TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Evaldas Petnjunas

GobExec: programmianalüsaatorite hindamisraamistik

Bakalaureusetöö (9 EAP)

Juhendaja: Simmo Saan

Tartu 2024

GobExec: programmianalüsaatorite hindamisraamistik

Lühikokkuvõte:

Goblint on staatilise programmianalüüsi tööriist, mis kasutab oma analüüside usaldusväärsuse ja tõhususe tagamiseks hindamisprogrammide komplekti. Praegune lahendus hindamiseks tugineb Ruby skriptidele, mis on muutunud hooldamatuks ja keeruliseks laiendada. See lõputöö kirjeldab ühtset jõudlusetestimise raamistikku programmianalüüsi tööriistade jaoks Python programmeerimiskeeles. Raamistik pakub vähemalt sama funktsionaalsust kui vana lahendus ning võimaldab vanade hindamisprogrammide teisendamist ja asendamist. Uus raamistik muudab jõudlusetestimise komplekti funktsionaalsuse laiendamise ja hooldamise lihtsamaks, kuna raamistiku koodi jagatakse enamasti erinevate moodulite vahel. Lahendust valideeriti esmalt vanade skriptide teisendamisega Python programmeerimiskeelde kasutades uut raamistikku, ja seejärel võrreldi neid skripte vanade skriptidega. Porditud skriptid on palju lihtsamad, kuid säilitavad sama funktsionaalsuse ja hindamisprogrammide tulemused.

Võtmesõnad:

Staatiline analüüs, hindamisraamistik, hindamisprogramm

CERCS:

P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

GobExec: benchmarking framework for program analysis tools

Abstract:

Goblint is a static program analysis tool which uses a benchmarking suite for ensuring soundness and effectiveness of its analyses. The current solution for benchmarking relies on a set of Ruby scripts which have become unmaintainable and difficult to extend. This thesis describes a unified benchmarking framework for program analysis tools in Python programming language. The framework provides at least the same functionality as the old solution for benchmarking, while making the porting and replacing of old benchmarking scripts to Python possible. The new framework makes extending and maintaining the functionality of the benchmarking suite easier, as most of the framework's code is shared across different modules. The solution was validated by first porting old scripts to Python using the new framework and then comparing them to scripts from the old benchmarking suite. The ported scripts are much simpler but retain the same functionality and benchmarking results.

Keywords:

Static analysis, benchmark, benchmarking framework

CERCS:

P170 Computer science, numerical analysis, systems, control

Sisukord

Sissejuhatus

Kirjutades koodi tänapäevastes koodiredaktorites (nt. IntelliJ, PyCharm) paljud vead, mis on koodi kirjutamisel tekkinud, näidatakse kasutajale. See on võimalik tänu staatilisele analüüsile. Iseeneses, staatiline analüüs on lähtekoodi analüüsimise metoodika, mis võimaldab tuvastada koodi vigu, haavatavusi ja potentsiaalseid probleeme ning annavad ka soovitusi nende parandamiseks. Selleks et rakendada seda meetodit kirjutatud koodi peal on olemas ka staatilise analüüsi tööriistad, mis toetavad selliseid keeli nagu Python, Go, Ruby, C jne.

Selleks et olla kindel staatilise analüüsi tööriista korrektsuses ja efektiivsuses, need tööriistad peavad olema katsetatud ja hinnatud. Selleks luuakse hindamisprogramme ehk *benchmark*'e. *Benchmark* on standardiseeritud test, mida kasutatakse tarkvara võimsuse ja jõudluse hindamisel. Omakorda hindamisraamistik on tööriist või nende komplekt, mis võimaldab hindamisprogrammide automaatset käivitamist, mõõtmist ja tulemuste kuvamist tarkvara eri komponentide jõudluse hindamiseks ning võrdlemiseks.

Goblint on staatilise analüüsi tööriist, mis võimaldab staatilist analüüsi C keeles kirjutatud programmides ja mille põhifunktsioon on paralleelsusega seotud vigade tuvastamine [7]. Käesoleva töö eesmärk on luua hindamisraamistik selle tööriista jaoks Python keeles, mis põhineb juba olemasolevatel Ruby keeles kirjutatud skriptidel, ja anda kasutajale võimalus luua oma testimisprogramme kasutades antud töö käigus loodud hindamisraamistiku vahendeid.

1. Mõisted ja terminid

Hindamisprogramm (ingl. *benchmark*) on tarkvara jõudluse mõõtmise meetod. *Benchmark*'e kasutatakse kitsaskohtade leidmiseks, erinevate implementatsioonide võrdlemisel ja ka programmide või koodi optimeerimiseks.

Raamistik (ingl. *framework*) on eelnevalt kirjutatud funktsionaalsuse ja tööriistade kogum mis annab kasutajale malli tarkvara loomiseks.

HTML kuvamine ehk HTML renderdamine **(**ingl. *HTML rendering***)** on HTML keeles märgistuse genereerimine, veebilehe brauseris kuvamiseks.

Hindamisprogrammide komplekt (ingl. *benchmarking suite*) on omavahel seotud hindamisprogrammide kogum.

2. Teoreetiline ülevaade

Goblint on C programmeerimiskeeles kirjutatud programmide staatiline analüsaator, mis on kirjutatud suures osas OCaml ehk objektorienteeritud Caml programmeerimiskeeles ja programmid selle hindamiseks on kirjutatud Ruby programmeerimiskeeles. Ruby keeles kirjutatud hindamisprogrammide taaskasutamine on koormav, kuna need olid kirjutatud sellisel viisil, et esmalt kopeeriti terve hindamisprogramm uude faili ja ainult siis lisati soovitud funktsionaalsust või muudeti programmis esinevaid muutujaid. Olemasolevad hindamisprogrammid ei ole kuidagi struktureeritud ühte raamistikku, vaid esinevad eraldi failides. Seega soovides luua uut hindamisprogrammi kasutades olemasolevaid Ruby programmeerimiskeeles kirjutatud programme, on vaja uuesti defineerida ja ümberkirjutada juba loodud funktsionaalsust. See omakorda toob programmidesse ootamatuid vigu, näiteks hooletusest valesti ümberkirjutatud muutuja nimi või omakorda muudab hindamisprogrammi nii, et see tagastab valed tulemused. Kui aga hindamise jaoks vajalik funktsionaalsus on ühendatud ühte raamistikku, on hindamisprogrammide käivitamisskripti loomine palju efektiivsem aja ja resursside suhtes. Seega hindamisraamistiku loomiseks otsustati kasutada Python programmeerimiskeelt, kuna see võimaldab ühe programmi funktsionaalsuse taaskasutamist importimise kaudu. See võimaldab teistel raamistiku kasutajatel luua oma hindamisprogramme, kasutades selle töö raames loodud hindamisprogramme ja nende funktsionaalsust. Selle funktsionaalsuse näiteks, peale oma hindamisprogrammide loomise, on võimalus modifitseerida olemasolevaid hindamisprogramme nii, et need vastaks kasutaja nõuetele, mis on omakorda suur pooltargument Python programmeerimiskeele kasutamiseks. Teine pooltargument Python'i kasutamiseks oleks selle programmeerimiskeele populaarsus, mis omakorda tähendab, et palju inimesi on selle süntaksiga tuttavad ja võivad kirjutada oma hindamisprogramme kasutades raamistikus oleva funktsioonalsust. Ruby programmeerimiskeele õppimiskõver on suur ja seega nõuab eelnevat kogemust selle programmeerimiskeelega, mis teeb hindamisprogrammide loomise kasutaja jaoks raskeks.

