VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Baigiamasis bakalauro darbas

Sistemos laiko savybių verifikavimas, naudojant modelių tikrinimo sistemą Uppaal-SMC

(Verification of System Timing Properties using the Model Checker Uppaal-SMC)

Atliko: 4 kurso 3 grupės studentas
Paulius Aleksiūnas (parašas)
Darbo vadovas:
Prof., Dr. Linas Laibinis (parašas)
Recenzentas:
Prof., Dr. Saulius Gražulis (parašas)

Vilnius
2020

Turinys

Sąvokų apibrėžimai

DAB - duomenų apdorojimo blokas.

LLL - linijinė laiko logika.

SML - skaičiavimo medžio logika.

SMT - statistinis modelių tikrinimas.

TC - telekomanda.

TM - telemetrijos duomenys.

Įvadas

Šis bakalauro baigiamasis darbas pristato, kaip tikrinti sistemos savybes naudojant statistinį modelių tikrinimo (SMT) įrankį Uppaal-SMC. Šio darbo tikslas yra susipažinti su SMT teorija ir Uppaal-SMC įrankiu bei jo pagalba ištirti sistemos atvejo analizę - BepiColombo erdvėlaivio Duomenų Apdorojimo Bloką (DAB) (angl. Data Processing Unit). BepiColombo - tai pirmoji Europos Kosmoso Agentūros (ang. European Space Agency, ESA) misija į Merkurijaus planetą. DAB yra erdvėlaivio sistemos dalis, kuri apdoroja BepiColombo telekomandas (TC) ir perduoda atgal mokslinius ir diagnostikos telemetrijos duomenis (TM). SMT - tai sudėtingų sistemų modelių tikrinimo būdas, kur yra tikrinamos sistemos reikalaujamos savybės remiantis simuliavimo ir statistikos metodais. Šis tikrinimas yra alternatyvus metodas lyginant su klasikiniu modelių tikrinimo būdu, kuris (žymia dalimi) išvengiantis būsenų erdvės sprogimo problemos.

Praktinėje darbo dalyje yra atliekami BepiColombo DAB sistemos modelių pakeitimai ir efektyvumo analizė Uppaal-SMC įrankio pagalba. Ankstesni sistemos modeliai [IRL+12] buvo sukurti naudojant klasikinio modelių tikrinimo sistemą Uppaal.

Šio darbo uždaviniai:

- susipažinti su SMT teorija;
- susipažinti su Uppaal-SMC įrankiu;
- susipažinti su BepiColombo DAB sistema;
- atlikti pakeitimus, kad sistema veiktų Uppaal-SMC įrankyje;
- pritaikyti sistemos modeliams Uppaal-SMC įrankio savybes: eksponentiniai dažniai ir tikimybiniai svoriai;
- atlikti sistemos efektyvumo analizę.

Tolesnė darbo struktūra yra tokia:

- pirmajame skyriuje yra supažindinama su SMT teorija;
- antrajame skyriuje yra apžvelgiamas Uppaal-SMC įrankis ir pagrindinės jo savybės;
- trečiajame skyriuje yra realizuojama ir tikrinama BepiColombo DAB sistema Uppaal-SMC įrankio pagalba;
- paskutiniuose skyriuose pateikiamos praktinės dalies darbo išvadas ir reziumė.

1. Modelių tikrinimas

SMT - tai kompromisas tarp pilno sistemos modelių verifikavimo (angl. verification) ir testavimo, kur yra stebimas sistemų vykdymas ar simuliavimas ir statistiniais metodais įvertinama, ar sistema tenkina tyrinėjamą (dažnai sistemos reikalavimuose apibrėžtą) savybę.

Šiuolaikiniame pasaulyje kompiuteriai užima svarbią vietą ir jų klaidos gali turėti dramatiškas pasekmes. Todėl nenuostabu, kad kompiuterinių sistemų teisingumo (angl. correctness) įrodymas yra ypač aktuali problema. Labiausiai paplitęs sistemos teisingumo įrodymas yra testavimas [BJK+05]. Testavimui naudojami testiniai atvejai (angl. test cases) su numatomais rezultatais (angl. predicted outcomes). Testavimo metodai dažnai yra veiksmingi, kai norime surasti sistemos klaidas (angl. bugs). Tačiau testavimas nėra tinkamas visiems sistemos teisingumo įrodymo atvejams, nes bendrai neįmanoma apibrėžti baigtinio testinių atvejų rinkinio, kuris patikrintų visas galimas sistemos vykdymo situacijas. Dėl šios priežasties klaidos (angl. errors) gali likti nepastebėtos.

Taip pat yra metodų, kurie užtikrina pilną sistemos ar bent jos modelio teisingumą. Šie metodai vadinami formaliais metodais ir remiasi matematiniais ir loginiais metodais, kurių pagalba tikrinama, ar sistema elgiasi teisingai visais galimai atvejais. Egzistuoja keletas matematinių sistemos vaizdavimo būdų. Šiame darbe bus aptartas parėjimų sistemų (angl. transition systems) vaizdavimo būdas. Perėjimų sistemos elgesys gali būti vaizduojamas galimai begalinėmis būsenų pokyčių ir laiko žymų (angl. time stamps) sekomis, kurias vadiname vykdymais (angl. executions). Sąryšis tarp parėjimų būsenų gaunamas remiantis taip vadinamu parėjimų sąryšiu (angl. transition relation). Šis sąryšis gali būti begalinis (angl. infinite) ir netiesioginis (angl. implicit).

Formalių metodų istorija yra ilga, pradedant nuo loginių įrodymų ir invariantų iki modelių tikrinimų [BK08]. Šiame darbe apžvelgsime tik modelių tikrinimą. Šis metodas tyrinėja sistemos būsenų erdvę (angl. state-space), kad galėtų patikrinti, ar kiekvienas sistemos elgesys tenkina nurodytus reikalavimus. Klasikiniuose modelių tikrinimo metoduose reikalavimai dažnai yra išreiškiami naudojant laiko logikas, tokias kaip linijine laiko logika (LLL) (angl. Linear Temporal Logic, LTL) [Pnu77] arba skaičiavimo medžio logika (SML) (angl. Computation Tree Logic, CTL) [CE81]. Šios logikos praplečia klasikinę Būlio (angl. Boolean) logiką pridedant laiko operatorius, kurie leidžia mums analizuoti laiko dimensiją nurodyto vykdymo metu.

Modelių tikrinimas susideda iš pakartotinų pasiekiamų būsenų skaičiavimų [CGK+18]. Paprastas būsenų erdvės tyrinėjimo metodas pradeda darbą nuo pradinių būsenų rinkinio ir tada prideda naujas pasiekiamas būsenas taikant pasiekiamumo sąryšius (angl. reachability relation), kuris yra pagrįstas anksčiau minėtu perėjimų sąryšiu. Jeigu būsenų skaičius yra baigtinis, tai kartojant šią operaciją galiausiai gausime stabilų rinkinį, kuris atitiks pasiekiamų sistemos būsenų rinkinį. Tačiau net ir paprastoms sistemoms baigtinių būsenų erdvės gali būti per didelės, kad kompiuteriai galėtų atlikti skaičiavimus ir juos atvaizduoti. Jau keletą dešimtmečių ieškoma kaip sumažinti resursus reikalingus tokiems skaičiavimams, siekiant išvengti ”būsenų erdvės sprogimo” problemos.

Pirmieji didelės būsenų erdvės problemos sprendimai bandė išnaudoti skirtingų sistemos vykdymo scenarijų panašumus ir pasikartojančią informaciją. Pavyzdžiui, tokiems sprendimams pri-

klauso dalinės redukcijos (angl. partial reduction) metodas [FG05; WG93]. Šis metodas išvengia papildomo būsenų sekų tyrinėjimo parodydamas, kad jų poveikis jau yra užfiksuotas nagrinėjant kitas būsenų sekas. Kitas metodas yra bisimuliacijos redukcija (angl. bisimuliation reduction) [DPP04], kuris išnaudoja panašių būsenų (būsenos, kurios sukuria vienodą elgesį) lygiavertiškumo klases (angl. equivalence classes), kad sumažintų tiriamų būsenų erdves. Predikatų abstrakcijos metodai [BMR05] išplečia bisimuliacinę redukciją abstrahuodami rinkinius su nurodytu predikatu. Predikatų abstrakcija paremti metodai gali būti suderinti su paneigiančių pavyzdžių (angl. counter examples) metodais, kurie naudojami abstrakcijos tikslumo nustatymui [CV03].

Neskaitant būsenų erdvės skaičiavimo, viena iš didžiausių modelių tikrinimo problemų yra efektyvus būsenų rinkinių atvaizdavimas. Vieni iš pirmųjų sprendimų yra paremti simboliniais metodais, kurie naudoja simbolinį vaizdavimą numanomam būsenų rinkinių valdymui. Simboliniams metodams pavyko išplėsti paprastų analizės metodų pritaikymą sistemoms, kurios turi didelius būsenų rinkinius. Vienas iš labiausiai naudojamų simbolinių atvaizdavimo būdų yra Dvejetainės Sprendimų Diagramos (angl. Binary Decision Diagrams) [Bry92], kur sistemos būsenos yra užkoduotos fiksuoto ilgio bitų vektoriais. Naudojant šį vaizdavimo būdą baigtinis būsenų rinkinys gali būti traktuojamas kaip Būlio formulių sprendimų rinkinys, kuris dažnai yra labiau kompaktiškas negu atitinkamos logiškai ekvivalenčios konjunkcinės ar disjunkcinės formos. Taip pat Dvejetainės Sprendimų Diagramos vaizdavimo būdas yra algoritmiškai lengvai apdorojamas ir leidžia efektyviai atvaizduoti įprastines (angl. regular) struktūras, kurios dažnai atsiranda sistemos pasiekiamų būsenų rinkiniuose. Šio metodo pagalba buvo tikrinama sistema, kuri turi daugiau negu 10^{20} pasiekiamų būsenų [BCM+92]. Per pastarąjį dešimtmetį Dvejetainės Sprendimų Diagramų metodas buvo pakeistas loginiu vaizdavimo būdu, kur būsenų sekos vaizduojamos formulėmis ir naudojami patenkinamumo sprendėjai (angl. satisfiability solvers, SAT solvers) patikrinimui, ar būsena yra pasiekiama [BCC+99; CCQ02].

Porą dešimtmečių loginiai ir formalūs modeliai neišnaudojo bei nemodeliavo realaus laiko (angl. real-time) ir tikimybinės informacijos. Tačiau tokia informacija yra reikalinga, kai nagrinėjame tokias dideles sistemas kaip išskirstytas (ang. distributed), įterptines (angl. embedded), kibernetiškai fizines (angl. cyber physical) ir biologines sistemas. Pavyzdžiui, gali būti svarbu apskaičiuoti, kiek reikės energijos, kad išlikti virš tam tikros ribos arba kiek reikės laiko, kol bus pasiekta nurodyta būsena. Dėl tokio tipo informacijos gavimo problemos buvo praplėstos perėjimo sistemos, kad galėtų apdoroti kiekybines savybes. Taip atsirado:

- laiko automatų formalizmas [Alu99], kuris išnaudoja realaus laiko informaciją;
- stochastinės sistemos, kurios gali nagrinėti neapibrėžtumą (angl. uncertainty) vykdymuose;
- svoriniai (angl. weighted) automatai, kurie leidžia kiekybiškai ar tikimybiškai įvertinti perėjimų rinkinių svorius [DG09];
- LLL ir SML praplėstos pridedant laiko ir kiekybinę informaciją.

Šie formalizmai buvo dažnai aptarinėjami mokslinėje literatūroje ir net išsiplėtė į kitas sritis, pavyzdžiui, energijos automatų arba hibridinių sistemų sritis. Buvo pastebėta, kad pridedant kiekybinę informaciją padidėja būsenų erdvės sprogimo problema. Tačiau tokie įrankiai kaip Uppaal arba PRISM suteikia efektyvius metodus susidoroti su šia problema.

1.1. Statistinis modelių tikrinimas

Tarp žinomų perėjimo sistemų plėtinių egzistuoja kiekybinės sistemos (angl. quantitative sytems), kurių perėjimams yra pritaikomi tikimybiniai pasiskirstymai. Tokioms sistemos priklauso diskretaus ir tęstinio (angl. continuous) laiko Markov Grandinės (angl. Chains). Pagrindinį dėmesį skirsime nurodytos stochastinės sistemos savybių tikimybių skaičiavimams. Šis kiekybinis įvertinimas pakeičia Būlio logiką ir leidžia kiekybiškai įvertinti padarytų pakeitimų poveikį nurodytai sistemai.

Kaip ir klasikinės perėjimo sistemos, kiekybinės stochastinių sistemų savybės dažniausiai yra aprašomos tiesine laiko logika, kuri leidžia palyginti vykdymus, kur yra tenkinamos tam tikros laiko savybės su slenksčiais (angl. tresholds). Stochastinių sistemų modelių tikrinimo problema dažniausiai išsprendžiama būsenos erdvės tyrinėjimu, t. y., iteratyviai skaičiuojant arba tikslinant (angl. approximates) kelius, kurie tenkina tinkamas subformules. Tokių kelių skaičiavimo algoritmo pasirinkimas priklauso nuo tyrinėjamas stochastinės sistemos klasės ir naudojamos logikos, kuri yra skirta aprašyti savybių teisingumui. Modelių tikrinimo algoritmai yra aprašyti šiuose: [BHH+03; CG04; CY95] ir kituose darbuose. Taip pat egzistuoja tinkami įrankiai [BC06; KNP04], kurie yra praktiškai naudojami įvairių tipo sistemų analizei.

Modelių tikrinimo algoritmai yra tinkami tik specialioms sistemoms, kurios atitinka tam tikrus struktūrinius reikalavimus, t.y., gali būti išreikštos kaip perėjimų sistemos. Taip pat šie algoritmai reikalauja daug laiko ir atminties, todėl didelių sistemų plėtimas (angl. scaling) tampa rimta problema. Be to modelių tikrinamo algoritmams naudojama logika yra praplėsta iš klasikinės laiko logikos, kuri dažnai nėra populiari tarp sistemų inžinierių. Galiausiai šie algoritmai neleidžia svarstyti apie apie praplėstus stochastinius modelius, kurių semantika priklauso nuo realaus laiko arba energijos savybių.

