KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Giedrė Maskoliūnaitė

# Testų sudarymo metodų vėlinimo gedimams tyrimas ir sudarymas

Magistro darbas

Darbo vadovas
Prof. V. Jusas

Kaunas, 2008

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Giedrė Maskoliūnaitė

# Testų sudarymo metodų vėlinimo gedimams tyrimas ir sudarymas

Magistro darbas

Recenzentas
Prof. K.Motiejūnas
2008-05-26

Vadovas
Prof. V. Jusas
2008-05-26

Atliko
IFM-2/5 gr. stud.
Giedrė Maskoliūnaitė
2008-05-26

Kaunas, 2008

# TURINYS

## ANOTACIJA (lietuvių k.)

Šiais laikais naujos technologijos sudaro vis didesnę mūsų gyvenimo dalį, bet jų jau greičiausiai neįsivaizduotume savęs. Tačiau ne visi naujų tachnologijų kūriniai tarnauja ilgai, taip pat kaip ne visi jie veikia teisingai. Tam, kad nustatytume, jog įrenginys veikia teisingai raikalinga atlikti testavimą.

Mažėjant schemų dydžiams, augant metalo tankiui ir didėjant lustų veikimo greičiui, vėlinimų testavimas tampa būtinybe tam, kad palaikyti kokybę susijusią su schemos veikimo greičių sutrikimais. Vėlinimo gedimų tyrimas atsako į klausimą ar schema veikia teisingai be didelių nuostolių.

Šiame darbe aptarsime vėlinimo gedimų atsiradimo priežastis bei galimas juos įtakojančias sąlygas. Taip pat analizuosime vėlinimo gedimų testavimo modelius, metodus, jų privalumus, trūkumus ir galimus trūkumų sprendimus. Tyrimui pilnai atlikti inicijuosime eksperimentą, siekdami išsiaiškinti tiesioginės ir netiesioginės įtakos poveikį vėlinimų gedimų funkciniam testui.

## ANOTACIJA (anglų k.)

Nowadays new technologies contains the bigger part of our lifes, we couldnt imagine our selves without them. Otherwise, not all creations of new technologies are serving so long, also not all of them are serving right and effectively. When the system size degrees, the metal grows, and the time working on chips grows – the delay fault testing becomes very important part of system creation process. The delay fault test answers to the question does the system works right and without any big loses.

The test process has important point in Delay test has been investigated for many years. We can say that test process contains on these steps: setting the purpose of the test, designing the test, controlling the test process, perform the test and analyze test results. Also we will shortly discuss about basic gauge of testing process.

The main point of this paper is to analyze the models of delay faults test processes. To see the best and the worst sides of delay faults and represent experimental test to demonstrate the the robust and non-robust test advantagies in delay fault test.

# 1 ĮVADAS

Sparčiai tobulėjant skaitmeninių įtaisų gamybos technologijoms, testavimo aparatūra turi tobulėti kartu. Testavimo aparatūra yra sudaryta iš tokių pat integrinių schemų, kurias ji turi testuoti ir ši aparatūra yra pakankamai brangi.

Nagrinėsime skaitmeninių įtaisų testavimą visose stadijose: kūrimo, realizacijos bei naudojimo. Kūrimo stadija galime suprasti kaip idėją, mintį ar tiesiog tam tikrų užsakovo raikalavimų apibrėžtą ir numatytą kurti įrenginį. Sekantis etapas – realizacija, kurią galime apibrėžti kaip projektavimo procesą iki galutinio produkto.

Ir paskutinė stadija – naudojimo, kurioje jau turint pagamintą įrenginį, galime tikrinti kaip jis veikia ir kiek teisingai veikia.

Rinkoje nuolat auga poreikis kurti vis greitesnes sistemas. Tam, kad sistemos funkcionuotų nepriekaištingai reikalinga atlikti testavimą. Tam, kad atlikti testavimą, visų pirma reikia įsigilinti į užsakovo nurodytas reikalavimų tam tikram įrenginiui specifikacijas, tuomet galima testuoti įrenginio veikimą.

Galime teigti, jog testavimo procesą sudaro tokie žingsniai:

* testo tikslo formulavimas,
* testo projektavimas,
* testo tikrinimas,
* testo vykdymas,
* rezultatų analizė.

Svarbiausia testavimą kuo geriau atlikti projektavimo eigoje. Testuojant gaminamą produkta reikalinga remtis reikalavimais kuriamam produktui.

Testavimo programas būtina sudaryti kuo tikslesnes, kad būtų galima aptikti kuo daugiau sistemos sutrikimų prieš atiduodant produktą vartotojams. Kadangi naujosios technologijos vystosi labai sparčiai, taip pat ir kuriamuose vis naujesniuose skaitmeniniuose įrenginiuose atrandama vis daugiau naujų ir dar nežinomų sutrikimų.

Sutrikimai galbūt nepakeičia sistemos veikimo principo, tačiau gali įtakoti sistemos veikimo greitį. Tokie sutrikimai vadinami sistemos veikimo laiko vėlinimų gedimais. Modeliuoti vėlinimo gedimus galima kuriant greitus sinchrosignalo poveikius schemose arba testavimą atlikti tam tikros schemos greičiu.

Kuriant testavimo programas labai svarbu atsižvelgti į testų kokybę, kadangi tai įtakoja sistemos vystymo eigą ir laiką iki galutinio produkto. Taip pat svarbu, kad testavimas būtų tikslus ir atsižvelgiant į būsimo produkto kainą. Kuo tikslesnis testavimas vykdomas projektavimo eigoje, tuo produkto kaina didesnė. Aišku, norint, kad testas būtų kuo tikslesnis, didėja jo kurimo laikas, o taip pat mažėja kaina. Aišku, norima, kad vartotojas gautų kuo kokybiškesnį produktą, todėl testavimą turime pradėti vykdyti kuo ankstesnėse produkto vystymo stadijose, kad užtikrinti kuriamo produkto kokybę.

# 2 TESTŲ SUDARYMO IR VYKDYMO ANALIZĖ

Šiame skyriuje bendrai aptarsime jau ištirtus bei išnagrinėtus vėlinimo gedimų testavimo metodus, jų problemas. Taip pat bendrai aptarsime kas tai yra vėlinimo gedimai, kaip jie atsiranda ir kaip galime juos užfiksuoti tiriamose sistemose.

## 2.1 Susiję tyrimai ir darbai

Jau yra ištirta nemažai metodų tiek kelio vėlinimo gedimų testavime tiek perjungimo testavime ( Transition Faults and the Path Delay Faults (PDF)). Atsižvelgiant į tai, kad schemas sudaro daug kelių, daug kelių junginių ir yra reikalinga juos testuoti, ištirti jų veikimą – testavimo atlikimo kokybė yra labai svarbus veiksnys. Didelė tyrimų dalis buvo atlikta klasifikuojant skirtingus kelių vėlinimo gedimų testus, tiksliau testus skirtus skirtingiems gedimams atrasti.Kelių vėlinimo gedimai buvo suklasifikuoti į dvi pagrindines grupes - tiesioginės ir netiesioginės kelių vėlinimo gedimų įtakos klaidos (Robust PDF and Non-Robust (NR) PDF) [12]. Detalizuojant galime pagal K. Fuchs, M. Pabst ir T. Rossel straipsnyje *"RESIST: A recursive test pattern generation algorithm for path delay faults considering various test classes"* nagrinėtus pavyzdžius [11] išskirti penkis pagrindines kategorijas:

* GRT (General Robust Tests) – Bendras tiesioginės įtakos testas

* GNRT (General Non-Robust Tests) – Bendras netiesioginės įtakos testas
* HFRT (Hazard Free Robust Tests) – Spėjamas nepairbotas tiesioginės įtakos testas
* RRT (Restricted Robust tests) – Apribotas tiesioginės įtakos testas
* RNRT (Restricted Non-Robust Tests) – Neapribotas tiesioginės įtakos testas

**Trūkumai:** Kad naudoti tokius detalizuotus testus reikalinga ypatingai ištobulinta vertinimo sistema. Kartu išauga ir ATPG sistemos sudėtingumas.[12] Sintezavimo procedūra pažingsniui identifikuoja perteklinius kelių vėlinimo gedimų rinkinius, jų nemurejant. šių metodų naudojimo apribojimas kaip trūkumas yra tai, kad jų naudojimas yra nepraktiškas didelių schemų testavime. patobulinti šiuos apribojimus, ir tokiu atveju buvo pasiūlyta [11] naudoti pasikartojantį metodą identifikuoti perteklinius kelius ir testų generavimui. Tokiu atveju, jokio kelių medžio nereikia ir metodas yra efektyvus identifikuojant dideles schemas(daug kelių. Todėl schemas su didelių kelių skaičiumi tama vis sudėtingiau testuoti.

Kitas žinomas ATPG mechanizmas yra aprašytas S. Yihe ir W. Qifa straipsnyje *"FSIMGEO: A Test Generation Method for Path Delay Fault Test Using Fault Simulation and Genetic Optimization"* [13]. Šio mechanizmo veikimas paremtas klaidų testavimu schemų keliuose, tačiau neužfiksuojamos klaidos mažiau kritiniuose schemos keliuose. Kad šią bėdą apeiti, segmentų vėlinimo gedimai buvo aprašyti ir išnagrinėti [14]. Šiuose straipsniuose autoriai nagrinėja vėlinimo gedimų testavimo techniką kritiniuose netestuojamuose keliuose, kur tiesiogiai testuojami didžiausi schemos kelių segmentai, kurie nėra padengiami jokiais testuojamais kritiniais keliais. Šio metodo trūkumas , kad neįmanoma testuoti didelio skaičiaus netestuojamų kritinių schemų kelių ir generuoti netiesioginės įtakos vėlinimo gedimų schemose

Visiškai skirtingas požiūris į kritinių testuojamų schemos kelių pasirinkimą yra išnagrinėta [15-16]. Kai kurie autoriai išnagrinėjo ilgiausių kelių generavimą per kiekvieną ventilį[16]. Kad tai įgyvendinti autoriai pristatė naują algoritmą, kuris paremtas sudėtingesniu Tiesioginio aciklinio grafiko (Directed Acyclic Graph (DAG)) *G* vekimu. Grafiko G viršūnė *v* ir kelio svoris *l* kaip įėjimo reikšmės apibrėžia visus schemos kelius, kurie pereina per viršūnę v ir turi kelio svorį *l* .

Taip pat yra keletas tyrinėjimų šiems metodams trumpinti laike, kurie taip pat įtraukiami į algoritmą. Keletas šių metodų trūkumų yra:

1) Metodo panaudojimas nepraktiškas schemoms, turinčioms daug ilgų netestuojamų kelių.[12]

2) Testinius rinkinius sudaro w skaičius, kuris reiškia, kad segmentas svoriu w pradendant nuo mazgo ar kažkokio įėjimo ir baigiant ties tam tikru išėjimo mazgu. Tačiau atsižvelgiant į tai, kad užfiksuotos schemos kelio klaidos yra pagrįstos tik rinkinių skaičiumi ir kelio svoriu, toks testavimas yra nepraktiškas, eikvoja papildomą laiką tiems, schemos keliams, kurie yra visiškai netestuojami.[12]

3) Kai visoje schemoje stengiamasi užfiksuoti ir ištirti ilgiausią schemos kelią, galime teigti, jog šis kelias gali būti vienas trumpiausių visoje schemoje. Tad grafiku paremtas metodas nėra tinkamas klaidoms tokiuose keliuose testuoti. Kaip alternatyva šiam testavimui yra pasiūlytas tikslesnis metodas, kai schemos keliai testuojami ir aptinkami remiantis jau nustatytais ir ištirtais kritinių kelių statistiniais kriterijais . Tačiau šiame metode problema atsiranda tame, kad visuomet reikia užtikrinti statistinių kriterijų atnaujinimą ir skirtingiems keliams turi būti nustatyti skirtingi kriterijai jiems testuoti. [12-17-18]

## 2.2 Vėlinimo gedimai, savybės

Sistemų gamybos ir fiziškos klaidos negali būti pilnai testuojamos, aptinkamos remiantis vien tiktais matematiniais metodais. Tokiu atveju, vėlinimo gedimai schemose laikomi loginėmis klaidomis ir visi skirtingi vėlinimo gedimų testavimo metodai gali būti modeliuojami kaip atlikto testo klaidos. Tad, trumpai aptarsime kas tai yra vėlinimo gedimai ir kaip juos galime aptikti testuojamose sistemose.

Vėlinimo gedimų testavimas – schemos tikrinimas, ar ji gali perduoti signalo pasikeitimą į išėjimo mazgą per laiką, kuris nebūtų ilgesnis, už laiko tarpą tarp dviejų signalo impulsų.

Vėlinimo gedimai –gedimai atsirandantys, kai signalo pokytis nespėja pasiekti išėjimo reikšmės lygio per visą sinchronizavimo laiką.

Tam tikri pramoniniai defektai nekeičia loginio sistemos funkcionalumo, bet gali sukelti vėlinimus laike.

Vėlinimo gedimus gali įtakoti šios sąlygos[19]:

- Priešingos arba pasikeitusios reikšmės keliuose ir mazguose
- Sistemos kelių, įėjimų-išėjimų elementų, laidų sujungimų klaidos
- Matinimo šaltinio reikšmių skirtumai, pasikeitimai
- Proceso pokyčiai, galintys pakeisti visą sistemos specifikaciją ir įtakoti vėlinimus

Vėlinimo gedimai gali būti modeliuojami įvairių metodų pagalba, iš kurių patikimiausiu laikomas schemos kelių testavimas *(angl. Path Delay Fault (PDF))* ir perėjimo kliadų (Transition fault) modeliai.

## 2.3 Apibendrintas vėlinimo gedimų modelis

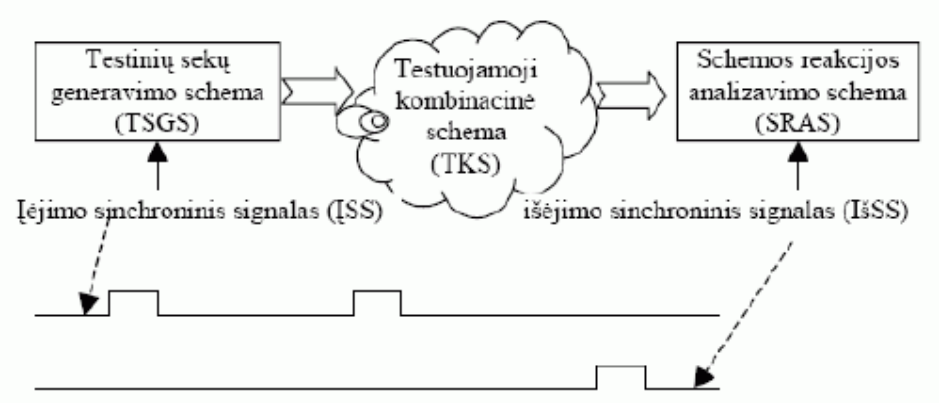
Aptarsime koks yra apibendrintas vėlinimo gedimų modelis.

