Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Temporális logikai specifikációk vizsgálata

Diplomaterv

Készítette
Segesdi Dániel

Konzulens
Vörös András
Dr. Bartha Tamás

2014. december 23.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Segesdi Dániel*, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2014. december 23.

Segesdi Dániel
hallgató

Kivonat

Biztonságkritikus rendszerek hibái akár katasztrofális következményekkel járhatnak, ezért kiemelt fontosságú a helyes működés biztosítása. Annak igazolásához, hogy a fejlesztett rendszer a specifikációnak megfelelően, hiba nélkül működik, célszerű formális módszereket alkalmazni. Egy ilyen módszer a modellellenőrzés, aminek segítségével még tervezési időben kiszűrhetőek az esetleges hibák. Modellellenőrzés során a rendszer modelljén vizsgáljuk az úgynevezett temporális logikák segítségével megfogalmazott követelmények teljesülését. Több féle temporális logika létezik, dolgozatomban a lineáris temporális logikán (LTL) alapuló modellellenőrzéssel foglalkozom.

Amennyiben egy vizsgált követelmény nem teljesül a modellen, az ellenőrzés eredményeképp kapunk egy ellenpéldát, aminek birtokában lehetőségünk van a hiba javítására. Ezzel szemben ha a követelmény teljesül a modellen, nem kapunk információt a teljesülés módjáról. Ez azért jelent problémát, mert gyakran előfordul, hogy a temporális logikai kifejezés hibás megfogalmazása miatt triviális módon teljesül egy követelmény, így az esetleges hibára nem derül fény.

A hibásan megfogalmazott követelmények felismerésére az egyik módszer az LTL-kifejezések úgynevezett vacuity értékeinek vizsgálata, ami arról ad információt, hogy a kifejezés mely részei feleslegesek a követelmény teljesülésének szempontjából.

Dolgozatomban bemutatok egy vacuity értékek vizsgálatán alapuló módszert, ami segítségével figyelmeztethetjük a felhasználót ilyen típusú hibák esetén is. Ez után ismertetem a módszer egy általam egyszerűsített változatát is, melynek implementációját elkészítettem a PetriDotNet modellellenőrző keretrendszer részeként.

Abstract

Since the failures of safety-critical systems may have disastrous consequences, ensuring their correct behaviour is an important task. To verify that the system is working without errors, the use of formal methods are practical. Model-checking is such a method which can be used during the design period to find faults in the system. During the execution of model-checking the validity of requirements expressed with temporal logics are examined. There exist several versions of temporal logics, of which in my work I use linear temporal logics.

If a requirement is not valid on a model, we recieve a counterexample as the result of the model-checking process, with which we can identify faults of the model. In contrast if the requirement is valid on the model, we do not recieve any additional information. This presents a problem because the incorrect translation of an invalid requirement frequently results in a trivially valid LTL-formula, therefore it is impossible to identify the fault.

Finding the so-called vacuity values of an LTL-formula can help us to identify these incorrect translations by providing information on which parts of the formula are unnecessary for the validity of the requirement.

In my work I present a method based on the examination of vacuity values, with which we can warn the user in case of this type of error. Furthermore I present my variation of this method, what I implemented as part of the PetriDotNet model-checking framework.

Bevezető

Napjainkban egyre nagyobb teret hódít a modell alapú szoftverfejlesztés és az ezzel kapcsolatos modellellenőrzés, ami segítségével formálisan bizonyíthatóvá válik a modellek helyes működése. Bár a módszer népszerűsége egyre növekszik, széleskörű elterjedéséhez szükség van az eddigi eszközök továbbfejlesztésére, hogy ezzel egyrészt javítani lehessen a teljesítménybeli problémákon, másrészt hogy növeljük a visszajelzések hasznosságát.

Jelen dolgozat témája az utóbbi kategóriába esik. A bemutatott módszerek használatával igyekszik olyan segítséget nyújtani a modellellenőrzési folyamat során, amivel mind a modellekben, mind a lineáris temporális logikai (LTL) kifejezések formájában megfogalmazott követelményekben található hibák felderíthetővé és javíthatóvá válnak.

A tárgyalt probléma alapja az, hogy míg egy nem teljesülő követelmény esetén a modellellenőrzés melléktermékeként kapunk egy ellenpéldát, ami alapján megtalálhatjuk a hiba forrását, addig teljesülő követelmények esetén nem kapunk ilyen típusú visszajelzést. Ennek hiányában nem tudunk egyszerűen megbizonyosodni arról, hogy a modellünk vajon tényleg helyesen működik, vagy például a követelmény hibás megfogalmazása miatt kaptunk ilyen eredményt[1].

A dolgozatban bemutatok egy olyan algoritmust, amivel teljesülő követelmények esetén további visszajelzést tudunk adni a felhasználónak egy a modellt pontosabban leíró LTL kifejezés formájában. Ez alapján könnyebben kiszűrhetővé válnak a követelmény felírásakor elkövetett hibák, pontosabb képet adhatunk arról, hogy a modell hogyan és miért teljesíti az ellenőrzéshez megadott formulát.

A továbbiakban először bemutatom a modellellenőrzés megértéséhez szükséges háttérismereteket (1. fejezet), majd ismertetem az algoritmus működésének alapjául szolgáló vacuity[1] fogalmát és az ehhez kapcsolódó matematikai struktúrákat (2. fejezet). Bemutatom továbbá az algoritmus működését, és annak egy egyszerűtett változatát (3.fejezet), aminek implementációját elkészítettem a PetriDotNet modellellenőrző keretrendszerbe illeszkedően. Végül bemutatom az elkészült implementációval végrehajtott mérések során nyert adatokat, és ez alapján értékelem annak működését (5. fejezet).

[1] A dolgozatban az angol terminológiát használom, mivel a magyar irodalomban nem találtam rá elfogadott kifejezést, a saját fordításaim pedig valamilyen szempontból mind félrevezetőek voltak.

1. fejezet

Háttérismeretek

Ebben a fejezetben bemutatom a munkám megértéséhez szükséges elméleti hátteret. Először a lineáris temporális logikákról és a modellellenőrzés során betöltött szerepükről lesz szó (1.1. szakasz), majd ismertetem az ellenőrzés köztes lépéseiben használt különböző automatatípusokat (1.2. szakasz), végül bemutatom a modellellenőrzés folyamatát (1.3. szakasz).

1.1. Követelmények specifikálása temporális logikákkal

A modellellenőrzés során a vizsgált rendszer tulajdonságait vetjük össze az elvárt viselkedéssel. Ezt a specifikált viselkedést sokszor deklaratívan, különböző logikák alkalmazásával célszerű megfogalmazni. A dinamikus viselkedésre vonatkozó tulajdonságok leírására használhatók a temporális logikák [13], amelyek közül a jelen dolgozatban bemutatott algoritmus a *lineáris idejű temporális logikát* (LTL) [15] használja. Az alábbiakban bemutatom az LTL felépítését.

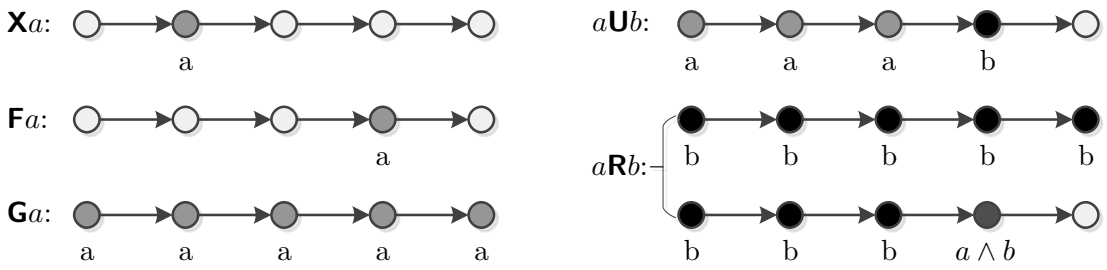
1.1.1. Lineáris idejű temporális logika

Mint a Bool-logika kiterjesztése, a lineáris idejű temporális logika (LTL) elemi kifejezésekből (ebben az esetben állapotkifejezésekből), illetve az ezek kapcsolatát kifejező logikai operátorokból épül fel. Ezeken kívül azonban reaktív rendszerekben az állapotok közötti átmenetek is érdekesek. Az ilyen rendszerek folyamatos kölcsönhatásban vannak a környezetükkel, és legtöbbször ciklikus viselkedésűek, azaz nem áll le a futásuk. Emiatt a vizsgálatuk során nem elég egy korlátos lefutást vizsgálni.

A temporális logikák ezeket az állapotátmeneteket és állapotszekvenciákat képesek formálisan leírni, megfogalmazni. A lineáris temporális logikában (akárcsak az elágazó idejű temporális logikában) az idő nem jelenik meg közvetlenül, csak logikai formában. Egy kifejezés például előírhatja, hogy egy állapotnak valamikor a jövőben elő kell állnia, vagy egyáltalán nem állhat elő. Ezeket a fogalmakat a temporális logikák speciális, ún. *temporális operátorokkal* írják le, amelyek egymásba ágyazhatók, logikai operátorokkal kombinálhatók.

Az alábbiakban informálisan definiálom a hat legtöbbet használt LTL temporális operátort:

- **X** („ne**X**t time”) a következő állapotra ír elő feltételt,
- **F** („in the **F**uture”) a feltétel azonnali vagy jövőbeli bekövetkezését követeli meg,
- **G** („**G**lobally”) akkor teljesül, ha a kifejezés az összes jövőbeli és a jelenlegi állapotra is igaz,
- **U** („**U**ntil”) egy kétoperandusú operátor, ami akkor teljesül, ha valamikor a jövőben létezik egy állapot, amire a második operandus igaz, és addig minden állapot kielégíti az első operandust,
- **R** („**R**elease”) az **U**operátor logikai duálisa, teljesüléséhez a második operandusnak kell teljesülnie mindaddig, míg egy állapotra nem teljesül az első operandus *is*. Az első operandusnak azonban nem kell mindenképpen teljesülnie a jövőben.



1.1. ábra. Az LTL temporális operátorok szemléletes jelentése.

Az előbbi operátorokkal definiált LTL kifejezések csak végtelen hosszú állapotszekvenciákon érvényesek. Időnként célszerű, hogy véges szekvenciákra is definiálni lehessen LTL kifejezéseket, így bevezetjük az alábbi operátort is:

- $\tilde{\mathbf{X}}$ („weak ne**X**t time”) az **X** logikai duálisa, amely ugyanúgy a következő állapotra ír elő feltételt, de akkor is teljesül, ha nem létezik következő állapot.

Végtelen szekvenciákon az **X** duálisa önmaga, vagyis ilyenkor $\mathbf{X} \equiv \tilde{\mathbf{X}}$. Véges lefutások vizsgálatakor azonban kérdés az is, hogy létezik-e következő állapot, így a duálisban ezt is másképp kell kezelni. Mivel minden más temporális operátor visszavezethető **X**-re (rekurzívan) [22], így elegendő ezt az egy operátort „felkészíteni” a véges szekvenciákra és a rekurzív képletekben megfelelően használni a kétféle operátort.
Az LTL kifejezések szintaxisa a következő módon definiálható [12]:

- Minden P atomi kifejezés egy állapotkifejezés,
- Ha p és q állapotkifejezések, akkor $\neg p$ és $p \wedge q$ is állapotkifejezés,
- Ha p és q állapotkifejezések, akkor $\mathbf{X}\ p$ és $p\ \mathbf{U}\ q$ is állapotkifejezések.