M. Gerrard jt [1] on kirjutanud, et programmi analüüsi põhimõttes seisab tingimuslik tõendamine, mis välistab olekuruumist need osad, mis on juba tõendusprotseduuriga kontrollitud, selleks et keskenduda nende olekuruumi osadele, mis veel tõendusprotseduuriga kontrollitud ei ole, kuid aga ei saa üht tõendusprotseduuri kasutada kõikide olekuruumi osade kontrollimiseks. Allika järgi on sellel ka üks negatiivne aspekt - kuna protseduurid on loodud keeruliste vigade leidmiseks, mõnikord ei leia need lihtsamaid vigu.

Analüsaatoris kasutatavaid tõendusprotseduure hindavad hindamisprogrammid ehk *benchmark'id,* milliseid on peamiselt kolme erinevat liiki: süntaksi kontrollija, mis kontrollib programmi süntaktilisi vigu ja kaks liiki semantilist kontrollijat: üks tagastab vigu, mis võivad tekkida programmi käivitamisel, sellist kontrollijat nimetatakse *unsound semantic analyzer.* Teine semantiline kontrollija kasutab rangeid matemaatilisi viisi vigade puudumise määramiseks nii, et kõik vead oleks raporteeritud ehk et ükski viga ei jääks vahele. Selleks, et need kontrollijad töötaks õigesti, olid Jörg Herter jt [2] loonud juhiseid hindamisprogrammi koostamiseks ja disainimiseks :

1. Testjuhtumid peavad olema ilma tahtmatu vigadeta ehk testjuhtum peab keskenduma ühe vea tuvastamisele ja ei tohi koosneda mitmest vigadest. Samuti peavad kasutatavad vead olema tõhusalt dokumenteeritud.
2. Vigade tüübid peavad olema kaalutud, kuna vead erinevad tõsisuse raames. Näiteks loogiline viga on tõsisem kui halb programmeerimisstiil.
3. Vigade tüübid peavad olema üldtunnustatud. Selleks võib kasutada MISRA C juhiseid [3], mis on C programmeerimiskeele jaoks loodud juhiste komplekt, mille eesmärk on tagada koodi ohutust, turvalisust ja töökindlust.
4. Testjuhtumid ei pea tuginema sobimatule koodile. Sobimatu koodi näited on samuti toodud MISRA C juhistes ja on firma valdkonna spetsiifilised.
5. Kõik oletused peavad olema dokumenteeritud. Testjuhtumid ei või kasutada muutujaid või konstante, tüüpe mis ei ole defineeritud ja dokumenteeritud. Samuti peavad olema dokumenteeritud kasutatud teekide funktsioonid
6. Testjuhtumid peavad automaatselt hindama analüüsi tulemusi. Testjuhtumid peavad olema koostatud nii, et hindamisprogramm saaks tagada analüüsi tulemusi automaatset hindamist. See tähendab, et testjuhtum peab järgima ülaltoodud juhiseid, selleks et ei tekiks olukorda kus testjuhtumis on olemas tahtmatu viga ja seepärast ei jõua testjuhtum veani, mis võib esineda koodis pärast tahtmatu vea leidmist. Leides aga tahtmatu vea, testjuhtum tagastab selle koha ilma analüüsita.

Allika järgi võib aga kindlasti tekkida ka negatiivseid aspekte isegi ülaltoodud juhiste kinni pidades. Programmid võivad olla koostatud sellisel viisil, et need on adapteeritud hindamisprogrammi testjuhtumitele ja seega ei tagasta hindamisprogramm tõelisi tulemusi. Samuti erinevad programmid kasutavad erinevas mahus ka ressursse, kuna erinevatel masinatel võivad olla erinevad mälu mahud ja erinevad protsessorid, seega ei ole võimalik

saada sama programmi hindamisel samad tulemused ja hindamisprotsess võib võtta palju aega lõpetamiseks ja tulemuste tagastamiseks.

3. Olemasolevad lahendused

Järgnevalt kirjeldatakse kahte olemasolevat lahendust, mida on Goblint'i hindamiseks kasutatud. Üks neist on Goblint'i autorite loodud, teine on üldisem.

3.1 Ruby skriptid

Eelmine lahendus Goblint'i korrektsuse ja võimsuse hindamiseks kasutas erinevaid Ruby programmeerimiskeeles kirjutatud skripte. Ruby keeles on olemas funktsionaalsus HTML märgendi genereerimiseks ja regulaaravaldiste kasutamiseks andmete väljavõtmiseks Goblint'i töö käigus loodud logifailidest.

Antud skriptid [8][9] olid tehtud sellisel moel, et oli tehtud esimene skript hindamisprogrammide käivitamiseks ja Goblint'i logifailidest vajaliku andmete väljavõtmiseks ja genereerimaks faili HTML märgendi antud hindamisprogrammide töö käigus saavutatud tulemustest ja kuvada need kasutajale. Kuna oli vajalik käivitada erinevaid hindamisprogramme ja kuvada erinevaid andmeid, oli vajadus ka kirjutada erinevaid Ruby skripte. Selleks oli võetud esimene skript, sellest tehti koopia ja lisati vajalik funktsionaalsus andmete vajalikus formaadis kuvamiseks. Nii tehti ka ülejäänud skriptidega ilma kasutu funktsionaalsuse kustutamiseta. Nii tekkisid skriptid mis olid üleküllastunud funktsionaalsusega, mis ei olnud vajalik mõnede skriptide jaoks.

3.2 BenchExec

BenchExec on avatud lähtekoodiga raamistik, mis võimaldab erinevate tarkvarasüsteemide hindamisprogrammide käivitamist, mõõtmist ja tulemuste kuvamist [5]. BenchExec on üldisem raamistik ja seega võimaldab erineva tarkvara ja tööriistade hindamist. Seepärast kasutatakse seda *SV-COMP*'is [10], mis on iga-aastane tarkvara valideerimisvõistlus, millest võtavad osa erinevad valideerimistööriistad. Kuna Goblint võttis osa sellest võistlusest, pidi see kasutama BenchExec'it oma tulemuste valideerimiseks.