Kitas būdas patikrinti stochastinių sistemų kiekybines savybes yra simuliuoti (angl. simulate) sistemą be galo daug kartų ir panaudoti metodus iš statistikos srities. Šių metodų pagalba galime nustatyti, ar gautos simuliacijos rezultatai statistiškai įrodo arba paneigia nagrinėjamą savybę [YS02]. Stochastinių sistemų simuliacijos gaunamos pagal sistemos apibrėžtus pasiskirstymus ir leidžia gauti vykdymų tikimybinius įvertinimus. Šie metodai vadinami statistiniu ar tikimybiniu modelių tikrinimu.

SMT turi daug privalumų. Pirma, šio tikrinimo algoritmai reikalauja tik, kad sistema būtu simuliuojama, t.y., vykdomi pavyzdiniai (angl. sample) sistemos vykdymai. Dėl šios priežasties SMT algoritmai gali būti pritaikomi didesnei sistemų klasei negu klasikinio modelių tikrinimo. Pavyzdžiui, šiuos algoritmus galime pritaikyti juodosios dėžės (angl. black-box) arba begalinių

būsenų sistemoms. Antra, šis metodas gali būti pritaikytas platesnėms savybių klasėms. Trečia, SMT algoritmai yra lengvai lygiagretinami (angl. parallelizable), o tai gali padėti plėsti dideles sistemas. Jeigu sprendžiam problema yra neišsprendžiama (angl. undecidable) arba per daug sudėtinga, tai SMT dažnai būna vienintelis galimas sprendimo būdas. Šio metodo algoritmai yra pritaikyti šiuose įrankiuose: Uppaal [DLL+11], PRISM [KNP11], Ymer [You05a] ir Plasma Lab [BCL+13].

Pristatytas SMT metodas neišsprendžia visų sistemos savybių patikrinimo problemų. Pavyzdžiui, šis metodas negali patikrinti neapibrėžtų (angl. unbounded) savybių. Taip pat SMT pilnai neišvengia būsenų erdvės sprogimo problemos, kuri susijusi su mažų tikimybių, t.y., retų įvykių (angl. rare events), skaičiavimu. Be to šis metodas dar nėra populiarus verslo srityje. Todėl nėra tiksliai žinoma, kokiu mastu šis metodas gali susidoroti su įvairiomis sudėtingomis ”realaus pasaulio” sistemomis [LL16].

2. Uppaal-SMC įrankis

Uppaal - tai integruota realaus laiko sistemų modeliavimo, simuliavimo ir verifikavimo įrankių aplinka. Šis įrankis yra tinkamas tokioms sistemoms, kurioms yra tinkamas modeliavimas panaudojant nedeterministiškus (angl. non-deterministic) procesus su baigtinėmis valdymo struktūromis (angl. finite control structures) ir realaus laiko laikrodžiais [YPD95a; LPY97]. Uppaal yra sudarytas iš trijų pagrindinių dalių: sistemos modelių aprašymo kalbos, simuliavimo ir modelių tikrinimo.

Uppaal-SMC - tai Uppaal įrankio išplėtimas, kuri atvaizduoja sistemas kaip laiko automatų (angl. timed automaton) tinklus, kurių elgsena gali priklausyti ir nuo stochastinių ir netiesinių dinaminių savybių. Šiame įrankyje kiekvienas sistemos modelis atitinka automatą, kurio laikrodžiai (angl. clocks) gali vystytis skirtingais dažniais (angl. rates). Uppaal-SMC pritaiko SMT teoriją [You05b; SVA04], kad galėtų efektyviai analizuoti tikimybines atlikimo (angl. performance) savybes. Sistemos tikrinimai yra paremti simuliaciniais metodais, kurie yra pranašesni už išsekinančius (angl. exhaustive) atminties, laiko ir išraiškos atžvilgiu.

2.1. Modeliavimo formalizmas

Uppaal-SMC modeliavimo teorija yra paremta laiko automatų formalizmo [AD94] stochastine interpretacija. Laiko automatų formali teorija yra naudojama klasikinėje Uppaal versijoje [BDL04]. Atskiriems laiko automatų komponentams stochastinė interpretacija pakeičia nedeterministiškus (angl. non-deterministic) galimų perėjimų (angl. transitions) pasirinkimus į tikimybinius. Šie tikimybiniai pasirinkimai gali būti automatiškai nustatomi Uppaal-SMC arba apibrėžti įrankio naudotojo. Panašiai nedeterministiški laiko uždelsimo (angl. time delay) pasirinkimai pakeičiami tikimybiniais pasiskirstymais (angl. distributions). Laiku apriboto uždelsimo atveju naudojami suvienodinti (angl. uniform) pasiskirstymai, o neapriboto (angl. unbounded) uždelsimo atveju naudojami eksponentiniai pasiskirstymai kurių dažnius apibrėžia naudotojas.

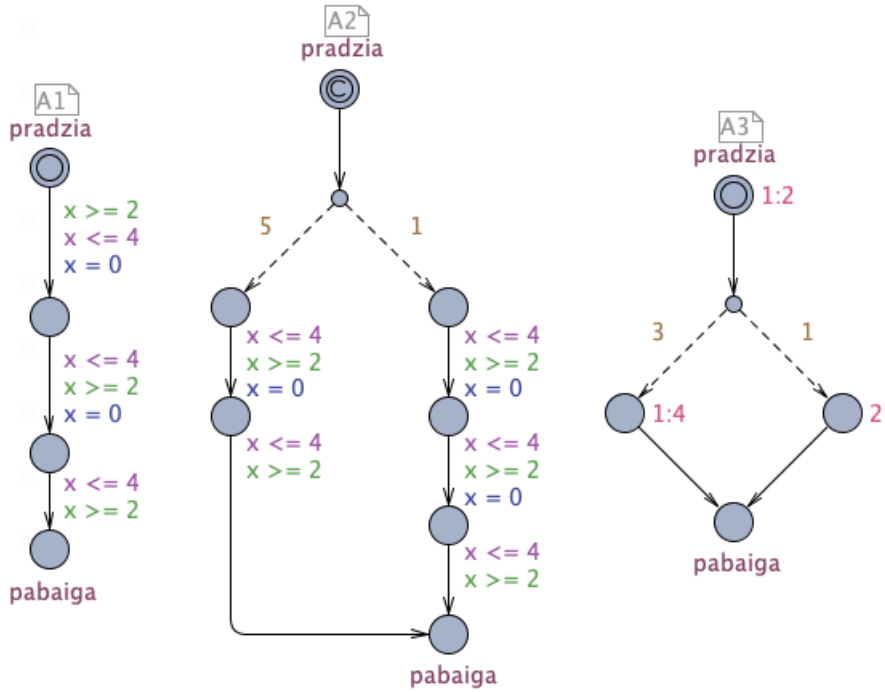
Uppaal-SMC yra naudojamos lokacijos (angl. locations), kurios yra sujungtos perėjimais (briaunomis) (angl. edges). Galimi lokacijų tipai:

- paprasta - lokacija, kuri neturi papildoma funkcionalumo;
- pradinė (angl. initial) - kiekvienas modelis privalo turėti vieną pradinę lokaciją. Ši lokacija žymi proceso pradžią;
- neatidėliotina (angl. urgent) - laikas negali eiti, kai procesas yra neatidėliotinoje lokacijoje;
- įsipareigojusi (angl. commited) - veikia panašiai kaip neatidėliotina lokacija. Jei bet kuris procesas yra įsipareigojusioje lokacijoje, tai sekantis perėjimas turi eiti iš įsipareigojusios lokacijos.

Lokacija gali turėti apibrėžtą invariantą arba eksponentinį dažnį. Invariantas nurodo lokacijos sąlygas, kurios gali būti aprašomos laikrodžių apribojimais arba skirtumais ir Būlio išraiškomis. Eksponentiniai dažniai nurodo lokacijos tikimybinį uždelsimą.

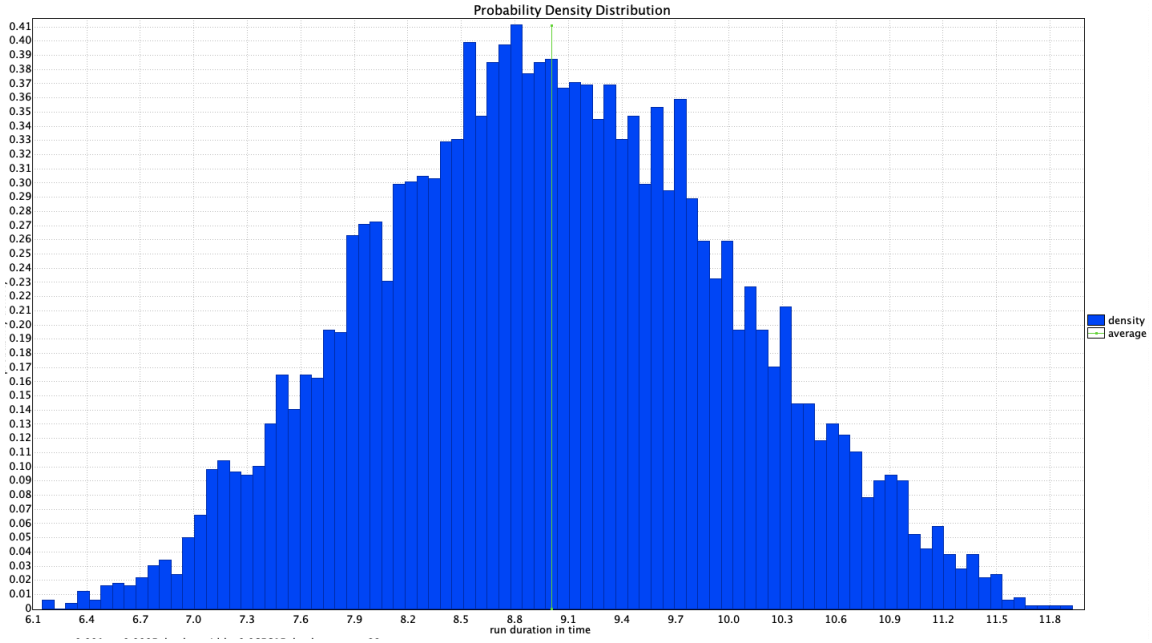
Perėjimai gali turėti apsaugas (angl. guard), sinchronizaciją (angl. synchronisation) ir kintamųjų atnaujinimus. Jei apsaugos išraiška yra teisinga, tai perėjimas yra įgalintas (angl. enabled). Taip pat atskirų modelių perėjimai gali sinchronizuotis kanalų (angl. channels) pagalba. Perėjimo vykdomo metu gali būti atnaujinami sistemos kintamieji. Be to gali būti apibrėžti perėjimų tikimybiniai svoriai (angl. probability weights), kurie nustato perėjimo įvykdymo tikimybę.

Panagrinėkime tris laiko automatus: A_1, A_2 ir A_3, kurie yra pateikti 1 pav.



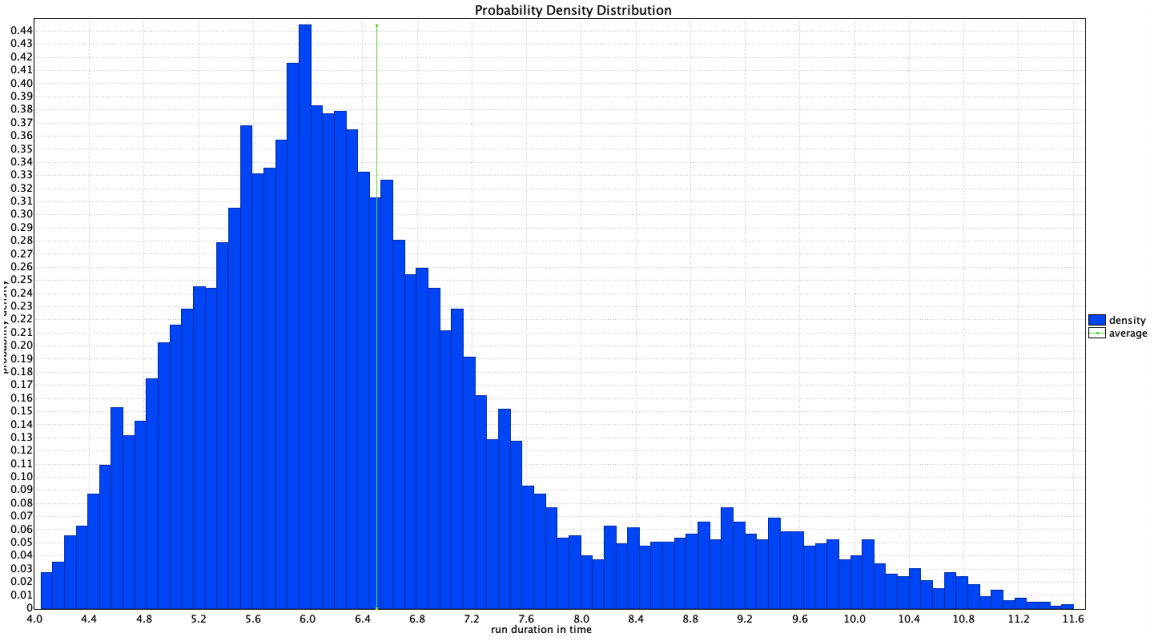
1 pav. Trys stochastiniai laiko automatai

Matome, kad pabaigos lokacija A_1 atveju pasiekiama tarp [6, 12], A_2 atveju pasiekiama tarp [4, 12] ir A_3 atveju pasiekiama tarp [0, $+\infty$] laiko intervaluose. Stochastinė šių trijų laiko automatų interpretacija suteikia tikimybinius pasiskirstymus per pasiekiamumo laiką (angl. reachability time). A_1 atveju visų perėjimų uždelsimas automatiškai nustatomas nepriklausomais ir vienodais pasiskirstymais, kurių reikšmė yra tarp [2, 4]. Taigi bendras pasiekiamumo laikas yra lygus visų perėjimų pasiskirstymų sumai, kurią galime pamatyti 2 pav.



