KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Olga Otčeskich

PASKIRSTYTŲJŲ SISTEMŲ AGREGATINIŲ SPECIFIKACIJŲ VALIDAVIMAS ANALIZUOJANT BŪSENŲ PASIEKIAMUMĄ

Magistro darbas

Vadovas

prof. habil. dr. H. Pranevičius

KAUNAS, 2005

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
prof. H. Pranevičius
2005-05-23

PASKIRSTYTŲJŲ SISTEMŲ AGREGATINIŲ SPECIFIKACIJŲ VALIDAVIMAS ANALIZUOJANT BŪSENŲ PASIEKIAMUMĄ

Informatikos mokslo magistro baigiamasis darbas

Kalbos konsultantė
Lietuvių kalbos katedros lekt.
dr. J. Mikelionienė
2005-05-23

Vadovas
prof. H. Pranevičius
2005-05-23

Recenzentas
doc. E. Bareiša
2005-05-23

Atliko
IFM 9/2 gr. stud.
O. Otčeskich
2005-05-23

KAUNAS, 2005

SANTRAUKA (anglų kalba)

Graph models for reachability analysis of distributed systems' aggregate specifications

SUMMARY

The problem of analyzing concurrent systems has been investigated by many researchers, and several solutions have been proposed. Among the proposed techniques, reachability analysis—systematic enumeration of reachable states in a finite-state model—is attractive because it is conceptually simple and relatively straightforward to automate and can be used in conjunction with model-checking procedures to check for application-specific as well as general properties.

The system validation problem considered here is the problem of verifying that the original specification is itself logically consistent. If, for instance, the specification has a design error, an implementation is expected to *pass* a conformance test if it contains the same error. A validation for the logical consistency of the system, however, must reveal the design error.

An automated analysis of all reachable states in a distributed system can be used to trace obscure logical errors that would be very hard to find manually. This type of validation is traditionally performed by the symbolic execution of a finite state machine model of the system studied.

The author presents an overview of the existing validation techniques and methods. Specified and analyzed systems are presented as reachable state graph. The implementation of the aggregate specifications validation system is also presented.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

1. ĮVADAS

1.1. Pratarmė

Didelės apimties taikomosios programos, ypač apimančios visą kompanijos darbo veiklą, turi būti kur kas daugiau nei programinio kodo modulių rinkinys. Moduliai turi sąveikauti tokiu būdu, kad visa informacinė sistema būtų saugi, greita, patikima dirbant su maksimaliu apkrovimu. Architektūra turi būti tiksliai apibrėžta, kad atradus sistemos klaidą, bet kuris programuotojas sugebėtų ją greitai ištaisyti, net jeigu tikrieji sistemos kūrėjai klaidą taisant nedalyvautų.

Formalieji metodai gali būti naudojami kaip papildomas kokybės matas, kuris jau pradinėse sistemos kūrimo stadijose gali atskleisti nesuderinamumus, neužbaigtumą ir kitus būsimos sistemos trūkumus. Formalieji metodai ir atitinkama programinė įranga sėkmingai naudojami duomenų perdavimo protokolų kūrimo, techninės bei programinės įrangos (įterptinės/realaus laiko/reikalaujančios aukšto patikimumo sistemos) verifikavimo srityse. Formaliai aprašyti sistemos reikalavimai bei sukurti prototipai pagerina bendras sistemos veikimo charakteristikas.

Tačiau netgi formalus būsimos sistemos aprašymas negarantuoja kuriamo produkto korektiškumo, taigi validavimas yra tiesiog būtinas.

Darbo aktualumas. Techniškai sudėtingų, paskirstytų, realaus laiko ir kitų kritinių sistemų valdymas reikalauja efektyvios programinės įrangos, kuri užtikrintų sistemoje vykstančių procesų griežtą kontrolę. Todėl jau ankstyvosiose sistemų (kaip programinės taip ir techninės įrangos) kūrimo stadijose reikia išanalizuoti galimas sistemos trūkumus ir juos pašalinti.

Daug žadantis būdas pasiekti aukštą sudėtingų sistemų patikimumą yra formaliųjų metodų naudojimas. Formalieji metodai yra rinkinys matematiškai pagrįstų specifikavimo kalbų bei specializuotų programinių įrankių, skirtų specifikuoti bei verifikuoti sistemų modelius. Pagrindinis formaliųjų specifikacijų privalumas yra galimybė jas automatizuoti.

Darbo tikslas. Sistemos korektiškumui tikrinti naudojami analitiniai bei imitaciniai metodai. Tačiau, šie du būdai reikalauja skirtingų tos pačios sistemos specifikacijų: analitiniam tikrinimui nereikia laikinių sistemos charakteristikų. Dviejų skirtingų specifikacijų naudojimas apsunkina analizės procesą ir padidina klaidų atsiradimo tikimybę pereinant nuo vienos specifikacijos prie kitos. Dėl šios priežasties aktualu yra turėti integruotos analizės sistemą, kuri veiktų vieningos specifikacijos bazėje.

Iškelti uždaviniai:

- Apibrėžti tikrinamas sistemos modelio savybes: saugumo, gyvybingumo, logines.
- Apibrėžti matematines schemas, taikomas formaliesiems modeliams kurti.
- Išanalizuoti pasiekiamumo analizės algoritmus.
- Sukurti validavimo posistemį.
- Atlikti eksperimentus su sukurtu validavimo įrankiu.

Mokslinis darbo elementas.

- Automatizuoto validavimo taikymas agregatinėms specifikacijoms tikrinti.
- Sistemos modelio analizė atliekama vieningos agregatinės specifikacijos bazėje.

Praktinė darbo vertė. Sukurtas agregatinių specifikacijų validavimo įrankis VALSYS leidžia automatizuotai vykdyti pasiekiamumo analizę.

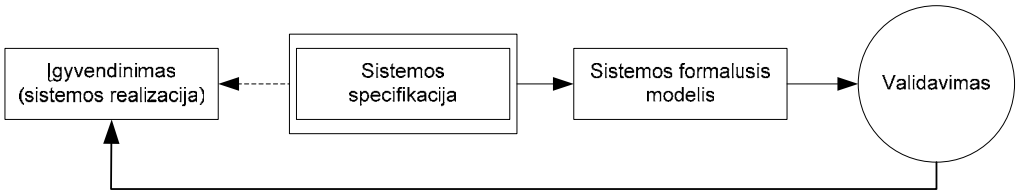
1.2. Dokumento struktūra

Šiame darbe aprašyti pasiekiamų būsenų aibės konstravimo algoritmai, pasiekiamumo analizės algoritmai, bei sukurtas agregatinių specifikacijų validavimo įrankis. Antrame skyriuje aprašomos matematinės sistemų modelių schemos. Trečias skyrius skirtas validavimo įrankio VALSYS architektūrai aprašyti. Ketvirtame ir penktame skyriuose pateikiami eksperimentų bei validavimo įrankio kokybės tyrimo rezultatai. Dokumento pabaigoje pateikiamos darbo išvados.

2. ANALITINĖ DALIS

2.1. PROBLEMA

1 pav. iliustruoja sistemos validavimo poreikį kuriant programinę įrangą.



1 pav. Sistemų specifikacijų validavimo vieta programinės įrangos kūrimo procese

Nagrinėjama validavimo problema yra sistemos specifikacijos logiškumo tikrinimas [16]. Pavyzdžiui, jeigu sistemos pradiniame modelyje (specifikacijoje) yra klaida, tai sistemos prototipas sėkmingai praeis atitikimo testavimą turėdamas tą pačią klaidą. Atitikimo testas išryškins problema tik tuo atveju, jeigu sukurtas prototipas neatitiks savo specifikacijos – o tai prieštarauja programinės įrangos projektavimo principams.

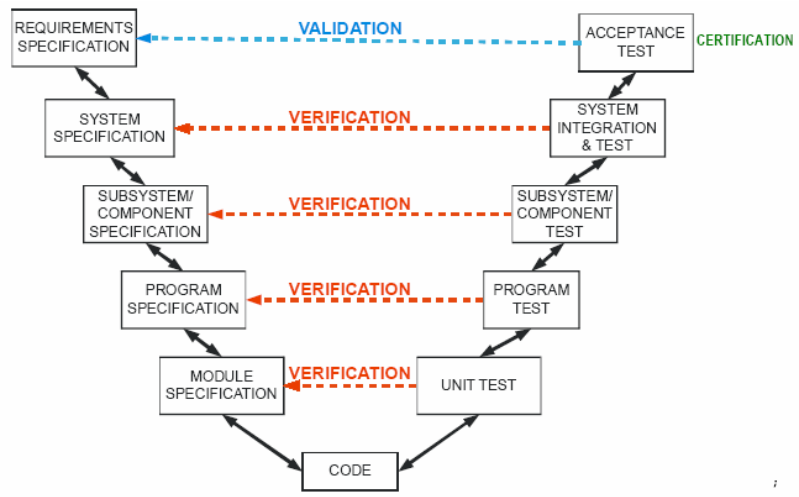
Sistemos specifikacijos validavimas turi užtikrinti specifikacijos logiškumą ir neprieštaringumą. Tam tikslui sukuriamas sistemos formalusis aprašymas (baigtinio automato pagrindu), kuris visapusiškai tiriamas.

2.2. SISTEMŲ SPECIFIKACIJŲ VALIDAVIMO UŽDAVINYS

Validavimas kaip ir verifikavimas yra skirtas būsimas sistemas tikrinti. Barry Boehm taip apibrėžė šiuos procesus: „Verifikavimo metu patikriname, *ar teisingai* kuriame produktą, o validavimo metu – *ar teisingą* produktą kuriame“ [17].

Sistemos validavimo stadijoje turime įsitikinti, kad kuriamas produktas atitinka vartotojo reikalavimus. Validavimas užima svarbią vietą sistemos gyvavimo cikle, bet dažniausiai šis etapas būna mažiausiai struktūriškai apibrėžtas [11].

Sistemos kūrimo procesą galima pavaizduoti *V* modeliu (2 pav.), iš kurio matyti, kad validavimą galima nagrinėti ne tik kaip sistemos modelio tikrinimą, bet ir kaip vieną iš testavimo rūšių – priėmimo testavimą. Specifikacijų validavimo fazė yra kritiška sistemos kūrimo procese, kadangi jinai užbaigia sistemos projektavimo etapą.



2 pav. Validavimo rolė programinės įrangos kūrimo cikle

Taigi, pateiksime sistemos specifikacijos validavimo apibrėžimą.

Specifikacijos validavimas – sistemos (arba jos komponento) specifikacijos vertinimo procesas, kai norima įsitikint, kad sukurtas galutinis produktas atitiks jam keliamus reikalavimus.

Šiame darbe agregatinių specifikacijų validavimo uždavinys bus sprendžiamas analizuojant sistemos pasiekiamų būsenų grafą.

Kiekviena pasiekiamų būsenų grafo viršūnė atitinka sistemos būseną, iš kurių tik viena būsena gali būti *dabartinė*. Kiekviena būsena pilnai specifikuojama reikšmių vektoriumi, kurį sudaro sistemos kintamieji. Būsenos vektoriaus kintamųjų reikšmės priklauso nuo specifikuojamos sistemos prigimties.

Sistemos pasiekiamų būsenų grafas gali būti analizuojamas tokiais aspektais:

- *Būsenos pasiekiamumas*: ar galima iš pradinės būsenos pasiekti galinę būseną.
- *Būsenos koordinačių apribojimas* (angl. *boundedness*): sistemos kintamųjų verčių patekimas į užduotus rėžius.
- *Aklaviečių paieška:* ar yra izoliuotų būsenų.
- *Ciklų paieška*: ar sistema nepatenka į neprogresyvų ciklą, kuriame kartojasi tam tikrų įvykių seka.
- *Invariantų tikrinimas.*

2.3. FORMALIZAVIMO SCHEMOS

Formalizavimas – programinės įrangos matematika

2.3.1. Baigtiniai automatai

2.3.1.1.Teorinė bazė

Baigtinių automatų metodas yra plačiai naudojamas kuriant duomenų perdavimo protokolų specifikacijas [7].

Formaliai baigtiniai automatai aprašomi penkiais parametrais (S, Σ, T, s, A), kur

- S – baigtinė aibė būsenų;
- Σ – baigtinė abėcėlė (aprašo įėjimo bei išėjimo simbolius);
- T – transakcijų funkcija ($T : S \times \Sigma \rightarrow S$);
- s – pradinė būsena ($s \in S$);
- A – kitos būsenos, į kurias gali pereiti automatas ($A \in S$)

Kadangi visos galimos automato būsenos turi būti aprioriškai užduotos, baigtinių automatų metodas tinka tiktai sistemoms, kurių veikimas gali būti suskaidytas į elementarius žingsnius su nedideliu skaičiumi galimų būsenų.

Priimta, kad baigtinio automato veikimas aprašomas transakcijų lentelėmis, kur stulpeliai atitinka signalus ir būsenas, o eilutės – operacijas.

1 lentelė. Baigtinio automato transakcijų lentelė

Condition		Effect	
Current State	In	Out	Next State
q0	–	1	q2
q1	–	0	q0
q2	0	0	q3
q2	1	0	q1
q3	0	0	q0
q3	1	0	q1

Kiekviena kontrolinė mašinos būsena aprašoma operacijų aibe. Pateiktoje lentelėje turime keturias kontrolines būsenas (q0, q1, q2, q3). Kiekviena transakcijos operacija sudaryta iš keturių dalių, kurias atitinka lentelės stulpeliai (iš kairės į dešinę):

- Esama kontrolinė mašinos būsena;
- Įėjimo signalo reikšmė;
- Išėjimo signalo reikšmė;

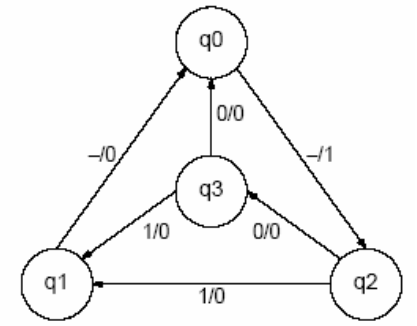
- Sekanti būsena, į kurią pereina mašina.

Tradiciniame baigtinio automato modelyje, mašinos „vidinę aplinką“ sudaro dvi nesikertančios ir baigtinės aibės: įėjimo signalų ir išėjimo signalų. Kiekvienas signalas turi sutartinę, tačiau būtinai baigtinę, reikšmių sritį. Brūkšnys įėjimo signalų lentelės stulpelyje reiškia, kad išėjimo signalo reikšmė nepriklauso nuo įėjimo signalo.

Kiekvienoje mašinos būsenoje egzistuoja nulis arba daugiau signalų perdavimo operacijų. Jeigu jų nėra (t. y. nulis), laikoma kad esama mašinos būsena yra darbo pabaigos būsena. Jeigu turime tiktai vieną galiojančią transakciją, jinai bus atlikta. Mašinos, kurios pačios gali „apsispręsti“ situacijoje, kai transakcijų yra daugiau nei viena, vadinamos *neapibrėžtomis*.

2.3.1.2. Grafinė notacija

Lentele aprašytos mašinos veikimą lengviau suprasti ir įsivaizduoti turint jos grafinį pavidalą. Viršūnės atitinka kontrolines būsenas, o kryptingi lankai – transakcijas.



3 pav. Baigtinio automato veikimo grafinis vaizdavimas

Pateiktas modelis yra klasikinis teorijoje. Praktikoje, atmetus baigtinumo sąlygą, gautume Tiuringo mašiną, kuri yra labiausiai apibendrintas baigtinių automatų modelis.

Tiuringo tezė: bet kuris procesas, kurį galima aprašyti veiksmų seka, gali būti realizuotas baigtiniu automatu (Tiuringo mašina).

2.3.2. Atkarpomis tiesiniai agregatai

Atkarpomis tiesinių agregatų formalizmo pavadinimas ir jo taikymas sudėtingų sistemų modeliavimui pirmą kartą buvo pasiūlytas 1971 metais N.Buslenko [6].

