KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Tomas Žemaitis

LOGINĖS FUNKCIJOS TERMŲ GENERAVIMO ALGORITMAS PAGRĮSTAS PROGRAMINIO PROTOTIPO MODELIU

Magistro darbas

Darbo vadovas
prof. habil. dr.
Rimantas Šeinauskas

KAUNAS, 2007

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Tomas Žemaitis

LOGINĖS FUNKCIJOS TERMŲ GENERAVIMO ALGORITMAS PAGRĮSTAS PROGRAMINIO PROTOTIPO MODELIU

Magistro darbas

Vadovas
prof. habil. dr. Rimantas Šeinauskas
2007-05-22

Recenzentas
dr. Kęstutis Motiejūnas
2007-05-24

Atliko
IFM-1/2 gr. stud.
Tomas Žemaitis
2007-05-21

KAUNAS, 2007

TERMS' OF LOGICAL FUNCTION GENERATION ALGORITHM BASED ON SOFTWARE PROTOTYPE MODEL

Summary

The technological development is enabling production of increasingly complex electronic systems. All those systems must be verified and tested to guarantee correct behavior. As the complexity grows, testing is becoming one of the most significant factors that contribute to the final product cost. The established low-level methods for hardware testing are not any more sufficient and more work has to be done at abstraction levels higher than the classical gate and register-transfer levels.

Realized algorithm, which random generates inputs, computes reaction based on software prototype model and deforming values of inputs one by one determines possible terms of logical functions. Analyzing other inputs determined terms of logical functions are corrected by eliminating partial terms. After random generating and analyzing a lot of inputs terminal terms of logical functions are derived. Algorithm doesn't guarantee that all and exact terms of logical functions are obtained but those terms could be used when generating test vectors. Derived terms of logical functions' outputs are recorded with input that formed them and following terms can be used as inspecting tests. Collected results can be used for further researches: schemes testing, defect detection, comparing elements of logical function, improving algorithm.

Main aspects of design are introduced. Experimental accurateness of results and factors (initial number of random generated test vectors, improvement coefficient, maximum terms number of one output, etc.) that influences algorithm computation progress and quality of results are examined. For improving algorithm a novel hierarchical test generation approach is used: task of functional model is divided into several smaller tasks and their results are combined. Experimental progresses and results of performed analysis' (hierarchical and common test generation approaches) are compared.

In conclusion opinion about important questions is pithily stated: accepted design solutions, point of experiments and results.

TURINYS

1. ĮVADAS

1.1. SANTRAUKA

Technologijų plėtojimas leidžia vis labiau vystytis sudėtingų elektroninių sistemų produkcijai. Visos šios sistemos turi būti patikrintos ir ištestuotos tam, kad užtikrinti tikslų jų funkcionavimą. Kai sistemų sudėtingumas didėja, testavimas tampa vienas iš svarbiausių faktorių nustatant galutinę produkto kainą. Žinomų žemo lygio metodų, skirtų techninės įrangos testavimui, nepakanka ir daugiau darbo turi būti atlikta abstrakčiame lygyje pradiniuose projektavimo etapuose negu klasikiniame ventiliniame ir registrų perdavimo lygiuose.

Realizuotas algoritmas, kuris atsitiktinai generuoja įėjimo poveikį, pagal programinio prototipo modelį paskaičiuoja poveikio reakciją ir iškraipant po vieną įėjimo poveikio signalo reikšmę apibrėžia galimus išėjimų loginių funkcijų termus. Nagrinėjant kitus įėjimo poveikius apibrėžti išėjimų loginių funkcijų termai patikslinami išmetant dalinius termus. Atsitiktinai sugeneravus ir išnagrinėjus daug įėjimo poveikių gaunami galutiniai išėjimų loginių funkcijų termai. Algoritmas negarantuoja , kad bus gauti visi ir tikslūs išėjimų loginių funkcijų termai, bet gauti termai gali būti naudojami testų generavimui. Gauti išėjimų loginių funkcijų termai užrašomi kartu su įėjimo poveikiu, pagal kurį termas buvo nustatytas, ir patys paskaičiuoti termai jau gali būti naudojami kaip tikrinantys testai. Gauti rezultatai galės būti panaudoti tolimesniems tyrimams: schemų testavimui, defektų šalinimui, funkcijos elementų palyginimui, algoritmo gerinimui.

Pateikiami techninės-projektinės realizacijos esminiai aspektai. Eksperimentų pagalba tikrinamas rezultatų tikslumas ir tiriami veiksniai (pradinis generuojamų poveikių kiekis, pagerėjimo koeficientas, maksimalus vieno išėjimo termų kiekis ir kiti) įtakojantys algoritmo skaičiavimų eigą, rezultatų kokybę. Algoritmo patobulinimui taikoma hierarchinė testų sudarymo metodika: analizuojamo funkcinio modelio uždavinys yra skaidomas į keletą mažesnių uždavinių, gauti rezultatai apjungiami. Eksperimentų metu atliktų analizių (neskaidant testinių vektorių generavimo uždavinio ir skaidant) eigos ir gauti rezultatai palyginami.

Išvadose glaustai išdėstoma nuomonė apie svarbiausius klausimus: projektavimo metu priimtus sprendimus, atliktų tyrimų esmę ir gautus rezultatus.

2. ANALITINĖ DALIS

2.1. ĮŽANGA

Vykstant technologijų plėtrai didėja sudėtingų elektroninių sistemų produkcija. Sukurtos sistemos turi būti ištestuotos užtikrinant jų teisingą funkcionavimą. Didėjantis sistemų sudėtingumas turi įtakos elektroninių sistemų testavimui. Jis tampa sudėtingesnis ir reikalauja vis daugiau kaštų. Tai įtakoja galutinę produkto kainą. Žinomos žemo lygio metodikos, skirtos techninės įrangos testavimui, remiasi elektroninių sistemų struktūra. Tačiau jų nepakanka ir turi būti kuriamos naujos metodikos, kurios vykdytų testinių duomenų generavimą aukštesniame abstrakcijos lygyje.

Nepaisant fakto, kad nauji projektavimo automatizuoti įrankiai sudaro sąlygas projektuotojams dirbti aukštesniuose abstrakcijos lygiuose, su testavimu susijusi veikla vis dar yra iš esmės atliekama žemesniuose abstrakcijos lygiuose. Tuo pačiu metu testavimas tampa viena iš daugiausiai laiko ir resursų reikalaujančia užduotimi elektroninės sistemos plėtojimo ir gamybos cikle. Taigi įprasti įėjimų lygio metodai jau nėra tinkami dabartiniu metu ir testavimo veikla turi būti perkelta į aukštesnį abstrakcijos lygį. Taip pat yra labai svarbu kad visos modeliavimo užduotys gali būti atliktos su kruopščiu sistemos pilno testavimo apsvarstymu [1].

Patikimos elektroninės sistemos yra reikalingos ne tik tose srityse, kur gedimai gali sąlygoti katastrofiškas pasekmes, bet taip pat vis labiau poreikis didėja kitų sferų pritaikomume. Pagrindinis reikalavimas pasiekti patikimą elektroninę sistemą yra galimybė pripažinti, jog sistema yra be klaidų. Todėl didelis dėmesys turi būti skirtas testavimui [2]. Testavimas yra vienas iš etapų, kuriam yra skiriama daug išlaidų integrinių schemų kūrimo ir gamybos procesuose, apimantis iki 35% visų išlaidų. Testavimas, diagnozės ir klaidų taisymas sudėtingų elektroninių sistemų siekia 40-50% galutinės produkto realizacijos kainos ir labai greitai pramonė susidurs su sunkumu, jog tranzistoriaus testavimas bus daug brangesnis nei jo gamyba [3].

2.2. SKAITMENINIŲ SISTEMŲ TESTAVIMAS

Nors elektroninę sistemą sudaro techninė bei programinė įrangos, tačiau šios sistemos pagrindinis dėmesys yra sutelktas į skaitmeninės sistemos testavimą. Techninės įrangos testavimas – procesas, kuris randa gamybos, nusidėvėjimo, aplinkos poveikio ir kitus gedimus. Tai gali būti atlikta tik tuomet, kai sistema yra realizuota, stimuliuojant jos veikimą ir tikrinant jos reakciją. Tokių poveikių generavimas su apskaičiavimu tikėtinų atsakymų yra vadinamas testinių modelių generavimu.

Testinių modelių generavimas priklauso apskaičiavimų problemos klasei. Yra išvystyta keletas būdų valdyti testinių vektorių generavimą palyginti didelėm schemom, tačiau jie yra nepakankami.

Testų generavimas didelėms nuoseklioms schemoms išlieka neišspręsta problema nepaisant spartaus skaičiavimo didėjimo galios [11].

Remiantis [4] testavimo technologijos gali būti klasifikuojamos į tokias klases:

- Funkcinis testavimas (testuojamas įrenginys normaliame darbo režime bei jo projektuojamame darbo dažnyje) nustato ar schema veikia pagal specifikaciją;
- Testavimas nuolatinių struktūrinių defektų, kurie nereikalauja schemos funkcionavimo projektiniu dažniu testo metu;
- Testavimas pagrįstas sąveikų klaidų nagrinėjimu, kur klaidos atsiranda iš defektų generavimo mechanizmų simuliacijos integruotoje schemoje (tokie defektai linkę būti pastoviai ir nereikalauja schemos testavimo projektiniame dažnyje);
- Vėlinimo defektų testavimas, kurio metu schema turi funkcionuoti projektiniu dažniu;
- Paplitusiais matavimais grįstos testavimo metodikos, kurios dažniausiai aptinka klaidas schemose vertinant išgautas reakcijas po skirtingų įėjimo signalų būsenų kai schema yra pastovioje būklėje.

2.3. KLAIDŲ IR DEFEKTŲ MODELIAI

Gedimas yra apibrėžtas kaip neteisinga schemos reakcija elgsenoje. Remiantis [4] gedimai yra dviejų tipų:

- Fizikinės/projektavimo srities: defektai (jie sukelia specifikacijos nukrypimus)
    - Įrenginių lygyje: elementų gedimai, dielektrikų gedimai, priemaišos, trumpi sujungimai sąsajose, nepakankamas sulitavimas ir kt.
    - Schemos lygyje: trūkstama detalė, klaidingas komponentas, sugedę ryšiai ir kt.
    - Neteisingo projektavimo (funkcinis defektas).
    - Nusidėvėjimo/aplinkos sutrikimai: priklausomumas nuo temperatūros, drėgmė, vibracija, įtampa, radiacija ir kt.
- Loginė srities: klaidos (struktūrinės klaidos).
    - Sujungimų klaidos: ir, arba, neigimas.
    - Vėlinimo klaidos: įėjimų bei sąsajų.

Seniausios testavimo formos siejasi su funkciniu testavimu, kai įrenginys testuojamas normaliame darbo režime bei jo projektuojamame darbo dažnyje. Pagrindinis šio testavimo uždavinys yra užtikrinti, jog schema funkcionuoja pagal specifikaciją (be klaidų). Funkciniame testavime gali būti naudojami tie patys testų rinkiniai, kuriuos naudojo projektuotojas verifikavimui projektavimo fazėje. Funkcinis testavimas gali padengti daug procentų klaidų schemose, bet šios metodikos

trūkumas yra tas, kad norint pasiekti aukštą testavimo kokybę reikia labai didelio kiekio testų sekų. Testuojant sudėtingas schemas šis būdas praktikoje nėra taikomas.

2.4. TESTINIO MODELIO GENERAVIMAS

Testinio modelio generavimas – tai procesas kai nustatoma testiniai vektoriai, kurie reikalingi elektroninės sistemos testavimui. Paprasčiausias būdas tam atlikti yra nuodugnus schemų testavimas, kur visi galimi įėjimo signalai bus patikrinti. Tai reiškia, jog bus patikrinta 2^n testinių vektorių (n įėjimų kiekis). Toks didelis testinių vektorių kiekis reiškia, jog nuodugnus testavimas galimas tik su mažomis schemomis. Pavyzdžiui, schemai su 100 įėjimų reikės $2^{100} \approx 10^{30}$ testinių vektorių ir tai praktiškai yra neįvykdoma. Alternatyvus būdas nuodugniam testavimui yra atsitiktinis testavimas, kai testiniai vektoriai yra generuojami atsitiktinai. Nors šio testavimo būdo išlaidos yra žymiai sumažintos, tačiau atsitiktinis testavimas negali aptikti visų klaidų. Atsparios atsitiktiniam testavimui klaidos verčia toliau ieškoti geresnių testavimo būdų [1]. Keletas struktūrinių testų generavimo metodų yra išvystyta tam, kad išspręstų šias problemas. Šiuo atveju yra laikoma, jog elementarūs komponentai yra be klaidų ir tik jų sąsajos yra paveiktos [5]. Tačiau visi šie būdai yra atliekami žemame abstrakcijos lygyje.

2.5. AUKŠTO LYGIO KLAIDŲ MODELIS

Pagrindinė užduotis šio modelio yra sudaryti įėjimų sekas, kurios gali sąlygoti klaidingas sistemos reakcijas. Kaip jau minėta, gali būti daug priežasčių dėl ko atsiranda tos klaidos: projektavimo klaidos, aplinkos, nusidėvėjimo, klaidos specifikacijoje ir kitos. Testų generavimo tikslui reikia klaidingą elgseną apibrėžti tam tikru matematiniu formalizavimu, kuris yra vadinamas klaidų modeliu. Vienas klaidų modelis neturi padengti visų galimų defektų, dažniausiai jie padengia tik tam tikrą galimų defektų rinkinį, kuris turi didelę galimybę pasitaikyti.

Viena iš pagrindinių problemų aukšto lygio klaidų modelio yra ta, kad testai yra sudaromi aukštame abstrakcijos lygyje, kai yra žinoma labai mažai arba visai nežinoma apie galutinę realizuotą sistemą ir negalima nustatyti ryšio tarp gamybos defektų bei klaidų modelio [1].

Paprasčiausių aukšto lygio klaidų modelių gerosios savybės yra tos, kad dirbama aukštame abstrakcijos lygyje paimant nuoseklumą iš programinės įrangos testavimo sferos:

- Kelių padengime yra skaičiuojamas procentas, kiek srautų kelių buvo paveikta (sužadinta) su atitinkamu testų modeliu. Šakų padengime yra skaičiuojamas procentas, kiek modelio šakų buvo paveikta su atitinkamu testų modeliu [6].
- Bitų padengimas. Kiekvienas kintamasis, signalas, jungtis modelyje gali būti pakeistas 0 arba 1. Skaičiuojamas procentas tokių bitų, kurie atsiranda modelio išėjimuose su atitinkama testų seka [7].

- Būsenų padengimas. Susijęs su klaidomis sudėtingų sistemų valdymo įrenginio loginiame realizavime. Skaičiuojama kiek būsenų gali būti pakeista teisinga (angl. true) ar neteisinga (angl. false) reikšme. Tada yra skaičiuojamas procentas pakeistų būsenų, kurios atsiranda modelio išėjimuose su atitinkama testų seka [7].
- Mutacijos testavimas koncentruojasi atrenkant testinius vektorius, kurie gali atskirti programą nuo rinkinio klaidingų versijų arba pakaitalų. Pakaitalas yra generuojamas įterpiant paprastą trūkumą programoje [12].