BenchExec vajab oma struktuuri tõttu kindlat seadistust ja selle saavutamine võib osutada tülikaks, kuna seadistamise jaoks kasutatakse ebamugavat struktuuri. Seda struktuuri saavutakse XML faili või mitme XML failiga, ja need failid kirjeldavad Goblint'i või mõne teise tööriista seadistust, benchmark programmide hulka, väljundtabelis soovitud veerge. Samuti peab olema ka Bash keeles kirjutatud skript selleks et kõik need seadistused samal ajal käivitada ühe käsuga.

Ebamugav on ka *BenchExec'i* installeerimine kuna see vajab stabiilset Linux kerneli versiooni, cgroups seadistamist (cgroups on Linux kerneli funktsioon, mis võimaldab ressursside jaotamist) ja samuti ka Python virtuaalkeskkonna seadistamist.

4. Uus lahendus

Töö eesmärkide saavutamiseks olid defineeritud ka nõuded mis pidid olema rahuldatud:

- Peab olema raamistik, kus skriptide vahel enamus koodist peab olema jagatud.
- Raamistik peab olema kirjutatud keeles, mida Goblint'i arendajad tunnevad paremini kui Ruby programmeerimiskeelt.
- Analüüside käivitamisel kasutatakse paralleelsust.
- Võimaldaks realiseerida kõik loogika ja funktsionaalsus, mis olemasolevates skriptides on vaja.

4.1 Kasutatud tehnoloogiad

Järgnevalt kirjeldatakse olulisemaid tehnoloogiaid, mida on kasutatud GobExec'i raamistikus.

4.1.1 Jinja

Jinja on Python programmeerimiskeele jaoks loodud mallimootor, mis võimaldab HTML'i loomist kasutades Python keelele sarnast süntaksi. Loodud mallid on taaskasutatavad ja võimaldavad raamistiku loogika eraldamist HTML keeles genereeritud väljundist. Antud töös kasutatakse jinja malle väljundtabeli HTML märgendis genereerimiseks kui ka hindamise protsessis loodud klasside väärtuste kuvamiseks.

4.1.2 Asyncio

Asyncio ehk asünkroonne sisend-väljund (ingl. *asynchronous I/O*) on Python programmeerimiskeele teek mis võimaldab asünkroonse programmeerimise paradigma kasutamist tarkvara loomisel. Antud teek kasutab async ja await süntaksit.

Asünkroonses programmeerimises funktsioone nimetatakse korutinideks. Andmekaitse ja infoturbe leksikoni sõnaraamat defineerib rutiini tarkvaras üldotstarbelisena või tihti kasutatava programmina või programmiosana. Korutiin on funktsioon, mille täitmist võib peatada ja jätkata.

Async märksõna kasutatakse koos “def” märksõnaga korutiini defineerimisel. Sellist korutiini defineerib Python programmeerimiskeele sõnastik korutiinseks funktsiooniks. Await märksõna kasutatakse korutiinse funktsiooni kutsumiseks. Kutsumise protseessis tekib *awaitable* objekt, mis on spetsiifiline *await* avaldisele. Kutsudes korutiinse funktsiooni, antud funktsiooni käivitust peatatakse kuni korutiin mis on kutsutud enne lõpetab täitmist.

4.1.3 Dataclasses

Dataclasses on Python programmeerimiskeele teek mis pakub kasutajale dekoraatori ja funktsionaalsuse spetsiaalsete funktsioonide nt. *__init__( )* automaatseks lisamiseks kasutaja poolt loodud klassidele.

__init__( ) on spetsiaalne funktsioon, mis kutsutakse peale klassi objekti loomist, kuid enne selle tagastamist. Seda spetsiaalset funktsiooni kasutatakse objekti atribuutide initsialiseerimiseks vaikimisi või kasutaja poolt määratud väärtustega.

4.1.4 Re

Re on Python programmeerimiskeele teek võimaldab sobitamisoperaatsioonidee kasutamist regulaaravaldiste süntaksiga mis on sarnane regulaaravaldiste sobitamisega Perl keeles.

4.2 Metoodika

Hindamisraamistiku loomiseks valiti Python programmeerimiskeel, kuna see on populaarne keel, mis ei oma keerulist süntaksit ja suurem osa arendajatest valdab seda tasemel, mis on piisav hindamisraamistiku kasutamiseks.

Võeti olemasolevad Ruby programmeerimiskeeles kirjutatud skriptid ja pandi kirja nõuded, mis kirjeldasid vajaliku funktsionaalsust samade tulemuste saavutamiseks, ning realiseeriti uues raamistikus kasutades loodud klasse mis võimaldavad saavutada sama funktsionaalsust mis on juba Ruby skriptides implementeeritud.

Kuna ennem *GobExec* (hindamisraamistik) oli ainult prototüübi olekus, ei olnud koodibaas kuidagi kirjeldatud ega dokumenteeritud ja seega oli vaja ka olemasolevat koodi analüüsida ja muuta mõnede komponendide funktsionaalsust. Näiteks võib tuua renderdamise

protsessi. Algne renderdamisalgoritm, mis oli kirjutatud kasutades *jinja2* Python programmeerimiskeele teeki, pani hindamisprogrammi käivitamiseks võetud aja väärtuse sama lahtrisse koos *Goblint'i* logifailidest väljavõetud väärtustega, mis võiks tekitada segadusi hindamisprogrammi väljundfaili analüüsimisel. Samuti tekkis probleem andmete kuvamisega, kui käivitati erinevaid skripte, mis oli seotud sellega, et renderdamisalgoritm oli *hardcode*'itud ühe testskripti jaoks. Pärast nende vigade kõrvaldamist, renderdamisalgoritm kuvas väljundifaili oodatud formaadis.

Andmete väljavõtmiseks loodi klassid kasutades Python *dataclasses* teeki, mis sisaldab staatilist meetodi andmete väljavõtmiseks. Olemasolevatest Ruby skriptidest võeti regulaaravaldised, mis olid loodud erinevate andmete Goblint'i logifailidest väljavõtmiseks.

Olukordade jaoks, kus oleks vaja võrrelda erinevade hindamisprogrammide erinevaid käivitamiskonfiguratsioone, loodi klassid *Goblint'i* võrdlemistööriistade (privPrecCompare, apronPrecCompare) käivitamiseks. Need klassid võtavad sisendina *dump* failid, mis luuakse Goblint'i tööriista töö käigus. Iga *Goblint'i* tööriista käivitamine toimub kasutades *GoblintTool* klassi, mis oli *GobExec'i* prototüübis juba olemas, kuid oli vaja lisada parameetreid, mis tagaksid võrdlemistööriistade töö, selleks lisati *GoblintTool* klassi dump parameeter, mis lisab *Goblint'i* argumentide jadasse parameetri *dump* faili loomiseks. GoblintTool'i klassi oli lisatud ka validate parameeter, mis lisas argumentide jadasse parameetri .yml formaadis failide valideerimiseks. Need .yml formaadi failid kasutatakse skriptis *update_bench_traces_rel_yaml.py* .