A többi operátor a következő szabályok segítségével fejezhető ki:

- $p \vee q \equiv \neg(\neg p \wedge \neg q)$
- $\tilde{\mathbf{X}}\ p \equiv \neg\mathbf{X}\ \neg p$
- $p\ \mathbf{R}\ q \equiv \neg(\neg p\ \mathbf{U}\ \neg q)$
- $\mathbf{F}\ p \equiv igaz\ \mathbf{U}\ p$
- $\mathbf{G}\ p \equiv \neg\mathbf{F}\ \neg p$

Egy LTL kifejezés *negált normál formáján* egy olyan ekvivalens kifejezést értünk, amiben kizárólag a $\wedge$, $\vee$ és $\neg$ Boole-operátorok, valamint az $\mathbf{X}$, $\tilde{\mathbf{X}}$, $\mathbf{U}$ és $\mathbf{R}$ temporális operátorok szerepelhetnek, és negálás csak atomi kijelentések előtt állhat. Ahhoz, hogy egy kifejezést negált normál formába hozzunk, elsőként át kell írnunk az $\mathbf{F}\ \psi$ és $\mathbf{G}\ \psi$ alakú kifejezéseket az alábbi azonosságok szerint:

- $\mathbf{F}\ \psi \equiv igaz\ \mathbf{U}\ \psi$
- $\mathbf{G}\ \psi \equiv hamis\ \mathbf{R}\ \psi$

Ezután a megfelelő Boole-algebrai azonosságok segítségével minden logikai operátort ki kell fejeznünk az *és* ($\wedge$), *vagy* ($\vee$), illetve *nem* ($\neg$) operátorok segítségével. Végül a negálásokat „be kell vinnünk" az atomi kijelentések elé a De Morgan-azonosságok és az alábbi három temporális azonosság segítségével:

- $\neg(\mu\ \mathbf{U}\ \eta) \equiv (\neg\mu)\ \mathbf{R}\ (\neg\eta)$
- $\neg(\mu\ \mathbf{R}\ \eta) \equiv (\neg\mu)\ \mathbf{U}\ (\neg\eta)$
- $\neg\mathbf{X}\ \mu \equiv \tilde{\mathbf{X}}\ (\neg\mu)$

1.2. Automataelmélet

Ebben a fejezetben a modellellenőrző algoritmus alapját képező automatákat, azok típusait és a rajtuk végezhető műveleteket mutatom be.

1.2.1. Véges automaták

Véges automata alatt egy olyan matematikai számítási modellt értünk, ami a bemenetére adott szimbólumsorozatokról, ún. *szavakról* a bemenet méretétől független, véges és állandó mennyiségű memória felhasználásával eldönti, hogy egy *nyelv* részét képezik-e. Egy véges automata működhet véges és végtelen bemeneteken (úgynevezett szavakon).

Formálisan, egy (véges szavakon működő) véges $\mathcal{A}$ automata egy $\langle \Sigma, Q, \Delta, Q^0, F \rangle$ ötös, ahol

- Σ a véges *ábécé*.
- Q a lehetséges *állapotok* véges halmaza.
- $\Delta \subseteq Q \times \Sigma \times Q$ az *állapotátmeneti reláció*.

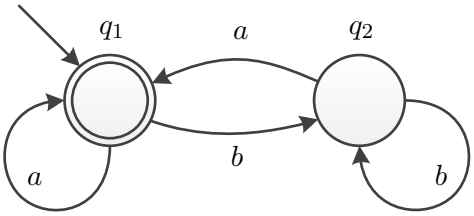
- $Q^0 \subseteq Q$ a *kiinduló állapotok* halmaza.
- $F \subseteq Q$ az *elfogadó állapotok* halmaza.

Egy automatát grafikusan egy olyan élcímkézett gráffal ábrázolhatunk, amiben a csomópontoknak Q, az éleknek pedig Δ elemei felelnek meg. Az élek címkéi Σ elemeiből állnak elő, ezért a betűket sokszor élkifejezéseknek is hívjuk.

Legyen $v \in \Sigma^*$ egy szó, melynek hossza $|v|$. $\rho : \{0, 1, \ldots, |v|\} \mapsto Q$ leképezést $\mathcal{A}$ egy *lefutásának* nevezzük v-n, ahol:

- $\rho(0) \in Q^0$, tehát az első állapot egy kiinduló állapot.
- Az i. állapotból az $i+1$. állapotba a bemenet i. betűjének olvasásakor akkor léphetünk, ha az átmenet szerepel az állapotátmeneti relációban (*engedélyezett*), vagyis minden $0 \leq i < |v|$-re $(\rho(i), v(i), \rho(i+1)) \in \Delta$.

Ekkor azt mondjuk, hogy $\mathcal{A}$ *olvassa* v-t, vagyis v *bemenete* $\mathcal{A}$-nak. Egy ρ lefutást v-n *elfogadónak* nevezünk, ha egy elfogadó állapotban ér véget, vagyis $\rho(|v|) \in F$. Egy $\mathcal{A}$ automata akkor és csak akkor *fogadja el* a v szót, ha létezik $\mathcal{A}$-nak v-n elfogadó lefutása. Az $\mathcal{A}$ által elfogadott $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^*$ *nyelv* az összes $\mathcal{A}$ által elfogadott szó halmaza.



1.2. ábra. Egy egyszerű véges automata.

Példa 1. *Tekintsük az 1.2. ábrán látható automatát. A kezdőállapot q_1, ezt a forrás nélküli nyíl jelöli. Ez egyben az egyetlen elfogadó állapot is, amit a dupla kör jelöl.*

Ez az automata elfogadja például az „aabba" szót, mivel ez a bemenet a $q_1q_1q_1q_2q_2q_1$ lefutást eredményezi, aminek utolsó állapota elfogadó állapot.

*Az összes elfogadott szó, vagyis az automata nyelve együtt $\varepsilon + (a+b)^*a$ alakban írható fel, ahol a + választást jelöl, a * pedig tetszőleges, de véges számú ismétlődést. A nyelv tehát az ε üres szóból, vagy olyan szavakból áll, amikben tetszőleges számú „a" vagy „b" után „a" zárja a sort.*

Egy véges automata lehet determinisztikus vagy nem-determinisztikus. Előbbi esetben Δ-t egy egyértelmű leképezésként értelmezzük, vagyis $\Delta : Q \times \Sigma \mapsto Q$. Ezzel kikötjük, hogy egy (q, a, q') és egy (q, a, q'') alakú tranzíció esetén $q' = q''$ mindig teljesül. Nem-determinisztikus esetben $q' \neq q''$ is megengedett. A továbbiakban, ha nincs más külön jelezve, nem-determinisztikus automatákkal foglalkozunk.

Mint korábban is említettem, a modellellenőrzés által vizsgált rendszerek többnyire reaktív rendszerek, ami azt jelenti, hogy rendeltetésszerű működésük során nem állnak le. A továbbiakban ezért bemutatjuk azokat a végtelen szavakon működő automata osztályokat is, amik ilyen rendszerek tulajdonságainak modellezésére alkalmazhatók. Ezeknek az

automatáknak a struktúrája megegyezik a véges szavakon működő automatákéval, azonban Σ^ω-beli szavakat ismernek fel, ahol az ω felső index a szóban szereplő végtelen számú betűre, vagyis állapotátmenetre utal.

1.2.2. Büchi-automaták

A legegyszerűbb végtelen szavakon működő véges automaták a Büchi-automaták[3]. Egy Büchi-automatának ugyanolyan komponensei vannak, mint egy véges szavakon működő automatának. Egy $\mathcal{A}$ Büchi-automata egy lefutása $v \in \Sigma^\omega$-n is majdnem ugyanúgy kerül definiálásra, ezúttal azonban $|v| = \omega$. Így a ρ lefutás értelmezési tartománya a természetes számok halmaza, vagyis $\rho : \mathbb{N} \mapsto Q$.

Jelöljük $inf(\rho)$-val azoknak az állapotoknak a halmazát, amelyek végtelenül gyakran szerepelnek egy lefutásban. Egy ρ lefutás $\mathcal{A}$-n akkor és csak akkor *elfogadó*, ha van olyan elfogadó állapot, ami végtelenül gyakran jelenik meg ρ-ban, vagyis $inf(\rho) \cap F \neq \emptyset$.

A Büchi-automaták ún. ω-reguláris nyelveket fogadnak el. Ugyanilyen nyelveket képesek leírni az LTL kifejezések is, azonban a Büchi-automaták kifejezőereje nagyobb. Elmondható, hogy egy LTL kifejezéshez mindig létezik azonos szavakat elfogadó Büchi-automata [19], azonban nem minden Büchi-automatához alkotható ekvivalens LTL kifejezés [25].

Példa 2. *Az 1.2. ábrán látható automata Büchi-automataként is értelmezhető. Ebben az esetben elfogadja például az $(ab)^\omega$ szót, ami „a”-k és „b”-k „a”-val kezdődő végtelen hosszú váltakozása. Az automata által elfogadott nyelv tartalmaz minden olyan szót, ami végtelenül sok a-t tartalmaz. Ezeket a szavakat a $(b^*a)^\omega$ ω-reguláris kifejezés írja le.*

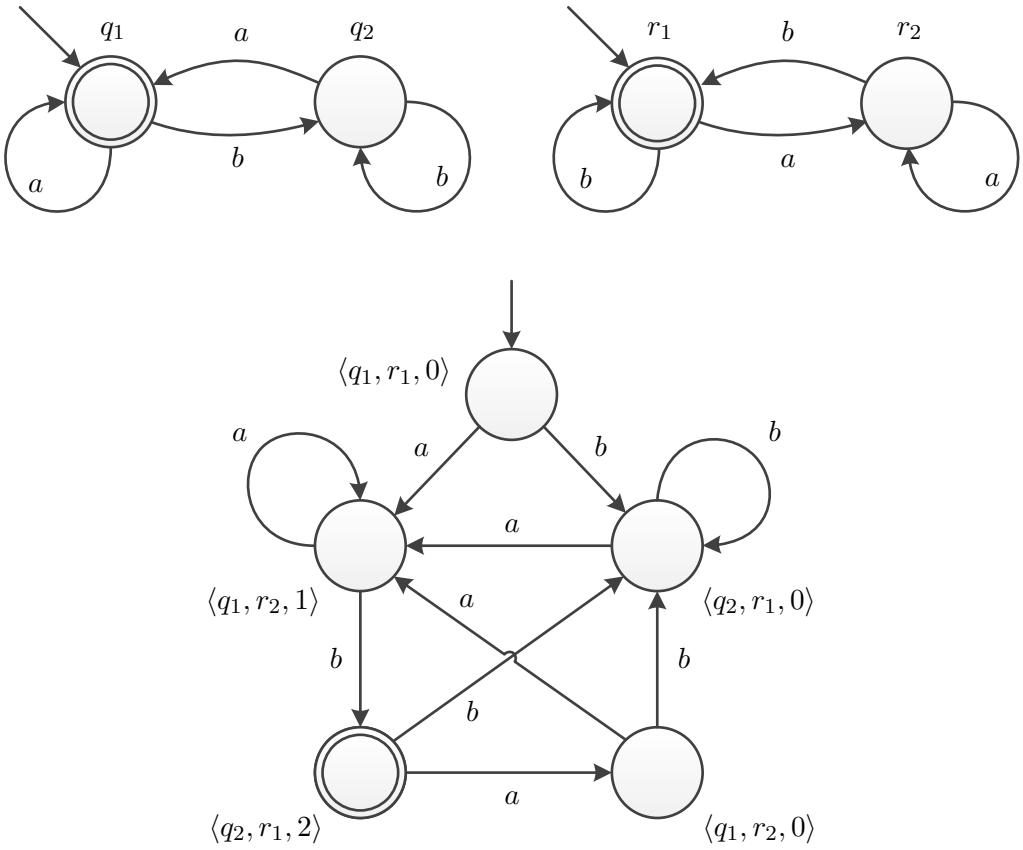
1.2.3. Automaták szinkron szorzata

Két egyszerű Büchi-automata *szinkron szorzatán* egy olyan automatát értünk, ami pontosan azokat a szavakat fogadja el, amelyeket mindkét automata elfogad. Formálisan, ha adottak $\mathcal{A}_1 = \langle \Sigma, Q_1, \Delta_1, Q_1^0, F_1 \rangle$ és $\mathcal{A}_2 = \langle \Sigma, Q_2, \Delta_2, Q_2^0, F_2 \rangle$ Büchi-automaták, a szorzat automata által elfogadott nyelv $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$, az automata pedig

$$\mathcal{A}_1 \cap \mathcal{A}_2 = \langle \Sigma, Q_1 \times Q_2 \times \{0,1,2\}, \Delta, Q_1^0 \times Q_2^0 \times \{0\}, Q_1 \times Q_2 \times \{2\} \rangle$$

alakban adódik. A tranzíciókat tekintve $(\langle r_i, q_j, x \rangle, a, \langle r_m, q_n, y \rangle) \in \Delta$ akkor és csak akkor áll fenn, ha

- $(r_i, a, r_m) \in \Delta_1$ és $(q_j, a, q_n) \in \Delta_2$, tehát a megfelelő állapotátmenetek a két szorzandó automatában is megléphetők
- a harmadik komponens állapota az $\mathcal{A}_1$ és $\mathcal{A}_2$ elfogadó állapotaitól függ:
  - ha $x = 0$ és $r_m \in F_1$, akkor $y = 1$
  - ha $x = 1$ és $q_n \in F_2$, akkor $y = 2$
  - ha $x = 2$, akkor $y = 0$
  - egyébként $y = x$.