Šiandien agregatų [8,9] notacija yra naudojama įvairioms sistemoms projektuoti ir imitaciniam modeliavimui. Agregatinis specifikavimo metodas gali būti sėkmingai panaudotas sistemos korektiškumo analizei ir jos funkcionavimui tikrinti. Teorinis agregatinio metodo pagrindas yra atkarpomis tiesiniai Markovo procesai. Klasikinis Markovo proceso

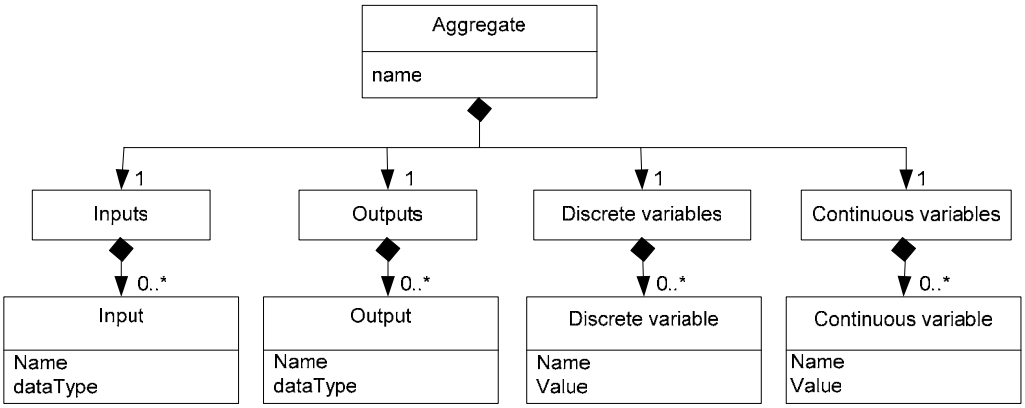
pavyzdys yra klientų aptarnavimo sistema, kur naujo kliento pasirodymas ir jo aptarnavimo laikas yra eksponentiniai atsitiktiniai dydžiai.

2.3.2.1.Teorinė bazė

Agregatinis metodas laiko sistemą kaip sąveikaujančių posistemių visuma, kur kiekviena posistemė aprašomas kaip objektas, turintis:

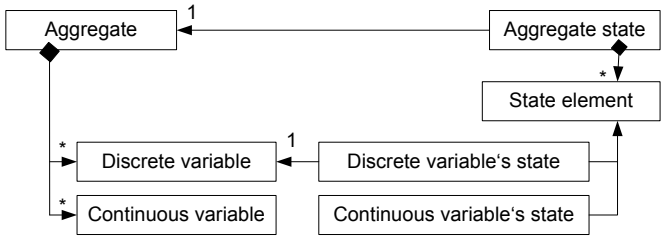
- būsenų aibę *Z*,
- įėjimo signalų aibę *X,*
- išėjimo signalų aibę *Y*,
- vidinių įvykių aibę E'' ,
- išorinių įvykių aibę E' ,
- perėjimo operatorių *H,*
- išėjimo operatorių *G*.

Sistemos funkcionavimas apibrėžiamas laiko momentais *t* ($t \in T$). Tokiu būdu būsenos $z \in Z$, įėjimo signalai $x \in X$ ir išėjimo signalai $y \in Y$ yra laikomi laiko funkcijomis. Taip pat yra aprašomi perėjimo ir išėjimo operatoriai: *H* operatorius nusako naują agregato būseną, o *G* - aprašo išėjimo signalą (išėjimo signalai gali būti generuojami tiktai kai sistemoje įvyksta įvykis).



4 pav. Agregato duomenų struktūrų klasių diagrama

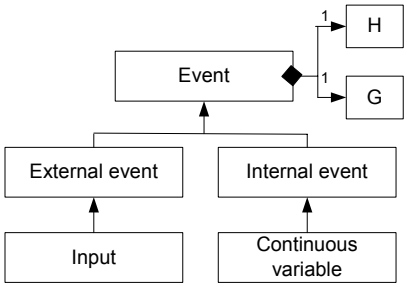
Agregato būseną sudaro dvi komponentės: $z(t) = (\upsilon(t),\ z_{\upsilon}(t))$, čia $\upsilon(t)$ yra diskrečioji būsenos komponentė, sauganti suskaičiuojamas kintamųjų reikšmes (pvz., *Q* – agregato įėjime pasirodžiusių signalų eilės ilgis); $z_{\upsilon}(t)$ - tolydžioji būsenos komponentė, aprašanti kintančias laike koordinates $z_{\upsilon 1}(t), z_{\upsilon 2}(t), \text{K}\ , z_{\upsilon k}(t)$. Kartu diskrečioji ir tolydžioji komponentės sudaro būsenos vektorių.



5 pav. Agregato būsenos vektoriaus struktūra

Agregato būsena gali pasikeisti dėl dviejų priežasčių: 1) kai agregato įėjime pasirodo signalas, 2) kai tolydi komponentė įgyja apibrėžtą reikšmę.

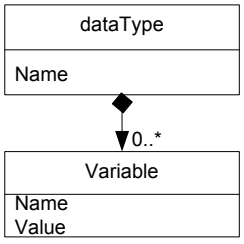
Agregato funkcionavimas nagrinėjamas laiko momentais $T = \{t_0, t_1, ..., t_m, ...\}$, kiekvienu iš jų gali įvykti vienas ar daugiau įvykių. Įvykiai būna vidiniai $E'' = \{e''_1, e''_2, ..., e''_f\}$ arba išoriniai $E' = \{e'_1, e'_2, ..., e'_N\}$. Kiekvienas vidinis įvykis turi valdančiąją seką, kuri nusako įvykio pasirodymo periodiškumą.



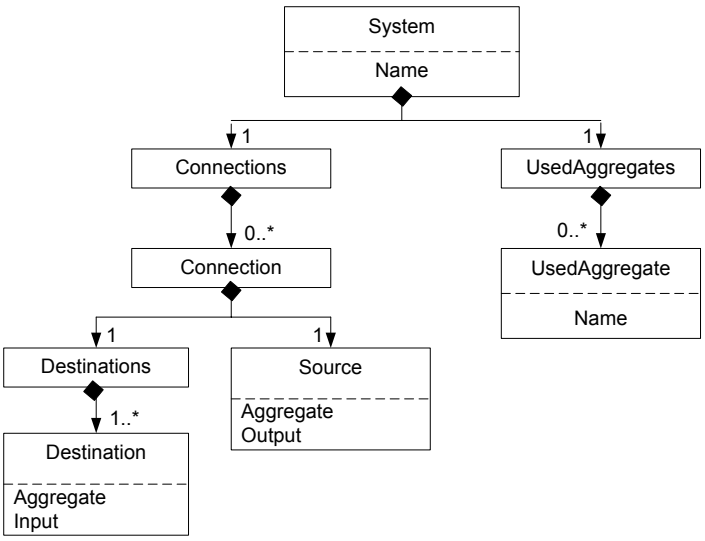
6 pav. Agregato įvykių apdorojimo schema

Analizuojant agregatų veikimą paprastai yra skaičiuojamos tokios charakteristikos: vidutinis signalo (paraiškos) būvimo laikas eilėje / sistemoje, vidutinis signalų eilės ilgis.

Modeliuojama sistema gali būti sudaryta iš daugiau nei vieno agregato. Tokioje sistemoje agregatai bendrauja tarpusavyje siųsdami signalus ryšio kanalais:



7 pav. Agregato įėjimo/išėjimo signalo struktūra



8 pav. Agregatų sistemos schema

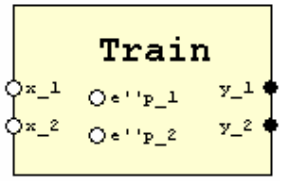
Agregato sampratą galima palyginti su procesų algebros *proceso* samprata [10]. Procesų algebra nagrinėja bet kokią sistemą tokiame aukštame specifikavimo lygyje, kad netgi neatsižvelgiama į pačios sistemos prigimtį. Sistema aprašoma procesais – matematiniais objektais, turinčiais savo apibrėžimo sritį. Tarpusavyje procesai gali būti jungiami operatoriais ir sudaryti nuoseklias arba lygiagrečias kompozicijas.

Nesunku pastebėti, jog tie patys principai taikomi ir atkarpomis tiesinių agregatų formalizme: kiekvienas agregatas yra tarsi atskiras sistemos procesas, o išėjimo operatoriaus *H* pagalba agregatai jungiami į kompozicijas.

2.3.2.2.Grafinė notacija

Agregatinės specifikacijos vaizduojamos grafinėmis diagramomis, kas palengvina jų sudarymą. Agregatinės specifikacijos diagramose vaizduojami tik dviejų tipų elementai: agregatai ir ryšiai tarp jų.

Agregatas vaizduojamas stačiakampiu, kuriam priskiriamas vardas. Vidiniai įvykai atvaizduojami apskritimu stačiakampio viduje, šalia užrašant atitinkamos operacijos pavadinimą. Agregato išėjimai žymimi tamsiu apskritimu, šalia užrašomas išėjimo signalas. Įėjimai žymimi tuščiaviduriu apskritimu, šalai užrašomas įėjimo signalas.



9 pav. Agregatas

9 pav. pateiktas agregato pavyzdys.

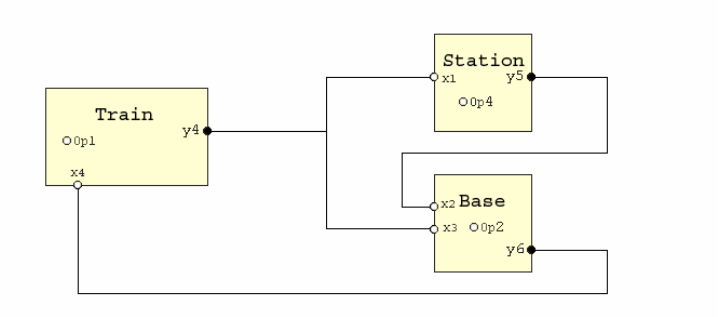
Pavadinimas: Train

Įėjimo signalų aibė: X = { x_1, x_2 };

Išėjimo signalų aibe: Y = { y_1, y_2 };

Vidiniai įvykai: e''p_1, e''p_2.

Kaip buvo rašoma skyriuje 2.2.2.1, agregatus galime jungti į kompoziciją, tokiu būdu vaizduojant pilną sistemą (10 pav.)



10 pav. Agregatų diagrama

Agregatus jungia signalų perdavimo kanalai. Tai yra daug patogiau negu naudojant lenteles, aprašančias sujungimo schemą. Kanalas jungia vieno agregato išėjimą su kito agregato įėjimu. Sujungimo linijos neturi eiti per agregatą vaizduojančio stačiakampio vidų. Kanalai taip pat gali išsišakoti ir tas pats kanalas išeidamas iš vieno agregato gali būti prijungtas prie kelių agregatų įėjimų.

2.3.3. SDL

SDL (*Specification and Description Language*) atsirado 1972 metais, kai telekomunikacijų kompanijos (tokios kaip *Bellcore, Ericsson, Motorola*) inicijavo dėl telekomunikacijų procesų aprašymo standartizavimą. Nuo to laiko pasirodė šešios SDL kalbos versijos.

Šiandien SDL yra pilnai išbaigta modeliavimo kalba, skirta sudėtingoms sistemoms (realaus laiko, paskirstytosioms sistemoms, įvykių valdomoms sistemoms, kt.) aprašyti.

SDL kalbos charakteristikos:

- Standartas – SDL yra standartizuota (ITU), naudojama visame pasaulyje.
- Formalizmas - SDL-PR tekstinė notacija skirta formaliam sistemos aprašymui sudaryti ir užtikrina specifikacijos tikslumą, vienareikšmiškumą, aiškumą bei vientisumą.
- Grafinė notacija – SDL-GR grafiškai pateikia sistemos specifikaciją.
- Objektiškai orientuota – SDL yra pilnai objektiškai orientuota kalba, palaikanti sistemos dinamiką.

Iš SDL sistemos aprašymo gali buvo sugeneruotas žemesnio lygio kalbos kodas (pvz., C/C++). Tai reiškia, jog vykdomasis kodas gali būti sukurtas automatiškai, kas sutrumpina programinio produkto kūrimo laiką ir mažina klaidų tikimybę. Vaizdinga grafinė specifikacijos notacija pati savaime yra kuriamo produkto dokumentacija.

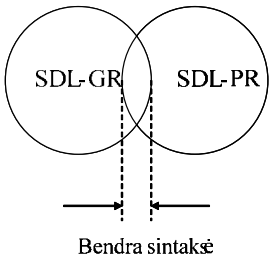
2.3.3.1. Teorinė bazė

Teorinis SDL pagrindas yra baigtinių automatų rinkinys, kurie veikia lygiagrečiai [13]. Automatai yra nepriklausomi vieni nuo kitų ir komunikuoja tarpusavyje siusdami diskrečius signalus.

SDL sistemą sudaro tokios komponentės:

- Struktūra – sistema, blokas, procesas ir procedūrų hierarchija.
- Komunikacija – parametrizuoti signalai ir jų perdavimo kanalai.
- Elgsena – procesai.
- Duomenys – abstraktūs duomenų tipai.
- Paveldimumas – ryšių ir specializacijų aprašymas.

Kaip jau buvo minėta, SDL sudaro dvi dalis – grafinė SDL-GR ir tekstinė SDL-PR.



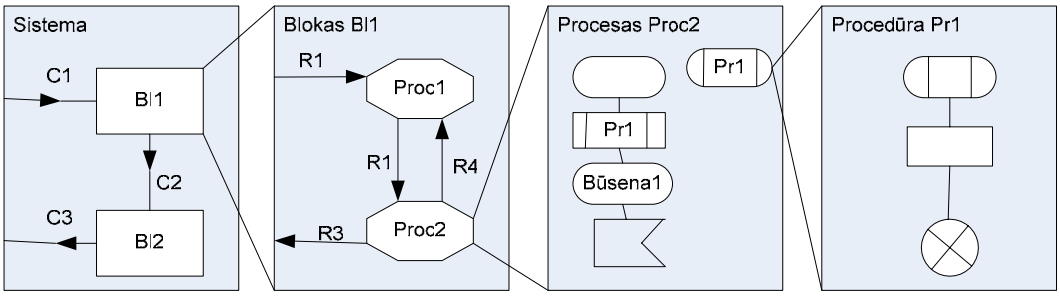
11 pav. SDL kalbos sandara

Tekstinė notacija apsirašo CIF (*Common Interchange Format*) formate, kuris reglamentuojamas ir standartizuojamas ITU. Grafinė notacija skirta vartotojui, tačiau irgi turi tekstinės notacijos elementus, tam kad duomenų ir signalų SDL-GR specifikacija būtų identiška tekstinei SDL-PR specifikacijos daliai.

2.3.3.2.Grafinė notacija

SDL modelio struktūrą sudaro keturi hierarchijos lygiai:

- Sistema
- Blokas
- Procesas
- Procedūra



12 pav. SDL modelio struktūra

Sistemos hierarchinės struktūros sudarymas turi tokius tikslus:

- Nereikalingos informacijos paslėpimas
- Natūralios funkcijų hierarchijos siekimas
- Racionalaus dydžio (žmogaus proto aprėpiamo) modelio kūrimas
- Siekti asociacijos su programinės ir techninės įrangos logine struktūra
- Egzistuojančių specifikacijų pakartotinis panaudojimas

Kiekvienas SDL procesas aprašomas hierarchiniu būsenų automatu, kur procedūra realizuoja vidinių būsenų automatą. Procedūros gali būti kaip vietinės (angl. *private*) taip ir viešosios (angl. *public*). Be to SDL palaiko distancinės procedūros (angl. *remote procedure*) paradigmą, kai vieno proceso ribose sukurta procedūra gali būti vartojama kito proceso kontekste.

Logiškai sugrupuoti procesai sudaro blokus. Paskutiniai gali sudaryti rekursinę struktūrą, kuri vaizduoja visą sistemą kaip posistemių aibę.

2.3.4. Formalizavimo schemos parinkimas

Visos trys anksčiau aprašytos formalizavimo notacijos remiasi baigtinių automatų teorija, tik kiekviena jų turi savo specifinių savybių.

Agregato ir baigtinio automato struktūros yra panašios: abiem atvejais sistemos būsena apsirašo koordinačių vektoriumi. Tačiau baigtiniai automatai neturi įvykius valdančiųjų sekų, t. y. perėjimas į kitą būseną priklauso nuo įėjimo bei išėjimo signalų ir niekaip nesusijęs su laiku. Tuomet agregate kiekvienas vidinis įvykis turi valdančiąją seką, kuri nusako įvykio pasirodymo periodiškumą.

Dabar atsakysime į klausimą, kodėl šiame darbe kalbėsime apie atkarpomis tiesinių agregatų formalizmo taikymą sistemų specifikavimui:

- Pagrindinis apsisprendimo faktorius – projekto autorių ir vadovo turima patirtis [14].
- Kadangi agregatinių specifikacijų integruotos analizės sistema veikia vieningos specifikacijos bazėje, tai reiškia, jog imitacinio modeliavimo tikslams mums būtina turėti laiko parametrą.
- Reikia pastebėti, kad agregatinis metodas yra pranašesnis už baigtinių automatų metodą, kadangi leidžia nustatyti sistemos objekto vidinių įvykių priežastis. Ši savybė gali būti naudojama pasiekiamumo analizėje - jinai leidžia išdirbti algoritmus, skirtus visiems galimiems įvykiams ir su jais susijusiems perėjimams (kurie atsiranda sistemos veikimo metu) nustatyti.
- Agregatinio metodo universalumas - kadangi validavimui nenaudojame laiko sąvokos, o pasiekiamų būsenų aibė yra baigtinė, validavimo tikslams galime traktuoti sistemoje vykstančius procesus kaip aibę baigtinių automatų.