“Apibendrintas modelis, skirtas vėlinimo gedimams testuoti, pavaizduotas 1 pav. Testinių sekų generavimo schemoje (TSGS) surašytos testinių vektorių poros (TVP), kuriomis sugeneruojamas signalo fronto sklidimas keliu, kuriame reikia patikrinti vėlinimo gedimą. Pirmu įėjimo sinchroninio signalo (ĮSS) frontu kombinacinės schemos (TKS) įėjimuose pateikiamas pirmasis testinis vektorius iš TVP. Tai atlikus, palaukiama (laiko tarpą, ne trumpesnį už laiko tarpą tarp dviejų gretimų sinchroninio signalo, kuris yra siunčiamas į realiai veikiančią schemą, impulsų), kol TKS išėjime gaunamas stabilus išėjimo signalas. Tuomet siunčiamas antrasis ĮSS frontas, kurio metu TSGS schema pateikia antrąjį testinį vektorių iš TVP.[1]

Antrasis testinis vektorius iš TVP sudarytas taip, kad generuotų signalų pasikeitimus iš loginio „0“ į loginį „1“ (arba atvirkščiai) TSGS išėjimuose, bet tik

tuose, kurie gali sugeneruoti fronto sklidimą reikiamu patikrinti keliu. Po antrojo ĮSS fronto palaukiama laiko tarpą, lygų intervalui tarp dviejų gretimų sinchroninio signalo impulsų, kurie siunčiami realiai veikiančiai schemai, ir tuomet sugeneruojamas išėjimo schemos sinchroninis signalas (IšSS). IšSS signalo momentu schemos reakcijos analizavimo schema (SRAS) analizuoja TKS schemos išėjimo vektorių ir nustato, ar yra vėlinimo gedimas testuojamajame kelyje, ar nėra.[1]

Schemos TKS įėjime sugeneruotas signalo pasikeitimas nebūtinai gali būti perduotas į TKS išėjimą. Visiškai įmanoma, kad TKS įėjime sugeneruotas signalo fronto pasikeitimas kur nors testuojamosios kombinacinės schemos viduje dėl loginių operacijų „užges“, t.y. nesugeneruos signalo pasikeitimo TKS išėjime.”[1]



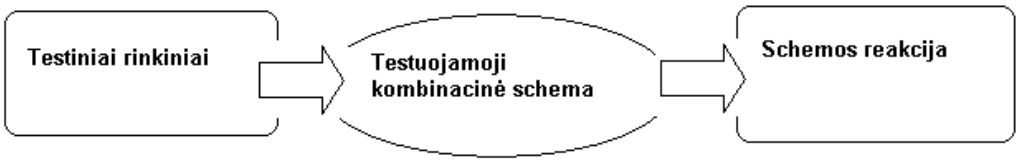

*LENTELĖ 1 **APIBENDRINTAS** VĖLINIMO GEDIMŲ MODELIS*

Laikas, reikiamas signalui sklisti iš testuojamosios schemos įėjimo į išėjimą, yra lygus kiekvieno loginio elemento, esančio signalo sklidimo kelyje, vėlinimo trukmių sumai. Šiame modelyje taip pat tariama, kad kiekvieno elemento vėlinimo trukmė gali labai skirtis priklausomai nuo aplinkos temperatūros, maitinimo įtampos svyravimų ir kitų veiksnių.

## 2.4 Vėlinimo gedimų testavimas

Testo generavimo metu naudojamas imitacinis schemos modelis ir nežinoma konkreti schemos realizacija, tenka naudotis poveikio porų atsitiktine paieška. Šiems gedimams aptikti naudojami testai, kurie kombinacinėje schemoje aptinka vėlinimus, kai signalas iš schemos įėjimo į išėjimą sklinda ilgiau nei praeina laikas tarp dviejų sinchroninio signalo impulsų. Tiriant vėlinimo gedimus dažniausiai tariama, kad kitų schemos gedimų nėra, tuomet vėlinimo gedimų testavimas būna teisingas.[1]

2 pav. Pavaizduota  testavimo veiksmų schema.




*LENTELĖ 2 – TESTAVIMO VEIKSMŲ SCHEMA[1]*

Egzistuoja įvairių gedimų, kurie pasitaiko projektuojant schemas. Taip pat yra sukurta įvairių modelių, kurie aprašo gedimus.

Vėlinimus tikriname naudodami tam tikras duomenų rinkinių poras. Šios rinkinių poros signalų reikšmės apsprendžia, kurie įėjimai keičiasi ir kurie išlieka pastovūs. Gali keistis vienas ar daugiau įėjimų. Pokyčiai įėjimuose, kurie iššaukia pokyčius ir išėjimuose, gali patikrinti vėlinimo gedimus. Pokyčiai iš schemos įėjimų į išėjimus perduodami schemos keliais. Vėlinimo gedimų tikrinimui tikslinga pokyčius iš įėjimų į išėjimus perduoti kuo ilgesniais ir kuo įvairesniais keliais. Jeigu pokytis įėjime įtakoja pokytį išėjime, tai galima daryti prielaidą, kad gali būti patikrinti kai kurie vėlinimo gedimai.

Funkcinio testo teisingumui įvertinti svarbu nustatyti, koks įėjimo pokytis kokius išėjimus įtakoja.Funkcinis testas remiasi schemoje vykdoma funkcija, kuri gali būti realizuota daugeliu būdu. Schemos galimi gedimai priklauso nuo schemos realizacijos. Praktikoje apsiribojama schemos konkrečios  realizacijos galimais gedimais ir testas sudaromas tik vienos realizacijos gedimams. Testą galima sudaryti tik turint konkrečią schemos realizaciją. Tuo tarpu funkcinis testas nepriristas prie konkrečios realizacijos. Tokiu atveju funkcinis testas turi tikrinti visas galimas realizacijas kas yra žymiai sudėtingesnė problema.

Iki šiol mažai tyrinėta, kaip gauti funkcinį testą vėlinimo gedimams ir kaip funkcinis testas tikrina konkrečios realizacijos vėlinimo gedimus. Be to mums  nėra žinoma pasiūlytų funkcinio testo kokybės vertinimo kriterijų vėlinimo gedimų atžvilgiu. Naudojami ir ištirti funkcinio testo kokybės kriterijai konstantinių gedimų atžvilgiu nėra labiausiai tinkami vėlinimo gedimams, nes vėlinimams reikia vertinti testinių rinkinių poras. Šis darbas skirtas minėtoms problemoms spręsti. [9]

Ventiliniame schemos lygmenyje daug metų yra naudojamas įprastas vienetinio konstantinio gedimo modelis. Nepaisant šio modelio efektyvumo, išaugusios integracijos schemose reikalingi kiti modeliai, atsirado nemažai naujų defektų, kurių efektų negali apimti vienetinio konstantinio gedimo modelis. Dauguma šių defektų nekeičia schemos funkcionalumo, tačiau keičia schemos vėlinimo laikus. Šie gedimai vadinami vėlinimo gedimais. Vėlinimo gedimų tikrinimui plačiausiai yra naudojami 2 modeliai: perėjimo vėlinimo modelis (*angl.Transition fault*) ir kelio vėlinimo modelis(*angl.Path delay fault*). [9]

Perėjimo vėlinimo gedimo (*angl.Transition fault*) modelis modeliuoja didelio vėlinimo gedimą atskirame schemos taške.

Kelio vėlinimo (*angl.Path delay fault*) modelis modeliuoja schemos kelio, prasidedančio schemos įėjime ir pasibaigiančio schemos išėjime, vėlinimą. Kombinacinės sinchroninės schemos vėlinimo gedimų testavimu vadinamas tos schemos tikrinimas, ar ji gali perduoti signalo pasikeitimą į išėjimą per laiką, neviršijantį laiko tarpo, lygaus intervalui tarp dviejų gretimų sinchroninio signalo impulsų, kurie siunčiami realiai veikiančiai schemai.

Jei toks testas duoda neigiamą atsakymą, tai reiškia, kad tikrinamoji schema turi kelio vėlinimo gedimą. Reikia pažymėti, kad laiko intervalai tarp dviejų gretimų sinchroninio signalo impulsų yra vienodi, nesvarbu, kuris TKS schemos kelias yra tikrinamas.

Kelio vėlinimo gedimas gali atsirasti bet kurioje elektroninėje schemoje. Tai gali sukelti net ir vienas tokios schemos elementas, esantis tikrinamajame kelyje, kurio realus vėlinimas viršija nurodytą vėlinimo trukmę jo specifikacijoje.[1]

Tačiau gali atsitikti ir taip, kad schema veiks korektiškai, nors tikrinamąjį kelią sudarys elementai, kurių vėlinimo trukmės viršys nurodytas specifikacijoje.

## 2.5 Vėlinimo gedimų testavimo modeliai

Vėlinimo gedimai – jau plačiau aptarti, atsirandantys, kai signalo pokytis nespėja pasiekti išėjimo reikšmės lygio per visą sinchronizavimo laiką.

Funkciniai testai, pritaikyti naudojamo schemos prototipo keliuose, kad tikrinti vėlinimo gedimus pagal pokyčius išėjimuose. Tad, nuo šiol funkciniu vėlinimo testu vadinsime testą atliktą nesiremiant tirtos schemos struktūra.

**Apibendrinimas:** vėlinimo gedimų testo metodologija - vėlinimo gedimai tikrinami rinkinių pora. Rinkinių poros signalų reikšmės apsprendžia, kurie įėjimai keičiasi ir kurie išlieka pastovūs. Gali keistis vienas ar daugiau įėjimų.

Galimi vėlinimo gedimų testavvimo modeliai yra:

- Vėlinimo gedimų testavimas funkciniame(ventiliniame) lygmenyje
- Segmentų vėlinimo modelis
- Perėjimo vėlinimo gedimų testavimas
- Kelio vėlinimo gedimų testavima

Šiuos metodus plačiau aptarsime šiame skyriuje.

## 2.6 Vėlinimo gedimų testavimas funkciniame(ventiliniame) lygmenyje

Vienas iš plačiai naudojamų vėlinimo gedimų metodų yra generuojantis vienetą arba nulį*(angl. stack-at).* Šio metodo pagalba modeliuojamos klaidos, trumpi sujungimai, laidų nebuvimas, nutrūkimai, įtakojantys kuriamos schemos išėjime suformuotą loginį vienetą arba nulį. [2]

Tokio gedimo testavimo etapus sudaro:

- Testinių vektorių sugeneravimas.
- Testinių vektorių sukelto klaidos poveikio perdavimas testiniais duomenimis pradiniams schemoms įėjimams.[2]

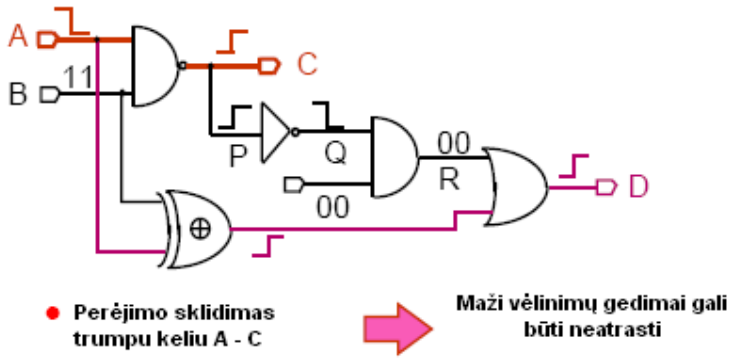
## 2.7 *Perjungimo vėlinimo gedimų modelis*

Šis vėlinimo gedimo modelis *(angl. Transition delay fault)* , naudojamas, kai klaidos poveikis pastebimas, kuomet į įėjimą paduotas aukštas arba žemas signalo frontas nepasiekia išėjimo per nustatytą laiko tarpą. Šis vėlinimo gedimų modelis išėjime generuoja loginį 0 arba 1 (*kaip ir angl. Stack-at*).[2-4]

Taip pat pagal šį modelį tikrinami didelius vėlinimo gedimai, sukoncentruoti viename loginiame mazge(ar aptinkami ilguose keliuose), tokiame, jog bet koks signalo perjungimas, praeinantis pro šį mazgą, užlaikomas tam tikrą laiko periodą.

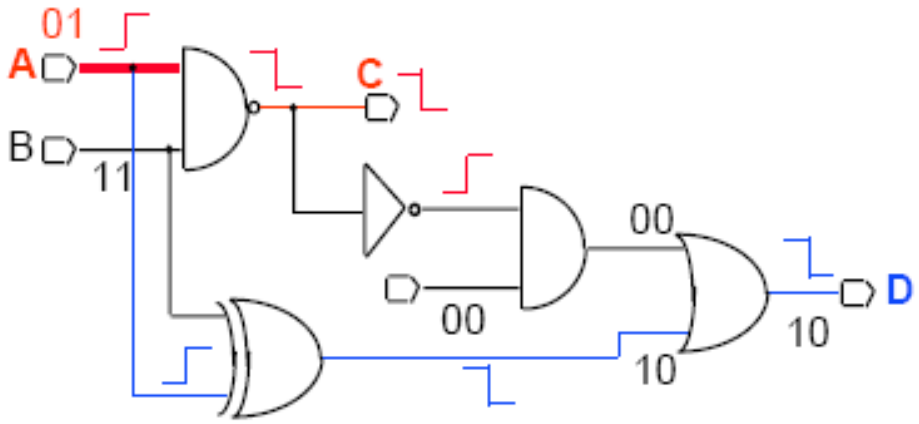
Perėjimo vėlinimo gedimų modelio testavimą galime skirti į dvi rūšis:

- *STR (slow-to-rise) – kylantis signalo frontas.*
  - *Tiriamos dvi testinių rinkinių poros ($V_1$,$V_2$), linijoje(kelyje) K, jei :*
    - *$V_1$ nustato K->0*
    - *$V_2$ tikrina K stuck-at-0*
- *STF (slow-to-fall) – krintantis signalo frontas(pav.3)[3]*
  - *Tiriamos dvi testinių rinkinių poros ($V_1$,$V_2$), linijoje(kelyje) K, jei :*
    - *$V_1$ nustato K->1*
    - *$V_2$ tikrina K stuck-at-1*




*LENTELĖ 3 STF (SLOW-TO-FALL) – SCHEMOS PAVYZDYS*

Pavyzdys: (Lentelė.4 ) STF(Slow-to-fall) vėlinimo gedimai A: $V_1$ : perjungia A -> 1; $V_2$ : testuojama “Stuck-at” gedimas mazge A;



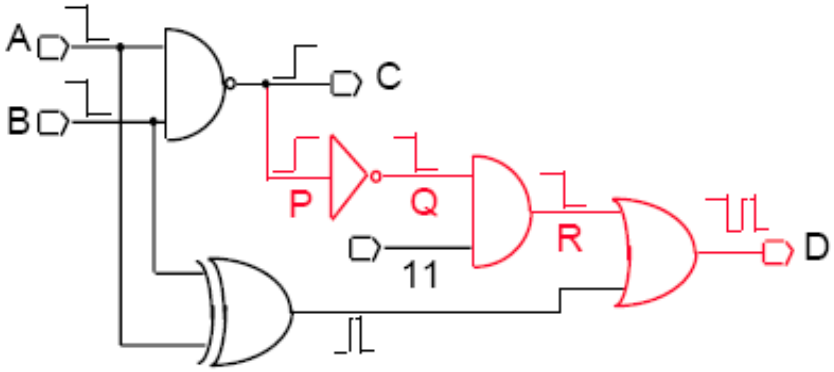

*LENTELĖ 4 PERJUNGIMO VĖLINIMO GEDIMO TIKRINIMO MODELIS*

Panagrinėkime antrą pavyzdį(Pav.4): linijoje A dideli gedimai bus aptikti C išėjime, jei vėlinimai kelyje A-C bus viršys schemos reikalavimus; nedideli gedimai linijoje A gali būti nepastebėti kelyje A-C, tokie gedimai turėtų būti tikrinami ilgesniame schemos kelyje A-D.