1.3. ábra. Két Büchi-automata és szinkron szorzatuk [12]. A szorzat automatában csak az elérhető állapotok szerepelnek.

A harmadik komponens biztosítja, hogy a két automata elfogadó állapotai közül mindkettőből végtelenül gyakran jelenjen meg állapot. Önmagában az $F = F_1 \times F_2$ megkötés nem lenne elégséges, mivel a szorzat automatát a két szorzandó automata olyan együtteseként kell elképzelnünk, ahol mindkettő egyszerre lép a bemenet hatására. A szorzat akkor fogad el egy szót, ha $\mathcal{A}_1$ és $\mathcal{A}_2$ is elfogadja. Ez viszont nem jelenti azt, hogy a két automatának bármikor is egyszerre kellene elfogadó állapotban lennie, elég, ha mindkettő végtelen gyakran halad át ilyen állapoton (pl. felváltva).

A harmadik komponens kezdetben 0. Akkor változik 1-re, ha megjelenik egy állapot az első automata elfogadó állapotai közül. Ha ekkor a második automata elfogadó állapotai közül is érkezik egy állapot, akkor a harmadik komponens 2-re növekszik, és megkapjuk a szorzat automata egy elfogadó állapotát. A következő állapotban a számláló visszatér 0-ba, és újra az első automata elfogadó állapotait várjuk. A szorzat automata akkor fogad el egy lefutást, ha az végtelen sok olyan állapotot tartalmaz, amiben a harmadik komponens értéke 2.

Ezzel biztosítottuk, hogy *mindkét* automata elfogadó állapotai végtelen gyakran jelenjenek meg. Ha valamely ponton az egyik automatától nem találunk több elfogadó állapotot, akkor az az automata nem fogadja el a szót. A számláló nem növekszik tovább, így a szorzat automata lefutása sem lesz elfogadó.

Megjegyzendő, hogy az így kapott szorzat automata a két állapottér és a $\{0, 1, 2\}$ halmaz Descartes szorzata miatt olyan állapotokat is tartalmaz, amik nem elérhetők a kezdőállapotokból. A gyakorlati alkalmazás során ezeket érdemes felderíteni és eltávolítani.

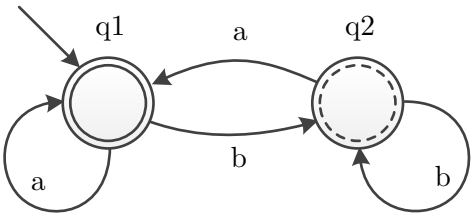
Szerencsére a szorzat automata előállítása ennél sokkal egyszerűbb, ha az egyik automata minden állapota elfogadó állapot. Ilyen automatát kapunk például, ha egy modell állapotterét automataként reprezentáljuk, ami – mint később bemutatjuk – éppen így is lesz az LTL modellellenőrzés során.

A szorzat automata ilyenkor $\mathcal{A}_1 \cap \mathcal{A}_2 = \langle \Sigma, Q_1 \times Q_2, \Delta', Q_1^0 \times Q_2^0, Q_1 \times F_2 \rangle$ alakú. Az elfogadó állapotok $Q_1 \times F_2$-beli párok, amikben a második tag a második automata egy elfogadó állapota. Az állapotátmenetek esetében $(\langle r_i, q_j \rangle, a, \langle r_m, q_n \rangle) \in \Delta'$ akkor és csak akkor áll fenn, ha $(r_i, a, r_m) \in \Delta_1$ és $(q_j, a, q_n) \in \Delta_2$.

1.2.4. Általánosított Büchi-automaták

Időnként kényelmesebb lehet a Büchi-automaták egy olyan osztályát használni, amelyben több elfogadó állapot halmaz van. Ez nem befolyásolja az automata kifejezőerejét, de kompaktabb reprezentációt tesz lehetővé. [12]

Egy *általánosított Büchi-automata* elfogadó állapotokat kódoló komponense $F \subseteq 2^Q$ alakú, tehát Q valamely részhalmazainak halmaza. Egy ilyen automata valamely ρ lefutása akkor és csak akkor elfogadó, ha minden $P_i \in F$-re $inf(\rho) \cap P_i \neq \emptyset$. Ez azt jelenti, hogy minden elfogadó állapot halmaz legalább egy elemének végtelenül gyakran kell előfordulnia ρ-ban. Vegyük észre, hogy ez egyben azzal is jár, hogy *üres F esetén az automata minden lefutása elfogadó.*



1.4. ábra. Egy Büchi-automata.

Példa 3. *Tekintsük most az 1.4. ábrát, amin az előző automatához képest a q_2 állapot most egy második elfogadó állapot halmazhoz tartozik.*

Az automata most azokat a szavakat fogadja el, amiben végtelenül gyakran szerepel q_1 és q_2 is. Ezek a szavak az $(a^+b^+)^\omega$ alakúak, ahol a $^+$ tetszőlegesen véges sok, de legalább egy megjelenést jelenti. Ez a kifejezés lényegében azt írja le, hogy egyik betűből sem lehet végtelenül sok közvetlenül egymás után*, de az egész szóban mindkettőnek végtelenül sokszor kell szerepelnie.*

Egy általánosított Büchi-automata átalakítható egy egyszerű Büchi-automatává. Az $\mathcal{A} = \langle \Sigma, Q, \Delta, Q^0, F \rangle$ általánosított Büchi-automata „hagyományos” megfelelője $F = \{P_1, \ldots, P_n\}$ mellett $\mathcal{A}' = \langle \Sigma, Q \times \{0, \ldots, n\}, \Delta', Q^0 \times \{0\}, Q \times \{n\} \rangle$.

A Δ' állapotátmeneti reláció úgy épül fel, hogy $(\langle q, x \rangle, a, \langle q', y \rangle) \in \Delta'$ akkor és csak akkor áll fenn, ha $(q, a, q') \in \Delta$, x-re és y-ra pedig a következő szabályok érvényesek:

- Ha $q' \in P_i$ és $x = i - 1$, akkor $y = i$.
- Ha $x = n$, akkor $y = 0$.
- Minden egyéb esetben $y = x$.

Az elv nagyon hasonló a szorzat automaták elkészítésénél látottakhoz: a második komponens felelős azért, hogy minden elfogadó állapothalmazból végtelenül gyakran szerepeljenek állapotok. Akárcsak akkor, most is érdemes a kiinduló állapotokból nem elérhető állapotokat felderíteni és eltávolítani. Az átalakítás az automata méretét legfeljebb $n + 1$-szeresére növeli.

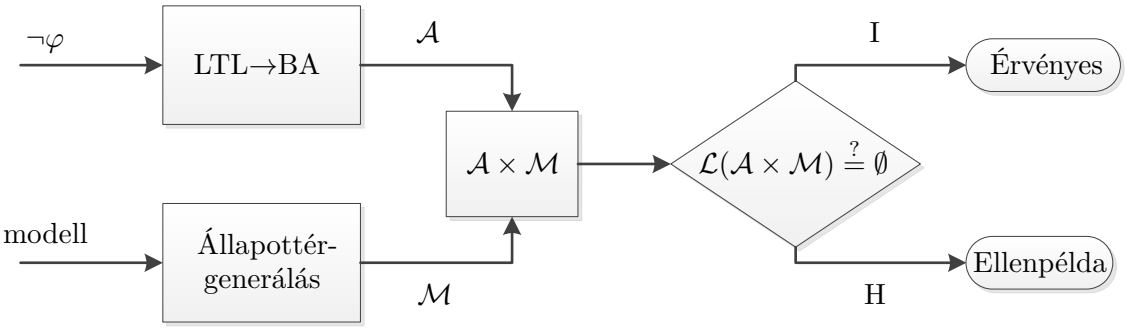
1.3. Modellellenőrzés

A modellellenőrzés egy elterjedt verifikációs technika, amely általában a rendszer egy véges modelljén vizsgálja meg, hogy a specifikációs tulajdonságok teljesülnek-e. Formálisan azt vizsgáljuk, hogy egy M modellre egy adott r specifikációs követelmény igaz-e [12]. Az r követelmény többféle, különböző kifejezőerejű formalizmussal megadható, amikkel eltérő típusú kifejezések értékelhetőek ki.

Mivel a munkám a lineáris temporális logikai specifikációk vizsgálatára irányult, ezért az alábbiakban az ezekhez kapcsolódó háttérismereteket mutatom be.

1.3.1. Automataelméleti megközelítés

Az LTL temporális logika könnyű használhatósága és intuitív jellege miatt többféle modellellenőrző algoritmust is kifejlesztettek már hozzá. Ezek jellemzően automataelméleti megközelítést alkalmaznak [14].



1.5. ábra. LTL modellellenőrzés automatákkal.

Az LTL modellellenőrzés automataelméleti megközelítés esetén a következő módon vázolható:

1. Adott a vizsgálandó kifejezés. Ennek képezzük a negáltját, és építünk egy *specifikációs automatát*, amelynek ábécéje a vizsgálandó modell állapotaiból (pontosabban az azokra érvényes állapotkifejezésekből) áll, és az elfogadott nyelve megegyezik az LTL kifejezést nem kielégítő állapotsorozatokkal (szavakkal).

2. A vizsgálandó modell állapotterét szintén automataként reprezentálva számítsuk ki a két automata szinkron szorzatát olyan módon, hogy a modell állapotváltozásaikor a célállapottal léptetjük a specifikációs automatát.[1]

3. Vizsgáljuk meg, hogy a *szorzat automata* által elfogadott nyelv üres-e. Ha üres, akkor a modell nem tartalmaz olyan lefutást, amely kielégítené a vizsgálandó kifejezés negáltját, tehát a kifejezés érvényes a modellre. Ha nem üres, az adódó nyelv szavai megfelelnek azoknak a lefutásoknak, amelyek megsértik a specifikációt, tehát értékes ellenpéldákhoz jutottunk.

A második lépésben az állapottér automatává alakítása technikailag történhet úgy, hogy az automata-reprezentációban a modell állapotait definiáljuk bemenetekként, és minden átmenetet a célállapottal címkézünk. Ekkor a két automata szótára megegyezik, és a korábban ismertetett módon képezhető a szorzat automata. A gyakorlatban célszerűbb lehet a szinkron szorzatot úgy képezni, hogy először a modell állapotterében lepünk, majd az elért állapotot bemenetként használva a specifikációs automatában is. Sőt, hogy a specifikációs automata független legyen az állapottér nagyságától, célszerű csak az állapotokra érvényes predikátumokat (atomi kijelentéseket) bemenetnek választani, és a léptetés előtt elvégezni a célállapot-predikátumhalmaz átalakítást. Így a specifikációs automata a modelltől függetlenül, csak a predikátumokon dolgozva egyszerű maradhat, a modell állapottere pedig a Büchi-automatától különböző formalizmusokban (pl. Kripke-struktúra) is megadható.

A különböző megvalósításoknak tehát a specifikációs automata építését, a szorzat képzésének módját, valamint a nyelv ürességének ellenőrzését kell definiálnia. Utóbbi probléma véges modellek esetén egy körkeresési feladat, ugyanis ekkor a szorzat állapottér is véges, ebben pedig csak úgy lehet elfogadó állapotot végtelen sokszor érintő lefutás, ha a lefutás „vége” egy elfogadó állapotot is tartalmazó hurok.

[1] Technikailag ez úgy definiálható, hogy az állapottér automata-reprezentációjában is a modell állapotai a bemenetek, tehát minden átmenetet a célállapottal címkézünk. Ekkor a két automata szótára megegyezik, és a korábban ismertetett módon képezhető a szorzat automata.