2.4. PASIEKIAMŲ BŪSENŲ GRAFO ANALIZĖ

Tam, kad galėtume spręsti apie sistemos (modelio) veikimo korektiškumą ir atitikimą specifikacijai, turime apsibrėžti svarbiausias modelio charakteristikas (savybes) ir būdus jiems tikrinti.

Aprašysime kelias svarbias sąvokas, kuriomis bus operuojama toliau:

- Globali sistemos būsena – kontrolinis proceso taškas, charakterizuojamas visa eile parametrų (būsenos vektoriumi).
- Pasiekiama būsena – t. y. tokia būsena, iki kurios egzistuoja vykdomoji seka iš pradinės būsenos.
- Sistemos savybės – korektiškumo kriterijai, išreikšti loginėmis sąlygomis, kurios turi būti išpildytos sistemai būnant vienoje ar kitoje būsenoje.

2.4.1. Sistemos savybės

Sistemos savybes galima nagrinėti kaip reikalavimų sistemai visumą ir suklasifikuoti jas tokiu būdu [3]:

- bendrosios ir specifinės;
- dalinio korektiškumo ir baigtinumo;
- sintaksės ir semantikos;
- statikos ir dinamikos;
- saugumo ir gyvybingumo.

Bendrosios sistemos savybės:

- Ribotumas – signalų (pranešimų, paketų) skaičius kanale tarp objektų neturi viršyti šio kanalo pralaidumo.

Formaliai ši savybė užrašoma taip:

$$S = \left\{ Z_i \mid N_i \in \overset{k}{\underset{j=1}{\mathrm{X}}} \left[a_j, b_j \right] \right\},$$

kur N_i – būsenos vektoriaus diskrečiosios komponentės reikšmės globalioje sistemos būsenoje Z_i,

$\left[a_j, b_j \right]$ - j-osios diskrečiosios koordinatės užduotas kitimo intervalas,

k – diskrečiųjų koordinačių vektoriaus dydis.

Tokiu būdu, ribotumo savybė aprašo globalių sistemos būsenų aibę, kurių diskrečiųjų koordinačių reikšmės priklauso užduotai reikšmių kitimo sričiai.

- Sistemos specifikacijos neperteklišKumas – neatliekamos operacijos su signalais neaprašomos.

- Sistemos specifikacijos pilnumas – visos leistinos operacijos su signalais yra aprašytos.
- Baigtinumas – sistema būtinai pasiekia iš anksto užduotą darbo pabaigos būseną.

Specifinės sistemos savybės:

- Korektiškas operacijų vykdymas, priklausomai nuo sistemos paskirties.
- Sistemos darbo atsistatymas įvykus klaidai.

Dalinis sistemos veikimo korektiškumas reiškia, kad sistema garantuoja paslaugų teikimą tik esant sąlygai, jog sistemos darbas būtinai pasibaigs iš anksto užduotoje darbo pabaigos būsenoje. Taigi, ši savybė aprašo sąlyginį sistemos veikimą. Dalinio korektiškumo ir baigtinumo savybės kartu sudaro pilno korektiškumo savybę.

Saugumo savybės aprašo situacijas, kurios *niekada* neturėtų įvykti (arba priešinga formuluotė – *visada* turi įvykti), gyvybingumo savybės aprašo tai, kas *būtinai* turi įvykti [2].

Sistemos saugumo savybės:

- Statinių aklaviečių nebuvimas - sistema neturi patekti į situaciją (būseną) be išeities.
- Dinaminių aklaviečių nebuvimas - sistema neturi patekti į neprogresyvius ciklus, kur vyksta beprasmis apsikeitimas tais pačiais signalais be galimybės pereiti į kitas būsenas.

Sistemos gyvybingumo savybės:

- Neefektyvūs ciklai – dalinis dinaminių aklaviečių atvejis. Atsiranda neteisingai uždavus sistemos veikimo parametrus. Neefektyvūs ciklai turi ryšį su kitomis būsenomis, tačiau sulėtina sistemos veikimo tempą.
- Sistemos darbo baigtinumas - garantija, jog sistema pasieks užduotą darbo pabaigos (arba darbo ciklo pabaigos) būseną.

2.4.2. Sistemos pasiekiamų būsenų aibės konstravimas

Dirbant su modeliais, kurių pasiekiamų būsenų grafus gali sudaryti šimtai tūkstančių ir net milijonai viršūnių – iš kurių ne visos gali būti pasiekiamos iš pradinės viršūnės, - sistemos analitikui yra tikslinga išskirti iš pasiekiamų būsenų grafo būsenų aibę, kuri dalyvaus pasiekiamumo analizėje.

Sistemos modelio pasiekiamų būsenų aibės išskyrimo algoritmų tikslas yra sugeneruoti ir patikrinti visas paskirstytos sistemos būsenas, kurios gali būti pasiektos iš pradinės būsenos [1].

Egzistuoja trys pagrindiniai pasiekiamų būsenų aibės konstravimo algoritmai (priklausomai nuo validuojamos sistemos sudėtingumo):

a) pilnos paieškos (sistemoms turinčioms iki 10^5 būsenų);

b) kontroliuojamos dalinės paieškos (sistemoms turinčioms iki 10^8 būsenų);

c) atsitiktinio modeliavimo (sistemoms turinčioms daugiau nei 10^8 būsenų).

Jau pasiekiamų būsenų paieškos metu gali būti išspręsti kai kurie validavimo uždaviniai (aklaviečių nebuvimo tikrinimas, invariantų tikrinimas).

2.4.2.1. Pilna būsenų paieška

Išsami modelio analizė reiškia, kad bus išnagrinėtos visos įmanomos situacijos, kurios gali įvykti vykdant paskirstytą algoritmą, kurį sudaro keli tarpusavyje susiję procesai. Kiekvienas iš tokių procesų gali būti laikomas baigtiniu automatu [4].

Šis algoritmas yra paprasčiausias iš trijų išvardytų anksčiau. Jis atlieka išsamiausią analizę, tačiau gali būti pritaikytas tiktai žemiausios klasės sistemoms. Tačiau, jeigu pilnos paieškos algoritmas išeina už savo ribų, jis tampa nekontroliuojamos dalinės apieškos algoritmu, o analizės kokybė žymiai krenta.

Išsami pasiekiamumo analizė turi nustatyti būsenas, kurios yra pasiekiamos, o kurios ne. Kiekviena pasiekiama būsena ir pasiekiamų būsenų seka gali būti patikrinta pagal kokį nors korektiškumo kriterijų (pvz., visiems duomenų perdavimo protokolams turi būti išpildyta aklaviečių nebuvimo sąlyga). Daugelyje atvejų reikalavimai sistemai gali būti aprašyti sistemos invariantais, kurie galėtų būti patikrinti loginiu testu kiekvienoje pasiekiamoje sistemos būsenoje.

Algoritmo privalumas: aptikus klaidą, visada egzistuoja kelias nuo pradinės būsenos iki klaidos.

Algoritmo trūkumas: jo panaudojimas apribotas sistemos sudėtingumo laipsniu.

2.4.2.2. Kontroliuojama dalinė paieška

Jeigu analizuojamų būsenų aibė yra didesnė nei prieinama kompiuterio atmintis gali sutalpinti ir apdoroti, tai pilna paieška sumažėja iki dalinės, be garantijos kad svarbiausios sistemos dalys bus išanalizuotos. Dėl šios priežasties buvo sukurta nauja algoritmų klasė, kuri naudoja dalinės paieškos pranašumus. Tokie algoritmai remiasi prielaida, jog daugelyje atvejų projektuotojus dominanti pasiekiamų būsenų aibė A yra tiktai dalis visų pasiekiamų būsenų R. Taigi, dalinės paieškos tikslas yra:

- Išanalizuoti būsenų aibę A, kurią sudaro M/S būsenų (čia S – atminties talpa, reikalinga vienai būsenai saugoti, M – atminties talpa, skirta vienam baiginiam

automatui. Tokiu būdu mes galime sugeneruoti ir išanalizuoti ne daugiau kaip *M/S* automato būsenų).

- Parinkti būsenas šiai aibei *A* iš pilnos pasiekiamų būsenų aibės *R*, kad būtų išanalizuotos visos pagrindinės sistemos funkcijos.
- Parinkti tokias būsenas aibei *A*, kad *kokybės* paieška (t. y. tikimybė rasti klaidą) būtų geresnę už *stebėjimo sritį A/R*.

Dalinės paieškos algoritmo mechanizmas yra toks pat kaip ir pilnos paieškos, tik skirtumas tas, kad yra analizuojamos ne visos būsenos einančios po duotos pasiekiamos būsenos.

Būdai dalinei paieškai organizuoti:

- *Gilio apribojimas*. Analizuojamoms vykdomosioms sekoms uždedama ilgio riba, tai apriboja paieška iki rezultatyvaus sistemos elgsenų poaibio (pvz., eliminuoja daugkartinių persidengimų atvejus).
- *Išsidėstymo (išsibarstymo) analizė*. Išrenkamos tokios vykdomosios sekos, kurios potencialiai veda į aklavietes. Vienas iš aklavietės atpažinimo požymių yra signalų nebuvimas perdavimo kanaluose.
- *Valdomoji paieška*. Projektuotojas pats pasirenka kitą sistemos būseną (pvz., validavimo įrankio grafinės vartotojo sąsajos pagalba).
- *Tikimybinė paieška*. Pasiekiamos būsenos analizuojamos jų pasirodymo tikimybės mažėjimo tvarka. Pranešimai turi "aukštos" ar "žemos" tikimybės žymeklius, kurie pasitarnauja kaip atrankos kriterijus.
- *Atsitiktiniai rinkiniai*. Nededama pastangų į būsenos tipo prognozavimą (klaida, aklavietė). Ši priemonė vienintelė atitinka visus tris kontroliuojamos dalinės paieškos tikslus.

Plačiau apie šių metodų taikymą ir palyginimą rašoma [16].

2.4.2.3. Sudėtingumo valdymas (validuojant ypač didelius grafus)

1) *Kompozicijos didinimas*: analizuojant paskirstyto algoritmo procesus, kiekvienas procesas yra laikomas baigtiniu automatu. Į automatą įtraukiamos tik pasiekiamos būsenos. Pirmame žingsnyje apjungiame du tokius automatus, prie kurių vėliau bus prijungtas kitas procesas (kitas baigtinis automatas). Projektuoto užduotis yra kiekvieną kartą surasti tokius du procesus, kurių apjungimas duotų didesnį būsenų skaičiaus mažinimo efektą. Mažinimas reiškia vidinių apjungtų procesų būsenų naikinimas – toliau bus nagrinėjamos tiktai išorinės procesams būsenos. Tokiu būdu galima sakyti, kad šis metodas geriausiai veikia glaudžiai tarpusavyje susijusiems procesams (kurie turi daug apsikeitimo pranešimų). Jeigu per klaidą

projektuotojas apjungtų du nesusijusius procesus, rezultate gautųsi būsenų Dekarto sandauga (visos galimos būsenų kombinacijos).

2) *Redukcijos metodas*: modelis gali būti apribotas pagal vieną ar kitą parametrą (eilės ilgis, procesų skaičius, kt.), tokiu būdu mažinant pasiekiamų būsenų kiekį. Aišku tokiu atveju validavimas tampa daliniu, tačiau ypač didelių grafų atveju dažnai tenka rinktis tarp tikrinimo atlikimo greičio ir pilnumo.

IŠVADA: pilnas modelis gali būti analizuojamas dalimis, išskiriant nuo 1 iki *n* pasiekiamų būsenų aibių.

2.4.3. Pasiekiamumo analizė

Pasiekiamumo analizei naudojamos globalios sistemos būsenos, kurios yra atskirų sistemos objektų sąveikos aplinka. Užduota pradinė sistemos būsena aktyvuojama vartotojo komandomis, įėjimo signalais arba pertraukos (angl. *time-out*) pabaiga ir to pasėkoje sistema pereina į kitas, naujas globalias būsenas. Tos kitos būsenos savo ruožtu tai pat yra aktyvuojamos ir procesas kartojasi. Naujų globalių sistemos būsenų generavimas baigiasi tuomet, kai nebegalima suaktyvuoti naujų, skirtingų nuo jau sugeneruotų, būsenų. Visą šį procesą galima pavaizduoti orientuotu grafu, kurio viršūnės atitinka globalias sistemos būsenas, o kryptingi lankai – perėjimus iš vienos būsenos į kitą. Tokiu būdu gautas pasiekiamų būsenų grafas gali būti panaudotas sistemos savybėms tikrinti, kadangi kai kurios jų susijusios su orientuoto grafo struktūra.

Pasiekiamų būsenų grafo analizės metodo privalumas – gali būti lengvai automatizuotas. Trūkumas – globalių būsenų aibė auga eksponentiškai, todėl pasiekiamumo analizė gali būti taikoma tiktai nesudėtingoms sistemoms, arba sudėtingų sistemų dalims, turinčioms baigtinę aibę būsenų.

Analizuojant sistemos pasiekiamų būsenų grafą, galima patikrinti tokias sistemos savybes kaip statinių aklaviečių, neprogresyvių ciklų nebuvimas, ribotumas (angl. *boundedness*, t. y. būsenos vektoriaus reikšmių patekimas į užduotą intervalą), būsenų pasiekiamumą iš pradinės būsenos arba iš bet kurios kitos.

Aklavietėms bei neprogresyviems ciklams pasiekiamų būsenų grafe aptikti pasinaudosime R.Tarjano algoritmu [12]. Pateikiamą pasiekiamumo analizės metodo taikymą galima pavadinti *statiniu apribotu*, kadangi pasiekiamų būsenų grafas nėra generuojamas programos vykdymo metu.

2.4.3.1. Stipriai susietų komponenčių orientuotame grafe išskyrimo algoritmas

Mus domina stipriai susietos grafo komponentės, kurios gali būti statinė aklavietė arba neprogresyvus ciklas. Tačiau, jeigu grafą sudaro tik viena komponentė, galima teigti jog grafe nėra nei aklaviečių nei neprogresyvių ciklų ir visos jo viršūnės yra pasiekiamos iš pradinės (ir viena iš kitos).

Stipriai susietų komponenčių savybės:

Stipriai susietoje grafo komponentėje egzistuoja kelias tarp bet kurių dviejų komponentės viršūnių (t.y dvi orientuoto grafo viršūnės priklauso vienai komponentei tada ir tik tada, jeigu jos yra pasiekiamos viena iš kitos). Tarp skirtingų stipriai susietų komponenčių kelias neegzistuoja arba egzistuoja, bet yra vienkryptis.

Pavyzdžiui, turime dvi stipriai susietas grafo G komponentes G1 ir G2. Galimi tik tokie kelio egzistavimo tarp G1 ir G2 variantai:

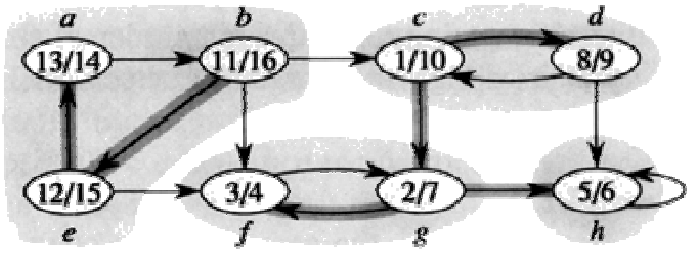
- kelias tarp bet kurios viršūnės v1 iš komponentės G1 ir bet kurios viršūnės v2 iš komponentės G2 neegzistuoja,
- kelias iš bet kurios komponentės G1 viršūnės v1 į bet kurią viršūnę v2 iš komponentės G2 egzistuoja, bet iš v2 į v1 neegzistuoja,
- kelias iš bet kurios komponentės G2 viršūnės v2 į bet kurią viršūnę v1 iš komponentės G1 egzistuoja, bet iš v1 į v2 neegzistuoja.

Pagal tai ar tarp susijusių komponenčių egzistuoja keliai ir kokios krypties jie yra, galima sudaryti šių komponenčių grafą.

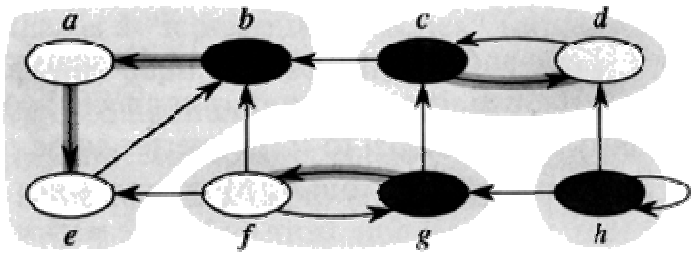
Pritaikius Tarjan‘o algoritmą bet koks grafas gali būti pateiktas kaip stipriai susietų komponenčių aibė, plius keletas viršūnių tom komponentėms surišti.