Kad aptikti šiuos gedimus būtina generuoti keletą testinių vektorių: vieną, kad nustatytų reikiamą įėjimo formą, o kitas tam, kad sugeneruotų perėjimą įėjime ir perėjimo sklidimą į išėjimą. [3]

## 2.8 *Segmento vėlinimo gedimų modelis*

Šis modelis remiasi testuojamų nedideliu viso schemos kelio segmentu, dalimi. Schemiškai pavyzdys pateikiamas pav. 5.



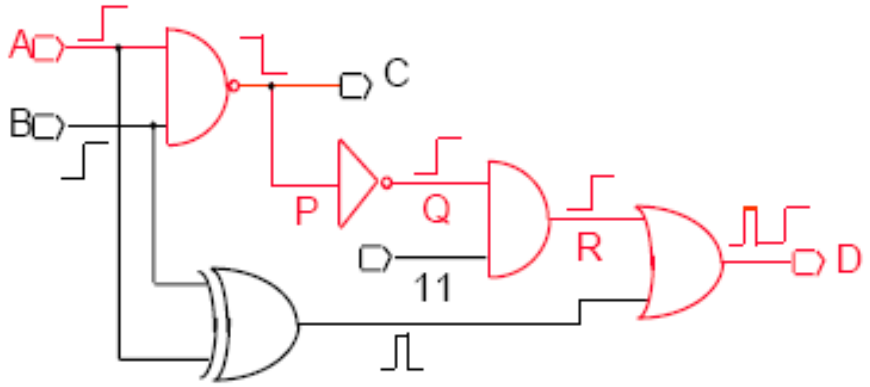

*LENTELĖ 5 KELIO SEGMENTO  VĖLINIMO GEDIMO TIKRINIMO MODELIS*

Kelias A-P-Q-R-D yra Netikrinamasis krintančiam perėjimui, pradėtam A, bet jo dalies dalyje, būtent padalina P-Q-R-D į dalis, yra tikrinamasis.

## 2.9 Kelio vėlinimo gedimų modelis

Kelio vėlinimo gedimas *(angl. Path delay fault model)* gali atsirasti bet kurioje schemoje, tokį gedimą sukelti gali net tik vienas kuris schemos elementas. Aišku, tas elementas turi būti tikrinamame schemos kelyje (grandinėje) ir jo vėlinimo laikas turi viršyti jo specifikacijoje nurodytą vėlinimo laiką. Kombinacinė schema neturi vėlinimo gedimų tada ir tik tada, kai jų nerandama schemos keliuose.[2-3]

Jei grandinėje esančių elemento kiekvieno atskirai vėlinimo trukmės neviršys nurodytų jų specifikacijoje, o tik pasieks maksimumus, tuomet visgi negalima teigti, kad schema veikia be vėlinimo gedimų. Šiuo atveju esminis bus suminis vėlinimo laikas visų grandinėje esančių elementų.




*LENTELĖ 6 KELIO VĖLINIMO GEDIMO TIKRINIMO MODELIS*

Suminiam (A-P-Q-R-D) (pav.6 ) vėlinimo gedimų laikui viršijus siganlo periodą, taps aišku, kad tiriamoje schemoje yra vėlinimo gedimas, nors ir elementai teisingi.[2] Tokiu atveju galima išskirti dvi kelio vėlinimo gedimų priežastis:

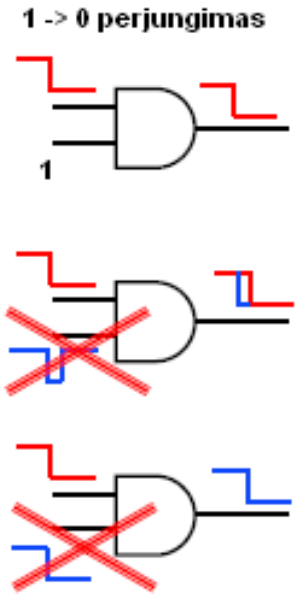
Kai yra nors vienas elementas, kurio vėlinimas didesnis nei nurodytą jo specifikacijoje ir tai sukelia vėlinimo gedimą tiriamajame schemos kelyje. Kai suminis tiriamojo schemos kelio elementų vėlinimo laikas viršija sinchroninio signalo periodą ir tai sukelia vėlinimo gedimą kelyje.

Kitaip galime pasakyti [12], kad tarkim fizinis kelio *P* sujungimas tarp pradinio kelio mazgo (PI) iki galutinio mazgo (PO). Kylantis (krentantis) kelio lygmuo *Pr (Pf)* apibūdinamas kaip atsakas į kylantį (krentantį) perjungimo lygmenį schemos įėjime PI. Kelio svoris (*angl. Path Length)* apibūdinamas kaip ventilių skaičius pasirinktame schemos kelyje. *Segmentas S* dalis kelio *P*. Segmentas gali prasidėti ir baigtis bet kuriame duoto kelio P taške. Kaip minėjome, kelio vėlinimų gedimo testavimo modelis naudojamas atrasti ir ištirti nedidelius vėlinimo gedimus pasirinktame schemos kelyje. Schemos veikimo proceso metu klaidos viename schemos kelyje gali būti verifikuotos kaip vėlinimo gedimas, tačiau tokie sutrikimai nereiškia , kad schema veikia neteisingai [12].

Kadangi papildomi vėlinimo gedimai įėjime-išėjime yra nedideli, perėjimo vėlinimo gedimų (angl. Transition fault) modelio naudojimas yra netinkamas, kadangi tokių gedimų jo pagalba nepatiksime. O naudodami kelio vėlinimo gedimo modelį galime aptikti ir nedidelius vėlinimo gedimus pasirinktuose (atrinkuose) schemos keliuose ar jų dalyse. Šį Kelio vėlinimo gedimų modelio bruožą galime laikyti vienu svarbiausiu privalumų prieš perėjimo gedimų modelio taikymą. Tačiau Kelio vėlinimo gedimų modelio taikyme sudėtingą situaciją sukelia reikalingų kelių pasirinkimas. Kadangi schemas sudaro didelis kiekis schemos kelių, sunku atrinkti reikiamus, būtent kritinius kelius. Šį klausimą nagrinėsime šio darbo sekančiuose skyriuose.

Išskiriama keletas kelio vėlinimo gedimus įtakojančių veiksnių bei variacijų. Galime išskirti dvi kelių pokyčių įtakas:

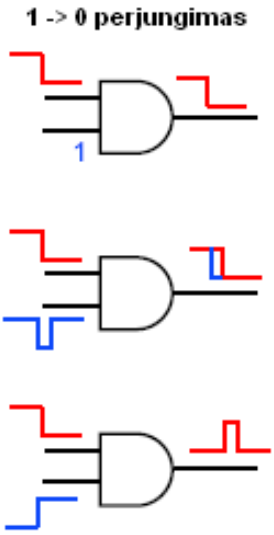
**- Tiesioginė(angl.robust) įtaka** – vienas įėjimo pasikeitimo panaikinimas įtakoja, jog bus panaikintas ir pasikeitimas išėjime; tiksliau vieno gedimo panaikinimas įėjime nuo kitų kelyje esančių vėlinimų nepriklauso ir nekeičia išėjimo būsenos.(Pav.7)




*LENTELĖ 7 TIESIOGINĖ ĮTAKA TESTAVIMUI*

Tokie kelių tipai gali būti testuojami nepriklausomai nuo to ar visus būtina testuoti. T.y. jei visi schemos keliai bus testuojamieji, tuomet bus nebereikalinga atlikti jokius papildomus testavimus.

– **Netiesioginė** (angl. nonrobust) įtaka – vienas įėjimo pasikeitimo panaikinimas, įtakoja naujo pasikeitimo atsiradimą išėjime. Tai reiškia, kad įėjimo pokytis blokavo kito pokyčio įtaką į išėjimą ir pokytį panaikinus išnyko šis blokavimas, o išėjime atsirado naujas pokytis.(Pav.8)




*LENTELĖ 8 NETIESIOGINĖ ĮTAKA TESTAVIMUI*

**Trūkumai:** jei tiriamojoje schemoje atlikus testavimą, randame vėlinimo gedimą, negalima nustatyti kurioje vietoje ir kas sukėlė gedimą. Kadangi gali atsitikti taip, kad gedimą sukėlė ne vienas kažkuris elementas, o keletas (jų suminis vėlinimas). Arba gedimą gali įtakoti bet krui schemos grandinė, nes kaip bebūtų visos grandinės susijungia.[3]

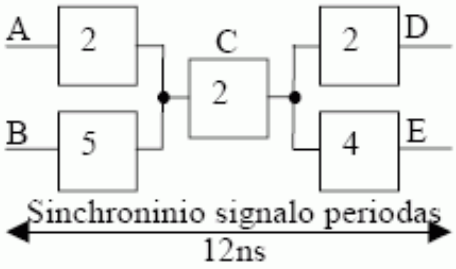
Kartais kelio vėlinimo gedimas gali būti ir neaptiktas. Kai schemoje egzistuoja loginė klaida: kokio nors elemento išėjime yra pastovus loginis 0 arba loginis 1.

**Privalumai:** naudojantis vėlinimo gedimų testavimo modeliu galime atrasti ir loginių elementų gedimus, galime atrasti labai mažus vėlinimo gedimus. Tam reikalinga sąlyga, kad loginis elemento gedimas testuojamosios schemos išėjimo vertę paveiktų, kad ji taptų priešinga tai vertei, kai schema yra teisinga.[3] Testuojant vieną konkretų loginį elementą galime aptikti vėlinimo gedimą, jei jo vėlinimas viršija specifikacijoje nurodytą, tačiau šis metodas turi trukūmą – didelė testavimo rezultatų tesingumo priklausomybė nuo kitų schemos loginių elementų.

## 2.10 Tiriamųjų kelių idetifikavimas

Jei schemoje yra didelis skaičius kelių, jų visų tirti nebūtina – užtenka tam tikro skaičiaus kelių tyrimui, kad būtų galima nustatyti ar schemoje yra vėlinimo gedimų.

Tam reiktų atrasti taip vadinama schemos vėlinimo atsargą, kurią galima rasti iš išėjimo signalo ir paskutinio keliu sklidusio išėjimo signalo pokycio skirtumo (skaičiuojamos vėlinimo laiko vertės).” Kombinacinė schema neturi vėlinimo gedimų tuomet ir tik tuomet, kai joje nėra kelių vėlinimo gedimų. Tačiau loginių elementų faktinių vėlinimo gedimų analizė rodo, kad pakanka testuoti nedidelį visų kombinacinės schemos kelių poaibį tam, kad su patenkinama tikimybe būtų galima pasakyti, ar testuojamojoje schemoje yra kelių vėlinimo gedimų, ar ne.”[1]




| | Kelias | Sklidimo laikas | Klaidos aptinkamumas |
|---|---|---|---|
| Be klaidų | ACD | 6 | — |
| | ACE | 8 | — |
| | BCD | 9 | — |
| | BCE | 11 | — |
| Klaida elemente C: +2 | ACD | 8 | Ne |
| | ACE | 10 | Ne |
| | BCD | 11 | Ne |
| | BCE | 13 | Taip |
| Klaida elemente C: +7 | ACD | 13 | Taip |
| | ACE | 15 | Taip |
| | BCD | 16 | Taip |
| | BCE | 18 | Taip |

*LENTELĖ 9 ELEMENTO VĖLINIMO POKYČIO ĮTAKA KLAIDŲ APTINKAMUMUI [1]*

Tarkim, surandame ir pasirenkame kiekvienam schemos elementui per jį einantį kelią, kurio vėlinimo atsarga mažiausia. Tuomet testuojame šiuos kelius, jei vėlinimo gedimų nerasta, su didele tikimybė galime tvirtinti, kad schemoje vėlinimo gedimų nėra.

“ Patyrinėkime kombinacinę schemą, pateiktą 8 pav.[1] Per C elementą eina keturi keliai. Jei elementas C turi vėlinimo gedimą, C elemento vėlinimo atsargos dydis

turi įtakos kelio BCE vėlinimo atsargai. Šio kelio vėlinimo atsargos dydis, be kelio vėlinimo gedimo, yra 12 – (5 + 2 + 4) = +1. Vėlinimo gedimas elemente C neįtakos kelio BCE vėlinimui tol, kol elemento vėlinimo atsarga neviršija 3.

Tokiu atveju tik kelias BCE turi neigiamą kelio vėlinimo atsargą, todėl šį gedimą galima aptikti tik testiniais poveikiais, kurie tikrina kelią BCE.

Kelių, turinčių vėlinimo gedimų, skaičius didėja didėjant neigiamai elemento C vėlinimo atsargai. Bendruoju atveju daug lengviau aptikti gedimą, kai neigiamos vėlinimo atsargos modulis kuo didesnis. Dėl šios priežasties labai nepatogu naudoti vėlinimo gedimų modelį, kuriame vertinami tik atskirų elementų vėlinimo gedimai.” [1].