  Pavyzdžiui: x := (a + b) – c; tam, kad valdyti trūkumą, „+“ yra pakeičiamas į „-“, b neturi būti „0“ (nes a + 0 = a - 0 ir šis trūkumas negali būti aptiktas). Be to tam, kad valdyti ydas vietoje „+“ yra „x“, mes turime įsitikinti, jog a + b ≠ a x b.

Visi šie klaidų modeliai yra grįsti schemos elgsena, o ne jos struktūrą. Tam, kad apimti klaidas galutiniame realizuotame produkte yra labai svarbu nustatyti ryšį tarp aukšto lygio klaidų modelių ir žemesnių. Tai yra atlikta tik eksperimentiškai ir šiuo metu nėra tinkamų sisteminių metodų. Bent iš dalies išvengti šių problemų yra siūloma naudoti hierarchinį klaidų modelį bei hierarchinį testų generavimą.

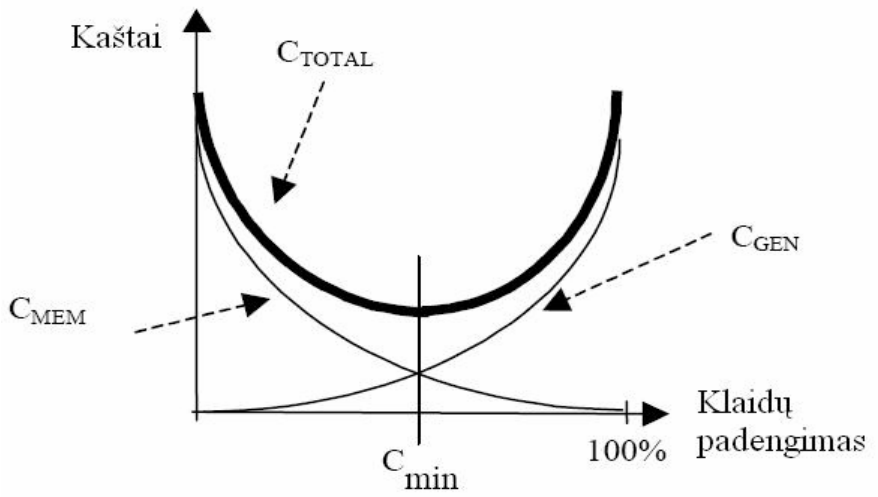
2.6. HIERARCHINIS TESTŲ GENERAVIMAS

Pagrindinė idėja hierarchinio testų generavimo (HTG) metodikos yra naudoti informaciją iš skirtingų abstrakcijos lygių generuojant testus. Vienas iš pagrindinių principų yra naudoti modulinį projektavimo stilių, kuris leidžia klasifikuoti didesnį uždavinį į keletą mažesnių ir juos atskirai išspręsti. Šis būdas leidžia generuoti testinius vektorius žemesnio lygio moduliams, kurie remiasi skirtingomis metodikomis pritaikytomis atitinkamoms esybėms.

Hierarchiniame testavime yra žinomos dvi strategijos: iš viršaus į apačią (arba smulkinanti) ir iš apačios į viršų (arba stambinanti). Iš apačios į viršų metodikoje sugeneruoti testai žemesniuose lygiuose bus apjungiami aukštesniame abstrakcijos lygyje [8]. Iš viršaus į apačią strategijoje naudojama informacija gauta aukštesniame lygyje tam, kad išvesti testus žemesniuose lygiuose [9].

2.7. MIŠRUS TESTAVIMAS

Mišrus testavimas – kai testuoti schemai yra naudojamos kelios testavimo metodikos. Pavyzdžiui: naudojamas atsitiktinis testų generavimas ir paruošti testiniai vektoriai (sugeneruoti klaidų modelio), kurie saugomi atmintyje. Šios metodikos trūkumas yra tas, jog jos kaštai yra visų metodikų kaštų suma, tačiau naudojami metodai vienas kitą papildo teigiamomis savybėmis:

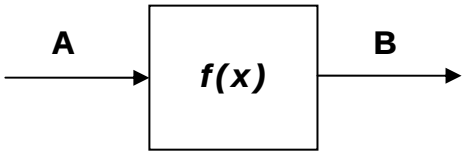


1 pav. Mišraus testavimo kaštai

C_{GEN} yra kaštai reikalingi testinių vektorių atsitiktiniam generavimui (vertinamas laikas skirtas generavimui). Šio testavimo kaštai eksponentiškai didėja norint užtikrinti didesnį sistemos pasikliautinumo lygį. C_{MEM} yra atminties kaštai, kurių reikia saugoti papildomus testinius vektorius, gautus tam tikrų skaičiavimų metu. Norint pasiekti aukštesnį pasikliautinumo lygį šios metodikos kaštai mažėja, nes papildomai prisideda vis mažiau testinių vektorių sekų, kurios geriausiai testuoja schemą. Mišraus testavimo kaštai C_{TOTAL} yra abiejų būdų suma ($C_{TOTAL} = C_{GEN} + C_{MEM}$). Tam, kad optimizuoti mišraus testavimo kaštus yra ieškomas C_{MIN} taškas, kuriame $C_{GEN} = C_{MEM}$ [10].

2.8. ALGORITMAS

Modelio funkcija turi n dvejetainių įėjimų ir m dvejetainių išėjimų. Įėjimo poveikis apibrėžiamas vektoriumi $A=<a_1,a_2, \ldots,a_i, \ldots,a_n>$, kur $a_i \in \{0,1\}$. Funkcijos reakcija apibrėžiama vektoriumi $B=<b_1,b_2, \ldots,b_j, \ldots,b_m>$, $b_j \in \{0,1\}$. Bendru atveju B=f(A).



2 pav. Modelio funkcija

Nagrinėjamos funkcijos: c432, c3540, c2670, c1908, c1355, c17, c499, c5315, c6288, c7552, c880. Sprendimo paieškos apimtis apibrėžiama nagrinėjamų įėjimo poveikių kiekiu *kiekis*. Sprendinį apibrėžia atrinktų įėjimo poveikių aibė V. Aibių V ir V1 sujungimą žymėsime V!!V1. Sprendinio gerinimo procentą žymėsime PK%.

Algoritmo pradžioje esminiai poveikiai yra pasirenkami ir įvedami iš atminties į aibę V. Įvedimas dažniausiai vykdomas norint tęsti analizę, kuri buvo atlikta anksčiau ir jos metu buvo suformuotas duomenų failas. Jei poveikiai buvo įvesti, tuomet atliekama aibės V analizė ir nustatoma

jos kokybė k. Jei poveikiai nebuvo įvesti, tuomet aibė V yra tuščia ir k = 0. Toliau poveikiai yra generuojami atsitiktinai ir analizės metu atrenkami į aibę V1. Sugeneravus atitinkamą kiekį poveikių prie aibės V1 yra prijungiami aibės V poveikiai. Analizė atliekama iš karto apjungiant, tuomet iš aibės V nebus pridėti poveikiai, kurie suformavus aibę V1 tapo neesminiai. Taip gaunama nauja aibė V. Baigus apjungimą yra nustatoma analizės kokybė k1. Tada apskaičiuojamas pagerėjimo koeficientas. Jei jis yra didesnis už nurodytą PK%, tuomet grįžtama į analizės pradžią ir generuojami nauji poveikiai ir formuojama nauja aibė V1, prie jos vėl prijungiama aibė V ir toliau tęsiama analizė. Jei gautas koeficientas buvo mažesnis už nurodytą PK%, tada atsitiktinai generuojamų poveikių kiekis dvigubinamas ir formuojama nauja aibė V1. Jei po generuojamų poveikių kiekio padidinimo pakartojus analizę gautas pagerėjimo koeficientas vėl nėra didesnis už nurodytą PK%, tuomet analizė baigiama.

Pagrindinį algoritmą galima suskirstyti į etapus:

1. Nuskaityti įėjimo poveikių aibę V; pabaiga := 0, DV := 0;
2. Analizuoti aibę V išmetant perteklinius poveikius ir nustatant poveikių kokybę k;
3. Naujų įėjimo poveikių V1 paieška apimtyje kiekis;
4. V1 aibės išplėtimas. Apjungimas V := V1!!V;
5. Analizuoti aibę V1!!V išmetant perteklinius poveikius ir nustatant poveikių kokybę k1;
6. P := ((k1 - k) / k1) * 100; k := k1;
7. Jeigu P > PK%, tada DV := 0, grįžti į 3 etapą;
8. Jeigu DV = 0, tada kiekis := 2 * kiekis, DV := 0, grįžti į 3 etapą;
9. Pabaiga.

Pagal pateiktus veiksmus sudaroma algoritmo struktūrograma, kuri pateikta 1 lentelėje.

1 lentelė. Pagrindinio algoritmo struktūrograma

<table>
<tr><td colspan="4">A: Nuskaitomi įėjimo poveikių aibė V, modelio parametrai, kiekis, PK% ir kiti. Sukompiliuojame modelį. Pabaiga := 0; DV := 0;</td></tr>
<tr><td colspan="4">B: Aibės V analizė išmetant perteklinius poveikius ir nustatant poveikių kokybę k.</td></tr>
<tr><td colspan="4">Kartoti kol Pabaiga = 0</td></tr>
<tr><td colspan="4">C: Įėjimo poveikių V1 atsitiktinė paieška apimtyje kiekis.
D: V1 aibės išplėtimas. Apjungimas V := V1!!V;</td></tr>
<tr><td colspan="4">B: Aibės V1!!V analizė išmetant perteklinius poveikius ir nustatant poveikių kokybę k1.
P := ((k1 - k) / k1) * 100; k := k1;</td></tr>
<tr><td colspan="4">P > PK%
N T</td></tr>
<tr><td colspan="2">DV = 0
N T</td><td colspan="2" rowspan="2">DV := 0;</td></tr>
<tr><td>Pabaiga := 1;</td><td>kiekis := 2 * kiekis;
DV := 1;</td></tr>
</table>

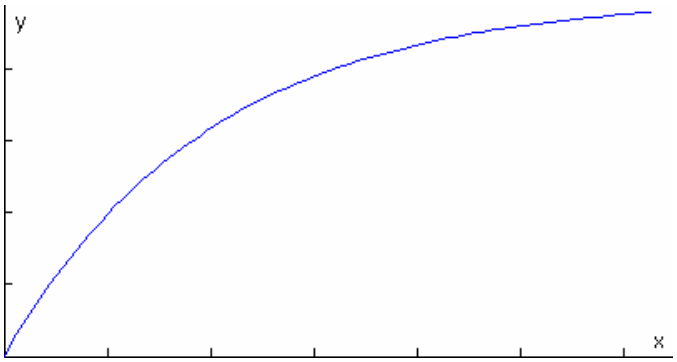
Užduotis reikalauja rasti kuo daugiau esminių termų, todėl sprendinio kokybę rodys sudarytų termų kiekis.

Dydis *kokybė* iteracijų metu gali didėti ir mažėti. Didėjimas ar mažėjimas nenusako analizės rezultatų gerėjimo ar blogėjimo. Nes termų kiekio mažėjimas gali reikšti, jog buvo rastas naujas ilgesnis termas, kuris išmetė kelis kitus. Taip pat termų kiekio nekitimas ir negali reikšti, jog kokybė nepakito, nes analizės metu naujas poveikis suformavo geresnį termą nei buvo jau rastas ankstesnėje iteracijoje. Kiekvienos iteracijos metu yra randamas vis didesnis kiekis termų arba keli neesminiai termai pakeičiami vienu geresniu. Termų atžvilgiu analizė visada artės prie galutinio rezultato.

Analizės metu kiekvienam įėjimo poveikiui paskaičiuojama matrica $X= ||x_{i,j}||_{n \times m}$. Matricos elementas $x_{i,j}=1$, jei pakeitus poveikio įėjimo a_i reikšmę į priešingą, išėjimo b_j reikšmė taip pat pasikeičia į priešingą. Stulpelių nenuliniai elementai apsprendžia termą. Įėjimo *i* tiesioginis (atvirkštinis) kintamasis priklauso išėjimo *j* tiesioginės (atvirkštinės) funkcijos termui , jei $x_{i,j}=1$, $a_i=1(a_i=0)$, o $b_j=1(b_j=0)$. Funkcijos termas nėra esminis, jei kitas termas turi jo visus kintamuosius. Nagrinėjant įėjimo poveikius atrenkami tik tokie poveikiai, kurie turi bent vieną esminį termą. Atrinkti poveikiai surašomi į aibę V.

Apjungiant išėjimo tiesioginį ir atvirkštinį termus gaunami du įėjimo poveikiai, besiskiriantys prieštaraujančiais kintamaisiais. Jei tiesioginis ir atvirkštinis termas neturi nei vieno prieštaraujančio kintamojo, tuomet apjungimo metu gaunamas vienas poveikis, į kurį įeina abiejų termų elementai. Kintamieji, kurių nėra abiejuose termuose, parenkami atsitiktinai.

Dalis suformuotų termų nėra visiškai teisingi, todėl algoritmas ieško tik galimus schemos termus. Jie galutinai patikslinti gali būti suradus visus termus ir apjungus tiesioginės bei atvirkštinės funkcijų termus. O tai praktiškai nėra įmanoma, nes vieno išėjimo termų kiekis yra ribojamas iki n^2 ir pilnai ištirti modelį reikia išanalizuoti 2^n poveikių. Kai paprasčiausio funkcinio modelio įėjimų kiekis yra 36, tai ištirti 2^{36} poveikių užims nemažai laiko ir termų patikslinimo funkcija praktiškai bus nenaudojama. Nors ir algoritmo rezultatai tik artėja prie baigtinio sprendinio, tai nėra problema sudarant testinius poveikius. Eksperimentiniu būdu nustatyta, kad paprastai analizės sprendinys konverguoja eksponentiškai. Tai pavaizduota 3 paveiksle (x – esminių poveikių kiekis, y – esminių termų kiekis).



3 pav. Sprendinio konvergavimas

Šio algoritmo įnašas yra dvejopas. Pirmoji dalis tiria galimybes generuoti testinius vektorius ankstesniuose projektavimo ciklo lygiuose, vadovaujantis tik elgsenos aprašymu ir su ribotai mažomis žiniomis apie galutinai realizuotą architektūrą. Antroji dalis sukoncentruota testavimo projektavimui. Testuoti šiuolaikiškas sudėtingas elektronines sistemas yra labai brangi procedūra, specialios struktūros skirtos supaprastinti šį procesą gali būti įterptos į sistemą projektavimo fazėje.