2. fejezet

Vacuity

Modellellenőrzési feladatok során kétféle eredményt kaphatunk. Az egyik lehetőség, hogy nem teljesül az általunk megadott specifikáció, ilyenkor az ellenőrzés során generált ellenpélda alapján ellenőrizhetjük a modellünket, javíthatjuk az esetleges hibákat. A másik lehetőség, hogy teljesül a specifikált tulajdonság, ilyenkor viszont nem kapunk semmilyen információt arról, hogy milyen módon teljesíti a modellünk a megadott követelményt[1]. A vacuity fogalmának bevezetésével teljesülő specifikációk esetén is hasznosabb visszajelzést adhatunk a felhasználónak[2].

2.1. Vacuity értékek

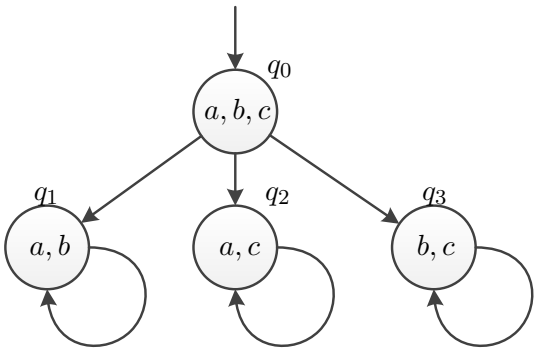
A továbbiakban az LTL kifejezéseket negált normál formájúra hozott kifejezéseknek tekintjük, a modelleket pedig Kripke-struktúrával adottnak vesszük. Formálisan, egy M Kripke-struktúra egy $\langle AP, Q, \Delta, Q_0, L\rangle$ ötös, ahol

- AP az atomi kijelentések halmaza
- Q a lehetséges *állapotok* véges halmaza.
- $\Delta \subseteq Q \times Q$ az *állapotátmeneti reláció.*
- $Q_0 \subseteq Q$ a *kiinduló állapotok* halmaza.
- $L : Q \to 2^{AP}$ a *címkézőfüggvény.*

Egy *(végtelen) útvonal* egy Kripke-struktúrán állapotok egy $\pi = q_0, q_1, \ldots$ végtelen sorozata, ahol minden $i \geq 0$ esetén a q_{i+1} állapot a q_i állapot utódja (azaz $(q_i, q_{i+1}) \in \Delta$).

Példa 4. *Tekintsük a 2.1 ábrán látható Kripke-struktúrát. A kezdőállapot a q_0, az állapotok az a, b és c atomi kijelentésekkel vannak címkézve. Egy végtelen útvonal például a $\pi = q_0, q_1, q_1, q_1 \ldots$ állapotsorozat.*

Definíció 1. Egy φ logikai formula esetén *literáloknak* nevezzük a formulában szereplő összes ponált és negált atomi kijelentést. ▪



2.1. ábra. Egy egyszerű Kripke-struktúra.

Példa 5. *A* $\varphi = \boldsymbol{G}(a \vee (\neg b \boldsymbol{U} \neg c))$ *LTL formula literáljainak halmaza* $\{a, \neg b, \neg c\}$.

Az általam használt vacuity fogalom az LTL formulákban megtalálható literálok előfordulásain értelmezett [17, 21]. Ehhez feltételezhetjük, hogy minden a formulában szereplő literál csak egyszer fordul elő. Amennyiben ez nem áll fenn, az egyes előfordulások átnevezésével a kifejezés ilyen alakra hozható. Ennek a feltételezésnek megfelelően a továbbiakban minden *literálokra* megfogalmazott állítás valójában *literál előfordulásokra* vonatkozik.

Jelölje $\varphi[l \leftarrow hamis]$ azt a formulát, melyet a φ formulából úgy kapunk, hogy az l literált *hamis* értékre cseréljük. Ezen kívül literálok S halmaza esetén legyen φ^S az a formula, melyben S összes elemét *hamis*ra cseréljük, azaz

$$\varphi^S = \varphi[l \leftarrow hamis \mid l \in S]$$

Definíció 2. *Egyszerű vacuity*-ről beszélünk, amennyiben egy φ formula úgy teljesül egy M modellen, hogy létezik egy l literál, amelyet φ-ben *hamis*ra cserélve a kapott formula még mindig teljesül a modellen, azaz

$$M \models_v \varphi \iff \exists l \in lit(\varphi), M \models \varphi[l \leftarrow hamis] \quad .$$

Definíció 3. *Kölcsönös vacuity*-ről beszélünk, amennyiben egy φ formula úgy teljesül egy M modellen, hogy létezik literálok S halmaza, melynek minden elemét φ-ben *hamis*ra cserélve kapott φ^S még mindig teljesül a modellen, azaz

$$M \models_v \varphi \iff \exists S, M \models \varphi^S \quad .$$

Példa 6. *Tekintsük ismét a 2.1 ábrán szereplő Kripke-struktúrát és a* $\varphi = \boldsymbol{G}(a \vee b \vee c)$ *LTL formulát. Ekkor a* φ *formula egyszerű vacuityvel teljesül a modellen, hiszen például* $\varphi[a \leftarrow hamis] = \boldsymbol{G}(hamis \vee b \vee c)$ *is minden útvonalon teljesül. Ha a modellből elhagynánk a* q_1 *állapotot, a* $\varphi' = \boldsymbol{G}(hamis \vee hamis \vee c)$ *formula is teljesülne minden útvonalon, így ekkor a* φ *formula kölcsönös vacuityvel teljesülne a modellen.*

A fenti definíciókat nem csak egy modellre értelmezhetjük, hanem a modellben szereplő egyes π útvonalakra is (hiszen egy ilyen útvonal önmagában is tekinthető egy modellnek).

Ezen túlmenően az útvonalakhoz vacuity értékeket rendelhetünk, melyek fogalmát aztán kiterjeszthetjük a teljes modellre is.

Definíció 4. Egy φ formula π útvonalon vett *vacuity értékének* nevezzük azon S_i literálhalmazok halmazát, melyekre a π útvonal esetén minden i-re teljesül, hogy $\pi \models \varphi^{S_i}$, azaz

$$vac(\pi, \varphi) = \{S \mid \pi \models \varphi^S\}$$

.

Definíció 5. Egy φ formula M modellen vett *vacuity értékének* nevezzük azon S_i literálhalmazok halmazát, melyekre az M modell minden lehetséges π útvonala esetén minden i-re teljesül, hogy $\pi \models \varphi^{S_i}$, azaz

$$vac(M, \varphi) = \{S \mid \pi \models \varphi^S, \forall \pi \in M\}$$

.

Megfigyelhetjük, hogy amennyiben egy S halmaz szerepel egy vacuity értékben, annak minden részhalmaza is szerepelni fog benne, ezért a továbbiakban vacuity értékeket úgy ábrázolunk, hogy $\{|\ |\}$ zárójelek között csak a maximális halmazokat tüntetjük fel.

Példa 7. *A korábbi példánál maradva a 2.1 ábrán látható modell $\pi = q_0, q_1, q_1, q_1 \ldots$ útvonalán a $\varphi = \boldsymbol{G}(a \vee b \vee c)$ formula akkor is teljesül, ha az $\{a, c\}$ literálokat, vagy a $\{b, c\}$ literálokat hamisra cseréljük. Ennek megfelelően a formula vacuity értéke az útvonalon*

$$vac(\pi, \varphi) = \{|\{a, c\}, \{b, c\}|\} = \{\{a, c\}, \{b, c\}, \{a\}, \{b\}, \{c\}, \emptyset\}.$$

Ha ezt a számítást az összes útvonalra elvégezzük, akkor azt kapjuk, hogy φ teljes modellre vett vacuity értéke

$$vac(M, \varphi) = \{|\{a\}, \{b\}, \{c\}|\}.$$

2.2. Vacuity rácsok

Definíció 6. *Rácsnak* nevezünk egy olyan részlegesen rendezett halmazt, aminek bármely két elemének van közös *infimuma* és *szuprémuma*. Egy rács elemein értelmezve vannak a szokásos rendezési relációk ($\sqsubseteq, \sqsupseteq$), illetve a *meet* ($\sqcap$) és *join* ($\sqcup$) műveletek.

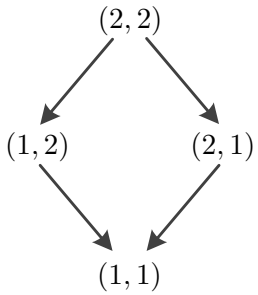
Amennyiben u és v egy rács elemei, $u \sqcap v$-vel jelöljük u és v *infimumát*, azaz a legnagyobb elemet a rácsban, amelyre igaz, hogy u-nál és v-nél is kisebb, vagy egyenlő.

Hasonlóan $u \sqcup v$-vel jelöljük u és v *szuprémumát*, azaz a legkisebb elemet a rácsban, amelyre igaz, hogy u-nál és v-nél is nagyobb, vagy egyenlő.

.

Egy rácsot ábrázolhatunk irányított gráfként, aminek a csúcsai a rács elemei, és az irányított élek jelzik a részleges rendezés szerinti rákövetkező elemeket.

Amennyiben u a rács egy eleme, a nála egyel nagyobb elemek halmazát nevezzük u szüleinek, és $\hat{u}$-val jelöljük. Grafikus ábrázolás esetén $\hat{u}$ azokat az elemeket jelenti, amikből u-ba mutat irányított él.



2.2. ábra. Egy egyszerű rács.

Példa 8. *Tekintsük a 2.2 ábrán látható számpárokból álló rácsot. Ekkor az elemeken értelmezett relációk és műveletek közül néhány példa:*

- $(1,2) \sqsubseteq (2,2)$
- $(1,2) \sqsubseteq (1,2)$ *és* $(1,2) \sqsupseteq (1,2)$
- $(1,2) \not\sqsubseteq (2,1)$ *és* $(1,2) \not\sqsupseteq (2,1)$
- $(1,2) \sqcap (2,1) = (1,1)$
- $(1,2) \sqcap (1,1) = (1,2)$
- $(1,2) \sqcup (2,1) = (2,2)$
- $(1,1) \sqcup (1,1) = (1,1)$
- $u = (1,1)$ *esetén* $\hat{u} = \{(1,2),(2,1)\}$

Amennyiben P literálok halmaza, $V(P)$-vel jelöljük a P halmaz elemeiből előállítható összes vacuity érték halmazát. Ez a halmaz a részhalmazrendezésre nézve egy rácsot alkot, így a továbbiakban *vacuity rácsként* fogok hivatkozni rá.

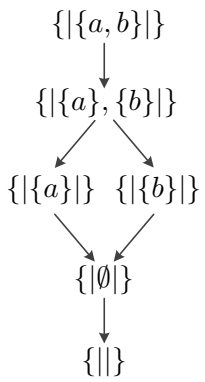
Ha egy vacuity rács u és v eleme között egy (u,v) irányított él található, az azt jelenti, hogy v kisebb, mint u, illetve a v vacuity értéket úgy kapjuk, hogy az u értékből elhagyjuk az egyik maximális elemet. Ennek megfelelően a rács egy u eleme pontosan akkor kisebb a rács egy v eleménél ($u \sqsubseteq v$), ha u elérhető irányított úton v-ből. Amennyiben két elem között nem létezik irányított út, azok nem összehasonlíthatóak.

A *meet* művelet egy vacuity rács esetében megegyezik az elemeken alkalmazott hagyományos metszet halmazművelettel, míg a *join* művelet a hagyományos unió műveletet jelenti.

Ezeken túl vacuity rácsokon a *negálás* műveletét a P alaphalmaz mellett az alábbi módon definiáljuk.

$$\sim v = 2^P \setminus \{P \setminus x \mid x \in v\}$$

Ez azt jelenti, hogy egy vacuity rácsban egy v vacuity érték negáltját úgy kapjuk meg, hogy az alaphalmaz összes részhalmaza közül elhagyjuk azokat a halmazokat, amik v vacuity értékben szereplő halmazok komplementerei.



2.3. ábra. Egy egyszerű vacuity rács.