2 pav. A_1 pasiekia *pabaigos* lokaciją

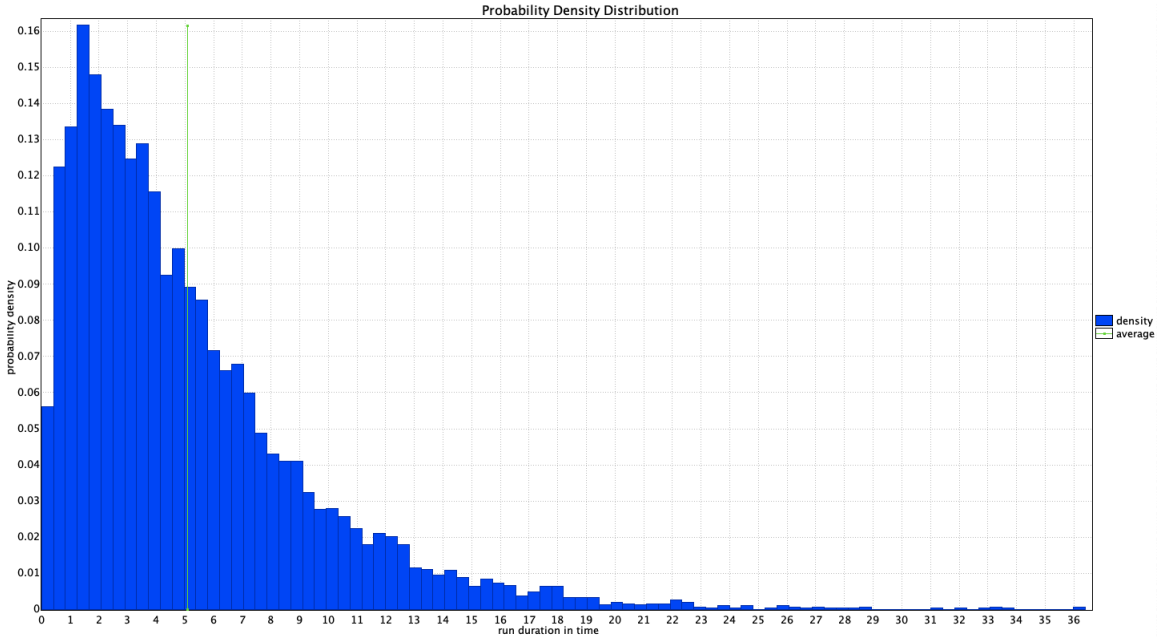
A_2 atveju uždelsimo pasiskirstymai nustatomi pagal kairįjį ir dešinįjį kelią iki pabaigos lokacijos. Panašiai kaip ir A_1 atveju bendras pasiekiamumo laikas yra lygus visų perėjimų pasiskirstymų sumai. Taip pat yra atsižvelgiama į pradžios lokacijos išsišakojimo svorius ($\frac{1}{6}$ ir $\frac{5}{6}$). Bendras A_2 pasiekiamumo laikas pavaizduotas 3 pav.



3 pav. A_2 pasiekia *pabaigos* lokaciją

A_3 atveju uždelsimas nustatomas pagal eksponentinius pasiskirstymus, kuriuos pateikia įrankio naudotojas. Šiuo atveju A_3 eksponentinių pasiskirstymų reikšmės yra $\frac{1}{2}$, $\frac{1}{4}$ ir 2. Taip pat atsižvelgiama į pradžios lokacijos svorius ($\frac{1}{4}$ ir $\frac{3}{4}$). Bendras A_3 pasiekiamumo laikas pavaizduotas 4 pav. Gauti

stochastinės semantikos pasiskirstymai atitinka standartinės semantikos uždelsimo intervalus.

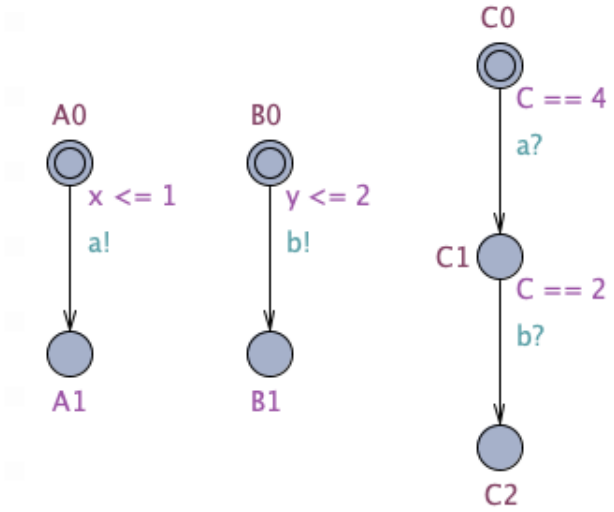


4 pav. A_3 pasiekia *pabaigos* lokaciją

2.2. Tinklai

Uppaal-SMC įrankyje modelis yra sudarytas iš sąveikaujančių stochastinių laiko automatų komponentų tinklo. Šie komponentai yra deterministiški (apibrėžtos paveldėtojų tikimybinės reikšmės). Komponentai perduota pranešimus vienas kitam per transliavimo (angl. broadcast) kanalus ir bendrus kintamuosius, kad galėtų sukurti stochastinių laiko automatų tinklą. Pranešimai perduodami transliavimo sinchronizacijomis neužblokuotiems komponentams, kurie lenktyniauja vienas su kitu su atitinkamais lokaliais pasiskirstymais.

5 pav. pavaizduotas stochastinis laiko automatų tinklas sudarytą iš trijų lygiagrečių komponentų: A, B ir C.

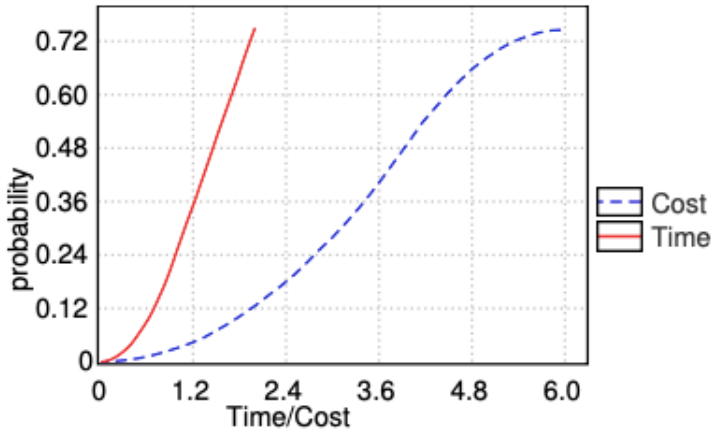


5 pav. Stochastinis laiko automatų tinklas

Nesunkiai galime pamatyti, kad sudėtinė sistema (A|B|C) turi tokią perėjimų seką:

$$((A_0, B_0, C_0)[x = 0, y = 0, C = 0]) \xrightarrow{1} \xrightarrow{a!}$$
$$((A_1, B_0, C_1)[x = 1, y = 1, C = 4]) \xrightarrow{1} \xrightarrow{b!}$$
$$((A_1, B_1, C_2)[x = 2, y = 2, C = 6])$$

Ši seka rodo, kad C pabaigos lokacija C_2 yra pasiekiama, kur sanaudos (angl. cost) ir laikas priklauso nuo to kada ir kokia seka A ir B modeliai atliks sinchronizacinius veiksmus a! ir b!. C_2 lokacijos pasiekiamumo sąnaudų ir laiko pasiskirstymą galime pamatyti 6 pav.

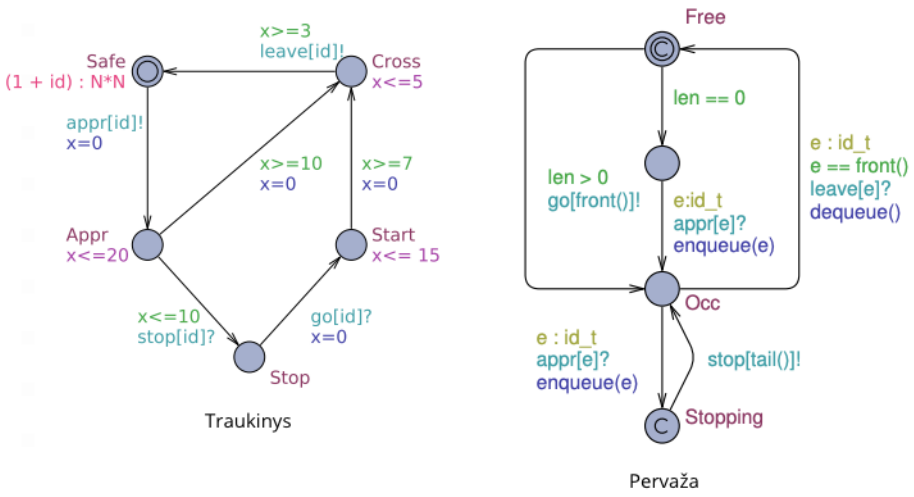


6 pav. C_2 lokacijos pasiekiamumo sąnaudų ir laiko pasiskirstymas

2.3. Traukinių pervažos pavyzdys

Reikia paminėti, kad Uppaal-SMC palaiko visas klasikinio Uppaal įrankio funkcijas: sveikųjų skaičių kintamuosius, duomenų struktūras ir naudotojo apibrėžtas funkcijas. Taip pat Uppaal-SMC leidžia apibrėžti lokacijų laikrodžių dažnius ir perėjimų išsišakojimų svorius.

Šiame darbe bus pristatytas traukinių pervažos pavyzdys [YPD95b], kuriam bus pritaikytas Uppaal-SMC funkcionalumas. 7 pav. galime pamatyti traukinio ir pervažos modelius.



7 pav. Traukinio ir pervažos modeliai

Tam tikras skaičius traukinių artėja prie tilto, kur vienu metu gali važiuoti tik vienas traukinys. Pervažos modelis yra atsakingas, kad nebūtų susidūrimų tarp traukinių. Šis modelis siunčia signalą, kada traukinys gali važiuoti per tiltą. Traukiniams nustatyti laiko apribojimai, kad nebūtų įmanoma jų sustabdyti akimirksniu. *Safe* lokacija neturi invariantų; jos uždelsimas apibrėžtas eksponentinių dažnių pasiskirstymu. Traukiniai uždelsia pagal šį pasiskirstymą ir tuomet keliauja prie pervažos, t.y., vykdo *appr[i]* sinchronizaciją. *Safe* lokacijas eksponentinio dažnio reikšmė yra $\frac{1+id}{N^2}$, kur *id* yra traukinio identifikatorius, o N - traukinių skaičius. Traukiniai su didesniu *id* skaičiumi atvyksta greičiau. Perėjimai iš lokacijų su invariantais nustatomi pagal suvienodintus pasiskirstymus per laiką. Tokios lokacijos yra *Appr*, *Cross* ir *Start*. Pavyzdžiui, pereiti iš *Cross* lokacijos užtruks tarp 3 ir 5 laiko vienetų. Pervažos modelis saugo traukinių informaciją vidinėje eilės (angl. queue) struktūroje. Šis modelis naudoja statymo į eilę funkciją, kai traukinys keliauja į *Occ* lokaciją ir išėmimo iš eilės funkciją, kai traukinys atsilaisvina.

2.4. Užklausos

Uppaal-SMC gali apdoroti klasikines Uppaal įrankio užklausas: pasiekiamumą, invariantinių savybių išsaugojimą ir neišvengiamumą (angl. inevitability). Taip pat šis įrankis gali apdoroti užklausas, kurios yra susiję su stochastine laiko automatų interpretacija. Uppaal-SMC leidžia naudotojui simuliuoti ir vizualizuoti išraiškų reikšmes. Simuliacijos užklausos sintaksė Uppaal-SMC

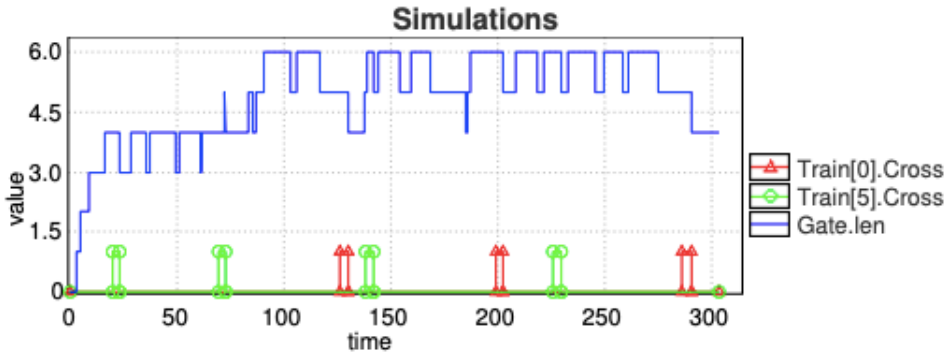
įrankyje:

$$simulate\ N\ [<= bound]\ \{E_1, .., E_k\},$$

kur *N* yra atliktų simuliacijų skaičius, *bound* yra laiko apribojimas ir E_1, ..., E_k yra *k* būsenomis paremtos išraiškos, kurios bus stebimos (angl. monitored) ir vizualizuojamos. Panaudosime šią užklausą su prieš tai pateiktu traukinių pervažos pavyzdžiu. Stebėsime kada 1-as ir 5-as traukiniai važiuoja per tiltą bei pervažos eilės ilgį. Vykdoma užklausa:

$$simulate\ 1\ [<= 300]\ \{Train(0).Cross, Train(5).Cross, Gate.len\}$$

Ši užklausa sukuria grafiką, kuris yra pateiktas 8 pav.



8 pav. Traukinių ir pervažos vizualizacija

Matome, kad 5-asis traukinys važiuoja per tiltą dažniau negu 1-asis. Nenaudojant simuliacijos būtų sunku numatyti, kad eilė po maždaug 20 laiko vienetų visada bus 3 arba didesnio ilgio. Tuo galime įsitikinti įvykdę *Pr [<=300] (<> Gate.len < 3 and t > 20)* užklausą ir pridėję laikrodį *t*. Gauname atsakymą, kur tikimybė yra tarp [0.102, 0.123] [DLL+15].

3. Praktinė dalis

Šiame darbe bus analizuojama ir tyrinėjama BepiColombo DAB sistema Uppaal-SMC įrankio pagalba. BepiColombo – tai bendra misija tarp Europos Kosmoso Agentūros (angl. European Space Agency, ESA) ir Japonijos Kosmoso Tyrimų Agentūros (angl. Japan Aerospace Exploration Agenc, JAXA) į Merkurijaus planetą. Ankstesnės ESA tarpplanetinės misijos buvo siunčiamos į gana šaltas Saulės sistemos dalis. BepiColombo yra išskirtinė misija, nes Merkurijaus orbita yra arti Saulės. Šios žvaigždės gravitacija kelia iššūkį erdvėlaivio pastatymui į stabilią Merkurijaus orbitą.

