Kauno technologijos universitetas
Informatikos fakultetas
Programų inžinerijos katedra

Tomas Lukošius

Loginių schemų struktūros analizė

Magistro darbas

Darbo vadovas
prof., habil. dr. Rimantas Šeinauskas

Kaunas
2004

Turinys

1. Summary

Author: Tomas LUKOŠIUS

Title: Structure analysis of combinational logic circuit

Adviser: prof., habil. dr. Rimantas Šeinauskas

The combinational logic circuits, which are performing some kind of logic function, can have several realizations. Realizations differ from each other, because of the elements of the database used in circuit. The test of one scheme realization do not necessarily fully verify mistakes of the other realization. The number of pathways in different realizations may also differ.

The determination of dependence between test's propriety and the number of pathways for different circuits is the main task in this paper. After finding pathways in different realizations of circuit, and comparing these pathways, the number of fictional pathways in every realization is detected.

Special software was developed for calculating pathways in VHDL combinational logic circuits. The software was used for testing pathway calculation operations of circuit realization and for comparing pathways of realizations.

The main purpose of this paper was to develop software for pathways in VHDL circuit calculation, to perform experiments using this software, and to estimate the dependence for the number of pathways.

The practise of developed software is wide. This system may be implemented for optimising the algorithm of test generator. Usually the test generation program generates more than minimum of possible tests. If the number of pathways in circuit is known, the developed software will help to optimise the algorithm of test generation so that the minimum number of tests would be generated .

2. Santrauka

Autorius: Tomas LUKOŠIUS

Darbo tema: Loginių schemų struktūros analizė

Vadovas: prof., habil. dr. Rimantas Šeinauskas

Loginės schemos atliekančios kokią nors loginę funkciją gali turėti keletą realizacijų. Realizacijos viena nuo kitos skiriasi schemoje naudota elementų baze. Vienos schemos realizacijos testai nebūtinai pilnai patikrina kitos realizacijos klaidas. Kelių skaičius skirtingose realizacijose taip pat gali būti skirtingas.

Šiame darbe ieškoma priklausomybės tarp testų tinkamumo ir kelių skaičiaus skirtingoms schemos realizacijoms. Suskaičiavus kelius skirtingose schemos realizacijose, ir vykdant rastų kelių sulyginimą yra aptinkama kiek kiekviena realizacija turi fiktyvių kelių.

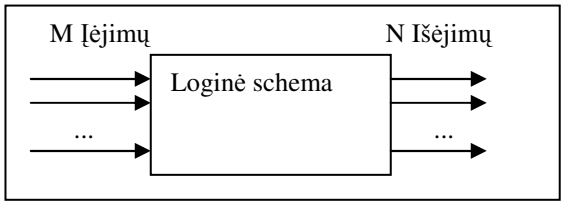
Kelių skaičiavimui VHDL loginėse schemose buvo sukurta speciali programinė įranga. Ji buvo naudojama atliekant eksperimentus schemų realizacijų kelių skaičiavimui ir suskaičiuotų skirtingų realizacijų kelių sulyginimui.

Darbo tikslas – sukurti programinę įrangą VHDL schemų kelių skaičiavimui; su sukurta sistema atlikti eksperimentus ir nustatyti priklausomybes, susijusias su kelių skaičiumi.

Sukurtos programinės įrangos pritaikymas yra platesnis. Ši sistema gali būti naudojama ir testų generatorių algoritmų optimizavimui. Dažniausiai testų generavimą atliekančios programos sugeneruoja didesnį negu minimalų galimą testų skaičių. Žinant schemoje esančių kelių skaičių, galima optimizuoti testų generavimo algoritmą taip, kad būtų generuojamas minimalus testų skaičius.

3. Įvadas

Loginė schema – tai aibė susijusių loginių elementų, kurie realizuoja loginę funkciją. Kiekvienos funkcijos rezultatai priklauso nuo funkcijos kintamųjų. Loginėse schemose, kurios realizuoja logines funkcijas, kintamuosius atitinka schemos įėjimai. Kiekvienos loginės schemos bendra struktūra pavaizduota 1 paveiksle.



1. paveikslas Loginės schemos bendra struktūra.

Elementarus loginis elementas – tai smulkiausia loginės schemos architektūros dalis. Elementarių loginių elementų yra tik trys: elementas realizuojantis loginę IR operaciją (angl. *AND gate*), elementas realizuojantis loginę ARBA operaciją (angl.*OR gate)* ir inversiją realizuojantys elementai (angl. *NOT gate*). Loginės schemos realizuoja funkcijas, kurios yra aprašytos Būlio algebra (angl. *Boolean Algebra*) [8].

Šiuolaikinių loginių schemų architektūrą sudaro gausybė elementarių loginių elementų. Didelės schemos vadinamos procesoriais arba mikroprocesoriais. Kad viską supaprastinti, kai kurios susijusių elementarių elementų grupės yra išskiriamos iš procesoriaus į smulkesnes logines schemas ir joms suteikiami vardai. Išskirtos schemos atlieka tam tikrą vaidmenį (tam tikrą loginę funkciją) mikroprocesoriuje. Jos gali būti tobulinamos, persintezuojamos. Persintezuotos schemos gali būti realizuotos skirtingomis elementų aibėmis – jos vadinamos schemų realizacijomis.

Prieš paleidžiant sukurtą procesorių į gamybą reikia būti tikram, kad jo veikimas teisingas. Teisingumą gali užtikrinti testai. Patogiau tikrinti išskirtas procesoriaus dalis negu visą procesorių. Ištestavus atskiras dalis ir atlikus tik svarbiausius testus visam procesoriui galima teigti, kad procesoriaus darbas teisingas.

Šiame darbe bus nagrinėjamos tik išskiriamos procesoriaus dalys ir jų testavimas. Bus analizuojama, kaip skiriasi schemos dalių realizacijos šiais charakterizavimo faktoriais:

1. Kiek yra kelių (nuo vieno įėjimo į išėjimus) pasirinktose realizacijos.
2. Kiek yra lyginių ir kiek nelyginių kelių (nuo vieno įėjimo į išėjimus) pasirinktose realizacijos.
3. Kiek yra padengiančių kelių (nuo vieno įėjimo į išėjimus) pasirinktose realizacijos.

Remiantis [4] sudarytais testų skaičių ir tinkamumo rezultatais bus bandoma nustatyti priklausomybes tarp kelių ir testų skaičiaus skirtingose realizacijose.

Darbo tikslas – rasti priklausomybę tarp kelių skaičiaus ir testų skaičiaus skirtingose realizacijose. Tai reikalinga tam, kad nereiktų sudarinėti naujų testų skirtingoms realizacijoms. Toks priklausomybės nustatymas leis negeneruojant naujų testų atlikti dalinį testavimą kitoms schemos realizacijoms.

Schemoms analizuoti ir sulyginti buvo specialiai sukurta programinė įranga, kuri bus naudojama atliekant eksperimentus. Sukurtos programinės įrangos specifikacijos ir architektūra taip pat bus apžvelgta.

Dokumento struktūra: suformuosime uždavinius ir apžvelgsime literatūroje susijusius darbus skyriuje **4.Analitinė dalis**. Naudotos programinės įrangos kūrimo aprašymas skyriuje **5.Projektinė dalis**. Vykdyti eksperimentai naudojant sukurtą programinę įrangą skyriuje **6.Eksperimentinė dalis**. Programinės įrangos naudingumo išvados bei gautų eksperimentuose rezultatų apibendrinimas skyriuje **7.Išvados**.

3.1 Darbo uždaviniai

Sudaryti algoritmą ir programą VHDL loginių schemų analizei, kuri paskaičiuotų lyginių ir nelyginių kelių kiekį tarp schemos įėjimų ir išėjimų. Remiantis analizės rezultatais nustatyti priklausomybę tarp schemos struktūrą charakterizuojančių parametrų ir testo tinkamumo skirtingoms schemos realizacijoms. Vykdant schemų sulyginimus nustatyti fiktyvių kelių skaičių realizacijose.

4. Analitinė dalis

4.1 Užduotis

Sukurti teisingai veikiantį algoritmą ir programą VHDL programavimo kalba aprašytų loginių schemų ryšiams tarp įėjimų ir išėjimų nustatyti. Atlikti eksperimentą ir ieškoti priklausomybės tarp kelių skaičiaus skirtingose realizacijose ir skirtingų realizacijų testų tinkamumo. Rasti netestuojamų ryšių (fiktyvių kelių) skaičių skirtingose schemos realizacijose vykdant palyginimus. Eksperimentui naudoti ISCAS'85 schemas ir jų skirtingas realizacijas.

4.2 Literatūros apžvalga

4.2.1 Probleminės srities apžvalga

Tyrimo objektas – loginė schema. Loginė schemos veikimo teisingumą užtikrina atliekami testavimai. Testavimai dažniausia atliekami aptikti dviejų tipų klaidoms: testai aptinkantys vėlinimo klaidas (angl. *delay faults*) [5,8] ir testai nustatantys konstantinio gedimo (angl. *stuck-at faults*) klaidas [7]. Šiame darbe bus nagrinėjamas testų, aptinkančių konstantines klaidas (toliau vartosime terminą schemos klaidos arba klaidos), ir schemų kelių sąryšis.

Sugeneruoti testai pasirinktai testų realizacijai privalo padengti visas galimas klaidas. Maksimalus testų skaičius visoms schemos realizacijos klaidoms padengti lygus kelių skaičiui esančiam realizacijoje (kiekvienas testas tikrina po vieną realizacijos kelią) [13]. Svarbesnis yra minimalus testų vektorių skaičius visoms schemos klaidoms aptikti. Žinant kelių skaičių tiriamoje schemoje yra lengviau surasti minimalų testų vektorių skaičių visų klaidų aptikimui [11].

Didelėse schemose elementų skaičius yra labai didelis. Taigi ir kelių tarp schemos įėjimų ir išėjimų gali būti labai daug. Kai kuriais atvejais schemos kelių perrinkimas nuo vieno įėjimo iki visų išėjimų yra labai ilgas procesas. Buvo sugalvoti kelių paskaičiavimo metodai jų neperrinkinėjant. Vienas iš būdų paskaičiuoti schemos kelius yra skaidymas didelės schemos į mažesnes schemeles. Suskaidytų schemų kelius paskaičiuoti perrenkant kelius. Bendras visos schemos kelių skaičius gaunamas sudauginant susijusių smulkesnių schemų kelių skaičius. Problema yra išskirti smulkesnes schemos dalis.

Kitas kelių paskaičiavimo metodas yra kiekvieno schemos ryšio įvertinimas pagal elementą iš kurio šis ryšys prasideda [6]. Šis metodas taip pat yra taikomas sudarant minimalią testų vektorių aibę

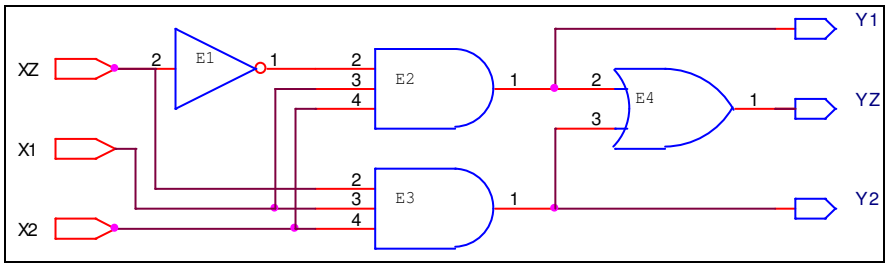
[11]. Tokį įvertinimą atliekanti procedūra taip pat nėra greita ir ji visiškai neįvertina kelyje esančių signalų invertavimo ir kelių padengimo.

Apytikslis kelių skaičiaus paskaičiavimas taip pat naudojamas. Tada nėra tikslumo, kuris reikalingas norint gauti santykį tarp kelių skaičiaus su testų vektorių skaičiaus. Tačiau apytikslis kelių skaičius naudingas generuojant minimalią testų vektorių aibę. Vienas iš būdų rasti minimalią tetų vektorių aibę yra aprašytas [6].

Šiame darbe buvo naudotas elementarus visų kelių perrinkimo algoritmas. Jis naudingas nes gaunama tiksli schemos kelių reikšmė. Šis algoritmas leidžia ne tik nustatyti schemos kelių skaičių, bet ir detalizuoti surastą kelią. Detalizavimo kriterijai yra: ar kelias yra lyginis/nelyginis, ar surastas kelias yra padengiantis.

Taip pat yra keletas minimalaus testų vektorių skaičiaus nustatymo metodų. Vienas iš algoritmų yra aprašytas [11]. Jis paskaičiuoja minimalų testų vektorių skaičių įvertindamas kiekvienam schemos ryšiui, tarp visų schemos elementų, reikalingą testų skaičių. Šis metodas yra pakankamai sudėtingas ir lėtas. Tačiau šiame darbe tirsime tik priklausomybę tarp jau turimų minimalių testų vektorių skaičiaus ir paskaičiuotų kelių skirtingoms schemos realizacijoms.

Nustačius tiriamos schemos skirtingų realizacijų kelių skaičių ir žinant minimalų testų vektorių skaičių šioms realizacijoms tiriama priklausomybė tarp testų skaičiaus ir kelių skaičiaus. Taip pat įdomus santykis kaip vienos realizacijos testai tinka kitos realizacijos testams, kai yra žinomas šių realizacijų kelių santykis. Lyginant kelių skaičius skirtingoms realizacijoms galima nustatyti fiktyvių kelių kiekį tiriamoje realizacijoje. Fiktyvaus kelio paaiškinimas 1 schemoje ir 1 lentelėje.



1. schema Schemos įėjimų sąryšis su išėjimais.

Paveiksle pavaizduotą loginę schemą realizuoja tokią funkciją (žiūrėti 1 formulę):

$$\begin{cases} Y_1 = \neg X_z \wedge X_1 \wedge X_2 \\ Y_2 = X_z \wedge X_1 \wedge X_2 \\ Y_z = (\neg X_z \wedge X_1 \wedge X_2) \vee (X_z \wedge X_1 \wedge X_2) \end{cases}$$

1. formulė 1 schemoje pavaizduotos loginės schemos loginė funkcija.

Vizualiai iš 1 schemos matome, kad ryšys tarp schemos įėjimo X_z iki išėjimo Y_z egzistuoja. Tai rodo ir Y_z loginė funkcija (1 formulė).

1 schemoje pavaizduotos loginės schemos teisingumo lentelė tokia (žiūrėti 1 lentelę):

1. lentelė 1 schemoje pavaizduotos loginės schemos teisingumo lentelė.

Nr.	**XZ**	**X1**	**X2**	**Y1**	**YZ**	**Y2**
1	0	0	0	0	0	0
2	0	0	1	0	0	0
3	0	1	0	0	0	0
4	***0***	***1***	***1***	***1***	***1***	***0***
5	1	0	0	0	0	0
6	1	0	1	0	0	0
7	1	1	0	0	0	0
8	***1***	***1***	***1***	***0***	***1***	***1***

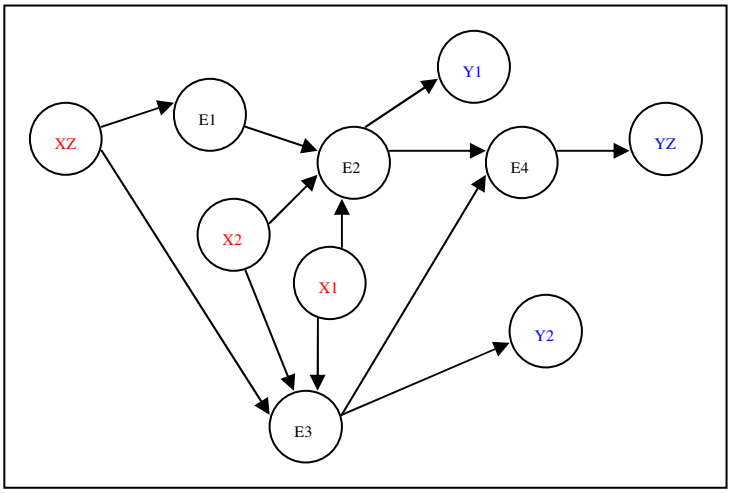
Paimkime 4 ir 8 eilutes iš 1 lentelės: 011 110 (4 lentelės eilutė) ir 111 011 (8 lentelės eilutė). Pirmi trys eilutės skaitmenys vaizduoja schemos įėjimus (011 ir 111), paskutiniai trys – išėjimus (110 ir 011). Sulyginę pasirinktų eilučių įėjimų vektorius, pastebime, kad jos skiriasi tik vienu elementu. Tokie įėjimų vektoriai, kurie skiriasi tik vienu elementu, vadinami *gretimi įėjimų vektoriai*. Pasirinktuose gretimuose įėjimų vektoriuose skiriasi tik X_z elemento reikšmės. Nors schemos įėjimai skirtingi, tačiau Y_z išėjimo reikšmė pastovi (vidurinis skaitmuo išėjimuose). Taigi ryšys X_z -> Y_z yra fiktyvus, nes įėjimo X_z pokytis nekeičia išėjimo Y_z reikšmės.

Sulyginant dvi tos pačios loginės funkcijos skirtingas realizacijas nustatoma, kiek yra tokių ryšių tarp realizacijų. Fiktyviems keliams aptikti nėra jokio specifinio algoritmo. Užtenka turėti lyginamų schemų visus ryšius tarp schemos įėjimų ir išėjimų.

4.2.2 Loginių schemų struktūros analizė

Loginių schemų struktūros analizė – tai schemos elementų ir ryšių tarp jų nustatymas. Į loginę schemą galima žiūrėti kaip į orientuotą grafą, kurio viršūnės tai loginiai elementai, o briaunos tai ryšiai tarp loginių elementų.

1 schemoje atvaizduotos loginės schemos struktūros grafas (žiūrėkite 2 schemą).



2. schema Loginės schemos (iš 1 schemos) atvaizdavimas grafu.

Schemos įėjimai X_1, X_2, X_z ir išėjimai Y_1, Y_2 ir Y_z yra atvaizduojami grafo viršūnėmis. Mus domina visi keliai nuo schemos įėjimų iki išėjimų.

Grafų teorijoje yra du pagrindiniai grafo padengimo modeliai [2]. Pirmas – grafo padengimas naudojantis paieška į plotį, antras – paieška į gylį. Šie du algoritmai detaliai aprašyti [10]. Kadangi šio darbo tikslas yra bendras kelių skaičiavimas, o ne trumpiausio ar ilgiausio kelio radimas, buvo pasirinka paieška į gylį. Paieška į gylį geresnė už paiešką į plotį tuo, kad tiriamu atveju mes žinome mus dominančio grafo pradžios viršūnes (schemos įėjimus) ir viršūnes, iki kurių turime nustatyti ryšius (schemos išėjimus). Paieškos vykdymas bet kokiu minėtu metodu didelėse schemose gali užtrukti labai ilgai. Didelėje loginėje schemoje paieška į gylį geresnė už paieška į plotį, nes vykdant paiešką į gylį kažkokius rezultatus visada gausime net nutraukę paieškos procesą, o paieška į plotį gali neduoti net menkiausių rezultatų įvykdžius nutraukimą.

5. Projektinė dalis

5.1 Sistemos reikalavimai

5.1.1 Funkciniai reikalavimai

- ✓ Pagrindinė sistemos paskirtis skaičiuoti kelius VHDL loginėse schemose.
- ✓ Turi būti galimybė sulyginti dviejų schemų paskaičiuotų kelių rezultatus.
- ✓ Sistema turėtų leisti įkrauti MR failus.
- ✓ Turi būti galimybė sulyginti pasirinktos schemos paskaičiuotų kelių rezultatus su MR failo duomenimis.
- ✓ Sistema privalo turėti komandinės eilutės ir patogią grafinę vartotojo sąsają.
- ✓ Sistema privalo leisti keisti sistemos nustatymus.

5.1.2 Nefunkciniai reikalavimai

- ✓ Sistema turi būti realizuota aukšto lygio programavimo kalba.
- ✓ Sistemos architektūra privalo būti aiški ir lengvai praplečiama esant naujam funkcionalumui.
- ✓ Norint pakeisti sistemos vartotojo sąsajos kalbą tai nesukeltų didelių nesklandumų.
- ✓ Sistema turi būti intuityviai aiški.
- ✓ Turi būti galimybė bet kuriuo momentu nutraukti sistemos darbą.
- ✓ Sistema turi naudoti minimalius OS resursus.

5.2 Sistemos realizavimo žingsniai

Pirmas žingsnis buvo nuspręsti, kokia programavimo kalba sistema bus realizuota. Buvo pasirinkta aukšto lygio objektinio programavimo kalba JAVA. Šiuo metu JAVA yra viena populiariausių objektinių kalbų. Ji yra pakankamai paprasta, gerai dokumentuota ir turinti daugybę specifinių paketų. Dar vienas privalumas tai, jog JAVA nepriklauso nuo OS.

VHDL kalbos sintaksės analizė buvo svarbus sistemos kūrimo etapas. Jo metu buvo išsiaiškintos esminės VHDL kalbos struktūros aprašančios logines schemas. Buvo remtasi standarte IEEE 1076-1993 aprašyta VHDL struktūrų sintakse.

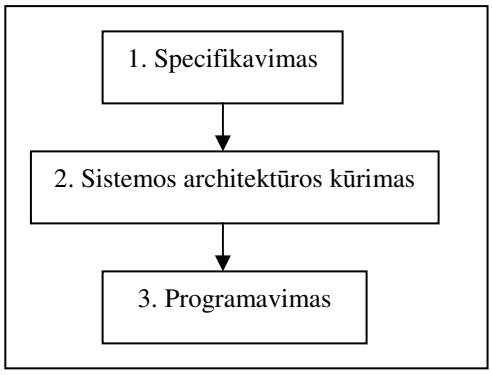
Tam, kad struktūrizuotus VHDL duomenis užkrauti į JAVA objektus, reikalinga gerai sudaryta sistemos architektūra. Schemos architektūra buvo kuriama panašiomis struktūromis, kokios yra naudojamos VHDL schemų aprašymuose. Buvo išskirtos pagrindinės VHDL kalbos struktūros naudojamos schemų aprašyme, tokios kaip: *Entity*, *Port*, *PortMap* ir kitos. Išsamiai apie sistemos architektūra skyriuje **5.4 Sistemos architektūra**.

Turint nuskaitytų duomenų struktūras reikėjo sukurti algoritmą, kuris atliktų struktūros analizę. Kaip jau buvo minėta, kiekvieną loginę schemą galima atvaizduoti grafu. Iš nuskaitytų duomenų buvo sudarytas grafas, kuriame taikytas paieškos į gylį metodas tam, kad paskaičiuoti schemos kelius. Detalus naudoto algoritmo taikymas įkrautai VHDL schemos duomenų struktūrai skyriaus **5.4 Sistemos architektūra** dalyje **5.5.2 Sekos diagramos**.

Rezultatus vaizduoti buvo nuspręsta HTML formatu. Šis formatas užtikrina, kad rezultatai bus matomi neatsižvelgiant į OS.

5.3 Sistemos kūrimo procesai

Sistema buvo kuriama išskaidžius visą procesą į tokias dalis (žiūrėkite 1 diagramą):



1. diagrama Sistemos kūrimo procesai.

1 diagramoje matosi pagrindiniai sistemos kūrimo procesai. Detalizuosime kiekvieną kūrimo proceso dalį.