Lõpuks kontrolliti, et vana ja uue skripti poolt kuvatud andmed oleks samad, et veenduda, et ümberkirjutamise käigus ei tekkinud vigu, mis võiks hindamise tulemust muuta.

4.3 Näidisskript

Näitena tooks *update_bench_traces_rel.py* skripti, mis vanas lahenduses oli nimega *update_bench_traces_rel.rb* .

```
from pathlib import Path

import gobexec.main
from gobexec.goblint import tool
from gobexec.goblint.bench import txtindex
from gobexec.goblint.result import ThreadSummary
from gobexec.goblint.tool import GoblintTool
```

```
from gobexec.model.result import TimeResult
from gobexec.model.tools import ExtractTool
from gobexec.output.renderer import FileRenderer, ConsoleRenderer, MultiRenderer


def index_tool_factory(name, args):
       goblint = GoblintTool(
           name=name,
           program=str(Path("../analyzer/goblint").absolute()),
           args=["--conf",
str(Path("../analyzer/conf/traces-rel.json").absolute()), "--enable",
"warn.debug"] + args,
           dump= 'apron'
       )

       return ExtractTool(
           goblint,
           TimeResult,
           ThreadSummary,

       )

matrix =
txtindex.load(Path("../bench/index/traces-relational.txt").absolute(),index_tool
_factory)
apronprec = tool.ApronPrecTool(args= matrix.tools.copy())
matrix.tools.append(apronprec)
html_renderer = FileRenderer(Path("out.html"))
console_renderer = ConsoleRenderer()
renderer = MultiRenderer([html_renderer, console_renderer])

gobexec.main.run(matrix, renderer)
```

Antud skriptis imporditakse klassid *Goblint'i* käivitamiseks (*GoblintTool*), renderdamiseks (*FileRenderer, ConsoleRenderer, MultiRenderer*), andmete logifailidest väljavõtmiseks (*ThreadSummary*). Imporditakse ka tool.py fail, kus on defineeritud kõik tööriistade klassid, selles skriptis on vaja kasutada võrdlemistööriista *ApronPrecTool*. Pärast importimist defineeritakse funktsioon *index_tool_factory*, kus konfigureeritakse *Goblint'it* ja tagastatakse *ExtractTool* objekt, mis sisaldab *GoblintTool* objekti ning klasse aja ja andmete väljavõtmiseks. *ExtractTool* vajab ka meetodit andmete väljavõtmiseks, selleks on igas andmeklassis loodud asünkroonne staatiline *extract* funktsioon. Kasutades *txtindex.py* faili laaditakse hindamisprogrammi fail, mis selle skripti puhul on *traces-relational.txt* ja asub Goblint'i bench repositooriumis [6], ja kutsutakse *index_tool_factory* funktsiooni. Luuakse

ApronPrecCompare klassi objekt, antakse sellele argumentina kõik *GoblintTool'ide* konfiguratsioonid ja lisatakse seda tööriistade jadasse. Lõpuks renderdatakse väljundfail. Võrreldes vana Ruby programmeerimiskeeles kirjutatud skriptiga, uus skript on lühem, 36 rida Python eelneva 326-realise Ruby skripti asemel, arusaadavam tänu koodi jagamisele, kuid realiseerib sama funktsionaalsust kui Ruby keeles kirjutatud skript. Kõik uuendatud skriptid koos loodud funktsionaalsusega on lisatud *GitHub* repositooriumi, millele link on toodud lisades.

5. Valideerimine

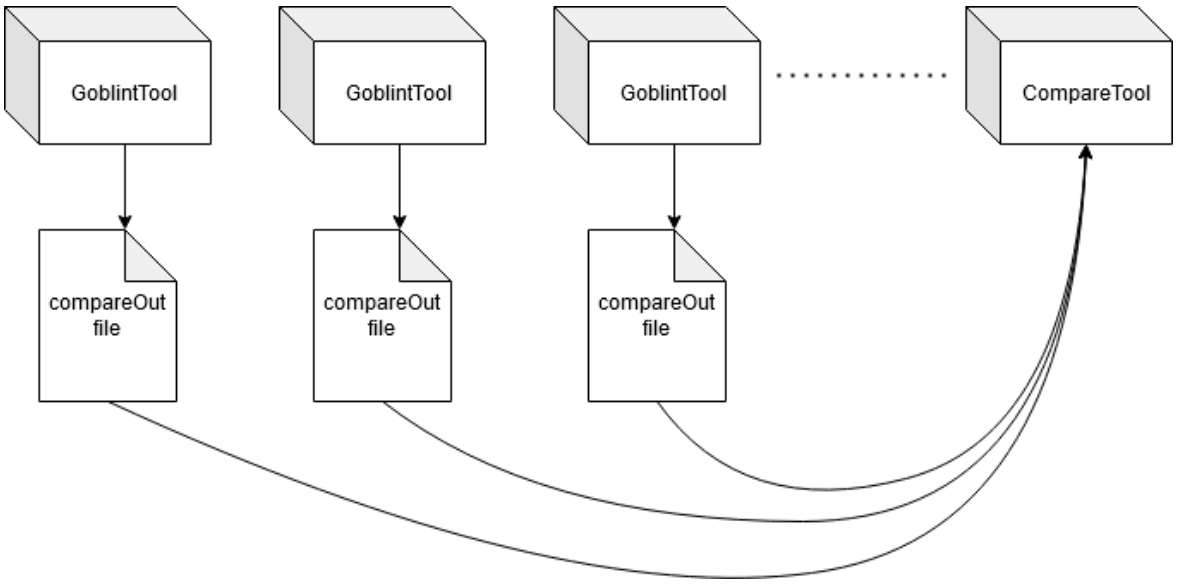
Antud töö tulemuste valideerimisiks võeti uues hindamisraamistikus realiseeritud skriptide väljundfailid ning need võrreldi vana Ruby keeles kirjutatud skriptide väljundfailidega.

5.1 update_bench_traces

Antud skripti logifailidest võetakse saavutatavad koodi ridade arvud, saavutamata koodi ridade arvud ja kõikide koodi ridade arvud. Samuti selles skriptis kasutakse võrdlemist.

5.1.1 Võrdlemine

Mitmed Ruby programmeerimiskeeles kirjutatud skriptidest kasutavad võrdlemist, ehk privPrecCompare ja apronPrecCompare tööriistu, seega oli vaja neid implementeerida Python raamistikus. Joonisel 5.1.1 on esitatud valideerimistööriista töö skeem, kus Goblint tööriista töö protsessis tekitatakse võrdlemise jaoks väljundfaile ja edastatakse võrdlemistööriistale. Tööriistad võrdlevad erinevate analüüside tulemuste täpsust, kasutades abstraktseid väärtuseid, mille Goblint iga seadistusega välja arvutas.