Példa 9. *Tekintsük a 2.3 ábrán látható vacuity rácsot. A negálás alkalmazása a* $P = \{a, b\}$ *alaphalmaz mellett az* $\{|\{a\}, \{b\}|\}$ *vacuity értékre az alábbi módon történik.*

$$\sim \{|\{a\}, \{b\}|\} = 2^{\{a,b\}} \setminus \{\{a\}, \{b\}, \{a, b\}\} = \{|\emptyset|\}$$

Azaz alaphalmaz hatványhalmazából kivonjuk a vacuity értékünkben szereplő $\{a\}$*,* $\{b\}$ *és* $\emptyset$ *halmazok komplementereit.*

Definíció 7. *De Morgan-rácsnak* nevezünk egy rácsot, amennyiben a meet ($\sqcap$) és join ($\sqcup$) műveletek disztributívak egymásra nézve ($u \sqcap (v \sqcup w) = (u \sqcap v) \sqcup (u \sqcap w)$), egy elem negáltjának negáltja önmaga ($\sim\sim\ v = v$) és teljesülnek a De Morgan-azonosságok ($\sim (u \sqcup v) = (\sim u) \sqcap (\sim v)$). ▪

Megfigyelhetjük, hogy a vacuity rácsok zártak a negálás műveletére és teljesülnek rájuk a fenti feltételek, így minden vacuity rács egyben De Morgan-rács is. Erre a tulajdonságra hamarosan szükség lesz a 2.3.1. fejezetben, az LTL-formalizmus általánosításához.

2.3. Vacuity automaták

Ahogy arról már volt szó, egy vacuity értéket alapvetően egy π útvonalhoz és egy φ LTL formulához kapcsolódóan értelmezünk. Ebben a fejezetben azt mutatom be, hogy hogyan lehet egy adott útvonal és egy LTL formula esetén meghatározni a hozzájuk kapcsolódó vacuity értéket. Amikor LTL formulák értékeit szeretnénk kiszámolni egy adott útvonalon, tipikusan Büchi-automatákat használunk, amik egy logikai értéket rendelnek az adott lefutáshoz. Ehhez hasonlóan létrehozhatunk olyan vacuity automatákat, amelyek egy lefutáshoz nem egy logikai értéket, hanem az eredeti formula vacuity értékét rendelik. Ahhoz, hogy ilyen automatákat létre tudjunk hozni, először bemutatom az LTL és a Büchi-automaták rácsokhoz kapcsolódó kiterjesztéseit, azaz az LLTL-t (latticed LTL) és az úgynevezett rácsautomatákat.

2.3.1. Latticed LTL

A *latticed LTL (LLTL)* [20] a hagyományos lineáris temporális logika kiegészítése oly módon, hogy az a korábban is megengedett atomi kijelentések (AP) mellett egy $\mathcal{L}$ rács elemeit is tartalmazhatja. Az ilyen LLTL formulák nem megszokott logikai (igaz - hamis) értékeket vehetnek fel, hanem a hozzá kapcsolódó rács elemei közül kaphatnak értéket.

Definíció 8. Legyen $\mathcal{L}$ egy De Morgan rács. Ekkor egy φ LLTL-kifejezés értékét egy $\pi = w_1, w_2, \ldots$ útvonalon az i-edik pozícióban $val(\pi^i, \varphi)$-vel jelöljük, és az alábbi rekurzív módon definiáljuk.

- Ha $v \in \mathcal{L}$, akkor $val(\pi^i, v) = v$
- Ha $p \in AP$ egy atomi kijelentés, akkor $val(\pi^i, p) = \top$, ha $p \in w_i$, és $val(\pi^i, p) = \bot$, ha $p \notin w_i$
- $val(\pi^i, \neg\varphi) = \sim val(\pi^i, \varphi)$
- $val(\pi^i, \varphi_1 \wedge \varphi_2) = val(\pi^i, \varphi_1) \sqcap val(\pi^i, \varphi_2)$
- $val(\pi^i, \mathbf{X}(\varphi)) = val(\pi^{i+1}, \varphi)$
- $val(\pi^i, \varphi_1 \ \mathbf{U} \ \varphi_2) = \bigsqcup_{k \geq i} (val(\pi^k, \varphi_2) \sqcap \bigsqcap_{i \leq j < k} val(\pi^j, \varphi_1))$
- $val(\pi^i, \mathbf{G}(\varphi)) = \bigsqcap_{k \geq i} val(\pi^k, \varphi)$
- $val(\pi^i, \mathbf{F}(\varphi)) = \bigsqcup_{k \geq i} val(\pi^k, \varphi)$ ▪

Az így definiált LLTL formalizmus magában foglalja a hagyományos lineáris temporális logikát, ugyanis az a speciális eset, amikor az $\mathcal{L}$ rács csak két elemből áll ($\top$ és $\bot$) megegyezik a hagyományos LTL formalizmussal.

Ahogy azt [6]-ban megmutatták, minden φ LTL formulához létezik olyan φ_v LLTL formula a φ literáljaiból létrehozott vacuity rács felett, hogy annak minden π útvonalon a kezdőállapotban vett értéke megegyezik az eredeti φ kifejezés π-n vett vacuity értékével, azaz

$$val(\pi^0, \varphi_v) = vac(\pi, \varphi)$$

Egy ilyen LLTL formula az alábbi módon definiált kifejezés, ahol $NV(l)$ az a legnagyobb vacuity érték, amelynek egyik elemében sem szerepel az l literál.

$$\varphi_v = \varphi[l \leftarrow (l \wedge NV(l)) \mid l \in lit(\varphi)]$$

Ahhoz, hogy ezt hasznosítani tudjuk, létre kell hoznunk olyan automatákat, melyek meg tudják határozni egy LLTL formula értékét egy útvonalon ahhoz hasonlóan, ahogy a Büchi-automatákkal képesek vagyunk a nekik megfelelő LTL kifejezések által felvett értékek meghatározására.

2.3.2. Rácsautomaták

A rácsautomaták[5] a véges állapotú automaták kiterjesztései oly módon, hogy az átmenetek címkézhetőek egy *De Morgan-rács* elemeivel[20]. A hagyományos automatákhoz hasonlóan egy rácsautomata is működhet véges és végtelen szavakon is.

Formálisan, egy végtelen szavakon működő $\mathcal{A}$ rácsautomata egy $\langle \mathcal{L}, \Sigma, Q, \delta, Q^0, F \rangle$ hatos, ahol

- $\mathcal{L}$ a De Morgan *rács*.
- Σ a véges *ábécé*.
- Q a lehetséges *állapotok* véges halmaza.
- $\delta : Q \times \Sigma \times Q \mapsto L$ az *állapotátmeneti függvény*, ami az átmeneteket képezi le a rács elemeire.
- $Q^0 \subseteq Q$ a *kiinduló állapotok* halmaza.
- $F \subseteq Q$ az *elfogadó állapotok* halmaza.

Az $\mathcal{A}$ rácsautomata és a $w = \sigma_1, \sigma_2, \ldots$ végtelen szó esetén egy $r = q_1, q_1, \ldots$ lefutás értéke

$$val(r, w) = \bigsqcap_{i \geq 0} \delta(q_i, \sigma_{i+1}, q_{i+1})$$

A hagyományos Büchi-automatákhoz hasonlóan egy lefutás pontosan akkor elfogadó, ha végtelenül gyakran áthalad elfogadó állapotokon, azaz

$$acc(r) = \bigsqcap_{i \geq 0} \bigsqcup_{j \geq i} (q_j \in F)$$

Egy w végtelen szó *elfogadási értéke* az $\mathcal{A}$ rácsautomatán az összes lehetséges elfogadó lefutás értékén végrehajtott join művelet eredménye.

$$\mathcal{A}(w) = \bigsqcup \{ val(r, w) \mid acc(r) \}$$

A továbbiakban szükség lesz a rácsautomaták *ürességi értékére*, melyet az alábbi módon definiálunk.

$$e\_val(\mathcal{A}) = \bigsqcup \{ \mathcal{A}(w) \mid w \in \Sigma^\omega \}$$

Amennyiben egy automata ürességi értéke $e\_val(\mathcal{A}) = \bot$, az azt jelenti, hogy az automata által elfogadott nyelv üres. Egy automata ürességi értékének meghatározását nevezzük *ürességi problémának*.

Definíció 9. Azt mondjuk, hogy egy $\mathcal{A}_{\varphi_v}$ rácsautomata a φ LTL formula *vacuity automatája*, ha minden π útvonalat a formula vacuity értékére képez le, azaz

$$\mathcal{A}_{\varphi_v}(\pi) = vac(\pi, \varphi)$$

.

Minden φ LTL kifejezéshez készíthető ilyen $\mathcal{A}_{\varphi_v}$ automata, amit a korábban ismertetett φ_v LLTL formula alapján hozhatunk létre, ahogy azt [20]-ban megmutatták.

Amennyiben $\mathcal{A}_{\varphi_v}$ a φ LTL kifejezéshez tartozó vacuity automata és $\mathcal{A}_\pi$ egy olyan hagyományos automata, ami pontosan a π útvonalat fogadja el, akkor a $\mathcal{A}_{\varphi_v} \times \mathcal{A}_\pi$ szinkron szorzatként előálló automata minden π útvonaltól különböző w bemenethez a $\perp$ értéket fogja rendelni, a $w = \pi$ bemenethez pedig a φ formula π-n vett vacuity értékét. Ennek megfelelően

$$e\_val(\mathcal{A}_{\varphi_v} \times \mathcal{A}_\pi) = \mathcal{A}_{\varphi_v}(\pi) = vac(\pi, \varphi),$$

azaz egy útvonal vacuity értékének kiszámolását visszavezettük egy olyan lattice automata üresség értékének meghatározására, aminek csak egyetlen elfogadó lefutása létezik. Megemlítendő, hogy egy automata üresség értékének kiszámítása felfogható egy többértékű (pl. rács alapú) modellellenőrzési problémaként [4, 16], de ennek tárgyalása túlmutat dolgozatom témakörén.

3. fejezet

A legerősebb teljesülő formula

Könnyen belátható, hogy a kölcsönös vacuity önmagában történő használata, azaz egyes literálhalmazok *hamis*ra cserélése még nem eredményezi a létrehozható legerősebb teljesülő formulát, már csak azért sem, mert egy ilyen vacuity értékben több maximális *hamis*ra cserélhető literálhalmaz is szerepelhet, amiket egyszerre nem lehet lecserélni. Másrészt azért sem, mert egy kiszámolt vacuity érték általános esetben a modell egyetlen útvonalára vonatkozik. Ebben a fejezetben bemutatom, hogy milyen módon lehet egy a modellellenőrzés során teljesülő φ LTL formulához vacuity értékek használatával létrehozni egy olyan teljesülő formulát, ami megtartja az eredeti kifejezés szerkezetét, viszont sokkal pontosabban írja le a modell viselkedését.

Ahhoz hogy ezt megtehessem, definiálni kell a φ formula P literáljaiból létrehozható $V(P)$ vacuity rács bizonyos részhalmazait.