Stipriai susietų komponenčių grafe išskyrimo schema:

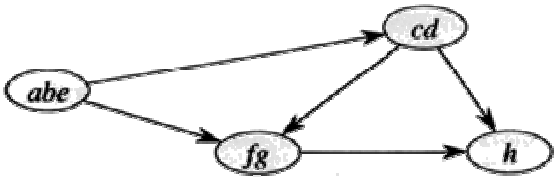
1) Rasti visų viršūnių apdorojimo pabaigos laikus.



2) Sudaryti transponuotą grafą, kuriame pakeistos lankų kryptys.



3) Išoriniame cikle peržiūrėti grafo viršūnes apdorojimo laiko mažėjimo tvarka.



Stipriai susietomis grafo komponentėmis bus paieškos medžiai, gauti 3 žingsnyje:

$\{a, b, e\}$, $\{c,d\}$, $\{f, g\}$ ir $\{h\}$.

Stipriai susietų grafo komponenčių išskyrimo algoritmą sudaro du etapai:

I etapas: Grafas pereinamas į gylį. Viršūnėms priskiriami apdorojimo pabaigos laikai.

II etapas: Pagal viršūnių apdorojimo pabaigos laikus formuojamos stipriai susietos grafo komponentės.

Stipriai susietų komponenčių grafe iškyrimo algoritmas

I etapas

Viršūnės priskiriamos vienai iš aibių:

V neapl. - dar neaplankytų grafo viršūnių aibė,

V apl. - aplankytų bet neapdorotų grafo viršūnių aibė,

V apd. - apdorotų grafo viršūnių aibė.

Trijų aibių viršūnių sąjunga turi sudaryti pilną grafo viršūnių aibę:

$$V_{apd.} \cup V_{apl.} \cup V_{neapl.} = V,$$

Be to, aibės neturi kirstis, t. y. viršūnė vienu metu gali priklausyti vienai ir tik vienai aibei:

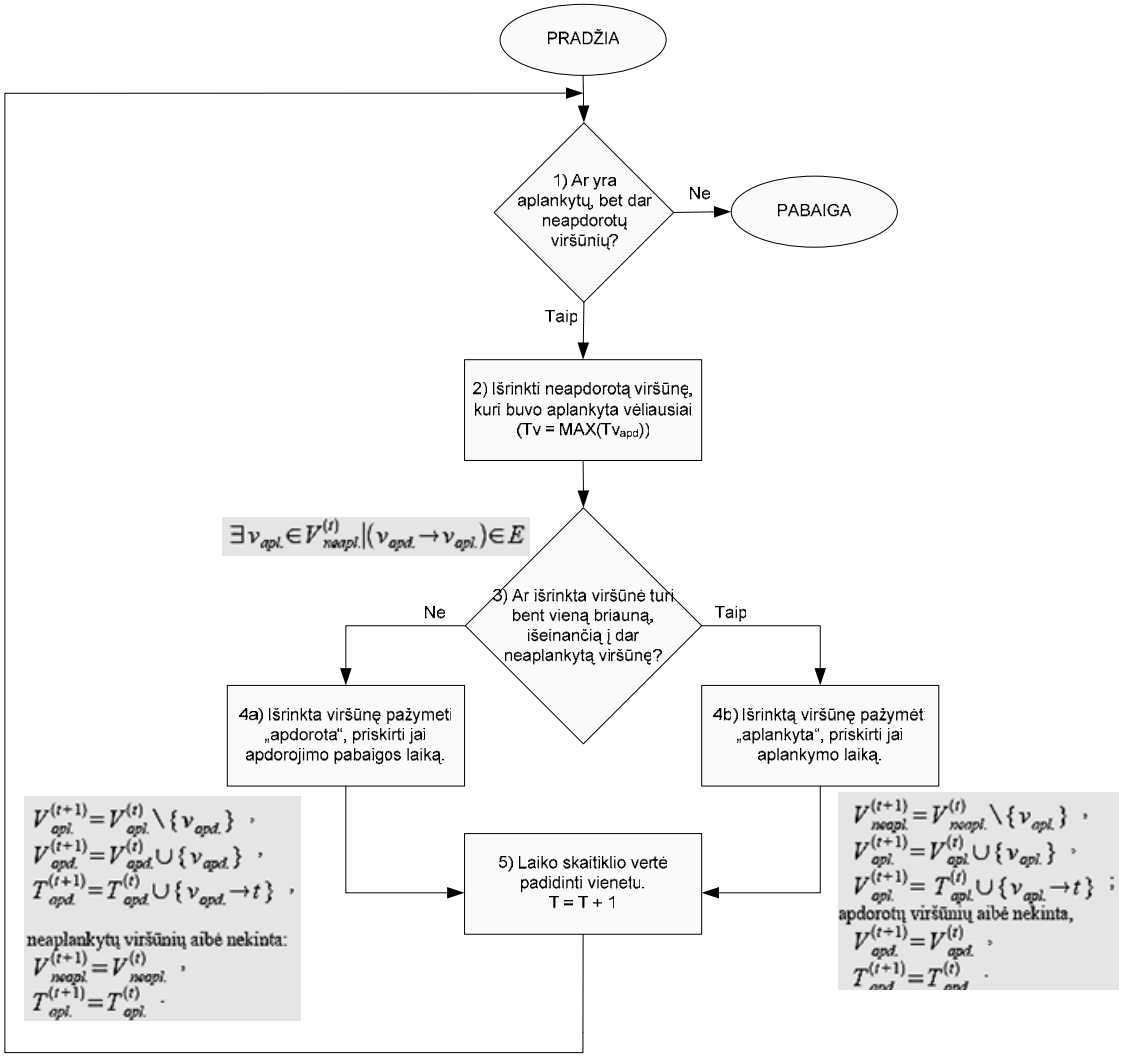
$$V_{apd.} \cap V_{apl.} = \emptyset,\ V_{apd.} \cap V_{neapl.} = \emptyset,\ V_{apl.} \cap V_{neap.} = \emptyset$$

Pradedant grafo perėjimą į gylį visos viršūnės laikomos neaplankytomis ir priklauso aibei *Vneapl.* Po viršūnės aplankymo, jinai priskiriama aibei *Vapl.,* o apskaičiavus viršūnės apdorojimo laiką - aibei *Vapd.* Viršūnės priskyrimas minėtoms aibėms yra nuoseklus

procesas ir viršūnės statuso (neaplankyta, aplankyta, apdorota) pasikeitimo tvarka negali keistis.

Viršūnių aplankymo taisyklė: aplankomos tik tos viršūnės, kurios turi išeinančias briaunas į dar neaplankytas viršūnes. Viršūnė tampa apdorota, kai yra patikrintos visos jos išeinančios briaunos.

Algoritmo pradžioje apdorojimo laiko skaitikliui (naudojamas globalus apdorojimo laiko skaitiklis) priskiriama nulinė vertė. Kiekvienos naujos viršūnės aplankymo metu skaitiklio vertė padidinama vienetu. Einant grafu į gylį susidaro tokios situacijos, kai iš einamos viršūnės nėra briaunų į dar neaplankytas grafo viršūnes, tuomet laikoma, kad šios viršūnės apdorojimas baigtas ir jai priskiriama einamoji skaitiklio vertė. Einamoji viršūnė dabar yra ta, iš kurios buvo ateita į nagrinėjamą viršūnę. Grįžimo metu skaitiklio vertė taip pat yra didinama vienetu.



13 pav. Algoritmo I etapo blokinė schema

Jeigu grafą sudaro daugiau nei viena stipri komponentė, pereinant grafą gilyn greičiausiai bus aplankytos ne visos viršūnės. Todėl reikia pakartotinai atlikti grafo perėjimą, kol bus apdorotos visos grafo viršūnės. Kiekvieno naujo perėjimo metu pradinė viršūnė pasirenkama viena iš dar neaplankytų viršūnių aibės.

Pabaigus pirmą algoritmo etapą, visos viršūnės turi būti apdorotos, t. y. joms turi būti priskirtas apdorojimo pabaigos laikas.

II etapas

Jeigu pirmojo etapo tikslas buvo priskirti kiekvienai viršūnei apdorojimo pabaigos laiką, tai antrajame etape reikia sukomponuoti stipriai susietas komponentes pagal viršūnių apdorojimo pabaigos laikus (apskaičiuotus pirmajame etape).

Viršūnės priskiriamos vienai iš aibių (šių aibių sąjunga sudaro pilna grafo viršūnių aibę):

V neapl. - dar neaplankytų grafo viršūnių aibė,

V apl. – aplankytų, bet neapdorotų grafo viršūnių aibė,

V apd. - apdorotų grafo viršūnių aibė.

V prisk. – aibė viršūnių, priskirtų vienai iš stipriai susietų komponenčių.

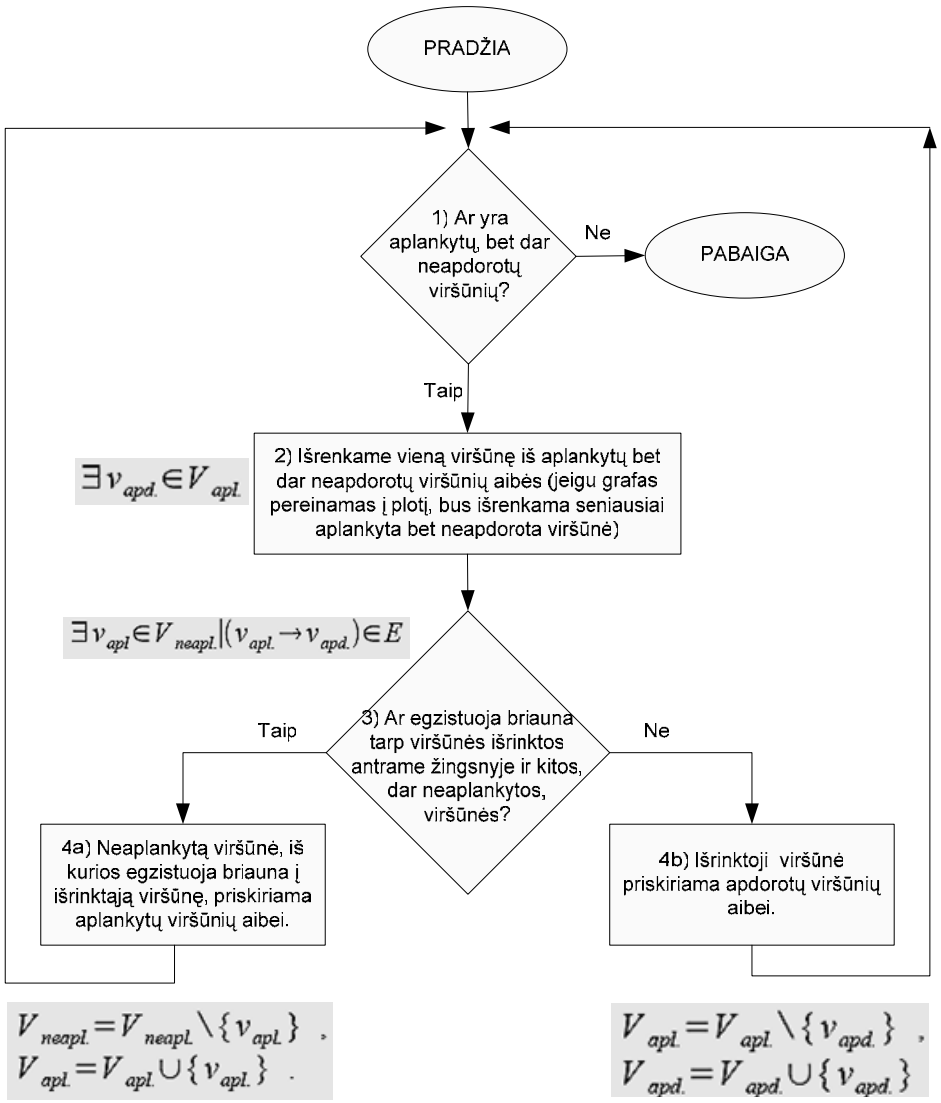
Viršūnė vienu metu gali priklausyti vienai ir tik vienai aibei:

$$V_{prisk.} \cap V_{apd.} = \emptyset \;,\; V_{prisk.} \cap V_{apl.} = \emptyset \;,\; V_{prisk.} \cap V_{neapl.} = \emptyset \;,$$
$$V_{apd.} \cap V_{neapl.} = \emptyset \;,\; V_{apd.} \cap V_{apl.} = \emptyset \;,\; V_{apl.} \cap V_{neapl.} = \emptyset \;.$$

Šiame etape reikia vėl atlikti grafo perėjimą. Tam tikslui galima sudaryti transponuotą grafą, arba tiesiog eiti priešingomis grafo briaunų kryptimis. Dabar jau nesvarbu ar jis bus pereinamas į gylį ar į plotį. Pradine viršūne pasirenkama ta, kurios apdorojimo laikas yra pats didžiausias. Grafas pereinamas pradedant pasirinktąja viršūne tol, kol iš visų pereitų viršūnių nebelieka briaunų į dar nepereitas viršūnes. Šios pereitos viršūnės yra pirmoji stipriai susijusi grafo komponentė.

Toliau imama dar nepereita viršūnė su didžiausiu apdorojimo pabaigos laiku, ir vėl atliekamas grafo perėjimas į plotį arba į gylį, kol nebelieka briaunų kuriomis būtų galima pasiekti nepereitas viršūnes iš jau pereitų viršūnių. Toliu būdu gaunama sekanti stipriai susijusi grafo komponentė.

Šis žingsnis kartojamas tol kol nelieka nei vienos nepereitos viršūnės. Tokiu būdu gaunamos visos stipriai susietos grafo komponentės.

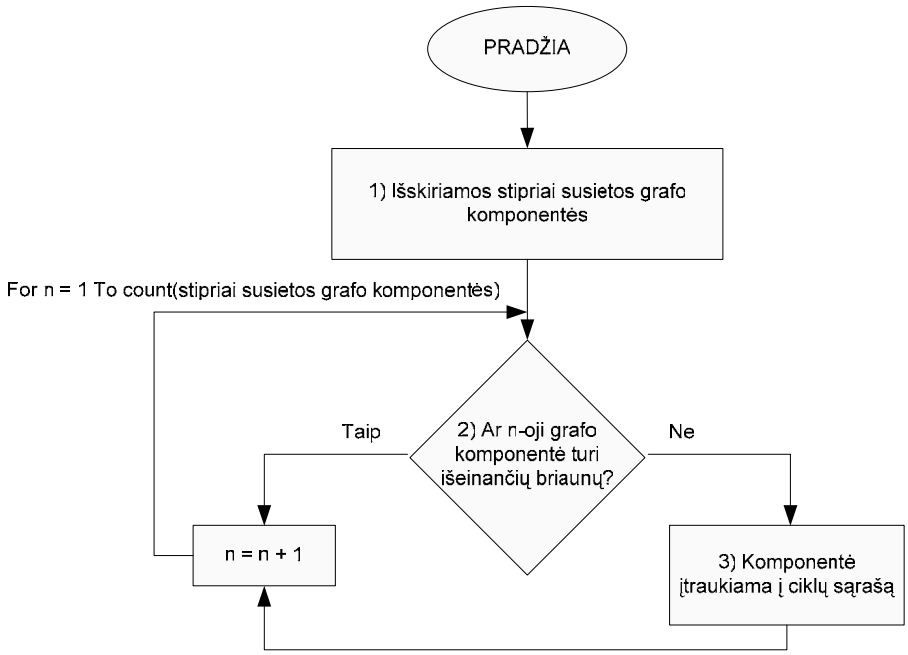


14 pav. Algoritmo II etapo blokinė schema

Po to, kai stipriai susietos komponentes yra išskirtos, reikia sudaryti jų grafą. Tokiame grafe pačios komponentės dalyvauja kaip viršūnės. Jungianti dvi komponentes briauna yra ta briauna, kuri jungia dvi grafo viršūnes, priklausančias skirtingoms stipriosioms komponentėms.

Aprašytas algoritmas sėkmingai taikomas praktikoje aklavietėms ir neprogresyviems ciklams sistemos pasiekiamų būsenų grafe aptikti.

Kaip jau buvo minėta, neprogresyvus ciklas tai yra grafo komponentė, sudaryta iš daugiau nei vienos viršūnės ir neturinti išeinančių briaunų. Ciklo aptikimo algoritmo schema pateikta 15 pav.



15 pav. Neprogresyvaus ciklo radimo algoritmas

Egzistuoja efektyvios alternatyvos Tarjano algoritmui [18][19][21]. Naudojant kitus metodus paieškos gilyn metu kiekviena būsena aplankoma du kartus, tačiau saugoma tik vieną kartą.

Pavyzdžiui, SPIN verifikavimo procedūra grindžiasi optimizuotu grafo trasavimo gilyn metodu ir stekinės paieškos gilyn procedūra.