Iki šiol Merkurijų pasiekė tik NASA Mariner 10 ir Messenger misijų erdvėlaiviai. Mariner 10 1974-1975 m. pateikė pirmuosius iš arti nufotografuotus planetos vaizdus. Messenger 2008 m. skrido pro Merkurijaus planetą tris kartus rinkdamas naujus duomenis ir darydamas nuotraukas.

BepiColombo misiją sudaro erdvėlaivis, kuris susideda iš perkėlimo modulio ir dviejų orbiterių:

- Merkurijaus Perkėlimo Modulis (angl. Mercury Transfer Module);
- Merkurijaus Planetinis Orbiteris (angl. Mercury Planetary Orbiter);
- Merkurijaus Magnetosferinis Orbiteris (angl. Mercury Magnetospheric Orbiter).

Kelionės metu orbiteriai ir perkėlimo modulis, kuris yra sudarytas iš elektrinės varomosios jėgos (angl. electric propulsion) ir tradicinių cheminių raketų bus sujungti į vieną sudėtinį erdvėlaivį.

Misijos įvyko 2018 metų spalio 20 dieną. 2025 metų gale yra planuojama, kad BepiColombo nusileis Merkurijuje, kur vienerius metus rinks ir studijuos duomenis apie planetos struktūrą, geofiziką, atmosferą, magnetosferą ir jos istoriją. Taip pat BepiColombo sudarys pilną Merkurijaus planetos žemėlapį.

BepiColombo DAB – tai vienas iš Merkurijaus Planetinio Orbiterio įrenginių. Jis susideda iš pagrindinės programinės įrangos ir dviejų mokslinių instrumentų. BepiColombo DAB gauna TC iš BepiColombo erdvėlaivio ir perduoda jam mokslinius ir diagnostikos TM. Pagrindinė programinė įranga gautas TC saugo tam skirtame buferyje. Taip ši įranga yra atsakinga už kiekvienos TC sintaksinį ir semantinį patikrinimą (angl. validation). Jeigu TC patikrinimas yra nesėkmingas, tuomet yra sugeneruojami atitinkami TM. TC gali prašyti programinės įrangos pakeisti komponento veikimo režimą, aktyvuoti arba deaktyvuoti mokslinių duomenų generavimą, kurti diagnostikos ataskaitas ir pan. DAB programinė įranga įranga dekoduoja po vieną TC vienu metu ir perduoda ją atitinkamam instrumentui. Tuomet instrumentas gali atlikti tam tikrus veiksmus ir grąžinti patvirtintus (angl. acknowledged) TM. Visi TM yra saugomi atitinkamame buferyje [IRL+12].

3.1. BepiColombo DAB Uppaal sistemoje

Šiame skyriuje bus pristatyta BepiColombo DAB sistema ir jos realizacija klasikiniame Uppaal įrankyje. Sistemą sudaro šie modeliai: TC apdorojimo sistema, TC ir TM buferiai, gavėjas,

siuntėjas, instrumentas, diagnostikos duomenų gamintojas bei paprastas ir tingus stebėtojai.

9 pav. matome Uppaal deklaracijų skiltį, kurioje yra aprašyti sistemos kintamieji ir funkcijos.

```
const int Buffer_size = 2;
const int HK_period = 2;
const int Inst_maxtime = 7;
const int Rec_maxtime = 3;
const int Valid_maxtime = 2;

int[0, Buffer_size*3+1] tc_buffer [Buffer_size] = {0, 0};
int[0, Buffer_size*3+1] tm_buffer [Buffer_size] = {0, 0};
int[0, Buffer_size-1] first_tc_index = 0;
int[0, Buffer_size-1] first_tm_index = 0;
int[0, Buffer_size-1] next_tc_index = 0;
int[0, Buffer_size-1] next_tm_index = 0;
int[0, Buffer_size*3+1] new_tc =  0;
int[0, Buffer_size*3+1] new_tm = 0;
int[0, Buffer_size*3+1] last_tm = 0;
int[1, Buffer_size*3+1] last_id = Buffer_size*3+1;

int[1, Buffer_size*3+1] unique_id() {
    if (last_id == Buffer_size*3+1) {
        last_id = 1;
    } else {
        last_id = (last_id+1) % (Buffer_size*3+2);
    };

    return last_id;
}

int[1, Buffer_size*3+1] unique_id2() {
    if (last_id == Buffer_size*3+1) {
        last_id = 2;
    } else {
        last_id = (last_id+1) % (Buffer_size*3+2);
    };

    return last_id;
}
chan newtc; // new tc appeared
chan newtm; // new tm produced
urgent chan readtc; // tc is read from the buffer
urgent chan readtm; // tm is read from the buffer
urgent chan request; // request to instrument
urgent chan start;
urgent chan stop;
chan message; // resulting message from instrument
chan tmready;
```

9 pav. Deklaracijos

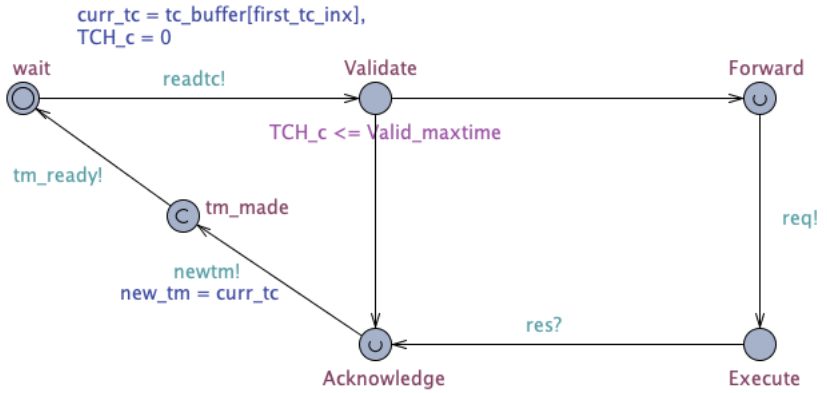
Šioje skiltyje yra apibrėžtos modeliuose naudojamos konstantos. Pavyzdžiui, buferių dydžiai ir diagnostikos TM generavimo dažnis. Taip pat yra apibrėžiami buferių masyvai, TC ir TM kintamieji. Ši sistema naudoja dvi funkcijas, kurios skirtos ”šviežių” buferio elementų identifikacijos numerių sukūrimui. Be to čia yra apibrėžiami sistemoje naudojami kanalai. Šiuos visus parametrus galime keisti ir stebėti, kaip keičiasi sistemos elgesys.

10 pav. matome bendrų sistemos deklaracijų skiltį, kurioje yra sukuriami visi reikalingi sistemos procesai.

```
TC_buffer = Buffer(newtc, readtc, tc_buffer, first_tc_index, next_tc_index, new_tc);
TM_buffer = Buffer(newtm, readtm, tm_buffer, first_tm_index, next_tm_index, new_tm);

Observer1 = Observer(start, stop);
Observer2 = Idle_observer(start, stop);

Sender = Plat_sender(start);
Receiver = Plat_receiver(stop);

system TC_buffer, TM_buffer, Observer1, Observer2, Sender, Receiver, TC_handling, Instrument, HK_producer;
```

10 pav. Sistemos deklaracijos

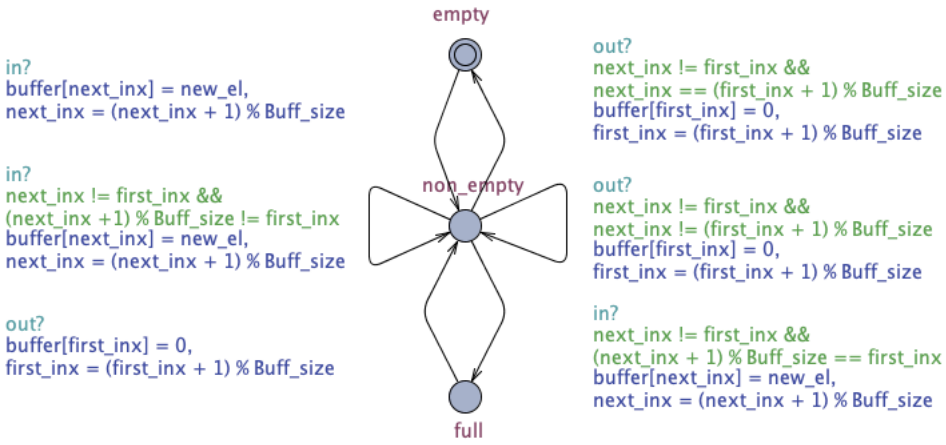
11 pav. matome TC valdymo modelį (komponentės šabloną pagal Uppaal-SMC terminologiją).



11 pav. TC valdymo modelis

Šis modelis yra atsakingas už TC apdorojimą, kurias gauna iš BepiColombo erdvėlaivio. Darbas pradedamas laukimo būsenoje, t.y., laukiama, kol buferyje atsiras bent viena TC. Atsiradus TC ji yra perskaitoma ir validuojama. Jei TC yra validi, tai ji perduodama instrumento modeliui, kuris ją apdoroja ir grąžina atgal. Kitu atveju yra praleidžiamas TC perdavimo instrumentui etapas. Paskutiniame etape TC valdymo modelis sukuria naujus TM, kurie yra patalpinami į atitinkamą buferį. Galiausiai yra siunčiamas pranešimas gavėjo modeliui apie naujus TM. Šis modelis perskaito TM ir perduoda juos stebėtojo modeliui.

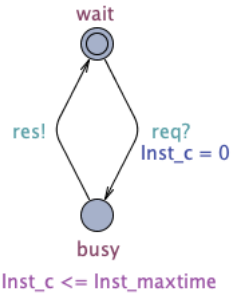
12 pav. matome buferio modelį.



12 pav. Buferio modelis

Buferio modelis naudojamas TC bei TM buferių procesams sukurti. Šis modelis įgyvendina žiedinį (angl. circular) buferį. Buferis turi tris būsenas: *empty* (liet. tuščias), *not_empty* (liet. nepilnas) ir *full* (liet. pilnas). Šiame modelyje elementai yra saugomi pagal indekso numerį. Buferis gali gauti įvedimo (angl. in) ir išvedimo (angl. out) pranešimus, kurie atitinkamai patalpina arba pašalina elementą iš buferio.

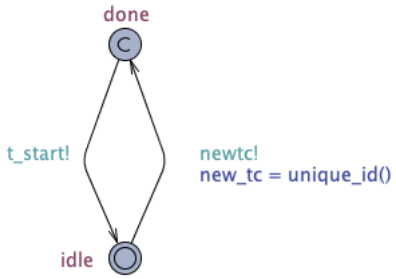
13 pav. matome instrumento modelį.



13 pav. Instrumento modelis

Tai yra gana paprastas modelis, kuris yra atsakingas už TC priėmimą ir perdavimą TC valdymo modeliui.

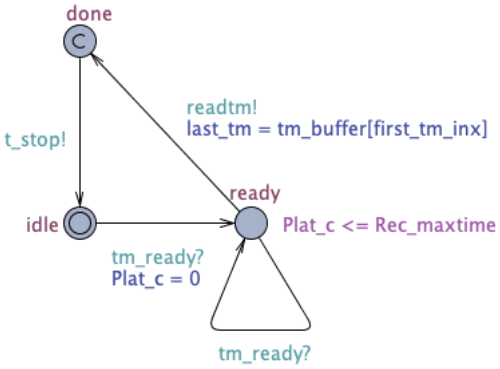
14 pav. matome siuntėjo modelį.



14 pav. Siuntėjo modelis

Siuntėjo modelis praneša apie naują TC ir priskiria jai unikalų numerį bei siunčia pranešimą stebėtojų modeliams.

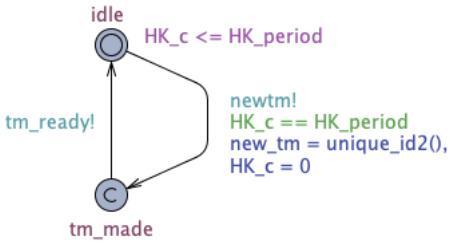
15 pav. matome gavėjo modelį.



15 pav. Gavėjo modelis

Gavėjo modelis laukia ir praneša, kad perskaitė naujus TM. Perskaitęs TM, jis siunčia pranešimą stebėtojui.

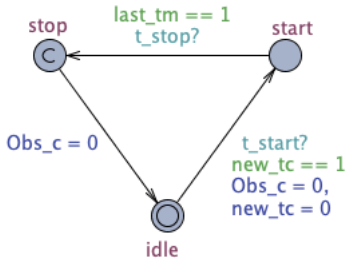
16 pav. matome diagnostikos gamintojo modelį.



16 pav. Diagnostikos gamintojo modelis

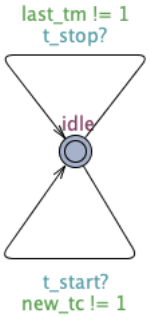
Diagnostikos gamintojo modelis sukuria naujus TM ir praneša apie tai gavėjui.

17 pav. matome stebėtojo modelį.



17 pav. Stebėtojo modelis

18 pav. matome tingaus (angl. idle) stebėtojo modelį.



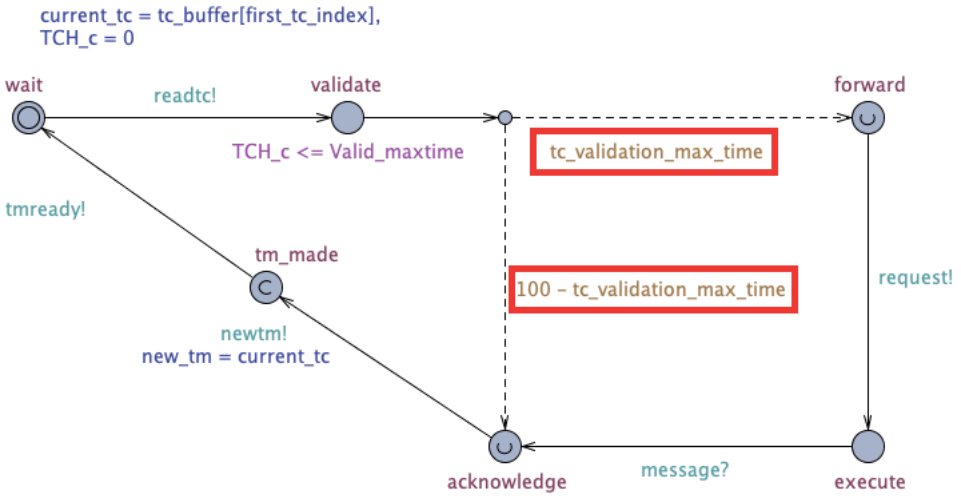
18 pav. Tingaus stebėtojo modelis

Stebėtojas ir tingus stebėtojo modeliai laukia pranešimų iš siuntėjo ir gavėjo.