1. Specifikavimo dalyje buvo surinkta visa reikalinga informacija apie kuriamos sistemos funkcinius ir nefunkcinius reikalavimus. Buvo išanalizuota VHDL failų struktūra ir išskirtos esminės VHDL schemų aprašymo dalys. Išskirti reikalingi sistemos nustatymų parametrai reikalingi sistemos darbui.
2. Sistemos architektūros kūrimo proceso metu buvo modeliuojama sistemos architektūra. Modeliavimas

buvo atliekamas naudojant UML [1,3]. Modeliuojant architektūrą buvo stengtasi įvesti klasių apibendrinimus ir išskirti tas vietas architektūroje kurios gali būti naudojamos norint pridėti naują funkcionalumą. UML modeliavimas susidėjo iš:

- ✓ Modeliuojant buvo išskirti sistemos panaudojimo atvejai. Panaudojimo atvejai atspindi pagrindinius sistemos atliekamus veiksmus.
- ✓ Buvo sudaryta klasių diagrama. Klasių diagramą sudarė duomenų saugojimo struktūros (susijusių klasių aibė, kuriuose išsaugomi nuskaityti duomenys) ir klasės reikalingos duomenų struktūroms užpildyti (aibė klasių, kurios vykdo duomenų struktūrų užpildymą duomenimis).
- ✓ Veiksmų sekos diagramos atspindi kaip ir kokie veiksmai bus vykdomi atliekant sistemos panaudojimo atvejų operacijas.
- ✓ Būsenos diagramos vaizduoja kaip siejasi specifinių algoritmų arba pačios sistemos būsenos ir veiksmai pasiekti tas būsenas.

3. Programavimas arba sistemos realizavimas pasirinkta programavimo kalba. Šio etapo metu buvo realizuojama apibrėžta sistemos architektūra. Išskirtos pagal prasmę klasės sudėtos į skirtingus paketus (bibliotekas). Schemų analizės rezultatai buvo gaunami JAVA objektuose kuriuos reikėjo pateikti vartotojui HTML formate. Rezultatų formavimui JAVA objektai buvo surašomi į XML failą, o paskui naudojant standartinius XML analizatorius (angl. *XML parsers*) XML failai konvertuojami į HTML‘ą.

5.4 Naudotos technologijos ir įrankiai

Sistema buvo kurta naudojant JAVA programavimo kalbą. Pasirinkta ir naudota 1.4.2_03 JAVA kompiliatoriaus versija.

Modeliuojant kuriamos sistemos architektūrą buvo pasirinkta MagicDraw UML braižymo sistema. Ši sistema leidžia ne tik nubraižyti UML diagramas, bet kai kurias jų konvertuoti į realias JAVA klases/objektus.

Kuriant sistemą buvo naudota patogi JAVA projektų kūrimo aplinka IntelliJ. Ši aplinka leidžia greitai ir patogiai į kuriamą projektą įtraukti ir naudoti papildomas bibliotekas. Aplinkoje integruota patogi sąsaja kurti ir redaguoti kuriamos sistemos vartotojo grafinę aplinką.

Rezultatų formavimui iš JAVA objektų į XML buvo naudotas XPP analizatorius. Jo privalumas tas, kad jis kurdamas XML dokumentą nereikalauja daug OS resursų.

5.5 Sistemos architektūra

Sistemos architektūra buvo modeliuojama naudojant UML metodus. Buvo išskirti sistemos panaudojimo atvejai (skyrius 5.5.1 Sistemos panaudojimo atvejai). Sekų diagramos – rodančios kaip sistemos objektai susiję tarpusavyje (skyrius 5.5.2 Sekos diagramos). Būsenos diagramos atspindinčios, kaip ir į kokias būsenas po kokių veiksmų pereina kuriama sistema (skyrius 5.5.3 Būsenos diagramos).

5.5.1 Sistemos panaudojimo atvejai

UML panaudojimo atvejų diagrama naudojama apibūdinti sistemos funkcionalumui. Panaudojimo atvejai skiriasi nuo sekų ir būsenos diagramų, nes jie nevaizduoja eiliškumo sistemos vykdymo veiksmų, o tik abstrakčiai juos apibrėžia.

Panaudojimo atvejų diagramoje svarbu išskirti veiksmų (funkcijų) aktorius, kurie galės atlikinėti tam tikrus veiksmus. Kiekvienas aktorius gali atlikti tik jam nurodytus veiksmus panaudojimo atvejų diagramoje.

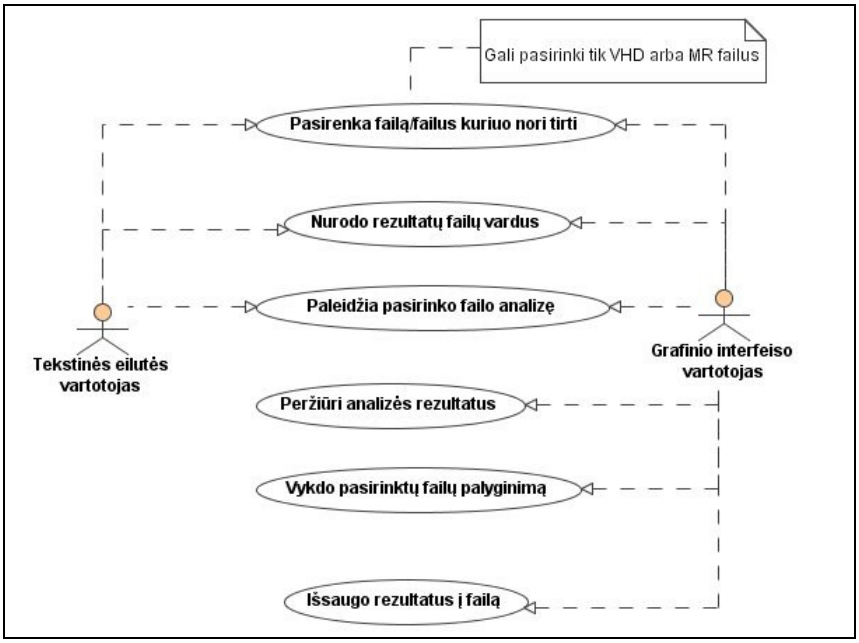
Išskirti kuriamos sistemos aktoriai (veikėjai):

- ✓ komandinės eilutės vartotojas,
- ✓ grafinės sąsajos vartotojas.

Sistemos aktorių PA:

- ✓ pasirinkti failą/failus tyrimui,
- ✓ nurodo rezultatų failų vardus,
- ✓ paleidžia pasirinko failo analizę,
- ✓ peržiūri analizės rezultatus,
- ✓ vykdo pasirinktų failų palyginimą,
- ✓ išsaugo rezultatus į failą.

Sistemos aktorių ir jų panaudojimo atvejų grafinis atvaizdavimas matomas 2 diagramoje.



2. diagrama Sistemos panaudojimo atvejai

Kiekvieno panaudojimo atvejo iš 2 diagramos paaiškinimams naudota tokia struktūra:

1. Panaudojimo atvejis – tai diagramoje pavaizduoto PA aprašymas.
2. Aprašas – trumpas PA aprašymas.
3. Aktorius – nurodyti aktoriai, kurie susiję su nagrinėjamu panaudojimo atveju.
4. Išankstinė sąlyga – sąlyga, kuri turi būti įvykdyta, prieš atliekant nagrinėjamą PA.
5. Sužadinimo sąlyga – sąlyga kuri iššaukia nagrinėjamą PA.
6. Tolimesnis galimas veiksmas – veiksmas, kuris leistinas atlikus nagrinėjamą PA.

PA paaiškinimai:

Naudojami sutrumpinimai reikalingi atskirti skirtingų vartotojų sąlygoms:

- ✓ KEV – komandinės eilutės vartotojas
- ✓ GSV – grafinės sąsajos vartotojas

1.

Panaudojimo atvejis:	Failo/failų tyrimui pasirinkimas
Aprašas:	Aktoriai nurodo kokį failą/failus reikia analizuoti.
Vartotojas/Aktorius:	KEV, GSV
Išankstinė sąlyga:	KEV privalo žinoti tikslų kelia iki norimo tirti failo. GSV privalo būti pasileidęs sistemos grafinį interfeisą.
Sužadinimo sąlyga:	GSV paleidžia failų pasirinkimo dialogą.
Tolimesnis galimas veiksmas:	GSV mato schemos vizualų vaizdą.

2.

Panaudojimo atvejis:	Rezultatų failų vardų nurodymas
Aprašas:	Aktoriai nurodo į kokį failą reikia saugoti rezultatus.
Vartotojas/Aktorius:	KEV, GSV
Išankstinė sąlyga:	KEV privalo žinoti tikslų kelia iki rezultatų failo. GSV privalo būti pasileidęs sistemos grafinį interfeisą.
Sužadinimo sąlyga:	GSV paleidžia failų saugojimo dialogą.
Tolimesnis galimas veiksmas:	GSV jau gali matyti rezultatus nurodytame faile.

3.

Panaudojimo atvejis:	Pasirinko failo analizės paleidimas
Aprašas:	KEV paleidžia vykdyti pagrindinę sistemos klasę su nurodytais parametrais. GSV pasirenka punktą „vykdyti analizę“ grafiniame interfeise.
Vartotojas/Aktorius:	KEV, GSV
Išankstinė sąlyga:	KEV privalo per parametrus nurodyti tiriamą failą. Taip pat privalo nurodyti kur bus saugomi rezultatai. GSV privalo turėti atidarytą ir aktyvų nors vieną VHD failą.
Sužadinimo sąlyga:	KEV iš komandinės eilutės paleidžia vykdomąjį failą. GSV pasirenka punktą „vykdyti analizę“ grafiniame interfeise. Šis punktas matomas meniu juostoje ir įrankių juostoje.
Tolimesnis galimas veiksmas:	KEV jau gali matyti rezultatus nurodytame faile. GSV mato vizualų schemos vaizdą. Gali vizualiai tyrinėti atskirų schemos įėjimų/išėjimų sąryšius su išėjimais/įėjimais.

4.

Panaudojimo atvejis:	Analizės rezultatų peržiūra
Aprašas:	Aktorius gali matyti analizės rezultatus vizualiai.
Vartotojas/Aktorius:	GSV
Išankstinė sąlyga:	Privalo būti atidarytas MR failas arba atidarytas ir išanalizuotas VHD failas.
Sužadinimo sąlyga:	Vartotojas gali schemos brėžinyje pažymėti įėjimą/išėjimą ir matyti ryšius su išėjimais/įėjimais.
Tolimesnis galimas veiksmas:	Vartotojas gali vykdyti rezultatų saugojimą.

5.

Panaudojimo atvejis:	Pasirinktų failų palyginimo vykdymas
Aprašas:	Vykdomas dviejų schemų sulyginimas ir randami skirtumai tarp schemų.
Vartotojas/Aktorius:	KEV, GSV
Išankstinė sąlyga:	KEV privalo per parametrus nurodyti tiriamus failus kuriuos norės sulyginti. Taip pat privalo nurodyti kur bus saugomi rezultatai. GSV privalo turėti atidarytas bent dvi vienodą įėjimų ir išėjimų skaičių turinčias schemas. Atidarytos VHD schemos privalo būti išanalizuotos.
Sužadinimo sąlyga:	KEV iš komandinės eilutės paleidžia vykdomąjį failą su nurodytais parametrais. GSV pasirenka punktą „vykdyti palyginimą“ grafiniame interfeise. Šis punktas matomas meniu juostoje ir įrankių juostoje.
Tolimesnis galimas veiksmas:	Nurodytame rezultatų faile galima matyti rezultatus.

6.

Panaudojimo atvejis:	Rezultatų išsaugojimas į rezultatų failą
Aprašas:	Rezultatai saugomi iš JAVA objektų į vartotojui patogų HTML. GSV rezultatus gali išsaugoti sistemos vykdymo metu pasirinktame faile.
Vartotojas/Aktorius:	GSV
Išankstinė sąlyga:	Privalo būti atidarytas nors vienas MR failas, arba atidarytas ir išanalizuotas VHD failas. Jeigu vykdomas palyginimas ir reikia nurodyti rezultatų failą, privalo būti atidarytos bent dvi vienodą įėjimų ir išėjimų skaičių turinčias schemas. Atidarytos VHD schemos privalo būti išanalizuotos.
Sužadinimo sąlyga:	GSV pasirenka punktą „pasirinkti failą“ grafiniame interfeise. Šis punktas matomas palyginimų dialoge.
Tolimesnis galimas veiksmas:	Jeigu buvo saugoma tik struktūra, rezultatus jau galima matyti rezultatų faile. Jeigu buvo nurodomas palyginimo rezultatų failo vardas – tai jau galima atlikinėti palyginimo operaciją.

5.5.2 Veiksmų sekos diagramos

Veiksmų sekos diagramų (toliau sekų diagramos) modelis vizualiai parodo veiksmų seką sistemoje. Tokios diagramos skirtos dokumentuoti ir patikrinti veiksmų logiką sistemoje.

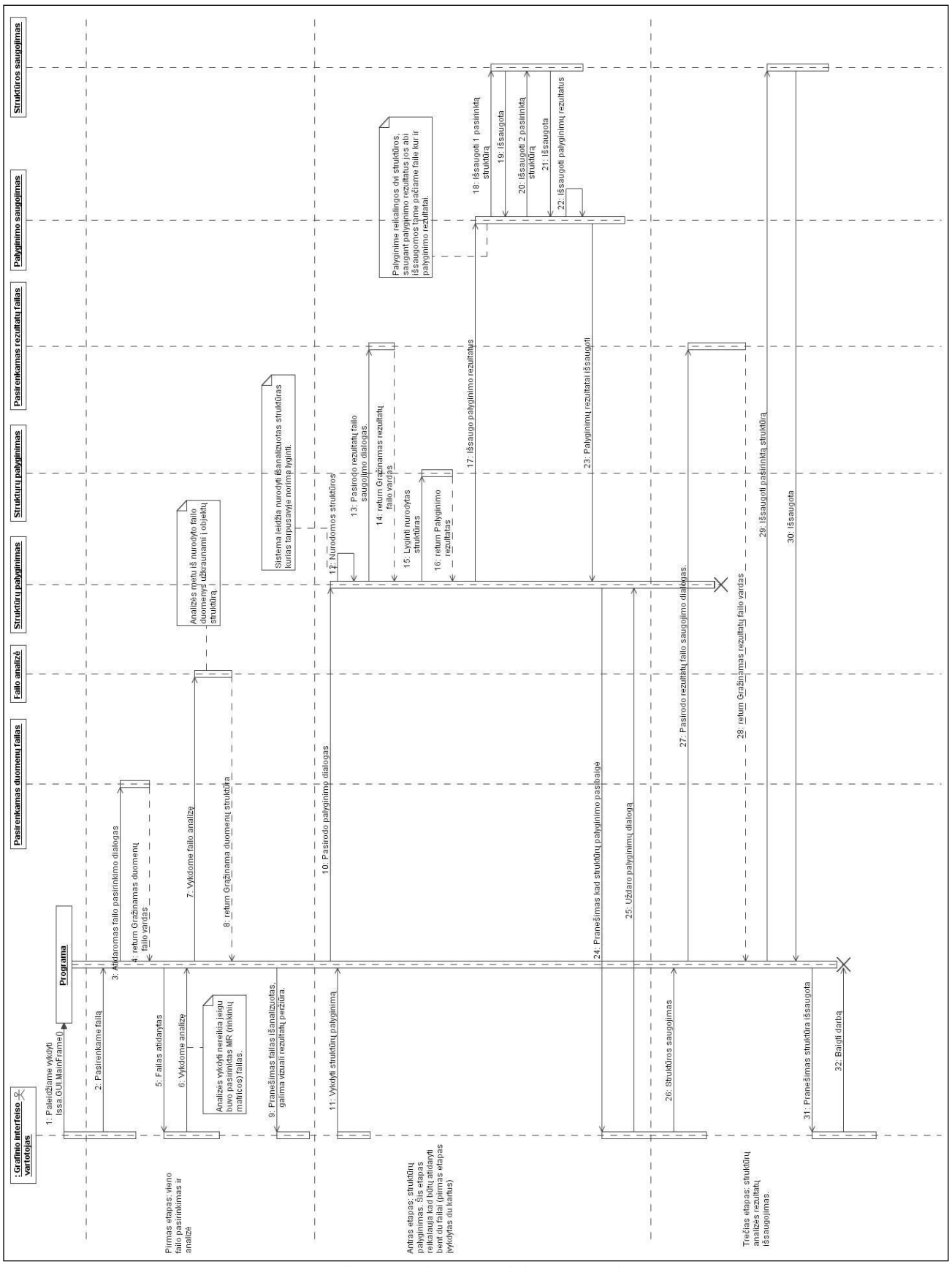
Sekų diagramos dažniausia skirtos modeliuoti:

- ✓ Naudojimo scenarijus – tai būdas pavaizduoti, kaip naudojama sistema. Scenarijaus logiką gali sudaryti dalis panaudojimo atvejų.
- ✓ Metodų logiką, kuri skirta pavaizduoti sudėtingų operacijų (funkcijų arba procedūrų) veikimo logiką.

- ✓ Teikiamų paslaugų logika – paslauga tai aukštesnio lygio operacija, kuri skiriasi skirtingoms vartotojų grupėms.

5.5.2.1 Sistemos veiksmų sekos diagramos

Sistemos veiksmų diagrama – tai sistemos naudojimo scenarijus. Ši diagrama vaizduoja kaip galima naudoti sistemą ir kokia eilės tvarka turėtų vykti veiksmai. Žiūrėkite 3 diagramą.

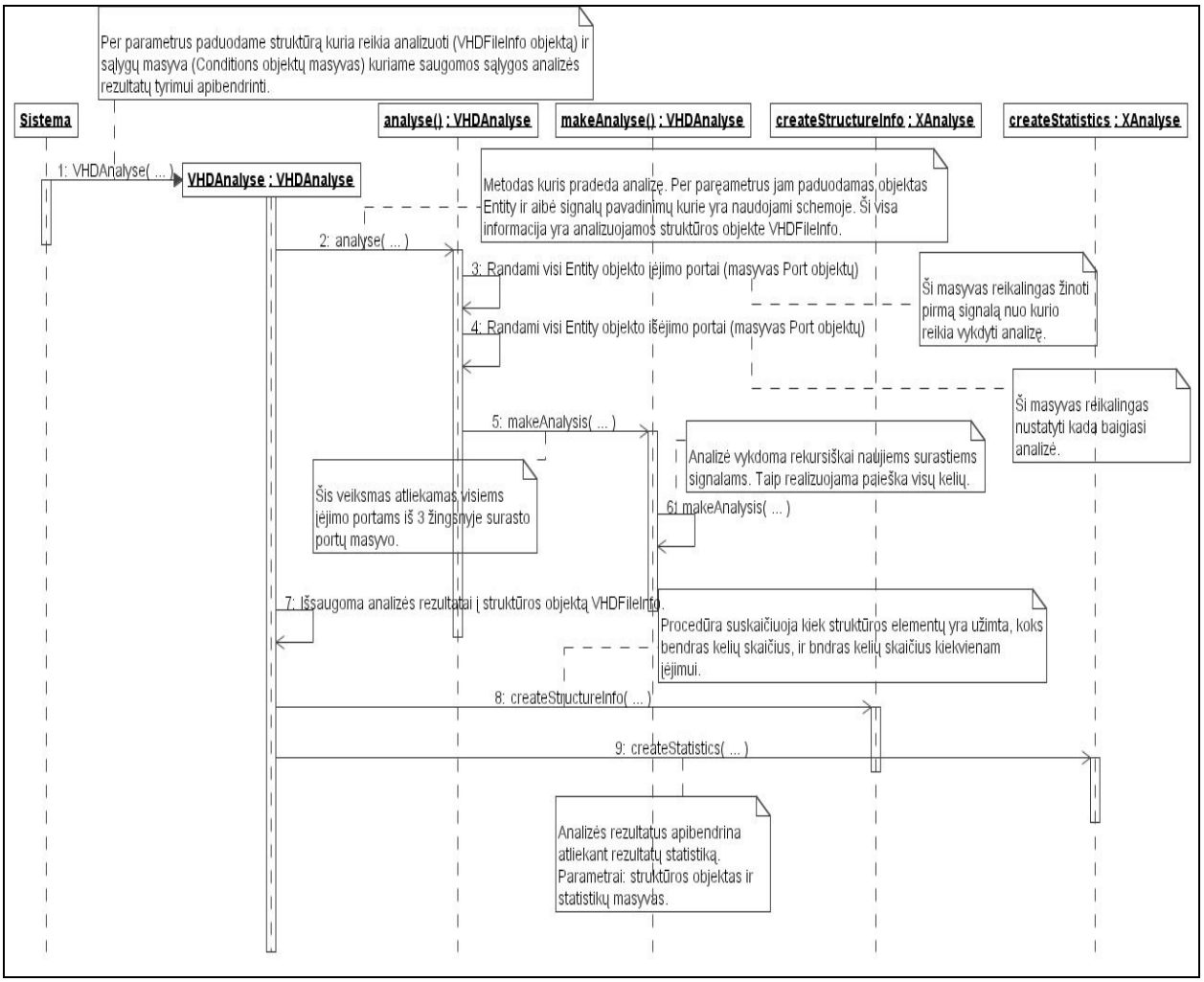


3. diagrama Sistemos veikimo sekos diagrama.

3 diagrama vaizduoja abstrakčią veiksmų seką, neišskiriant konkrečių architektūros klasių ar metodų.

5.5.2.2 Analizės algoritmo veiksmų sekos diagrama

Analizės veiksmų sekos diagrama atvaizduoja analizės operacijos veiksmų seką abstrakčiame lygmenyje. Ši diagrama susieja konkrečius veiksmus, atliekamus šios operacijos metu, bet nedetalizuoja jų (4 diagrama).

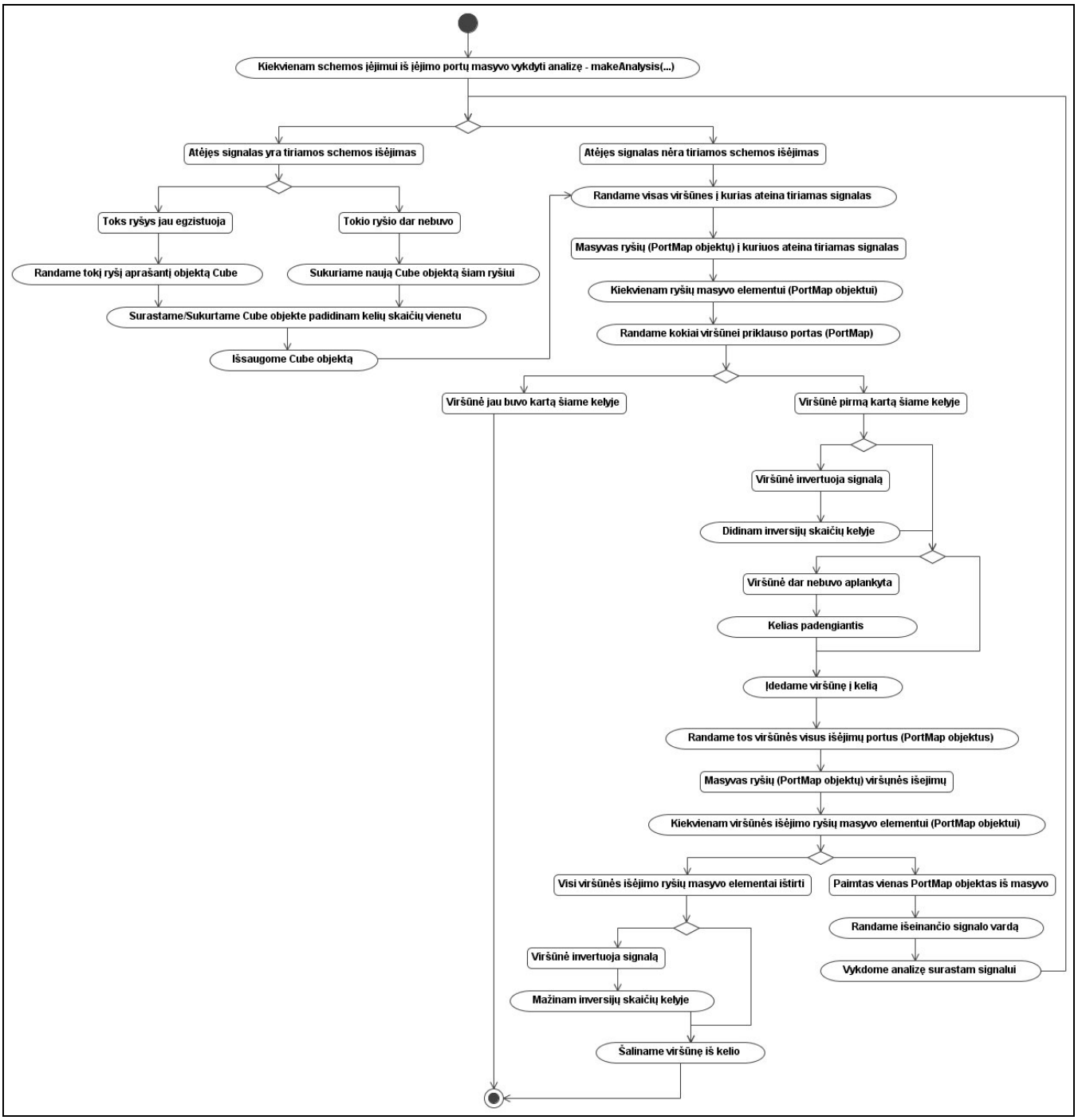


4. diagrama Bendra analizės veikimo sekos diagrama.