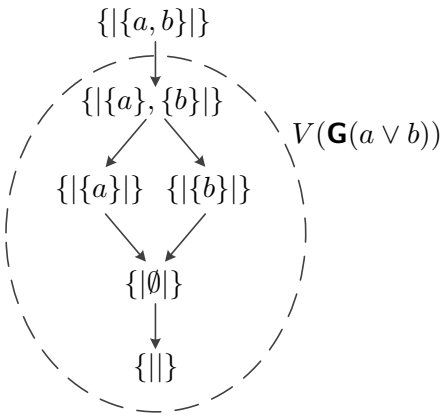
A továbbiakban jelöljük $V(\varphi)$-vel azon vacuity értékek halmazát, amelyek szerepelnek a $V(P)$ rácsban (azaz a φ kifejezés literáljaiból épített vacuity rácsban), és nem teszik kielégíthetetlenné a φ formulát. Másképpen fogalmazva ez azon vacuity értékek halmaza, melyekhez létezik olyan π útvonal, hogy $vac(\pi, \varphi) = v$. Formálisan

$$V(\varphi) = \{v \in V(P) \mid \exists \pi, vac(\pi, \varphi) = v\}$$

Példa 10. *Tekintsük a $\varphi = \mathbf{G}(a \vee b)$ kifejezést és az ehhez tartozó $V(\{a, b\})$ vacuity rácsot (lásd 3.1. ábra). Ekkor a rács $V(\varphi)$ részhalmazát úgy kapjuk, hogy a teljes rácsból elhagyjuk az $\{|\{a, b\}|\}$ vacuity értéket, ugyanis az ez alapján létrehozott $\varphi' = \mathbf{G}(hamis \vee hamis)$ formula semmilyen úton nem lehet kielégíthető. Mivel a többi vacuity érték esetén elképzelhető olyan útvonal, ami mellett még kielégíthető a létrehozott formula, ezért ezek elemei a $V(\varphi)$ halmaznak is.*

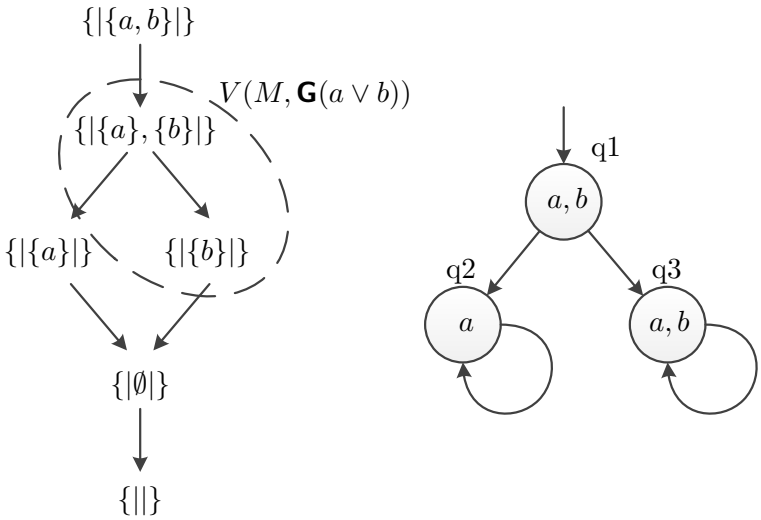
Amennyiben a φ kifejezésen kívül adott egy M modell is, hasonlóképpen definiálhatjuk a $V(M, \varphi)$ halmazt, ami a $V(\varphi)$ halmaz további szűkítése az M modell szerint. Ez azt jelenti, hogy csak olyan vacuity értékeket tartalmaz, amihez az M modellben mutatható olyan π útvonal, hogy $vac(\pi, \varphi) = v$. Formálisan

$$V(M, \varphi) = \{v \in V(\varphi) \mid \exists \pi \in M, vac(\pi, \varphi) = v\}$$



3.1. ábra. A $\varphi = \mathbf{G}(a \vee b)$ formulához tartozó $V(\{a, b\})$ vacuity rács és annak $V(\mathbf{G}(a \vee b))$ részhalmaza.

Példa 11. *Tekintsük ismét a $\varphi = \mathbf{G}(a \vee b)$ kifejezést, az ehhez tartozó $V(\{a, b\})$ vacuity rácsot és egy M modellt (lásd 3.2. ábra). Az ábrán látható modellen csupán két végtelen lefutás elképzelhető, amik közül a $\pi_1 = q_1, q_2, q_2, \ldots$ útvonalhoz a $vac(\pi_1, \varphi) = \{|\{b\}|\}$, míg a $\pi_2 = q_1, q_3, q_3, \ldots$ útvonalhoz a $vac(\pi_2, \varphi) = \{|\{a\}, \{b\}|\}$ vacuity érték tartozik. Mivel a modell nem tartalmaz több végtelen útvonalat, ezért a vacuity rács $V(M, \varphi)$ részhalmaza pontosan ebből a két értékből áll.*



3.2. ábra. A $V(\{a, b\})$ vacuity rács, egy M modell és az ezekhez tartozó $V(M, \mathbf{G}(a \vee b))$ részhalmaz.

A vacuity rács általunk használt legszűkebb részhalmaza a $V_{min}(M, \varphi)$ halmaz, ami a $V(M, \varphi)$ halmazból a minimális elemeket tartalmazza, azaz

$$V_{min}(M, \varphi) = \{v \in V(M, \varphi) \mid \nexists \pi \in M, vac(\pi, \varphi) \sqsubset v\}$$

Példa 12. *Az előző példánál maradva a $\varphi = \mathbf{G}(a \vee b)$ formulához és a a 3.2. ábrán látható M modellhez tartozó $V_{min}(M, \varphi)$ halmaz csupán a $\{|\{b\}|\}$ vacuity értékből áll, ugyanis a $V(M, \varphi)$-ben szereplő másik $\{|\{a\}, \{b\}|\}$ érték ennél nagyobb, így az nem lehet eleme a minimális értékeket tartalmazó halmaznak.*

Megfigyelhetjük, hogy ez a halmaz a definíciójából adódóan csak olyan elemeket tartalmazhat, amik nem összehasonlíthatóak egymással, hiszen ha azok lennének, akkor a nagyobbik elem nem lehetne tagja a halmaznak. Ebből következik, hogy a $V_{min}(M, \varphi)$ halmaz (egynél több elem esetén) nem alkot rácsot, mert bármely két elemhez nem tudunk azoknál nagyobb (vagy kisebb) elemet mutatni. Az eredeti vacuity rács korábban definiált részhalmazai ugyan továbbra is alkothatnak rácsot, de általános esetben ez nem feltétlenül van így.

3.1. A keresett formula

Arra a problémára, hogy egy φ LTL formula esetén egy π útvonalhoz tartozó v vacuity értékben több maximális literálhalmaz is szerepelhet, megoldást jelent, ha tartalmazott halmazonként lecseréljük a literálokat *hamis*ra, majd az így kapott kifejezéseknek vesszük a konjunkcióját.

$$C_\varphi(v) = \bigwedge_{S \in v} \varphi^S$$

Mivel az így kapott konjunkcióban a vacuity érték minden halmaza szerepel, így az azok alapján külön-külön létrehozható kifejezések mindegyiknél szigorúbb megkötéseket fogalmaz meg. További optimalizációt jelent, ha a C_φ formula készítésekor csak a v vacuity értékben szereplő maximális literálhalmazokat (azaz a vacuity érték jelölésében is szereplő halmazokat) vesszük figyelembe az összes halmaz helyett.

A második problémára, miszerint az így létrehozott C_φ formula csak az M modell egyetlen útvonalát írja le, megoldást jelent az ilyen formulák diszjunkciója a modell minden, egymástól különböző vacuity értékkel rendelkező útvonalára nézve.

$$\Phi(M, \varphi) = \bigvee_{v \in V(M,\varphi)} C_\varphi(v)$$

Ez a formula két módon is javítja a hagyományos kölcsönös vacuity detektálás eredményeit, hiszen egyrészt részletesebb és erősebb formulákkal ír le egy-egy útvonalat, másrészt megkülönbözteti a modellben található útvonalakat egymástól.

A $\Phi(M, \varphi)$ formula viszont még mindig tartalmazhat felesleges részeket, hiszen a modellben szereplő két különböző útvonalhoz tartozhatnak olyan v_1 és v_2 vacuity értékek, hogy $v_1 \sqsubset v_2$, ami azt jelenti, hogy a belőlük létrehozott C_φ formulák közül $C_\varphi(v_2)$ elhagyható. Ez azért van így, mert a nagyobb v_2 vacuity érték szigorúbb C_φ formulát eredményez, mint a kisebb v_1, így a teljes modellre vonatkozó $\Phi(M, \varphi)$ kifejezésben a diszjunkció miatt ez feleslegessé válik.

Az általunk keresett formulát tehát úgy kaphatjuk meg, hogy a $\Phi(M, \varphi)$ konstruálásakor nem az összes lehetséges útvonalhoz tartozó vacuity értéket használjuk fel, hanem csak a

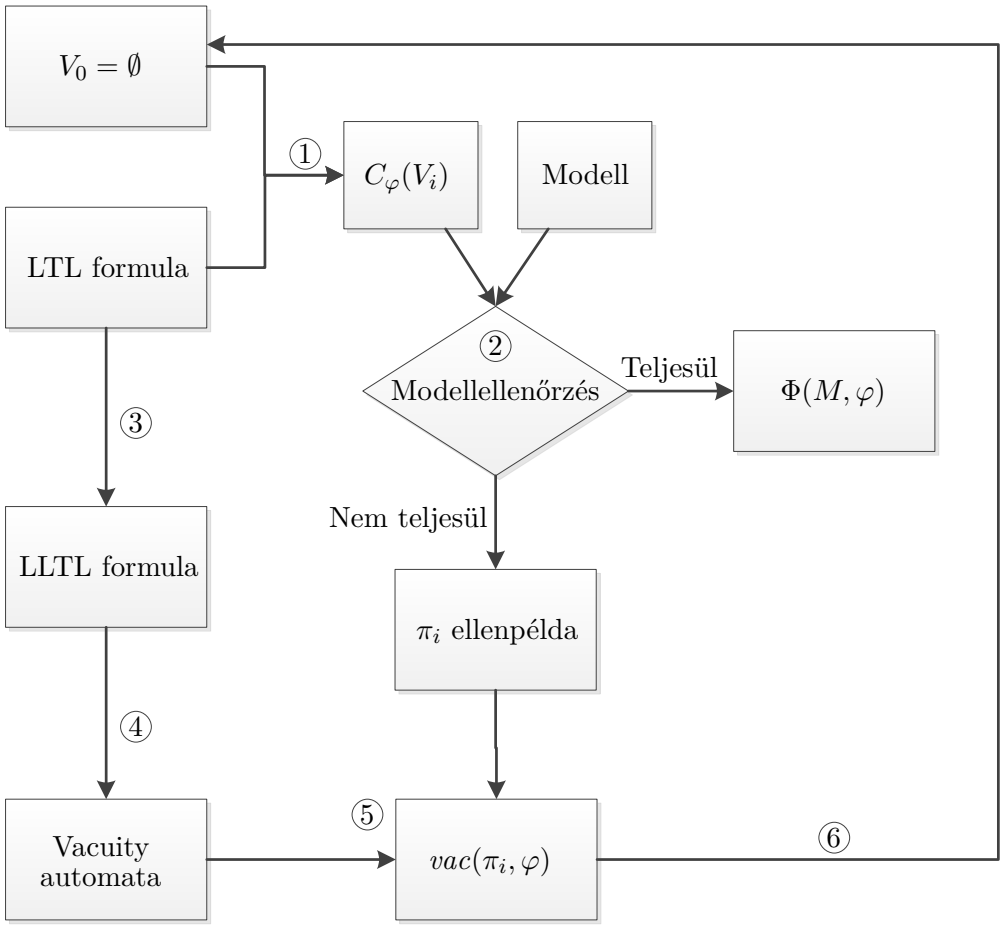
korábban definiált, $V_{min}(M, \varphi)$-ben szereplő minimális értékeket. Formálisan

$$\Phi_{min}(M, \varphi) = \bigvee_{v \in V_{min}(M,\varphi)} C_{\varphi}(v)$$

Ahogy azt [6]-ban megmutatták, ez a $\Phi_{min}(M, \varphi)$ a legerősebb formula, ami még kielégíti az M modellt és a φ LTL formula olyan erősebb verzióinak pozitív lezártjába tartozik, amiket literálok *hamis* értékre cserélésével kaphatunk.

3.2. A formula meghatározása

Ahhoz, hogy meg tudjuk határozni a $\Phi_{min}(M, \varphi)$ formulát, csak arra van szükségünk, hogy ismerjük a $V_{min}(M, \varphi)$ halmazt, azaz a modell útvonalaihoz tartozó vacuity értékek közül az összes minimálist. A szükséges vacuity értékek megkeresésére a 3.3. ábrán látható algoritmust használhatjuk, melynek lényege, hogy olyan útvonalakat keresünk a modellben, amik vacuity értékeinél kisebb értéket még nem tároltuk el a V_i halmazban, majd ez alapján frissítjük az eddig megtalált értékek halmazát.



3.3. ábra. A legerősebb teljesülő formula meghatározása.