3.2. BepiColombo DAB Uppaal-SMC sistemoje

Šiame skyriuje BepiColombo DAB sistema bus pritaikyta Uppaal-SMC įrankiui. Toliau bus apžvelgta priimti sistemos pakeitimai. Šie pakeitimai suteikia didesnį sistemos konfigūracijos lankstumą ir leidžia modeliams padengti daugiau įvairių situacijų.

TC valdymo modelyje pridėti tikimybinių svorių (angl. probability weights) kintamieji iš lokacijos *validate*. *tc_validate_max_time* kintamajam priskirta reikšmė 90. Todėl *forward* lokacija pasiekiama su 90% tikimybe, o *acknowledge* lokacija pasiekiama su 10% tikimybe. Pakeitimus matome 19 pav.

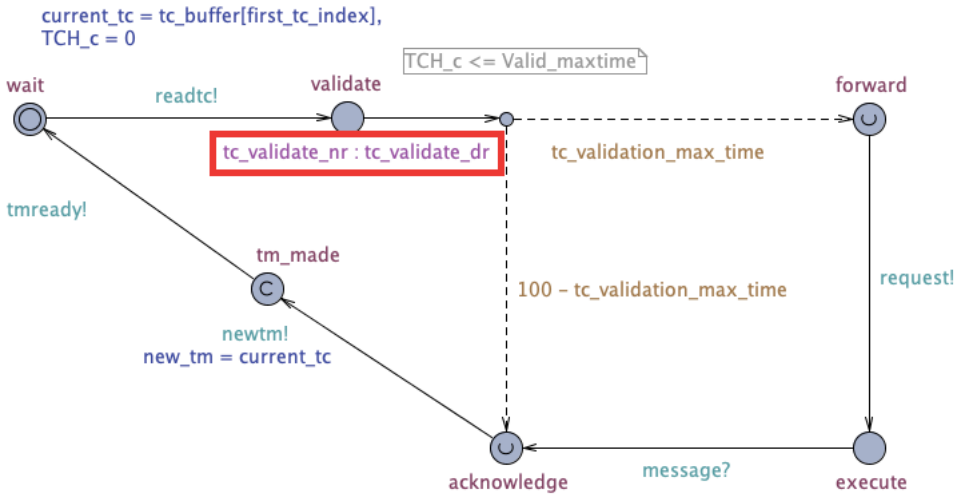


19 pav. TC valdymo tikimybiniai svoriai

TC valdymo modelyje pridėti *validate* lokacijos eksponentinio dažnio (angl. rate of exponential) kintamieji, kur:

$$const\ int\ tc\_validate\_nr\ =\ 2;$$
$$const\ int\ tc\_validate\_dr\ =\ 1;$$

Čia nr (angl. numerator) atitinka skaitiklį, o dr (angl. denumerator) - vardiklį. Kuo mažesnis dažnis, tuo ilgesnis uždelsimas. Pridėjus šį dažnį buvo pašalintas šios lokacijos invariantas: *TCH_c <= Valid_maxtime*. Pakeitimai yra atvaizduoti 20 pav.

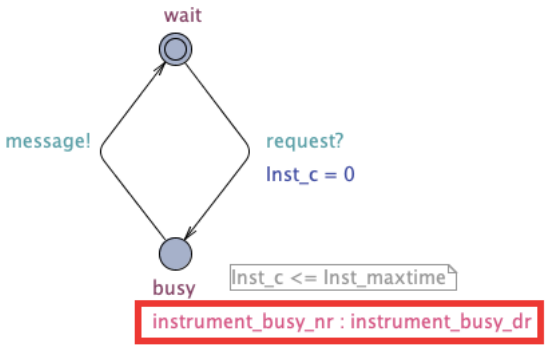


20 pav. TC valdymo *validate* lokacijos eksponentiniai dažniai

Pridėti instrumento *busy* lokacijos eksponentinio dažnio kintamieji, kur:

$$const\ int\ instrument\_busy\_nr\ =\ 2;$$
$$const\ int\ instrument\_busy\_dr\ =\ 3;$$

Pridėjus šį dažnį buvo pašalintas lokacijos invariantas: *Inst_c <= Inst_maxtime*. Pakeitimai yra atvaizduoti 21 pav.

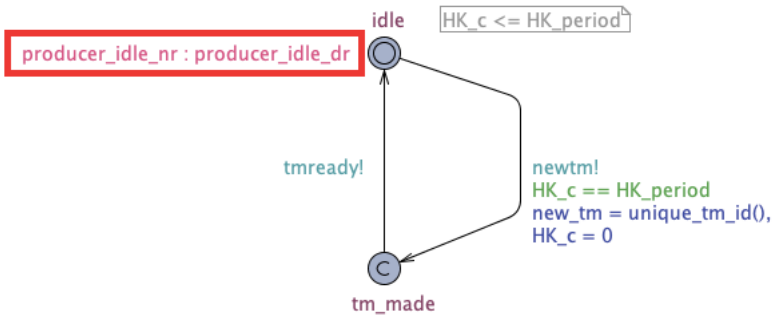


21 pav. Instrumento *busy* lokacijos eksponentiniai dažniai

Pridėti diagnostikos gamintojo *idle* lokacijos eksponentinio dažnio kintamieji, kur:

$$const\ int\ producer\_idle\_nr\ =\ 2;$$
$$const\ int\ producer\_idle\_dr\ =\ 1;$$

Pridėjus šį dažnį buvo pašalintas šios lokacijos invariantas: *HK_c <= HK_period*. Pakeitimai yra atvaizduoti 22 pav.

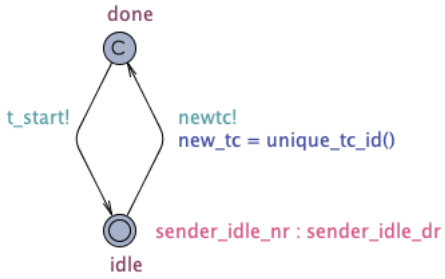


22 pav. Diagnostikos gamintojo *idle* lokacijos eksponentiniai dažniai

Pridėti siuntėjo *idle* lokacijos eksponentinio dažnio kintamieji, kur:

$$const\ int\ sender\_idle\_nr\ =\ 4;$$
$$const\ int\ sender\_idle\_dr\ =\ 1;$$

Nuo šiol nauji TC pranešimai turės uždelsimą. Pakeitimai yra atvaizduoti 23 pav.

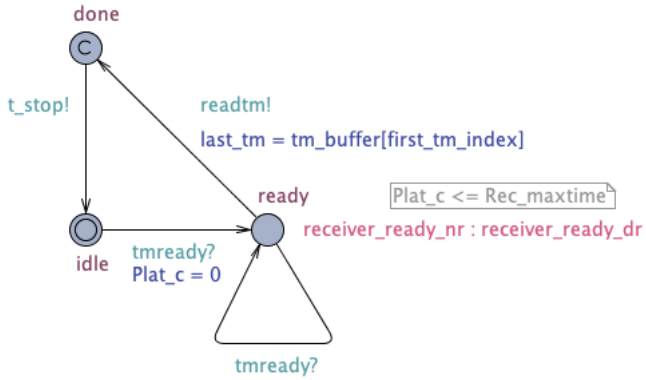


23 pav. Siuntėjo *idle* lokacijos eksponentiniai dažniai

Pridėti gavėjo *ready* lokacijos eksponentinio dažnio kintamieji, kur:

$$const\ int\ receiver\_ready\_nr\ =\ 1;$$
$$const\ int\ receiver\_ready\_dr\ =\ 1;$$

Pridėjus šį dažnį buvo pašalintas lokacijos invariantas: *Plat_c* <= *Rec_maxtime*. Pakeitimai yra atvaizduoti 24 pav.



24 pav. Gavėjo *ready* lokacijos eksponentiniai dažniai

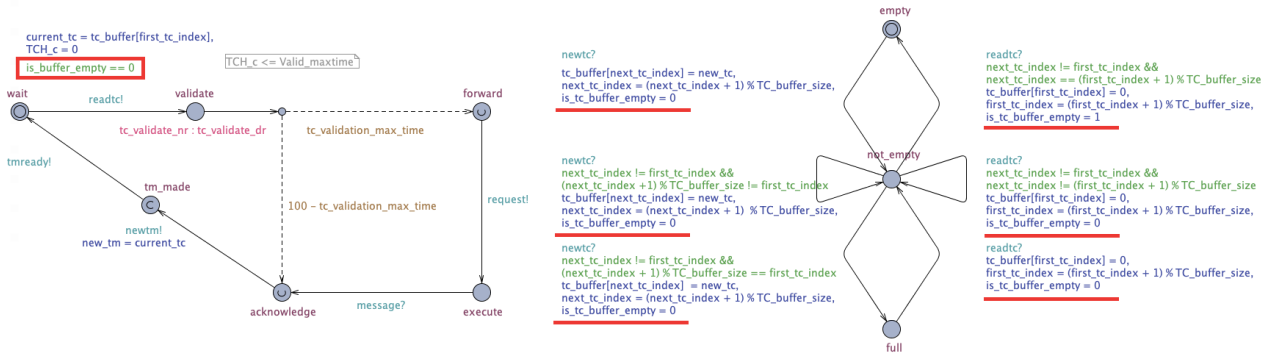
Kad galima būtų generuoti grafikus reikėjo visus kanalus (angl. channels) pakeisti į transliuojamus (angl. broadcast) kanalus. Pakeitimai yra atvaizduoti 25 pav.

```
broadcast chan newtc; // new tc appeared
broadcast chan newtm; // new tm produced
urgent broadcast chan readtc; // tc is read from the buffer
urgent broadcast chan readtm; // tm is read from the buffer
urgent broadcast chan request; // request to instrument
urgent broadcast chan start;
urgent broadcast chan stop;
broadcast chan message; // resulting message from instrument
broadcast chan tmready;
```

25 pav. Transliuojami kanalai

Tyrinėjant sistemą Uppaal-SMC įrankyje buvo pastebėta, kad sistema pradėjo neteisingai veikti, kai buvo pridėti transliavimo kanalai. Problema, kad TC_handling modelis gali pereiti iš *wait* į *validate* būseną, kai TC buferyje nėra nei vienos TC. Einant iš *wait* būsenos atliekama *readtc!* sinchronizacija. Iki šiol ši sinchronizacija pati užtikrindavo, kad bus skaitoma iš netuščio TC buferio. Tačiau transliavimo kanalai visada sinchronizuojasi nepriklausomai, ar kas nors klausosi jo ar ne, todėl *wait* į *validate* perėjime reikia pridėti papildomą apsaugą.

Deklaracijų skiltyje buvo sukurti nauji globalūs kintamieji: *is_tc_buffer_empty* ir *is_tm_buffer_empty*, kurie suteikia informaciją, ar buferis yra tuščias. Šie kintamieji atnaujinami, kai atliekami veiksmai su buferiais. Atlikti pakeitimai yra atvaizduoti 26 pav.



26 pav. *is_tc_buffer_empty* kintamasis

Pirmoje modelio versijoje sistema naudoja bendrą kintamąjį TC ir TM buferių dydžiams nustatyti. Pirmiausia reikia pašalinti buferių priklausomybę nuo vieno bendro kintamojo, kad galėtume keisti tik TM buferio dydį. Pakeitimai yra atvaizduoti 27 pav.

```
const int TC_buffer_size = 2;
const int TM_buffer_size = 2;
const int HK_period = 2;
//const int Inst_maxtime = 7;
//const int Rec_maxtime = 3;
//const int Valid_maxtime = 2;

int[0, TC_buffer_size*3+1] tc_buffer [TC_buffer_size] = {0, 0};
int[0, TM_buffer_size*3+1] tm_buffer [TM_buffer_size] = {0, 0};
int[0, TC_buffer_size-1] first_tc_index = 0;
int[0, TM_buffer_size-1] first_tm_index = 0;
int[0, TC_buffer_size-1] next_tc_index = 0;
int[0, TM_buffer_size-1] next_tm_index = 0;
int[0, TC_buffer_size*3+1] new_tc =  0;
int[0, TM_buffer_size*3+1] new_tm = 0;
int[0, TM_buffer_size*3+1] last_tm = 0;
int[1, TC_buffer_size*3+1] last_tc_id = TC_buffer_size*3+1;
int[1, TM_buffer_size*3+1] last_tm_id = TM_buffer_size*3+1;

int[1, TC_buffer_size*3+1] unique_tc_id() {
    if (last_tc_id == TC_buffer_size*3+1) {
        last_tc_id = 1;
    } else {
        last_tc_id = (last_tc_id+1) % (TC_buffer_size*3+2);
    };

    return last_tc_id;
}

int[1, TM_buffer_size*3+1] unique_tm_id() {
    if (last_tm_id == TM_buffer_size*3+1) {
        last_tm_id = 2;
    } else {
        last_tm_id = (last_tm_id+1) % (TM_buffer_size*3+2);
    };

    return last_tm_id;
}
```

Pridėti atskiri buferių dydžių kintamieji

Atnaujinti esami kintamieji pagal naujus TC_buffer_size ir TM_buffer_size kintamuosius. Taip pat vietoj last_id pridėti last_tc_id ir last_tm_id

unique_id() pakeista į unique_tc_id() ir atnaujinta atsižvelgiant į naujus TC kintamuosius

unique_id2() pakeista į unique_tm_id() ir atnaujinta atsižvelgiant į naujus TM kintamuosius

27 pav. Buferių kintamųjų pakeitimai

TC ir TM buferiai naudoja bendrą Buffer modelį. Šio modelio parametrų reikšmėse naudojamas *Buffer_size* kintamasis. Pavyzdžiui, vienas iš parametrų yra *int[0, Buffer_size*3+1] &buffer[Buffer_size]*. Kadangi yra atskirtas *Buffer_size* į *TC_buffer_size* ir *TM_buffer_size*, tai šis modelis nebegali būti panaudotas ir TC ir TM buferiams. Dėl šios priežasties tenka išskaidyti Buffer modelį į TC_buffer ir TM_buffer modelius, kurie iš esmės skiriasi tik kintamųjų reikšmėmis.