Joonis 5.1.1 Võrdlemistööriista töö skeem

traces

Group	Benchmark	protection	
POSIX Apps (ordered by logical LoC within this group)	pfscan	3.17s	522 40 562
	aget	7.32s	583 4 587
	ctrace	7.61s	338 314 652
	knot	7.01s	566 415 981
	ypbind	54.19s	960 32 992
	smtprc	37.24s	2893 144 3037
SV-COMP (ordered by logical LoC within this group)	iowarrior	6.04s	889 456 1345
	adutux	7.59s	1072 448 1520
	w83977af	15.32s	1043 458 1501
	tegra20	17.37s	1194 353 1547
	nsc	31.20s	1643 736 2379
	marvell1	24.61s	1420 1045 2465
	marvell2	29.75s	1421 1044 2465

Joonis 5.1.2 *update_bench_traces.py* skripti väljundfaili osa

#	Name	Size	protection
1	pfscan	1295 lines	1.61 s (3; **40**+**522**=562)
2	aget	1280 lines	3.01 s (**4**+**583**=587)
3	ctrace	1407 lines	2.90 s (17; **314**+**338**=652)
4	knot	2255 lines	4.23 s (32; **415**+**566**=981)
5	ypbind	6588 lines	13.77 s (1; **15**+**979**=994)
6	smtprc	5787 lines	8.64 s (4; **144**+**2892**=3036)

#	Name	Size	protection
7	iowarrior	7687 lines	1.44 s (119; **456**+**889**=1345)
8	adutux	8114 lines	2.08 s (107; **448**+**1072**=1520)
9	w83977af	10071 lines	3.55 s (147; **458**+**1043**=1501)
10	tegra20	7111 lines	12.56 s (86; **353**+**1194**=1547)
11	nsc	12778 lines	7.86 s (168; **736**+**1643**=2379)
12	marvell1	12246 lines	8.10 s (225; **1043**+**1422**=2465)
13	marvell2	12256 lines	8.06 s (225; **1042**+**1423**=2465)

Joonis 5.1.3 *update_bench_traces.rb* skripti väljundfaili osa

5.2 update_bench_traces_rel

Antud skripti logifailidest võetakse välja lõimede identifikaatorite arv, unikaalsete lõimede identifikaatorite arv, maksimaalne arv muutujaid, mida kaitseb üks lukk ja nende summa, lukkude arv. Selles skriptis kasutakse ka võrdlemist.

traces-relational

Group	Benchmark	box	
POSIX Apps (ordered by logical LoC within this group)	pfscan	7.72s	3 2 3 6 4
	aget	8.51s	6 4 0 0 0
	ctrace	10.27s	3 3 1 1 1
	knot	13.25s	9 5 0 0 0
	ypbind	9.04s	0 0 0 0 0
	smtprc	66.23s	4 3 0 0 0
SV-COMP (ordered by logical LoC within this group)	iowarrior	13.30s	4 4 20 44 3
	adutux	17.86s	4 4 18 33 3
	w83977af	24.07s	6 4 7 7 1
	tegra20	58.10s	7 5 29 29 1
	nsc	276.56s	11 7 44 44 1
	marvell1	88.71s	6 5 35 58 2
	marvell2	105.34s	6 5 34 57 2

Joonis 5.2.1 *update_bench_traces_rel.py* skripti väljundfaili osa

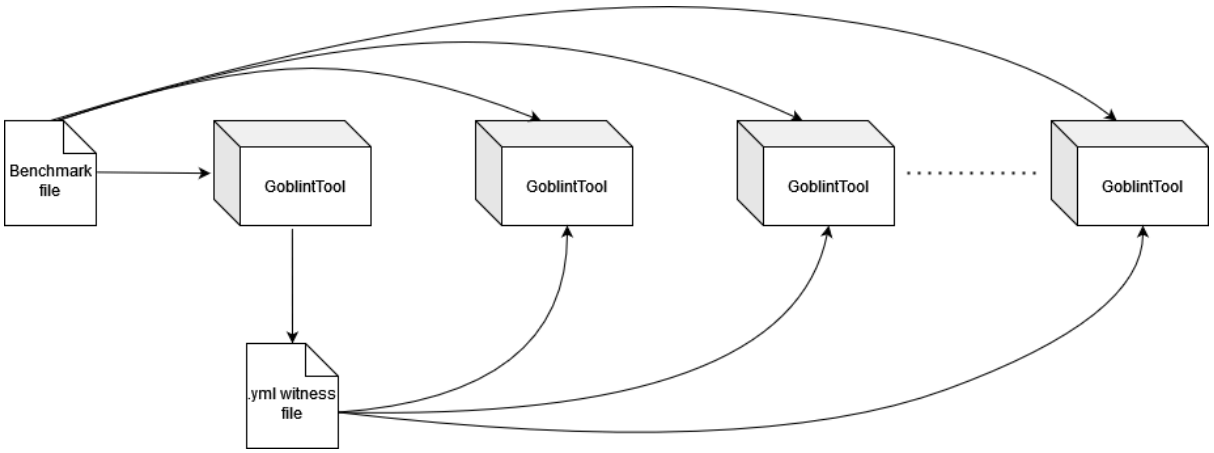
#	Name	Size	protection
1	pfscan	1295 lines	2.41 s (562;**T: 3 (2)**;**M: 4 (3;6;1)**)
2	aget	1280 lines	3.59 s (587;**T: 6 (4)**;**M: 0 (0;0;0)**)
3	ctrace	1407 lines	3.65 s (652;**T: 3 (3)**;**M: 1 (1;1;1)**)
4	knot	2255 lines	4.31 s (981;**T: 9 (5)**;**M: 0 (0;0;0)**)
5	ypbind	6588 lines	33.67 s (992;**T: 59 (41)**;**M: 3 (21;23;7)**)
6	smtprc	5787 lines	19.50 s (3037;**T: 4 (3)**;**M: 0 (0;0;0)**)

#	Name	Size	protection
7	iowarrior	7687 lines	4.05 s (1345;**T: 4 (4)**;**M: 3 (20;44;14)**)
8	adutux	8114 lines	3.69 s (1520;**T: 4 (4)**;**M: 3 (18;33;11)**)
9	w83977af	10071 lines	7.66 s (1501;**T: 6 (4)**;**M: 1 (7;7;7)**)
10	tegra20	7111 lines	10.49 s (1547;**T: 7 (5)**;**M: 1 (29;29;29)**)
11	nsc	12778 lines	14.38 s (2379;**T: 11 (7)**;**M: 1 (44;44;44)**)
12	marvell1	12246 lines	12.07 s (2465;**T: 6 (5)**;**M: 2 (35;58;29)**)
13	marvell2	12256 lines	8.03 s (2465;**T: 6 (5)**;**M: 2 (34;57;28)**)