Kiekvienam schemos elementui surandus per jį einantį kelią, kurio vėlinimo atsarga yra mažiausia, ir testavus tik šiuos kelius, ir nustačius, kad juose nėra kelių vėlinimo gedimų, su nemaža tikimybe galima teigti, kad schemoje kelių vėlinimo gedimų nėra. Šiai tikimybei padidinti galima imti daugiau kelių, einančių per tą patį elementą, nors jo vėlinimo atsarga ir nėra mažiausia.

Kelio vėlinimo atsargą galime laikyti  pagrindiniu kriterijum, kuriuo remdamiesi galime pasirinkti schemos kelios vėlinimo gedimams tirti.

Testinius rinkinius, naudojamus daugelyje testavimo metodų, galime sugeneruoti naudojant SCAN grandinę arba naudojant automatines testinių rinkinių programas(ATPG).

## 2.11 Netestuojamų kelių identifikavimas

Pateiksime metodą, kuriuo galima nustatyti, kurie keliai iš visos schemą sudarančios kelių aibės yra testuoti vėlinimo gedimų prasme. Tam sudaroma visų kelių, aprėpiančių tyrinėjamą kombinacinę schemą, aibė KA. Vėliau ciklišk ai atliekami žingsniai aprašyti toliau. Šie žingsniai vykdomi tol, kol sudarytoje kelių aibėje nebelieka nė vieno kelio, arba yra tenkinama kokia nors kita darbo nutraukimo sąlyga:

1. Generuojamas testinis poveikis ir siunčiamas į testuojamosios kombinacinės schemos įėjimus;
2. Pagal suadrytas teisingumo lenteles apskaičiuojamos keliais sklindančių signalų vertės;

3. Kelias, kuriuo sklinda signalas, turintis P simbolį, pažymimas kaip netestuotinas ir išmetamas iš kelių aibės KA.[1-2-3]

**Testinis poveikis išsaugomas, kaip tikrinantis šio kelio vėlinimo gedimą.**

Šios procedūros trukmė yra tiesiškai proporcinga loginių elementų ir kelių tarp jų skaičiui, nes kiekvienu keliu sklindančių signalų vertės skaičiuojamos tik vieną kartą.

## 2.12 Testo sudarymo kokybės įvertinimas

Didžiausia testavimo problema – surasti optimaliausią testinių rinkinių seką esant tam tikriems reikalavimams:

- visu (idealiu atveju) gedimų aptikimą pasirinktam gedimu modeliui;
- nesudėtingas testinių rinkinių generavimas bei saugojimas;
- testinių rinkinių kompaktiškumas.

Visas elektronines schemas galima suskirstyti i dvi grupes:

- schemos be atminties (kombinacines schemos);
- schemos su atmintimi (nuoseklios schemos). [2]

Pirmosios grupės schemų testavimas ne toks sudėtingas, kaip antrosios, nes galima paprasčiausiai perrinkti N visus galimus variantus. Tam reikes atlikti 2 (n – schemos išvadų skaičius) testavimo žingsnių. Tačiau nuosekliosios schemos testavimo klausimas iki šiol neišspręstas, nes atskiri gedimai gali buti aptinkami tik nustačius schemą i tam tikrą buseną, o tik tada paduoti testinį rinkinį. [2-6-7]

Pagrindiniai testinių sekuų generavimo principai yra determinuotasis ir atsitiktinis. Determinuotuoju principu gauta testinė seka visada turi atgalinį ryšį su apsibrėžtu gedimų modeliu: pirmame žingsnyje sudaromi visi galimi gedimai, kuriuos galime gauti pasirinktu gedimų modelio pagalba. Antrame žingsnyje paduodamas testinis rinkinys ir išbraukiami visi gedimai, kuriuos galime aptikti šio rinkinio pagalba. Antrasis žingsnis kartojamas tol, kol gedimų aibė tampa tuščia. Specialusis determinuotosios testinės sekos atvejis – kai reakcijos gaunamos panaudojus visus įmanomus testinius rinkinius. Tokiu būdu gautas testas yra geriausias, bet jo panaudojimas yra nepraktiskas, nes pareikalautų labai daug resursu.[2]

## 2.13 *Testinių rinkinių įtaka testo kokybei*

Kaip kalbėjome anksčiau funkcinio testo kokybės vertinimui svarbu nustatyti, koks įėjimo pokytis kokius išėjimus įtakoja. Tikslinga nagrinėti atskirų įėjimo pokyčių panaikinimo įtaką išėjimo pokyčiams.

Pavyzdžiui testavimo rinkiniai sudaromi iš poros vektorių {V1, V2} kur V1 inicijuojamas kaip kelio mazgas, o V2 privalo turėti atitinkamą perėjimą išėjimo mazge ir perduoti šį poveikį schemos keliais.

Patekiame literatūroje nagrinėtą pavyzdį [20]:

„Turime rinkinių porą {1010,0111}, kur keičiasi pirmo, antro ir ketvirto įėjimų reikšmės. Panaikinant pokytį pirmam įėjime gauname rinkinių porą {1010,1111}, panaikinant pokytį antram įėjime gauname rinkinių porą {1010,0011}, o panaikinant pokytį ketvirtam įėjime gauname rinkinių porą {1010,0110}. Įėjimo pokyčio panaikinimas gali sąlygoti ir kai kurių pokyčių išnykimą išėjimuose. Jei taip atsitinka, tai galima tvirtinti, kad įėjimas, kuriam buvo panaikintas pokytis įtakoja išėjimą, kuriam pokytis išnyko. Įtaka pasireiškia schemos keliais. Čia galima išskirti keturis atvejus: įėjimo pokyčio 0→1 panaikinimas sąlygoja išėjimo pokyčio 0→1 išnykimą, įėjimo pokyčio 0→1 panaikinimas sąlygoja išėjimo pokyčio 1→0 išnykimą, įėjimo pokyčio 1→0 panaikinimas sąlygoja išėjimo pokyčio 0→1 išnykimą, įėjimo pokyčio 1→0 panaikinimas sąlygoja išėjimo pokyčio 1→0 išnykimą. Kadangi vieno įėjimo pokyčio panaikinimas tiesiogiai sąlygoja pokyčio išnykimą išėjime, tai tokią įtaką pagal analogiją su schemos kelių tikrinimu vadinsime robust įtaka. Gali būti ir netiesioginis įtakojimas, kai įėjimo pokyčio panaikinimas iššaukia papildomo išėjimo pokyčio atsiradimą. Tai reiškia, kad įėjimo pokytis blokavo kito pokyčio įtaką į išėjimą ir pokytį panaikinus išnyko šis blokavimas, o išėjime atsirado naujas pokytis. Čia taip pat galima išskirti keturis atvejus: įėjimo pokyčio 0→1 panaikinimas sąlygoja papildomo išėjimo pokyčio 0→1 atsiradimą, įėjimo pokyčio 0→1 panaikinimas sąlygoja papildomo išėjimo pokyčio 1→0 atsiradimą, įėjimo pokyčio 1→0 panaikinimas sąlygoja papildomo išėjimo pokyčio 0→1 atsiradimą, įėjimo pokyčio 1→0 panaikinimas sąlygoja papildomo išėjimo pokyčio 1→0 atsiradima. [20] „

„Bendru atveju turime du įėjimo poveikius $P^1=<p_1^1,p_2^1,p_3^1,\ldots,p_i^1,\ldots,p_n^1>$ ir $P^2=<p_1^2,p_2^2,p_3^2,\ldots,p_i^2,\ldots,p_n^2>$ ir dvi reakcijas į tuos poveikius $R^1=<r_1^1,r_2^1,r_3^1,\ldots,r_j^1,\ldots,r_m^1>$ ir $R^2=<r_1^2,r_2^2,r_3^2,\ldots,r_j^2,\ldots,r_m^2>$.

Įėjimų įtaką į išėjimus galima atvaizduoti matrica $||X||_{2n \times 4m}$. Matricos elementas $x_{2i-1,4j-3}=1$, jei $p_i^1=0$, $p_i^2=1$, $r_j^1=0$, $r_j^2=1$, ir prilyginus $p_i^2=0$ gauname $r_j^2=0$. Matricos elementas $x_{2i-1,4j-2}=1$, jei $p_i^1=0$, $p_i^2=1$, $r_j^1=1$, $r_j^2=0$, ir prilyginus $p_i^2=0$ gauname $r_j^2=1$. Matricos elementas $x_{2i-1,4j-1}=1$, jei $p_i^1=0$, $p_i^2=1$, $r_j^1=0$, $r_j^2=0$, ir prilyginus $p_i^2=0$ gauname $r_j^2=1$. Matricos elementas $x_{2i-1,4j}=1$, jei $p_i^1=0$, $p_i^2=1$, $r_j^1=1$, $r_j^2=1$, ir prilyginus $p_i^2=0$ gauname $r_j^2=0$. Matricos elementas $x_{2i,4j-3}=1$, jei $p_i^1=1$, $p_i^2=0$, $r_j^1=0$, $r_j^2=1$, ir prilyginus $p_i^2=1$ gauname $r_j^2=0$. Matricos elementas $x_{2i,4j-2}=1$, jei $p_i^1=1$, $p_i^2=0$, $r_j^1=1$, $r_j^2=0$, ir prilyginus $p_i^2=1$ gauname $r_j^2=1$. Matricos elementas $x_{2i,4j-1}=1$, jei $p_i^1=1$, $p_i^2=0$, $r_j^1=0$, $r_j^2=0$, ir prilyginus $p_i^2=1$ gauname $r_j^2=1$. Matricos elementas $x_{2i,4j}=1$, jei $p_i^1=1$, $p_i^2=0$, $r_j^1=1$, $r_j^2=1$, ir prilyginus $p_i^2=1$ gauname $r_j^2=0$. Matome, kad visais atvejais $p_i^1$ ir $p_i^2$ reikšmės yra priešingos ir keičiant $p_i^2$ reikšmę turi keistis ir $r_j^2$ reikšmė. Įėjimą i matricoje X atitinka dvi eilutės 2(i-1) ir 2i. Eilutė 2(i-1) atitinka įėjimo pokytį 0→1, o eilutė 2i atitinka įėjimo pokytį 1→0. Išėjimą j matricoje X atitinka keturi stulpeliai. Stulpelis 4(j-3) rodo robust įtaką išėjimo pokyčiui 0→1, stulpelis 4(j-2) rodo robust įtaką išėjimo pokyčiui 1→0, stulpelis 4(j-1) rodo nonrobust įtaką išėjimo pokyčiui 0→1, stulpelis 4j rodo nonrobust įtaką išėjimo pokyčiui 1→0. „

Paimkime paprastą dviejų įėjimų AND ventilį ir poveikių porą $P^1=<0,0>$, $P^2=<1,1>$ su reakcijomis $R^1=<0>$, $R^2=<1>$. Abiejuose įėjimuose ir išėjime yra pokyčiai $p_1(0\rightarrow1)$, $p_2(0\rightarrow1)$, $r_1(0\rightarrow1)$. Įėjimų įtaką į išėjimą vaizduos matrica $||X||_{4x4}$. Panaikinus pokytį pirmam įėjime $p_1(0\rightarrow0)$ išnyksta pokytis ir išėjime $r_1(0\rightarrow0)$ ir atžymimas matricos elementas $x_{1,1}=1$. Analogiškai panaikinus pokytį antram įėjime $p_2(0\rightarrow0)$ išnyksta pokytis ir išėjime $r_1(0\rightarrow0)$ ir atžymimas matricos elementas $x_{3,1}=1$. Poveikių porai $P^1=<1,1>$, $P^2=<0,1>$ su reakcijomis $R^1=<1>$, $R^2=<0>$ ir esant pokyčiams $p_1(1\rightarrow0)$, $p_2(1\rightarrow1)$, $r_1(1\rightarrow0)$ atžymimas matricos elementas $x_{2,2}=1$, o poveikių porai $P^1=<1,1>$, $P^2=<1,0>$ su reakcijomis $R^1=<1>$, $R^2=<0>$ ir esant pokyčiams $p_1(1\rightarrow1)$, $p_2(1\rightarrow0)$, $r_1(1\rightarrow0)$ atžymimas matricos elementas $x_{4,2}=1$. Poveikių pora $P^1=<1,1>$, $P^2=<0,0>$ su reakcijomis $R^1=<1>$, $R^2=<0>$ esant pokyčiams $p_1(1\rightarrow0)$, $p_2(1\rightarrow0)$, $r_1(1\rightarrow0)$ neatžymi jokio matricos elemento, nes panaikinus pokytį bet kuriam įėjime išėjime pokytis neišnyksta. Poveikių poros $P^1=<0,1>$, $P^2=<1,0>$ su reakcijomis $R^1=<0>$, $R^2=<0>$ ir $P^1=<1,0>$, $P^2=<0,1>$ su reakcijomis $R^1=<0>$, $R^2=<0>$, esant pokyčiams $p_1(0\rightarrow1)$, $p_2(1\rightarrow0)$, $r_1(0\rightarrow0)$ ir $p_1(1\rightarrow0)$, $p_2(0\rightarrow1)$, $r_1(0\rightarrow0)$ nekeičia išėjimo reikšmės tačiau panaikinus pokytį $p_2(1\rightarrow0)$ arba $p_1(1\rightarrow0)$, t. y. padavus signalus (1→1) išėjime atsiranda pokytis $r_1(0\rightarrow1)$. Tai įėjimo įtaka į išėjimą netiesioginio tipo ir matricoje X atžymimi elementai $x_{4,3}=1$ ir $x_{2,3}=1$. Rezultatai visoms galimoms poveikių poroms

parodyti Pav.10. Kiekvienai poveikių porai apatinėse eilutėse parodyta kokie pokyčių panaikinimai įtakoja išėjimą ir kokie atžymimi matricos elementai. Matome, kad penkios poveikių poros nenustato įėjimų įtakos į išėjimus. Poveikių pora 3 nustato tą pačią įtaką tarp įėjimų ir išėjimų kaip ir poveikių poros 6 ir 9 kartu. Pav.11 parodyta matrica X. [20]„