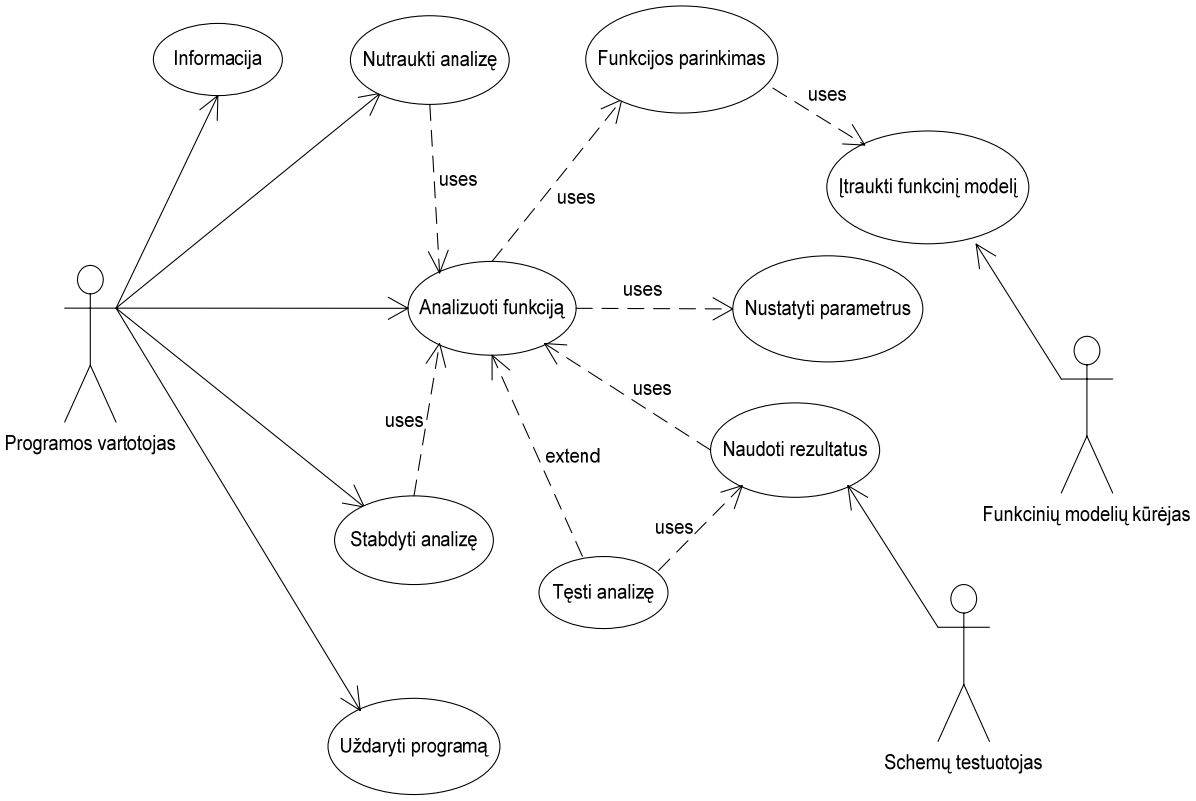
Algoritmas abstrakčiame lygyje atrenka poveikius (įėjimo signalus). Tikimasi, kad taip atrinkti poveikiai geriau tikrins realizuotas funkcijas. Gauti rezultatai galės būti panaudoti tolimesniems tyrimams: schemų testavimui, defektų šalinimui, funkcijos elementų palyginimui.

3. PROJEKTINĖ DALIS

3.1. SISTEMOS APRAŠYMAS

Programa pagal vartotojo nurodytus parametrus pradeda vykdyti esminių poveikių bei esminių termų atrinkimo algoritmą remdamasi tik funkcinio modelio elgsena (įėjimo bei išėjimo signalais). Programos rezultatai yra esminiai poveikiai ir iš jų sudaryti esminiai termai. Vartotojui yra suteikiama galimybė stebėti analizės eigą. Pateikiama skaičiavimų statistika kiekvienai iteracijai: sugeneruotų poveikių skaičius iteracijos metu, viso išanalizuota poveikių, viso atrinktų esminių poveikių bei termų kiekiai, kokybės pokytis. Šių rezultatų pagalba galima tirti algoritmo veikimą bei imtis veiksmų jo gerinimui.

3.2. PANAUDOS ATVEJAI



4 pav. PA diagrama

Sistemos aktoriai:

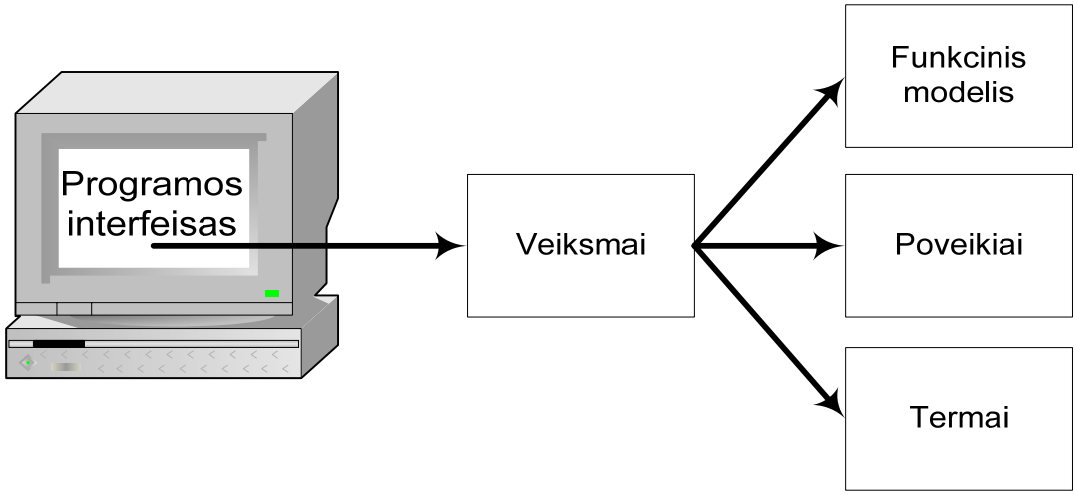
- Programos vartotojas: asmuo, kuris naudosis programa. Gali vykdyti iš naujo arba tęsti funkcinio modelio analizę, formuoti testinius vektorius (rezultatus). Nebūtina žinoti funkcinio modelio veikimo principo.

- Funkcinių modelių kūrėjas (programuotojas): asmuo, kuris sudarinėja matematinius modelius. Jis gali papildyti sistemą nauju funkciniu modeliu. Jis turi išmanyti funkcinių modelių sudarymą C++ kalboje.
- Testuotojas: asmuo, kuris naudoja programos rezultatus (termus, poveikius) schemų testavimui, analizei, tyrimams. Reikalingos žinios testavimo srityje, funkcinių modelių veikimo principo žinojimas. Testuotoju gali būti ir paprastas vartotojas (jis gautus rezultatus naudoja kitos analizės metu arba kitose panašaus pobūdžio sistemose).

Panaudos atvejai:

- **Informacija.** Vartotojui išvedama informacija apie programą ir jos autorių.
- **Analizuoti funkciją.** Pradedama analizuoti pasirinktą funkcinį modelį; vykdoma pagal vartotojo nurodytus parametrus. Turi būti paleista programa, nustatyti parametrai, parinktas funkcinis modelis (jei nebus nustatyti parametrai ir parinktas funkcinis modelis analizė bus atliekama pagal tai duomenis, kurie buvo atsitiktinai nustatyti paleidus programą arba likę iš prieš tai atliktos analizės).
- **Nutraukti analizę.** Nutraukiama pradėta analizė prarandant rezultatus. Reikalinga tam, kad įvedus klaidingus duomenis ar dėl kitų priežasčių nereikėtų laukti skaičiavimų pabaigos.
- **Stabdyti analizę.** Stabdoma vykdoma analizė. Programa baigia vykdomos iteracijos skaičiavimus ir jei reikia vykdyti sekančia iteraciją, jos nevykdo. Jei vartotojas buvo nurodęs saugoti rezultatu, jie yra išsaugomi. Šio PA pagalba kai kurie parametrai gali būti pakeisti analizės eigoje arba analizė atlikta iki galo per kelis kartus.
- **Tęsti analizę.** Tęsiama pasirinkto funkcinio modelio analizė, kuri vykdoma pagal vartotojo nurodytus parametrus bei nuskaitant poveikius iš pasirinkto failo.
- **Nustatyti parametrus.** Nustatymas pagrindinių kintamųjų (pagerėjimo koeficiento, išplėtimo koeficiento, pradinio generuojamų poveikių kiekio), jų neįvedus programa pradės analizę su reikšmėmis, kurios buvo nustatytos programos užkrovimo metu.
- **Funkcijos parinkimas.** Analizuojamo funkcinio modelio parinkimas, būtinas analizei vykdyti.
- **Įtraukti funkcinį modelį.** Sistemos papildymas nauju funkciniu modeliu.
- **Naudoti rezultatus.** Suformuotų sistemos rezultatų panaudojimas schemų testavime, tyrimams, tolimesnėse analizėse ir kt.
- **Uždaryti programą.** Programos lango uždarymas, baigiamas darbas su programa.

3.3. SISTEMOS KOMPONENTŲ VAIZDAS



5 pav. Sistemos komponentų vaizdas

Kompiuteryje esama programa valdoma per vartotojo sąsają („“Vartotojo interfeisas“). Jos pagalba vartotojas gali pasirinkti norimas operacijas:

- Parinkti funkcinį modelį;
- Nustatyti analizės parametrus;
- Vykdyti naują (arba tęsti) analizę;
- Peržiūrėti informaciją apie programą, jos autorių.

Programinis modulis „Veiksmai“ yra sužadinamas per vartotojo sąsaja. Jis naudoja programinius modulius „Funkcinis modelis“, „Poveikiai“ ir „Termai“ bei atlieka analizės veiksmus:

- Generuoja poveikius;
- Sudaro termus;
- Atrenka esminius poveikius bei termus.

Programinis modulis „Funkcinis modelis“ yra skirta funkcinio modelio saugojimui. Pagal vartotojo pasirinkimą jis yra užkraunamas iš funkcinių modelių katalogo, kuriame yra saugomi visi funkciniai modeliai.

„Poveikiai“ – šis modulis atlieka pagrindinius veiksmus susijusius su poveikių aibe.

Programinis modulis „Termai “ atlieka pagrindinius veiksmus susijusius su termų aibe bei jos kintamaisiais.

3.4. DUOMENŲ VAIZDAS

Analizės metu pagrindiniai duomenys yra poveikių bei termų aibės. Jiems saugoti sukuriamos klasės. Poveikiam saugoti sukuriama duomenų struktūra:

```
struct poveikiai{
        int *poveikis;          //n įėjimų aibė
        int kiekis;             //kiek esminių termų sugeneruoja poveikis
        poveikiai *next;        //rodyklė į sekantį poveikį
};
```

3.4.1. Poveikiai

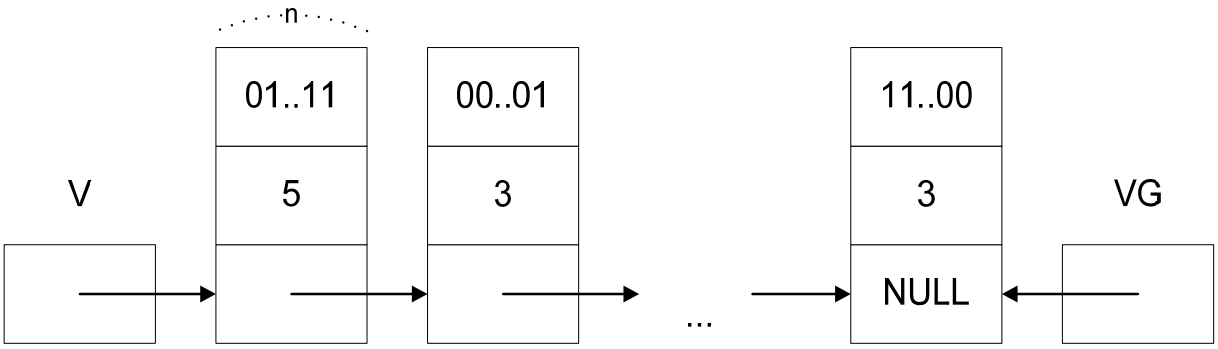
Ši klasė kaupia duomenis, susijusius su esminiais poveikiais. Poveikių aibei saugoti naudojami šie pagrindiniai *poveikiai* tipo kintamieji:

V – pradžia dinaminio sąrašo, kuriame saugomi poveikiai.

VG – sąrašo V pabaiga. Patogu naudoti įrašant naują poveikį.

EP – analizės pabaigoje iš aibės V atrenkami esminiai poveikiai, kurie sugeneruoja bent vieną esminį termą.

EPG – sąrašo EP pabaiga. Patogu naudoti įrašant naują poveikį.