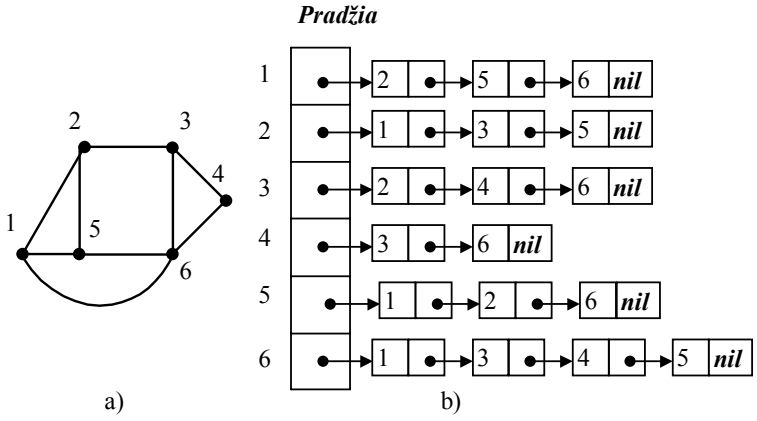
2.4.3.2. Stekinė paieška gilyn

Dėmesio centre yra ciklų aptikimas. Metodui keliamas reikalavimas – jis turi būti suderinamas su visais verifikavimo režimais, įskaitant pilną paiešką, bitų hešavimą ir dalinės tvarkos mažinimo priemonėmis.

Klasikinis ciklo aptikimo grafe algoritmas yra Tarjan‘o paieška gilyn, kuris tiesiniame laike (angl. *linear time*) išskiria stipriai susietas komponentes papildant kiekvieną būseną dviem sveikaisiais skaičiais: būsenos *aplankymo* ir *apdorojimo pabaigos* momentai. Kadangi SPIN gali sugeneruoti milijardus pasiekiamų būsenų, šiems dviems skaičiams saugoti mažiausiai reikia 32 bitų (kiekvienam). Jeigu pasiekiamų būsenų grafo stipriai susieta komponentė turi bent vieną priimančią būseną (t. y. būseną, turinčią bent vieną įeinančią briauną), galima tvirtinti, jog egzistuoja pasiekiamas ciklas. Tarjan‘o algoritmas pagrįstas

aplankymo ir *apdorojimo pabaigos* parametrų tikslumu ir nėra suderinamas su bitų hešavimo priemonėmis, kurios taipogi įeina į SPIN.

Stekinė paieška gilyn gali būti realizuota papildant kiekvieną būseną tik dviem bitais (o ne 64-iais, kaip Tarjan‘o algoritme) naudojant paprastą kodavimo metodą. Tokiu būdu, stekinės paieškos gilyn principas yra toks: tam, kad būtų galima tvirtinti, jog pasiekiamų būsenų grafe egzistuoja ciklas, turi egzistuoti bent viena būsena pasiekiama iš pradinės būsenos (grafo šakninės viršūnės) ir iš savęs pačios. Pirmas paieškos gilyn etapas nustato būsenas, pasiekiamas iš pradinės viršūnės. Antroji (stekinė) paieška prasideda kiekvienoje pirmame etape aptiktoje būsenoje (pav. 16 b), šiuo metu tikrinama ar būsena gali būti pasiekta iš savęs pačios. Jeigu taip, mes radome vykdomąją seką, kuri aprašo ciklą.



16 pav. a) grafo gretimumo struktūra; b) vienryšių sąrašų masyvas

SPIN kontekste tokia vykdomoji seka prilyginama vartotojo apibrėžto korektiškumo reikalavimo kontrargumentui (prieštaravimui) ir gali pasitarnauti kaip korektiškumo reikalavimo klaidingumo įrodymas.

Stekinė paieška gilyn neužtikrina *visų* pasiekiamų būsenų grafo ciklų aptikimo. Ji tik gali parodyti, kad egzistuoja *bent vienas* ciklas.

2.4.4. Korektiškumo loginiai įrodinėjimai (invariantai)

Sistemos invariantas - predikatas, aprašytas sistemos būsenos vektoriaus kintamaisiais, kuris privalo būti tenkinamas visoms sistemos būsenoms, nepriklausomai nuo sistemos įvykių.

Užduotis – patikrinti, ar invariantas tenkinamas visose pasiekiamose grafo būsenose (t. y. visuose kontroliniuose sistemoje vykstančių procesų taškuose).

Sistemos invariantams sukurti naudojama predikatų logika. Paprasčiausius invariantus gali aprašyti pats vartotojas. Sudėtingų sistemų invariantams kurti galima pasinaudoti automatiniais įrankiais [20].

2.5. SPECIFIKACIJŲ VALIDAVIMO PROBLEMOS SPRENDIMAS PASAULYJE

Pirmieji automatizuoti modelių tikrinimo ir verifikavimo įrankiai pasirodė daugiau nei prieš dvidešimt metų. Šiuo metu populiariausias ir labiausiai paplitęs tarp vartotojų yra SPIN (*Simple Promela INterpretor*) [5]. SPIN įrankis buvo sukurtas 1980 metais *Bell Labs* laboratorijoje, pradinė jo paskirtis buvo iliustruoti vieną iš J.Holzmann[1] rengiamų straipsnių. Šiandienos SPIN skiria nuo kitų verifikavimo sistemų tokios savybės:

- SPIN skirtas programinės, bet ne techninės įrangos specifikacijoms tikrinti. Įrankis palaiko aukšto lygio sistemos reikalavimų specifikavimo kalbą PROMELA (*PROcess Meta LAnguage*). SPIN sėkmingai naudojamas loginėms klaidoms aptikti paskirstytų sistemų modeliuose (pvz., operacinės sistemos, duomenų perdavimo protokolai, paskirstyti algoritmai, kt.)
- SPIN palaiko C kodo vartojimą kaip modelio specifikacijos dalį. Tokiu būdu SPIN gali būti naudojamas ne tik kaip loginio tikrinimo modulis, bet ir kaip specifikacijos programinės realizacijos variklis.
- SPIN gali būti naudojamas kaip LTL (*Linear Temporal Logic*) modelių tikrinimo sistema. Jis palaiko visus reikalavimus sistemai, aprašytai linijinio laiko logikos sintakse.
- Įrankis lanksčiai valdo dinamiškai besikeičiantį sistemos procesų skaičių. Tam naudojamas būsenos vektoriaus metodas.

SPIN palaiko atsitiktinį, interaktyvų bei valdomą sistemos darbo imitavimą.

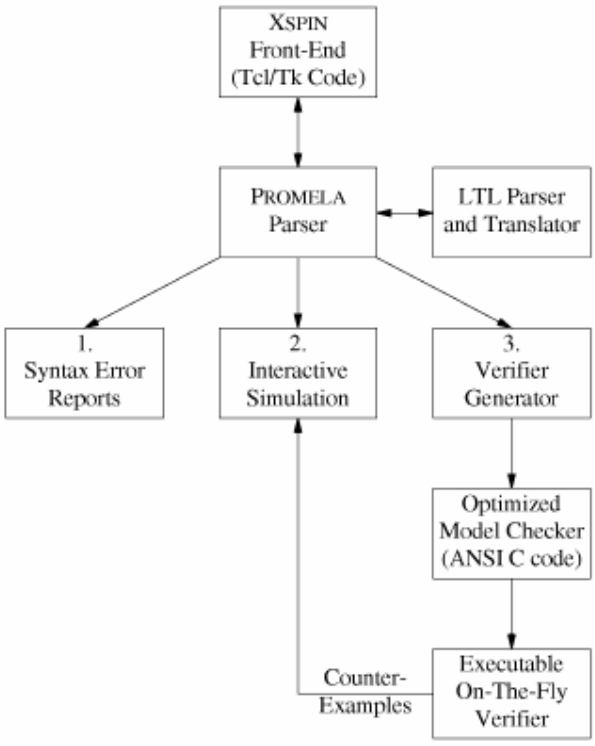
---

[1] Gerard J. Holzmann 1979 m. gavo technikos mokslų daktaro laipsnį, nuo 1980 m. dirba Bell Laboratorijos Skaičiavimo Centre, Murray Hill, New Jersey. Sukūrė kelis automatizuoto specifikacijų verifikavimo įrankius, populiariausias iš jų yra SPIN. Parašė keletą knygų apie skaitmeninį vaizdo apdorojimą, komunikacijų istoriją, protokolų verifikavimą. Dirba *Formal Methods in Systems Design* žurnalo redaktoriumi.

SPIN turi tris pagrindinius darbo režimus:

- Imitavimo – leidžia greitai suimituoti modelio darbą atsitiktinio, interaktyvaus ar vartotojo valdomo būsenų pasirinkimo principu.
- Išsamaus verifikavimo – skirtas vartotojo aprašytiems korektiškumo reikalavimams sistemai tikrinti.
- Teisingumo aproksimavimo – gali validuoti netgi labai didelius modelius maksimaliai padengiant jo galimų būsenų aibę.

Visų pirma specifikacijos analizė atliekama su atsitiktiniu arba interaktyviu (dialoginiu) imitavimu. Detalesniam sistemos nagrinėjimui validavimo subįrankis patikrina specifikaciją aklaviečių, ciklų be išėjimo atžvilgiu. Jeigu sistema yra tiek didelė, kad neužtenka sisteminių resursų (pvz., kompiuterio atminties), validavimas atliekamas su atsitiktinai pasirinktų būsenų aibėmis. Tokios priemonės yra pakankamos korektiškumui ir funkciniams reikalavimams, kurie gali būti realizuoti sistemos prototipe, įvertinti [15].



17 pav. SPIN struktūra: verifikavimo ir modeliavimo modulis

Bendra SPIN modelio tikrintojo struktūra pateikta 17 pav. Tipinis darbo režimas prasideda nuo aukšto lygio lygiagrečios sistemos (arba paskirstyto algoritmo) specifikacijos panaudojant grafinį įrankį XSPIN. Po to, kai sintaksės klaidos yra pašalintos, atliekamas dialoginis (interaktyvus) imitacinis modeliavimas kol bus pasiektas pradinis įsitikinimas, kad sistema veikia kaip buvo suplanuota.

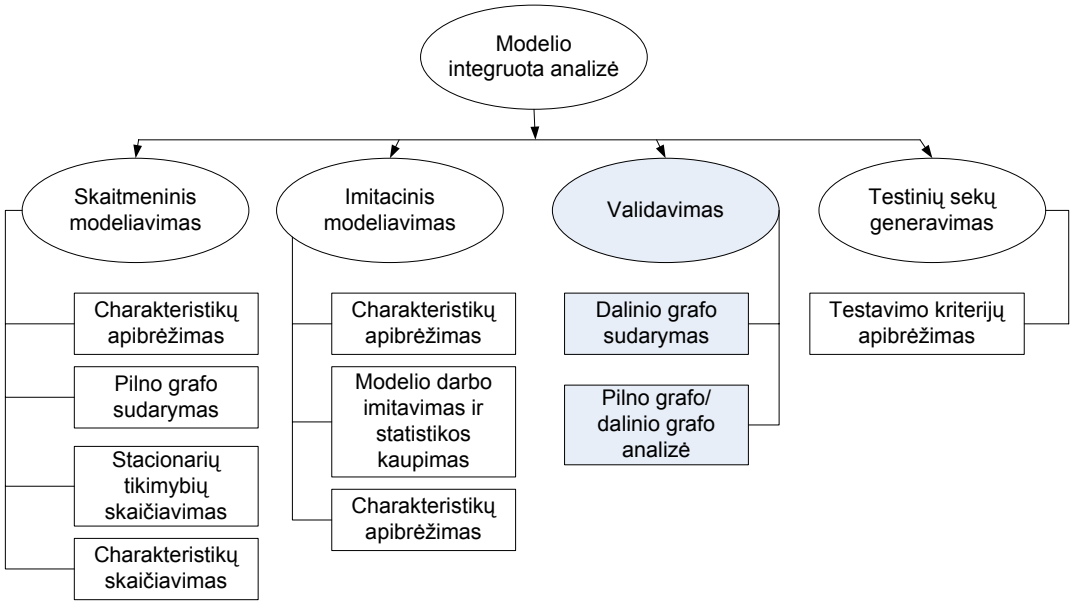
Toliau, trečiame žingsnyje, SPIN generuoja optimizuotą verifikavimo programą iš aukšto lygio specifikacijos (vykdymo metu - *„on-the-fly“*). Verifikatorius sukompiliuotas atsižvelgiant į įvairius kompiliavimo laikus, kurie priklauso nuo naudojamų ir vykdomų mažinimo (redukcijos) algoritmų tipų.

Jeigu korektiškumo reikalavimuose aptinkami prieštaravimai, jie gali būti grąžinti interaktyviajam simuliatoriui ir detaliai išnagrinėti, o prieštaravimų priežastys – pašalintos.

3. PROJEKTINĖ DALIS

3.1. VALIDAVIMO POSISTEMIO VIETA AGREGATINIŲ SPECIFIKACIJŲ INTEGRUOTOS ANALIZĖS SISTEMOJE

Agregatinių specifikacijų integruotos analizės sistemą sudaro keli savarankiški įrankiai: skaitmeninio modeliavimo, imitacinio modeliavimo, validavimo ir testų generavimo. Nors šiuo metu išvardyti įrankiai nėra apjungti į bendrą sistemą, tačiau šie atskiri moduliai veikia vieningos agregatinės specifikacijos bazėje.

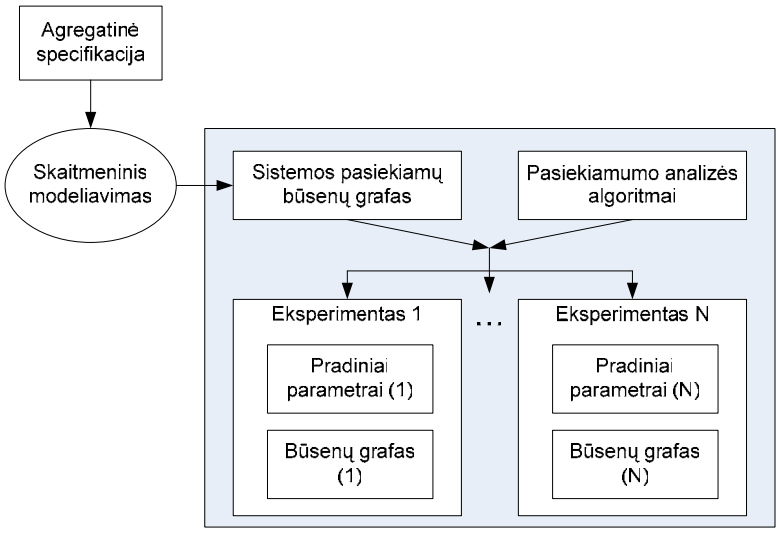


18 pav. Validavimo įrankio vieta bendroje integruotos analizės sistemoje

Validavimo posistemio pradiniai duomenys generuojami skaitmeninio modeliavimo modulyje. Skaitmeninio modeliavimo posistemė sugeneruoja tiktai *pasiekiamų* būsenų grafus, todėl validavimo posistemėje pasiekiamų būsenų aibės konstravimo atlikti nereikia.

Turint didelę pasiekiamų būsenų aibę, vartotojas gali išskirti dalinį pografį ir atlikti jo analizę.

Sukurto validavimo įrankio VALSYS veikimo koncepcija parodyta 19 pav.



19 pav. Validavimo posistemis kaip „juodoji dėžė“

3.2. REIKALAVIMAI SISTEMAI

Agregatinių specifikacijų validavimo įrankis skirtas sistemos agregatinės specifikacijos analizei atlikti.

Funkciniai reikalavimai sistemai:

- pasiekiamų būsenų grafo validavimo uždavinio sprendimas: aklaviečių, neprogresyvių ciklų paieška, koordinačių ribų tikrinimas, invariantų tikrinimas;
- pasiekiamų būsenų grafo trasavimas;
- dalinio pasiekiamų būsenų pografio išskyrimas ir jo analizė;
- vartotojo išskirtų pografių saugojimas.

Nefunkciniai reikalavimai sistemai:

- paprastas panaudojamumas, nereikalaujantis specialaus vartotojų apmokymo;
- lengvas sistemos pernešamumas;
- sistemos išplečiamumas: esant poreikiui sistemą turi būti papildyta naujomis funkcijomis.

Reikalavimai duomenims:

- Kaip pradinius duomenis VALSYS turi naudoti skaitmeninio modeliavimo posistemės sugeneruotą pasiekiamų būsenų grafą.

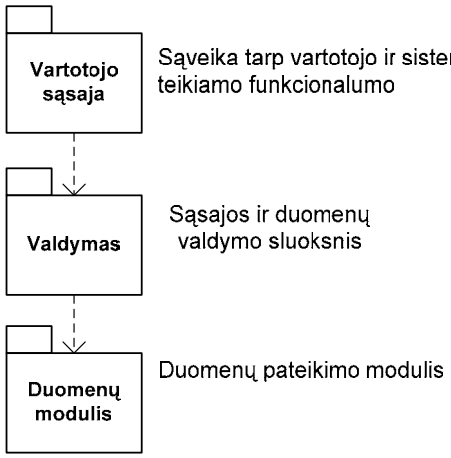
3.3. ARCHITEKTŪRINIAI SPRENDIMAI

Suformuluoti reikalavimai sistemai įtakoja architektūros pasirinkimą.