| No. | Poveikų pora | $P_1$ | $P_2$ | $R_1$ | X |
|---|---|---|---|---|---|
| 1 | $P^1$=<0,0>, $P^2$=<0,1>,$R^1$=<0>, $R^2$=<0> | 0→0 | 0→1 | 0→0 | |
| 2 | $P^1$=<0,0>, $P^2$=<1,0>,$R^1$=<0>, $R^2$=<0> | 0→1 | 0→0 | 0→0 | |
| 3 | $P^1$=<0,0>, $P^2$=<1,1>,$R^1$=<0>, $R^2$=<1> | 0→1 | 0→1 | 0→1 | |
| | | 0→0 | 0→1 | 0→0 | $X_{1,1}$=1 |
| | | 0→1 | 0→0 | 0→0 | $X_{3,1}$=1 |
| 4 | $P^1$=<0,1>, $P^2$=<0,0>,$R^1$=<0>, $R^2$=<0> | 0→0 | 1→0 | 0→0 | |
| 5 | $P^1$=<0,1>, $P^2$=<1,0>,$R^1$=<0>, $R^2$=<0> | 0→1 | 1→0 | 0→0 | |
| | | 0→1 | 1→1 | 0→1 | $X_{4,3}$=1 |
| 6 | $P^1$=<0,1>, $P^2$=<1,1>,$R^1$=<0>, $R^2$=<1> | 0→1 | 1→1 | 0→1 | |
| | | 0→0 | 1→1 | 0→0 | $X_{1,1}$=1 |
| 7 | $P^1$=<1,0>, $P^2$=<0,0>,$R^1$=<0>, $R^2$=<0> | 1→0 | 0→0 | 0→0 | |
| 8 | $P^1$=<1,0>, $P^2$=<0,1>,$R^1$=<0>, $R^2$=<0> | 1→0 | 0→1 | 0→0 | |
| | | 1→1 | 0→1 | 0→1 | $X_{2,3}$=1 |
| 9 | $P^1$=<1,0>, $P^2$=<1,1>,$R^1$=<0>, $R^2$=<1> | 1→1 | 0→1 | 0→1 | |
| | | 1→1 | 0→0 | 0→0 | $X_{3,1}$=1 |
| 10 | $P^1$=<1,1>, $P^2$=<0,0>,$R^1$=<1>, $R^2$=<0> | 1→0 | 1→0 | 1→0 | |
| 11 | $P^1$=<1,1>, $P^2$=<0,1>,$R^1$=<1>, $R^2$=<0> | 1→0 | 1→1 | 1→0 | |
| | | 1→1 | 1→1 | 1→1 | $X_{2,2}$=1 |
| 12 | $P^1$=<1,1>, $P^2$=<1,0>,$R^1$=<1>, $R^2$=<0> | 1→1 | 1→0 | 1→0 | |
| | | 1→1 | 1→1 | 1→1 | $X_{4,2}$=1 |

*LENTELĖ 10 POVEIKIO PORŲ ANALIZĖ [20]*

| Įėjimai | Išėjimas | | | |
|---|---|---|---|---|
| | Robust | | Nonrobust | |
| Pirmas įėjimas | 1 | 0 | 0 | 0 |
| | 0 | 1 | 1 | 0 |
| Antras įėjimas | 1 | 0 | 0 | 0 |
| | 0 | 1 | 1 | 0 |

*LENTELĖ 11 MATRICOS X SUDĖTIS [20]*

Šiam pavyzdžiui remdamiesi straipsnio medžiaga apibrėžiamas ir algoritmas, kuris gali būti naudojamas eksperimentiniam tyrimui.

„Duotiems įėjimo poveikiams $P^1$ ir $P^2$ suformuoja matricą $||X||_{2n \times 4m}$. Pradžioje visi matricos elementai prilyginami nuliui. Nagrinėjami visi įėjimai ir kintamasis d prilyginamas invertuotai pirmo poveikio reikšmei (eil. 3). Besiskiriantiems įėjimo signalo reikšmėms (eil. 4) trečiam poveikyje $P^3$ įėjimo signalo reikšmė invertuojama,

paskaičiuojama išėjimo reakcija $R^3$ ir išnagrinėjus atstatoma (eil. 12). Toliau nagrinėjami visi išėjimai (eil. 5). Jeigu keičiasi išėjimo signalo reikšmė ( eil. 6) ir to išėjimo poveikių $P^1$ ir $P^2$reakcijos taip skirtingos, matricoje X užrašomi atitinkami vienetukai. Kintamųjų d ir c naudojimas leidžia kompaktiškai indeksuoti matricos X elementus. „

```
1.  X=||x_{i,j}:=0||_{2nx4m};
2.  R^1:=f(P^1); R^2:=f(P^2);
3.  DO i: =1,2,3,...,n; P^3: =P^2; d:=1- p^1_i;
4.          IF (p^1_i≠p^2_i) THEN p^3_i:=1- p^2_i; R^3=f(P^3);
5.                  DO j: =1,2,3,...,m; c:=3- r^1_j;
6.                          IF (r^3_j≠r^2_j) THEN
7.                                  IF(r^2_j≠r^1_j) THEN x_{2i-d,4j-c}:=1;
8.                                          ELSE x_{2i-d,4j-c+2}:=1;
9.                                  ENDIF;
10.                         ENDIF;
11.                 ENDDO;
12.         ENDIF;
13. ENDDO;
14.
```

*LENTELĖ 12 MATRICOS X FORMAVIMAS*

„Dvi poveikių poros nustatančios tą pačią įtaką tarp schemos įėjimų ir išėjimų nėra ekvivalentiškos, nes gali tikrinti skirtingus vėlinimo gedimus net ir tos pačios schemos realizacijos. Įėjimo įtaka į išėjimą pasireiškia įvairiais schemos keliais. Todėl kaip nekokybiškas galima atmęsti tik tokias poveikių poras, kurios nenustato jokios įtakos tarp įėjimų ir išėjimų. Vykdant įvairius eksperimentinius tyrimus buvo pastebėta, kad tokių poveikio porų didesnėm schemom yra nedaug. Todėl apibūdinant funkcinio testo kokybę tenka ieškoti kompromiso tarp funkcinio testo ilgio ir jo kokybės. Kompromisiniu atveju galima apriboti kiek testo poveikių porų turi nustatyti įėjimo įtaką į išėjimą. Įvedus koeficientą Δ, galima lanksčiai surasti kompromisą tarp testo ilgio ir jo kokybės. Ribiniu atveju, kai Δ=1, pakanka tik vieną kartą nustatyti įėjimo įtaką į išėjimą ir testo kokybę tuo atveju charakterizuos funkcinio testo poveikio porų matricos X vienetinių elementų kiekis. Tai yra funkcinio testo kokybės apatinė riba, nes akivaizdu, kad jeigu funkcinis testas nenustato kai kurių įėjimų įtakos į išėjimus, tai toks testas tikrai netikrina  dalies schemos kelių, konstantinių gedimų ir tuo pačiu

vėlinimo gedimų. Reikia pastebėti, kad matricos X kai kurie elementai visuomet lygūs nuliui ( įėjimas neturi elektrinio ryšio su išėjimu), kas apsunkina funkcinio testo kokybės laipsnio (pilnumo) vertinimą, tačiau netrukdo palyginti kokybę skirtingų funkcinių testų.“ [20]

Iš čia suformuojamas tikslas, pagal pateiktus suprojektuoti ir ištirti schemų pilnumą, tiesioginę ir netiesioginę įtaką schemoms.

# 3 VĖLINIMO GEDIMŲ TESTAVIMO MODELIO PROJEKTAVIMO GALIMYBĖS

Šio darbo tikslas ištirti keletą funkcinio testo generavimo variantų:

- Pagal stipriosios - tiesioginės įtakos taisykles *(angl.“robust”);*
- Pagal silpnosios – netiesiginės įtakos taisykles *(angl.“nonrobust”);*
- Pagal modifikuotą bendrą pirminiu dviejų testų variantą;

## 3.1 Projektinio sprendimo analizė

Kad ištirti nustatytas šio darbo tyrimo sritis, remsimės užsakovo pateiktais algoritmų šablonais. Realizavę algoritmus galėsime palyginti ir įsitikinti kaip veikia vėlinimo gedimų metodai schemas.

## 3.2 Programavimo kalbos alternatyvos

Programavimo kalbą aš rinkausi iš „C++ Builder”(Visual Studio), „Borland Delfi”, „Java Builder“.

C++ Builder - tai objektiškai orientuotų programų kūrimo sistema, kurios priemonėmis galima paruošti įvairaus lygio ir paskirties programas: tiek mėgėjiškas, tiek profesionalias. Programos C++ Builder terpėje kuriamos, įtraukiant į programą reikalingus objektus (komponentus), suteikiant jiems reikalingas savybes ir programuojant C++ kalba reakcijas į įvykius, kurie gali įvykti programos vykdymo metu. Visų tų dalių sujungimą į vientisą C++ kalbos sintaksę atitinkančią programą automatiškai atlieka C++ Builder sistema.

Programos vartotojui nėra labai svarbu, kokia kalba parašyta programa, tačiau svarbu, kad ji atliktų numatytas funkcijas ir būtų patogi vartojimui. Galima šiek tiek palyginti šiuo metu naudojamas objektinio programavimo kalbas.

Programuotojai naudoja Microsoft Visual C++ arba jos modifikacijas. Pagrindinis šios kalbos privalumas yra labai nedidelės talpos EXE byla, kurios dydį galima lyginti su Asemblerio exe byla. C++ exe failą generuoja vidinis statinis kompiliatorius ir jo paleidimui nereikalingos papildomos (runtime) bibliotekos. Jei programa yra kraunama per internetą, jos apimtis yra vienas iš svarbesnių rodiklių. Visual C++ kalba yra rašomos DLL bylos ir Active X komponentai. Kadangi sukurtų komponentų kodas yra dvejetainis, darbo greitis yra didelis. Dar vienas C++ privalumas yra bylų pernešamumas į kitas platformas, pvz., programos, sukurtos Windows OS, gali būti lengvai transformuojamos į Linux ar BeOS operacinių sistemų aplinką.

Be abejo, C++ turi ir trūkumų. Kadangi ją daugiausiai naudoja profesionalūs programuotojai, labai mažai yra nemokamų bibliotekų ir komponentų. Pats programavimo kalbos mokymasis yra sunkus, nors literatūros šia tema pastaruoju metu daugėja.

Kita šiuo metu sparčiai populiarėjanti programavimo kalba yra Borland Delfi. Jos populiarumą apsprendžia dar ir tai, kad komandų kodas primena Paskalio kalbos kodą, kurio programavimo pagrindai yra dėstomi vidurinėse mokyklose. Kaip ir C++ Delfi kalboje sukurtos programos yra kompiliuojamos į exe bylas nenaudojant papildomų bibliotekų. Paleidžiamųjų bylų apimtis viršija analogiškų programų, sukurtų C++ apimtį, bet paprastai nėra labai didelės. Tuo Delfi programos skiriasi nuo pagrindinio savo konkurento Visual Basic. Pagrindiniai sunkumai programuojant Delfi kalba yra rezidentinių programų kūrimas. Šių programų talpa ribojama keliais šimtais kilobaitų. Kurti Active X komponentus Delfi kalba yra beprasmiška: kompiliatoriaus generuojama bylų talpa yra didelė ir dėl to komponentų darbo greitis yra mažas.

Programos Visual Component Library (VCL) yra labai plati ir beveik visi jie platinami nemokamai. Be to beveik visų komponentų programų kodai yra su paaiškinimais, todėl juos naudoti neprofesionaliam programuotojui nėra labai sudėtinga. Atviras kodas leidžia juos modifikuoti arba naudoti tik kodo fragmentus. Dar

viena gera ypatybė – komponentus išleidžia įvairios firmos, todėl jei yra klaidų, galima atsisiųsti kito gamintojo produktą.

Kitas trūkumas yra Active X komponentų panaudojimas programose. Šie komponentai paprastai sukurti konkrečiai Windows versijai. Perėjus į kitą platformą gerai dirbanti programa gali strigti. Taip pat atsiranda konfliktai tarp seno ir naujo tipo Active X komponentų.

Apžvelgus programavimo kalbų paketus galima daryti išvadą, kad mažiausios apimties produktai gaunami naudojant Visual C++ kalbą (neskaitant Asemblerio, kuriuo programuoti moka nedaugelis ir, be to, būtinas geras procesorių architektūros žinojimas). Sekanti po Visual C++ būtų Delfi. Ir tik kuriant programas, kurių apimtis nėra kritinis rodiklis, galima naudoti Visual Basic. Dar liko nepaminėta Java programavimo kalba, kuri šio metu sparčiai vystosi ir pagal galimybes lygiuojasi su C++.

Magistro darbo realizavimui aš pasirinkau C++ Visual Studio, kadangi jos failu apimtis yra mažiausia, taip pat didelė greitaveika (kadangi tą patį veiksmą atlikti reikia rašyti mažiau eilučių nei su kita programavimo kalba) ir dėl to, kad yra tekę dirbti su senesne C++ versija.

## 3.3 Poveikių poros funkcinio vėlinimo gedimų testo generavime

Šiame skyriuje remiantis teorija [20] aptarsime kokiu būdu gali būti generuojamos funkcinio testo poveikių poros, kad generuoti visą testą.

„Turint funkcinio vėlinimo testo kokybės kriterijus galima vykdyti funkcinio testo generavimą. Kai testo generavimo metu naudojamas imitacinis schemos modelis ir nežinoma konkreti schemos realizacija(pav.1), tenka naudotis poveikio porų atsitiktine paieška. Tačiau žinoma prieš tai panagrinėsime, kaip galima poveikių porą modifikuoti, kad modifikavimo pasekoje daugiau įėjimų turėtų įtakos į išėjimus. Tam tikslui galima keisti kai kurias fiksuotas įėjimo signalų reikšmes formuojant pokytį (0→1) arba (1→0) arba panaikinti pokytį. Įvedus pakeitimą galimi trys atvejai. Vienu atveju pakeitimas neįtakoja išėjimų ir jau nustatytų įėjimų įtakos į išėjimus. Šiuo atveju įvesti pakeitimą netikslinga. Pakeitimą tikslinga įvesti tuo atveju, kai jis nustato naujas įtakas į išėjimą ir nekeičia jau nustatytų įėjimų įtakos į išėjimus. Jeigu įvedant

pakeitimą prarandamos kai kurios jau nustatytos įtakos į išėjimus, tikslinga turimos poveikių poros nekeisti, bet formuoti naują poveikių porą su kitais pakeitimais. Tokiu būdu išnagrinėje visas poveikių poros įėjimo signalų reikšmes, gausime modifikuotą pirminę poveikių porą ir naujas poveikių poras nustatančias papildomas įėjimų įtakas į išėjimus.“[20]

Aprašysime algoritmą PPG(Poveikio porų generavimas), kuris modifikuoja duotą poveikių porą ir suformuoja kitas panašias poveikių poras.