Taip pat buvo pakeisti statistinio įrankio nustatymai, kad būtų generuojami grafikai su didesniu bandymų skaičiumi ir tuo pačiu rezultatų tikslumu. Tuo tikslu buvo atlikti sistemos parametrų pakeitimai:

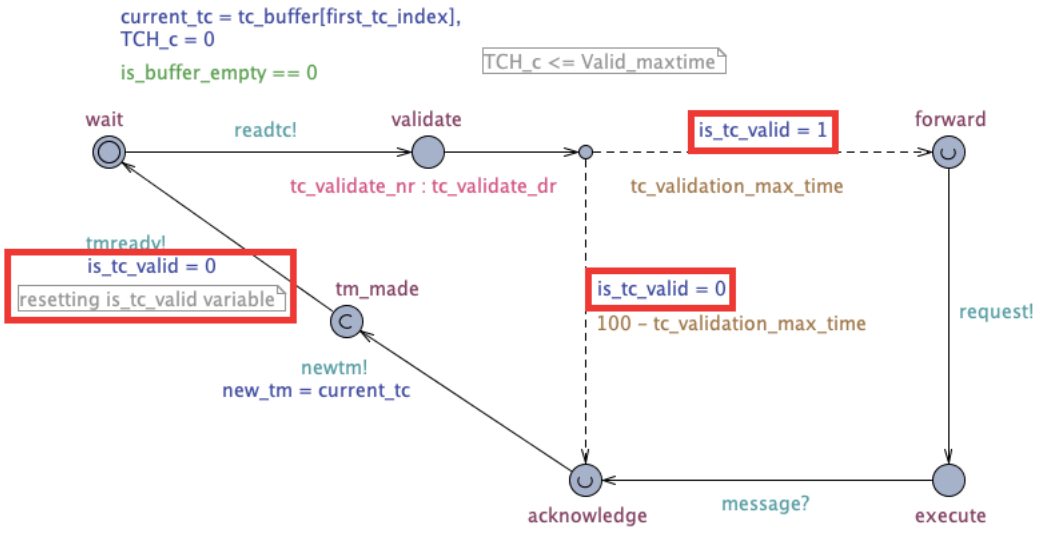
- probability of false negatives: 0.05 → 0.001
- probability of false positives: 0.05 → 0.001
- probability uncertainty: 0.05 → 0.0005

Be to reikėjo pridėti papildomą kintamąjį, kuris atskirtų sėkmingus ir nesėkmingus TC validavimus. Tam buvo įvestas naujas kintamasis:

$$int[0,1]\ is\_tc\_valid = 0;$$

,

kuris nurodo, ar TC validacija yra sėkminga. Su šiuo nauju kintamuoju galime tirti sėkmingai arba nesėkmingai validuotas TC. Atlikti pakeitimai yra atvaizduoti 28 pav.



28 pav. *is_tc_valid* kintamasis

3.3. Sistemos efektyvumo analizė

Šiame skyriuje bus pateikta BepiColombo DAB sistemos efektyvumo analizė. Skirtingos sistemos savybės bus analizuojamos pateikus skirtingas užklausas Uppaal-SMC verifikavimo posistemei.

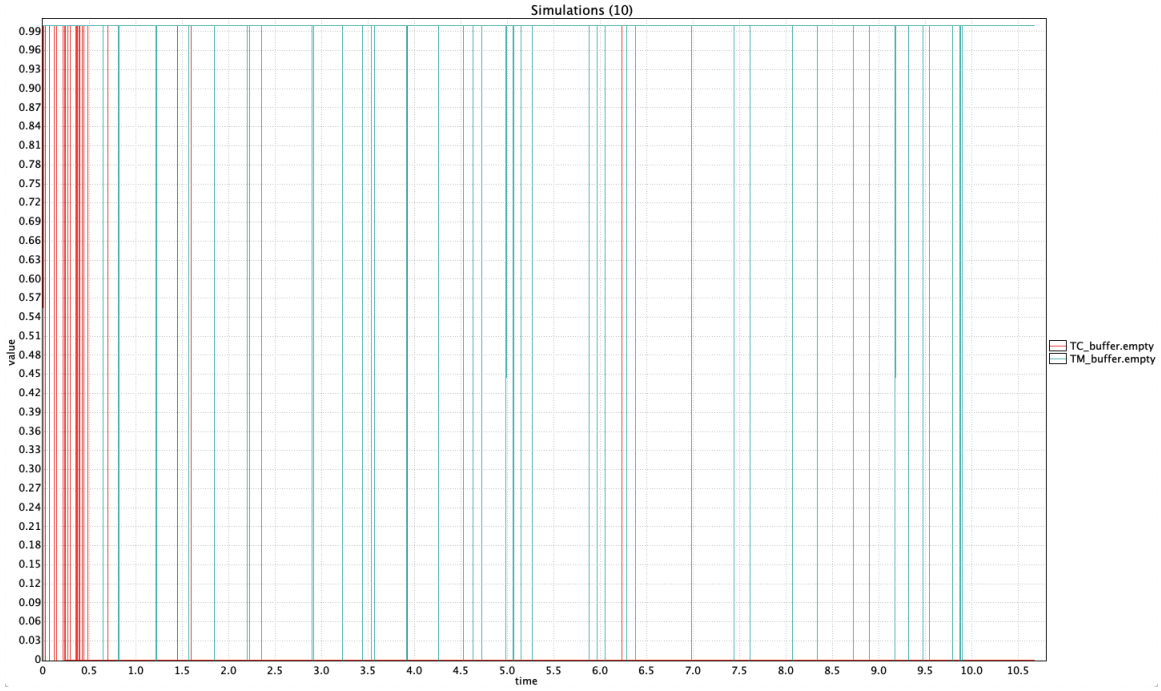
3.3.1. TC ir TM buferiai

Pirmiausia panagrinėsime, kaip funkcionuoja sistemos buferiai, pavyzdžiui, kaip dažnai jie yra savo skirtingose būsenose. Panaudosime *simulate* funkciją, kuri vykdo sistemos simuliavimą. Šios funkcijos pagalba galime vizualizuoti sistemos modelių kintamųjų ir laikrodžių reikšmes.

Pirmoji užklausa:
simulate 10 [<= 10] { TC_buffer.empty, TM_buffer.empty }, kur:

- $N = 10$ - atliktų simuliacijų skaičius;
- *bound* $= 10$ - simuliacijos laikas;
- *TC_buffer.empty, TM_buffer.empty* - buferio lokacijos, kurios yra simuliuojamos.

Užklausos atlikimo rezultatai yra matomi 29 pav.



29 pav. *simulate 10 [<= 10] { TC_buffer.empty, TM_buffer.empty }*

Matome, kad TC buferis įsibegėjus sistemos darbui beveik niekada nebūna tuščias, o TM buferis beveik reguliariais intervalais yra tuščias.

Sekanti užklausa:

simulate 10 [<= 10] { TM_buffer.not_empty, TM_buffer.full }

Užklausos atlikimo rezultatai yra matomi 30 pav.

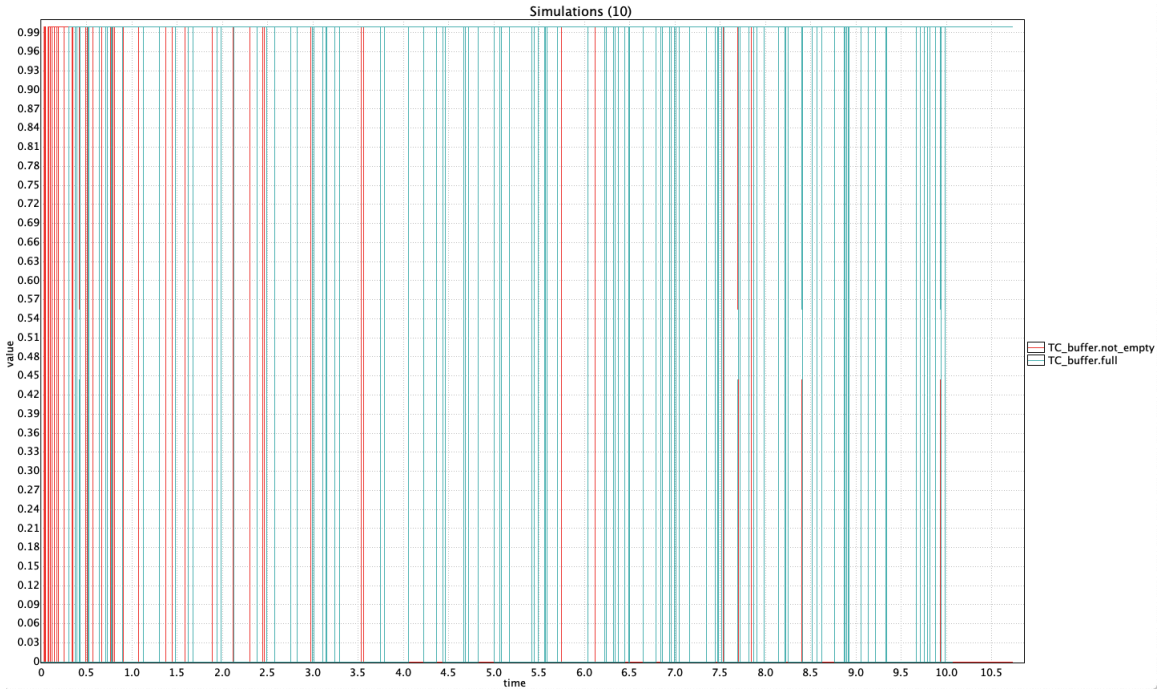
30 pav. *simulate 10 [<= 10] { TM_buffer.not_empty, TM_buffer.full }*

Matome, kad TM buferis niekada nebūna pilnas.

Paskutinė *simulate* užklausa:

simulate 10 [<= 10] { TC_buffer.not_empty, TC_buffer.full }

Užklausos atlikimo rezultatai yra matomi 31 pav.



31 pav. *simulate 10 [<= 10] { TC_buffer.not_empty, TC_buffer.full }*

Matome priešingą situaciją su TC buferiu. Dažnai jis būna pilnas.

Toliau panaudosime tikimybių palyginimo funkciją *Pr*. Vykdome:

Pr [<= 1000] (<> TC_buffer.full) >= Pr [<= 1000] (<> TC_buffer.not_empty)

Ši funkcija palygina dvi tikimybes: tikimybė, kad buferis yra pilnas ir ne tuščias. Įvykdžius šią užklausą pamatysime, ar per 1000 laiko vienetų TC buferius dažniau būna pilnas negu ne tuščias.

Gauname atsakymą, kad tikrinama savybė yra tenkinama su 90% užtikrintumu. Vadinasi, TC buferis dažniau būna pilnas negu ne tuščias.

3.3.2. Atskiros užklauso apdorojimas

Tolesnius eksperimentus darysime su prielaida, kad sėkmingos TC validacijos tikimybė yra 90% (šią prielaidą galima visada pakeisti keičiant atitinkamą sistemos parametrą).

Atskiros užklausos id atitinka *current_tc = tc_buffer[first_tc_index]* reikšmę, t.y., užklausos id yra einamojo TC buferio elemento reikšmė. *current_tc* yra apibrėžtas taip:

$$int[0, Buffer\_size * 3 + 1]\ current\_tc;$$

current_tc gali įgyti reikšmes nuo 0 iki 7. Visos id reikšmės turėtų būti vienareikšmiškos užklausos vykdymo atžvilgiu. Dėl šios priežasties atsitiktiniu būdu galime išsirinkti užklausos id

reikšmę ir įterpti ją į tikrinimo užklausas. Tarkime, seksime užklausą, kurios id yra 7. Reikia paminėti, kad *current_tc* reikšmė yra priskiriama *new_tm* reikšmei, kai TC_handling pasiekia *tm_made* būseną, t.y., sukuriamas grąžinamas TM pranešimas.

Vykdome:

Pr [<= 100] (<> TC_handling.tm_made && (TC_handling.current_tc < 0 || TC_handling.current_tc > 7))

Gauname, kad tikimybė, kad per 100 laiko vienetų bus pagaminta TC, kurios id < 0 arba id > 7 yra [0,0.000999882]. Taigi, galime įsitikinti, kad užklausos id tikrai yra nuo 0 iki 7.

Vykdome:

Pr [<= 100] (<> TC_handling.tm_made && TC_handling.current_tc == 7)

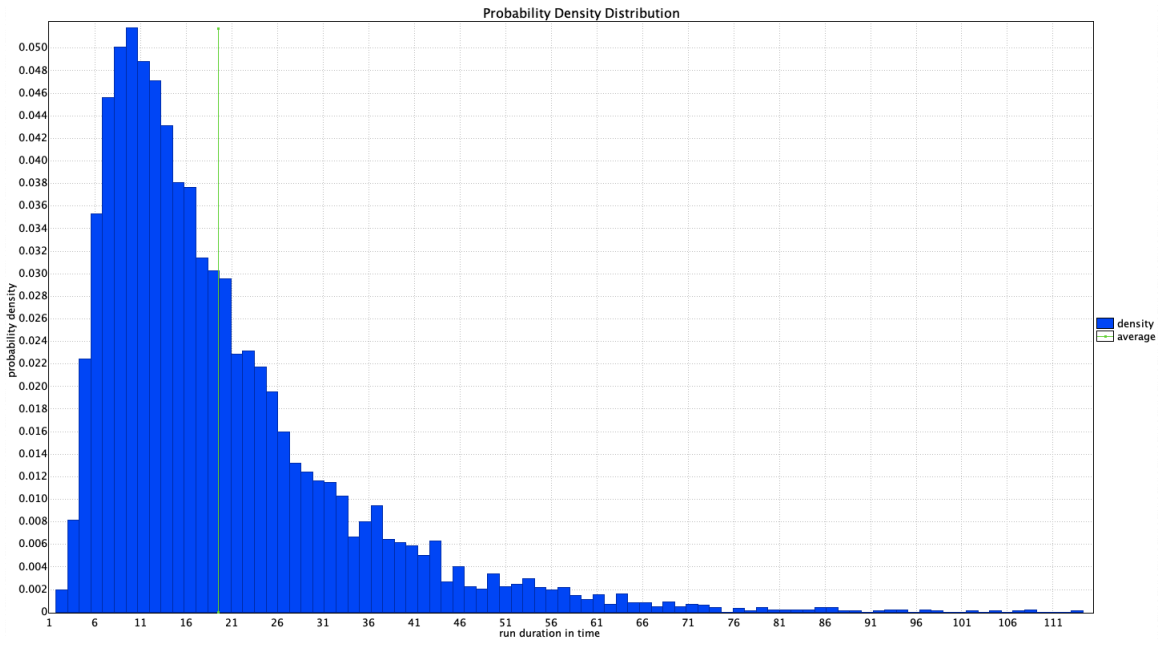
Gauname, kad tikimybė, kad per 100 laiko vienetų bus pagaminta TC, kurios id yra 7 [0.998848,0.999848]. Lygiai tokias pat tikimybes gaunamos su reikšmėmis nuo 1 iki 6.