Priklausomybė nuo išorinių duomenų šaltinių diktuoja duomenų nuskaitymo bei saugojimo taisykles. Operacijos su duomenimis turi būti atskirtos, kad pasikeitus duomenų formatui sistema būtų kuo greičiau ir kuo lengviau pritaikyta pakeitimams.

Sistemos valdymo uždaviniui spręsti visos atliekamos analizės funkcijos buvo iškeltos į atskirą paketą. Funkcionalumo atskyrimas nuo duomenų ne tik užtikrina kuriamos sistemos objektiškumą, bet ir palengvina jos išplečiamumą.

Įvertinus visa tai, buvo pasirinktas trijų lygių architektūros modelis (pav. 20).



20 pav. Sistemos architektūros modelis

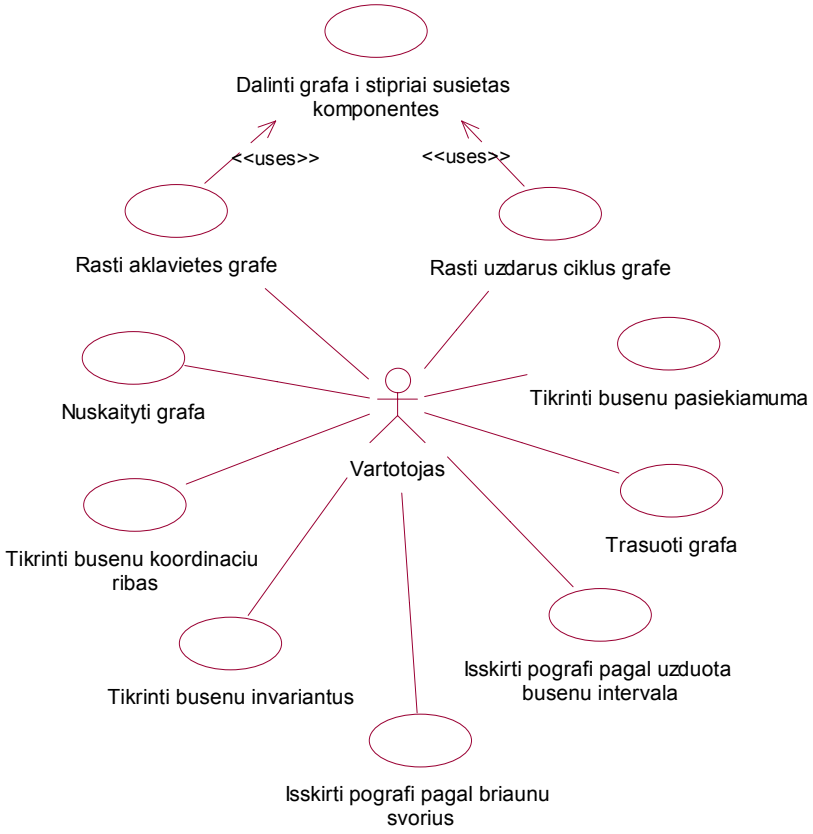
3.4. SISTEMOS FUNKCIJŲ APŽVALGA

Agregatinių specifikacijų validavimo VALSYS sistema teikia tokias paslaugas:

- pasiekiamų būsenų grafo (plius kiekvienos būsenos vektoriaus) nuskaitymas;
- pilno grafo dalijimas į stipriai susietas komponentes;
- pografio iškyrimas pagal:
    - briaunų svorius,
    - užduotą būsenų intervalą (pagal grafo viršūnių numerius);
- grafo/pografio/stipriai susietos komponentės informacijos peržiūra;
- galimybė išsaugoti (atskirai nuo pradinio grafo duomenų) vartotojo išskirtą pografį ar stipriai susietą komponentę;
- analizės funkcijos validavimo uždavinio sprendimo ribose:
    - aklaviečių paieška grafe
    - neprogresyvių ciklų paieška grafe
    - būsenų koordinačių ribų tikrinimas pagal vartotojo užduotas koordinačių reikšmes

- būsenų pasiekiamumo tikrinimas
- grafo trasavimas
- invariantų tikrinimas.

Sistemos ir jos aplinkos sąveikos detalizavimui bei konkretizavimui, sudarytas panaudojimo atvejų modelis (UML notacija), kuris pateikiamas 21 paveiksle. Šis modelis atskleidžia ne tik sistemos kontekstą, bet ir vartotojų sąveiką su sistema.



21 pav. Sistemos VALSYS panaudojimo atvejų modelis

3.5. SISTEMOS KOMPONENTŲ ARCHITEKTŪRA IR FUNKCIONAVIMAS

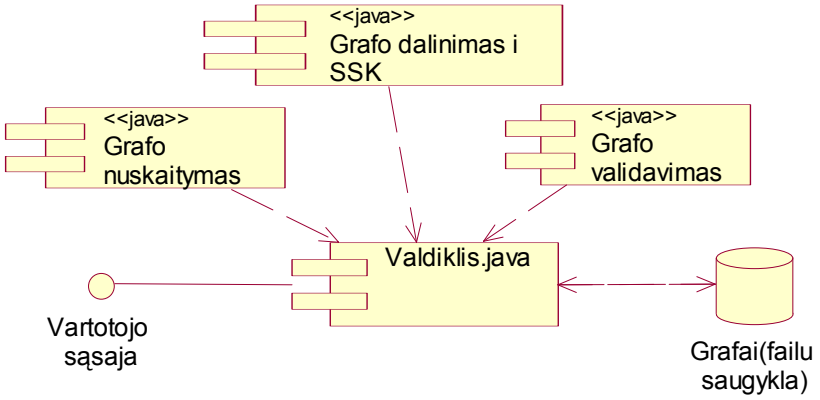
Vartotojas bendrauja su sistema per grafinę sąsają. Jos realizacija nebus detalizuojama.

Tam, kad atlikti specifikacijos validavimą, grafas turi būti nuskaitytas ir išskirtos jo stipriai susietos komponentės.

Pagrindinis valdiklis kontroliuoja kitų sistemos komponentų darbą. Informacijos surinkimo ir apdorojimo moduliams perduodami veiklos parametrai (pvz., vartotojo nurodymai formuoti pografį).

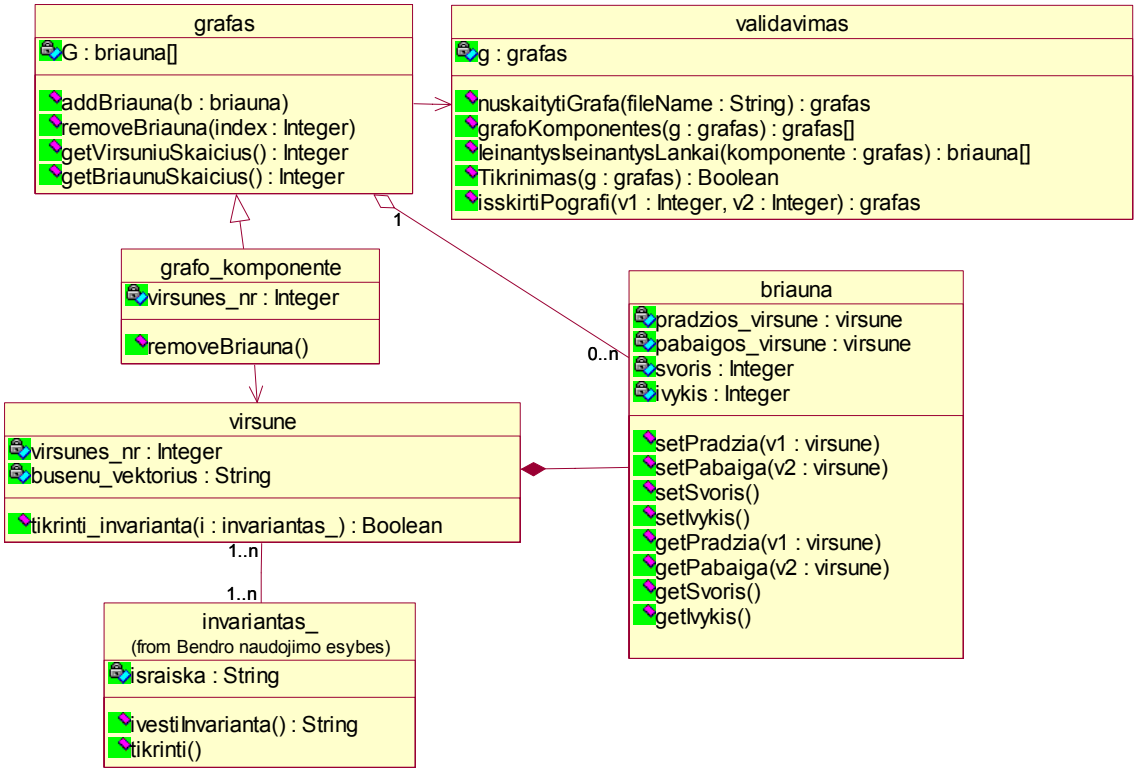
Pagrindinis valdiklis suteikia priėjimą prie sistemos valdymo vartotojui, atlikdamas vartotojo informacijos persiuntimą kitiems sistemos moduliams ir apdorotos informacijos perdavimą vartotojo sąsajai.

Sistemos komponentų sąveika parodyta 22 pav.



22 pav. Sistemos komponentų diagrama

Apibendrinta sistemos klasių diagrama (23 pav.) rodo duomenų struktūrų ir valdymo klasės tarpusavio ryšius, kurie buvo nustatyti remiantis probleminės srities bei sprendžiamo validavimo uždavinio analizės metu.



23 pav. Klasių diagrama

3.5.1. Duomenų nuskaitymo modulis

Pradiniai grafo duomenys laikomi trijuose failuose: viršūnių, briaunų ir būsenos vektoriaus aprašymo. Sėkmingam grafo formavimui būtini du failai – briaunų bei viršūnių. Būsenos vektoriaus failas turi detalią informaciją apie vektoriaus kintamuosius.

2 lentelė. Įėjimo duomenų formatas

Failo pavadinimas	Duomenų formatas	Duomenų pavyzdys

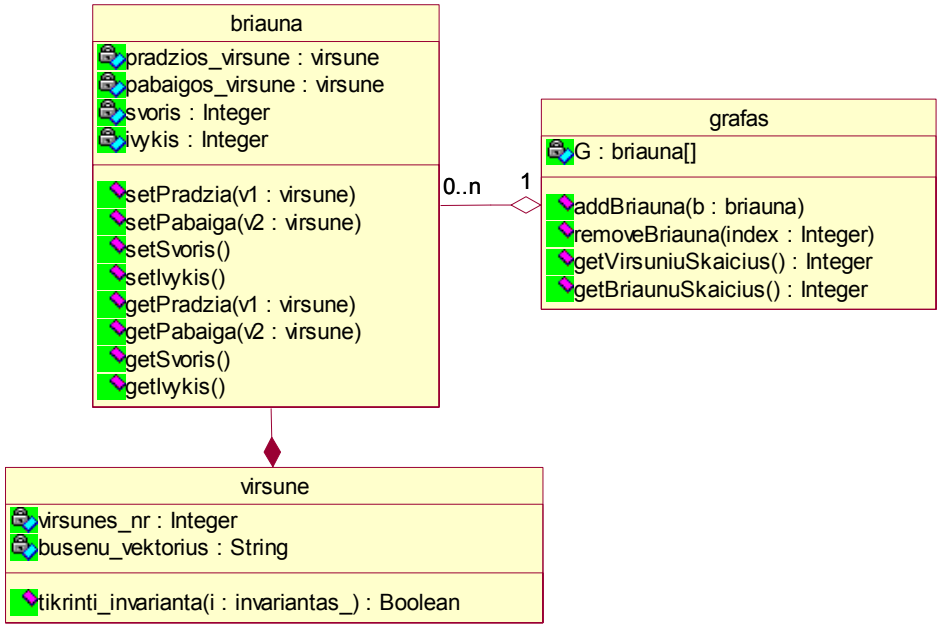
Viršūnių failas *nodes.txt**	<komentarai>
	@ <skyrybos ženklas>
	<viršūnės numeris> : <būsenos vektorius>
		`NODES=6`
		`@`
		`0:1.1 4.0 0.0 0.0`
		`1:1.1 3.0 0.9 0.0`

Briaunų failas *edges.txt**	<komentarai>
	@ <skyrybos ženklas>
	<briaunos pradžios viršūnė> :
	<briaunos pabaigos viršūnė >: <briaunos svoris> :
	<įvykis dėl kurio įvyko perėjimas> : <papildoma informacija>
		`EDGES=26`
		`@`
		`0:1:1.0:User1.getR1:,User1.get R1,Resource1.X2,User1.X1.`
		`0:2:1.0:User1.getR2:,User1.get R2,Resource2.X1,User1.X2.`

Vektoriaus failas *vector.txt**	<komentarai>
	@ <skyrybos ženklas>
	<kintamojo indeksas būsenos vektoriuje> : <kintamojo aprašymas>
		`COORDINATES=12`
		`@`
		`0:User1.getR1`
		`1:User1.getR2`

* failų pavadinimai nėra reglamentuojami, čia nurodyti tik kaip pavyzdys

Grafą sudaro bent viena briauna, kuri turi pradžios ir pabaigos viršūnes. Klasių diagrama (pav. 24) rodo klasių, sudarančių duomenų struktūras, tarpusavio ryšius.



24 pav. Duomenų modelio klasių diagrama

Grafo formavimas vyksta pagal tokį scenarijų:

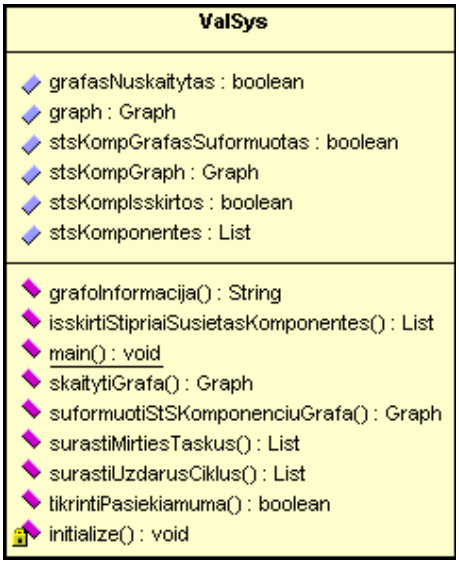
1. Nuskaitomas viršūnių failas ir sudaromas grafo viršūnių sąrašas.

2. Nuskaitomas briaunų failas ir pagal viršūnes sudedamos briaunos. Apie briauną surenkama tokia informacija: pradžios viršūnė, pabaigos viršūnė, briaunos svoris. Turime patikrinti, ar abi briaunos viršūnės egzistuoja (ar jos yra įtrauktos į viršūnių sąrašą), jeigu taip – formuojame briauna ir įdedame ją į briaunų sąrašą. Kitais atvejais briauna praleidžiama.

3.5.2. Validavimo operacijų atlikimo modulis

Modulis atsako už grafo stipriai susietų komponenčių išskyrimo operacijos kvietimą bei už validavimo operacijų atlikimą.

Suformuotas grafas dalinamas į stipriai susietas komponentes, kurios vėliau naudojamos aklaviečių, neprogresyvių ciklų paieškos bei būsenų pasiekiamumo tikrinimo operacijose. Jeigu kažkokiu būdu stipriai susietų komponenčių grafas nebuvo suformuotas sistemos darbo iniciajavimo metu, tai bus padaryta vėliau. Komponenčių išskirymo požymis saugomas loginio tipo kintamajame.



25 pav. Pagrindinė validavimo klasė

Šis modulis turi sąsaja su duomenų pateikimo bei su vartotojo sąsajos moduliais.

3.6. TECHNINĖS REALIZACIJOS PRIEMONĖS

Siekiant patenkinti maksimalaus ekonominio naudingumo kriterijų, sistemos realizacijai pasirinktos nemokamos programų kūrimo bei įdiegimo priemonės. Naudota Java programavimo kalba, leidžianti kurti nepriklausomą nuo operacinės sistemos platformos programinę įrangą. VALSYS sistemos modulių kūrimui pasirinkta Eclipse 3.0 platforma.

Vartotojo grafinei sąsajai naudojama Eclipse 3.0 platforma bei Swing grafikos biblioteka.