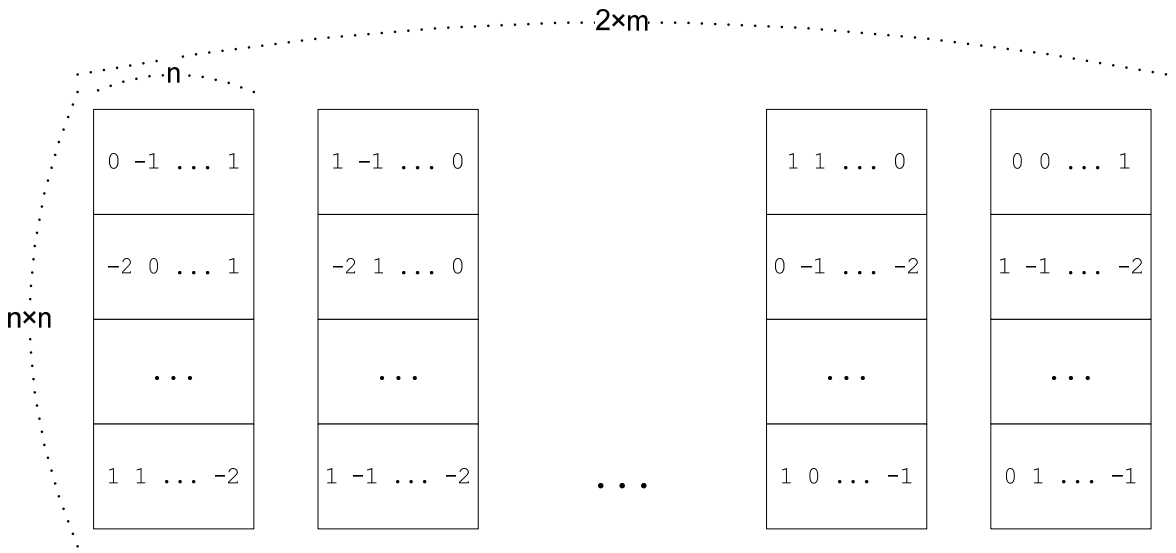


6 pav. Poveikių aibė

3.4.2. Termai

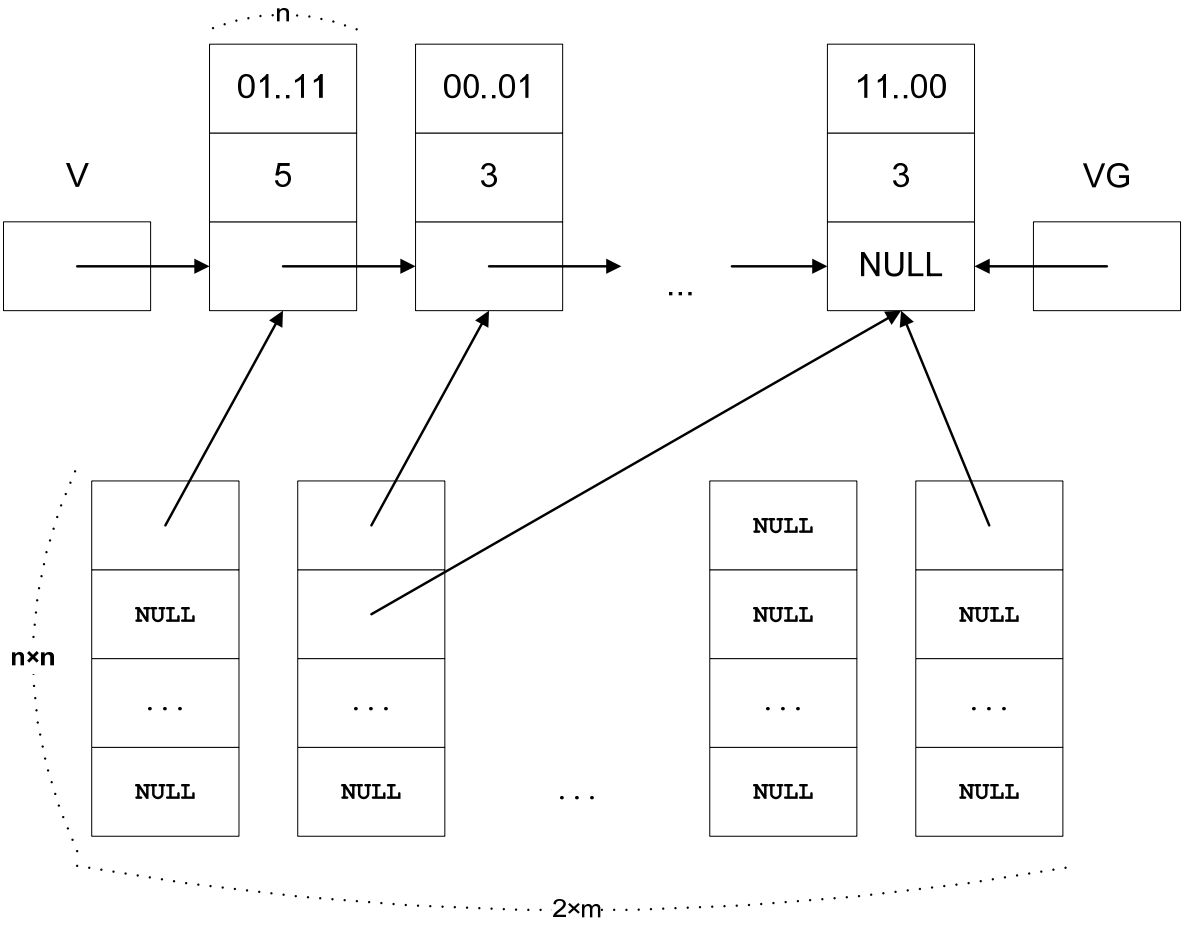
Ši klasė kaupia duomenis, susijusius su esminiais termais. Termų aibei saugoti naudojamos šios pagrindinės duomenų struktūros:

int termai[2m][n^2][n] – kiekvienam išėjimui (jų kiekis yra m) formuojami termai tiesioginei ir atvirkštinei funkcijoms, todėl aibė termai bus sudaryta iš 2m stulpelių. Kiekvieno stulpelio maksimalus elementų kiekis yra n^2. Ir kiekvienas elementas gali būti sudarytas iš n įėjimų. Neaktyvūs įėjimai žymimi neigiamomis reikšmėmis. Prie termo neigiamų reikšmių pridėjus 2, gaunamas poveikis, iš kurio buvo sugeneruotas termas. Patogu pateikiant rezultatus (iš termo matyti esminis poveikis).



7 pav. Termų aibės

poveikiai P[2m][n^2] – ši duomenų struktūra sieja aibę termai su poveikių aibe V (poveikiai[i][j] elementas nukreiptas į tą aibės V elementą, kuris suformavo termai[i][j]). Patogu naudoti, kai yra ištrinamas neesminis termas ir reikia sumažinti poveikio sugeneruotų termų kiekį vienetu. Nereikia vykdyti paieškos visoje poveikių aibėje. Ši struktūra yra naudojama pagreitinti skaičiavimus.



8 pav. Termų ir poveikių sąsaja

3.4.3. Funkcinis modelis

Funkciniai modeliai yra sudaromi C kalboje. Jo pagrindas yra metodas, kuriam padavus įėjimų signalus grąžinama reakcija (išėjimų signalai). Funkcinio modelio aprašas:

void funkcija(char *gg, char *og, int in, int on, int *gerai)

gg – įėjimų aibė;

og – išėjimų aibė;

in – įėjimų kiekis;

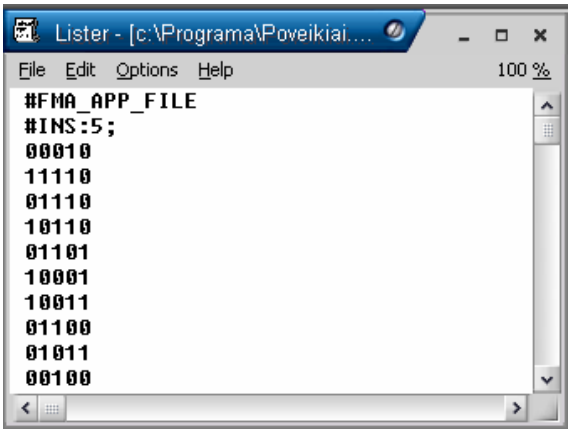
on – išėjimų kiekis;

gerai – atlikimo požymis.

Funkcinio modelio struktūra pateikta priede 9.1 skyriuje.

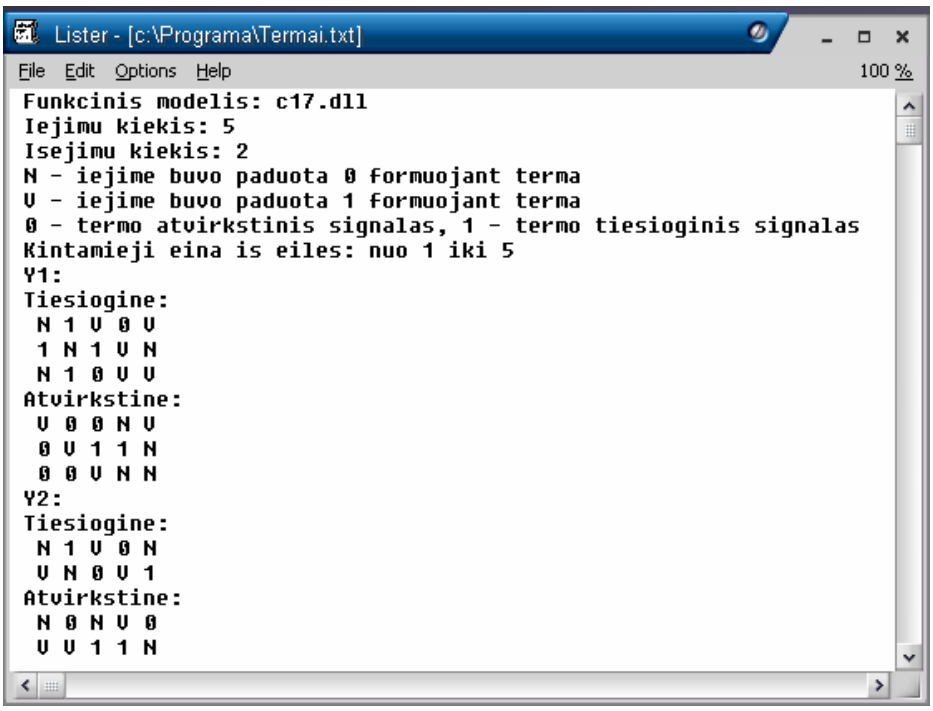
3.4.4. Duomenų/rezultatų failai

Poveikių duomenų/rezultatų failo formatas pagrinde susideda iš trijų eilučių. Pirmoji eilutė: #FMA_APP_FILE (angl. „function model analysis application file“). Ji identifikuoja, jog skaitomas failas yra pritaikytas šiai programai. Antroji eilutė: #INS:36. Ji nurodo įėjimų kiekį (šiuo atveju 36 įėjimai). Trečioji eilutė (ir visos kitos) susideda iš n skaitmenų sekos (0 arba 1), kur n yra įėjimų kiekis.



9 pav. Poveikių rezultatų/duomenų failas

Termų rezultatų faile yra pateikiama funkcinio modelio aprašas (pavadinimas, įėjimų bei išėjimų kiekiai) ir kiekvienam išėjimui tiesioginės bei atvirkštinės funkcijų termai vartotojui suprantamoje formoje.



10 pav. Termų rezultatų failas

Skaičiai žymi aktyvius termų įėjimus. 0 žymi atvirkštinį įėjimo signalą, 1- tiesioginį. „N“ ir „V“ žymi, koks signalas (nulis ar vienas) buvo paduotas neaktyviame įėjime formuojant termą. „N 1 V $X_2\overline{X_4}$ 0 V“ – termas suformuotas iš poveikio 01101.

4. TYRIMO DALIS

Dauguma algoritmų termų generavimui remiasi funkcinių modelių struktūra. Realizuotas algoritmas vykdo skaičiavimus remdamasis tik įėjimo signalais ir reakcija į juos. Termus generuoja pagal poveikius, kurių įėjimai turi daugiausiai įtakos išėjimams. Tikimasi, kad atrinkti poveikiai geriausiai testuoja funkcinius modelius. Funkcinio modelio termų generavimo procesą vadinsime analize. Atlikus keletą eksperimentų nustatyta, jog analizės eiga ir jos rezultatai priklauso nuo parinktų pradinių parametrų: funkcinio modelio, pradinio generuojamų poveikių kiekio, pagerėjimo koeficiento ir atsitiktinai generuojamų įėjimų signalų eilės tvarkos. Analizei bei jos rezultatams taip pat įtakos turi nustatytas maksimalus vieno išėjimo tiesioginės ir atvirkštinės funkcijų termų kiekis.

4.1. FUNKCINIS MODELIS

Funkciniai modeliai realizuoja įvairias logines funkcijas, kurios skiriasi įėjimų skaičiumi ir/arba veikimo principu (vienos gali realizuoti daugybę sudėtingų loginių operacijų, kitos kelias paprastas). Funkcinio modelio įėjimų kiekis turi įtakos kiekvienos funkcijos termų ilgiui ir skaičiui. Funkcijai, turinčiai n įėjimų, gali būti sugeneruoti termai, kurių ilgis nuo 1 iki n kintamųjų ir vienodo ilgio termai gali skirtis vienu ar keliais kintamaisiais. Analizės vykdymo laiką įtakoja funkcinio modelio loginių funkcijų skaičius (išėjimai) ir jų sudėtingumas. Priede 9.2 skyriuje pateikti eksperimentai su keliais funkciniais modeliais (testinio, c17, c432). Didėjant funkcinio modelio sudėtingumui bei įėjimų ir išėjimų kiekiams didėja ir sugeneruotų termų skaičius bei analizės trukmės laikas. Testinei funkcijai (1 įėjimas, 3 išėjimai) sugeneruoti 5 termai, funkciniam modeliui c17 (5 įėjimai, 2 išėjimai) sugeneruota 12 termų, o funkciniam modeliui c432 (36 įėjimai, 7 išėjimai) sugeneruota apie 15 tūkstančių termų. Vykdant termų generavimą testinei funkcijai ir funkciniam modeliui c17 analizės netruko net 1 sekundės. O žymiai sudėtingesnio funkcinio modelio (c432) termų generavimas užsitęsdavo iki 1 valandos (9.2.3 skyriaus 7-10 lentelės).

4.2. GENERUOJAMŲ POVEIKIŲ EILĖS TVARKA

Atliktų analizių rezultatai skiriasi. Skirtingi sprendiniai gaunami dėl atsitiktinai generuojamų poveikių eilės tvarkos, kuri kiekvienos analizės metu yra skirtinga. Akivaizdu, jog kokia eilės tvarka bus analizuojami esminiai poveikiai taip bus ir rašomi į aibę esminiai termai. Dviejų analizių metu gavus vienodus termų rinkinius esminių poveikių kiekiai gali skirtis. Šį rezultatų skirtumą lemia atsitiktinai generuojamų poveikių eilės tvarka. Pavyzdžiui, vienas poveikis sugeneruoja du esminius termus, sekantis sugeneruoja termą, kuris yra toks pat kaip pirmojo poveikio, šiuo atveju termų aibėje toks jau egzistuoja ir antrasis poveikis nėra esminis. Jeigu sukeistume šių dviejų poveikių eilės tvarką tuomet, jie abu būtų atrinkti kaip esminiai. Vieną termą sugeneravęs poveikis būtų įrašytas, du termus

sugeneravęs poveikis papildytų aibę jau ne dviem, o tik vienu termu, nes kitas jau buvo įrašytas anksčiau su prieš tai analizuotu poveikiu. Galimas atvejis, kai vienas poveikis gali generuoti visus termus, kurie gaunami ir su keliais kitais poveikiais. Todėl priklausomai nuo generavimo eilės priklausys, ar bus vienas esminis poveikis, ar keli. Taip vyko ir atliekant eksperimentą su funkciniu modeliu c17 (pateiktas priede 9.2.2 skyriuje). Analizuojant šį funkcinį modelį buvo gauti visi termai su skirtingais esminių poveikių kiekiais. Pateiktas pavyzdys rodo, jog kuo anksčiau sugeneruoti poveikiai turi didesnę tikimybę būti traktuojami kaip esminiai. Todėl didelę įtaką turi pirmųjų iteracijų metu sugeneruoti įėjimų signalai. Kad šią priklausomybę perkelti paskutiniųjų iteracijų generavimui, algoritmas turimų rezultatų prijungimą vykdo prie naujos iteracijos rezultatų. Tuomet pirmiau išanalizuojami yra naujai sugeneruoti poveikiai ir po to tik senesnėse iteracijose atrinkti poveikiai. Tokiu būdu yra vykdomas rezultatų „maišymas“.