4 diagramoje pavaizduota, kokia veiksmų seka vykdoma įkrautų VHDL schemos duomenų analizė, ieškant schemos kelių. Schemos keliai ieškomi rekursiškai – paieška į gylį.

5.5.3 Analizės algoritmo veiklos diagrama

Veiklos diagramos apibūdina darbo procesą sistemoje arba sistemos dalyje. Schemos kelių paieškos algoritmo būsenų sąryšis matomas 5 diagramoje. Algoritmas atspindi kaip vykdoma paieška į gylį grafų teorijoje.

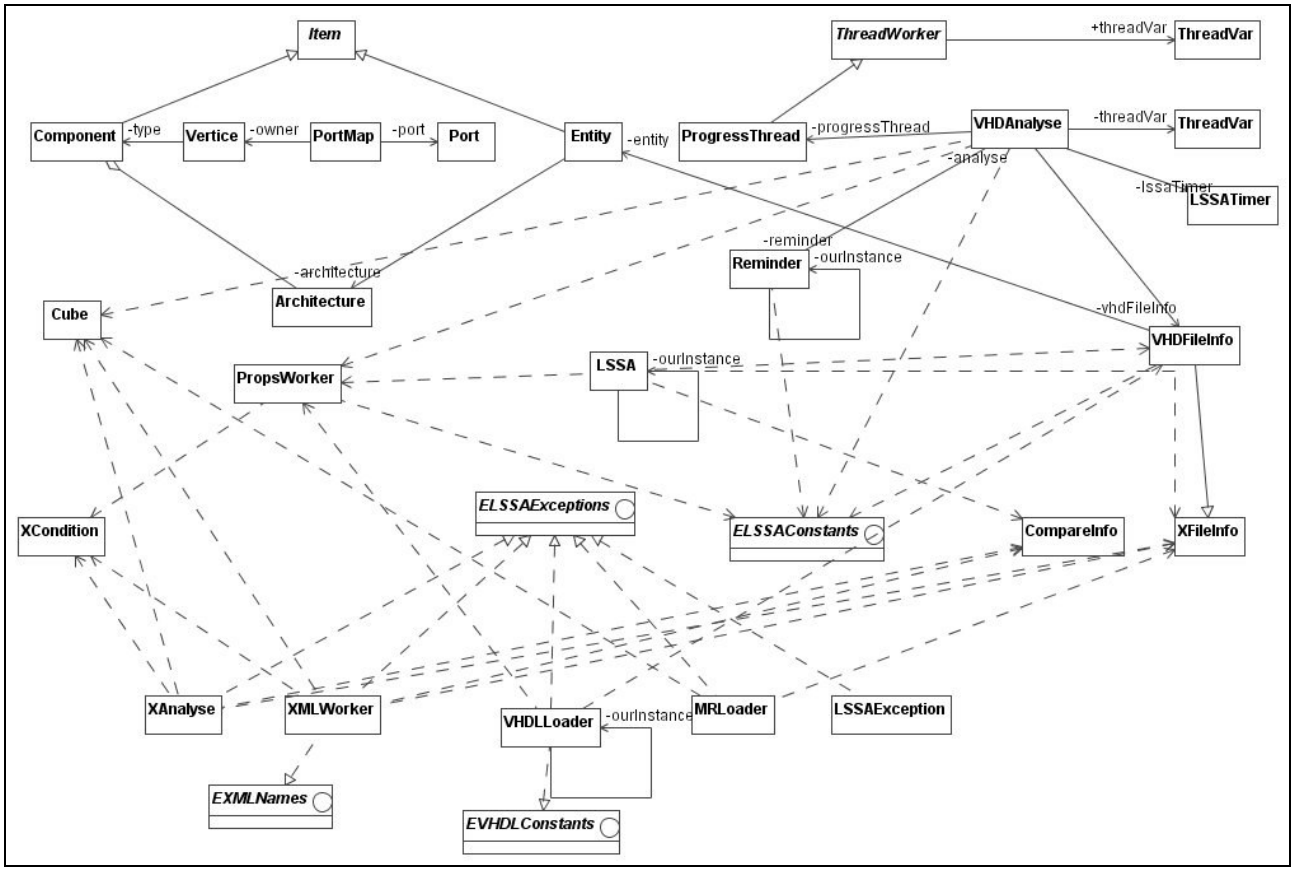


5. diagrama Kelių paieškos algoritmo veiklos diagrama

5.5.4 Sistemos klasių diagrama

Klasių diagrama modeliuoja klasių struktūras, naudodama tokius dizaino elementus: klases, paketus ir objektus. Ši diagrama atspindi ryšius tarp dizaino elementų.

6 diagramoje pavaizduotas klasių sąryšis atskleidžia sistemos logiką realizuojančias klases. Šiame paveiksle matomos visos klasės, reikalingos įkrauti, analizuoti ir išsaugoti analizuojamas VHDL schemas.



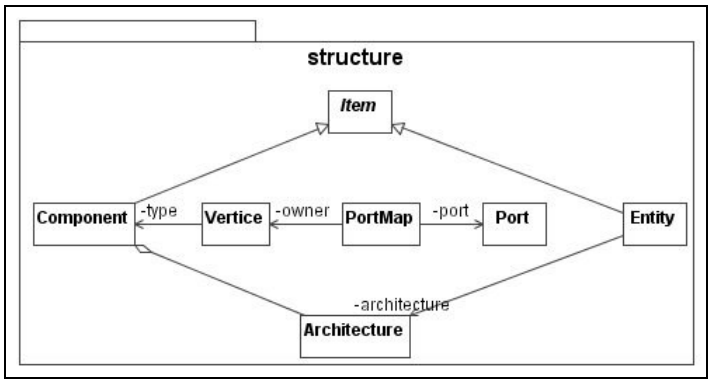
6. diagrama Klasių diagrama atvaizduojanti klasių sąryšius.

Detalus kiekvienos sistemos klasės aprašymas yra priede **B**.

5.5.5 Išskirtos duomenų struktūros

Šiame skyriuje bus aptarta struktūra, kuri yra naudojama įkrautų duomenų saugojimui JAVA objektuose.

Kad suprasti paieškos į gylį algoritmą įkrautose VHDL schemose reikia turėti pilną informaciją, kaip grafo viršūnės (įkrautos schemos elementai) susiję tarpusavyje. Tam reikia žinoti duomenų saugojimo struktūrą. Nuskaitytų VHDL schemos duomenų saugojimo struktūra JAVA objektuose pavaizduota 7 diagramoje.



7. diagrama Nuskaitytos VHDL schemos duomenų išsaugojimo JAVA objektuose struktūra.

7 diagramoje pavaizduota struktūra rodo, kaip saugoma VHDL schemos informacija. Schemą atspindi klasė *Entity*, schemoje naudoti abstraktūs elementai, aprašyti *Component* klase. *PortMap* klasė vaizduoja, kaip susiję schemos elementai (komponentai) su signalais. Schemos signalai saugomi *Architecture* klasėje. Abstrakti klasė *Item* simbolizuoja VHDL faile esančias struktūras. Struktūros gali būti: visa schema – klasė *Entity* ir schemoje naudoti abstraktūs komponentai (schemos sudedamosios dalys – angl. *gate*) – klasė *Component*. Klasė *Vertice* (viršūnė), tai konkrečios komponentės panaudojimas. Kiekviena abstrakti komponentė privalo turėti bent dvi jungtis (angl. *port*). Kiekvieną jungtį atspindi klasė *Port*.

5.6 Sistemos nustatymų ir pagalbiniai failai

Sistemoje yra naudojami keli nustatymų failai ir vienas pagalbinis failas. Nustatymų failai skirti tam, kad vartotojui paleidus sistemą ir įvykdžius kažkokius veiksmus veiksmų iššaukti pakeitimai atsispindėtų kitą kartą vartotojui paleidus sistemą.

Sistemoje reikalingi šie nustatymų failai: *lssa.properties* ir *inversion.properties*. Šiuose failuose kiekvienas vartotojas pagal poreikius, gali išsaugoti savo nustatymų sistemos parametrus. Sekantį kartą paleidus sistemą bus atstatomi nustatyti parametrai.

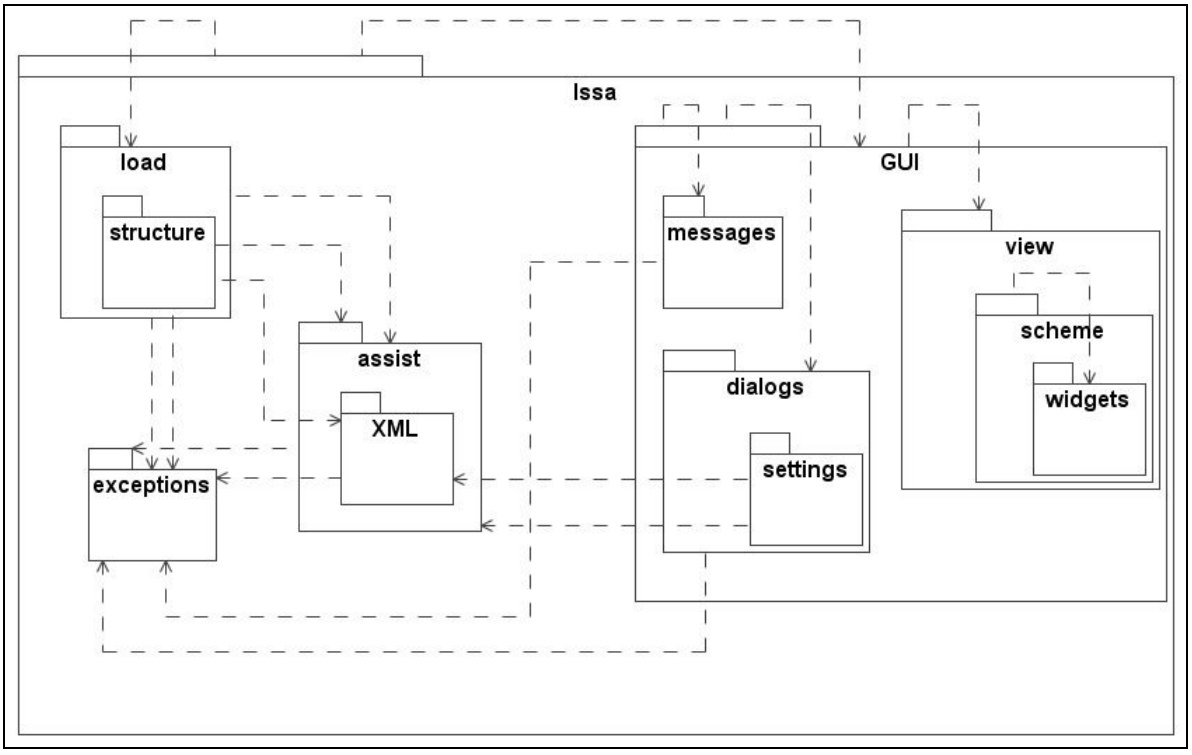
Faile *inversions.properties* yra saugomi schemos elementų vardai, kurie vykdo signalo inversiją.

Faile *lssa.properties* saugoma:

- ✓ Rezultatų apibendrinimo sąlygos, kurios reikalingos ištirtos schemos rezultatams apibendrinti.
- ✓ Paskutinė atidaryto ir išsaugoto failo direktorijos.
- ✓ Laiko trukmė, kiek ilgiausia gali užtrukti analizė. Jeigu analizė tęsiasi ilgiau nei nurodyta šiame faile, ji automatiškai nutraukiama.
- ✓ Reikšmė nusakanti ar reikia rodyti kelių pasiskirstymo matricą suformuotame HTML rezultatų faile.

5.7 PĮ priežiūra ir palaikymas

Sistemos architektūra buvo realizuota taip, kad jos praplėtimas ir palaikymas nebūtų daug resursų ir analizės reikalaujantis procesas. Architektūrą sudarančios klasės buvo išskaidytos į paketus pagal prasmę (žiūrėkite 8 diagramą):



8. diagrama Sistemos paketų ryšių diagrama.

Kiekvieno paketo, pavaizduoto 8 diagramoje, aprašymas:

- ✓ *lssa* yra pagrindinis paketas kuriame saugoma visa projekto informacija. Šiame pakete esančios klasės valdo sistemos darbą.

- ✓ *lssa.load* pakete saugomos klasės, kurios reikalingos informacijos saugojimui JAVA objektuose. Pakete realizuotos klasės yra skirtos tik duomenų saugojimui atlikus analizę, bet ne nuskaičiuos VHDL schemos duomenis. Nuskaitytų VHDL schemos duomenų saugojimui yra skirtas *lssa.load.structure* paketas.
- ✓ *lssa.load.structure* paketo klasės saugo konkrečios nuskaitytos VHDL schemos informaciją. Paketo klasės išskirtos iš *lssa.load* paketo, nes yra naudojamos tik nuskaitytų VHDL schemų duomenų saugojimui.
- ✓ *lssa.assist* paketas skirtas saugoti klasėms, kurios atlieka operacijas su duomenimis. Paketo klasės atlieka įkrautų VHDL schemų analizę ir sulyginimą.
- ✓ *lssa.assist.XML* paketo klasės formuoja XML failus iš JAVA objektuose saugomos informacijos.
- ✓ *lssa.exceptions* paketas saugo klases, vykdančias iškilusių klaidų apdorojimą sistemoje. Čia saugomi visi galintys pasirodyti klaidų pranešimai.
- ✓ *lssa.GUI* pakete saugomos klasės kuriančios grafinę vartotojo sąsają.
- ✓ *lssa.GUI.messages* – tai paketas, kuriame formuojami sistemoje naudojami maži dialogai (pvz. „Toks failas egzistuoja. Vykdyti išsaugojimą?“ dialogo pasirinkimai Taip ir Ne).
- ✓ *lssa.GUI.dialogs* – dialogų paketas. Šiame pakete saugomos klasės, kurios vaizduoja „Sulyginimo“ ir „Darbo proceso“ dialogus.
- ✓ *lssa.GUI.dialogs.settings* – tai paketas, kuriame saugomos klasės atvaizduojančios nustatymų dialogą. Šiame dialoge galima pasirinkti ir nustatyti įvairius sistemos parametrus, kurie bus išsaugoti nustatymų failuose.
- ✓ *lssa.GUI.view* – tai paketas, kuris realizuoja užkrautos loginės schemos grafinį vaizdą.
- ✓ *lssa.GUI.view.scheme* – šio paketo klasės braižo schemą kaip „juodą dėžę“.
- ✓ *lssa.GUI.view.scheme.widgets* paketo klasės simbolizuoja po vieną grafinį elementą, naudojamą schemos grafiniam atvaizdavimui.

Numatyti schemos pakeitimai buvo šie:

- ✓ Nuskaityti ir analizuoti kitokio tipo aprašus nei VHDL schemos aprašai.
- ✓ Pakeisti grafinio vartotojo sąsajos kalbą.

Kad analizuoti kitokio tipo aprašus nei VHDL schemų, reikia tik parašyti atskirą analizatorių, kuris užkrautų schemos duomenis į minėtą struktūrą. Taigi reikalinga viena papildoma klasė, kuri tai atliktų. Tokią klasę reiktų talpint *lssa.load* pakete. Šiame pakete jau yra dvi panašios klasės tai: *MRLoader* ir

VHDLLoader.

Norint pakeisti grafinio vartotojo sąsajos kalbą, reikėtų išversti grafinės sąsajos elementų (pvz. mygtukai, meniu punktai ir pan.) pavadinimus į pasirinktą kalbą. Visi grafinės sąsajos elementų pavadinimai saugomi JAVA interfeisuose[1]:

- ✓ *lssa.GUI.EGUIConstants;*
- ✓ *lssa.GUI.dialogs.ECompareConstants;*
- ✓ *lssa.GUI.dialogs.ERunDialogConstants;*
- ✓ *lssa.GUI.dialogs.settings.ESettingsDialogConstants;*
- ✓ *lssa.GUI.view. EViewConstants;*

Taip pat reiktų išversti klaidų pranešimus ir žinučių pranešimus, esančius JAVA interfeisuose:

- ✓ *lssa.GUI.messages. EMessageConstants;*
- ✓ *lssa.exceptions. ELSSAExceptions;*

Tačiau tai nėra patogus būdas keisti vartotojo grafinės sąsajos kalbą. Sistema būtų kur kas lankstesnė, jeigu visa informaciją, kurią turime keisti, siekiant įvesti naują kalbą, būtų saugoma nustatymų faile. Tačiau toks funkcionalumas nebuvo realizuotas, tai galėtų būti pirmas žingsnis tobulinant sistemą.

[1] JAVA interfeisas – tai specifinis JAVA kalboje naudojamas objektų tipas.

6. Eksperimentinė dalis

Eksperimentui buvo naudotos ISCAS'85 schemos ir jų skirtingos realizacijas. Kiekviena tirta schema turėjo po tris skirtingas realizacijas. Tos pačios schemos ir jų realizacijos buvo tiriamos darbe [4]. Remiantis [4] gautais rezultatais (2 ir 3 lentelės), darbe siekiama rasti priklausomybes tarp realizacijų testų skaičiaus ir kelių skaičiaus. Naudojantis sukurta programine įranga, atliekama visų schemų realizacijų struktūros analizė. Randami kelių skaičiai kiekvienos schemos realizacijai, išskiriant kiek buvo *lyginių*, kiek *nelyginių* kelių. Lyginiai ir nelyginiai keliai taip gali būti *padengiantys* arba *nepadengiantys*. Ši informacija gaunama atlikus analizę. Schemos realizacijų tyrimo rezultatai yra pateikiami priede **C**.

Atliekamas skirtingų schemų realizacijų palyginimas. Atlikus šį palyginimą, nustatoma kiek skiriasi realizacijų kelių skaičius, kiek viena realizacija turi ryšių, kurių kita realizacija neturi. Ieškoma fiktyvių ryšių tarp schemos realizacijų ir MR matricos. Kadangi tos pačios schemos realizacijos atlieka vienodą loginę funkciją, reiškia ir jų teisingumo lentelės vienodos. Egzistuojantis ryšys vienoje realizacijoje, tačiau ryšio nebuvimas kitoje, reiškia, kad surastas ryšys nekeičia teisingumo lentelės. Tokiam ryšiui nereikia generuoti testo ir jis vadinamas fiktyviu.

6.1 Testai tiriamose realizacijose

Testai buvo paimti iš [4]. Šio straipsnio autoriai analizavo tas pačias schemas ir jų realizacijas, bei nustatinėjo testų tinkamumą tarp skirtingų realizacijų.

2. lentelė Klaidų skaičius skirtingose schemos realizacijose.

Gedimų skaičius realizacijose			
Schema	R1	R2	R3
C432	507	426	460
C499	750	978	1246
C880	942	857	928
C1355	1566	1316	1406
C1908	1862	876	1224
C2670	1990	1500	1658
C3540	3126	2474	2520
C5315	5248	3879	4130
C6288	7638	6680	7498
C7552	7039	4570	4798

Stulpelių paaiškinimai:

- ✓ Schema – schemos pavadinimas;
- ✓ R1 – standartinė schemos realizacija;
- ✓ R2 – persintezuota schemos realizacija, naudojant biblioteką – *class.db*;
- ✓ R3 – persintezuota schemos realizacija, naudojant biblioteką – *and_or.db*;

2 lentelė rodo, kiek yra pastovios būsenos klaidų (angl. *stuck-at fault*) kiekvienos schemos realizacijose.

Lentelėje 3 yra testų skaičius kiekvienai tirtai schemos realizacijai. Šis testų skaičius patikrina visus pastovios būsenos gedimus, nurodytai schemos realizacijai.

3. lentelė Testo vektorių skaičius schemos realizacijoms.

Testų vektorių skaičius			
Schema	**R1**	**R2**	**R3**
C432	57	46	45
C499	54	74	80
C880	62	49	50
C1355	86	83	80
C1908	118	57	75
C2670	105	120	116
C3540	167	143	147
C5315	130	99	89
C6288	43	47	34
C7552	211	146	138

Stulpelių paaiškinimai:

- ✓ Schema – schemos pavadinimas;
- ✓ R1 – standartinė schemos realizacija;
- ✓ R2 – persintezuota schemos realizacija, naudojant biblioteką – *class.db*;
- ✓ R3 – persintezuota schemos realizacija, naudojant biblioteką – *and_or.db*;

Lentelė rodo, kiek reikia testų vektorių, kad patikrinti visas pastovios būsenos klaidas (angl. *stuck-at fault*) visoms schemos realizacijoms.

4 lentelė rodo kiek vienos schemos realizacijos testai patikrina klaidų kitose realizacijose.

4. lentelė Neaptiktos klaidos skirtingoms realizacijoms.

Schemos realizacijos		**Testo tinkamumas**		
		R1	**R2**	**R3**
C432	R1	0	21	16
	R2	11	0	9
	R3	1	7	0
C499	R1	0	6	16
	R2	44	0	8
	R3	116	22	0
C880	R1	0	29	18
	R2	0	0	2
	R3	0	7	0
C1355	R1	0	8	12
	R2	25	0	12
	R3	20	10	0
C1908	R1	0	158	129
	R2	3	0	12
	R3	1	41	0
C2670	R1	0	24	21
	R2	36	0	4
	R3	29	8	0
C3540	R1	0	57	53
	R2	6	0	6
	R3	6	8	0
C5315	R1	0	72	77
	R2	9	0	17
	R3	11	10	0
C6288	R1	0	0	0
	R2	39	0	21
	R3	18	27	0
C7552	R1	0	190	241
	R2	24	0	44
	R3	17	16	0

Lentelės paaiškinimas:

Eilučių reikšmės:

- ✓ R1 – standartinė schemos realizacija;
- ✓ R2 – persintezuota schemos realizacija, naudojant biblioteką – *class.db*;
- ✓ R3 – persintezuota schemos realizacija, naudojant biblioteką – *and_or.db*;

Stulpelių reikšmės:

- ✓ R1 – kiek klaidų neaptinka standartinės realizacijos testai kitose realizacijose;
- ✓ R2 – kiek šios persintezuotos realizacijos (naudojant biblioteką – *class.db*) testas neaptinka klaidų kitose realizacijose;
- ✓ R3 – kiek šios persintezuotos realizacijos (naudojant biblioteką – *and_or.db*) testas neaptinka klaidų kitose realizacijose;

Šios lentelės stulpeliai rodo realizacijos testą, o eilutės realizaciją. Iš lentelės galima nustatyti, kaip vienos realizacijos testas tinka kitai realizacijai.

6.2 Tyrimas

Tyrimui buvo naudota sukurta programinė įranga. Jos dėka buvo greitai ir tiksliai ištirtos, ir palygintos visos tyrimui pasirinktos schemų realizacijos. Tyrimą sudarė atskirų schemų realizacijų kelių analizė ir rezultatų palyginimas. Visų schemų realizacijų tyrimo rezultatai pateikiami priede **C**. Ištyrus visas pasirinktų schemų realizacijas, buvo atlikti apibendrinimai.

Į eksperimentą nebuvo įtrauktos schemos C6288 realizacijos dėl milžiniško kelių skaičiaus. Šios schemos kelių skaičius artimas 10^{20}, todėl elementarus tokios schemos kelių perrinkimas labai ilgas procesas, kuriam nebuvo techninių resursų. Apytikslis kelių skaičius buvo paskaičiuotas remiantis [6] sudarytu metodu.