Joonis 5.2.2 *update_bench_traces_rel.rb* skripti väljundfaili osa

5.3 update_bench_traces_yaml

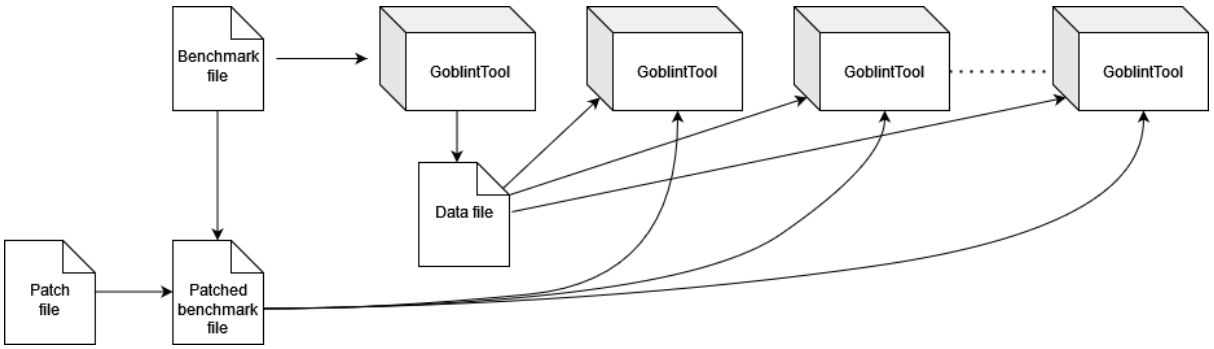
Antud skript põhineb genereeritud *yaml* formaadi *witness* failidel, mis sisaldavad analüüsitud programmi invariante, mida pärast valideeritakse. Genereerimise skeem on sarnane ApronPrecCompare ja PrivPrecCompare võrdlemistööriistadega, kuid antud skriptis witness faili genereerib ainult üks GoblintTool objekt ja edastab seda teistele GoblintTool objektidele, mis witness faili valideerivad. Samuti skripti logifailidest võetakse välja invariantide arvud ehk *assertion*-id ja tõestatud invariantide arvud, kus *Goblint* suutis veenduda et invariant kehtib. See protsess on toodud skeemina joonisel 5.3.



Joonis 5.3 Yaml formaadi witness faili edastamise skeem

5.4 update_bench_traces_incremental

Antud skript kasutab inkrementaalseid analüüse ehk esimene analüüs rakendatakse tavapäraselt, kuid ülejäänud analüüsid põhinevad esimese analüüsi tulemustel. Pärast esimest analüüsi benchmark failile rakendatakse *patch*, ehk fail mis koosneb muudatustest mis peavad olema rakendatud benchmark failile. Järgmine analüüs analüüsib ainult muudetud benchmark faili osad, kuna ülejäänud osad on juba analüüsitud. Joonisel 5.4.1 on kujutatud programmmi töö skeem. Esimene *Goblint* tööriist analüüsib programmi ja tekitab väljundfaili, mis sisaldab esimese analüüsi andmeid ja edastab selle ülejäänud *Goblint* tööriistadele. Samuti ka rakendatakse esimese *Goblint* tööriista töö lõppus hindamisprogrammile *patch* ja edastatakse ülejäänud *Goblint* tööriistadele.



Joonis 5.4.1 update_bench_traces_incremental.py faili töö skeem

incremental

Group	Benchmark	from_scratch		Incremental		Reluctant	
posix	aget	7.92s	2865 5940	1.61s	17 73	1.51s	15 18
	knot	11.55s	7456 17654	8.88s	4853 14119	8.61s	4847 14071
	ypbind	54.37s	41980 71865	16.12s	6209 11041	16.13s	6209 11041
	smtprc	37.79s	18941 30941	6.50s	90 385	6.23s	76 245

Joonis 5.4.2 update_bench_incremental.py väljundfail

5.5 Ülejäänud skriptid

update_bench_traces_rel_watts.py skripti logifailidest võetakse välja tingimuse kontrollid ehk *assertion*-id, kus *Goblint* suutis veenduda et kontroll kehtib, ei kehti, ei suutnud veenduda kas kontroll kehtib või mitte ja saavutamatute kontrollide arv (kontrollimiseni programm kunagi ei jõua).

Sellega analoogsed skriptid on *update_bench_traces_rel_toy.py*, *update_bench_traces_rel_ratcop.py*, kuid nendes skriptides võetakse välja ainult kehtivate kontrollide arvu ja kontrollide arvu, kus *Goblint* ei suutnud veenduda kontrolli kehtivuses.

Skript *update_bench_traces_rel_assert.py* on väga sarnane skriptiga *update_bench_traces_yaml.py*, kuid selles skriptis yaml witness failide valideerimise asemel analüüsivad ülejäänud *Goblint*'id *assert*'idega programmi.

Group	Benchmark	asserts	base-pb	
Watts benchmarks by Kusano, Wang	thread01	1	0.21s	0, 0, 1, 0
	threadcreate01	1	0.13s	0, 1, 0, 0
	threadcreate02	1	0.14s	0, 1, 0, 0
	sync_01_true	1	0.19s	0, 0, 0, 1
	sync_02_true	1	0.17s	0, 0, 0, 1
	intra01	1	0.12s	0, 0, 0, 1
	dekker1	1	0.17s	0, 0, 1, 0
	fk2012_v2	1	0.17s	1, 0, 0, 0
	keybISR	2	0.15s	0, 0, 0, 2
	ib700wdt_01	1	0.19s	0, 0, 0, 1
	ib700wdt_02	1	0.32s	0, 0, 0, 1
	ib700wdt_03	81	0.57s	40, 40, 0, 1
	i8xx_tco_01	1	0.15s	0, 0, 0, 1
	i8xx_tco_02	1	0.24s	0, 0, 0, 1
	i8xx_tco_03	104	0.72s	60, 43, 0, 1
	machzwd_01	1	0.25s	0, 0, 0, 1
	machzwd_02	1	0.29s	0, 0, 0, 1
	machzwd_03	83	0.57s	82, 0, 0, 1
	mixcomwd_01	1	0.26s	0, 0, 0, 1
	mixcomwd_02	62	0.58s	1, 60, 0, 1
	pcwd_01	1	0.24s	0, 0, 0, 1
	pcwd_02	82	0.57s	41, 40, 0, 1
	sbc60xxwdt_01	1	0.36s	0, 0, 0, 1
	sc1200wdt_01	1	0.32s	0, 0, 0, 1
	sc1200wdt_02	93	0.49s	30, 62, 0, 1
	smsc37b787wdt_01	1	0.28s	0, 0, 0, 1