4.3. MAKSIMALUS TERMŲ KIEKIS VIENAM IŠĖJIMUI

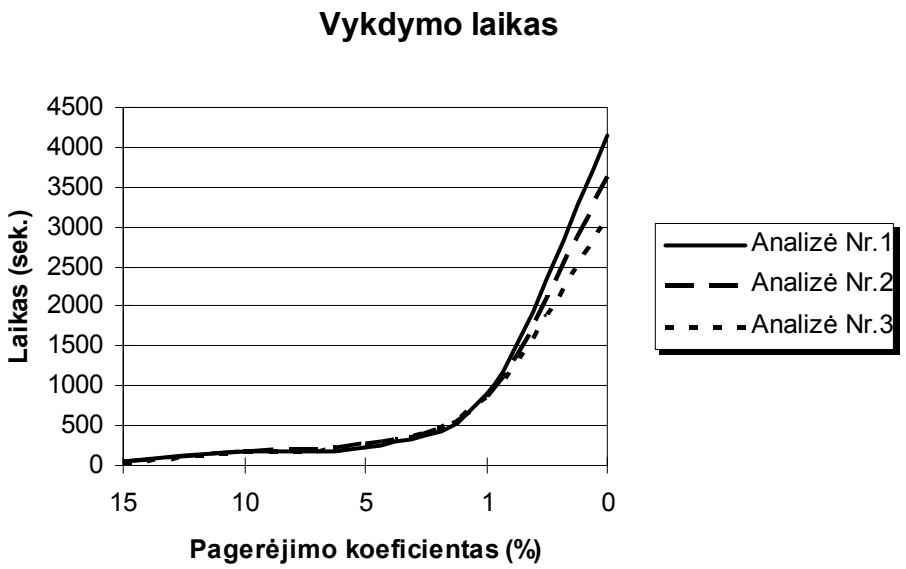
Rezultatų kiekybės kitimą sąlygoja ir termų ribojimas kiekvienam išėjimui (maksimalus termų kiekis vieno išėjimo tiesioginei ir atvirkštinei funkcijoms). Poveikis gali sugeneruoti termą, kuriam nėra vietos aibėje (pilna). Sekantis poveikis gali sugeneruoti termą, kuris yra geresnis už du jau esančius termus aibėje (seni termai yra daliniai lyginant su naujuoju), todėl yra įrašomas į vieno vietą ir kito termo vieta lieka laisva. Termų kiekis sumažėja, nors kokybė pagerėjo (gautas geresnis termas), tačiau ir vienas termas prarastas, nes nebuvo įrašytas į aibę. Čia įtakos turi ir poveikių generavimo eilė. Nes aprašytu atveju, jeigu pirmiau bus sudarytas geresnis termas, tuomet bus vietos ir neįrašytam termui. Todėl loginei funkcijai, turinčiai tokį pat termų kiekį, koks yra ir maksimalus leidžiamas vienam išėjimui, ne visada gali būti sudaryti visi termai. Tai priklausys nuo atsitiktinai generuojamų poveikių eilės tvarkos.

4.4. PAGERĖJIMO KOEFICIENTAS

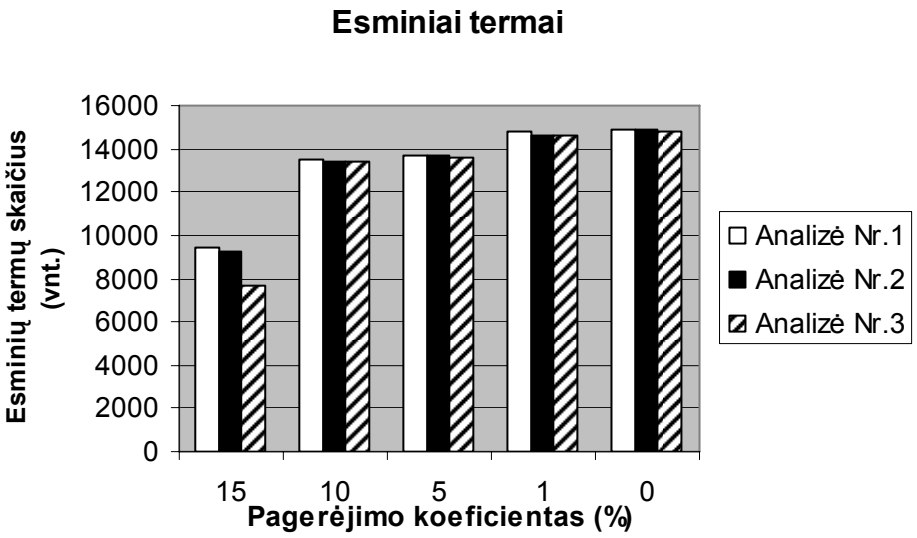
Funkcinio modelio loginių funkcijų termų generavimo eigą bei rezultatus įtakoja pagerėjimo koeficientas. Tai dydis, kuris rodo, kiek procentų turi padidėti termų kiekis po atliktos iteracijos. Jei termų kiekis padidėja mažiau nei nurodytas pagerėjimo koeficientas, tuomet generuojamų poveikių kiekis yra dvigubinamas. Jei antrą kartą iš eilės termų kiekis nepadidėja per nurodytą koeficientą, tai funkcinio modelio analizė yra stabdoma.

Suprantama, jog kuo didesnis pagerėjimo koeficientas, tuo greičiau yra baigiama analizė ir mažiau termų yra sudaroma. Didėjant rezultatų kiekiui, mažėja naujų termų. Norint gauti galutinį sprendinį pagerėjimo koeficientas turi būti lygus 0%. Tiriant algoritmo skaičiavimų eigos ir rezultatų priklausomybę nuo pagerėjimo koeficiento buvo atlikti eksperimentai. Norint įsitikinti eksperimento

tikslumu palyginimui buvo naudojami trijų analizių su vienodais parametrais rezultatai (9.2.3 skyriaus 7 -10 lentelės).



11 pav. C42 analizių vykdymo laikai



12 pav. C42 analizių esminiai termai

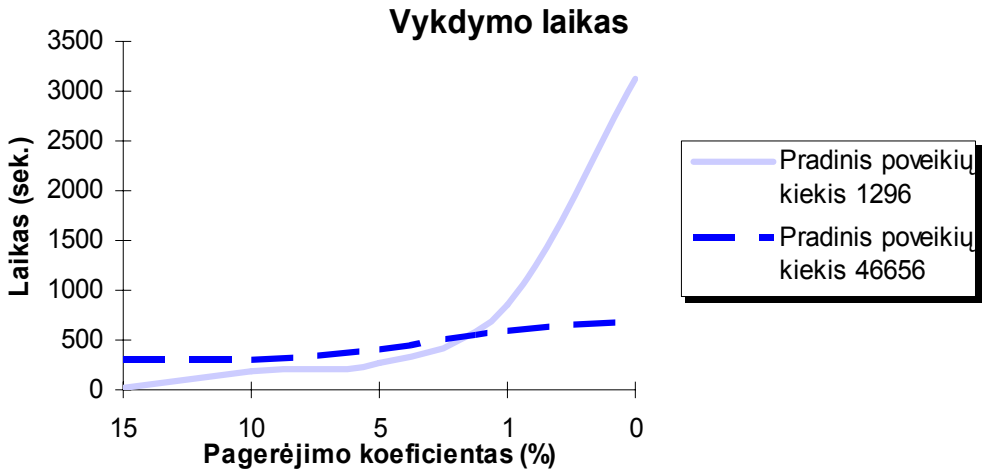
Iš grafikų (11 ir 12 pav.) matyti, jog norint gauti didesnį termų kiekį reikia nurodyti mažesnį pagerėjimo koeficientą. Mažinant pagerėjimo koeficientą eksponentiškai didėja ir skaičiavimų laikas. Vykdomų iteracijų kiekio priklausomybė nuo pagerėjimo koeficiento yra analogiška laiko priklausomybei (9.2.3 skyriaus 25 pav.). Skaičiavimų laikas žymiai padidėja ieškant galutinio sprendinio. Esminių poveikių kiekio didėjimas mažėjant pagerėjimo koeficientui vyksta panašiai kaip ir termų (9.2.3 skyriaus 27 pav.). Skirtumas tarp galutinio sprendinio (PK = 0%) ir artimo galutiniam (PK = 1%) yra labai mažas (apie 1%), nors pastaruoju atveju ir 1,5 karto mažiau poveikių buvo sugeneruota (9.2.3 skyriaus 26 pav.). Galutinis sprendinys reikalauja ne tik sugeneruoti žymiai daugiau

poveikių, bet ir laiko skaičiavimams. Atlikto eksperimento metu, kad gauti „paskutinį“ 1% termų, reikėjo išanalizuoti 1,5 karto daugiau poveikių ir tam skirti 4 kartus daugiau laiko.

Skaičiavimų pradžioje sparčiai didėja termų kiekis, o poveikių šiek tiek mažiau. Kai PK = 15% buvo rasta 60% esminių termų ir 40% esminių poveikių. Taip yra todėl, kad pradžioje yra didesnė tikimybė, jog poveikis sugeneruos daugiau esminių termų. Vėlesni poveikiai sugeneruoja kelis termus, tačiau jie jau būna įrašyti į aibę su kitais poveikiais arba yra daliniai ir nėra traktuojami kaip esminiai.

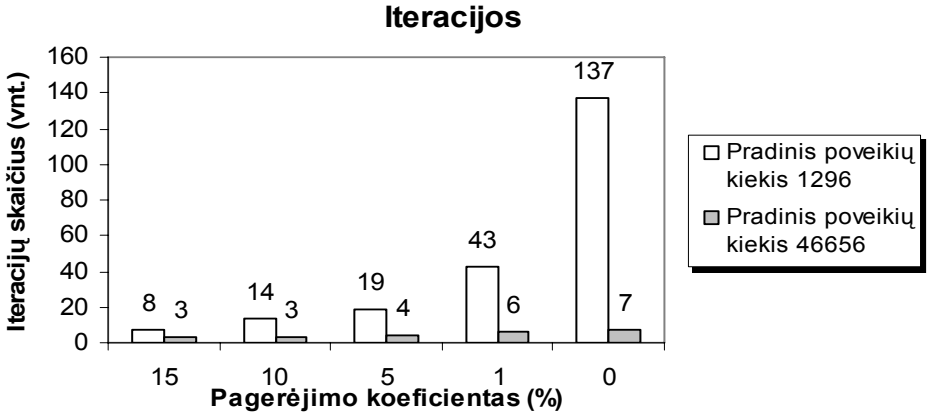
4.5. PRADINIS GENERUOJAMŲ POVEIKIŲ KIEKIS

Pradinis generuojamų poveikių kiekis taip pat labai įtakoja termų generavimo eigą. Tyrimas buvo atliktas su funkciniu modeliu c432. Analizės buvo vykdytos su pradiniu poveikių kiekiu n^2 (ištisinė linija) ir n^3 (brūkšninė linija), kur n – įėjimų kiekis (13 – 17 pav.).



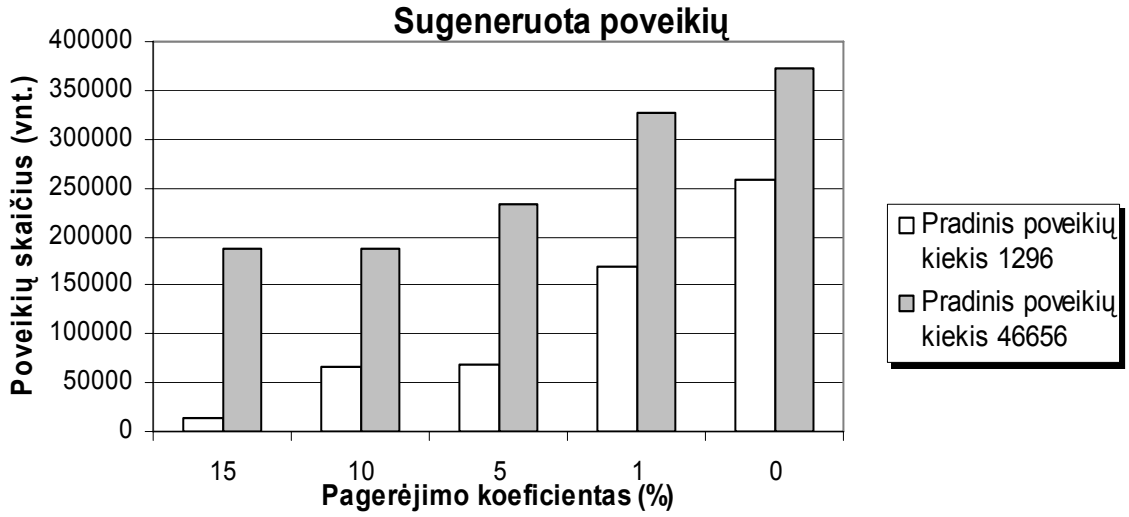
13 pav. C42 analizių vykdymo laikai

Iš grafiko matyti, jog funkciniam modeliui skaičiuojant galutinį sprendinį (kai paskutinių iteracijų metu kokybė nepagerėja daugiau kaip per 0%) yra labai svarbu parinkti pradinį generuojamų poveikių kiekį. Algoritmas po įvykdytos iteracijos gautus rezultatus prijungia prie sekančios iteracijos rezultatų ir per naują su jais atlieka skaičiavimus. Todėl reikia parinkti tokį pradinį poveikių kiekį, kad po iteracijos atrinktų termų kiekis būtų artimas galutiniam. Tada daugiau skaičiavimų bus atliekama ne su turimais rezultatais, o naujų iteracijų rezultatų sudarymui. Parinkus per didelį pradinių poveikių kiekį bus vykdomi pertekliniai skaičiavimai. Pasiekiamas maksimalus termų kiekis ir atliekami skaičiavimai tik gerins termus (ieškos ilgesnių). Funkciniam modeliui c432 parinkus pradinį poveikių kiekį n^3 analizė truko 4 kartus trumpiau, rastas apylygis esminių termų bei poveikių skaičius, išanalizuota 1,44 karto daugiau poveikių ir žymiai mažesnis (19 kartų) iteracijų skaičius lyginant su pradiniu poveikių kiekiu n^2.



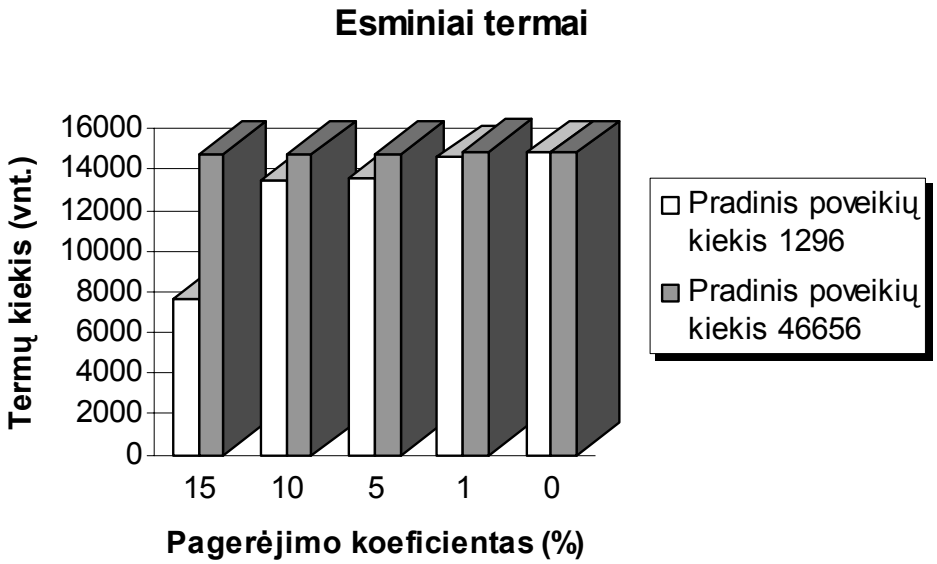
14 pav. C42 iteracijų kiekis

Kai pradinis poveikių kiekis n^3, vykdomų iteracijų kiekis sumažėja, tačiau kiekvienos iteracijos metu yra sugeneruojama žymiai daugiau poveikių ir iš viso išanalizuojama daugiau poveikių. Esant dideliam (15%) pagerėjimo koeficientui sugeneruojama 12 kartų daugiau poveikių, o ieškant galutinio sprendinio šis skirtumas sumažėja iki 1,44 karto (15 pav.).