Tyrimo metu apžvelgta keletas galimų sąryšių tarp kelių skaičiaus ir testų skaičiaus skirtingose realizacijose. Programinės įrangos pateikti rezultatai turi daug aspektų, kurie neaptariami šiame darbe. Visa eksperimento informacija ir programinė įranga yra pateikiama kompaktiniame diske.

6.2.1 Schemų realizacijų analizė

Ištirtų schemų realizacijų kelių skaičius, pateikiamas lentelėje 5.

5. lentelė Ištirtų schemų kelių pasiskirstymo lentelė

Kelių pasiskirstymas realizacijose				
	Keliai	R1	R2	R3
C432	LKS	39560	20386	31160
	NKS	39571	20041	31185
	BKS	79131	40427	62345
C499	LKS	1376	10400	107552
	NKS	6272	10522	107552
	BKS	7648	20922	215104
C880	LKS	4727	2096	3887
	NKS	3915	2362	3180
	BKS	8642	4458	7067
C1355	LKS	1687392	37872	132640
	NKS	1690944	37456	132640
	BKS	3378336	75328	265280
C1908	LKS	364432	15483	425102
	NKS	364593	15537	425260
	BKS	729025	31020	850362
C2670	LKS	8479	3332	9321
	NKS	8637	3224	9497
	BKS	17116	6556	18818
C3540	LKS	3777885	775972	2103676
	NKS	3777840	778138	2103319
	BKS	7555725	1554110	4206995
C5315	LKS	626837	20106	200082
	NKS	626757	20229	199926
	BKS	1253594	40335	400008
C7552	LKS	327745	35414	125265
	NKS	327344	34979	106040
	BKS	655089	70393	231305

5 lentelės paaiškinimas:

LKS – Lyginis Kelių Skaičius

NKS – Nelyginis Kelių Skaičius

BKS – Bendras Kelių Skaičius.

Atliktas procentinis originalios realizacijos sulyginimas su persintezuotomis realizacijomis. Jis rodo, kaip originalios realizacijos testų vektorių skaičius, kelių skaičius, gedimų skaičius ir tinkamumo skaičius procentaliai siejasi su persintezuotomis schemos realizacijomis. Tai pateikiama 6 lentelėje.

6. lentelė Originalių schemos realizacijų procentinis santykis su persintezuotomis schemos realizacijomis.

Originalios schemos realizacijos santykis su persintezuotomis schemos realizacijomis					
Realizacija		**Kelių santykis**	**Testų skaičiaus santykis**	**Gedimų santykis**	**Tinkamumo santykis**
C432	R2	195,74%	123,91%	119,01%	185,00%
	R3	126,92%	126,67%	110,22%	462,50%
C499	R2	***36,55%***	***72,97%***	***76,69%***	***42,31%***
	R3	***3,56%***	***67,50%***	***60,19%***	***15,94%***
C880	R2	193,85%	126,53%	109,92%	2350,00%
	R3	122,29%	124,00%	101,51%	671,43%
C1355	R2	4484,83%	103,61%	119,00%	***54,05%***
	R3	1273,50%	107,50%	111,38%	***66,67%***
C1908	R2	2350,18%	207,02%	212,56%	1913,33%
	R3	***85,73%***	157,33%	152,12%	683,33%
C2670	R2	261,07%	***87,50%***	132,67%	112,50%
	R3	***90,96%***	***90,52%***	120,02%	121,62%
C3540	R2	486,18%	116,78%	126,35%	916,67%
	R3	179,60%	113,61%	124,05%	785,71%
C5315	R2	3107,96%	131,31%	135,29%	573,08%
	R3	313,39%	146,07%	127,07%	709,52%
C7552	R2	930,62%	144,52%	154,03%	633,82%
	R3	283,21%	152,90%	146,71%	1306,06%

6 lentelės paaiškinimas:

- ✓ Procentinis kelių santykis – tai santykis tarp standartinės realizacijos BKS ir persintezuotų realizacijų kelių skaičiaus.
- ✓ Procentinis testų skaičiaus – tai santykis tarp standartinės realizacijos testų skaičių ir persintezuotų realizacijų testų skaičiaus.
- ✓ Procentinis gedimų skaičiaus – tai santykis tarp standartinės realizacijos gedimų skaičių ir persintezuotų realizacijų gedimų skaičiaus.
- ✓ Procentinis tinkamumas – tai santykis tarp standartinės realizacijos gedimų skaičiaus sumos (visų realizacijų testams) ir persintezuotų realizacijų gedimų skaičiaus sumos (visų realizacijų testams).

Apibendrinant 6 lentelę reikia pastebėti, kad tik schemos C499 abiejose persintezuotose realizacijose visi santykiai mažesni už 100%. Tai reiškia, kad tik C499 schemoje persintezuotų realizacijų

kelių skaičius, testų vektorių skaičius, gedimų skaičius, ir tinkamumo skaičius yra didesni už originalios schemos realizacijos tuos pačius parametrus.

Schemoms C432, C880, C3540, C5315 ir C7552 visi rodikliai yra didesni nei 100%. Taigi šių persintezuotų schemų realizacijų parametrai mažesni už originalios schemos realizacijos parametrus.

Schemos C1355 kelių skaičiaus santykis, testų skaičiaus santykis ir gedimų skaičiaus santykis persintezuotose schemos realizacijose yra didesnis už 100%. Bet testų tinkamumo skaičiaus santykis yra mažesnis nei 100%. Galime teigti, kad priklausomybė tarp kelių skaičiaus ir testų tinkamumo, šios persintezuotos schemos realizacijose yra atvirkštinė.

Schemų C1908 ir C2670 R3 sintezės realizacijose, kelių skaičius yra didesnis už originalios schemos realizacijos kelių skaičių, o gedimų skaičius ir testų tinkamumo skaičius yra mažesni už originalios schemos realizacijos tuos pačius parametrus.

Testų ir kelių santykių procentinis sulyginimas pateikiamas 7 lentelėje ir 1 grafike. Šis sulyginimas leidžia spręsti ar skirtingoms realizacijoms kelių ir testų santykiai (matomi 6 lentelėje) vienodai pasiskirstę. Jeigu keliai ir testai vienodai pasiskirstę, tai reiškia, kad tiesioginė priklausomybė tarp kelių ir testų egzistuoja.

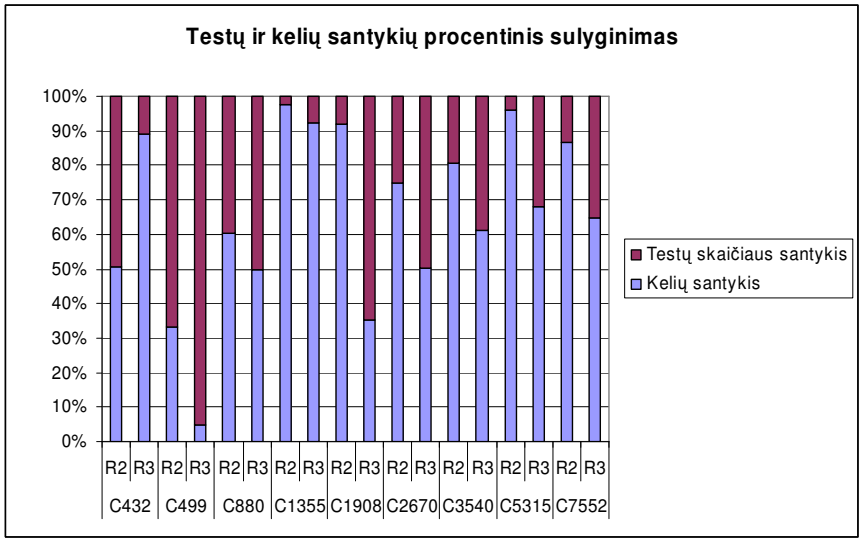
7. lentelė Testų ir kelių santykių procentinis sulyginimas.

Testų ir kelių santykių procentinis sulyginimas				
		Keliu santykis %	Testu santykis %	% skirtumas
C432	R2	61,23%	38,77%	22,47%
	R3	50,05%	49,95%	0,10%
C499	R2	33,37%	66,63%	-33,25%
	R3	5,00%	95,00%	-89,99%
C880	R2	60,51%	39,49%	21,01%
	R3	49,65%	50,35%	-0,70%
C1355	R2	97,74%	2,26%	95,48%
	R3	92,22%	7,78%	84,43%
C1908	R2	91,90%	8,10%	83,81%
	R3	35,27%	64,73%	-29,46%
C2670	R2	74,90%	25,10%	49,80%
	R3	50,12%	49,88%	0,24%
C3540	R2	80,63%	19,37%	61,26%
	R3	61,25%	38,75%	22,51%
C5315	R2	95,95%	4,05%	91,89%
	R3	68,21%	31,79%	36,42%
C7552	R2	86,56%	13,44%	73,12%
	R3	64,94%	35,06%	29,88%

Lentelės 7 paaiškinimas:

Lentelė sudaryta remiantis 6 lentelės rezultatais. Joje parodyta, kaip procentaliai susiję kelių ir testų santykiai.

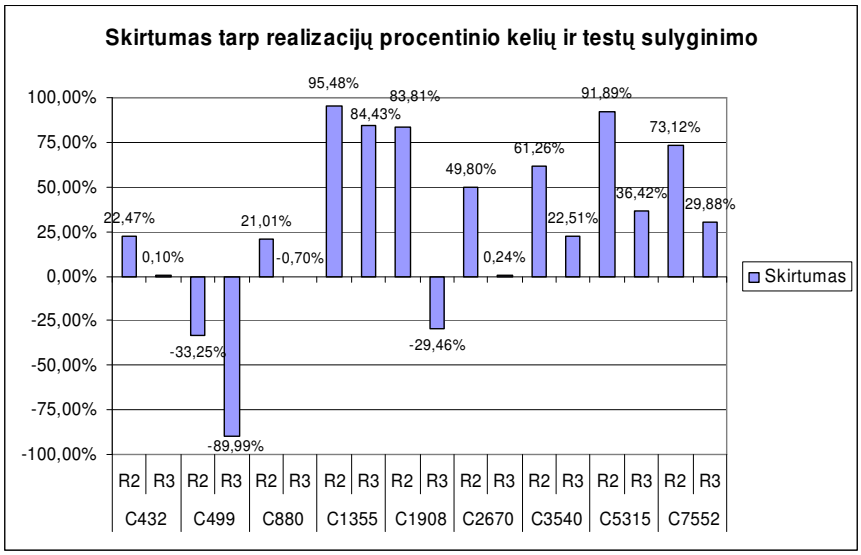
- ✓ Kelių santykis % buvo gautas kelių santykį (žr. 6 lentelę) padalinus iš kelių ir testo santykių sumos (žr. 6 lentelę).
- ✓ Testu santykis % buvo gautas testų santykį (žr. 6 lentelę) padalinus iš kelių ir testo santykių sumos (žr. 6 lentelę).
- ✓ % skirtumas (procentinis skirtumas) tai skirtumas tarp kelių santykio % ir testų santykio %. Šis skirtumas reikalingas vykdant atskirų schemos realizacijų apibendrinimą, kelių ir testų atžvilgiu (žiūrėti 1 grafiką).



1. grafikas Testų skaičiaus santykio procentinis ryšys su kelių santykiu.

1 grafikas rodo kaip procentiškai siejasi testų skaičiaus santykis iš 6 lentelės su kelių santykiu. Matome, kad kelių santykių ir testų santykių pasiskirstymas, tarp tos pačios schemos skirtingų realizacijų, yra labai nevienodas. Galima teigti, kad testų skaičius nepriklauso nuo kelių skaičiaus skirtingose realizacijose. 77,78% visų testų ir kelių sulyginimo rezultatų rodo, kad kelių santykis yra didesnis už testų skaičiaus santykį tos pačios schemos realizacijoms. Taigi kelių santykis tarp persintezuotų schemų realizacijų yra žymiai didesnis, lyginant su tų pačių schemų testų vektorių skaičiaus santykiu.

Testų ir kelių santykių procentinio sulyginimo apibendrinimas yra procentinis skirtumas (% skirtumo iš 7 lentelės). Šio skirtumo grafinis atvaizdavimas pateikiamas 2 grafike.



2. grafikas Testų ir kelių santykių procentinio sulyginimo apibendrinimas.

2 grafike matomi labai įvairūs skirtumai tarp procentinio schemų realizacijų kelių ir testų santykių. Maksimalų skirtumą tarp realizacijų R2 ir R3 kelių ir testų santykių procentinio pasiskirstymo turi schema C1908 – 113,27%. Minimalų skirtumą tarp tų pačių realizacijų kelių ir testų santykių procentinio pasiskirstymo turi schema C1355 – 11,05%. Skirtumai pakankamai dideli, todėl galime teigti, kad priklausomybės tarp schemos kelių skaičiaus santykio ir testų skaičiaus santykio, o tuo pačiu ir tarp schemos kelių skaičiaus ir schemos testų skaičiaus, nėra.

Testų tinkamumo ir kelių santykių procentinis sulyginimas (8 lentelė ir 3 grafikas) rodo, kaip kelių santykis procentaliai susijęs su realizacijų testų tinkamumu. 3 grafikas pateikia galimybę surasti priklausomybes tarp kelių santykių ir testų tinkamumo tirtoms realizacijoms. Jeigu santykinis sulyginimas pasirinktai schemai abiejose realizacijose vienodas, reiškia priklausomybė yra.

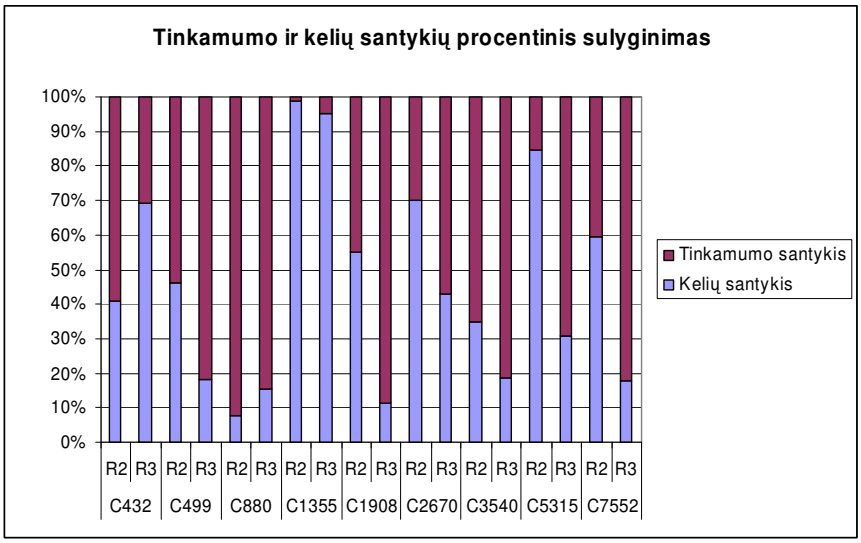
8. lentelė Testų tinkamumo ir kelių santykio procentinis sulyginimas.

		Keliu santykis %	Tinkamumo santykis %	% skirtumas
C432	R2	51,41%	48,59%	2,82%
	R3	21,53%	78,47%	-56,93%
C499	R2	46,35%	53,65%	-7,29%
	R3	18,24%	81,76%	-63,53%
C880	R2	7,62%	92,38%	-84,76%
	R3	15,41%	84,59%	-69,19%
C1355	R2	98,81%	1,19%	97,62%
	R3	95,03%	4,97%	90,05%
C1908	R2	55,12%	44,88%	10,25%
	R3	11,15%	88,85%	-77,71%
C2670	R2	69,89%	30,11%	39,77%
	R3	42,79%	57,21%	-14,43%
C3540	R2	34,66%	65,34%	-30,69%
	R3	18,61%	81,39%	-62,79%
C5315	R2	84,43%	15,57%	68,86%
	R3	30,64%	69,36%	-38,73%
C7552	R2	59,49%	40,51%	18,97%
	R3	17,82%	82,18%	-64,36%

Lentelės 8 paaiškinimas:

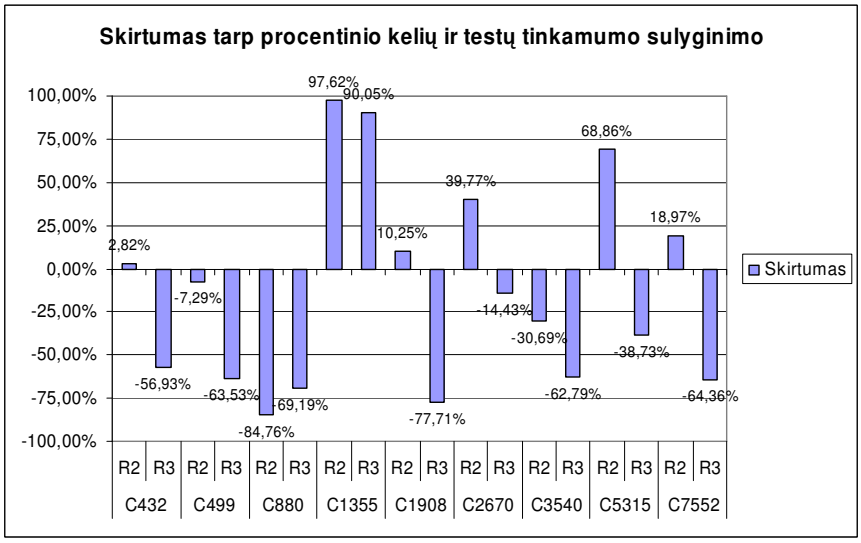
Lentelė sudaryta remiantis 6 lentelės rezultatais. Joje atvaizduota, kaip procentaliai susiję kelių ir testų tinkamumo santykiai.

- ✓ Kelių santykis % buvo gautas kelių santykį (žr. 6 lentelę) padalinus iš kelių ir tinkamumo santykių sumos (žr. 6 lentelę).
- ✓ Tinkamumo santykis % buvo gautas tinkamumo santykį (žr. 6 lentelę) padalinus iš kelių ir tinkamumo santykių sumos (žr. 6 lentelę).
- ✓ % skirtumas (procentinis skirtumas) – tai skirtumas tarp kelių santykio % ir tinkamumo santykio %. Šis skirtumas reikalingas vykdant atskirų schemos realizacijų apibendrinimą kelių ir testų tinkamumo atžvilgiu (žr. 4 grafiką).



3. grafikas Testų tinkamumo ir kelių santykių procentinis sulyginimas

3 grafike matoma, kaip procentaliai siejasi kelių santykis su testų tinkamumo santykiu. Kelių santykio ir tinkamumo santykio pasiskirstymas labai nevienodas. Skirtumas tarp kelių procentinio santykio ir testų tinkamumo procentinio santykio, kiekvienai schemos realizacijai, pateikiamas 4 grafike.



4. grafikas Schemos realizacijų kelių santykio skirtumas su schemos realizacijų testų tinkamumu.

4 grafiko skirtumai tarp procentinio schemų realizacijų kelių ir testų tinkamumo santykių yra įvairūs. Maksimalų skirtumą tarp realizacijų R2 ir R3 kelių ir testų tinkamumo santykių procentinio pasiskirstymo turi schema C5315 – 107,59%. Minimalų skirtumą tarp tų pačių realizacijų kelių ir testų tinkamumo santykių procentinio pasiskirstymo turi schema C1355 – 7,57%. Skirtumai pakankamai dideli, todėl galime teigti kad priklausomybės tarp schemos kelių skaičiaus santykio ir testų tinkamumo skaičiaus

santykio, o tuo pačiu ir tarp schemos kelių skaičiaus ir schemos testų tinkamumo skaičiaus, nėra.

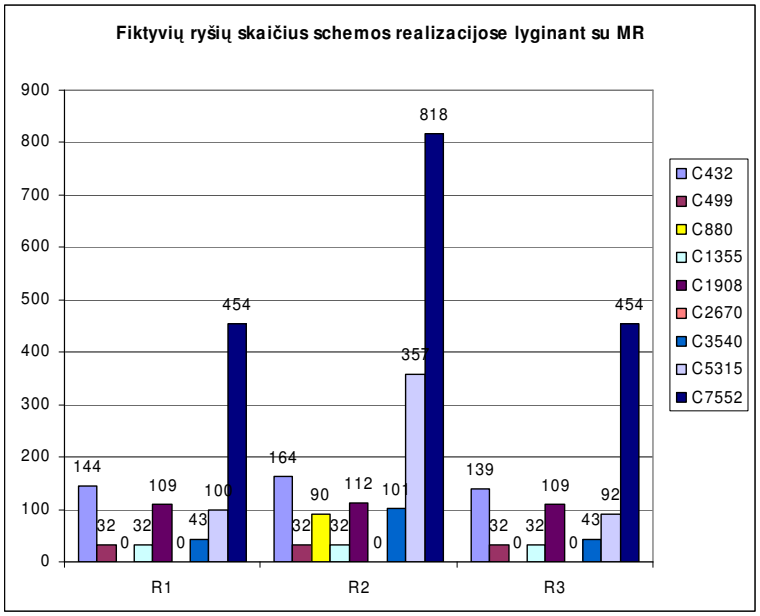
6.2.2 Schemų realizacijų palyginimas

Visos, kiekvienos tirtos schemos, realizacijos buvo sulygintos su rinkinių matrica (toliau MR matrica), kuri apibrėžia ar egzistuoja testų rinkinys tarp konkretaus schemos įėjimo ir išėjimo. Buvo užtikrinta, kad šios matricos elementai absoliučiai teisingi. Kad įsitikinti sukurtos programinės įrangos teisingumu, reikėjo realizuoti gautų schemų analizės rezultatų sulyginimą su MR matrica. Fiktyvių kelių skaičiai pateikiami 9 lentelėje.

9. lentelė Fiktyvių kelių skaičius lyginant schemos realizacijas su ryšių matrica.

Schema	R1	R2	R3
C432	144	164	139
C499	32	32	32
C880	0	90	0
C1355	32	32	32
C1908	109	112	109
C2670	0	0	0
C3540	43	101	43
C5315	100	357	92
C7552	454	818	454

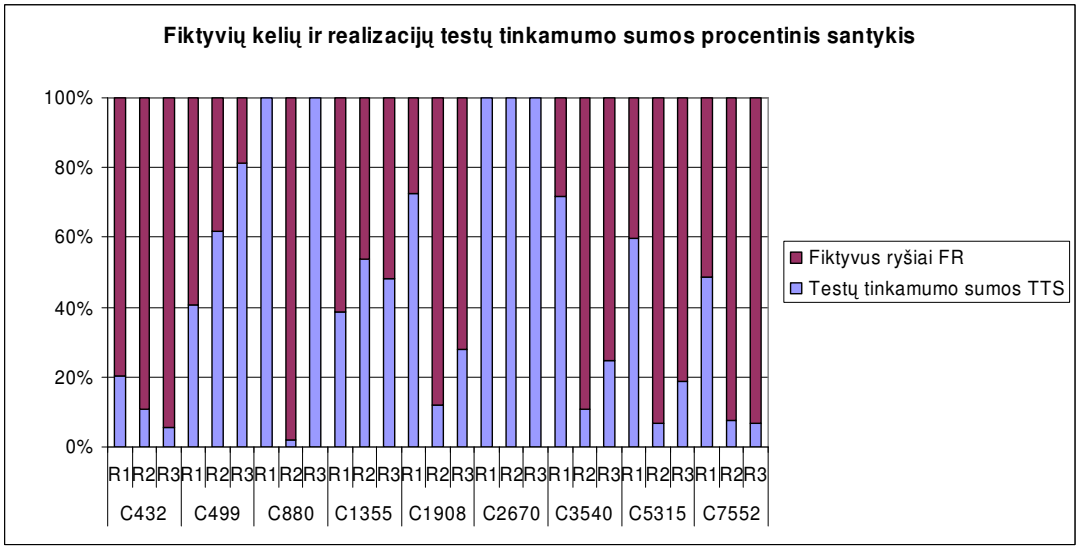
9 lentelė vaizduoja kiek fiktyvių kelių buvo surasta kiekvienos tirtos schemos realizacijose. Fiktyvių kelių skaičius labai priklauso nuo schemos realizacijoje naudotos elementų bazės. Mažiausia fiktyvių ryšių surasta R3 schemų realizacijose. Ši realizacija išsiskiria tuo, kad jos visi elementai yra elementarūs (žiūrėkite priedą A).



