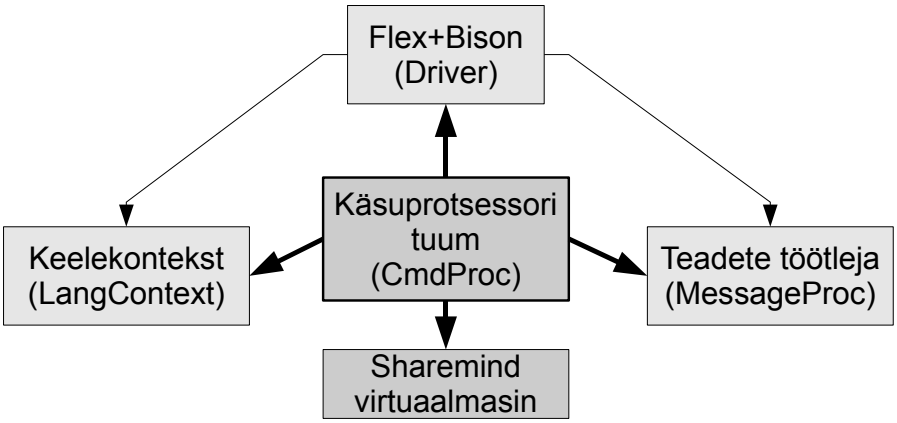
```
class Instruction
{
op_codes op_code;
vector<Expression> op_params;
string location;
}
```

Instruktsiooni klass sisaldab käsukoodi `op_code`, käsu operandide vektorit `op_params`

ja süntaksianalüsaatori poolt täidetavat ning käsu asukohta programmi lähtekoodis kirjeldavat sõne `location`. Käsukood ütleb interpretaatorile, millist käsku tuleb täita. Igal instruktsioonil on oma kindel kood. Operandide vektoris hoitakse vastava käsu jaoks vajalikke operande. Kuna operandideks olevad avaldised sisaldavad infot enda tüüpide kohta, siis saavad instruktsioonid enne operandide kasutuselevõttu tüüpe kontrollida ning mittesobivuse korral teatada interpretaatorile programmi veast. Käsu asukohta kasutatakse veateadete väljastamisel juhtimaks kasutaja tähelepanu vastavale kohale programmi lähtetekstis.

3.3.7 Käsuprotsessor

Käsuprotsessor on olemuselt virtuaalmasin, mis ühendab see endas mitut alamplokki. Ülevaate käsuprotsessori ehitusest annab joonis 2.



Joonis 2: Käsuprotsessori ehitus

Käsuprotsessori tuum. Kõige tähtsam plokk on käsuprotsessori tuum, mille realiseerib klass `CmdProc`. Selle sees toimub avaldiste väärtustamine, käskude käivitamine ning vea- ja tüübikontroll. Samuti annab see klass lihtsa liidese käskude sõnest või failist lugemiseks ning käivitamiseks. Tuumas luuakse klasside `LangContext`, `Driver`, `MessageProc` ja Sharemindi virtuaalmasina instantsid ning juhitakse nende tööd.

Antud plokis on realiseeritud kõik avaliku virtuaalmasina instruktsioonid. Nende täitmisega tegeleb otseselt funktsioon `_ExecuteInstruction(Instruction* i)`, mis enne iga käsu käivitamist kontrollib käsu operandide arvu ning käivitab operandide evaluaatori. Kui tegemist on privaatsust säilitava instruktsiooniga, siis kasutatakse nende täitmisel Sharemindi virtuaalmasinat. Funktsioonid `Execute()` ja `Execute(string& line)` täidavad programmi käske. Esimene neist käivitab failist laetud programmi ning teine täidab talle ette antud sõnest loetud käsud.

Evaluaator `Expression* CmdProc::_Eval(Expression* exp)` kontrollib, et talle ette antud avaldised oleksid legaalsed ning arvutab nende väärtusi. Kui tegemist on muutuja või vektori elemendiga, siis otsib ta selle keelekontekstist üles. Samuti on

evaluaatorisse sisse programmeeritud kõik tehted avaldistega. Kui avaldist ei õnnestu väärtustada, tagastab evaluaator tühja viite, mille peale programmi töö lõpetatakse. Õnnestunud väärtustamisel aga tagastatakse viit ette antud avaldisele, millele on lisatud tema väärtus.

Muud plokid. `Flexi` ja `Bisoni` poolt genereeritud leksika- ja süntaksianalüsaatoreid kapseldav klass `Driver` annab liidese nende juhtimiseks. See võtab vastu kas sõne või faili nime, millest programmi lugeda ning käseb analüsaatoritel vastavalt sõne või faili sisu töödelda ning vigade puudumisel programmi puu ehitada.