Kitose užklausų vykdymuose susikoncentruosime į analizę, kiek užtrunka atskiros TC komandos apdorojimas iki atitinkamo TM pranešimo sukūrimo. Atskirai panagrinėsime du atvejus, kai TC yra sėkmingai validuota ir kai jos validacija buvo nesėkminga.

Sėkminga TC validacija:

Pr [<= 1000] (<> TC_handling.tm_made && TC_handling.current_tc == 7 && TC_handling.is_tc_valid == 1):

- 7598 bandymai;
- tikimybė: [0.999,1].



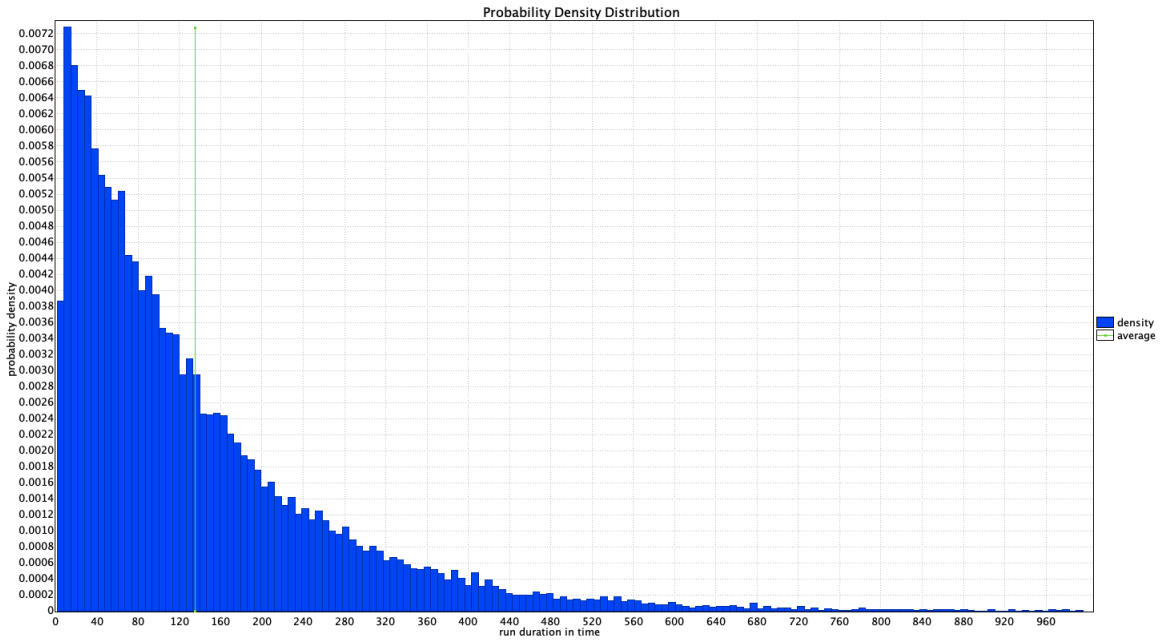
32 pav. Sėkminga TC validacija (tikimybinis laiko pasiskirstymas)

32 pav. vaizduoja tikimybinį laiko pasiskirstymą sėkmingos TC validacijos atveju. Vienas

akivaizdus ”pikas” yra gana stabilios ir subalansuotos sistemos požymis. Laiko vidurkio tikimybė deja nėra labai arti ”piko”, kas būtų visiškai stabilios sistemos atveju.

Nesėkminga TC validacija: *Pr [<= 1000] (<> TC_handling.tm_made && TC_handling.current_tc == 7 && TC_handling.is_tc_valid == 0)*:

- 22548 bandymai;
- tikimybė: [0.99888,0.99988].

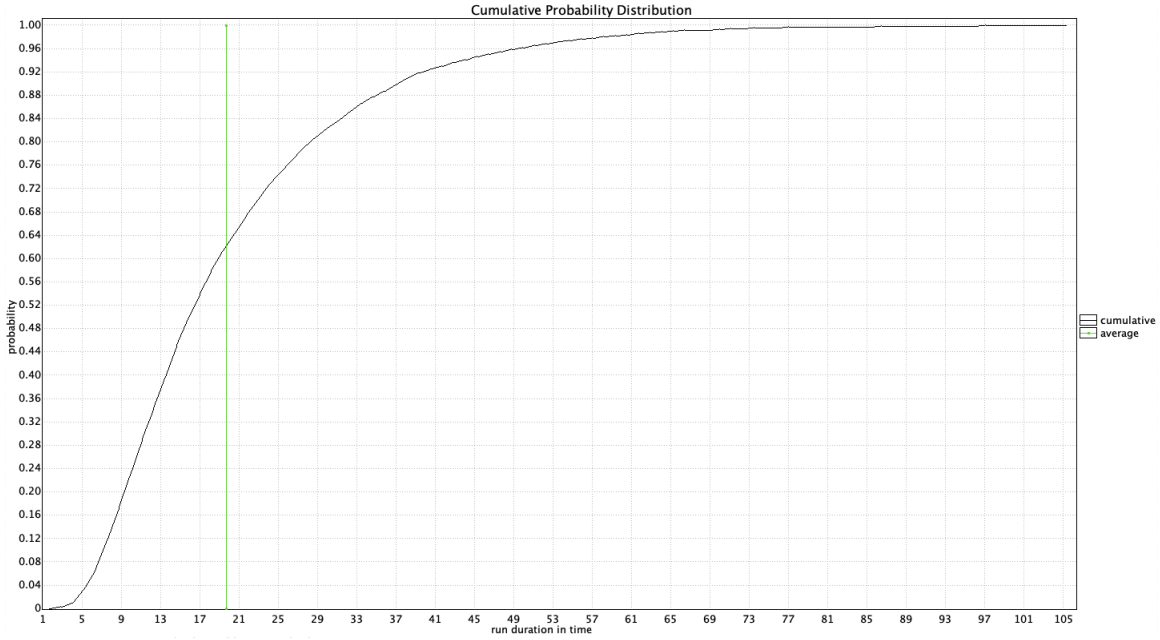


33 pav. Nesėkminga TC validacija (tikimybinis laiko pasiskirstymas)

	Dažn. apdorojimo intervalas	Vidurkis	Ilg. apdorojimo laikas
Sėkminga TC validacija	9-10	18	113
Nesėkminga TC validacija	8-14	133	989

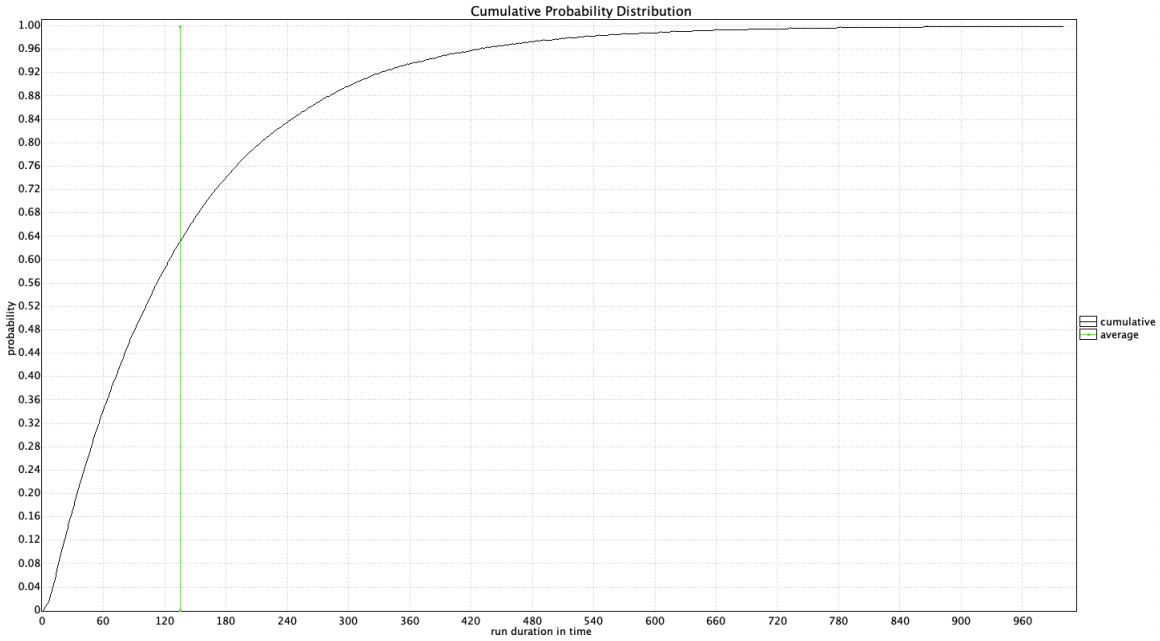
Dabar apibendrinsime gautas atskiros užklausos apdorojimo laiko reikšmes. Pagal gautus tikimybinio laiko pasiskirstymo grafikų duomenis matome, kad užklausos apdorojimas su sėkminga TC validacija vidutiniškai įvyksta žymiai greičiau negu nesėkmingos validacijos atveju. Vidutiniškai sėkmingas apdorojimas įvyksta ∼7,3 kartų greičiau. Taip pat nesėkmingos validacijos atveju užklausos apdorojimas gali įvykti 989 laiko vienete. Primename, kad šie duomenys atitinka 90% sėkmingos TC validacijos tikimybę.

Toliau apžvelgsime kumuliacinius tikimybinio pasiskirstymo grafikus.



34 pav. Sėkminga TC validacija (kumuliacinis tikimybinis pasiskirstymas)

Matome, kad sėkmingos TC validacijos atveju užklausos apdorojimas neužims daugiau negu ∼49 laiko vientų su ∼95% tikimybe.



35 pav. Nesėkminga TC validacija (kumuliacinis tikimybinis pasiskirstymas)

Matome, kad nesėkmingos TC validacijos atveju užklausos apdorojimas neužims daugiau negu ∼420 laiko vientų su ∼95% tikimybe.

Toliau bus tyrinėjama sėkmingos TC užklausos validacijos apdorojimas. Nuo šiol vykdoma užklausa yra: *Pr [<= 1000] (<> TC_handling.tm_made && TC_handling.current_tc == 7 && TC_handling.is_tc_valid == 1)*

Jei lentelės eilutės pradžioje yra (*) simbolis, tai reiškia, kad šie duomenys yra gauti su pradiniais sistemos nustatymais.

Sėkm. TC validacijos tikimybė	Dažn. apdorojimo intervalas	Vidurkis	Ilg. apdorojimo laikas
5%	9-17	84	728
10%	14-18	48	377
50%	8-9	20	136
* 90%	9-10	18	113
100%	11-12	18	111

Matome, kad sėkmingos TC validacijos tikimybė beveik nedaro įtakos dažniausio apdorojimo intervalui, tačiau stipriai keičia vidurkį ir ilgiausią apdorojimo laiką. Galime teigti, kad validavimo sėkmės tikimybė daro įtaką sėkmingos užklausos apdorojimo laikui.

Pateikta lentelė iliustruoja, kaip vieno sistemos parametro (šiuo atveju sėkmingos TC validacijos tikimybės) skirtingos reikšmės gali paveikti bendrą sistemos darbo laiką apdorojant gautas užklausas. Toliau panagrinėsime kitų sistemos parametrų poveikį.

3.3.2.1. TC validavimo laikas

TC validavimo laikui apibrėžti yra naudojami *tc_validate_nr* ir *tc_validate_dr* eksponentiniai dažniai.

Validavimo dažnis	Dažn. apdorojimo intervalas	Vidurkis	Ilg. apdorojimo laikas
$\frac{1}{10}$	59-69	121	901
$\frac{1}{5}$	33-37	66	423
$\frac{1}{2}$	17-19	34	226
1	11-14	24	161
* 2	9-10	18	113
10	8-9	14	96
100	8-9	13	100

Galima sakyti, kad validavimo dažnio didinimas beveik nedaro įtakos atskiros užklausos apdorojimo laikui; šiek tiek sumažėja vidurkis. Tačiau dažnio reikšmės mažinimas stipriai paveikia laiką. Taigi, galime teigti, kad sistemos validavimo proceso laikas stipriai veikia atskiros užklausos apdorojimo laiką. Konkrečiau, mažinant validavimo laiką žemiau 1 apdorojimo laikas eksponentiškai auga, tuo tarpu didinant jį virš 1 laiko reikšmės stabilizuojasi.

3.3.2.2. TM buferio dydis

TM buferio dydis	Dažn. apdorojimo intervalas	Vidurkis	Ilg. apdorojimo laikas
* 2	9-10	18	113
4	8-10	19	133
6	9-11	19	153
8	8-10	19	135
10	9-10	19	124
20	9-11	19	121

Aiškiai matome, kad TM buferio dydis nedaro įtakos atskiros užklausos apdorojimo laikui.