5. grafikas Fiktyvių kelių skaičius lyginant schemos realizacijas su ryšių matrica.

9 lentelė ir 5 grafikas rodo fiktyvius ryšius tirtose schemų realizacijose. Nagrinėjant realizacijas, matyti, kad tik 18,51% visų tirtų schemų realizacijų neturi fiktyvių ryšių.

Fiktyvių kelių ir skirtingų schemos realizacijų testų tinkamumo sumos procentinis santykis 6 grafike.



6. grafikas Fiktyvių kelių ir skirtingų realizacijų testų tinkamumo sumos procentinis santykis.

6 grafike pavaizduotas procentinis ryšys tarp schemos realizacijų fiktyvių ryšių ir testų tinkamumo

realizacijai sumų. Iš paveikslo matome, kad priklausomybės tarp šių parametrų nėra.

7. Išvados

Naudojant sukurtą programinę įrangą, buvo atlikti tyrimai su 9-iomis ISCAS'85 schemomis. Kiekviena iš tirtų schemų turėjo po tris skirtingas realizacijas:

- ✓ R1 – standartinė realizacija.
- ✓ R2 – persintezuota schemos realizacija, naudojanti biblioteką – *class.db*;
- ✓ R3 – persintezuota schemos realizacija, naudojanti biblioteką – *and_or.db*;

Kiekvienos schemos realizacijai buvo suskaičiuotas kelių skaičius. Didžiausią skirtumą tarp tirtų schemų realizacijų kelių skaičiaus turėjo schemos C1355 realizacijos. Šios schemos realizacijų maksimalaus ir minimalaus kelių skaičiaus skirtumo santykis su maksimaliu realizacijų kelių skaičiumi siekė net 97,77%. Mažiausias skirtumas tarp schemos realizacijų kelių nustatytas schemai C880. Santykis tarp šios schemos realizacijų maksimalaus ir minimalaus kelių skirtumo su maksimaliu realizacijų kelių skaičiumi buvo 48,41%.

Atliekant tiriamų schemų realizacijų sulyginimą su rinkinių matricomis, buvo ieškomi fiktyvūs keliai tiriamose realizacijose. Net 81,49% visų ištirtų realizacijų turėjo fiktyvių kelių. Išsiskyrė tik schemos C2670 visos realizacijos, nes jose fiktyvių kelių nebuvo. Didžiausias fiktyvių kelių skaičius aptiktas schemos C7552 R2 realizacijoje – 818 kelių. Fiktyvių kelių nustatymas buvo vykdomas sukurtos programinės įrangos pagalba.

Suskaičiavus schemų realizacijų kelius, buvo atliktas santykinis persintezuotų realizacijų parametrų lyginimas su standartine schemos realizacija. Santykinai buvo sulyginta persintezuotų schemų realizacijų kelių skaičius, testų skaičius, gedimų skaičius ir testų tinkamumo skaičius su standartinės schemos realizacijos tokiais pačiais parametrais. Išsiskyrė schema C499, kurios parsintezuotų realizacijų visi minėti parametrai buvo didesni už standartinės schemos parametrus. Likusių schemų persintezuotų realizacijų parametrai po sintezės buvo mažesni už standartinės schemos realizacijos parametrus.

Atlikus eksperimentus su tiriamom schemom ir apibendrinus rezultatus, buvo nustatyta, kad priklausomybių tarp kelių skaičiaus ir testų skaičiaus schemos realizacijose nėra. Ryšys tarp schemos realizacijos kelių skaičiaus ir realizacijos testų tinkamumo kitoms realizacijoms taip pat nebuvo nustatytas. Remiantis realizacijų sulyginimo eksperimento rezultatais, nebuvo aptikta tiesioginė priklausomybė tarp schemos realizacijos testų tinkamumo kitai realizacijai ir realizacijos fiktyvių ryšių skaičiaus.

8. Literatūra

[1] Ali Bahrami Object Oriented Systems Development: Using the Unified Modeling Language - McGraw-Hill College, 1999 – 521 p.

[2] Chris D. Godsil , Gordon F. Royle Algebraic Graph Theory - Springer Verlag, 2001 – 464 p.

[3] Craig Larman Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process (2nd Edition) - Prentice Hall PTR, 2001 – 656 p.

[4] Eduardas Bareiša, Vacius Jusas, Kęstutis Motiejūnas, Rimantas Šeinauskas Test Reuse for Re–synthesized Cores.

[5] Horng-Bin Wang, Shi-Yu Huang, Jing-Reng Huang Gate-Delay Fault Diagnosis Using the Inject-and-Evaluate Paradigm, 7th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'02), 2002, p. 117-128

[6] I. Pomeranz and S.M. Reddy. An efficient non-enumerative method to estimate the path delay fault coverage in combinational circuits//IEEE Transactions on CAD, 1994, p. 240-250

[7] J. A. Starzyk, D. Liu Locating Stuck Faults In Analog Circuits, The 2002 IEEE International Symposium on Circuits and Systems, 2002

[8] J.-J. Liou, L.-C. Wang, K.-T. Cheng, J. Dworak, M.R. Mercer, R. Kapur, T.W. Williams. Enhancing Test Efficency for Delay Fault Testing Using Multiple-Clocked Schemes. Proceedings of 39th Design Automation Conference, 2002, p. 371-374

[9] John R. Gregg Ones and Zeros : Understanding Boolean Algebra, Digital Circuits, and the Logic of Sets - Wiley-IEEE Computer Society Pr; 1998 – 296 p.

[10]Jonathan L. Gross, Thomas W. Tucker Topological Graph Theory - Dover Pubns, 2001 – 384 p.

[11]Pomeranz I. and Reddy S. M., Resynthesis of combinational logic circuits for improved path delay fault testability using comparison units//IEEE Transactions on Very Large Scale Integration (VLSI) Systems - 2001, Nr. 9, p. 679 – 689

[12]Robin J. Wilson Introduction to Graph Theory. - 2nd ed. - Prentice Hall, 2000 - 470p

[13]Sudhakar M. Reddy, Irith Pomeranz On the Number of Tests to Detect All Path Delay Faults in Combinational Logic Circuits// IEEE Transactions on Computers, v.45 n.1, 1996, p. 50-62

9. Terminų ir santrumpų sąrašas

- Loginė schema arba schemos realizacija – (toliau *schema* arba *realizacija*) tai loginės funkcijos realizacija.
- Schemos kelias – tai ryšys tarp schemos įėjimo ir išėjimo.
- Būlio algebra (angl. *Boolean algebra*) – žiūrėti [9]
- Loginis schemos kelias – tai ryšys tarp schemos įėjimo ir išėjimo, kai įėjimo pokytis pasireiškia išėjime.
- Lyginis kelias (LK) – tai toks schemos kelias, kuriame signalas buvo invertuojamas lyginį kartų skaičių.
- Nelyginis kelias (NK) – tai toks schemos kelias, kuriame signalas buvo invertuojamas nelyginį kartų skaičių.
- Padengiantis kelias (PK) – tai toks schemos kelias kuriame yra bent vienas elementas per kurį dar nebuvo praėjęs kitas kelias.
- Fiktyvus kelias – tai toks kelias tarp schemos įėjimo ir išėjimo, kai įėjimo loginis pokytis nekeičia išėjimo reikšmės.
- Vėlinimo klaidos, tai klaidos kurios iškyla dėl to, jog pokytis įėjimuose įvyksta greičiau nespėjęs pasirodyti išėjime.
- Konstantinio gedimo klaidos, tai klaidos kai išėjime gaunama pastovi (konstantinė) reikšmė.
- Testų vektorius, tai schemos įėjimų rinkinys.
- Kelių perrinkimas – tai kiekvieno įėjimo ryšių nustatymas su schemos išėjimais atliekant detalią schemos struktūros analizę.
- Grafas – terminas naudojamas grafų teorijoje. Plačiau [2,12].
- Grafo padengimas – grafo kelių suradimas. Plačiau [2,12].
- MR failai, kuriuose saugoma matrica atspindinti testų rinkinius duotai schemai.
- Sistemos nustatymai – parametrai reikalingi sistemai startuojant arba sistemos darbui.
- OS – operacinė sistema.
- LSSA (Loginių schemų struktūros analizė) – programinė įranga skirta loginių schemų (aprašytų VHDL programavimo kalba) kelių skaičiavimui. Taip pat dviejų realizacijų sulyginimui.
- VHDL – angl. VHSIC (*very high speed integrated circuits*) *Hardware Description Language*.
- JAVA specifinis paketas(biblioteka) – JAR failas –kuriame yra saugomos klasės specifiniam uždaviniui atlikti.

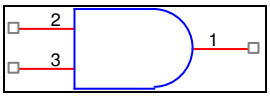
- UML - angl. *Unified Modeling Language*. Plačiau [1,3].
- XML – angl.*Extensible Markup Language*. Plačiau apie XML standarto aprašymą - http://www.w3c.org/.
- IntelliJ IDE – terpė JAVA projektams kurti. Plačiau IntelliJ JAVA projektų kūrimo terpė – http://www.jetbrains.com/idea/
- ISCAS'85 *bencmark circuits* – 1985 metais, konferencijos metu aptartos schemos..
- MR – failo plėtinys. Failuose su tokiu plėtiniu saugoma rinkinių matrica duotai schemai. Šis failas reikalingas schemų sulyginimui.
- XPP – angl. XML *Pull Parser*. Tai XML analizatorius (angl. *parser*). Jo privalumas yra tas kad jis pakankamai greitas ir nenaudojantis daug resursų kaip gerai žinomas DOM ar SAX analizatoriai.
- PA – panaudojimo atvejai
- DOM – angl. *Document Object Model* – dokumentų objekto modelis, firmos W3C produktas. Plačiau http://www.w3.org/.
- Juoda dėžė – terminas naudojamas, kai žinome schemos įėjimus ir išėjimus, tačiau nežinome konkrečių ryšių tarp jų. Pavyzdžiui 1 paveiksle pavaizduota loginė schema kaip juoda dėžė, o 1 schemoje matome pilną loginės schemos struktūros vaizdą.

10. Priedai

10.1 Priedas A

Elementarūs loginiai elementai yra trys, kiekvieno jų grafinis atvaizdavimas ir teisingumo lentelė pateikta šiame skyriuje.

1. IR elementas (angl. *AND gate*)

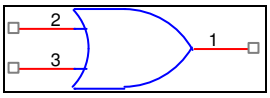


Paveikslas Nr. 1 IR loginio elemento grafinis atvaizdavimas.

10. lentelė Elementaraus loginio elemento IR teisingumo lentelė.

Įėjimai		**Išėjimas**
2	**3**	**1**
0	0	0
0	1	0
1	0	0
1	1	1

2. ARBA elementas (angl. *OR gate*)

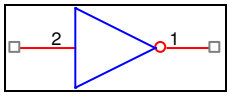


Paveikslas Nr. 2 ARBA loginio elemento grafinis atvaizdavimas.

11. lentelė Elementaraus loginio elemento ARBA teisingumo lentelė.

Įėjimai		**Išėjimas**
2	**3**	**1**
0	0	0
0	1	1
1	0	1
1	1	1

3. INVERSIJOS elementas (angl. *NOT gate*)



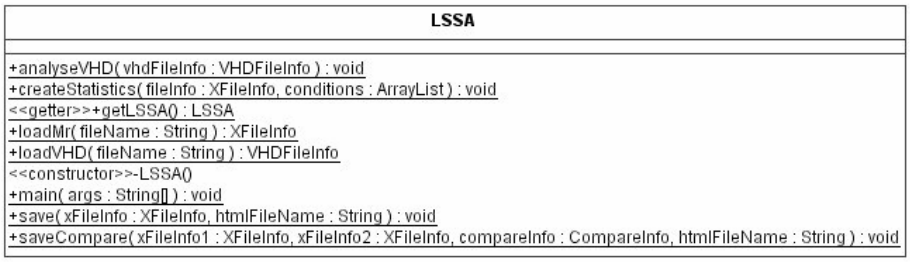
Paveikslas Nr. 3 Inversijos loginio elemento grafinis atvaizdavimas.

12. lentelė Elementaraus loginio elemento INVERSIJA teisingumo lentelė.

Įėjimas	**Išėjimas**
2	**1**
0	1
1	0

10.2 Priedas B

Priede detaliai paaiškintos sistemoje naudotos klasės ir interfeisai. Komentuoti kiekvienos klasės kintamieji ir esminiai metodai.



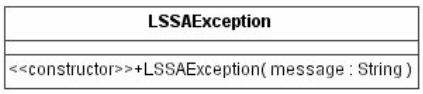
LSSA klasė kuri valdo sistemos darbą.

Kintamieji:

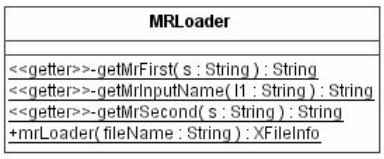
- ✓ *ourInstance* – private static LSSA– Šitos klasės objektas. Reikalingas realizuoti Singleton šabloną.

Metodai:

- ✓ *getLSSA* – public static LSSA getLSSA() – Singleton šabloną realizuojantis metodas.
- ✓ *main* – public static void main(java.lang.String[] args) – Paleidžiamasis metodas.
  - Parametrai:
    args – Šiuo parametru perduodamas į programą failas/failai, kurį reikia analizuoti.
- ✓ *loadVHD* – public static VHDFileInfo loadVHD(java.lang.String fileName) throws LSSAException – Funkcija nuskaito VHD failą.
  - Parametrai:
    fileName – VHD failo vardas.
  - Grąžina: grąžinama rezultatų struktūra.
- ✓ *analyseVHD* – public static void analyseVHD(VHDFileInfo vhdFileInfo) throws LSSAException – Vykdoma užkrauto VHD failo analizė.
  - Parametrai:
    vhdFileInfo – vhd failo struktūra.
- ✓ *loadMr* – public static XFileInfo loadMr(java.lang.String fileName) throws LSSAException – Funkcija nuskaito MR failą į rezultatų struktūrą.
  - Parametrai:
    fileName – MR failo vardas.
  - Grąžina: grąžinama rezultatų struktūra.
- ✓ *createStatistics* – public static void createStatistics(XFileInfo fileInfo, java.util.ArrayList conditions) – Procedūra pasirinktai schemai vykdo statistikos paskaičiavimą.
  - Parametrai:
    fileInfo – struktūra kuriai reikia atlikti statistiką.
    conditions – statistikos sąlygos.
- ✓ *save* – public static void save(XFileInfo xFileInfo, java.lang.String htmlFileName) throws LSSAException – Procedūra issaugo informacijos struktūrą į XML ir HTML failus.
  - Parametrai:
    xFileInfo – informacijos struktūra.
    htmlFileName – html failo vardas, jeigu paduodama null failo vardas pasidaro toks koks yra tiriamos schemos vardas pridedant plėtinį ".html".
- ✓ *saveCompare* – public static void saveCompare(XFileInfo xFileInfo1, XFileInfo xFileInfo2, CompareInfo compareInfo, java.lang.String htmlFileName) throws LSSAException – Procedūra issaugo palyginimo struktūras ir palyginimo rezultatus į XML ir HTML failus.
  - Parametrai:
    xFileInfo1 – pirmoji struktūra kuri buvo lyginta.
    xFileInfo2 – antroji struktūra kuri buvo lyginta.
    compareInfo – palyginimo rezultatai.
    htmlFileName – html failo vardas, jeigu paduodama null failo vardas pasidaro toks koks yra tiriamų schemų vardai pridedant plėtinį ".html".



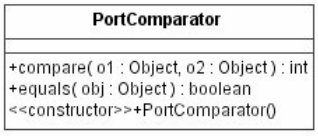
ELSSAExceptions – klaidų klasė implementuojanti standartinę JAVA klasę java.lang.Exception. Ji valdo visas programoje iškilusias klaidas.



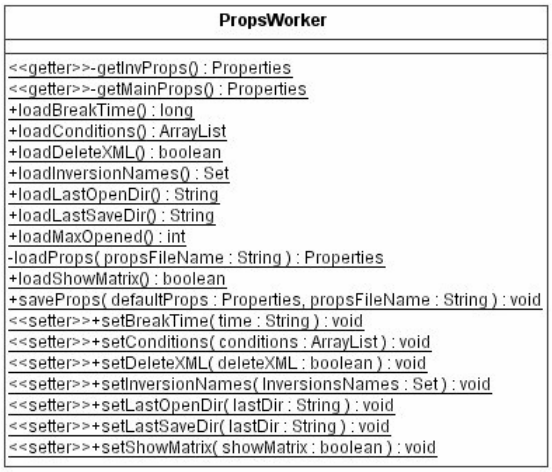
MRLoader – klasė kuri vykdo užkrovimus iš duomenų failų į XFileInfo objektus. Klasė implementuoja interfeisą ELSSAExceptions.

Metodai:

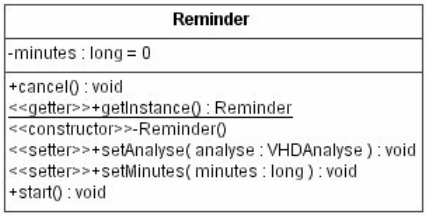
- ✓ *mrLoader* – public static final XFileInfo mrLoader(java.lang.String fileName) throws LSSAException – Funkcija nuskaitinėja "*.mr" failą ir užkrauna duomenis į XFileInfo struktūrą.
  - Parametrai:
    fileName – ".mr" failo, kurį reikia nuskaityti, pavadinimas.
  - Grąžina: grąžina XFileInfo struktūrą su nuskaitytais duomenimis.
- ✓ *getMrInputName* – private static final java.lang.String getMrInputName(java.lang.String l1) – Funkcija skirta nuskaitinėjant MR failus. Nuskaitinėjant pirmą matricos elės eilutę funkcija randa įėjimo vardą.
  - Parametrai:
    l1 – pirma matricos eilės eilutė.
  - Grąžina: grąžina įėjimo vardą.
- ✓ *getMrFirstprivate* – static final java.lang.String getMrFirst(java.lang.String s) – Procedūra analizuoja matricos eilės vienos eilutės elementus.
  - Parametrai:
    s – matricos eilės vienos eilutės info pvz. ": 0 26 :"
  - Grąžina: grąžina pirmą elemento sudedamąją dalį: 0
- ✓ *getMrSecond* – private static final java.lang.String getMrSecond(java.lang.String s) – Procedūra analizuoja matricos eilės vienos eilutės elementus.
  - Parametrai:
    s – matricos eilės vienos eilutės info pvz. ": 0 26 :"
  - Grąžina: grąžina antrą elemento sudedamąją dalį: 26



PortComparator klasė implementuoja standartinę JAVA klasę java.util.Comparator. Ši klasė atlieka Port klasių sulyginimą. Naudojama tam kad surikiuotų įėjimo ir išėjimo portus pagal jų vardus.



PropsWorker klasė nuskaitinėja ir įrašinėja duomenis į nustatymų (angl. properties) failus. Šiuose failuose saugoma vartotojo keičiama informacija, tai gali būti: direktorija iš kurios paskutinį kartą buvo atidarytas failas ir pan.



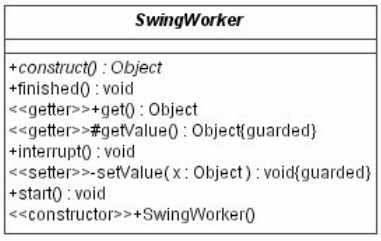
Reminder klasė vykdo VHD analizę ir sustabdo analizę jeigu analizė tęsiasi ilgiau negu buvo leista. Klasė implementuoja Singleton šabloną (šios klasės objektas gali būti sukurtas tik vieną kartą).

Kintamieji:

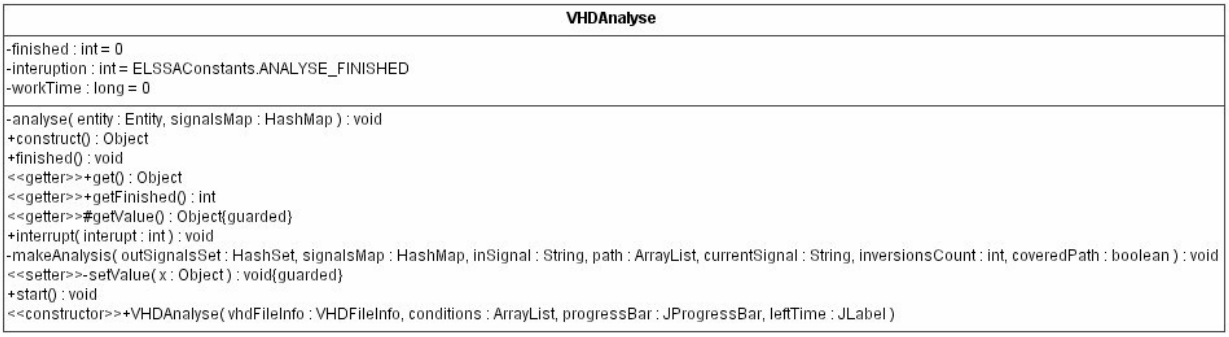
- ✓ timer – private java.util.Timer – Šis standartinės JAVOS kintamasis apibrėžia kada reikės sustabdyti analizę jeigu jai neužteko duoto laiko.
- ✓ analyse – private VHDAnalyse– analizę atliekantis objektas.
- ✓ minutes – private long– minučių skaičius kiek gali užtrukti analizė. Jeigu minučių skaičius lygus 0 analizė bus vykdoma tol kol nesibaigs arba nebus nutraukta vartotojo.
- ✓ ourInstance – private static Reminder – Singleton šablonui reikalinga klasė.

Metodai:

- ✓ *getInstance* – public static Reminder getInstance() – Singleton šabloną vykdantis metodas. Šis metodas visada grąžina vieną ir ta patį objektą.
    - o Grąžina: Grąžinamas šios klasės objektas.
- ✓ *star – t*public final void start()– Analizės paleidimo metodas.
- ✓ *cancel* – public final void cancel()– Analizės stabdymo metodas.



SwingWorker – klasė naudojama kai dirbama su grafiniu vartotojo interfeisu. Ji leidžia lengvai kurti naujas gijas lygiagrečiam darbui. Prireikus gijas galima nutraukti iškviečiant *interrupt()* metodą. Ši klasė siūloma http://java.sun.com/ (JAVA kalbos kūrėjo) kaip patogi galimybė naujų gijų kūrime. Plačiau – http://java.sun.com/.



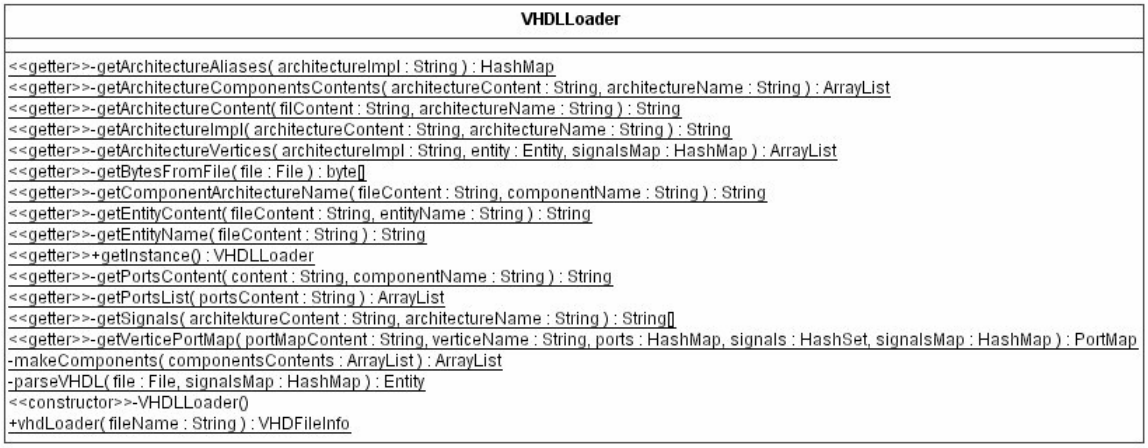
VHDAnalyse klasė atlieka nuskaityto VHD failo analizę.