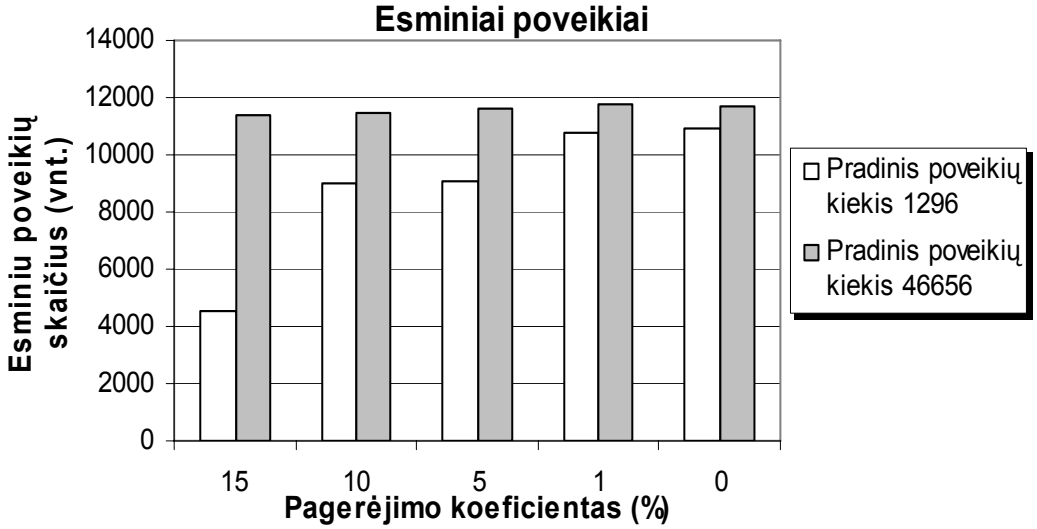


15 pav. C432 viso sugeneruota poveikių

Kai pradinis poveikių kiekis n^3 ir pagerėjimo koeficientas didelis (15%), yra išanalizuojama ne ką mažiau poveikių, kai jų pradinis kiekis n^2 ir pagerėjimo koeficientas lygus 0%. Kadangi išanalizuojama daugiau poveikių, tai ir sugeneruojama daugiau esminių termų bei atrenkama esminių poveikių (16-17 pav.).



16 pav. C42 esminių termų kiekis



17 pav. C432 Esminių poveikių kiekis

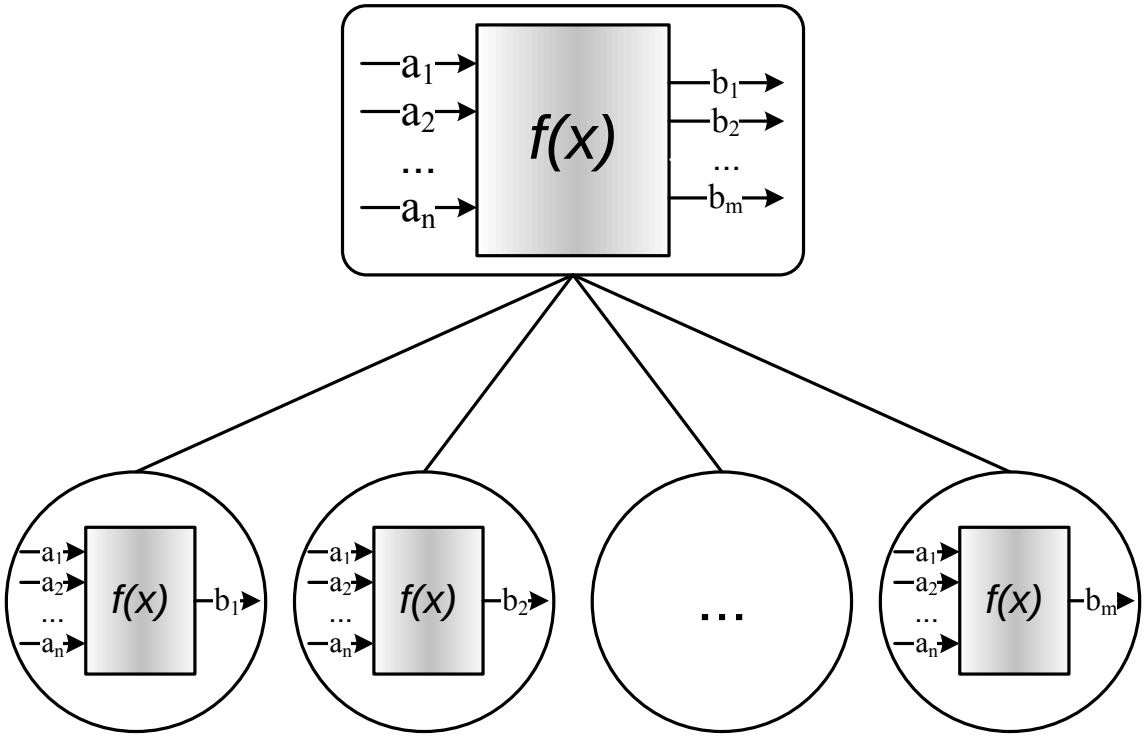
Iš grafikų matyti, jog parenkant pradinių poveikių kiekį n^3 gaunamas efektyvesnis skaičiavimas nei su pradiniu poveikių kiekiu n^2. Kai pradinis poveikių kiekis n^3 buvo sugeneruota daugiau poveikių ir nežiūrint į tai, žymiai trumpiau vyko analizė laiko bei iteracijų atžvilgiu, esminių termų bei poveikių kiekiai apylygiai. Dar galima pastebėti, jog vykdant analizę su pradiniu poveikių kiekiu n^3 po pirmų iteracijų buvo rasti esminių termų bei poveikių kiekiai labai artimi galutiniam sprendiniui.

4.6. GALIMI PATOBULINIMAI

Analizuojant algoritmą žemiausiame abstrakcijos lygyje lygiagretinimą galima atlikti matricos $X= ||x_{i,j}||_{n \times m}$ generavimą. Vieno įėjimo signalo poveikis išėjimams neturi įtakos kitų įėjimo signalų poveikių nustatymui. Lygiagrečių procesų skaičius būtų lygus įėjimų skaičiui n. Funkciniam modeliui paduodama pradinė įėjimų seka. Toliau kiekvienas procesas pradinėje įėjimų sekoje pakeičia po vieną atitinkamą įėjimo signalą (pirmas procesas pakeičia pirmąjį signalą, antras – antrąjį ir t.t. iki n). Gautos išėjimų sekos lyginamos su pradinės įėjimų sekos išėjimais. Taip nustatomas kiekvieno įėjimo signalo poveikis visiems išėjimams.

Gavus pakeisto įėjimo signalo išėjimų seką, lyginimą su pradine išėjimų seka galima dar skaidyti į m procesų. Tuomet lygiagrečių procesų kiekis bus lygus matricos $X= ||x_{i,j}||_{n \times m}$ elementų kiekiui, t.y. n×m. Šis lygiagretinimo būdas gali būti realizuotas modifikavus programos kodą taikant lygiagretaus programavimo principus.

Žvelgiant į algoritmą aukštesniu abstrakcijos lygiu matyti, jog analizuojamo funkcinio modelio termų sudarymo uždavinį galima skaidyti į kelis mažesnius uždavinius, kurių rezultatus pabaigoje galima apjungti tam, kad gauti viso funkcinio modelio sprendinį (18 pav.).



18 pav. Hierarchinis funkcinio modelio termų generavimo skaidymas

Išėjimai neturi įtakos vieni kitų termams. Kiekvienam išėjimui termai sudaromi atskirai ir baigus generuoti visiems išėjimams termus rezultatai yra apjungiami. Kiekvieno išėjimo atrinkti poveikiai gali sutapti su kitų išėjimų rezultatais (įėjimo sekomis). Apjungiant rezultatus vykdoma funkcinio modelio analizės viena iteracija. Jos metu negeneruojamos atsitiktinės įėjimų sekos,

naudojami tik poveikiai, gauti vykdant analizę kiekvienam išėjimui atskirai. Po rezultatų apjungimo gaunamas optimizuotas poveikių kiekis (nėra pasikartojančių), šie rezultatai (termai ir poveikiai) yra tokie pat kaip ir vykdant funkcinio modelio analizę neskaidant termų generavimo uždavinio. Šios metodikos privalumas yra tas, kad kiekvieno išėjimo termų generavimas vyksta atskirai ir atlikus vieno išėjimo analizę iš karto vykdoma sekanti iteracija. Neskaidant termų generavimo vieną iteraciją sudaro jau ne vieno išėjimo analizė, o visų. Todėl kiekvieno išėjimo analizė turi laukti kol baigsis kitų išėjimų analizės. Ir vykdymo laikas bus lygus visų išėjimų termų generavimo iteracijų laikų sumai. Skaidant termų sudarymą atskiriems procesams, vykdymo laikas bus lygus maksimalios vieno išėjimo analizės vykdymo trukmės ir apjungimo trukmės sumai.

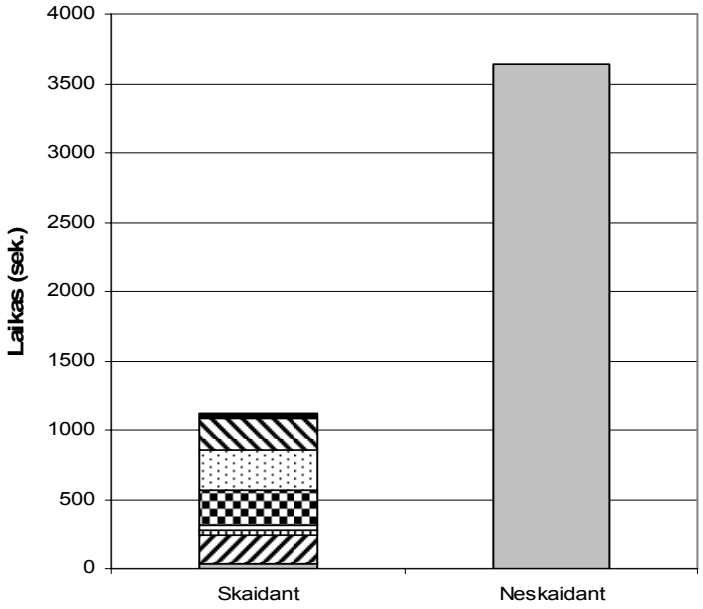
Funkcinio modelio kiekvieno išėjimo sudėtingumas skiriasi, vienam įtakos gali turėti daugiau įėjimų, kitam - mažiau, taip pat gali skirtis funkcijų veiksmų skaičiai. Todėl ir išėjimų analizių trukmė, termų kiekis skiriasi. Termų skirtumas gali įtakoti iteracijų skaičiui. Funkcinio modelio vienas išėjimas gali turėti mažiau termų ir jų skaičius surandamas vykdant mažiau iteracijų. Kito išėjimo termai gali būti sugeneruoti per didesnį iteracijų kiekį. Tai turi įtakos vykdymo laikui kai išėjimų generavimas nėra skaidomas. Bent vieno išėjimo termų padidėjimas turi įtakos kitų išėjimų termų generavimui, t.y. vykdoma sekanti iteracija. Jos metu yra atliekami skaičiavimai ir su tais išėjimais, kurių termų kiekis nekinta net papildomų iteracijų metu. Taip yra vykdomi laiko atžvilgiu nereikalingi skaičiavimai. Skaidant išėjimų termų generavimą ir vykdant net ne lygiagrečiai, o nuosekliai, kiekvienam išėjimui atskirai, skaičiavimų trukmė yra trumpesnė nei neskaidant generavimo, nes nėra vykdomos papildomos iteracijos ir neatliekami laiko atžvilgiu nereikalingi skaičiavimai.

5. EKSPERIMENTINĖ DALIS

5.1. HIERARCHINIS TESTŲ SUDARYMAS

Hierarchinis testinių vektorių sudarymo uždavinys yra atliekamas suskaidant viso funkcinio modelio poveikių generavimą atskiriems išėjimams. Kiekvienam išėjimui atskirai yra atliekami skaičiavimai. Gauti visų išėjimų rezultatai yra apjungiami ir gaunami viso funkcinio modelio esminiai termai ir poveikiai.

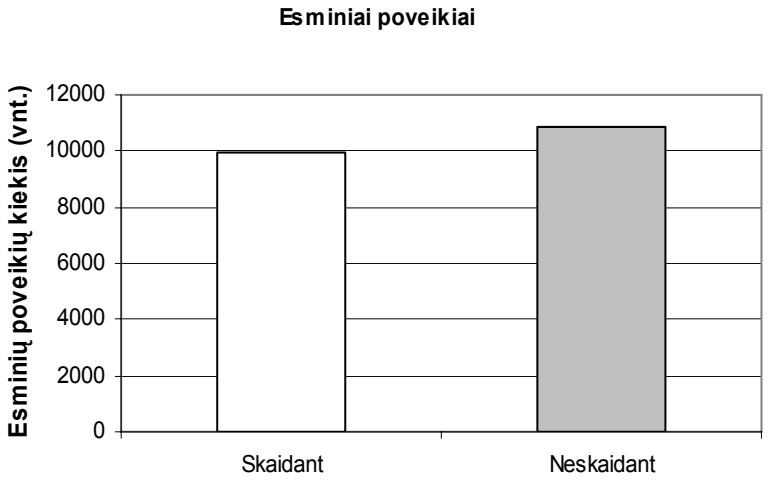
Eksperimentas buvo atliktas su funkciniu modeliu c432 (36 įėjimai ir 7 išėjimai). Pradinis poveikių kiekis n^2, pagerėjimo koeficientas 0% (skaičiuoja tol, kol termų kiekis didėja), maksimalus vieno išėjimo tiesioginės ir atvirkštinės funkcijų termų kiekis n^2.