Az algoritmus az ① lépésében az eddig megtalált vacuity értékek V_i halmazából és a felhasználó által megadott specifikációs kifejezésből létrehozza a 3.1. fejezetben leírtak

szerint az ezek alapján megalkotható legerősebb formulát, amit itt $C_\varphi(V_i)$-vel jelölünk. Formálisan

$$C_\varphi(V) = \bigvee_{v \in V_i} \bigwedge_{S \in v} (\varphi^S)$$

A ② lépésben ezzel a kifejezéssel indít egy modellellenőrzést, ami csak akkor teljesülhet, ha már minden keresett vacuity értéket megtalált az algoritmus. Amennyiben még létezik olyan π útvonal a modellben, aminek $vac(\pi, \varphi)$ vacuty értékét még nem adta hozzá az aktuális V_i halmazhoz, az azt jelenti, hogy a vizsgált $C_\varphi(V_i)$ kifejezés nem lesz érvényes a modellen, ugyanis az ehhez a π útvonalhoz tartozó viselkedést ez a formula még nem engedi meg. Nem teljesülő modellellenőrzéskor így megkapunk ellenpéldaként egy olyan π_i útvonalat, aminek a vacuity értékét fel kell venni a V_{i+1} halmazba. Ahhoz, hogy ezt meg lehessen tenni, a ③ lépésben létre kell hozni a 2.3.1. fejezetben definiált φ_v LLTL formulát, amire teljesül, hogy $val(\pi^0, \varphi_v) = vac(\pi, \varphi)$. A ④ lépésben ez alapján létre kell hozni az eredeti φ kifejezéshez tartozó $\mathcal{A}_{\varphi_v}$ vacuity automatát, aminek segítségével az ⑤ lépésben már meghatározhatjuk a modellellenőrzés során ellenpéldaként kapott π_i útvonal vacuity értékét. A ⑥ lépésben végül a megtalált vacuity értéket hozzávesszük a V_i halmazhoz, ezzel megkapva a V_{i+1} halmazt. Az algoritmus addig ismétli az ①-⑥ lépéseket, amíg a ② lépésben nem találunk egy teljesülő formulát (természetesen a ③-④ lépésekben létrehozott LLTL formulát és vacuity automatát nem kell újra elkészíteni). Amennyiben ilyen formulát találunk, az azt jelenti, hogy a modell már nem tartalmaz olyan útvonalat, aminek a vacuity értéke még szükséges számunkra, tehát megtaláltuk a keresett $\Phi(M, \varphi)$ kifejezést. Amennyiben az algoritmus futásának végén az utolsó V halmazból a nem minimális vacuity értékek elhagyásával előállítjuk a V_{min} halmazt, ebből megkaphatjuk a ténylegesen legerősebb teljesülő formulát, azaz $\Phi_{min}(M, \varphi)$-t.

Az algoritmus futási idejét tovább javíthatjuk, amennyiben minden iterációban elhagyjuk a V_i halmaz nem minimális elemeit, ugyanis a kevesebb vacuity érték alapján létrehozott $C_\varphi(V_i)$ formulák mérete is kisebb lesz, ezzel növelve a modellellenőrzések sebességét. Ezzel a módszerrel az algoritmus végén rögtön a V_{min} halmazt és a legerősebb teljesülő formulát kapjuk.

Amennyiben az egyes útvonalak vacuity értékének meghatározásához szükséges vacuity automatát adottnak vesszük, az algoritmus működése a következőképpen alakul:

Algorithm 1 Algoritmus a legerősebb teljesülő formula meghatározására

```
1: function FindStrongestFormula(M, φ)
2:     V ← ∅;
3:     while M ⊭ C_φ(V) do
4:         π ← CEX(M, C_φ(V))        ▷ A megtalált ellenpélda (CEX - Counterexample)
5:         v ← vac(π, φ)                          ▷ A vacuity automata segítségével
6:         V ← min(V ∪ {v})
7:     end while
8:     return C_φ(V);
9: end function
```

3.3. Módosított algoritmus

Az általam elkészített implementációban nem a fenti algoritmust, hanem annak egy egyszerűsített változatát valósítottam meg. A módosítás lényege, hogy egyszerűbb matematikai struktúrák használatával keresi meg a modellben szereplő vacuity értékeket, ezzel megváltoztatva a szükséges algoritmikus lépéseket is, amik jelentős befolyással lehetnek a végrehajtási időre.

Az egyszerűsített algoritmus szintén modellellenőrzések sorozatával keresi meg a szükséges vacuity értékeket, viszont nincs garancia arra, hogy minden modellellenőrzés futtatás eredményeként sikerül bővíteni a megtalált értékek halmazát, ezért több modellellenőrzést kell indítani, mint az eredeti algoritmus esetében.

Az algoritmus (lásd 3.4. ábra) az ① lépésben a megadott LTL-formula alapján felépíti az ehhez tartozó vacuity rácsot, majd a ② lépésben a rács felhasználásával kiválasztja az ellenőrizendő vacuity értékeket. Az első ilyen kiválasztott érték a rács legkisebb eleme ($\bot$). Ezek után a ③ lépésként a kiválasztott vacuity értékek közül egyhez (v) létrehozza a neki megfelelő módosított $C(v)$ kifejezést, majd ezzel végez modellellenőrzést a ④ lépésben. A korábbiaktól eltérően itt azonban nem a formula érvényességét ellenőrizzük a modellen, hanem annak kielégíthetőségét, azaz az eredményből az derül ki, hogy létezik-e olyan útvonal a modellben, amin az adott vacuity érték szerint átalakított formula teljesül. Amennyiben a v vacuity értékhez tartozó módosított kifejezés kielégíthető, az ⑤ lépésben egyrészt eltároljuk ezt az értéket, másrészt a továbbiakban ellenőrizendő vacuity értékek közé felvesszük (a vacuity rács alapján) a v-nél egyel nagyobb értékeket (azaz azon a vacuity értékek $\hat{v}$ halmazát, amikből a rácsban él mutat az aktuális v vacuity értékre).

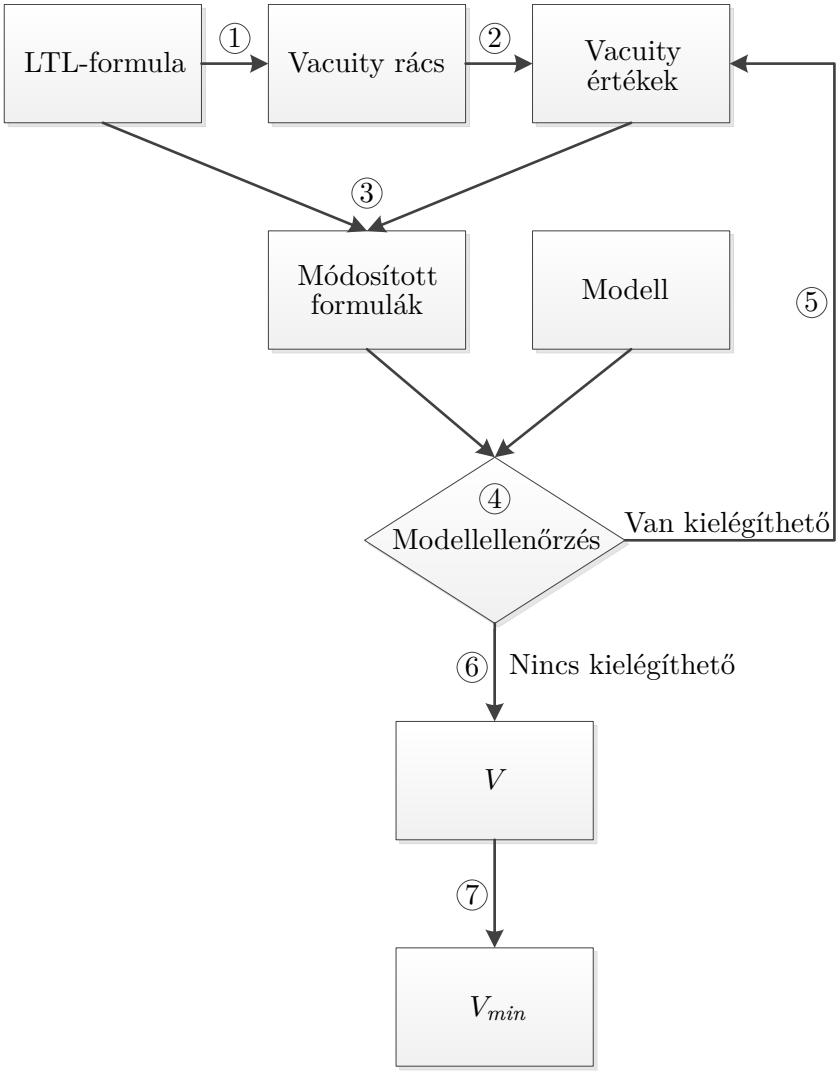
Amíg van ellenőrizendő vacuity érték, addig az algoritmus a ③-⑤ lépéseket ismétli. Amennyiben elfogytak a kiválasztott ellenőrizendő vacuity értékek, a megtalált értékek V halmaza az eredeti algoritmushoz hasonlóan tartalmazza az összes olyan v vacuity értéket, amelyekhez létezik egy olyan π útvonal a modellben, hogy $vac(\pi, \varphi) = v$, viszont ezen kívül tartalmazza az összes ezeknél kisebb vacuity értéket is. Ahhoz, hogy elő tudjuk állítani a legerősebb teljesülő formulát, a ⑥ lépésben az ilyen felesleges értékeket először el kell távolítani a V halmazból. Ezt modellellenőrzések újabb sorozatával tudjuk megtenni oly módon, hogy az alábbi kifejezés kielégíthetőségét vizsgáljuk:

$$\varphi'_v = C(v) \bigwedge_{v' \in \hat{v}} \neg C(v')$$

Amennyiben egy ilyen φ'_v formula kielégíthető, az azt jelenti, hogy létezik a modellben olyan útvonal, amelyen a v szerint átalakított $C(v)$ kifejezés teljesül, de a nála eggyel nagyobb $v' \in \hat{v}$ értékek alapján készített $C(v')$ kifejezések már nem teljesülnek. Más szóval a kielégíthető φ'_v formulák megkeresésével előállíthatjuk azt a V' halmazt, amiben már csak azok a v vacuity értékek szerepelnek, amikhez ténylegesen létezik a modellben olyan π útvonal, hogy $vac(\pi, \varphi) = v$. Ezzel tehát létrehoztuk ugyanazt a halmazt, ami az eredeti algoritmus eredménye volt. Amennyiben ebből még kiszűrjük a nem minimális elemeket, azzal megkapjuk a V_{min} halmazt, ami alapján elő tudjuk állítani a legerősebb teljesülő

$\Phi_{min}(M, \varphi)$ formulát a 3.1 fejezetben bemutatott módon az alábbi képlet alapján:

$$\Phi_{min}(M, \varphi) = \bigvee_{v \in V_{min}} \bigwedge_{S \in v} \varphi^S$$



3.4. ábra. Módosított algoritmus a legerősebb teljesülő formula meghatározására.

3.3.1. Kifejezések kezelése

Az előző fejezetekben bemutatott algoritmusok során végig azt feltételeztük, hogy az ellenőrizendő tulajdonságokat specifikáló LTL-kifejezésekben egyrészt minden atomi kifejezés egyedi névvel rendelkezik, másrészt hogy a kifejezések negált normál formában adottak, azonban ez a gyakorlatban nem feltétlen teljesül. Az algoritmus futása során ezért az egyes lépésekhez mindig a megfelelő alakra kell hozni a formulát, melynek így a végrehajtás alatt több reprezentációját használjuk (lásd 3.5. ábra).