Programmi puu ehitatakse keelekonteksti klassi `LangContext`. Selles klassis hoitakse ka täitmisaegset infot muutujate, märgendite, vektori elementide ja võrdluskäskude tulemuste kohta. Nii analüsaatorid kui käsuprotsessori tuum aktiivselt kasutavad seda klassi andmete töötlemisel.

Teadete töötleja klassis töötletakse ja väljastatakse ekraanile teateid. Klassis on kõik vajalikud meetmed teadete konstrueerimiseks. Samuti eristatakse siin erineva tasemega (silumiseks või kasutamiseks) tavalisi ja veateateid. Antud klassi kasutavad nii analüsaatorid kui käsuprotsessori tuum.

3.3.8 Vookontroll

Programmi kulgu kontrollitakse instruktsiooniloenduri abil. Instruktsiooni täites pannakse loenduri väärtuseks järgmise instruktsiooni järjekorranumbri. Kuni käsuloenduri väärtus on alla instruktsioonide koguarvu, võetakse alati järgmine käsk. Vastasel juhul loetakse programm lõpetatuks.

Selline süsteem võimaldab luua käske, mis panevad käsuloenduri väärtuseks järgmisest erineva käsu järjekorranumbri. Sellisteks käskudeks on vookontrolli jaoks mõeldud hüppekäsud `jt`, `jf` ja `jmp`. Need käsud võtavad argumendiks märgendi nime sisaldava avaldise ning leiavad keelekontekstist selle järgi märgendile vastava instruktsiooni järjekorranumbri. Viimane saab ka käsuloenduri uueks väärtuseks.

Kuna käsud `jt` ja `jf` teostavad tingimuslikku hüpet, siis peavad nad millegi järgi otsustama, kas hüpata või mitte. Enne neid instruktsioone tuleb käsu `cmp` abil väärtustada loogiline avaldis, mille järel salvestab käsk avaldise väärtuse keelekontekstis olevasse võrdlustulemuste pinusse. Sellest pinust saavadki käsud `jt` ja `jf` väärtuse, mille järgi hüppe otsus langetada.

4 Tulemused

Antud töö raames sai valmis programmeerimiskeel privaatsust säilitavate rakenduste loomiseks ning interpretaator selles programmeeritud rakenduste käivitamiseks. Loodud on lihtne liides käskude lugemiseks ning täitmiseks. Liides on realiseeritud klassi `CmdProc` näol ning seda on võimalik integreerida teise rakenduse sisse lisades sellega sinna keele ja interpretaatori funktsionaalsuse. Liides pakub meetmeid programmide otse failist lugemiseks ja käivitamiseks või ükshaaval sõne kujul ette antavate käskude täitmiseks.

Programmeerimiskeele ja interpretaatori struktuur on põhjalikult kirjeldatud käesolevas töös. Programmeerimiskeel on lihtsasti omandatav, ent sellele kaasa aitamiseks on peatükkides 3.1 ja 3.2 detailselt kirjeldatud kõik keele võimalused ja instruktsioonid.

Antud töö raames on loodud keeles kirjutatud kaks suuremat näidisprogrammi demonstreerimaks mõningaid keele võimalusi. Nendeks programmideks on lisades 1 ja 2 tekstilisel ning lisas 3 digitaalsel kujul toodud `SimpleDataEntry` ja `SimpleDataAnalysis`. Esimene neist küsib kasutajalt täisarve ning salvestab nad privaatse virtuaalmasina relatsioonilisse andmebaasi. Teine aga küsib kasutajalt täisarve ning turvalisi protokolle kasutades saab privaatselt virtuaalmasinalt vastuseks, mitu korda küsitud täisarvud andmebaasis esinevad. Virtuaalmasin andmebaasi sisestatud täisarvude väärtusi ei näe, samas oskab ta turvaliselt nendega operatsioone teostada.

5 Kokkuvõte

Käesoleva töö raames sai valmis assemblerilaadne, ent kõrgematest keeltest tuttavaid avaldisi toetav keel ja selle interpretaator. Keel võimaldab luua privaatsust säilitavaid arvutusi teostavaid rakendusi, ilma et programmeerija peaks tegelema arvutuste turvalisuse või nende käigus esineda võivate vigadega. Turvalisuse tagab privaatne vir-tuaal-masin Sharemind ja veakontroll ning avalike andmetega opereerimine teostatakse interpretaatori osana loodud avaliku virtuaalmasina poolt.