3.3.2.3. TC buferio dydis

TC buferio dydis	Dažn. apdorojimo intervalas	Vidurkis	Ilg. apdorojimo laikas
* 2	9-10	18	113
3	12-14	27	182
4	12-15	31	248
5	9-12	28	271
6	10-13	16	241
8	10-14	17	307
10	8-13	18	413
20	12-21	21	779

Jei TC buferio dydis yra nuo 3 iki 5, tai užklausos apdorojimo vidurkis padidėja $\sim$1.5 karto. Taip pat TC buferio dydis stipriai veikia ilgiausią apdorojimo laiką.

3.3.2.4. Siuntėjas

Siuntėjo *idle* (liet. tingioje) būsenoje naudojami *sender_idle_nr* ir *sender_idle_dr* eksponentiniai dažniai, nusakantys kaip dažnai siuntėjas atsisiunčia naują TC ir padeda ją į TC buferį. Pabandysime pažiūrėti, kaip keičiasi užklausos apdorojimo laikas keičiant *idle* būsenos dažnio reikšmę.

Siuntėjo *idle* dažnis	Dažn. apdorojimo intervalas	Vidurkis	Ilg. apdorojimo laikas
$\frac{1}{10}$	55-59	78	350
$\frac{1}{5}$	33-35	41	172
$\frac{1}{2}$	18-19	22	125
1	11-12	19	104
* 4	9-10	18	113
10	9-10	19	126
100	11-13	19	141

Siuntėjo *idle* dažnio didinimas beveik nedaro įtakos atskiros užklausos apdorojimo laikui. Tačiau dažnio reikšmės mažinimas stipriai paveikia laiką. Taigi, galime teigti, kad siuntėjo TC sukūrimo laikas stipriai veikia atskiros užklausos apdorojimo laiką.

3.3.2.5. Gavėjas

Analogiškai gavėjo *ready* būsenoje naudojami *receiver_ready_nr* ir *receiver_ready_dr* eksponentiniai dažniai, nusakantys, kaip dažnai gavėjas pasiima TM pranešimą iš TM buferio. Pažiūrėsime, kaip šio dažnio reikšmė veikia atskiros užklausos apdorojimo laiką.

Gavėjo *ready* dažnis	Dažn. apdorojimo intervalas	Vidurkis	Ilg. apdorojimo laikas
$\frac{1}{10}$	8-10	19	116
$\frac{1}{5}$	11-12	19	120
$\frac{1}{2}$	8-9	19	127
* 1	9-10	18	113
10	8-10	18	179
100	10-12	18	144

Aiškiai matome, kad gavėjo *ready* būsenos dažnio reikšmė nedaro įtakos atskiros užklausos apdorojimo laikui.

3.3.2.6. Instrumentas

Instrumento *busy* būsenoje naudojami *instrument_busy_nr* ir *instrument_busy_dr* eksponentiniai dažniai, nusakantys laiko uždelsimą gaminant TM duomenis. Pažiūrėsime, kaip šio dažnio reikšmė veikia atskiros užklausos apdorojimo laiką.

Instrumento *busy* dažnis	Dažn. apdorojimo intervalas	Vidurkis	Ilg. apdorojimo laikas
$\frac{1}{10}$	44-51	99	601
$\frac{1}{5}$	28-32	52	362
$\frac{1}{2}$	12-14	23	169
* $\frac{2}{3}$	9-10	18	113
2	4-5	10	62
10	3-4	6	41
100	3	5	34

Instrumento *busy* būsenos dažnis stipriai veikia užklausos apdorojimo laiką. Kuo didesnė *busy* dažnio reikšmė tuo mažesnis apdorojimo vidurkis.

3.3.2.7. Diagnostikos gamintojas

Diagnostikos gamintojo *idle* būsenoje naudojami *producer_idle_nr* ir *producer_idle_dr* eksponentiniai dažniai, nusakantys su kokiu uždelsimu yra generuojami nauji diagnostikos duomenys. Pažiūrėsime, kaip keičiasi užklausos apdorojimo laikas keičiant *idle* būsenos dažnio reikšmę.

Diag. gam. *idle* dažnis	Dažn. apdorojimo intervalas	Vidurkis	Ilg. apdorojimo laikas
$\frac{1}{10}$	10-12	19	123
$\frac{1}{5}$	10-12	18	148
$\frac{1}{2}$	10-12	18	143
* 2	9-10	18	113
10	10-13	19	119
100	8-10	19	138

Aiškiai matome, kad diagnostikos gamintojo *idle* būsenos dažnio reikšmė nedaro įtakos atskiros užklausos apdorojimo laikui.

Taip pat diagnostikos gamintojas pereidamas iš *idle* būsenos turi apsaugą (angl. guard): *HK_c == HK_period* – iš šios būsenos galima pereiti, kai diagnostikos gamintojo laikrodžio reikšmė yra lygi jo periodui.

Pažiūrėsime, kaip keičiasi užklausos apdorojimo laikas keičiant diagnostikos gamintojo *HK_period* reikšmę.

Diag. gam. *HK_period* reikšmė	Dažn. apdorojimo intervalas	Vidurkis	Ilg. apdorojimo laikas
1	10-12	20	122
* 2	9-10	18	113
10	10-12	19	123
100	9-10	19	133

Aiškiai matome, kad diagnostikos gamintojo *HK_period* reikšmė nedaro įtakos atskiros užklausos apdorojimo laikui.

Išvados

Šiame bakalauro darbe buvo susipažinta su statistinio modelių tikrinimo teorija ir ją palaikančiu Uppaal-SMC įrankiu. Taip pat buvo pristatyta BepiColombo Duomenų Apdorojimo Bloko (DAB) sistema ir jos modelis, kuris anksčiau buvo realizuotas Uppaal įrankio, sukurto remiantis klasikine modelių tikrinimo teorija, pagalba.

Praktinėje dalyje buvo pritaikytos Uppaal-SMC įrankio modeliavimo, simuliavimo ir verifikavimo galimybės BepiColombo DAB sistemos modeliui. To rezultate ankstesnis sistemos Uppaal modelis buvo gerokai išplėstas, siekiant pilniau išnaudoti naujas Uppaal-SMC teikiamas verifikavimo ir modelių analizės galimybes. Konkrečiau, modelių būsenų invariantai buvo pakeisti eksponentiniais dažniais. Be to, TC valdymo modeliui buvo pridėti tikimybiniai svoriai ir loginiai kintamieji. Taip pat išplėtus modelį buvo nagrinėjamas sistemos efektyvumas. Šios analizės metu buvo įsigilinta, kaip įvairi sistemos konfigūracija daro įtaką pagrindiniam sistemos funkcionalumui, t.y., atskiros užklausos apdorojimo laikui. Buvo pastebėta, kad TC sėkmingos validacijos tikimybė bei uždelsimo dažnis stipriai veikia sėkmingos užklausos apdorojimo laiką. Taip pat įsitikinta, kad kai TC buferio dydis yra tarp 3..5, tai smarkiai prailgsta vidutinis užklausos apdorojimo laikas. Be to apdorojimo laiką stipriai veikia instrumento ir siuntėjo uždelsimų eksponentinių dažnių reikšmės. Apibendrinant tyrimų rezultatus, galima pastebėti, kad bendrai sistema veikia pakankamai stabiliai daugelyje išnagrinėtų konfigūracijų sugrąžindama priimtinas analizės reikšmes.

Abstract

The purpose of this bachelor thesis is to gain an understanding of the statistical model checking theory, the Uppaal-SMC tool and, using the learned methods and techniques, analyze the effectiveness of the Data Processing Unit (DPU) of the BepiColombo spacecraft. BepiColombo is a joint mission of the European Space Agency (ESA) and the Japan Aerospace Exploration Agency (JAXA) to the planet Mercury. DPU is a part of the BepiColombo system and is responsible for receiving telecommands (TC) from the spacecraft and transmitting science and housekeeping telemetry data (TM) back to it.

In the practical part of the work the analysis of an individual BepiColombo request is performed with different system configurations. The obtained results showed that system effectiveness can be affected by the TC validation probability weights and exponential rates, as well as the TC buffer size and exponential rates of the DPU instrument and sender of TM. For all the considered configurations the system analysis showed that the system is rather stable, i.e., it does not exhibit unexpected or extreme behavior.

Literatūra

[AD94] Rajeev Alur ir David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[Alu99] Rajeev Alur. Timed automata. *International Conference on Computer Aided Verification*, p.p. 8–22. Springer, 1999.

[BC06] C Baier ir F Ciesinski. Liquor: a tool for qualitative and quantitative linear time analysis of reactive systems. *Third International Conference on the Quantitative Evaluation of Systems-(QEST'06)*, p.p. 131–132. IEEE, 2006.

[BCC+99] Armin Biere, Alessandro Cimatti, Edmund Clarke ir Yunshan Zhu. Symbolic model checking without bdds. *International conference on tools and algorithms for the construction and analysis of systems*, p.p. 193–207. Springer, 1999.

[BCL+13] Benoît Boyer, Kevin Corre, Axel Legay ir Sean Sedwards. Plasma-lab: a flexible, distributable statistical model checking library. *International Conference on Quantitative Evaluation of Systems*, p.p. 160–164. Springer, 2013.

[BCM+92] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill ir Lain-Jinn Hwang. Symbolic model checking: 1020 states and beyond. *Information and computation*, 98(2):142–170, 1992.

[BDL04] Gerd Behrmann, Alexandre David ir Kim G Larsen. A tutorial on uppaal. *Formal methods for the design of real-time systems*, p.p. 200–236. Springer, 2004.

[BHH+03] Christel Baier, Boudewijn Haverkort, Holger Hermanns ir J-P Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Transactions on software engineering*, 29(6):524–541, 2003.

[BJK+05] Manfred Broy, Bengt Jonsson, J-P Katoen, Martin Leucker ir Alexander Pretschner. Model-based testing of reactive systems. *Volume 3472 of Springer LNCS*. Springer, 2005.

[BK08] Christel Baier ir Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[BMR05] Thomas Ball, Todd Millstein ir Sriram K Rajamani. Polymorphic predicate abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 27(2):314–343, 2005.

[Bry92] Randal E Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992.

[CCQ02] Gianpiero Cabodi, Paolo Camurati ir Stefano Quer. Can bdds compete with sat solvers on bounded model checking? *Proceedings of the 39th annual Design Automation Conference*, p.p. 117–122, 2002.

[CE81] Edmund M Clarke ir E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. *Workshop on Logic of Programs*, p.p. 52–71. Springer, 1981.

[CG04] Frank Ciesinski ir Marcus Größer. On probabilistic computation tree logic. *Validation of Stochastic Systems*, p.p. 147–188. Springer, 2004.

[CGK+18] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled ir Helmut Veith. *Model checking*. MIT press, 2018.

[CY95] Costas Courcoubetis ir Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM (JACM)*, 42(4):857–907, 1995.

[CV03] Edmund Clarke ir Helmut Veith. Counterexamples revisited: principles, algorithms, applications. *Verification: Theory and Practice*, p.p. 208–224. Springer, 2003.

[DG09] Manfred Droste ir Paul Gastin. Weighted automata and weighted logics. *Handbook of weighted automata*, p.p. 175–211. Springer, 2009.

[DLL+11] Alexandre David, Kim G Larsen, Axel Legay, Marius Mikučionis ir Zheng Wang. Time for statistical model checking of real-time systems. *International Conference on Computer Aided Verification*, p.p. 349–355. Springer, 2011.

[DLL+15] Alexandre David, Kim G Larsen, Axel Legay, Marius Mikučionis ir Danny Bøgsted Poulsen. Uppaal smc tutorial. *International Journal on Software Tools for Technology Transfer*, 17(4):397–415, 2015.

[DPP04] Agostino Dovier, Carla Piazza ir Alberto Policriti. An efficient algorithm for computing bisimulation equivalence. *Theoretical Computer Science*, 311(1-3):221–256, 2004.

[FG05] Cormac Flanagan ir Patrice Godefroid. Dynamic partial-order reduction for model checking software. *ACM Sigplan Notices*, 40(1):110–121, 2005.

[You05a] Hakan L Younes. Verification and planning for stochastic processes with asynchronous events. Tech. atask., CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2005.

[You05b] Hakan L Younes. Verification and planning for stochastic processes with asynchronous events. Tech. atask., CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2005.

[YPD95a] Wang Yi, Paul Pettersson ir Mats Daniels. Automatic verification of real-time communicating systems by constraint-solving. *Formal Description Techniques VII*, p.p. 243–258. Springer, 1995.

[YPD95b] Wang Yi, Paul Pettersson ir Mats Daniels. Automatic verification of real-time communicating systems by constraint-solving. *Formal Description Techniques VII*, p.p. 243–258. Springer, 1995.

[IRL+12] Alexei Iliasov, Alexander Romanovsky, Linas Laibinis, Elena Troubitsyna ir Timo Latvala. Augmenting event-b modelling with real-time verification. *2012 First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA)*, p.p. 51–57. IEEE, 2012.

[YS02] Håkan LS Younes ir Reid G Simmons. Probabilistic verification of discrete event systems using acceptance sampling. *International Conference on Computer Aided Verification*, p.p. 223–235. Springer, 2002.

[KNP04] Marta Kwiatkowska, Gethin Norman ir David Parker. Prism 2.0: a tool for probabilistic model checking. *First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings.* P.p. 322–323. IEEE, 2004.

[KNP11] Marta Kwiatkowska, Gethin Norman ir David Parker. Prism 4.0: verification of probabilistic real-time systems. *International conference on computer aided verification*, p.p. 585–591. Springer, 2011.

[LL16] Kim G Larsen ir Axel Legay. Statistical model checking: past, present, and future. *International Symposium on Leveraging Applications of Formal Methods*, p.p. 3–15. Springer, 2016.

[LPY97] Kim G Larsen, Paul Pettersson ir Wang Yi. Uppaal in a nutshell. *International journal on software tools for technology transfer*, 1(1-2):134–152, 1997.

[Pnu77] Amir Pnueli. The temporal logic of programs. *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, p.p. 46–57. IEEE, 1977.

[SVA04] Koushik Sen, Mahesh Viswanathan ir Gul Agha. Statistical model checking of black-box probabilistic systems. *International Conference on Computer Aided Verification*, p.p. 202–215. Springer, 2004.

[WG93] Pierre Wolper ir Patrice Godefroid. Partial-order methods for temporal verification. *International Conference on Concurrency Theory*, p.p. 233–246. Springer, 1993.