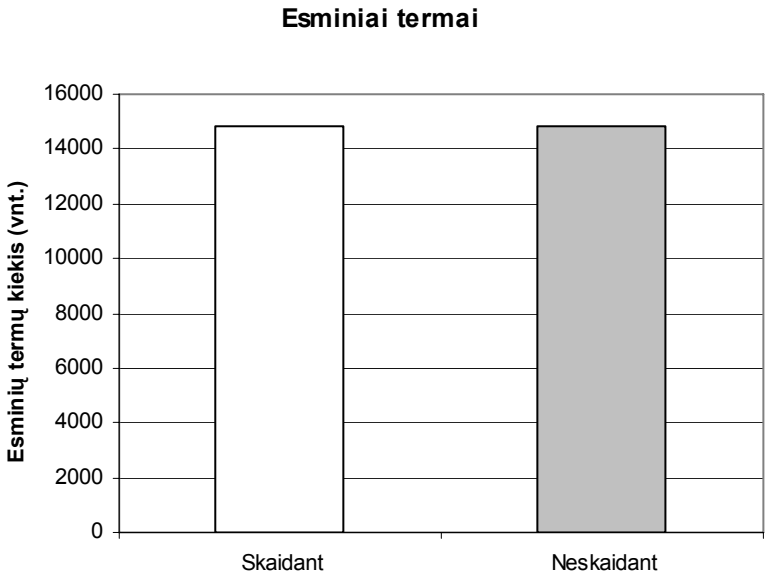
19 pav. Skaidant termų generavimą išėjimams atskirai ir neskaidant laikų palyginimas

Atlikus eksperimentą nustatyta, jog skaidant funkcinio modelio išėjimų termų generavimą analizė buvo vykdoma 3 kartus greičiau nei neskaidant (19 pav.). Skirtingi segmentai žymi atskirų išėjimų termų skaičiavimo laikus bei apjungimo laiką. Šį skirtumą lėmė tai, kad ne visi funkcinio modelio išėjimai turi vienodą kiekį termų ir vieniem reikia atlikti daugiau skaičiavimų kitiem mažiau, kad rasti galutinį sprendinį. C432 pirmo, antro ir ketvirto išėjimų termai buvo rasti mažiau nei per 40 iteracijų, o likusių išėjimų termų generavimui prireikė daugiau nei dvigubai didesnio iteracijų skaičiaus (9.2.3 skyriaus 12 ir 13 lentelės). Neskaidant funkcinio modelio termų generavimo visiems išėjimams būtų atliekamas vienodas iteracijų skaičius nepriklausomai ar kažkuriam išėjimui rastas maksimalus termų kiekis. Todėl yra vykdomi pertekliniai skaičiavimai, kurie žymiai įtakoja vykdymo laiką.

Skaidant funkcinio modelio termų generavimą buvo gauti rezultatai, kurių kokybė (esminių poveikių ir termų kiekiai) buvo lygiavertė neskaidant termų generavimo (20-21 pav.).

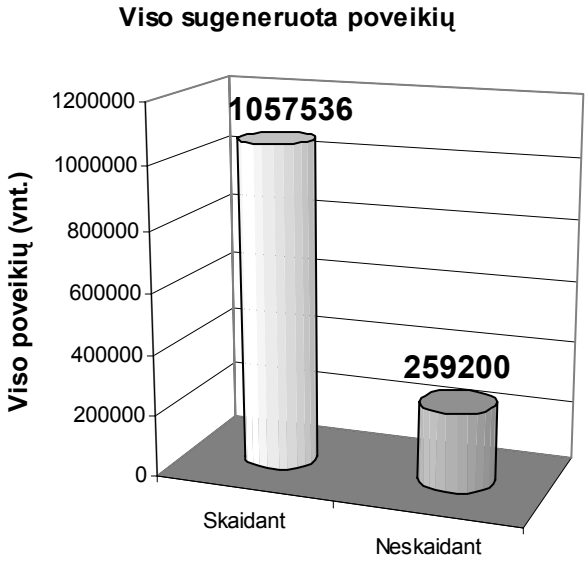


20 pav. Skaidant ir neskaidant termų generavimą esminių poveikių palyginimas



21 pav. Skaidant ir neskaidant termų generavimą esminių termų palyginimas

Eksperimento metu funkcinio modelio c432 išėjimų termų generavimo skaidymas davė didesnį visų sugeneruotų poveikių kiekį (4 kartus daugiau). Tačiau ne visi iš jų buvo vykdyti su kiekvienu išėjimu. Neskaidant termų generavimo iš viso buvo sugeneruota žymiai mažiau poveikių, tačiau visi jie buvo vykdyti su kiekvienu išėjimu (22 pav.).



22 pav. Skaidant ir neskaidant termų generavimą visų poveikių kiekių palyginimas

Sugeneravus termus visiems išėjimams atskirai ir juos apjungus gauti rezultatai (esminių poveikių bei termų kiekiai) panašūs į funkcinio modelio analizę su tais pačiais parametrais skaičiuojant visiems išėjimams termus kartu. Skaidant išėjimų generavimą sutaupoma laiko vykdant generavimus, nes naują iteracijos vykdymą sąlygoja tik vieno išėjimo termų kiekio padidėjimas, o neskaidant - visų išėjimų. Ir radus galutinį išėjimo termų kiekį, daugiau skaičiavimų su juo nėra vykdoma, todėl išvengiama perteklinių skaičiavimų.

6. IŠVADOS

Ištyrus algoritmo veikimą, nustatyta, jog funkcinio modelio termų sudarymo eigą ir rezultatus įtakoja funkcinio modelio sudėtingumas, pradinis generuojamų poveikių kiekis, pagerėjimo koeficientas, atsitiktinai generuojamų poveikių eilės tvarka ir maksimalus (ribojamas) termų kiekis vienam išėjimui. Taip pat eksperimentų pagalba nustatyta, kaip jie įtakoja algoritmo skaičiavimų kokybę. Norint efektyviai sudaryti loginių funkcijų termus ir atrinkti esminius poveikius, kurie gali geriausiai testuoti atitinkamą funkcinį modelį, reikia atsižvelgti į šiuos veiksnius. Todėl suprojektuotoje sistemoje šie dydžiai kinta dinamiškai (pagal vartotojo pasirinkimą).

Kadangi algoritmo veikimas paremtas ne funkcinio modelio struktūra, o įėjimų signalais bei reakcija į juos, ir išėjimų reikšmės viena kitai neturi įtakos, sistema buvo patobulinta taikant hierarchinę stambinančią testų generavimo strategiją: funkcinio modelio termų sudarymo uždavinys yra skaidomas į kelis mažesnius (kiek išėjimų) ir gauti rezultatai gale apjungiami. Remiantis eksperimentais nustatyta, jog skaidant termų generavimą išėjimams yra nevykdomi pertekliniai skaičiavimai ir analizės eiga paspartėja.

7. LITERATŪRA

1. Gert Jervan, “High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems”, Sweden, 2002.
2. M. A. Breuer, A. D. Friedman, “Diagnosis and Reliable Design of Digital Systems”, *Computer Science Press*, 1976.
3. “The International Technology Roadmap for Semiconductors (ITRS 2001)”, *Semiconductor Industry Association,* 2001.
4. A. Grochowski, D. Bhattacharya, T. R. Viswanathan, K. Laker, “Integrated Circuit Testing for Quality Assurance in Manufacturing: History, Current Status, and Future Trends”, *IEEE Trans. on Circuits and Systems – II,* Vol. 44, pp. 610-633, August 1997.
5. M. Abramovici, M. A. Breuer, A. D. Friedman, “Digital Systems Testing and Testable Design”, *IEEE Press*, 1990.
6. B. Beizer, “Software Testing Techniques”, (2nd edition), *Van Nostrand Reinhold*, 1990.
7. F. Ferrandi, G. Ferrara, D. Scuito, A. Fin, F. Fummi, “Functional Test Generation for Behaviorally Sequential Models”, *Design, Automation and Test in Europe (DATE 2001)*, pp. 403-410, 2001.
8. B. T. Murray, J. P. Hayes, “Hierarchical Test Generation Using Precomputed Tests for Modules”, *International Test Conference*, pp. 221-229, 1988.
9. J. Lee, J. H. Patel, “Architectural Level Test Generation for Microprocessors”, *IEEE Trans. CAD,* vol. 13, no. 10, pp. 1288-1300, October 1994.
10. S. Hellebrand, H.-J. Wunderlich, A. Hertwig, “Mixed-Mode BIST Using Embedded Processors”, *Journal of Electronic Testing: Theory and Applications*, pp. 127-138, No. 12, 1998.
11. O. H. Ibarra, S. Sahni, “Polynomially Complete Fault Detection Problems”, *IEEE Transactions on Computers,* Vol. C-24, No. 3, pp. 242-249, March 1975.
12. R. A. DeMillo, R. J. Lipton, F. G. Sayward, “Hints on Test Data Selection: Help for the Practical Programmer”, IEEE Computer, Vol.11, No.4, April 1978.
13. Irith Pomeranz and Sudhakar M. Reddy, “*Functional Test Generation for Full Scan Circuits*”, Electrical and Computer Engineering Department University of Iowa.
14. M. B. Santos, F.M. Gonçalves, I.C. Teixeira and J. P. Teixeira, “RTL-based Functional Test Generation for High Defects Coverage in Digital SOCs”, IEEE Computer, 2000
15. Serdar Tasiran, Kurt Keutzer, “*Coverage Metrics for Functional Validation of Hardware* Designs”, IEEE Design & Test of Computers, July-August 2001.

16. Jorge Campos, Hussain Al-Asaad, “Mutation-based Validation of High-level microprocessor implementations”, Department of Electrical and Computer Engineering University of California.
17. Abraitis Vidas, Bulotaitė Ieva, Masalskis Giedrius, “Programuojamosios logikos elemento gedimų modelio tyrimas”, [žiūrėta 2005-12-03] http://www.ktu.lt/lt/mokslas/konf05/konf_02/IT2005/Sekc08.pdf

8. TERMINŲ IR SANTRUMPŲ ŽODYNAS

Funkcinis modelis – logine funkcija išreikštas schemos veikimo principas.

Termas – n kintamųjų Bulio funkcijos išraiška, sudaryta iš m ($m \leq n$, kur n – įėjimų kiekis) kintamųjų, apjungtų vienos iš operacijų „konjunkcija“ arba „disjunkcija“ ženklais.

Poveikis – įėjimų seka ($x_1 \div x_n$).

Esminis termas – galimas loginės funkcijos termas, kuris nėra dalinis kitų atžvilgiu (pvz., X_2X_3 yra dalinis termo $X_1X_2X_3$ atžvilgiu, ir esminis yra tik pastarasis).

Esminis poveikis – poveikis, kuris sudaro bent vieną esminį termą.

Ins (angl. inputs) – įėjimai.

Outs (angl. outputs) – išėjimai.

DLL (angl. Dynamic Link Library) – dinaminė biblioteka.

UML (angl. Unified Modeling Language) - unifikuota modeliavimo kalba.

PA – panaudos atvejai.

Sutrikimas - programa netenkina specifikacijos.

Gedimas - vidinė programos būsena prieštarauja laukiamai būsenai.

Defektas – programos kodas, kuris sąlygoja sistemos gedimą.

Klaida – klaida, kurią padarė programuotojas, sukurdamas defektą.

PK – pagerėjimo koeficientas (kiek procentų turi pagerėti rezultatai).

9. PRIEDAI

9.1. FUNKCINIO MODELIO SUDARYMAS

Funkcinių modelių *.dll failai yra sudaromi naudojant *Microsoft Visual Studio .NET 20003* programinę įrangą. Funkcinių modelių kūrimui yra skirtas „dll“ katalogas, kuriame yra saugoma *MS Visual Studio .NET 2003* projektas. Funkcijos matematinis modelis sudaromas panaudojus šį kodą:

```
#include "stdafx.h"
BOOL APIENTRY DllMain (HANDLE hModule, DWORD  ul_reason_for_call, LPVOID lpReserved)
{
    return TRUE;
}
#define INS    //Įrašomas įėjimų kiekis
#define OUTS   //Įrašomas išėjimų kiekis
// Funkcijos parametrai: n - įėjimų kiekis; m - išėjimų kiekis
extern "C" __declspec(dllexport) void LogicFunctionParams (int *pIns, int *pOuts);
extern "C" __declspec(dllexport) void LogicFunctionParams (int *pIns, int *pOuts)
{
        *pIns = INS;
        *pOuts = OUTS;
}
// Loginė funkcija
extern "C" __declspec(dllexport) void LogicFunction (char *gg,char *og,int in,int on,int
*gerai);
extern "C" __declspec(dllexport) void LogicFunction (char *gg,char *og,int in,int on,int
*gerai)
{
 //Čia įterpiamas funkcijos matematinis modelis
}
```

Į užkomentuotas vietas reikia įrašyti reikiamus duomenis: n reikia pakeisti į įėjimų kiekį, m – į išėjimų. Sukompiliavus duotą kodą, gaunamas funkcinio modelio ddl.dll failas. Tereikia funkcinį modelį pervadinti atitinkamu pavadinimu (pvz.: c432.dll) ir patalpinti „functions“ kataloge. Funkcinis modelis paruoštas.

9.2. EKSPERIMENTŲ MEDŽIAGA

9.2.1. Testinė funkcija

Pradžioje eksperimentas buvo atliktas su nedidele testine funkcija (įėjimų kiekis n = 3, išėjimų kiekis m = 1). Tiriama loginė funkcija:

$$f_1(x_1, x_2, x_3) = \overline{x_1} x_2 \overline{x_3} + x_1 \overline{x_2} x_3 + x_1 x_2 \overline{x_3} = x_2 \overline{x_3} + x_1 \overline{x_2} x_3$$

Šis užrašas atitinka tiesioginės funkcijos aprašą (t.y. kada funkcija grąžina reikšmę 1). Netiesioginę funkciją (t.y. kada funkcija grąžina 0) aprašoma taip:

$$f_0(x_1, x_2, x_3) = \overline{x_2} \overline{x_3} + x_2 x_3 + \overline{x_1} \overline{x_2} = \overline{x_2} \overline{x_3} + x_2 x_3 + \overline{x_1} x_3 = \overline{x_2} \overline{x_3} + x_2 x_3 + \overline{x_1} x_2 x_3$$

Funkcijos reikšmės pateiktos 9 lentelėje.