*LENTELĖ 13 PPG ALGORITMAS – POVEIKIO PORŲ GENERAVIMAS*

```
1.  Suformuojame vektorių V, kur v_i=0, i=1,2,...,n ;PAB:=0;
2.  DO WHILE (PAB=0) PAB:=1;
3.          Duotiems įėjimo poveikiams P^1 ir P^2 paskaičiuojame matricą X.
4.          /* modifikuojam poveikių porą
5.          DO i:=1 iki n
6.              IF p_i^1≠p_i^2 THEN p_i^2:= p_i^1;
7.                  Pakeistai poveikių porai paskaičiuojame matricą X';
8.                  IF(X'>X)| (X'=X) THEN; X=X';ELSE p_i^2:= 1-p_i^1;
    ENDIF;
9.              ELSE p_i^2:= 1-p_i^1
10.                 Pakeistai poveikių porai paskaičiuojame matricą X';
11.                 IF (X'>X) THEN X=X'; ELSE p_i^2:= p_i^1; ENDIF;
12.             ENDIF;
13.     ENDDO; Užsaugom modifikuotą poveikių porą P^1 ir P^2;
14.     /* Formuojam panašią poveikių porą
15.     DO i:=1 iki n
16.         IF (PAB=1) THEN
17.             IF v_i=0 THEN v_i:=1;
18.                 IF p_i^1≠p_i^2 THEN p_i^2:= p_i^1;
19.                     Pakeistai poveikių porai paskaičiuojame
    matricą X';
20.                     IF X'≠X THEN PAB:=0; ELSE p_i^2:=1-p_i^1;
    ENDIF;
21.                 ELSE p_i^2:= 1-p_i^1
22.                     Pakeistai poveikių porai paskaičiuojame
    matricą X';
23.                     IF X'≠ X THEN PAB:=0; ELSE p_i^2:=p_i^1;
    ENDIF;
24.                 ENDIF;
25.             ENDIF;
26.         ENDIF;
27.     ENDDO;
28. ENDWHILE;
```

X'> X – visi matricos X elementai yra didesni arba lygūs už matricos X elementus. Atitinkamai, jei X'= X – reiškia visi matricos elementai yra lygūs ir pan. O X'≠X reiškia, kad kai kurie matricos X elementai yra didesni už matricos X', o kai kurie mažesni ir atvikščiai.

Suformuojamas vektorius V, kurio elementai rodo išnagrinėtus poveikio įėjimus panašių poveikių porų formavimui ir pradžioje visi elementai prilyginami nuliui. Toliau duotai poveikių porai paskaičiuojama kokybę charakterizuojanti matrica X pagal antram skyriuje aprašytas taisykles. Algoritmas užbaigia darbą kai nei vienam įėjimui negalima suformuoti panašios poveikių poros, t. y. kai vektoriaus V visi elementai lygūs vienetui. Pirmiausia modifikuojamos įėjimo signalų reikšmės kurios nemažina

matricos X vienetukų kiekio (punktai 4-13). Jeigu įėjimo poveikių poros reikšmės keičiasi, tai bandoma tą pokytį panaikinti ir jei šis panaikinimas padidina arba palieka tą patį matricos X vienetukų kiekį, tai šis pokyčio panaikinimas paliekamas ir toliau nagrinėjama taip modifikuota poveikių pora. Jeigu pokyčio panaikinimas matricoje X kai kuriuos vienetukus prideda, o kai kuriuos panaikina, tai pokytis nekeičiamas ir tuo bus pasinaudota formuojant panašius poveikių poras. Tuo tarpu naujo pokyčio suformavimas įėjime jau gali padidinti matricos X vienetukų kiekį ir toks pakeitimas paliekamas. Išnagrinėjus visus įėjimus gaunama modifikuota poveikių pora atžyminti matricoje X nemažiau elementų ir ji užsaugojama. Medžiaga naudojama remiantis specialiais šaltiniais [20].

## 3.4 Algoritmas eksperimentiniam tyrimui

Aprašysime algoritmą, kuris modifikuoja duotą poveikių porą ir suformuoja kitas panašias poveikių poras.

Pirmiausia suformuojamas vektorius V, kurio elementai rodo išnagrinėtus poveikio įėjimus panašių poveikių porų formavimui ir pradžioje visi elementai prilyginami nuliui.

Toliau duotai poveikių porai paskaičiuojama kokybę charakterizuojanti matrica X.

Algoritmas užbaigia darbą kai nei vienam įėjimui negalima suformuoti panašios poveikių poros, t. y. kai vektoriaus V visi elementai lygūs vienetui.

Pirmiausia modifikuojamos įėjimo signalų reikšmės kurios nemažina matricos X vienetukų kiekio.

Jeigu įėjimo poveikių poros reikšmės keičiasi, tai bandoma tą pokytį panaikinti ir jei šis panaikinimas padidina arba palieka tą patį matricos X vienetukų kiekį, tai šis pokyčio panaikinimas paliekamas ir toliau nagrinėjama taip modifikuota poveikių pora.

Jeigu pokyčio panaikinimas matricoje X kai kuriuos vienetukus prideda, o kai kuriuos panaikina, tai pokytis nekeičiamas ir tuo bus pasinaudota formuojant panašius

poveikių poras. Tuo tarpu naujo pokyčio suformavimas įėjime jau gali padidinti matricos X vienetukų kiekį ir toks pakeitimas paliekamas.

Išnagrinėjus visus įėjimus gaunama modifikuota poveikių pora atžyminti matricoje X nemažiau elementų ir ji užsaugojama.

Toliau keičiant antro poveikio signalų reikšmes bandoma suformuoti panašias poveikių poras.

Jeigu įėjimo pokyčio panaikinimas matricoje X vienetukų ir sumažina ir padidina, tai pakeista poveikų pora traktuojama kaip pradinė ir pereinama prie jos modifikavimo.

Jeigu matricoje X neatsiranda naujų vienetukų tai įėjime paliekama tas pats pokytis ir uždedamas požymis $v_i:=1$, kad įėjimas jau išnagrinėtas.

Analogiškai nagrinėjami ir įėjimai, kurių poveikių poros reikšmės yra vienodos, tik čia bandoma įėjime suformuoti naują pokytį. Jei naujas pokytis ir padidina ir sumažina matricos X vienetukų skaičių tai gauta panašių poveikių pora perduodama nagrinėti modifikavimui.

Jeigu matricos X vienetukų skaičius nesikeičia ar mažėja tai įėjime paliekama ta pati pastovi reikšmė ir uždedamas požymis $v_i:=1$.

Algoritmas baigia darbą, kai visi įėjimai išnagrinėjami panašių poveikio porų formavimui.

Kad galėtume ištirti kaip veikia vėlinimų gedimų testavimo metodai reikalinga suprojektuoti ir ištirti funkcinių vėlinimo testų generavimas pagal pokyčius išėjimuose su panašių nagrinėjimu [20].

Nuskaitom iš failo funkcinį testą,nustatom rinkinių kiekį tk; (Nurodom L)

$X=||x_{i,j}:=0||_{2n \times 4m}$; tp:=1;

```
1.          DO WHILE (tp≠tk) tpp:=tp; PZ:=0; h:=tp;
2.              //Testiniams rinkiniams paskaičiuojam matricą X
3.                  DO t:=tp,…, tk, (kas 2) ;
4.                      P^1:=P^t; R^1:=f(P^1); P^2:=P^(t+1); R^2:=f(P^2);
5.                      DO i: =1,2,3,...,n; P^3: =P^2; d:=1- p^1_i;
6.                          IF (p^1_i≠p^2_i) THEN p^3_i:=p^1_i; R^3=f(P^3);
7.                              DO j: =1,2,3,...,m; c:=3- r^1_j;
8.                                  IF (r^3_j≠r^2_j) THEN
9.                                      IF(r^2_j≠r^1_j) THEN IF
    (x_2i-d,4j-c=0) THEN  x_2i-d,4j-c:=1; PZ:=1;
10.
            ELSE x_2i-d,4j-c:= x_2i-d,4j-c +1/( x_2i-d,4j-c+1);
11.
            ENDIF;
12.                                     ELSE IF (x_2i-d,4j-
    c+2=0) THEN  x_2i-d,4j-c+2:=1; PZ:=1;
13.                                         ELSE x_2i-
    d,4j-c+2:= x_2i-d,4j-c+2+1/( x_2i-d,4j-c+2+1); ENDIF;
14.                                     ENDIF;
15.                                 ENDIF;
16.                             ENDDO;
17.                         ENDIF;
18.                     ENDDO;
19.                     IF (PZ=1) THEN P^h:=P^1; P^(h+1):=P^2;h:=h+2; tk:=h-
    1; ENDIF;
20.                 ENDDO; tp:=tk+1;
21.                 h:=tk+1;
22.                 DO ii: =1,2,3,...,n;
23.                     DO t:=tpp,…, tk, (kas 2) ;
24.                         P^1:=P^t; R^1:=f(P^1); P^2:=P^(t+1); p^2_ii:=1- p^2_ii;
    R^2:=f(P^2); SK:=0;
25.                             DO i: =1,2,3,...,n; P^3: =P^2; d:=1-
    p^1_i;
26.                                 IF (p^1_i≠p^2_i) THEN p^3_i:=p^1_i;
    R^3=f(P^3);
27.                                     DO j: =1,2,3,...,m;
    c:=3- r^1_j;
28.                                         IF (r^3_j≠r^2_j)
    THEN
29.
            IF(r^2_j≠r^1_j) THEN IF  (x_2i-d,4j-c=0
30.
                    THEN SK:=SK+1;ENDIF;
      31.
        ELSE IF (x_2i-d,4j-c+2=0
32.
                THEN SK:=SK+1;ENDIF;
33.
            ENDIF;
34.                                         ENDIF;
```

35. ENDDO;
36. ENDIF;
37. ENDDO;
38. IF (SK>0) THEN $P^h:=P^1$; $P^{h+1}:=P^2$;h:=h+2; tk:=h-1; ENDIF;
39. ENDDO;
40. ENDDO;
41. SKM:=0;
42. DO k:=1,2,3, , L;
43. $P^2:= (p^2_1,p^2_2,...,p^2_i,...,p^2_n)$, $p^2_i$:=Random(0,1); $R^2$:=f($P^2$);
44. $P^1:= (p^1_1,p^1_2,...,p^1_i,...,p^1_n)$, $p^1_i$:=Random(0,1); $R^1$:=f($P^1$);
45. SK1:=0;SK2:=0; SK3:=0;SK4:=0;
46. DO i: =1,2,3,...,n; $P^3$: =$P^2$; d:=1- $p^1_i$;
47. IF ($p^1_i \neq p^2_i$) THEN $p^3_i:=p^1_i$; $R^3$=f($P^3$);
48. DO j: =1,2,3,...,m; c:=3- $r^1_j$;
49. IF ($r^3_j \neq r^2_j$) THEN
50. IF($r^2_j \neq r^1_j$) THEN IF ($x_{2i-d,4j-c}$=0)
51. THEN SK1:= SK1+1;ENDIF;
52. ELSE SK3:= SK3+ $x_{2i-d,4j-c}$; ENDIF;
53. ELSE IF ($x_{2i-d,4j-c+2}$=0)
54. THEN SK2:= SK2+1; ENDIF;
55. ELSE SK4:= SK4+ $x_{2i-d,4j-c+2}$; ENDIF;
56. ENDIF;
57. ENDIF;
58. ENDDO;
59. ENDIF;
60. ENDDO;
61. SK:=k1*SK1+ k2*SK2+ k3*SK3+ k4*SK4;
62. IF (SK>SKM) THEN SKM:=SK; $P^{1M}$:=$P^1$;$P^{2M}$:=$P^2$; ENDIF;
63. ENDDO;
64. IF (SKM>0) THEN $P^h$:=$P^{1M}$; $P^{h+1}$:=$P^{2M}$; h:=h+2; tk:=h-1; ENDIF;
65. ENDWHILE;

Šio algoritmo realizavimą plačiau aptarsiem eksperimentinėje darbo dalyje.

## 3.5 Tyrimo objektas

Tyrimo objektu pasirinkta ISCAS'89 schemų aibė – s27 ….s13207 . Reikalingi šių schemų duomenys pateikti Pav.14. Schemų turinio pavyzdys pateiktas Priede 3.

Kad galėtume suprojektuoti anksčiau minėta algoritmą, reikalingą bus naudoti šiuos duomenis:n- schemos įėjimų skaičius;m- schemos išėjimų skaičius;L- atsitiktinių testinių rinkinių skaičius.

| Nr. | Schema | Įėjimai | Išėjimai | Įėjimai, n | Išėjimai, m | Ventiliai |
|---|---|---|---|---|---|---|
| 1. | S27 | 4 | 1 | 7 | 4 | 8 + 2 = 10 |
| 2. | S208 | 11 | 2 | 19 | 10 | 61 + 35 = 96 |
| 3. | S298 | 3 | 6 | 17 | 20 | 75 + 44 = 119 |
| 4. | S386 | 7 | 7 | 13 | 13 | 118 + 41 = 159 |
| 5. | S400 | 3 | 6 | 24 | 27 | 106 + 56 = 162 |
| 6. | S444 | 3 | 6 | 24 | 27 | 119 + 62 = 181 |
| 7. | S9234 | 36 | 39 | 247 | 250 | 2027 + 3570 = 5597 |
| 8. | S13207 | 62 | 152 | 700 | 790 | 2573 + 5378 = 7951 |