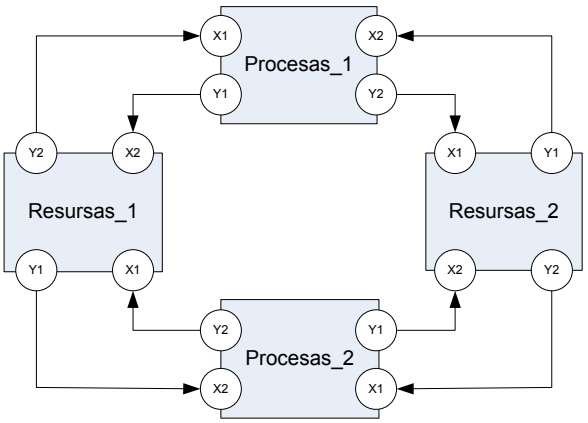
Duomenys saugomi tekstiniuose failuose, griežtai laikantis reikalavimų duomenų formatui (įėjimo duomenų pavyzdys pateiktas Priede 1).

4. EKSPERIMENTINĖ DALIS

4.1. „PIETAUJANČIŲ FILOSOFŲ“ UŽDAVINYS

Tai yra klasikinis bendrų resursų naudojimo uždavinys. Jo esmė yra ta, kad vienu metu tik vienas procesas gali naudoti vieną resursą (resursas gali būti sistemos servisas, kita taikomoji programa), kitas procesas turi sulaukti, kol resursas atsilaisvins ir tik tada jį užimti (abipusio pasišalinimo principas). Procesas („valgantis filosofas“) paleidžiamas veikti („pietauti“) tik tuomet, kai galės naudoti abu resursus („šakutę“ ir „peilį“) vienu metu.

Kaip pavyzdį paimsime sistemą, kurioje du nepriklausomi procesai naudoja du paskirstytus resursus. Pateiksime sistemą keturių agregatų kompozicijos pavidalu (26 pav.).



26 pav. Agregatų sujungimo schema

Aprašysime dviejų tipų agregatus: ***Procesas*** ir ***Resursas***.

Agregatas *Resursas*.

Resursas yra pasyvus agregatas be vidinių įvykių. Agregatas turi du įėjimus, naudojamus užklausoms iš skirtingų resursų naudotojų priimti, ir du išėjimus, skirtus resurso suteikimui naudotojui.

Signalo tipas: Single{a}. Galimos reikšmės: 0 - resurso užklausai, 1 - resurso grąžinimui.

Resource turi viena diskretu kintamaji, rodanti ar resursas nera naudojamas. 0 – kai resursas naudojamas ir jo nera, 1 – kai resursas nenaudojamas ir ji galima priskirti kitam naudotojui.

Agregato specifikacija

1. Įėjimai: {X1:Single, X2:Single}
2. Išėjimai: {Y1:Single, Y2:Single}
3. Diskretieji kintamieji: {available}

4. Išoriniai įvykiai

   Įvykis *X1*: iš agregato ***Procesas_1*** atėjo užklausa resursui gauti
   H:
   available=1, kai available=0 ir X1.a=1 // resursai gąžinti
   available=0, kai available=1 ir X1.a=0 // resursai suteikti
   G:
   Y1.a = 1, kai X1.a=0 ir available=1 // resursai suteikiami, kai jų prašoma ir jie yra laisvi

   Įvykis *X2*: iš agregato ***Procesas_2*** atėjo užklausa resursui gauti
   H:
   available=1, kai available=0 ir X2.a=1 // resursai grąžinti
   available=0, kai available=1 ir X2.a=0 // resursai suteikti
   G:
   Y2.a = 1, kai X2.a=0 ir available=1 // resursai suteikiami, kai ju prasoma ir jie yra laisvi

Agregatas *Procesas.*

Procesas turi du tolydžiuosius kintamuosius, skirtus resursų gavimo įvykiams generuoti, ir vieną tolydųjį kintamąjį - resurso naudojimo pabaigos įvykiui pažymėti. Diskretieji kintamieji rodo resursų naudojimą. Du agregato įėjimai skirti resursams priimti, ir du išėjimai – resursams sistemai grąžinti.

Agregato specifikacija

1. Įėjimai: {X1:Single, X2:Single}
2. Išėjimai: {Y1: Single, Y2: Single}
3. Tolydieji kintamieji: {getR1, getR2, valgo}
4. Diskretieji kintamieji: {R1, R2}.
5. Konstantos: {R1_intens, R2_intens, valgo_intens}
6. Išoriniai įvykiai:

   Įvykis *X1*:
   H:
   getR1 = 0, kai X1.a = 1
   R1 = 1, kai X1.a = 1
   valgo = valgo_intens , kai R2=1 ir X1.a = 1
   G: {}
   Įvykis *X2*:
   H:
   getR2 = 0, kai X2.a = 1
   R2 = 1, kai X2.a = 1
   valgo = valgo_intens, kai R1 = 1 ir X2.a = 1
   G: {}
7. Vidiniai įvykiai:

   Įvykis *getR1*: generuojama užklausą pirmajam resursui gauti
   H: {}

G:

Y1.a = 0 // prašoma resursų

Įvykis *getR2*: generuojama užklausą antrajam resursui gauti
H: {}
G:

Y2.a = 0 // prašoma resursų

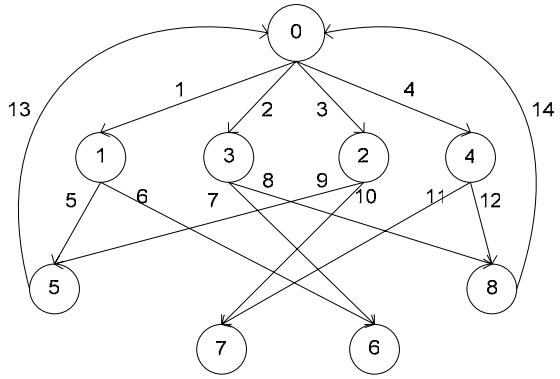
Įvykis *valgo*: paleidžiamas naujas procesas
H:

getR1 = R1_intens // prašoma resursų naujam proceso paleidimui
getR2 = R2_intens // prašoma resursų naujam proceso paleidimui
R1 = 0
R2 = 0
valgo = 0

G:

Y1.a = 1 // grąžinami resursai
Y2.a = 1 // grąžinami resursai

Sistemos pasiekiamų būsenų grafas.



27 pav. Sistemos pasiekiamų būsenų grafas

Viršūnių aprašymas:

0: sistemos paleidimas
1,3: Resursas_1 (įėjimai X1, X2 atitinkamai)
2,4: Resursas_2 (įėjimai X1, X2 atitinkamai)
5,7: Procesas_1 (įėjimai X1, X2 atitinkamai)
6,8: Procesas_2 (įėjimai X1, X2 atitinkamai)

Briaunų aprašymas:

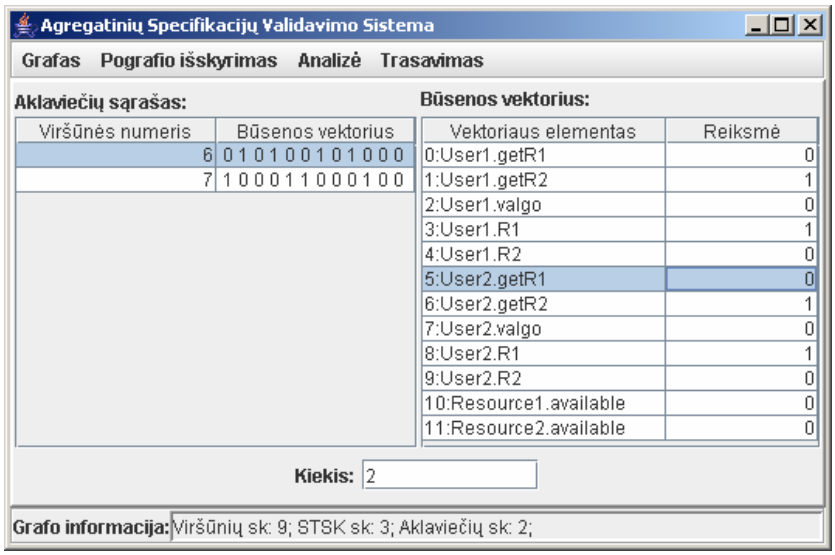
1: Procesas_1 siunčia užklausą pirmajam resursui gauti (Resursas_1)
2: Procesas_2 siunčia užklausą pirmajam resursui gauti (Resursas_1)
3: Procesas_1 siunčia užklausą antrajam resursui gauti (Resursas_2)
4: Procesas_2 siunčia užklausą antrajam resursui gauti (Resursas_2)
5: pirmasis resursas suteikiamas pirmajam procesui
6,7,8: pirmasis resursas suteikiamas antrajam procesui
9,10,11: antrasis resursas suteikiamas pirmajam procesui
12: antrasis resursas suteikiamas antrajam procesui

13: pirmasis procesas gavęs abu resursus pradeda darbą; sistema grįžta į pradinę būseną

14: antrasis procesas gavęs abu resursus pradeda darbą; sistema grįžta į pradinę būseną.

Specifikacijos analizė.

Sistemoje aptiktos dvi aklavietės, kurias galima pastebėti ir iš pasiekiamų būsenų grafo, pateikto 27 pav.



28 pav. Aptiktos aklavietės

Aklavietės gaunasi tuomet, kai abu resursai vienu metu pradeda naudoti tą patį procesą (29 pav.).

Vektoriaus elementas	Reiksmė
0:User1.getR1	0
1:User1.getR2	1
2:User1.valgo	0
3:User1.R1	1
4:User1.R2	0
5:User2.getR1	0
6:User2.getR2	1
7:User2.valgo	0
8:User2.R1	1
9:User2.R2	0
10:Resource1.available	0
11:Resource2.available	0

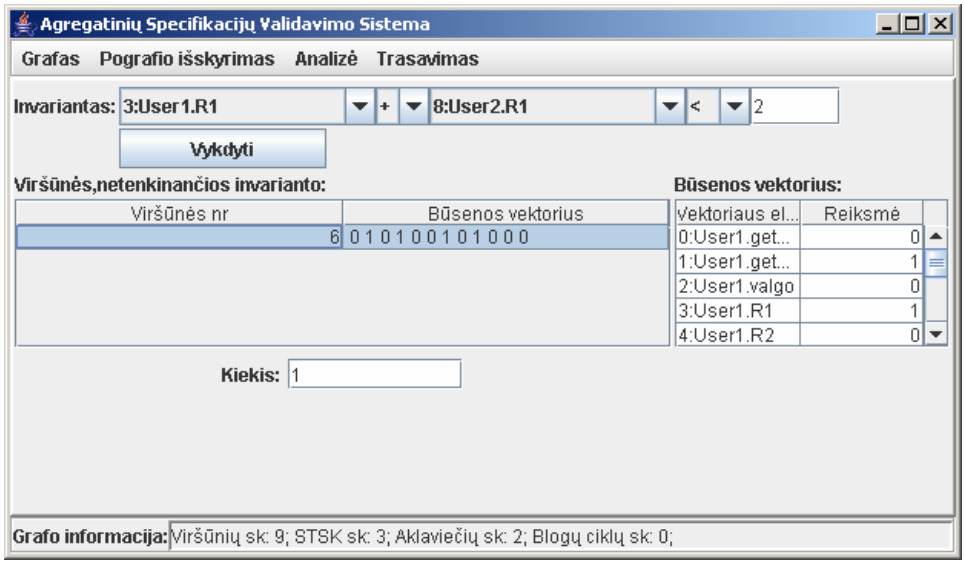
29 pav. Vienos iš aklavičių (šeštos viršūnės) būsenos vektorius

Ciklas gaunasi tuomet kai abiejų procesų darbas baigėsi ir sistema grįžta į pradinę būseną. Mūsų nagrinėjamame pavyzdyje toks ciklas negali būti laikomas neprogresyviu, kadangi sistemoje turime tiktai šiuos du procesus. Tačiau didesnėje sistemoje aptiktas ciklas būtų neprogresyvus, kadangi nėra galimybės perduoti sugeneruotų signalų į ciklo išorę.

Aprašysime nagrinėjamai sistemai paprastą invariantą – „abu procesai vienu metu negali naudotis resursu R1“. Sistemoje tokia išraiška fiksuojama tokiu būdu:

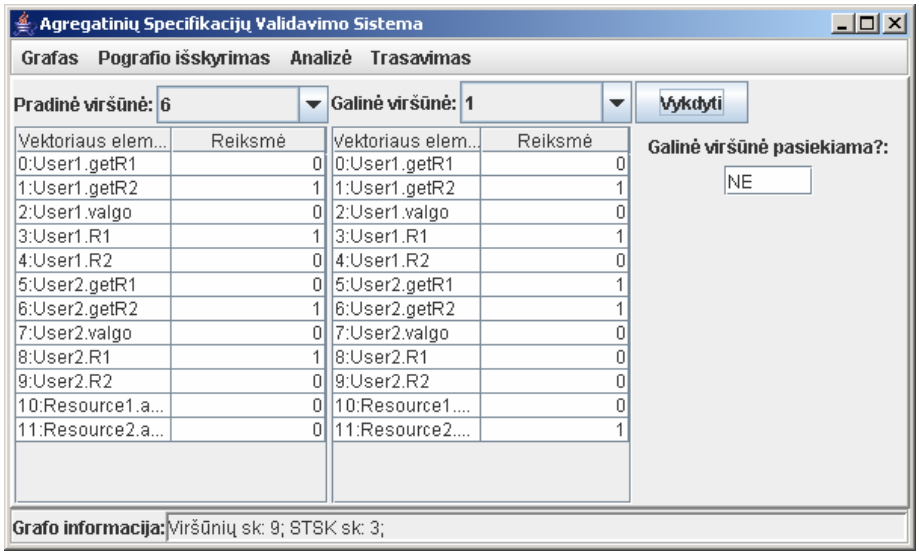
30 pav. Invarianto išraiškos užrašymas VALSYS sistemoje

Kitaip sakant, suma dviejų būsenos vektoriaus parametrų negali būti lygi dviems. Šeštoji būsena netenkina aprašyto invarianto, kadangi šita būsena yra aklavietė.

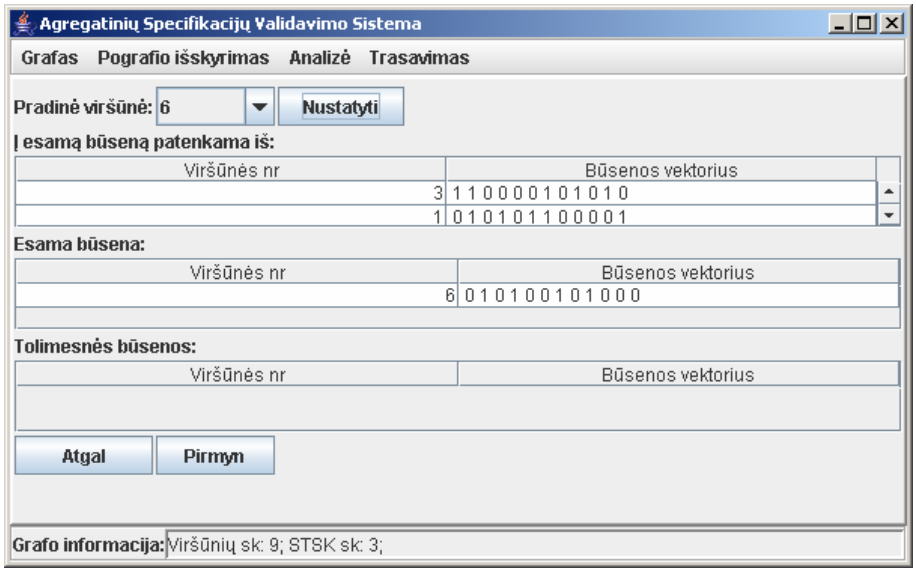


31 pav. Būsena-aklavietė netenkina sistemos invarianto

Natūralu, jog patekusios į aklavietę sistemos darbas turės būti nutrauktas, kadangi sistemai nėra galimybės pereiti į kitas būsenas. Įsitikinti tuo galima patikrinus vienos iš kitos būsenų pasiekiamumą arba tiesiog trasuojant grafą.



32 pav. Iš būsenos-aklavietės kitos būsenos yra nepasiekiamos

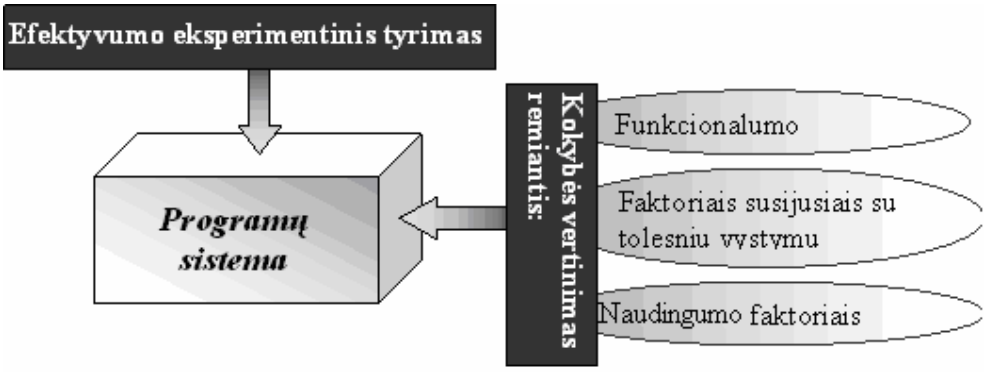


33 pav. Būsena-aklavietė neturi „įpėdinių“

5. TYRIMO DALIS

5.1. PRAKTINĖ SISTEMOS VEIKLOS ANALIZĖ

Šiame skyriuje apžvelgiamas atliktas agregatinių specifikacijų validavimo sistemos vertinimas ir gauti vertinimo rezultatai.