Joonis 5.5.1 *update_bench_traces_rel_watts.py* skripti väljundfaili osa

#	Name	Size	base-pb
1	thread01	29 lines	0.08 s (15;0;0;1;0)
2	threadcreate01	24 lines	0.06 s (8;0;1;0;0)
3	threadcreate02	28 lines	0.07 s (10;0;1;0;0)
4	sync_01_true	38 lines	0.07 s (16;0;0;0;1)
5	sync_02_true	36 lines	0.07 s (14;0;0;0;1)
6	intra01	41 lines	0.07 s (14;0;0;0;1)
7	dekker1	65 lines	0.08 s (28;0;0;1;0)
8	fk2012_v2	88 lines	0.08 s (29;1;0;0;0)
9	keybISR	62 lines	0.07 s (21;0;0;0;2)
10	ib700wdt_01	346 lines	0.08 s (14;0;0;0;1)
11	ib700wdt_02	466 lines	0.14 s (94;0;0;0;1)
12	ib700wdt_03	589 lines	0.24 s (165;40;40;0;1)
13	i8xx_tco_01	736 lines	0.08 s (34;0;0;0;1)
14	i8xx_tco_02	901 lines	0.14 s (132;0;0;0;1)
15	i8xx_tco_03	1029 lines	0.23 s (208;60;43;0;1)
16	machzwd_01	667 lines	0.09 s (65;0;0;0;1)
17	machzwd_02	795 lines	0.12 s (149;0;0;0;1)
18	machzwd_03	885 lines	0.18 s (197;82;0;0;1)
19	mixcomwd_01	457 lines	0.10 s (51;0;0;0;1)
20	mixcomwd_02	580 lines	0.18 s (127;1;60;0;1)
21	pcwd_01	1197 lines	0.09 s (33;0;0;0;1)
22	pcwd_02	1407 lines	0.22 s (165;41;40;0;1)
23	sbc60xxwdt_01	683 lines	0.11 s (101;0;0;0;1)
24	sc1200wdt_01	715 lines	0.11 s (98;0;0;0;1)
25	sc1200wdt_02	770 lines	0.29 s (125;30;62;0;1)
26	smsc37b787wdt_01	904 lines	0.11 s (50;0;0;0;1)

Näidis 5.5.2 *update_bench_traces_rel_watts.rb* skripti väljundfaili osa

traces-rel-toy

Group	Benchmark	base-pb	
box programs	escape-local-in-pthread-simple	0.34s	1, 1
	branched-not-too-brutal	0.21s	1, 0
	tid-curious	0.27s	1, 0
oct programs	traces-max-simple	0.21s	0, 1
	traces-min-rpb1	0.24s	0, 2
	traces-min-rpb2	0.18s	0, 2
	traces-cluster-based	0.33s	0, 4
	traces-write-centered-vs-meet-mutex	0.54s	0, 2
	traces-write-centered-problem	0.34s	0, 1
	airline	0.29s	0, 1
	queuesize-const	0.33s	0, 20
	mine14	0.24s	0, 1
	mine14-5b	0.32s	0, 1
tid programs	pfscan-workers-strengthening	0.25s	0, 1
	tid-toy1	0.22s	0, 1
	tid-toy3	0.21s	0, 2
	tid-toy5	0.20s	0, 2
	tid-toy6	0.27s	0, 2
	tid-toy7	0.25s	0, 2
	tid-toy8	0.23s	0, 3
	tid-toy9	0.21s	0, 1
	sync	0.22s	0, 2
	tid-toy10	0.23s	0, 1
	tid-toy11	0.25s	0, 1
	no-loc	0.23s	0, 1

Joonis 5.5.3 *update_bench_traces_rel_toy.py* skripti väljundfaili osa

#	Name	Size	base-pb
1	escape-local-in-pthread-simple	29 lines	0.09 s (1; 1)
2	branched-not-too-brutal	33 lines	0.09 s (1; 0)
3	tid-curious	29 lines	0.08 s (1; 0)

#	Name	Size	base-pb
4	traces-max-simple	30 lines	0.08 s (0; 1)
5	traces-min-rpb1	32 lines	0.09 s (0; 2)
6	traces-min-rpb2	53 lines	0.10 s (0; 2)
7	traces-cluster-based	71 lines	0.11 s (0; 4)
8	traces-write-centered-vs-meet-mutex	57 lines	0.11 s (0; 2)
9	traces-write-centered-problem	40 lines	0.09 s (0; 1)
10	airline	47 lines	0.11 s (0; 1)
11	queuesize-const	79 lines	0.11 s (0; 20)
12	mine14	35 lines	0.10 s (0; 1)
13	mine14-5b	51 lines	0.11 s (0; 1)

#	Name	Size	base-pb
14	pfscan-workers-strengthening	44 lines	0.10 s (0; 1)
15	tid-toy1	43 lines	0.09 s (0; 1)
16	tid-toy3	43 lines	0.09 s (0; 2)
17	tid-toy5	54 lines	0.08 s (0; 2)
18	tid-toy6	65 lines	0.09 s (0; 2)
19	tid-toy7	76 lines	0.10 s (0; 2)
20	tid-toy8	66 lines	0.09 s (0; 3)
21	tid-toy9	36 lines	0.08 s (0; 1)
22	sync	40 lines	0.10 s (0; 2)
23	tid-toy10	45 lines	0.09 s (0; 1)
24	tid-toy11	77 lines	0.09 s (0; 1)
25	no-loc	70 lines	0.09 s (0; 1)

Joonis 5.5.4 *update_bench_traces_rel_toy.rb* skripti väljundfaili osa

traces-rel-assert

Group	Benchmark	box	
POSIX Apps (ordered by logical LoC within this group)	pfscan	22.64s	10, 136
	aget	11.00s	0, 0
	ctrace	131.96s	720, 64
	knot	15.98s	0, 0
	smtprc	1.22s	0, 0
SV-COMP (ordered by logical LoC within this group)	iowarrior	14.79s	6, 0
	adutux	18.12s	0, 0
	w83977af	24.49s	2, 0
	tegra20	50.23s	0, 0
	nsc	0.95s	0, 0
	marvell1	0.95s	0, 0
	marvell2	0.94s	0, 0

Joonis 5.5.5 *update_bench_traces_rel_assert.py* skripti väljundifaili osa

POSIX Apps (order			
#	Name	Size	box
1	pfscan	3666 lines	58.94 s (10; 136)
2	aget	3140 lines	18.61 s (0; 0)
3	ctrace	5083 lines	187.84 s (720; 64)
4	knot	5045 lines	4.44 s (0; 0)
5	smtprc	11915 lines	failed (code: 2)
SV-COMP (ordere			
#	Name	Size	box
6	iowarrior	11674 lines	4.14 s (6; 0)
7	adutux	12045 lines	4.71 s (0; 0)
8	w83977af	14388 lines	7.01 s (2; 0)
9	tegra20	11162 lines	15.32 s (0; 0)
10	nsc	18613 lines	failed (code: 2)
11	marvell1	19159 lines	failed (code: 2)
12	marvell2	19029 lines	failed (code: 2)