Kintamieji:

- ✓ vhdFileInfo – private VHDFileInfo – Strukūra kuria reikia analizuoti.
- ✓ conditions – private java.util.ArrayList – Masyvas kuriame saugosime visas s1lygas.
- ✓ resultsMap – private java.util.HashMap – Struktūros rezultatų 'planas'.
- ✓ workTime – private long– Kiek laiko vyksta analizė.
- ✓ lssaTimer – private LSSATimer– Chronometras kuris skaičiuoja laiką.
- ✓ reminder – private Reminder – Priminėjas. Jis sustabdo analizę jeigu ji užtruko per ilgai.
- ✓ finished – private int – Kintamasis kuris parodo ar analizė jau pabaigta. Jeigu 0 – analizė tęsiasi, jeigu 1 – analizė nutraukta vartotojo, jeigu 2 – analizė nutraukta laiko limito, jeigu 3 – analizė pilnai baigta, .

Metodai:

- ✓ *analyse* – private void analyse(Entity entity, java.util.HashMap signalsMap) – Procedūra ieškanti ryšių tarp elemento įėjimo ir išėjimo signalų.
    - Parametrai:
      entity – elementas kuriam nustatinėjami ryšiai.
      signalsMap – ryšiai tarp signalų ir viršūnių įėjimo portMap'ų
- ✓ *makeAnalysis* – private void makeAnalysis(java.util.HashSet outSignalsSet, java.util.HashMap signalsMap, java.lang.String inSignal, java.util.ArrayList path, java.lang.String currentSignal, int inversionsCount, boolean coveredPath) – Procedūra vykdo analizę veikdama rekursiškai.
    - Parametrai:
      outSignalsSet – pagrindinio elemento išėjimo portų signalai.
      signalsMap – ryšiai tarp signalų ir viršūnių įėjimo portMap'ų
      inSignal – įėjimo signalas.
      path – viršųnių perėjimo kelių masyvas. Reikalingas tam kad nesusidarytų ciklai ieškant kelio iki išėjimo signalo.
      currentSignal – atėjęs signalas.
      inversionsCount – inversijų skaičius konkrečiame kelyje.
      coveredPath – ar kelias padengiantis.



VHDLLoader klasė implementuoja interfeisus ELSSAExceptions ir EVHDLConstants. Ši klasė vykdo visą VHD failo nuskaitymą, o nuskaitytus duomenis užkrauna į objektus: Entity, Architecture, Component, Port, Vertice, PortMap.

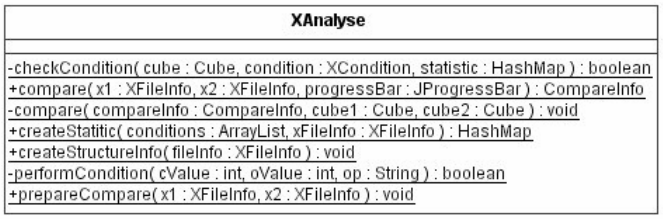
Kintamieji:

- ✓ *ourInstance* – private static VHDLLoader – Šitos klasės objektas. Reikalingas realizuoti Singleton šabloną.

Metodai:

- ✓ *getInstance* – public static VHDLLoader getInstance()– Singleton šabloną realizuojantis metodas.
  - Grąžina: Grąžina šio tipo klasę.
- ✓ *vhdLoader* – public static final VHDFileInfo vhdLoader(java.lang.String fileName) throws LSSAException – Nuskaito duomenis iš VHD failo ir užkrauna juos į XFileInfo struktūrą.
  - Parametrai:
    fileName – VHD failo vardas.
  - Grąžina: nuskaitytų duomenų struktūra.
- ✓ *parseVHDL* – private static Entity parseVHDL(java.io.File file, java.util.HashMap signalsMap) throws LSSAException – Analizuoja nurodytą failą
  - Parametrai:
    file – failas iš kurio bus nuskaitinėjami duomenys.
    signalsMap – saugo įėjimo tokia struktūrą: signalo_vardas–>(aibė port_mapų). Bus reikalingas ieškant kelių tarp viršūnių.
  - Grąžina: grąžina sukurtą ir užpildytą Entity objektą.
- ✓ *getBytesFromFile* – private static byte[] getBytesFromFile(java.io.File file) throws LSSAException – Užkrauna baitus iš failo į baitų masyvą.
  - Parametrai:
    file – failas iš kurio bus nuskaitinėjami duomenys.
  - Grąžina: failas baitų masyve.
- ✓ *getEntityName* – private static java.lang.String getEntityName(java.lang.String fileContent) throws LSSAException – Funkcija ieško elemento (ENTITY) vardo faile.
  - Parametrai:
    fileContent – failo tekstinis turinys.
  - Grąžina: Grąžina surastą elemento vardą, arba null jeigu elemento vardas nerastas.
- ✓ *getEntityContent* – private static java.lang.String getEntityContent(java.lang.String fileContent, java.lang.String entityName) throws LSSAException – Funkcija randa pagrindinio elemeto turinį.
  - Parametrai:
    fileContent – failo tekstinis turinys.
    entityName – pagrindinio elemento vardas.
  - Grąžina: elemento tekstinis turinys.
- ✓ *getPortsContent* – private static java.lang.String getPortsContent(java.lang.String content, java.lang.String componentName) throws LSSAException – Funkcija suranda elemento (ENTITY/COMPONENT) portus (PORT).
  - Parametrai:
    content – Tekstas kuriame reikia ieškoti portų.
  - Grąžina: Grąžina portus jeigu portai surasti, kitu atveju grąžina null.
- ✓ *getPortsList* – private static java.util.ArrayList getPortsList(java.lang.String portsContent) throws LSSAException – Funkcija išanalizuoja portus iš tekstinio.ir sukuria Port tipo elementų masyvą.
  - Parametrai:
    portsContent – partų tekstinis tekstinis turinys
  - Grąžina:Grąžinama Portų masyvas.
- ✓ *getComponentArchitectureName* – private static java.lang.String getComponentArchitectureName(java.lang.String fileContent, java.lang.String componentName) throws LSSAException – Funkcija randa faile komponento architektūros vardą, kuris vėliau bus naudojamas rasti visai komponento arhitektūrai.
  - Parametrai:
    fileContent – failo tekstinis turinys.
    componentName – komponento vardas.
  - Grąžina: grąžina surastos architektūros vardą nurodytam komponentui.
- ✓ *getArchitectureContent* private static java.lang.String getArchitectureContent(java.lang.String filContent, java.lang.String architectureName) throws LSSAException – Fukcija randa faile pilną architktūrą pagal architektūros vardą ir ją gražina.
  - Parametrai:
    filContent – failo tekstinis turinys.

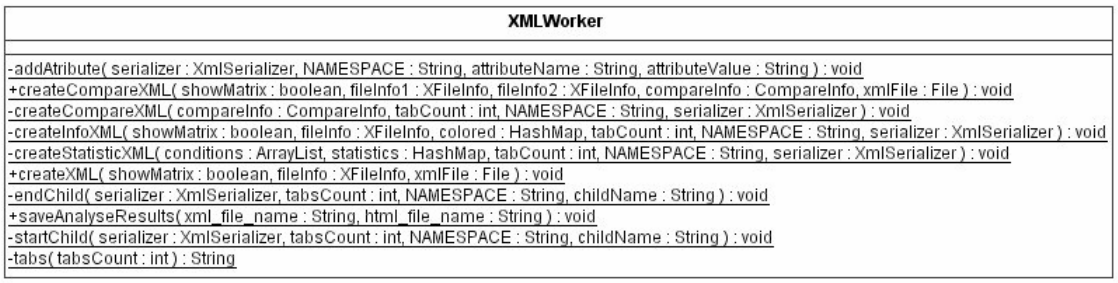
architectureName – architektūros vardas.
    - Grąžina: pilną rchitektūrą pagal architektūros vardą.
- ✓ *getSignals* – private static java.lang.String[] getSignals(java.lang.String architektureContent, java.lang.String architectureName) throws LSSAException – Funkcija randa architektūroje naudotus signalus.
    - Parametrai:
      architektureContent – architektūros tekstinis turinys
      architectureName – architektūros vardas.
    - Grąžina: grąžinama masyvas signalų kurie yra naudojami architektūroje.
- ✓ *getArchitectureComponentsContents* – private static java.util.ArrayList getArchitectureComponentsContents(java.lang.String architectureContent, java.lang.String architectureName) throws LSSAException – Funkcija randa architektūros naudojamų komponenčių turinį.
    - Parametrai:
      architectureContent – architektūros tekstinis turinys.
      architectureName – architektūros vardas.
    - Grąžina: masyva architektūroje naudojamų komponenčių turinių.
- ✓ *makeComponents* – private static java.util.ArrayList makeComponents(java.util.ArrayList componentsContents) throws LSSAException – Iš komponentų turinių formuojami komponentai.
    - Parametrai:
      componentsContents – masyvas komponentų turinių.
    - Grąžina: grąžina masyva komponentų objektų.
- ✓ *getArchitectureImpl* – private static java.lang.String getArchitectureImpl(java.lang.String architectureContent, java.lang.String architectureName) throws LSSAException – Funkcija randa architektūros viršūnių turinį.
    - Parametrai:
      architectureContent – architektūros tekstinis turinys.
      architectureName – architektūros vardas
    - Grąžina: grąžina surastą architektūros viršūnių turinį.
- ✓ *getArchitectureAliases* – private static java.util.HashMap getArchitectureAliases(java.lang.String architectureImpl) – Funkcija randa pagrindinio elemento (Entity) portų vardų slaptavardžius.
    - Parametrai:
      architectureImpl – architektūros viršūnių turinys.
    - Grąžina: grąžina portų ir slapyvardžių sąryšius.
- ✓ *getArchitectureVertices* private static java.util.ArrayList getArchitectureVertices(java.lang.String architectureImpl, Entity entity, java.util.HashMap signalsMap) throws LSSAException – Funkcija nuskaito architektūroje naudotas viršūnes.
    - Parametrai:
      architectureImpl – architektūros viršūnių turinys.
      entity – pagrindinis elementas kurį nagrinėsime.
      signalsMap – saugo įėjimo tokia struktūrą: signalo_vardas–>(aibė port_mapų). Bus reikalingas ieškant kelių tarp viršūnių.
    - Grąžina: masyva architektūros viršūnių.
- ✓ *getVerticePortMap* – private static PortMap getVerticePortMap(java.lang.String portMapContent, java.lang.String verticeName, java.util.HashMap ports, java.util.HashSet signals, java.util.HashMap signalsMap) throws LSSAException – Funkcija kuri sukuria PortMap objektą.
    - Parametrai:
      portMapContent – iš ko bus kuriamas objektas pvz. "A => n221"
      verticeName – viršūnės, kuri yra analizuojama, vardas
      ports – viršūnės tipo portai.
      signals – architektūroje naudotų signalų vardų masyvas.
      signalsMap – saugo įėjimo tokia struktūrą: signalo_vardas–>(aibė port_mapų). Bus reikalingas ieškant kelių tarp viršūnių.
    - Grąžina: PortMap objektą.



XAnalyse implementuoja interfeisą ELSSAExceptions. Ši klasė atlikinėja visas analizės operacijas.

Metodai:

- ✓ *compare* – public static CompareInfo compare(XFileInfo x1, XFileInfo x2, javax.swing.JProgressBar progressBar) – Funkcija vykdo palyginimą dviejų struktūrų, ir grąžina palyginimų objektą.
    - Parametrai:
      x1 – pirmoji struktūrqaa kurią reikia palyginti.
      x2 – antroji struktūrqaa kurią reikia palyginti.
    - Grąžina: grąžinamas objektas kuriame saugoma visa informacija apie palygintas struktūras.
- ✓ *compare* – private static void compare(CompareInfo compareInfo, Cube cube1, Cube cube2) – Procedūra lygina du elementus ir randa ieškomus skirtumus.
    - Parametrai:
      compareInfo – šiame objekte saugomi lyginimo rezultatai.
      cube1 – pirmas elementas.
      cube2 – antras elementas.
- ✓ *createStatitic* – public static final java.util.HashMap createStatitic(java.util.ArrayList conditions, XFileInfo xFileInfo) – Procedūra kuri vykdo vienos struktūros statistika. Kokią analizę reikia atlikti nusakoma per parametrą conditions.
    - Parametrai:
      conditions – masyvas sąlygų, kurios sudarys statistiką. Masyve laikoma aibė XCondition objketų.
      xFileInfo – struktūra kurią reikia ištirti.
    - Grąžina: Grąžina statistikos rezultatus.
- ✓ *checkCondition* – private static final boolean checkCondition(Cube cube, XCondition condition, java.util.HashMap statistic) – Procedūra kuri tikrina sąlygą kuri paduodama per parametrą condition.
    - Parametrai:
      cube – struktūros elementas kuriam reikia nustatyti ar jis tenkina sąlygą.
      condition – tikrinimo sąlyga.
      statistic – šiame parametre yra išsaugoma informacija ar sąlyga buvo tenkinama ar ne.
- ✓ *performCondition* – private static final boolean performCondition(int cValue, int oValue, java.lang.String op) – Funkcija vykdo sąlygą su realiais duomenimis.
    - Parametrai:
      cValue – struktūros elemento reikšmė.
      oValue – sąlygos aprašyta reikšmė.
      op – salygos aprašyta operacija.
    - Grąžina:Jeigu sąlyga tenkinama grąžinama TRUE kitu atveju – FALSE.
- ✓ *createStructureInfo* – public static final void createStructureInfo(XFileInfo fileInfo) – Procedūra suskaičiuoja kiek struktūros elementų yra užimta, koks bendras kelių skaičius, ir bndras kelių skaičius kiekvienam įėjimui.
    - Parametrai:
      fileInfo – struktūra kuri analizuojama.

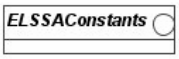


XMLWorker klasė implementuoja interfeisus EXMLNames ir ELSSAExceptions. Ši klasė vykdo visas operacijas susijusias su duomenų konvertavimu ir palyginimu XML formate.

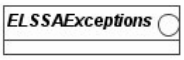
Metodai:

- ✓ *saveAnalyseResults* – public static final void saveAnalyseResults(java.lang.String xml_file_name, java.lang.String html_file_name) throws LSSAException – Procedūra formuoja HTML failą iš rezultatų XML failo.
  - Parametrai:
    xml_file_name – XML failo vardas.
    html_file_name – kuriamo HTML failo vardas.
- ✓ *createCompareXML* – public static final void createCompareXML(boolean showMatrix, XFileInfo fileInfo1, XFileInfo fileInfo2, CompareInfo compareInfo, java.io.File xmlFile) throws LSSAException – Sukuria XML failą dviems palygintoms struktūroms.
  - Parametrai:
    showMatrix – pirmoji struktūra kuri buvo lyginta.
    fileInfo1 – pirmoji struktūra kuri buvo lyginta.
    fileInfo2 – antroji struktūra kuri buvo lyginta.
    compareInfo – palyginimo rezultatai.
    xmlFile – XML failas į kurį bus rašoma informacija.
- ✓ *createCompareXML* – private static final void createCompareXML(CompareInfo compareInfo, int tabCount, java.lang.String NAMESPACE, org.xmlpull.v1.XmlSerializer serializer) throws java.io.IOException – Procedūra formuoja XML'ą palyginimo objektui
  - Parametrai:
    compareInfo – palyginimo objektas, gaunamas atlikus dviejų struktūrų palyginimą.
    tabCount – parametras reikalingas suformuoto XML failo struktūrai išlaikyti.
    NAMESPACE – parametras reikalingas kuriant XML'ui. Visada lygus tuščiai eilutei.
    serializer – parametras į kurį saugoma informacija. Šis parametras yra PP XML parserio objektas.
- ✓ *createXML* – public static final void createXML(boolean showMatrix, XFileInfo fileInfo, java.io.File xmlFile) throws LSSAException – Procedūra formuoja XML failą, kuriame išsaugoma visa elemento analizės metu surasta info.
  - Parametrai:
    showMatrix – loginis kintamasis kuris nusako ar formuojamame XML faile turi būti atvaizduota struktūros matrica.
    fileInfo – struktūra kurią reikia atvaizduoti XML'e.
    xmlFile – XML failas.
- ✓ *createStatisticXML* – private static final void createStatisticXML(java.util.ArrayList conditions, java.util.HashMap statistics, int tabCount, java.lang.String NAMESPACE, org.xmlpull.v1.XmlSerializer serializer) throws java.io.IOException – Procedūra formuoja XML'ą statistikos objektui.
  - Parametrai:
    conditions – statistikos sąlygų masyvas.
    statistics – statistikos rezultatų ryšiai.
    tabCount – parametras reikalingas suformuoto XML failo struktūrai išlaikyti.
    NAMESPACE – parametras reikalingas kuriant XML'ui. Visada lygus tuščiai eilutei.
    serializer – parametras į kurį saugoma informacija. Šis parametras yra PP XML parserio objektas.
- ✓ *tabs* – private static java.lang.String tabs(int tabsCount) – Funkcija padedanti formuoti gražų XML formatą.
  - Parametrai:
    tabsCount – parametras reikalingas suformuoto XML failo struktūrai išlaikyti.
  - Grąžina: grąžinama eilutė sudaryta iš tabuliacijos simbolių. Tabuliacijų skaičius lygus parametro tabsCount skaičiui.
- ✓ *startChild* – private static void startChild(org.xmlpull.v1.XmlSerializer serializer, int tabsCount, java.lang.String NAMESPACE, java.lang.String childName) throws java.io.IOException – Procedūra pradedanti kurti naują XML elementą.
  - Parametrai:
    serializer – parametras į kurį saugoma informacija. Šis parametras yra PP XML parserio objektas.
    tabsCount – parametras reikalingas suformuoto XML failo struktūrai išlaikyti.
    NAMESPACE – parametras reikalingas suformuoto XML failo struktūrai išlaikyti.
    childName – kuriamo elemnto vardas.
- ✓ *endChild* – private static void endChild(org.xmlpull.v1.XmlSerializer serializer, int tabsCount, java.lang.String NAMESPACE, java.lang.String childName) throws java.io.IOException – Procedūra užbaigianti kurti naują XML

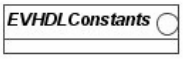
elementą.
    - Parametrai:
      serializer – parametras į kurį saugoma informacija. Šis parametras yra PP XML parserio objektas.
      tabsCount – parametras reikalingas suformuoto XML failo struktūrai išlaikyti.
      NAMESPACE – parametras reikalingas suformuoto XML failo struktūrai išlaikyti.
      childName – užbaigiamo elemnto vardas
- ✓ *addAtribute* – private static void addAtribute(org.xmlpull.v1.XmlSerializer serializer, java.lang.String NAMESPACE, java.lang.String attributeName, java.lang.String attributeValue) throws java.io.IOException – Procedūra kurianti XML elemento atributą.
    - Parametrai:
      serializer – parametras į kurį saugoma informacija. Šis parametras yra PP XML parserio objektas.
      NAMESPACE – parametras reikalingas suformuoto XML failo struktūrai išlaikyti.
      attributeName – naujo atributo vardas.
      attributeValue – atributo reikšmė.
- ✓ *createInfoXML* – private static void createInfoXML(boolean showMatrix, XFileInfo fileInfo, java.util.HashMap colored, int tabCount, java.lang.String NAMESPACE, org.xmlpull.v1.XmlSerializer serializer) throws java.io.IOException – Procedūra formuoja XML failą, kuriame išsaugoma visa elemento analizės metu surasta info. showMatrix loginis kintamasis kuris nusako ar formuojamame XML faile turi būti atvaizduota struktūros matrica.
    - Parametrai:
      fileInfo – struktūra kuriai reikia kurti XML'ą.
      colored – struktūros elementų, kuriuos reikia spalvinti kitokia spalva 'ryšiai'.
      tabCount – parametras reikalingas suformuoto XML failo struktūrai išlaikyti.
      NAMESPACE – parametras reikalingas suformuoto XML failo struktūrai išlaikyti.
      serializer – parametras į kurį saugoma informacija. Šis parametras yra PP XML parserio objektas.



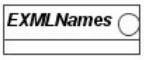
ELSSAConstants interfeise saugomos bendros konstantos naudotos programoje



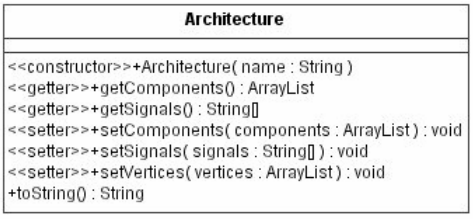
ELSSAExceptions interfeisas, kuriame saugomi visi klaidų pranešimai.



EVHDLConstants interfeisas aprašo VHD naudojamus raktinius žodžius, tokius kaip „entity“, „architecture“, „in“ ir pan.



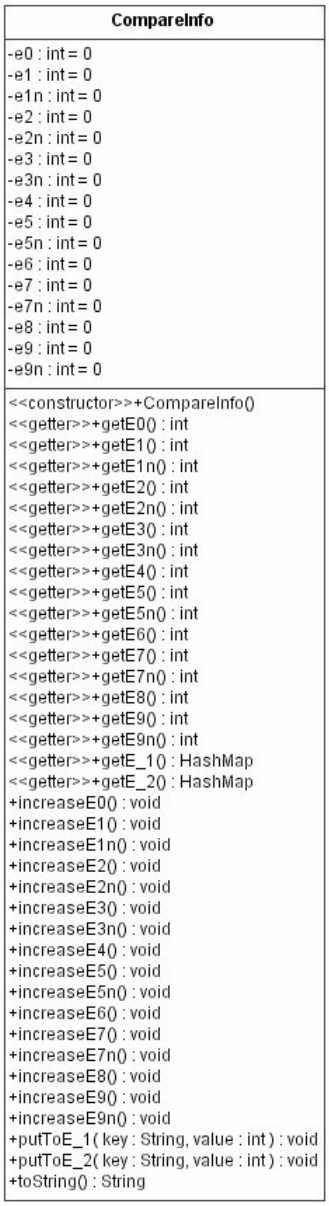
EXMLNames interfeisas kuriame saugomi TAG'ų vardai bei atributų vardai naudojami XML dokumento formavimui.



Architecture klasė kuri saugo informaciją apie schemos architektūrą.

Kintamieji:

- ✓ *name* – private final java.lang.String name – Architektūros vardas.
- ✓ *signals* – private java.lang.String[] signals – Architektūroje naudotų signalų masyvas.
- ✓ *components* – private java.util.ArrayList components – Architektūroje naudotų komponentų masyvas.
- ✓ *vertices* – private java.util.ArrayList vertices – Architektūroje esančių ventilių masyvas.

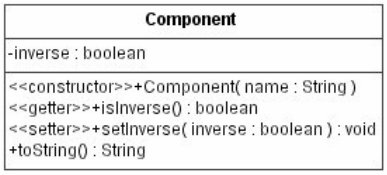


CompareInfo klasė kurioje saugoma visa informacija apie palygintas schemas.

Kintamieji:

- ✓ *e0* – private int e0 – Skaičius kiek abiejose strukūrose elemento lyginių ir nelyginių kelių skaičius lygus 0.
- ✓ *e1* – private int e1 – Skaičius kiek abiejose strukūrose elemento lyginių kelių skaičius lygus 0.
- ✓ *e1n* – private int e1n – Skaičius kiek abiejose strukūrose elemento nelyginių kelių skaičius lygus 0.
- ✓ *e2* – private int e2 – Skaičius kiek pirmoje strukūroje elemento lyginių kelių skaičius lygus 0, o kitoje struktūroje lyginių kelių skaičius didesni už 0.
- ✓ *e2n* – private int e2n – Skaičius kiek pirmoje strukūroje elemento nelyginių kelių skaičius lygus 0, o kitoje struktūroje nelyginių kelių skaičius didesni už 0.
- ✓ *e3* – private int e3 – Skaičius kiek pirmoje strukūroje elemento lyginių kelių skaičius didesnis už 0, o kitoje struktūroje lyginių kelių skaičius lygus 0.