2 lentelė. Testinės funkcijos reikšmių lentelė

X_1	X_2	X_3	Y_1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Kadangi funkcija nėra labai sudėtinga, todėl jos analizei buvo generuojami visi įmanomi poveikiai. Programai atlikus skaičiavimus gauti rezultatai:

3 lentelė. Tiesioginės testinės funkcijos termai

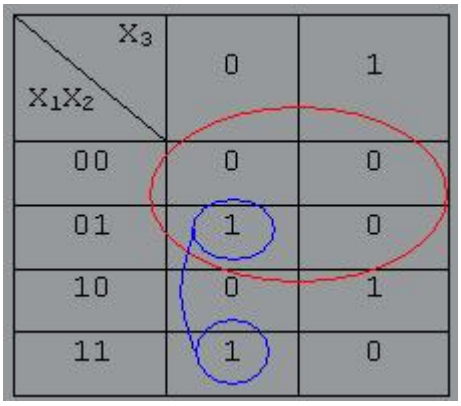
X_1	X_2	X_3
1	0	1
V	1	0

4 lentelė. Atvirkštinės testinės funkcijos termai

X_1	X_2	X_3
V	1	1
0	N	V
V	0	0

Gauti analizės rezultatai: termą formuoja viena eilutė: N arba V žymi, jog kintamasis į termą neįeina, o poveikyje formuojant šį termą buvo paduota atitinkamai 0 arba 1; 0 – invertuotas; 1 – tiesioginis. Galima pastebėti, jog programos sudaryti tiesioginės funkcijos termai atitinka analizuojamos funkcijos elementus. Atvirkštinės funkcijos pirmas ir trečias sudaryti termai atitinka

analizuojamos atvirkštinės funkcijos elementus. Tačiau gautas atvirkštinės funkcijos antrasis termas kertasi su tiesioginės funkcijos pirmuoju termu:



23 pav. Testinės funkcijos persikertantys termai

Apjungę šiuos du termus gauname pataisytą antrąjį atvirkštinės funkcijos termą:

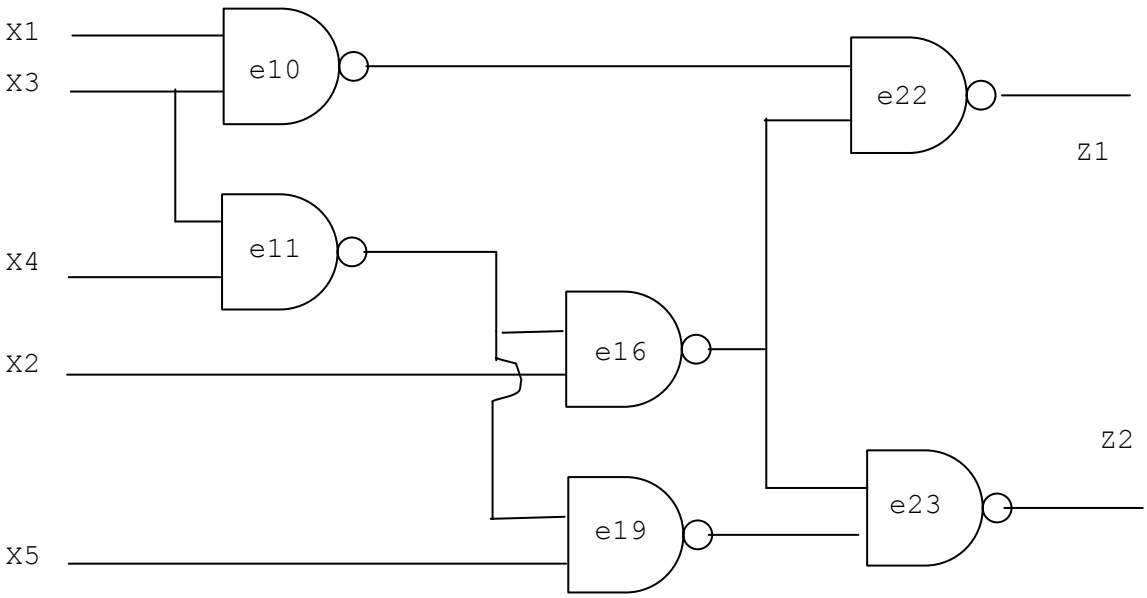
$\overline{x_1}\,\overline{x_2}x_3$

Tačiau šis apjungimas (patikslinimas) įmanomas tik tuomet, kai visi termai rasti. Kadangi realizuota programa analizuoja žymiai sudėtingesnes funkcijas, kurių analizės pabaigą sunku nustatyti, termų kiekis yra ribojamas, todėl nėra tikslinga realizuoti šią operaciją.

Analizės atrinkti esminiai poveikiai: 101, 110, 111, 001 ir 100.

9.2.2. Funkcinis modelis C17

Programa buvo paleista su c17 funkciniu modeliu, kurio schema yra pateikta žemiau:



24 pav. C17 schema

Funkcinio modelio c17 išėjimus galima išreikšti funkcijomis:

$$z1(x_1, x_2, x_3, x_4, x_5) = x_1 x_3 + x_2 \overline{x_3} + x_2 \overline{x_4}$$

$$z2(x_1, x_2, x_3, x_4, x_5) = x_2 \overline{x_3} + x_2 \overline{x_4} + \overline{x_3} x_5 + \overline{x_4} x_5$$

Atlikus eksperimentus gauti rezultatai:

```
Funkcinis modelis: c17.dll
Iejimu kiekis: 5
Didziausias galimas isejimu kiekis: 2
N - iejime buvo paduota 0 formuojant terma
V - iejime buvo paduota 1 formuojant terma
0 - termo atvirkstinis signalas, 1 - termo tiesioginis signalas
Kintamieji eina is eiles: nuo 1 iki 5
```

5 lentelė. C17 dviejų analizių gauti termai

Pirmoji analizė	Antroji analizė
Y1:	Y1:
Tiesiogine:	Tiesiogine:
N 1 0 V V	N 1 V 0 V
1 N 1 V V	N 1 0 V N
N 1 V 0 N	1 N 1 V V
Atvirkstine:	Atvirkstine:
0 0 V N N	0 V 1 1 V
V 0 0 N V	V 0 0 V V
0 V 1 1 N	0 0 V N V
Y2:	Y2:
Tiesiogine:	Tiesiogine:
N N V 0 1	V N 0 V 1
N N 0 V 1	N N V 0 1
V 1 0 V N	N 1 0 V N
N 1 V 0 N	N 1 V 0 N
Atvirkstine:	Atvirkstine:
N 0 V N 0	N V 1 1 V
N N 1 1 V	V 0 N N 0

Gauti poveikiai:

6 lentelė. C17 dviejų analizių atrinkti poveikiai

Pirmoji analizė	Antroji analizė
00100	01111
00101	10011
00011	10000
01011	00101
11010	01101
10001	01010
00111	10111
10111	01100
01110	
01100	

Kaip matyti, išėjimų funkcinio modelio c17 gauti termai sutampa su schemos aprašytų išėjimų loginių funkcijų termais (tiesioginių funkcijų). Algoritmas pilnai atstatė c17 išėjimų termus.

Eksperimentas buvo atliktas lyginant dviejų analizių rezultatus. Abiejų analizių metu buvo gauti tie patys termų rinkiniai (skiriasi tik eilės tvarka), tačiau su skirtingu poveikių kiekiu. Pirmoji analizė sudarė 12 esminių termų ir atrinko 10 esminių poveikių, o antroji tuos pačius termus sudarė atrenkant 8 poveikius. Eksperimentas rodo, jog rezultatų kokybę įtakoja atsitiktinai generuojamų poveikių eilės tvarka.

9.2.3. Funkcinio modelio C432 eksperimentų rezultatai

Toliau vyko programos vykdymas su funkciniu modeliu c432 vedant statistiką. Pradžioje buvo vykdomos trys vienodos analizės su vienodais pradiniais duomenimis. Visų trijų analizių duomenys apytiksliai sutapo. Vėliau padidintas pradinis poveikių kiekis ir rezultatai palyginti su prieš tai atliktų trijų analizių rezultatų vidurkiu.

C432 (36 įėjimai, 7 išėjimai) buvo vykdomos trys analizės su pradiniu poveikių kiekiu n^2, t.y. 1296. Gauti rezultatai pateikti lentelėse.

7 lentelė. Funkcinio modelio C432 analizės Nr.1 rezultatai

Pagerėjimo koef. %	15	10	5	1	0
Laikas (sek)	38,938	184,891	235,343	911,797	4144,843
Viso poveikių	16848	66096	75168	204768	238464
Esminiai poveikiai	4966	9032	9301	11111	10798
Esminiai termai	9428	13472	13673	14793	14859
Iteracijos	8	14	19	43	155

8 lentelė. Funkcinio modelio C432 analizės Nr.2 rezultatai

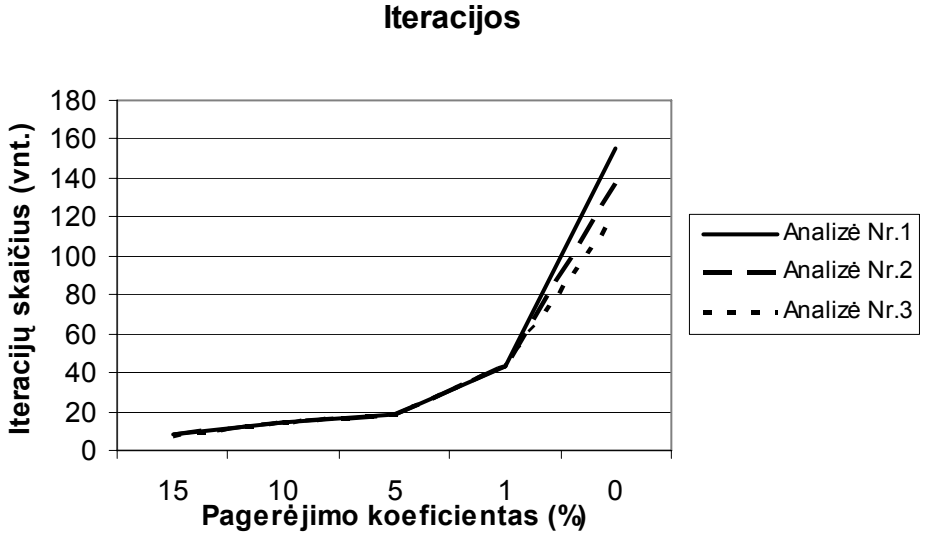
Pagerėjimo koef. %	15	10	5	1	0
Laikas (sek)	38,672	182,906	265,187	863,657	3632,485
Viso poveikių	16848	66096	77760	150336	256608
Esminiai poveikiai	4966	9054	9455	10625	10902
Esminiai termai	9274	13417	13709	14624	14849
Iteracijos	8	14	19	43	137

9 lentelė. Funkcinio modelio C432 analizės Nr.3 rezultatai

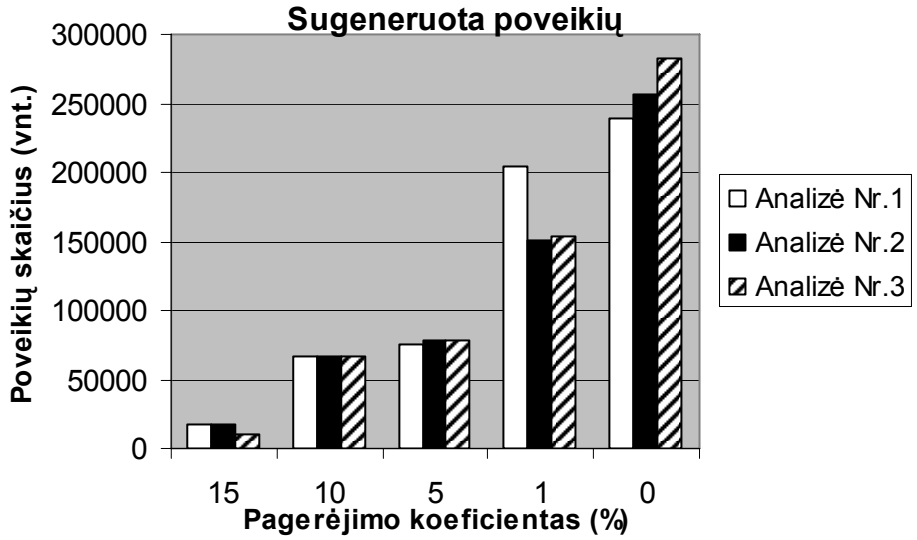
Pagerėjimo koef. %	15	10	5	1	0
Laikas (sek)	22,703	186,375	263,11	845,218	3121,547
Viso poveikių	10368	66096	77760	154224	282528
Esminiai poveikiai	3725	8974	9344	10517	10990
Esminiai termai	7688	13443	13633	14599	14819
Iteracijos	7	14	19	43	120

10 lentelė. Funkcinio modelio C432 analizių Nr.1-3 rezultatų vidurkis

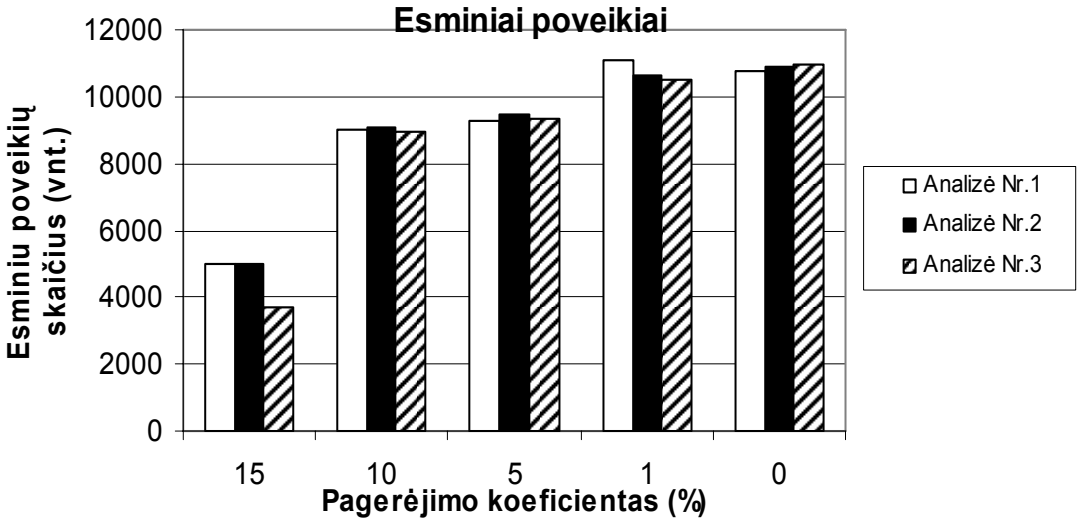
Pagerėjimo koef. %	15	10	5	1	0
Laikas (sek)	33,43767	184,724	254,5467	873,5573	3632,958
Viso poveikių	14688	66096	76896	169776	259200
Esminiai poveikiai	4552,333	9020	9366,667	10751	10896,67
Esminiai termai	8796,667	13444	13671,67	14672	14842,33
Iteracijos	7,666667	14	19	43	137,3333



25 pav. C42 analizių iteracijos



26 pav. C42 analizių viso sugeneruota poveikių



27 pav. C42 analizių esminiai poveikiai

11 lentelė. Funkcinio modelio C432 analizė, kai pradinis poveikių kiekis n^3 t.y. 46656

Pagerėjimo koef. %	15	10	5	1	0
Laikas (sek)	291,656	292,312	387,938	583,14	693,61
Viso poveikių	186624	186624	233280	326592	373248
Esminiai poveikiai	11364	11425	11599	11782	11697
Esminiai termai	14786	14797	14780	14867	14814
Iteracijos	3	3	4	6	7

Su funkciniu modeliu c432 buvo atliktas dar vienas eksperimentas, kurio metu termai buvo generuojami kiekvienam išėjimui atskirai. Gauti visų išėjimų rezultatai apjungti. Pradinis poveikių kiekis n^2, pagerėjimo koeficientas 0% (kol termų kiekis nedidės).

12 lentelė. C432 išėjimų analizė

Išėjimas	y1	y2	y3	y4	y5	y6	y7	Suma
Laikas (sek)	37,797	200,14	42,781	34,532	249,812	284,453	234,938	1084,453
Viso poveikių	104976	143856	27216	49248	209952	256608	265680	1057536
Esminiai poveikiai	571	2590	2592	1359	2592	2592	2591	14887
Esminiai termai	571	2590	2592	1359	2592	2592	2591	14887
Iteracijos	35	87	20	37	94	103	86	462

13 lentelė. C432 išėjimų rezultatų apjungimas

Laikas (sek)	31,219
Viso poveikių	14887
Esminiai poveikiai	9927
Esminiai termai	14822