Joonis 5.5.6 *update_bench_traces_rel_assert.rb* skripti väljundfaili osa

5.6 Kiirus

Uus Python programmeerimiskeeles kirjutatud raamistik kasutab paralleelsust, mis tagab kiirema andmete tagastamise kasutajale. Eelnevad Ruby keeles kirjutatud skriptid käivitasid programme järjest. Kiiruse all peame silmas *wall time*'i, ehk programmi käivitamiseks kulunud reaalaega koos andmete tagastamisega, sest kui võrrelda neid kasutades *CPU time*'i, ehk aega, mille jooksul protsessorit kasutati programmi käivitamiseks, Ruby programmeerimiskeeles kirjutatud skript võib olla isegi kiirem tänu järjest käivitusele, sest see ei koorma protsessorit sama palju kui paralleelne käivitamine, kuid aga ei tagasta andmeid sama kiirusega, kui Python'i programm.

```
evalpet@LAPTOP-7LBGB0JT:~/bench$ time ruby update_bench_traces_rel_toy.rb
```

Joonis 5.6.1 Käsk Ruby programmi käivitamiseks

```
real    2m49.244s
user    2m37.759s
sys     0m11.385s
```

Joonis 5.6.2 Aeg kulunud Ruby programmi käivitusele

```
evalpet@LAPTOP-7LBGB0JT:~/GobExec$ time python3.10 udpate_bench_traces_rel_toy.py
```

Joonis 5.6.3 Käsk Python programmi käivitamiseks

```
real    1m33.539s
user    3m32.768s
sys     0m11.927s
```

Joonis 5.6.4 Aeg kulunud Python programmi käivitusele

Kulunud aega mõõtmiseks kasutati Linux time käsku. See näitab reaalaega, CPU aega ja süsteemi aega, mis on süsteemi protsessidele kulunud CPU aeg.

Joonistest 5.6.2 ja 5.6.4 on näha, et Python'i programm on palju kiirem. Samuti on sellel programmil kulunud CPU aeg suurem kui kulunud reaalaeg, see on nii paralleelsuse tõttu, kuna mitmete tuumade aeg liidetakse kokku. Käivituse korral kasutati 3 tuuma 4-jast võimalikust.

Kokkuvõte

Praegune lahendus hindamiseks tugineb Ruby skriptidele, mis on muutunud hooldamatuks ja keeruliseks laiendada. See lõputöö kirjeldab ühtset jõudlusetestimise raamistikku programmianalüüsi tööriistade jaoks Python programmeerimiskeeles. Programmeerimiskeele valik raamistiku realiseerimiseks põhines keele populaarsuses ja süntaksi lihtsuses. Kuna Python on teatud rohkem kui Ruby ja Goblint'i arendajad on sellega rohkem tuttavad, valiti just see programmeerimiskeel. Raamistik pakub vähemalt sama funktsionaalsust kui vana lahendus ning võimaldab vanade hindamisprogrammide teisendamist ja asendamist. Uus raamistik muudab jõudlusetestimise komplekti funktsionaalsuse laiendamise ja hooldamise lihtsamaks, kuna raamistiku koodi jagatakse enamasti erinevate moodulite vahel. Samuti kasutab uus raamistik paralleelsust, mis teeb selle kiiremaks kui vanad skriptid, mis paralleelsust ei kasutanud. Lahendust valideeriti esmalt vanade skriptide teisendamisega Python programmeerimiskeelde kasutades uut raamistikku, ja seejärel võrreldi uusi skripte vanade skriptidega. Teisendatud skriptid on palju lihtsamad, kuid säilitavad sama funktsionaalsuse ja hindamisprogrammide tulemused.

Tulevikus võiks võiks muuta renderdamisalgoritmi. Hetkel ei ole võimalik mitu skripti samal ajal käivitada ega ohutult skripti tööd peatada, kuna praegune renderdamisalgoritm lõpetab väljendfaili värskendamise kui skript on oma töö lõpetanud, aga kui tuleb signaal töö peatamiseks, skripti töö ei lõppe ja väljendfail jääb lõpmatusse värskendustsüklisse.

Viidatud kirjandus

[1] Gerrard M., Borges M., Dwyer M. and Filieri A. Conditional Quantitative Program Analysis. IEEE Transactions on Software Engineering, vol. 48, no. 04, pp. 1212-1227, 2022.
doi: 10.1109/TSE.2020.3016778

[2] Herter J., Kästner D., Mallon Ch., Wilhelm R. Benchmarking static code analyzers. Reliability Engineering & System Safety, Volume 188, 2019, pp336-346, https://doi.org/10.1016/j.ress.2019.03.031

[3] https://www.misra.org.uk/ 09.05.2023

[4] Shiraishi S., Mohan V. & Marimuthu H. Test Suites for Benchmarks of Static Analysis Tools. 10.1109/ISSREW.2015.7392027. 2015.

[5] Beyer, D., Löwe, S. & Wendler, P. Reliable benchmarking: requirements and solutions. *Int J Softw Tools Technol Transfer* **21**, 1–29 (2019). https://doi.org/10.1007/s10009-017-0469-y

[6] https://github.com/goblint/bench 09.05.2023

[7] https://github.com/goblint/analyzer 09.05.2023

[8] Schwarz, M., Saan, S., Seidl, H., Apinis, K., Erhard, J., Vojdani, V. (2021). Improving Thread-Modular Abstract Interpretation. In: Drăgoi, C., Mukherjee, S., Namjoshi, K. (eds) Static Analysis. SAS 2021. Lecture Notes in Computer Science(), vol 12913. Springer, Cham. https://doi.org/10.1007/978-3-030-88806-0_18

[9] Schwarz, M., Saan, S., Seidl, H., Erhard, J., Vojdani, V. (2023). Clustered Relational Thread-Modular Abstract Interpretation with Local Traces. In: Wies, T. (eds) Programming Languages and Systems. ESOP 2023. Lecture Notes in Computer Science, vol 13990. Springer, Cham. https://doi.org/10.1007/978-3-031-30044-8_2

[10] Beyer, D. (2024). State of the Art in Software Verification and Witness Validation: SV-COMP 2024. In: Finkbeiner, B., Kovács, L. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2024. Lecture Notes in Computer Science, vol 14572. Springer, Cham. https://doi.org/10.1007/978-3-031-57256-2_15

Lisad

I. Link *GitHub* repositooriumile

Praktilise töö repositooriumi link: https://github.com/healthypunk/GobExec

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Evaldas Petnjunas,

(autori nimi)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

GobExec: programmianalüsaatorite hindamisraamistik,

(lõputöö pealkiri)

mille juhendaja on Simmo Saan,

reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Evaldas Petnjunas

15.05.2024