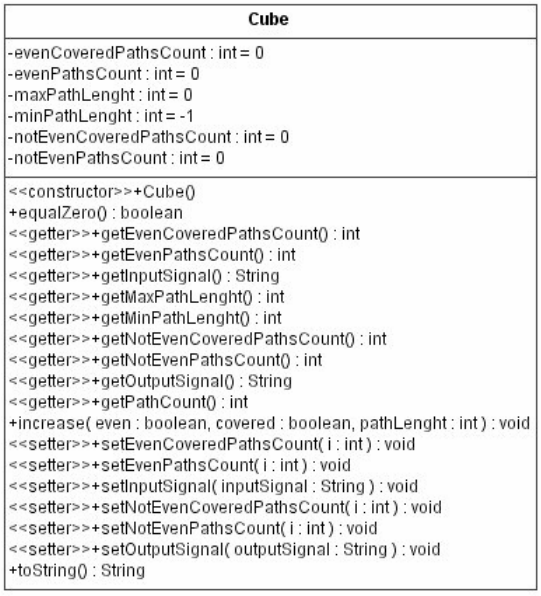
- ✓ *e3n* – private int e3n – Skaičius kiek pirmoje strukūroje elemento nelyginių kelių skaičius didesnis už 0, o kitoje struktūroje nelyginių kelių skaičius lygus 0.
- ✓ *e_1* – private java.util.HashMap e_1 – Sąryšiai tarp tokių elementų kur pirmoje struktūroje kelių skaičius nelygus 0, o kitoje lygus 0.
- ✓ *e_2* – private java.util.HashMap e_2 Sąryšiai tarp tokių elementų kur pirmoje struktūroje kelių skaičius lygus 0, o kitoje nelygus 0.
- ✓ *e4* – private int e4 – Skaičius kiek pirmoje struktūroje bendras kelių skaičius (daugiau už 0) lygus antroje struktūroje bendram kelių skaičiui.
- ✓ *e5* – private int e5 – Skaičius kiek abiejose struktūrose lyginių kelių skaičiai lygūs ir didesnis už 0.
- ✓ *e5n* – private int e5n – Skaičius kiek abiejose struktūrose nelyginių kelių skaičiai lygūs ir didesnis už 0.
- ✓ *e6* – private int e6 – Skaičius kiek pirmoje struktūroje bendras kelių skaičius (daugiau už 0) didesnis už antroje struktūroje bendrą kelių skaičių.
- ✓ *e7* – private int e7 – Skaičius kiek pirmoje struktūroje lyginių kelių skaičius (daugiau už 0) didesnis už antroje struktūroje lyginių kelių skaičių.
- ✓ *e7n* – private int e7n – Skaičius kiek pirmoje struktūroje nelyginių kelių skaičius (daugiau už 0) didesnis už antroje struktūroje nelyginių kelių skaičių.
- ✓ *e8* – private int e8 – Skaičius kiek pirmoje struktūroje bendras kelių skaičius (daugiau už 0) mažesnis už antroje struktūroje bendrą kelių skaičių.
- ✓ *e9* – private int e9 – Skaičius kiek pirmoje struktūroje lyginių kelių skaičius (daugiau už 0) mažesnis už antroje struktūroje lyginių kelių skaičių.
- ✓ *e9n* – private int e9n – Skaičius kiek pirmoje struktūroje nelyginių kelių skaičius (daugiau už 0) mažesnis už antroje struktūroje nelyginių kelių skaičių.



Component klasė paveldi klasę Item. Ši klasė kuri aprašo elementarų loginį elementą. VHDL pvz. "component AND_GATE port(O : out std_logic; I1, I2 : in std_logic); end component;". Komponentės vardas: AND_GATE, ar ji invertuoja signalą nustato jos vardas, portai yra 3: O, I1, I2.

Kintamieji:

- ✓ *inverse* – private boolean inverse – Ar komponentė invertuoja signalą.



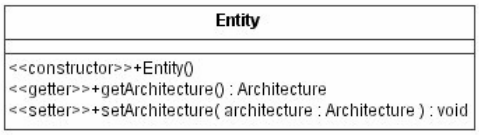
Cube klasė kuri atitinka vieną rezultatų matricos lauką. Klasės kintamieji inputSignal ir outputSignal pasako ryšį tarp elemento įėjimo ir išėjimo.

Kintamieji:

- ✓ *inputSignal* – private java.lang.String inputSignal – Įėjimo signalas.
- ✓ *outputSignal* – private java.lang.String outputSignal – Išėjimo signalas.
- ✓ *evenPathsCount* – private int evenPathsCount – Lyginių kelių skaičius.
- ✓ *notEvenPathsCount* – private int notEvenPathsCount – Nelyginių kelių skaičius.
- ✓ *evenCoveredPathsCount* – private int evenCoveredPathsCount – Lyginių kelių padengimų skaičius.
- ✓ *notEvenCoveredPathsCount* – private int notEvenCoveredPathsCount – Nelyginių kelių padengimų skaičius.
- ✓ *maxPathLenght* – private int maxPathLenght – Maksimalaus kelio ilgis.
- ✓ *minPathLenght* – private int minPathLenght – Minimalaus kelio ilgis.

Metodai:

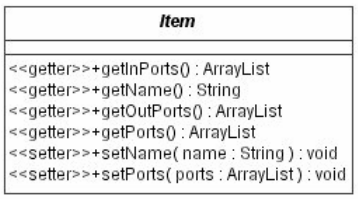
- ✓ *increase* – public final void increase(boolean even, boolean covered, int pathLenght) – Procedūra kviečiama kai randamas inputSignal outputSignal signalų porai naujas kelias. Ji padidina arba lyginių arba nelyginių kelių skaičių, o taip pat padidina lyginio arba nelyginio kelio padengimų skaičių (jeigu kelias buvo padengiantis). Taip pat nustatoma ar naujas kelias yra ilgiausias ar trumpiausias inputSignal outputSignal signalų porai.
  - Parametrai:
    even – kelias lyginis jeigu even = TRUE.
    covered – kelias padengiantis jeigu covered = TRUE.
    pathLenght – kelio ilgis.
- ✓ *getPathCount* – public final int getPathCount()– Funkcija grąžina bendrą kelių skaičių.
  - Grąžina: grąžinama lyginių ir nelyginių kelių suma.
- ✓ *equalZero* – public final boolean equalZero()– Funkcija lygina lyginius ir nelyginius elementus su duotąja reikšme.
  - Grąžina: Gražinama TRUE jeigu lyginis ir nelyginis elementas lygus "x".



Entity klasė saugo informaciją apie faile surastą schemą. Ši klasė implementuoja abstarkčią klasę Item kuri aprašo būdingiausius VHDL naudojamus elementus.

Kintamieji:

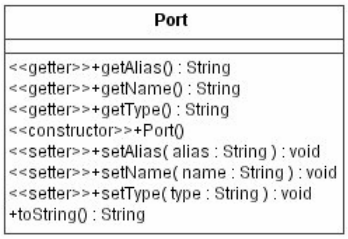
- ✓ *architecture* – private Architecture architecture – Schemos architektūra.



Item yra abstrakti klasė kuri aprašo būdingiausius VHDL naudojamus elementus: elemento vardą, elemento portų masyvą, elemento išėjimo ir įėjimo portų masyvus.

Kintamieji:

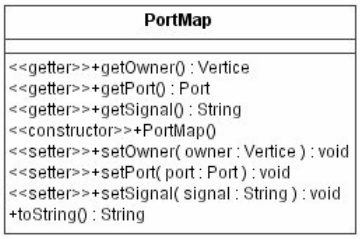
- ✓ *name* – public java.lang.String name – Elemento vardas.
- ✓ *ports* – private java.util.ArrayList ports – Elemento portai.
- ✓ *outPorts* – private java.util.ArrayList outPorts – Elemento išėjimo portų masyvas.
- ✓ *inPorts* – private java.util.ArrayList inPorts – Elemento įėjimo portų masyvas.



Port klasė realizuojanti komponentės portą. VHDL pvz. "port(O : out std_logic; I1, I2 : in std_logic);". Portus atitiks tokie rinkiniai: "O : out std_logic;" (kur porto vardas: O, porto tipas: out), "I1, I2 : in std_logic" (portų vardai: I1 ir I2, portų tipas: in)

Kintamieji:

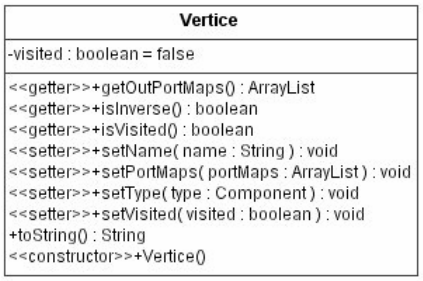
- ✓ *name* – private java.lang.String name – Porto vardas
- ✓ *type* – private java.lang.String type – Porto tipas
- ✓ *alias* – private java.lang.String alias – Architektūroje naudojamas porto vardo pervadinimas.



PortMap klasė sieja komponento realų portą ir architektūros signalą. VHDL pvz. "d1 : AND_GATE port map(O => X_10gat, I1 => X_1gat, I2 => X_3gat);" Šią klasę atitiks 3 rinkiniai: "O => X_10gat", "I1 => X_1gat", "I2 => X_3gat", kur pirmas parametras: O, I1, I2 – realūs komponentės AND_GATE portai, o X_10gat, X_1gat, X_3gat – architektūros signalai.

Kintamieji:

- ✓ *port* – private Port port – Komponentės portas.
- ✓ *signal* – private java.lang.String signal – Architektūros signalas.
- ✓ *owner* – private Vertice owner – Kokiam elemetui priklauso šitas porto ir signalo ryšys.



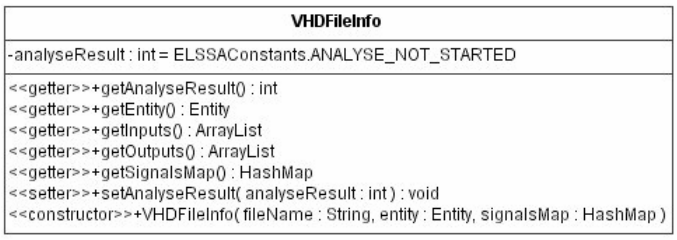
Vertice klasė, kuri simbolizuoja vieną viršūnę(ventilį) architektūroje. VHDL pvz. "d1 : AND_GATE port map(O => X_10gat, I1 => X_1gat, I2 => X_3gat);". Elemento vardas: d1. Elemento tipas: AND_GATE.

Kintamieji:

- ✓ *name* – private java.lang.String name – Viršūnės vardas
- ✓ *portMaps* – private java.util.ArrayList portMaps – Viršūnės ryšių su signalais masyvas. VHDL pvz. "port map(O => X_10gat, I1 => X_1gat, I2 => X_3gat)"
- ✓ *type* – private Component type – Elemento tipas. Nuoroda į viršųnės tipo elementą.
- ✓ *visited* – private boolean visited – Ar viršūnė jau buvo padengtą. Jeigu reikšmė TRUE – buvo padengta. Kitu atveju – nepadengta.
- ✓ *outPortMaps* – private java.util.ArrayList outPortMaps – Masyvas viršūnės ryšių tarp portų ir signalais kur portas yra išeinantis.

Metodai*:*

- ✓ *isInverse* – public final boolean isInverse()– Funkcija pasakanti ar viršūnės elemento tipas yra su inversija.
    - o Grąžina: grąžina TRUE jeigu elemento tipas yra invertuojantis, priešingai grąžinama – FALSE.



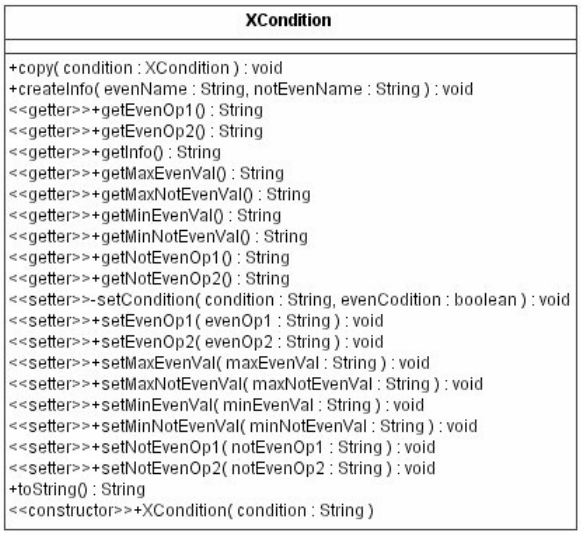
VHDFileInfo klasė saugo iš VHD failo nuskaitytą informaciją. Ši klasė yra vaikinė klasei XFileInfo.

Kintamieji:

- ✓ *entity* – private final Entity entity – Nuskaityto VHD failo pagrindinis elementas.
- ✓ *signalsMap* – private final java.util.HashMap signalsMap – Saugo įėjimo tokia struktūrą: signalo_vardas–>(aibė port_mapų). Bus reikalingas ieškant.
- ✓ *analyseResult* – private int analyseResult – Ar VHD failo analizė pasibaigė ar buvo nutraukta vartotojo arba laiko limito.

Metodai:

- ✓ *VHDFileInfo* – public VHDFileInfo(java.lang.String fileName, Entity entity, java.util.HashMap signalsMap) – Konstruktorius.
  - Parametrai:
    fileName – failo vardas iš kurio buvo nuskaitinėjami duomenys.
    entity – pagrindinis VHD failo elementas.
    signalsMap – saugo įėjimo tokia struktūrą: signalo_vardas–>(aibė port_mapų). Bus reikalingas ieškant. kelių tarp viršūnių.
- ✓ *getInputs* – public final java.util.ArrayList getInputs()– Funkcija grąžina pagrindinio elemento įėjimo portų masyvą.
  - Grąžina: grąžinamas pagrindinio elemento įėjimo portų masyvas.
- ✓ *getOutputs* – public final java.util.ArrayList getOutputs()– Funkcija grąžina pagrindinio elemento išėjimo portų masyvą.
  - Grąžina: grąžinamas pagrindinio elemento išėjimo portų masyvas.



XCondition klasė kuri saugo informaciją apie norimą atlikti statistiką.

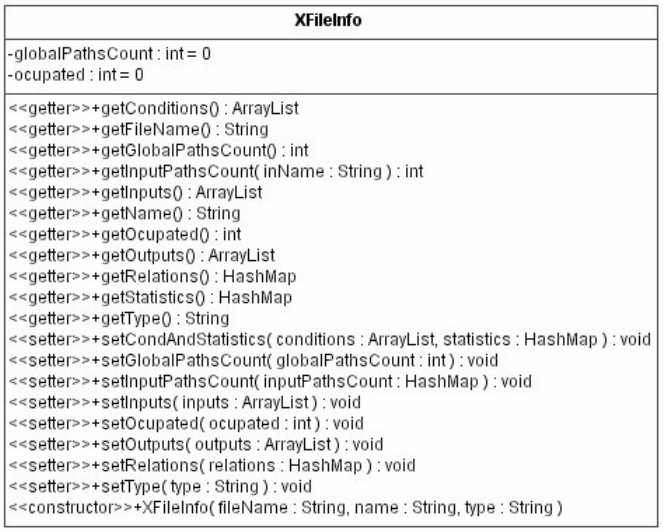
Kintamieji:

- ✓ *maxEvenVal* – private java.lang.String maxEvenVal – Maksimali galima įgyti lyginio elemento reikšmė.
- ✓ *minEvenVal* – private java.lang.String minEvenVal – Minimali galima įgyti lyginio elemento reikšmė.
- ✓ *evenOp1* – private java.lang.String evenOp1 – Koks galimas ryšys tarp maksimalios lyginio elemento reikšmės ir realios lyginės reikšmes. Galimos reikšmės: =; <; >; <=; >=; <>; !=
- ✓ *evenOp2* – private java.lang.String evenOp2 – Koks galimas ryšys tarp minimalios lyginio elemento reikšmės ir realios lyginės reikšmes. Galimos reikšmės: =; <; >; <=; >=; <>; !=
- ✓ *maxNotEvenVal* – private java.lang.String maxNotEvenVal – Maksimali galima įgyti nelyginio elemento reikšmė.

- ✓ *minNotEvenVal* – private java.lang.String minNotEvenVal – Minimali galima įgyti nelyginio elemento reikšmė.
- ✓ *notEvenOp1* – private java.lang.String notEvenOp1 – Koks galimas ryšys tarp maksimalios nelyginio elemento reikšmės ir realios nelyginės reikšmes. Galimos reikšmės: =; <; >; <=; >=; <>; !=
- ✓ *notEvenOp2* – private java.lang.String notEvenOp2 – Koks galimas ryšys tarp minimalios lyginio elemento reikšmės ir realios nelyginės reikšmes. Galimos reikšmės: =; <; >; <=; >=; <>; !=
- ✓ *info* – private java.lang.String info – Trumpas sąlygos aprašymas.

Metodai:

- ✓ *XCondition* – public XCondition(java.lang.String condition) – Konstruktorius. Sukuria sąlygos objektą. Trumpas sąlygos aprašymas priskiriamas sąlygos parametrui. Konstruktorius analizuoja paduotą salygą. Ją išskaido į dvi sąlygas, kurios atskirtos ";".
    - Parametrai:
      condition – sąlyga. pvz ("LKS > 0; NKS = 1" arba "0 > LKS > 10; 0 > NKS >= 5;").



XFileInfo klasė kuri saugos nuskaitytų iš failo duomenų failų informaciją.

Kintamieji:

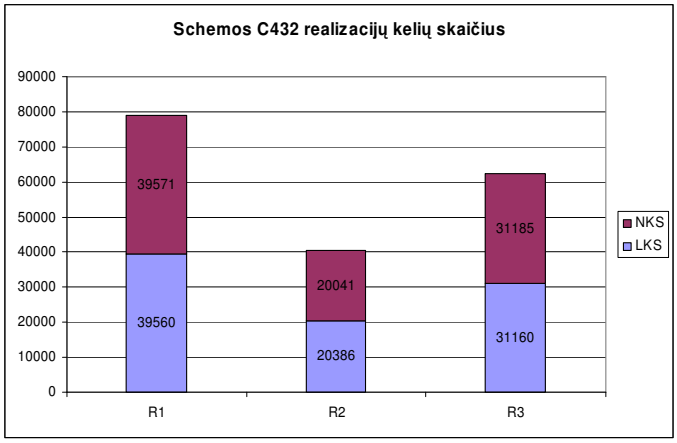
- ✓ *name* – private final java.lang.String name – Vardas tiriamos struktūros.
- ✓ *fileName* – private final java.lang.String fileName – Failo vardas kuriame buvi ši schema.
- ✓ *inputs* – private java.util.ArrayList inputs – Nuskaitytų iš failo įėjimų vardų masyvas.
- ✓ *outputs* – private java.util.ArrayList outputs – Nuskaitytų iš failo išėjimų vardų masyvas.
- ✓ *relations* – private java.util.HashMap relations – Nuskaitytų iš failo ryšių tarp įėjimų ir išejimų saryšiai.
- ✓ *conditions* – private java.util.ArrayList conditions – Masyvas sąlygų kurios bus reikalaingos atlikti statistiką.
- ✓ *statistics* – private java.util.HashMap statistics – Statistikos rezultatai. (sąlyga(XCondition)–>kiekis(Integer)).
- ✓ *type* – private java.lang.String type – Kokio tipo duomenys buvo nuskaityti į šią struktūrą.
- ✓ *globalPathsCount* – private int globalPathsCount – Koks bendras keliu skaičius.
- ✓ *ocupated* – private int ocupated – Kiek yra netuščių matricos langelių struktūroje.
- ✓ *inputPathsCount* – private java.util.HashMap inputPathsCount – Kiek vienas iėjimas turi kelių.

10.3 Priedas C

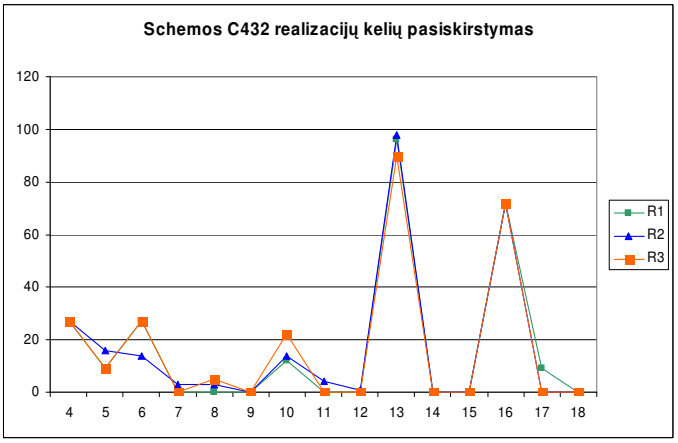
10.3.1 Schema C432

13. lentelė Kelių pasiskirstymas schemos C432 realizacijose.

Sąlygos	R1	R2	R3
Matricos elementu skaičius	252	252	252
Užpildytų matricos elementu dalių skaičius	225	225	225
Bendras keliu skaičius	79131	40427	62345
LKS = 0; NKS = 0	27	27	27
LKS = 1; NKS = 0	9	16	9
LKS = 0; NKS = 1	27	14	27
LKS = 1; NKS = 1	0	3	0
1 <= LKS <= 5; NKS = 0	0	3	5
LKS = 0; 1 <= NKS <= 5	0	0	0
1 <= LKS <= 5; 1 <= NKS <= 5	12	14	22
5 <= LKS <= 100; NKS = 0	0	4	0
LKS = 0; 5 <= NKS <= 100	0	1	0
0 <= LKS <= 100; 0 <= NKS <= 100	96	98	90
LKS >= 100; NKS = 0	0	0	0
LKS = 0; NKS >= 100	0	0	0
LKS >= 100; NKS >= 100	72	72	72
LKS >= 100; NKS < 100	9	0	0
LKS < 100; NKS >= 100	0	0	0



7. grafikas Kelių skaičius schemos C432 realizacijose.

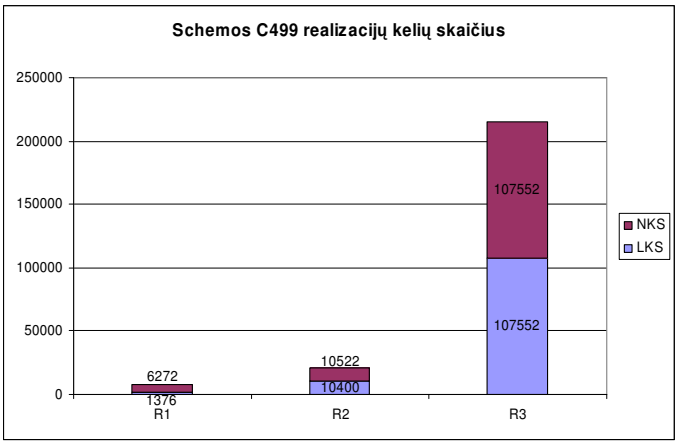


8. grafikas Kelių skaičius schemos C432 realizacijose.

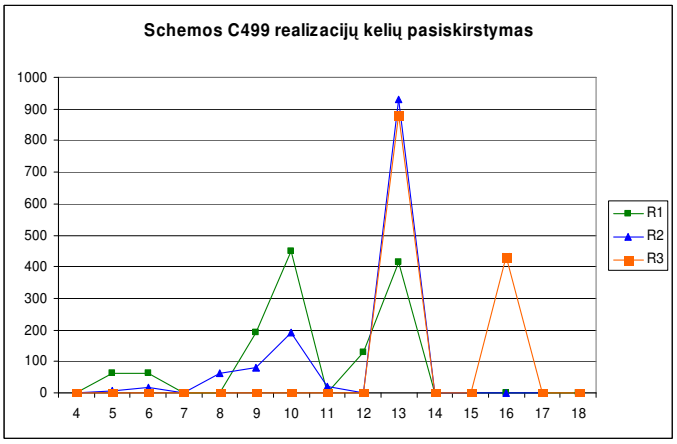
10.3.2 Schema C499

Paveikslas Nr. 4 Kelių pasiskirstymas schemos C499 realizacijose.