Keeles kirjutatud programme saab käivitada vastava interpretaatorrakendusega. Lisaks on programme võimalik käivitada teiste programmide sees, kui kasutada interpretaatori vastavaid komponente.

Töös kirjeldatakse privaatsust säilitava virtuaalmasina kasutamist interpretaatori loomisel. Samuti antakse põhjalik ülevaade keele konstruktsioonidest, interpretaatori ehitusest ja tööpõhimõtetest.

Loodud interpretaatori lähtekood koos näidisprogrammidega on toodud eraldi digitaalsel andmekandjal, kus see on esitatud Sharemind projekti osana.

6 A programming language for creating privacy-preserving applications.

Work: Bachelor's thesis.

Author: Roman Jagomägis

Resume

The preservation of privacy in data processing is crucial, especially in such fields as medicine, finance or social studies. It is often necessary to gather sensitive data, that belongs to other people or organisations and perform some analysis on it. However, this leads to serious security issues, as organisations who collect and process the data may abuse it or reveal the data to the third parties. This will compromise the privacy of the data and also the people it came from. It also slows down the progress of our society since the analysis of its activity are complicated.

There exist solutions for secure data analysis, but those are usually not flexible enough or hard to utilise and adapt to requirements. Therefore it is necessary to find easier ways for creating privacy preserving applications based on some of those solutions.

In this bachelor's thesis we create a programming language for developing privacy preserving applications. We also build an interpreter for that language. The interpreter is based on privacy preserving framework called Sharemind. The framework works as a private virtual machine that provides secure computation functionality. Another part of the interpreter acts as a public virtual machine that allows one to perform public data computations and hides all the error checking during those computations. This simplifies and quickens the secure application programming process.

This work includes a brief introduction of the Sharemind framework, an in-depth description of the created programming language, its instruction set and the interpreter. The source code of the interpreter is included on a separate digital carrier as part of Sharemind framework. Some of test applications are included as well to illustrate the methods of programming of practical privacy preserving applications that can be run using our interpreter.

7 Kasutatud kirjandus

[1] ”How to securely perform computations on secret-shared data”, magistritöö, Dan Bogdanov, Tartu Ülikool, 2007

[2] ”Formaalsed keeled, grammatikad ja translaatorid”, Jaak Henno, 2006

[3] ”Lex & Yacc”, John R. Levine, Tony Mason, Doug Brown, 1992

[4] ”Programmeerimiskeeled”, Ain Isotamm, 2007

Lisa 1. SimpleDataEntry

```
string      dbName
string      tableName
string      in
int         n
int[]       V
int[]       allValues
int         i
bool        tableIsThere
mov         dbName, "SimpleDataTestDB"
mov         tableName, "numbers"
mov         tableIsThere, false
dbload      dbName
dbtblexists  tableName, tableIsThere
cmp         !tableIsThere
jt          :go
dbdroptbl   tableName
go:
dbcreatetbl tableName, 1
dbusetbl    tableName
println     "Please enter some integers to save to database!"
println     "Empty input to stop entering integers!"
readvalue:
  print     "Enter integer #"+(n+1)+": "
  input     in
  cmp       in == ""
  jt        :printvalues
  vecclear   V
  vecinsert  V, in
  vecinsert  allValues, in
  dbsavevec  V, ""
  inc        n
  jmp        :readvalue
printvalues:
  println    "You entered " + n + " values!"
printvalue:
  cmp        i == n
  jt         :finalstep
```

```
  println    "Integer #" + (i+1) + " is " + allValues[i]
  inc        i
  jmp        :printvalue
finalstep:
  print      "Values saved to database '" + dbName + "'"
  println    " table '" + tableName + "'"
```

Lisa 2. SimpleDataAnalysis

```
string     dbName
string     tableName
string     query
int[]      querypar
int        i
int        size
int[]      res
int        tmp
mov        dbName, "SimpleDataTestDB"
mov        tableName, "numbers"
dbload     dbName
dbusetbl   tableName
dbpushcol  1, ""
askinteger:
  print     "Please enter some integer: "
  input     query
  cmp       query == ""
  jt        :end
  stacksize size
  dup       size
  mov       i, 0
  tsykk1:
    vecinsert   querypar, query
    inc         i
    cmp         i<size
    jt          :tsykk1
  push       querypar
  dup        2*size
  swap       size
  pushgteres 2*size
  pusheqres  size
  sum        size
  pop        res
  println    "" + query + " occurs in database " + res[0] + " times"
  vecclear   querypar
  jmp       :askinteger
end:
```

Lisa 3. Interpretaatori lähtekood digitaalsel andmekandjal