34 pav. Sukurtos sistemos vertinimo pjūviai

5.1.1. Sistemos efektyvumo tyrimas

Eksperimentams atlikti buvo sugeneruoti keli adaptyvaus trakto protokolo pasiekiamų būsenų grafai (su skirtingu viršūnių skaičiumi). Atlikti eksperimentai, kurių metų buvo fiksuojamas operacijos atlikimo laikas.

Eksperimentas 1.

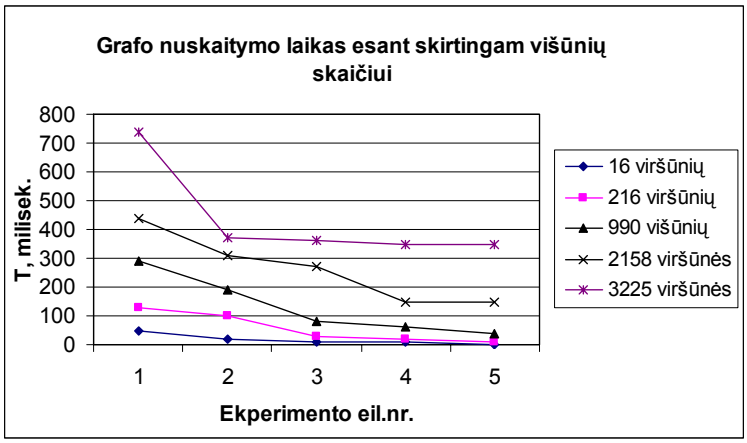
Operacija: pasiekiamų būsenų grafo formavimas.

Iteracijų skaičius: 5.

Suformuoti grafą reiškia nuskaityti duomenis iš vartotojų nurodytų viršūnių ir briaunų failų (viso du failai) bei nustatyti „viršūnė-briauna“ ryšius.

To pačio grafo nuskaitymas buvo atliktas penkis kartus iš eilės ir fiksuotas operacijos laikas. Kadangi suformuotas grafas saugojamas atsargos atmintinėje (angl. *cache*), kiekvienas sekantis nuskaitymas trunka trumpiau (esant sąlygai, kad kompiuterio resursų dalijimasis kitomis taikomosiomis programomis nesikeičia).

Iš grafiko matome, kad nuskaitymui gaištamas laikas mažėja su kiekviena nauja iteracija, tačiau anksčiau ar vėliau pasiekiama mažiausiai galima operacijos atlikimo trukmė. Ši trukmė priklauso nuo grafo dydžio.



35 pav. Pirmo eksperimento rezultatai

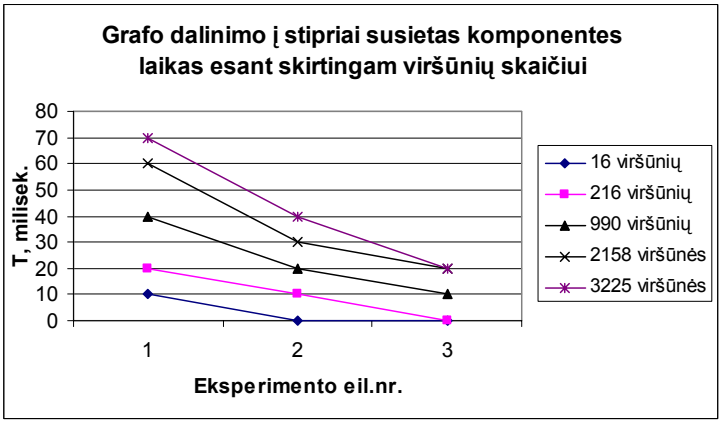
Eksperimentas 2.

Operacija: stipriai susietų komponenčių išskyrimas grafe.

Iteracijų skaičius: 3.

Pakartosime eksperimentą su kita operacija – dalinsime grafą į stipriai susietas komponentes.

Tris kartus iš eilės atlikus tą pačią operaciją stebima operacijos atlikimo laiko mažėjimo tendencija.



36 pav. Antro eksperimento rezultatai

IŠVADA:

Duomenų struktūrų (ir tarpinių rezultatų) saugojimas atsargos atmintinėje įgalina racionaliai naudoti turimus kompiuterio resursus.

5.1.2. Funkcionalumo tyrimas

3 lentelė. VALSYS įrankio vertinimas pagal kokybės kriterijus

Kokybės kriterijus	Programinės įrangos atitikimo aprašymas
Išbaigtumas	Nerealizuota viena iš numatytų funkcijų: kelių stipriai susietų komponenčių apjungimas.
	Posistemis gali būti naudojamas praktiniams darbams atlikti.
Efektyvumas	Java kalbos panaudojimas užtikrina minimalius reikalavimus techninei bei programinei įrangai vartotojo darbo vietoje.
	Įdiegta sistema kompiuterio kietajame diske užima 400 KB.
	Resursų naudojimas: apie 30 KB operatyviosios atmintinės.
Panaudojamumas	Vartotojo sąsaja yra lengvai suprantama, tačiau nėra skirta atsitiktiniam vartotojui. Vartotojas turi turėti žinių iš sistemų formalizavimo ir validavimo sričių, kad pagal meniu punkto pavadinimą galėtų numatyti funkcijos paskirtį bei rezultatą.
Patikimumas	Atliekama duomenų korektiškumo kontrolė.
	Vartotojas informuojamas apie sėkmingą/nesėkmingą operacijos atlikimo rezultatą.
Palaikomumas	Sistema yra specifikuota bei dokumentuota.
	Atskirų dalių funkcionalumas gali būti nesunkiai pakeistas be didelės įtakos visos sistemos darbui.
Lankstumas	Vartotojo aplinka negali būti tinkinama. Vartotojas mažai gali įtakoti sistemos funkcionalumą.
Pernešamumas	Sistema nėra instaliuojama, t. y. jos veikimui nėra kuriami darbą palaikantys failai.
	Vykdomasis failas paleidžiamas tiesiogiai iš bet kurios vietos kompiuterio diske, kur jis buvo patalpintas.
Pakartotinis panaudojamumas	Sistema gali veikti kaip savarankiškas modulis arba būti programiškai prijungta prie bendros paskirstytų sistemų analizės sistemos.
	Atskiri moduliai (pvz., duomenų nuskaitymas ir formavimas) gali būti panaudoti kitose sistemose.
Sąsajos su kitomis programomis galimybė	Sąsaja su skaitmeninio modeliavimo posistemiu – validavimui naudojamas jo pagalba sugeneruotas pasiekiamų būsenų grafas. Dėl šios priežasties bet kokie duomenų failai, gauti ne modeliavimo būdu o sukurti vartotojų, turi griežtai atitikti duomenų formatui keliamus reikalavimus.

5.1.3. Pastebėta sistemos nauda

1. Sukurtas validavimo įrankis atlieka pasiekiamų būsenų grafo (sugeneruoto iš sistemos agregatinės specifikacijos) analizę:
    - Aptinkamos aklavietės, neprogresyvūs ciklai (tikrinama sistemos saugumo savybė).
    - Atliekamas pografio išskyrimas pagal vartotojo užduotus parametrus (pagal briaunų svorius arba pasirinkus būsenų aibę).
    - Būsenų koordinačių ribų tikrinimas (tikrinama sistemos gyvybingumo savybė).
    - Grafo trasavimas (su galimybe nurodyti pradinę sistemos būseną).
    - Sistemos invariantų tikrinimas.
2. Vartotojo grafinė sąsaja neapkrauta, intuityviai suprantama ir lengvai valdoma.
3. VALSYS įrankis gali būti sėkmingai naudojamas studentų laboratoriniams darbams atlikti.

5.1.4. Pastebėti sistemos trūkumai

Didžiausias sistemos trūkumas – operacijų rezultatai nėra išvedami į išorines duomenų laikmenas. Vartotojas negauna sistemos darbo ataskaitos, kurioje matytųsi informacija apie analizuojamo grafo viršūnių kiekį, aptiktų aklaviečių/neprogresyvių ciklų kiekį, stipriai susietas komponentes sudarančių viršūnių sąrašai, kita. Šiuo metu tokia informacija yra išvedama į ekraną, tačiau niekur nefiksuojama.

Kitas galimas sistemos patobulinimas – grafinė vartotojo sąsają. Ateityje jinai galėtų būti papildyta grafiniu sistemos būsenų grafo atvaizdavimu. Šiuo metu vartotojas gali operuoti tiktai būsenų numeriais.

6. IŠVADOS

- Vieno ar kito formalizmo vartojimas sistemos specifikacijai sudaryti priklauso tiktai nuo sistemos projektuotojo patirties, sistemos sudėtingumo ir paskirties.
- Formaliųjų metodų naudojimas sudėtingų sistemų projektavimo etape leidžia mažesniais kaštais kurti patikimesnę bei kokybiškesnę programinę įrangą.
- Formaliojo sistemos aprašo sudarymą ir jo panaudojimą specifikacijos kodo generavimui galima nagrinėti kaip heterogeninį metaprogramavimą. Šiuo atveju pasirinkta matematinė formalizavimo schema (SDL, agregatai, kita) būtų aukšto lygio metaprogramavimo kalba, o tikslo kalbai galėtų atstovauti Promela, Estelle, XML bei kitos programavimo kalbos.
- Pakartotinio panaudojimo technologija įgalina ne tik programinių modulių, bet ir formaliai aprašytų sistemų komponentų vartojimą kitose kontekstuose, kitų specifikacijų ribose.
- Vienas iš patogiausių sistemų specifikacijų analizės metodų yra pasiekiamumo analizė, kadangi jis lengvai gali būti automatizuotas ir paprasta forma perteikia sistemos tyrimo rezultatus.
- Statinio apriboto pasiekiamumo analizės metodo privalumas yra baigtinis būsenų skaičius. Jo trūkumas – pasiekiamų būsenų aibė yra apibrėžta vienareikšmiškai be garantijos kad jinai apima visas svarbias sistemos būsenas.
- Sukurtas validavimo įrankis VALSYS leidžia tikrinti pagrindines sistemos savybes - saugumo ir gyvybingumo.
- Sudėtingumo valdymui VALSYS įrankyje taikomas redukcijos metodas – pilnas sistemos grafas dalinamas į stipriai susietas komponentes ir į pografius tam, kad apriboti pasiekiamų būsenų skaičių iki priimtino vartotojui.

SANTRUMPŲ IR TERMINŲ ŽODYNAS

ITU
International Telecommunications Union – Tarptautinė Telekomunikacijų sistemų Sąjunga.

Paskirstytoji sistema
Paskirstytoji sistema arba paskirstytieji skaičiavimai yra kelių procesų veikimo agregavimas tokiu būdu, kad kartu jie sudaro vieningą, centralizuotą sistemą.

PLA
Piece-linear aggregate formalism – atkarpomis tiesinių agregatų formalizmas.

UML
Unified Modeling Language – unifikuota modeliavimo kalba.

LITERATŪRA

(citavimo tekste eilės tvarka)

[1] Holzmann G. J. *Design and validation of computer protocols*. Prentice Hall, 1990.

[2] Biere A., Cimatti A., *Bounded Model Checking*. Vol. 58 of Advances in Computers, 2003.

[3] Хмеляускас А. В., *Создание средств спецификации и комплексного исследования протоколов сетей ЭВМ*. Каунас: 1989.

[4] Gao Q., Groz R., Bochmann G., Dargham J., Houssain Htite E., *Validation of distributed algorithms and protocols*. 1995.

[5] *Basic Spin Manual,* 1996. [Žiūrėta 2005.04.28]. Prieiga per internetą: <www.spinroot.com/spin>

[6] Buslenko N. *On a class of complex systems. Problems of applied mathematics and mechanics.* Maskva: Nauka, 1971.

[7] Bochmann G. V. *Finite State Description of Communication Protocols*. Liege : Proc. Computer Network Protocols Symp., 1978.

[8] Pranevičius H., Tumelis G., Germanavičius V., *The use of formal description of systems' behavior creating Markov models*. Kaunas: Informacinės technologijos ir valdymas, 2004, Nr.3(32), p. 55-60 .

[9] Pranevičius H., *Kompiuterių tinklų protokolų formalusis specifikavimas ir analizė: agregatinis metodas*. Monografija. Kaunas, 2004.

[10] Groote J. F., Reniers M. A. *Algebraic process verification.*

[11] Graham D. R., *Testing, Verification and validation.* 1996.

[12] Tarjan R. E. *Depth-first search and linear graph algorithms*, SIAM: Journal on Computing, 1(2), p. 146-160, 1972.

[13] Doldi L. *Validation of Communications Systems with SDL: the art of SDL simulation and reachability analysis*. ISBN: 0-470-85286-0.

[14] BALTPORTS-IT (IST-2001-33030) project. *Simulation and IT-Solutions: Applications in the Baltic Port Areas of the Newly Associated States.* Deliverable D7.2 “Description of business process structure in Klaipeda sea port using PLA formalism”, 2003.

[15] Babich F., Deotto L., *Formal Methods for Specification and Analysis of Communication Protocols.* EEE Communications Surveys, 2002.

[16] Holzmann G. J., *Algorithms for automated protocol validation*. AT&T Technical Journal, Vol.69, No.2, 1988.

[17] Boehm B., *Software Risk Management.* IEEE Computer Society Press, 1989.

[18] Courcoubetis C., Vardi M. Y., Wolper P., Yannakakis M., *Memory Efficient Algorithms for the Verification of Temporal Properties.* Formal Methods in Systems Design, vol. I, p. 275-288, 1992.

[19] Holzmann G. J., Peled D., Yannakakis M., *On Nested Depth-First Search*. Proc.Second SPIN Workshop, Rutgers Univ., New Brunswick, N.J., DIMACS/32, Am. Math. Soc., Aug. 1996.

[20] Tiwari A., Rueß H., Saïdi H., Shankar N., *A Technique for Invariant Generation.* Springer-Verlag, Genova, Italy, 2001.

[21] Holzmann G. J., *Model checker SPIN*. IEEE transactions on software engineering, VOL. 23, NO. 5, MAY 1997.

PRIEDAI

Priedas 1. Sistemos VALSYS įėjimo duomenų pavyzdys („pietaujančių filosofų“ uždaviniui)

Viršūnių failas

```
NODES=9
@
0:1.0 1.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 1.0 1.0
1:0.0 1.0 0.0 1.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 1.0
2:1.0 0.0 0.0 0.0 1.0 1.0 1.0 0.0 0.0 0.0 1.0 0.0
3:1.0 1.0 0.0 0.0 0.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0
4:1.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0
5:0.0 0.0 0.1 1.0 1.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0
6:0.0 1.0 0.0 1.0 0.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0
7:1.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0
8:1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.1 1.0 1.0 0.0 0.0
```

Briaunų failas

```
EDGES=14
@
0:1:1.0:User1.getR1:,User1.getR1,Resource1.X2,User1.X1.
0:2:1.0:User1.getR2:,User1.getR2,Resource2.X1,User1.X2.
0:3:1.0:User2.getR1:,User2.getR1,Resource2.X2,User2.X1.
0:4:1.0:User2.getR2:,User2.getR2,Resource1.X1,User2.X2.
1:5:1.0:User1.getR2:,User1.getR2,Resource2.X1,User1.X2.
1:6:1.0:User2.getR1:,User2.getR1,Resource2.X2,User2.X1.
2:5:1.0:User1.getR1:,User1.getR1,Resource1.X2,User1.X1.
2:7:1.0:User2.getR2:,User2.getR2,Resource1.X1,User2.X2.
3:6:1.0:User1.getR1:,User1.getR1,Resource1.X2,User1.X1.
3:8:1.0:User2.getR2:,User2.getR2,Resource1.X1,User2.X2.
4:7:1.0:User1.getR2:,User1.getR2,Resource2.X1,User1.X2.
4:8:1.0:User2.getR1:,User2.getR1,Resource2.X2,User2.X1.
5:0:0.1:User1.valgo:,User1.valgo,Resource1.X2,Resource2.X1.
8:0:0.1:User2.valgo:,User2.valgo,Resource2.X2,Resource1.X1.
```

Būsenos vektoriaus koordinačių aprašymo failas

```
COORDINATES=12
@
0:User1.getR1
1:User1.getR2
2:User1.valgo
3:User1.R1
4:User1.R2
5:User2.getR1
6:User2.getR2
7:User2.valgo
8:User2.R1
9:User2.R2
10:Resource1.available
11:Resource2.available
```