Sąlygos	R1	R2	R3
Matricos elementu skaičius	1312	1312	1312
Užpildytų matricos elementu dalių skaičius	1312	1312	1312
Bendras keliu skaičius	7648	20922	215104
LKS = 0; NKS = 0	0	0	0
LKS = 1; NKS = 0	64	8	0
LKS = 0; NKS = 1	64	16	0
LKS = 1; NKS = 1	0	0	0
1 <= LKS <= 5; NKS = 0	0	64	0
LKS = 0; 1 <= NKS <= 5	192	80	0
1 <= LKS <= 5; 1 <= NKS <= 5	448	192	0
5 <= LKS <= 100; NKS = 0	0	22	0
LKS = 0; 5 <= NKS <= 100	128	0	0
0 <= LKS <= 100; 0 <= NKS <= 100	416	930	880
LKS >= 100; NKS = 0	0	0	0
LKS = 0; NKS >= 100	0	0	0
LKS >= 100; NKS >= 100	0	0	432
LKS >= 100; NKS < 100	0	0	0
LKS < 100; NKS >= 100	0		0



9. grafikas Kelių skaičius schemos C499 realizacijose.

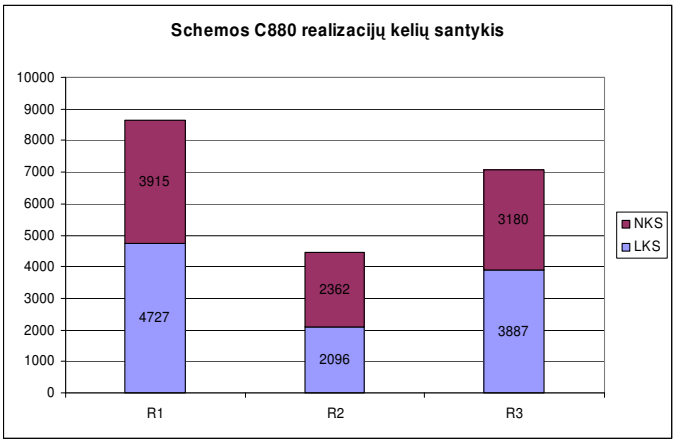


10. grafikas Kelių skaičius schemos C499 realizacijose.

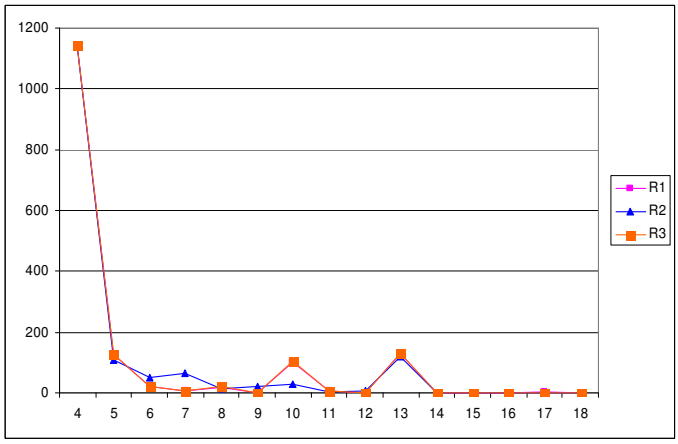
10.3.3 Schema C880

Paveikslas Nr. 5 Kelių pasiskirstymas schemos C880 realizacijose.

Sąlygos	R1	R2	R3
Matricos elementu skaičius	1560	1560	1560
Užpildytų matricos elementu dalių skaičius	419	419	419
Bendras keliu skaičius	8642	4458	7067
LKS = 0; NKS = 0	1141	1141	1141
LKS = 1; NKS = 0	126	107	126
LKS = 0; NKS = 1	20	50	20
LKS = 1; NKS = 1	8	66	8
1 <= LKS <= 5; NKS = 0	19	15	21
LKS = 0; 1 <= NKS <= 5	0	22	0
1 <= LKS <= 5; 1 <= NKS <= 5	103	30	106
5 <= LKS <= 100; NKS = 0	9	5	7
LKS = 0; 5 <= NKS <= 100	1	6	1
0 <= LKS <= 100; 0 <= NKS <= 100	130	118	130
LKS >= 100; NKS = 0	0	0	0
LKS = 0; NKS >= 100	0	0	0
LKS >= 100; NKS >= 100	1	0	0
LKS >= 100; NKS < 100	2	0	0
LKS < 100; NKS >= 100	0	0	0



11. grafikas Kelių skaičius schemos C880 realizacijose.

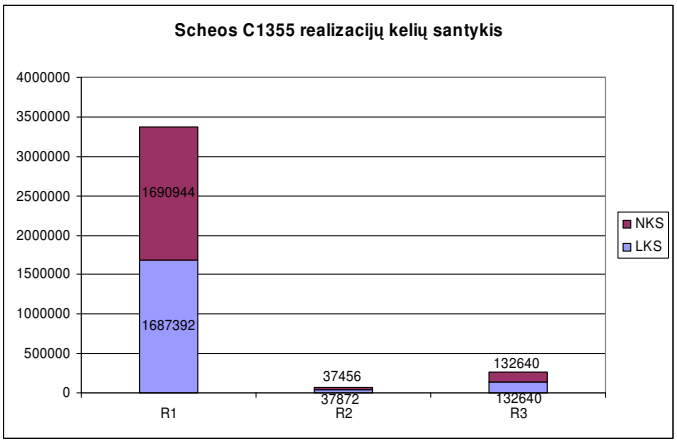


12. grafikas Kelių skaičius schemos C880 realizacijose.

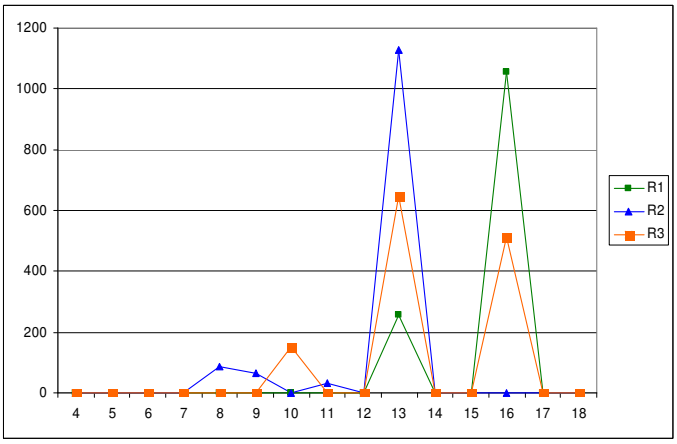
10.3.4 Schema C1355

Paveikslas Nr. 6 Kelių pasiskirstymas schemos C1355 realizacijose.

Sąlygos	R1	R2	R3
Matricos elementu skaičius	1312	1312	1312
Užpildytų matricos elementu dalių skaičius	1312	1312	1312
Bendras keliu skaičius	3378336	75328	265280
LKS = 0; NKS = 0	0	0	0
LKS = 1; NKS = 0	0	0	0
LKS = 0; NKS = 1	0	0	0
LKS = 1; NKS = 1	0	0	0
1 <= LKS <= 5; NKS = 0	0	88	0
LKS = 0; 1 <= NKS <= 5	0	64	0
1 <= LKS <= 5; 1 <= NKS <= 5	0	0	152
5 <= LKS <= 100; NKS = 0	0	32	0
LKS = 0; 5 <= NKS <= 100	0	0	0
0 <= LKS <= 100; 0 <= NKS <= 100	256	1128	648
LKS >= 100; NKS = 0	0	0	0
LKS = 0; NKS >= 100	0	0	0
LKS >= 100; NKS >= 100	1056	0	512
LKS >= 100; NKS < 100	0	0	0
LKS < 100; NKS >= 100	0	0	0



13. grafikas Kelių skaičius schemos C1355 realizacijose.

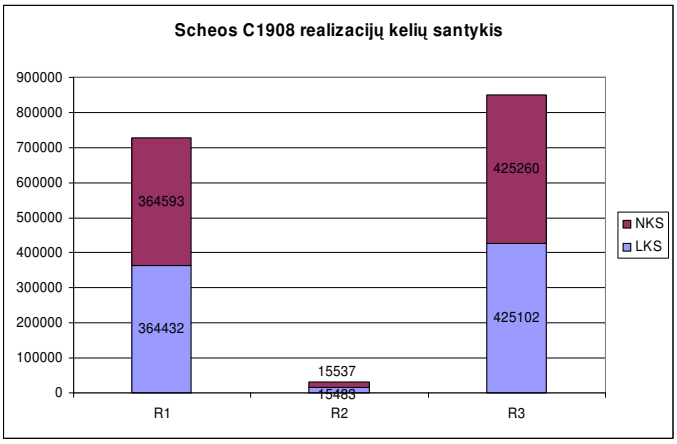


14. grafikas Kelių skaičius schemos C1355 realizacijose.

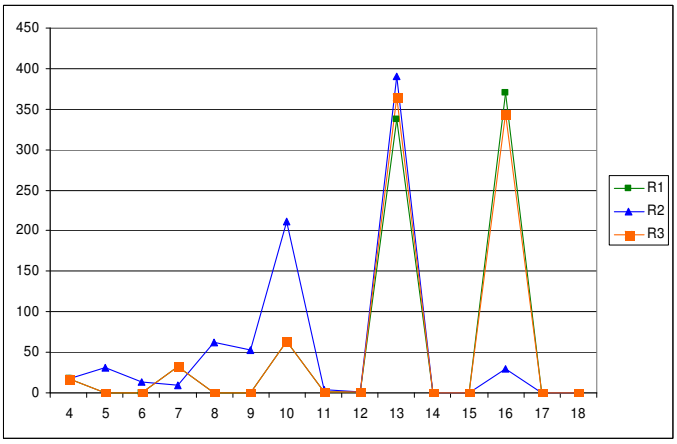
10.3.5 Schema C1908

Paveikslas Nr. 7 Kelių pasiskirstymas schemos C1908 realizacijose.

Sąlygos	R1	R2	R3
Matricos elementu skaičius	825	825	825
Užpildytų matricos elementu dalių skaičius	807	807	807
Bendras keliu skaičius	729025	31020	850362
LKS = 0; NKS = 0	18	18	18
LKS = 1; NKS = 0	0	31	0
LKS = 0; NKS = 1	0	14	0
LKS = 1; NKS = 1	32	9	32
1 <= LKS <= 5; NKS = 0	0	63	0
LKS = 0; 1 <= NKS <= 5	0	53	0
1 <= LKS <= 5; 1 <= NKS <= 5	64	211	64
5 <= LKS <= 100; NKS = 0	1	4	1
LKS = 0; 5 <= NKS <= 100	2	2	2
0 <= LKS <= 100; 0 <= NKS <= 100	338	390	364
LKS >= 100; NKS = 0	0	0	0
LKS = 0; NKS >= 100	0	0	0
LKS >= 100; NKS >= 100	370	30	344
LKS >= 100; NKS < 100	0	0	0
LKS < 100; NKS >= 100	0	0	0



15. grafikas Kelių skaičius schemos C1908 realizacijose.

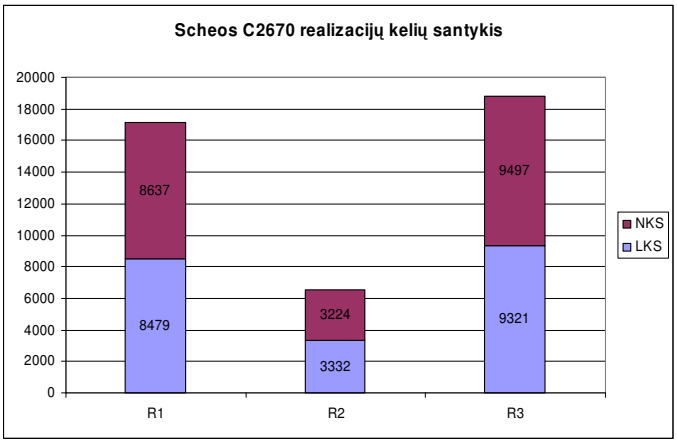


16. grafikas Kelių skaičius schemos C1908 realizacijose.

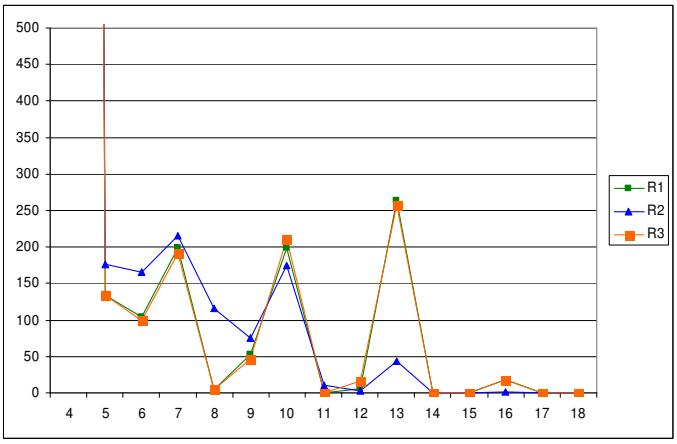
10.3.6 Schema C2670

Paveikslas Nr. 8 Kelių pasiskirstymas schemos C2670 realizacijose.

Sąlygos	R1	R2	R3
Matricos elementu skaičius	9891	9891	9891
Užpildytų matricos elementu dalių skaičius	981	981	981
Bendras keliu skaičius	17116	6556	18818
LKS = 0; NKS = 0	8910	8910	8910
LKS = 1; NKS = 0	134	176	134
LKS = 0; NKS = 1	104	166	100
LKS = 1; NKS = 1	199	215	192
1 <= LKS <= 5; NKS = 0	5	116	5
LKS = 0; 1 <= NKS <= 5	53	75	46
1 <= LKS <= 5; 1 <= NKS <= 5	199	174	211
5 <= LKS <= 100; NKS = 0	0	11	0
LKS = 0; 5 <= NKS <= 100	6	3	17
0 <= LKS <= 100; 0 <= NKS <= 100	263	43	258
LKS >= 100; NKS = 0	0	0	0
LKS = 0; NKS >= 100	0	0	0
LKS >= 100; NKS >= 100	18	2	18
LKS >= 100; NKS < 100	0	0	0
LKS < 100; NKS >= 100	0	0	0



17. grafikas Kelių skaičius schemos C2670 realizacijose.

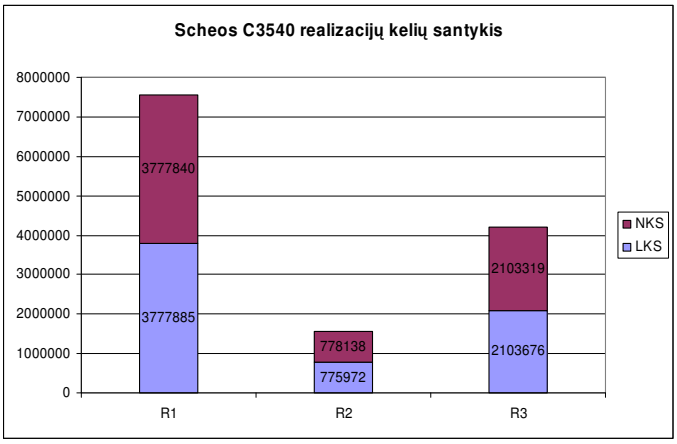


18. grafikas Kelių skaičius schemos C2670 realizacijose.

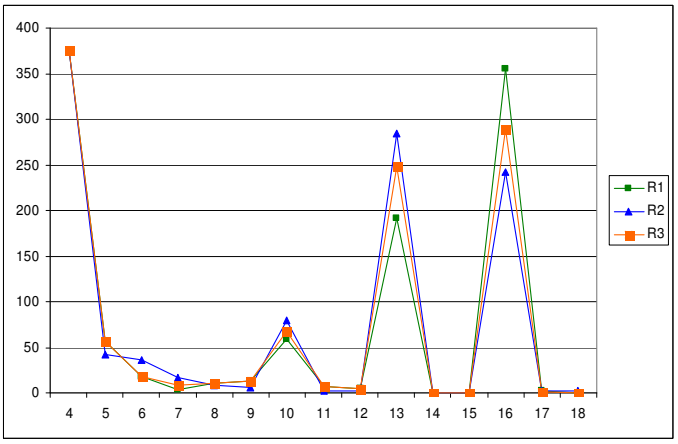
10.3.7 Schema C3540

Paveikslas Nr. 9 Kelių pasiskirstymas schemos C3540 realizacijose.

Sąlygos	R1	R2	R3
Matricos elementu skaičius	1100	1100	1100
Užpildytų matricos elementu dalių skaičius	724	724	724
Bendras keliu skaičius	7555725	1554110	4206995
LKS = 0; NKS = 0	376	376	376
LKS = 1; NKS = 0	57	42	57
LKS = 0; NKS = 1	18	36	18
LKS = 1; NKS = 1	4	17	8
1 <= LKS <= 5; NKS = 0	11	8	11
LKS = 0; 1 <= NKS <= 5	13	6	13
1 <= LKS <= 5; 1 <= NKS <= 5	59	79	67
5 <= LKS <= 100; NKS = 0	7	3	7
LKS = 0; 5 <= NKS <= 100	5	2	5
0 <= LKS <= 100; 0 <= NKS <= 100	192	284	248
LKS >= 100; NKS = 0	0	0	0
LKS = 0; NKS >= 100	0	0	0
LKS >= 100; NKS >= 100	356	242	289
LKS >= 100; NKS < 100	2	2	1
LKS < 100; NKS >= 100	0	3	0



19. grafikas Kelių skaičius schemos C3540 realizacijose.

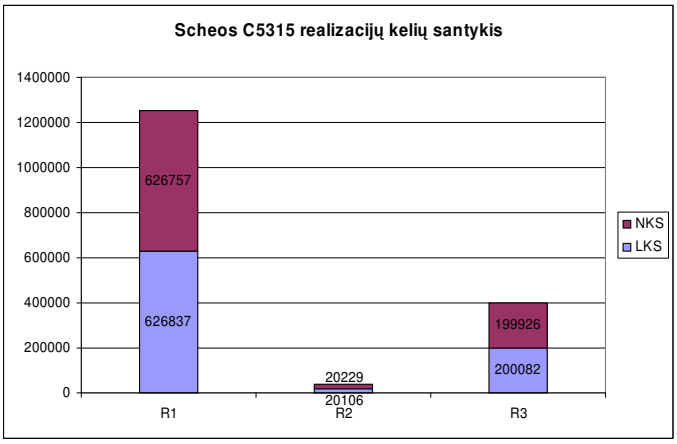


20. grafikas Kelių skaičius schemos C3540 realizacijose.

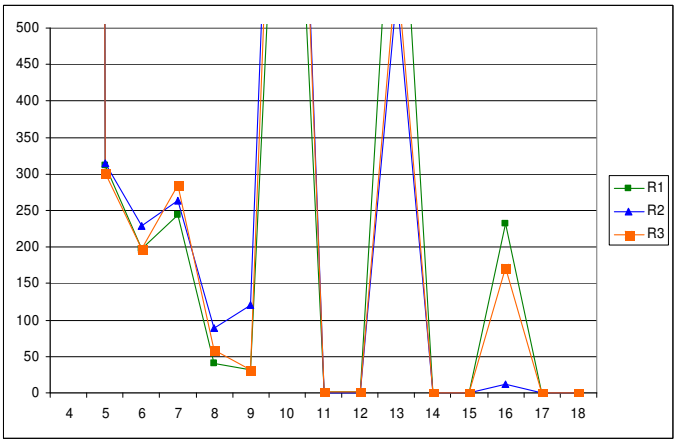
10.3.8 Schema C5315

Paveikslas Nr. 10 Kelių pasiskirstymas schemos C5315 realizacijose.

Sąlygos	R1	R2	R3
Matricos elementu skaičius	21894	21894	21894
Užpildytų matricos elementu dalių skaičius	2977	2977	2977
Bendras keliu skaičius	1253594	40335	400008
LKS = 0; NKS = 0	18917	18917	18917
LKS = 1; NKS = 0	311	315	301
LKS = 0; NKS = 1	197	229	197
LKS = 1; NKS = 1	244	263	284
1 <= LKS <= 5; NKS = 0	40	89	58
LKS = 0; 1 <= NKS <= 5	32	120	32
1 <= LKS <= 5; 1 <= NKS <= 5	1074	1402	1344
5 <= LKS <= 100; NKS = 0	2	0	2
LKS = 0; 5 <= NKS <= 100	2	0	2
0 <= LKS <= 100; 0 <= NKS <= 100	843	547	585
LKS >= 100; NKS = 0	0	0	0
LKS = 0; NKS >= 100	0	0	0
LKS >= 100; NKS >= 100	232	12	172
LKS >= 100; NKS < 100	0	0	0
LKS < 100; NKS >= 100	0	0	0



21. grafikas Kelių skaičius schemos C5315 realizacijose.

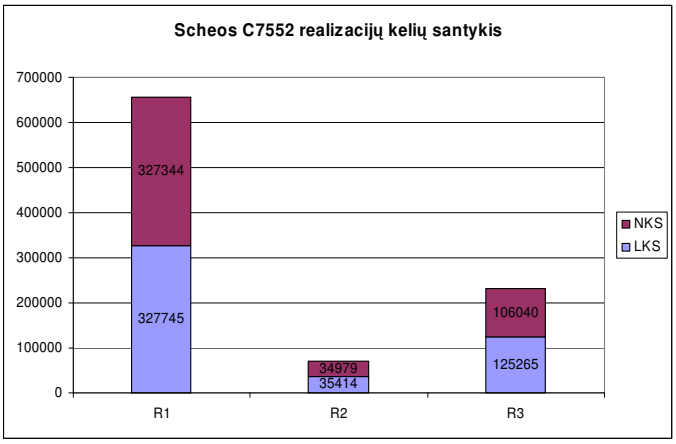


22. grafikas Kelių skaičius schemos C5315 realizacijose.

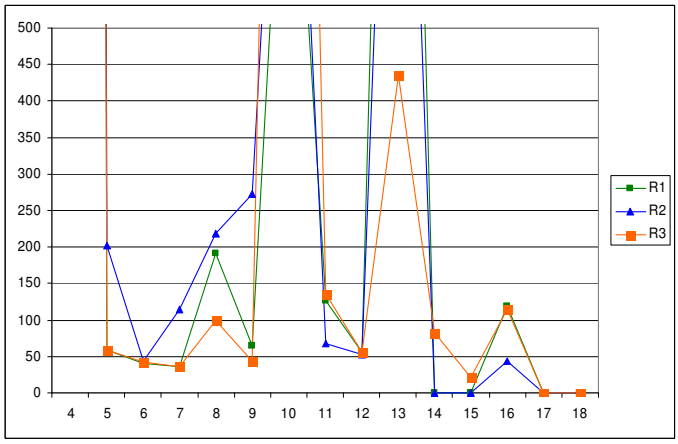
10.3.9 Schema C7552

Paveikslas Nr. 11 Kelių pasiskirstymas schemos C7552realizacijose.

Sąlygos	R1	R2	R3
Matricos elementu skaičius	22042	22042	22042
Užpildytų matricos elementu dalių skaičius	3543	3543	3543
Bendras keliu skaičius	655089	70393	231305
LKS = 0; NKS = 0	18499	18499	18499
LKS = 1; NKS = 0	58	202	58
LKS = 0; NKS = 1	41	43	42
LKS = 1; NKS = 1	36	114	36
1 <= LKS <= 5; NKS = 0	192	218	99
LKS = 0; 1 <= NKS <= 5	65	273	44
1 <= LKS <= 5; 1 <= NKS <= 5	950	1163	2419
5 <= LKS <= 100; NKS = 0	126	68	136
LKS = 0; 5 <= NKS <= 100	56	52	55
0 <= LKS <= 100; 0 <= NKS <= 100	1900	1366	435
LKS >= 100; NKS = 0	0	0	83
LKS = 0; NKS >= 100	0	0	21
LKS >= 100; NKS >= 100	119	44	115
LKS >= 100; NKS < 100	0	0	0
LKS < 100; NKS >= 100	0	0	0



23. grafikas Kelių skaičius schemos C7552 realizacijose.



24. grafikas Kelių skaičius schemos C7552 realizacijose.