*LENTELĖ 14 ISCAS'89 SCHEMŲ AIBĖ PANAUDOTA ŠIO DARBO EKSPERIMENTINIAME TYRIME*

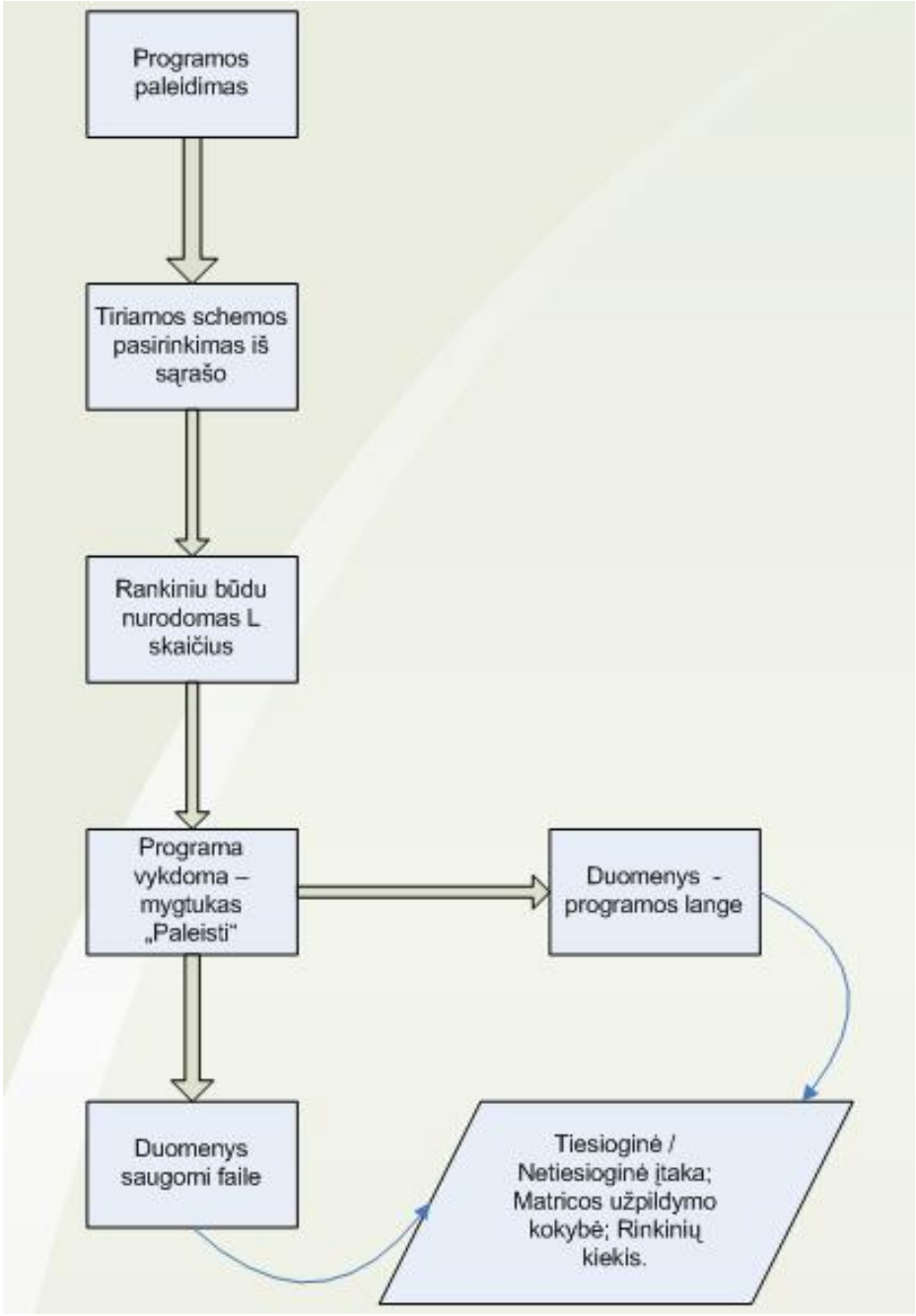
# 4 VĖLINIMŲ GEDIMŲ METODŲ TESTAVIMO EKSPERIMENTAS

Suprojektavus pasirinktus algoritmus pavyko ištirti s schemas.Šiame skyriuje pateiksime rezultatus.

## 4.1 Eksperimento realizacija

Kad galėtume atlikti testavimą pasirinktam algoritmui, t.y. funkcinių vėlinimo testų generavimas pagal pokyčius išėjimuose su panašių nagrinėjimu, reikalinga parinkti testuojamą schemą iš bendro jų sąrašo programos lange ir nurodyti atsitiktinių testinių rinkinių skaičių L (kuris šiame tyrime bus žymimas nuo 10-100 000).

Pav.15 pavaizduota eksperimentinės programos veiksmų diagrama.




*LENTELĖ 15 PROGRAMOS VEIKSMŲ DIAGRAMA.*

Kad būtų aiškus algoritmo veikimas paveiksle 16 pateikiame algoritmo veiksmų diagramą.

## 4.2 Vėlinimo gedimų eksperimento rezultatai

Generuojant rezultatus tyrėme kiek schemos yra užpildomos ir kokia tiesioginė ir netiesioginė įtakos sudaro testavimo rezultatus.

Eksperimento rezultatai labiausiai pavykę tiriant schemas s298 ir s208. Likusių schemų rezultatus pateikiame prieduose.

**Schema s298:**

| **Pirmas testas** | | | | |
|---|---|---|---|---|
| L | Rinkiniai | Tiesioginė įtaka | Netiesioginė įtaka | Užpildymas % |
| 10 | 220 | 4624 | 566 | 15 |
| 100 | 374 | 4624 | 709 | 16 |
| 1000 | 464 | 4624 | 736 | 16 |
| 10000 | 482 | 4624 | 741 | 16 |
| 100000 | 486 | 4624 | 741 | 16 |
| 1000000 | 486 | 4624 | 741 | 16 |
| **Antras testas** | | | | |
| L | Rinkiniai | Tiesioginė įtaka | Netiesioginė įtaka | Užpildymas % |
| 10 | 486 | 4624 | 741 | 16 |
| 100 | 486 | 4624 | 741 | 16 |
| 1000 | 486 | 4624 | 741 | 16 |
| 10000 | 486 | 4624 | 741 | 16 |
| 100000 | 486 | 4624 | 741 | 16 |

**Schema s208003A**

| **Pirmas testas** | | | | |
|---|---|---|---|---|
| L | Rinkiniai | Tiesioginė įtaka | Netiesioginė įtaka | Užpildymas % |
| 10 | 42 | 100 | 28 | 30 |
| 100 | 56 | 100 | 32 | 30 |
| 1000 | 56 | 100 | 32 | 30 |
| 10000 | 56 | 100 | 32 | 30 |
| 100000 | 56 | 100 | 32 | 30 |
| 1000000 | 56 | 100 | 32 | 30 |
| **Antras testas** | | | | |
| L | Rinkiniai | Tiesioginė įtaka | Netiesioginė įtaka | Užpildymas % |
| 10 | 56 | 100 | 32 | 30 |
| 100 | 56 | 100 | 32 | 30 |
| 1000 | 56 | 100 | 32 | 30 |
| 10000 | 56 | 100 | 32 | 30 |
| 100000 | 56 | 100 | 32 | 30 |

# 5 IŠVADOS

- Išanalizuota vėlinimo gedimų testavimo metodologija: vėlinimo gedimai tikrinami rinkinių pora. Rinkinių poros signalų reikšmės apsprendžia, kurie įėjimai keičiasi ir kurie lieka pastovūs. Gali keistis vienas ar daugiau įėjimų.
- Nagrinėjome jau aptartus, sukurtus vėlinimo testavimo modelius. Išanalizavome jų privalumus bei trūkumus.
- Apibrėžėme ir analizavome apibendrintą vėlinimo gedimo susidarymo modelį.
- Nagrinėjome dažniausiai naudojamus: Perjungimo vėlinimo gedimų metodą bei Kelio vėlinimo metodą ypatingą dėmesį skyrėme Kelio vėlinimo gedimų testavimo metodo savybėms.
- Nustatėme , kad efektyviausiai gedimų galima aptikti naudojantis vėlinimų testavimu pagal tam tikrus nustatytus kriterijus .Naudojant bendrą tiesioginės ir netiesioginės įtakos taisykles (angl. robust ir nonrobust)kelio vėlinimo gedimų testavime galima aptikti daugiausia gedimų.
- Taip pat buvo aptartos galimybes atrinkti testuojamus ir netestuojamus kelius tiriamose schemos, kai testuojami schemos kelio vėlinimo gedimai.
- Nustatėme ir išanalizavome vėlinimo testo kokybės kriterijus, bei testinių porų poveikį generuojamam testui. Aptarėm metodą, kuriuo galime modifikuoti poveikio poras, be papildomų pakeitimų.
- Pasirinktas ir ištirtas funkcinis testo generavimas pagal pokyčius tiriamos schemos išėjimuose. Kaip tyrimo objektas pasirinkta ISCAS'89 schemų aibės dalis. Tyrėme algoritmo matricos X užpildymo kokybę, tiesioginės bei netiesioginės įtakos poveikį. Tyrimą atlikome keisdami atsitiktinių rinkinių kiekius.

# 6 SANTRUMPŲ IR TERMINŲ ŽODYNAS

| | |
|---|---|
| ***GRT (General Robust Tests)*** | Bendras tiesioginės įtakos testas |
| ***GNRT (General Non-Robust Tests)*** | Bendras netiesioginės įtakos testas |
| ***HFRT (Hazard Free Robust Tests)*** | Spėjamas neapribotas tiesioginės įtakos testas |
| ***RRT (Restricted Robust tests)*** | Apribotas tiesioginės įtakos testas |
| ***RNRT (Restricted Non-Robust Tests)*** | Neapribotas tiesioginės įtakos testas |
| ***ATPG*** | Automatinis tstinių rinkinių generavimas |
| ***Directed Acyclic Graph (DAG)*** | Tiesioginio ciklinio grafiko |
| ***Path Delay Fault (PDF)*** | Kelio vėlinimo gedimai |
| ***TSGS*** | Testinių sekų generavimo schemoje |
| ***TVP*** | Testinių vektorių poros |
| ***ĮSS*** | Įėjimo sinchroninio signalo |
| ***TKS*** | Testinės kombinacinės schemos |
| ***IšSS*** | Išėjimo sinchroninio signalo |
| ***SRAS*** | Schemos reakcijos analizavimo schema |
| ***STR (slow-to-rise)*** | Kylantis signalo frontas |
| ***STF (slow-to-fall)*** | Krintantis signalo frontas |
| ***PPG*** | Poveikio porų generavimas |

# 7 LITERATŪROS ŠALTINIAI


[1] Ž. Tamoševičius, “Save testuojančių sistemų testavimas”.//Elektronika ir Elektrotechnika Nr.(7)63 , 2005. – www.elektronika.lt

[2] Tamoševičius Ž. Save testuojančių skaitmeninių schemų testavimo metodai // Informacinės technologijos 2005, konferencijos pranešimų medžiaga, 2005. – P. 371 – 376.

[3] VLSI Design Verification and Test, Delay Faults II – slides, 2006

[4] Jusas V., Z.Tamosevi ius, R.Benisevi iute. Testu issamumo uztikrinimas save testuojan iose skaitmeninese schemose.

Elektronika ir Elektrotechnika. ISSN 1392–1215. Kaunas, Technologija, 2004, Nr. 1(50), p. 56 – 61.

[5] Ian g Harris, “Hardware-Software Covalidation: Fault models and test Generation”.

[6] Brad Hill, “Transition Delay Faults”, ELEC 7250, April 2006

[7] http://www.softwaretechnews.com/stn3-3/testing-models.html

[8] http://netlib.bell-labs.com/who/god/public_psfiles/smart.pdf

[9] Delay Fault Functional Test generation, 2007

[1] A Tutorial on Delay Fault Testing, Janak H.Patel , 2005

[11] K. Fuchs, M. Pabst and T. Rossel, “RESIST: A recursive test pattern generation algorithmfor path delay faults considering various test classes”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,1994

[12] P.Gupta, „High quality Transition and Small Delay Fault ATPG“, 2004

[13] S. Yihe and W. Qifa, “FSIMGEO: A Test Generation Method for Path Delay Fault Test Using Fault Simulation and Genetic Optimization”, 2001.

[14] M.Sharma and J.H. Patel, “Testing of Critical Paths for Delay Faults”, 2001.

[15] M. Sharma, J.H. Patel, “Finding a small set of longest testable paths that cover every gate”, 2002.

[16]W. Qiu and D. M. H. Walker, “An Efficient Algorithm for Finding the K Longest Testable Paths Through Each Gate in a Combinational Circuit”, 2003.

[17] J.J. Liou, L.C. Wang, K.T. Cheng, “On theoretical and practical considerations of path selection for delay fault testing”, 2002.

[18] J.J. Liou, A. Krstic, L.C. Wang, K.T. Cheng “False-path-aware statistical timing analysis and efficient path selection for delay testing and timing validation”, 2002.

[19] UMBC, „Deign Verification and testing“, 2002

[20] prof.V.Jusas, prof.K.Motiejūnas “Generation of functional delay test with multiple input transitions”, 2007

# 8 PRIEDAI

## 8.1 Priedas 1

1. Funkcinių vėlinimo testų generavimo algoritmas pagal pokyčius išėjimuose su panašių nagrinėjimu
2. Nuskaitom iš failo funkcinį testą,nustatom rinkinių kiekį tk; (Nurodom L)
3. $X=\|x_{i,j}:=0\|_{2n \times tm}$; tp:=1;
4. DO WHILE (tp≠tk) tpp:=tp; PZ:=0; h:=tp;
5. //Testiniams rinkiniams paskaičiuojam matricą X
6. DO t:=tp,…, tk, (kas 2) ;
7. $P^1:=P^t$; $R^1:=f(P^1)$; $P^2:=P^{t+1}$; $R^2:=f(P^2)$;
8. DO i: =1,2,3,...,n; $P^3$: $=P^2$; d:=1- $p^1_i$;
9. IF ($p^1_i \neq p^2_i$) THEN $p^3_i:=p^1_i$; $R^3=f(P^3)$;
10. DO j: =1,2,3,...,m; c:=3- $r^1_j$;
11. IF ($r^3_j \neq r^2_j$) THEN
12. IF($r^2_j \neq r^1_j$) THEN IF ($x_{2i-d,4j-c}=0$) THEN $x_{2i-d,4j-c}:=1$; PZ:=1;
13. ELSE $x_{2i-d,4j-c}:= x_{2i-d,4j-c}+1/( x_{2i-d,4j-c}+1)$;
14. ENDIF;
15. ELSE IF ($x_{2i-d,4j-c+2}=0$) THEN $x_{2i-d,4j-c+2}:=1$; PZ:=1;
16. ELSE $x_{2i-d,4j-c+2}:= x_{2i-d,4j-c+2}+1/( x_{2i-d,4j-c+2}+1)$; ENDIF;
17. ENDIF;
18. ENDIF;
19. ENDDO;
20. ENDIF;
21. ENDDO;
22. IF (PZ=1) THEN $P^h:=P^1$; $P^{h+1}:=P^2$;h:=h+2; tk:=h-1; ENDIF;
23. ENDDO; tp:=tk+1;
24. // Panašių generavimas. Turėtų būti galimybė išjungti pagal požymį.
25. h:=tk+1;
26. DO ii: =1,2,3,...,n;
27. DO t:=tpp,…, tk, (kas 2) ;
28. $P^1:=P^t$; $R^1:=f(P^1)$; $P^2:=P^{t+1}$; $p^2_{ii}:=1- p^2_{ii}$; $R^2:=f(P^2)$; SK:=0;
29. DO i: =1,2,3,...,n; $P^3$: $=P^2$; d:=1- $p^1_i$;
30. IF ($p^1_i \neq p^2_i$) THEN $p^3_i:=p^1_i$; $R^3=f(P^3)$;
31. DO j: =1,2,3,...,m; c:=3- $r^1_j$;
32. IF ($r^3_j \neq r^2_j$) THEN
33. IF($r^2_j \neq r^1_j$) THEN IF ($x_{2i-d,4j-c}=0$
34. THEN SK:=SK+1;ENDIF;
35. ELSE IF ($x_{2i-d,4j-c+2}=0$
36. THEN SK:=SK+1;ENDIF;
37. ENDIF;
38. ENDIF;
39. ENDDO;
40. ENDIF;
41. ENDDO;
42. IF (SK>0) THEN $P^h:=P^1$; $P^{h+1}:=P^2$;h:=h+2; tk:=h-1; ENDIF;
43. ENDDO;
44. ENDDO;

```
45.       // Nagrinėjam 2L atsitiktinai sugeneruotų rinkinių ir iš jų išrenkam geriausią
                                                    porą
46.       SKM:=0;
47.       DO k:=1,2,3,   , L;
48.             P2:= (p21,p22,...,p2i,...,p2n),  p2i:=Random(0,1); R2:=f(P2);
49.             P1:= (p11,p12,...,p1i,...,p1n),  p1i:=Random(0,1); R1:=f(P1);
50.             SK1:=0;SK2:=0; SK3:=0;SK4:=0;
51.             DO i: =1,2,3,...,n; P3: =P2; d:=1- p1i;
52.                   IF (p1i≠p2i) THEN p3i:=p1i; R3=f(P3);
53.                         DO j: =1,2,3,...,m; c:=3- r1j;
54.                               IF (r3j≠r2j) THEN
55.                                     IF(r2j≠r1j) THEN IF (x2i-d,4j-c=0)
56.                                                      THEN SK1:= SK1+1;ENDIF;
57.                                                      ELSE SK3:= SK3+ x2i-d,4j-c; ENDIF;
58.                                           ELSE IF (x2i-d,4j-c+2=0)
59.                                                 THEN SK2:= SK2+1; ENDIF;
60.                                                 ELSE SK4:= SK4+ x2i-d,4j-c+2; ENDIF;
61.                                     ENDIF;
62.                               ENDIF;
63.                         ENDDO;
64.                   ENDIF;
65.             ENDDO;
66.             SK:=k1*SK1+ k2*SK2+ k3*SK3+ k4*SK4;
67.             IF (SK>SKM) THEN SKM:=SK; P1M:=P1;P2M:=P2; ENDIF;
68.       ENDDO;
69.       IF (SKM>0) THEN Ph:=P1M; Ph+1:=P2M; h:=h+2; tk:=h-1; ENDIF;
70. ENDWHILE;
```

## 8.2 Priedas 2

### S298.h

```
void funkcija(char *gg,char *og,int in,int on,int *gerai)
{
 int G28,G38,G40,G45,G46,G50,G51,G54,G55,G59,G60,G64;
 int I155,G66,I158,G67,G76,G82,G87,G91,G93,G96,G99;
 int G103,G108,G114,I210,G117,I213,G118,G120,G121;
 int I221,G124,G126,G127,I229,G130,I232,G131,I235;
 int G132,I238,G133,G26,G27,G31,G32,G33,G35,G36,G37;
 int G42,G48,G49,G57,G58,G62,G63,G74,G75,G88,G89,G90;
 int G94,G95,G100,G105,G110,G111,G115,G122,G123,G128;
 int G129,G24,G25,G68,G69,G70,G71,G72,G73,G77,G78;
 int G79,G80,G81,G83,G84,G85,G41,G43,G52,G65,G97,G101;
 int G106,G109,G116,G29,G30,G34,G39,G44,G47,G53,G56;
 int G61,G86,G92,G98,G102,G104,G107,G112,G113,G119;
 int G125;
// input 17, output 20
  int ag[17],od[20],ii,m1=1;
 *gerai=1;
 for(ii=0;ii<in;ii++)
  if(*(gg+ii)=='0')
   *(ag+ii)=0;
  else
   *(ag+ii)=1;

for(ii=0;ii<25;ii++){
 G28=m1-G130;
 G38=m1-*(ag+3);
 G40=m1-*(ag+6);
 G45=m1-*(ag+5);
 G46=m1-*(ag+4);
 G50=m1-*(ag+7);
 G51=m1-*(ag+16);
 G54=m1-*(ag+4);
 G55=m1-*(ag+6);
 G59=m1-*(ag+5);
 G60=m1-*(ag+15);
 G64=m1-*(ag+8);
 I155=m1-*(ag+9);
 *(od+2)=m1-I155;
 I158=m1-*(ag+10);
 *(od+5)=m1-I158;
 G76=m1-*(ag+3);
 G82=m1-*(ag+4);
 G87=m1-*(ag+9);
 G91=m1-*(ag+5);
 G93=m1-*(ag+10);
 G96=m1-*(ag+7);
 G99=m1-*(ag+11);
 G103=m1-*(ag+6);
 G108=m1-G112;
 G114=m1-*(ag+14);
 I210=m1-*(ag+11);
 *(od+0)=m1-I210;
 I213=m1-*(ag+12);
 *(od+3)=m1-I213;
 G120=m1-G124;
 G121=m1-*(ag+15);
 I221=m1-*(ag+2);
```

```
G124=m1-I221;
G126=m1-G131;
G127=m1-*(ag+16);
I229=m1-*(ag+0);
G130=m1-I229;
I232=m1-*(ag+1);
G131=m1-I232;
I235=m1-*(ag+13);
*(od+1)=m1-I235;
I238=m1-*(ag+14);
*(od+4)=m1-I238;
G26=G28&G50;
G27=G51&G28;
G31=*(ag+3)&G45&*(ag+6);
G32=*(ag+3)&*(ag+4);
G33=G38&G46;
G35=*(ag+3)&*(ag+4)&*(ag+5);
G36=G38&G45;
G37=G46&G45;
G42=G40&G41;
G48=G45&G46&*(ag+3)&G47;
G49=G50&G51&G52;
G57=G59&*(ag+4)&G60&G61;
G58=G64&G65;
G62=G59&*(ag+4)&G60&G61;
G63=G64&G65;
G74=*(ag+5)&*(ag+7)&*(ag+12);
G75=G82&G91&*(ag+7);
G88=*(ag+7)&G87;
G89=G103&G96;
G90=G91&G103;
G94=G93&*(ag+6);
G95=G96&*(ag+6);
G100=G99&*(ag+7)&*(ag+5);
G105=G103&G108&G104;
G110=G108&G109;
G111=*(ag+3)&G112;
G115=G114&*(ag+7);
G122=G120&G121;
G123=G124&*(ag+15);
G128=G126&G127;
G129=G131&*(ag+16);
G24=G38|G46|G45|G40;
G25=G38|*(ag+4)|*(ag+5);
G68=*(ag+4)|*(ag+5)|*(ag+6)|G96;
G69=G103|*(ag+11);
G70=G103|*(ag+7);
G71=G82|*(ag+5)|*(ag+6);
G72=G91|*(ag+13);
G73=G103|*(ag+13);
G77=G112|G103|G96|*(ag+12);
G78=G108|G76;
G79=G103|*(ag+7);
G80=*(ag+4)|*(ag+7);
G81=*(ag+5)|*(ag+6);
G83=*(ag+4)|*(ag+5)|*(ag+6)|G96;
G84=G82|G91|*(ag+7);
G85=G91|G96|*(ag+10);
G41=m1-(*(ag+5)&*(ag+4)&*(ag+3));
G43=m1-(G24&G25&G28);
G52=m1-(*(ag+6)&G45&G46&*(ag+3));
```

```
 G65=m1-(G59&G54&*(ag+15)&G61);
 G97=m1-(G83&G84&G85&G108);
 G101=m1-(G68&G69&G70&G108);
 G106=m1-(G77&G78);
 G109=m1-(G71&G72&G73&*(ag+7));
 G116=m1-(G79&G80&G81&G108);
 *(od+6)=m1-(*(ag+3)|G130);
 *(od+7)=m1-(G31|G32|G33|G130);
 *(od+8)=m1-(G35|G36|G37|G130);
 *(od+9)=m1-(G42|G43);
 *(od+10)=m1-(G48|G49|G53);
 G47=m1-(G50|G40);
 G53=m1-(G26|G27);
 *(od+11)=m1-(G57|G58|G130);
 G61=m1-(*(ag+7)|G55);
 *(od+12)=m1-(G88|G89|G90|G112);
 *(od+13)=m1-(G94|G95|G97);
 *(od+14)=m1-(G100|G101);
 *(od+15)=m1-(G105|G106);
 G104=m1-(G74|G75);
 *(od+16)=m1-(G110|G111);
 G112=m1-(G62|G63);
 *(od+17)=m1-(G115|G116);
 *(od+18)=m1-(G122|G123|G130);
 *(od+19)=m1-(G128|G129|G130);
}
 for(ii=0;ii<on;ii++)
  if(*(od+ii)==0)
   *(og+ii)='0';
    else
   *(og+ii)='1';
}
```

**s208.h**

```
void funkcija(char *gg,char *og,int in,int on,int *gerai)
{
 int II6,II50,II40,II41,II42,II43,II44,II3,II4,II158;
 int II202,II192,II193,II194,II195,II196,II155,II156;
 int II307_1,II318,II309,II310,II311,P_1,P_3,P_4,II368;
 int II369,II370,II371,II372,P_6,Z,II487,II489,II127_1;
 int II127_2,II131_1,II131_2,II279_1,II279_2,II283_1;
 int II283_2,II497_1,II500_1,II500_2,II504_1,II504_2;
 int II508_1,II508_2,II512_1,II512_2,II135_1,II135_2;
 int II287_1,II287_2,II5,II92,II96,II100,II104,II157;
 int II244,II248,II252,II256,II314,II316,II317,II341;
 int II350,II374,II378,II409,II495,II1_1,II46,II47;
 int II109,II113,W,II198,II199,II261,II265,P_2,II347;
 int P_5,P_7,P_8,II406,II446,II484,II488,II490,II494;
// input 19, output 10
  int ag[19],od[10],ii,m1=1;
 *gerai=1;
 for(ii=0;ii<in;ii++)
  if(*(gg+ii)=='0')
   *(ag+ii)=0;
  else
   *(ag+ii)=1;

for(ii=0;ii<25;ii++){
 *(od+5)=m1-II104;
 II50=m1-II92;
 II40=m1-*(ag+0);
```

```
II41=m1-*(ag+11);
II42=m1-*(ag+12);
II43=m1-*(ag+13);
II44=m1-*(ag+14);
*(od+2)=m1-II46;
*(od+3)=m1-II47;
*(od+9)=m1-II256;
II202=m1-II244;
II192=m1-II1_1;
II193=m1-*(ag+15);
II194=m1-*(ag+16);
II195=m1-*(ag+17);
II196=m1-*(ag+18);
*(od+6)=m1-II198;
*(od+7)=m1-II199;
II307_1=m1-II341;
II318=m1-II341;
II309=m1-*(ag+14);
II310=m1-*(ag+13);
II311=m1-*(ag+12);
P_1=m1-II314;
P_3=m1-II316;
P_4=m1-II317;
II368=m1-*(ag+11);
II369=m1-*(ag+18);
II370=m1-*(ag+17);
II371=m1-*(ag+16);
II372=m1-*(ag+15);
P_6=m1-II374;
*(od+1)=m1-II446;
II487=m1-II488;
II489=m1-II490;
II127_1=II41&*(ag+12)&II109;
II127_2=II96&II113&*(ag+11);
II131_1=II113&II92&*(ag+12);
II131_2=II42&II109;
II279_1=II193&*(ag+16)&II261;
II279_2=II248&II265&*(ag+15);
II283_1=II265&II244&*(ag+16);
II283_2=II194&II261;
II497_1=P_8&*(ag+2);
II500_1=P_5&*(ag+5);
II500_2=P_2&*(ag+8);
II504_1=P_3&*(ag+7);
II504_2=P_4&*(ag+6);
II508_1=*(ag+0)&*(ag+10);
II508_2=P_1&*(ag+9);
II512_1=P_6&*(ag+4);
II512_2=P_7&*(ag+3);
II135_1=II43|II104;
II135_2=*(ag+13)|II100;
II287_1=II195|II256;
II287_2=*(ag+17)|II252;
*(od+4)=m1-(II135_1&II135_2);
II92=m1-(*(ag+13)&*(ag+14));
II96=m1-(*(ag+12)&II50);
II100=m1-(*(ag+14)&II113);
II104=m1-(II44&II113);
*(od+8)=m1-(II287_1&II287_2);
II244=m1-(*(ag+17)&*(ag+18));
II248=m1-(*(ag+16)&II202);
```

```
 II252=m1-(*(ag+18)&II265);
 II256=m1-(II196&II265);
 II314=m1-(*(ag+0)&*(ag+14));
 II316=m1-(*(ag+12)&II347);
 II317=m1-(*(ag+11)&II318);
 II341=m1-(II311&II347);
 II350=m1-(*(ag+0)&II309);
 II374=m1-(*(ag+17)&II406);
 II378=m1-(II406&II370);
 II409=m1-(II368&II307_1);
 II495=m1-(II484&II494);
 II1_1=m1-(II41|II96);
 II46=m1-(II127_1|II127_2);
 II47=m1-(II131_1|II131_2);
 II109=m1-(II43|II100);
 II113=m1-(*(ag+1)|II40);
 *(od+0)=m1-(II193|II248);
 II198=m1-(II279_1|II279_2);
 II199=m1-(II283_1|II283_2);
 II261=m1-(II195|II252);
 II265=m1-(*(ag+1)|II192);
 P_2=m1-(II310|II350);
 II347=m1-(*(ag+13)|II350);
 P_5=m1-(II369|II409);
 P_7=m1-(II371|II378);
 P_8=m1-(*(ag+16)|II378|II372);
 II406=m1-(II409|*(ag+18));
 II446=m1-(II495|II487|II497_1);
 II484=m1-(II500_1|II500_2);
 II488=m1-(II504_1|II504_2);
 II490=m1-(II508_1|II508_2);
 II494=m1-(II512_1|II512_2|II489);
}
 for(ii=0;ii<on;ii++)
  if(*(od+ii)==0)
   *(og+ii)='0';
    else
   *(og+ii)='1';
}
```