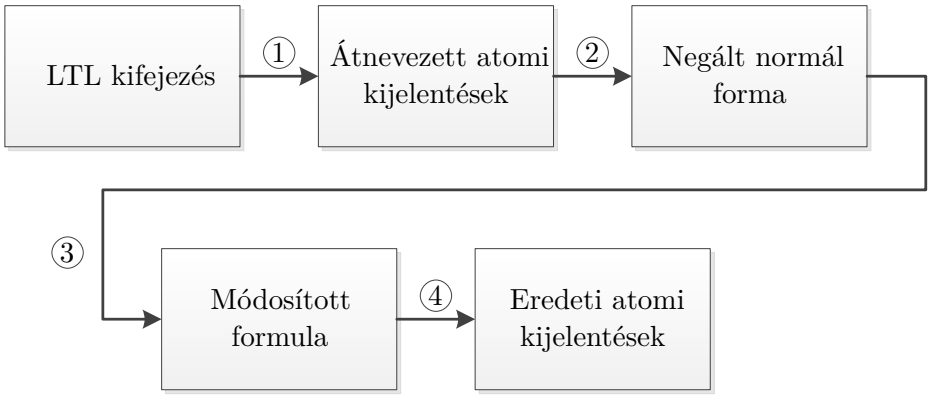
Mivel az algoritmusok literál előfordulásokkal dolgoznak, fontos, hogy a kifejezésben szereplő azonos atomi kijelentéseket meg tudjuk különböztetni egymástól. Ehhez az ①

Algorithm 2 Módosított algoritmus a legerősebb teljesülő formula meghatározására

```
1: function FindStrongestFormulaModified(M, φ)
2:     V ← ∅;
3:     VacuitiesToCheck ← {⊥}
4:     while VacuitiesToCheck ≠ ∅ do
5:         for all v ∈ VacuitiesToCheck do
6:             if sat(M, C(v)) then
7:                 V ← V ∪ {v}
8:                 NextVTC ← NextVTC ∪ v̂
9:             end if
10:        end for
11:        VacuitiesToCheck ← NextVTC
12:    end while
13:    V′ ← ∅
14:    for all v ∈ V do
15:        if sat(M, φ′_v) then
16:            V′ ← V′ ∪ {v}
17:        end if
18:    end for
19:    V_min ← min(V′)
20:    return V_min;
21: end function
```

lépésben át kell nevezni az atomi kijelentéseket, és az így kapott kifejezéssel kell dolgozni a továbbiakban. Azonban a modellellenőrzésekhez használt kifejezésekben és a felhasználónak eredményül adott legerősebb teljesülő kifejezésben az eredeti neveket kell használnunk, ezért el kell tárolni egy szótárt, ami alapján vissza tudjuk állítani azokat.

Mivel a felhasználó a specifikációs kifejezés elkészítésekor várhatóan az átláthatóságot, értelmezhetőséget tartja szem előtt, ezért az ritkán lesz negált normál formájú. Hogy az algoritmus helyes működését biztosítani tudjuk, a formulát negált normál formára hozzuk a ② lépésben. Megtehetnénk, hogy a továbbiakban csak ezt a formát használjuk fel a legerősebb teljesülő formula megkereséséhez, viszont mivel a legerősebb formula követi annak a kifejezésnek alakját, amiből létrehozzák, ezzel elveszítené az eredeti formula könnyű értelmezhetőségét. Ahhoz, hogy megtarthassuk az eredeti formula alakját, a negált nor-



3.5. ábra. A specifikációs kifejezés különböző reprezentációi.

mál formájú kifejezés használatával előbb megkeressük a szükséges vacuity értékeket, majd azokat felhasználva az eredeti formulából készítjük el a legerősebb teljesülő formulát.

Ezt a módszert alkalmazva viszont felmerül egy újabb probléma, hiszen az eredeti φ formulában szereplő literálok mások lehetnek, mint a negált normál formájú φ_n-ben szereplő literálok. Ez abból adódik, hogy előfordulhat, hogy míg az egyik formulában egy atomi kijelentés ponáltan szerepel, addig a másikban már negáltan. A probléma tehát az, hogy nem tudjuk egyszerűen a negált normál formában *hamis*ra cserélt literálokat az eredeti formulában is lecserélni, mivel azok lehet, hogy nem is léteznek abban. Ennek a problémának a kiküszöböléséhez arra van szükség, hogy a ③ lépésben ne az egyes literálokat cseréljük *hamis*ra, hanem csak az egyes atomi kijelentéseket cseréljük *hamis*ra, vagy *igaz*ra annak megfelelően, hogy ponált, vagy negált kijelentésekről van szó. Amennyiben a negált normál formájú kifejezésben történt átalakítások során eltároljuk, hogy melyik atomi kijelentést milyen logikai értékre cseréltünk le, ugyanezeket az átalakításokat el tudjuk végezni az eredeti kifejezésen is.

Végül a ④ lépésben a modellellenőrzések, illetve a felhasználónak történő visszajelzés előtt a korábban eltárolt szótár alapján vissza kell állítani az ① lépésben átnevezett atomi kijelentések eredeti nevét.

A kifejezésekben szereplő atomi kijelentések *hamis* vagy *igaz* értékre cserélésével gyakran olyan formulákat kaphatunk, amelyeknek nagy része leegyszerűsíthető rövidebb kifejezéssé. Az algoritmus során elkészített formulákat a könnyebb értelmezhetőség kedvéért az alábbi egyszerű azonosságok szerint alakítom át:

- $igaz \vee \varphi = igaz$
- $hamis \vee \varphi = \varphi$
- $igaz \wedge \varphi = \varphi$
- $hamis \wedge \varphi = hamis$
- $igaz\ \mathbf{R}\ igaz = igaz$
- $hamis\ \mathbf{R}\ hamis = hamis$
- $igaz\ \mathbf{U}\ igaz = igaz$
- $\varphi\ \mathbf{U}\ hamis = hamis$

Hasonló azonosságokat fel lehet írni más esetekre, több operátorra is, azonban a fent szereplők alkalmazása a legtöbb esetben elegendőnek bizonyul a kifejezések értelmezhetőségének javításához.

Példa 13. *Ebben a példában bemutatom, hogy hogyan változik egy kifejezés az algoritmus különböző lépései során. Tekintsük az alábbi LTL-kifejezést:*

$$\varphi_0 = \neg \boldsymbol{F}(a \vee (\neg a \wedge b))$$

A φ_0 formula a felhasználó által specifikált tulajdonság, ezért a legerősebb teljesülő formulát is hasonló alakban szeretnénk előállítani. A kifejezésben azonban egyrészt van olyan atomi

kijelentés, ami többször is szerepel, másrészt nem negált normálformájú. Ahhoz, hogy az algoritmust alkalmazni tudjuk, először át kell nevezni az atomi kifejezéseket, és el kel tárolni egy szótárat (lásd 3.1. táblázat), ami alapján vissza tudjuk majd állítani az eredeti neveket.

$$\varphi_1 = \neg \boldsymbol{F}(r1 \vee (\neg r2 \wedge r3))$$

Új név	Eredeti név
r1	a
r2	a
r3	b

3.1. táblázat. Szótár az átnevezett atomi kijelentésekhez

Az átnevezések után negált normál formára hozzuk a kifejezést:

$$\varphi_2 = \boldsymbol{G}(\neg r1 \wedge (r2 \vee \neg r3))$$

Amennyiben az átnevezett atomi kijelentésekből létrehoztunk vacuity értékeket, azok alapján már el tudjuk végezni a megfelelő atomi kijelentések cseréjét. Most ezt az átalakítást a $\{|\{r2, r3\}|\}$ *vacuity érték alapján végzem el. Az* $r2$ *kijelentés egyszerűen hamisra cserélhető, míg az* $r3$ *kijelentést a negálás miatt igazra kell cserélni (hiszen valójában a teljes* $\neg r3$ *literált kéne hamisra cserélni). A módosítások elvégzésével megkapjuk a* φ_3 *kifejezést.*

$$\varphi_3 = \boldsymbol{G}(\neg r1 \wedge (hamis \vee \neg igaz))$$

Ezt a kifejezést már át lehet adni a modellellenőrzőnek, így kiderülhet, hogy a vacuity értéket el kell-e tárolnunk a legerősebb formula meghatározásához.

Amennyiben az átalakított kifejezést már nem modellellenőrzéshez készítjük, hanem a legerősebb teljesülő formula egyes részeit állítjuk elő, akkor ehhez szeretnénk az eredeti formula alakját megtartani, és nem a negált normálformára hozott φ_2 *felépítését követni. Ehhez egy hasonló szótárra van szükség, mint amit az atomi kifejezések átnevezésénél használtunk, csak ebben azt kell tárolni, hogy melyik kijelentést milyen logikai értékre kell cserélni. Egy ilyen szótárat a 3.2. táblázatban láthatunk.*

Új név	Eredeti név
r2	*hamis*
r3	*igaz*

3.2. táblázat. Szótár az atomi kijelentések cseréjéhez

Egy ilyen szótár birtokában már elvégezhetjük az atomi kijelentések cseréjét az eredeti kifejezés alakját őrző φ_1 *formulán:*

$$\varphi_4 = \neg \boldsymbol{F}(r1 \vee (\neg hamis \wedge igaz))$$

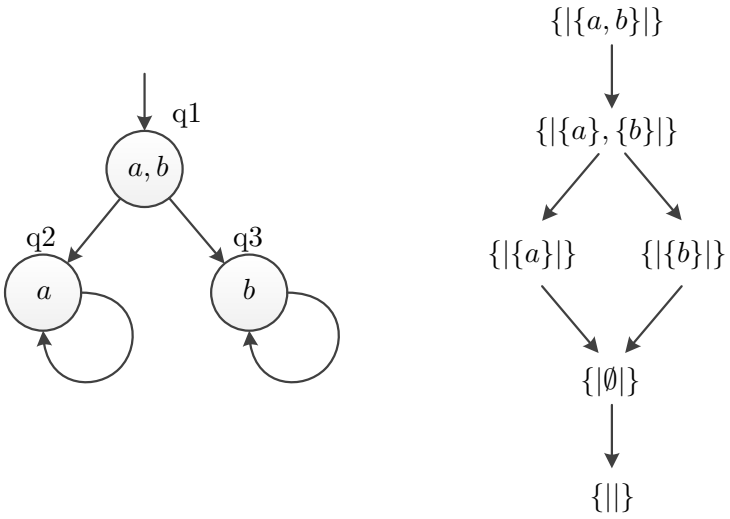
Az atomi kijelentések cseréje után végül a 3.1. táblázatban szereplő szótár segítségével a megmaradt atomi kijelentéseket visszaírhatjuk az eredeti nevükre (φ_4), majd néhány esetleges egyszerűsítés után megkaphatjuk a kívánt formulát (φ_5).

$$\varphi_4 = \neg \boldsymbol{F}(a \vee (\neg hamis \wedge igaz))$$
$$\varphi_5 = \neg \boldsymbol{F}(igaz)$$

3.3.2. A vacuity rács kezelése

Az algoritmus működéséhez szükség van az egyes megtalált v vacuity értékek szülőinek ($\hat{v}$) meghatározására, ami egy egyszerű feladat, amennyiben rendelkezésünkre áll a φ kifejezés atomi kijelentéseiből (AP) létrehozott $V(AP)$ vacuity rács. Emiatt az algoritmus a modellellenőrzések elvégzése előtt felépíti a szükséges vacuity rácsot. Ez a művelet azonban időigényes lehet, hiszen a vacuity rács mérete exponenciális a φ formulában szereplő atomi kijelentések számában, viszont mivel a szükséges modellellenőrzések száma is exponenciális a formula méretében, ezért ez várhatóan nem okoz jelentős változást a futási időben.

Itt jegyezném meg, hogy bár az algoritmus leírása szerint az első módosított formulát a rács $\perp$ eleme alapján kell elkészíteni, ez nem feltétlenül szükséges. Ez abból adódik, hogy minden vacuity rács legkisebb eleme a $\{||\}$ vacuity érték, ami azt jelenti, hogy az ez alapján módosított kifejezés megegyezik az eredeti, a vizsgált modellen teljesülő kifejezéssel (ha nem teljesülne, nem végeznénk ilyen vizsgálatokat), tehát biztos, hogy kielégíthető. Ez az állítás továbbá igaz az eggyel nagyobb $\{|\emptyset|\}$ értékre is, így a lényegi vizsgálatokhoz elég az ennél eggyel nagyobb, már atomi kijelentéseket ténylegesen tartalmazó vacuity értékek feldolgozásával kezdeni a modellellenőrzéseket.



3.6. ábra. Egy modell és egy vacuity rács.

Példa 14. *Ebben a példában bemutatom a teljes algoritmus működését. Legyen $\varphi = \boldsymbol{G}(a \vee b)$ az az LTL formula, ami alapján szeretnénk meghatározni egy erősebb formulát, ami szintén teljesül a 3.6. ábrán szereplő modellen. Mivel φ nem tartalmaz többször*

előforduló atomi kijelentéseket, ezért azok átnevezésére most nincs szükség. A φ kifejezéshez felépített vacuity rács szintén ezen az ábrán látható. Az első ténylegesen ellenőrizendő módosított kifejezések (a triviális alsó két szint átugrása után) az $\{|\{a\}|\}$ és $\{|\{b\}|\}$ vacuity értékek alapján létrehozott formulák:

$$C(\{|\{a\}|\}) = \varphi^{\{a\}} = \boldsymbol{G}(hamis \vee b) = \boldsymbol{G}(b)$$

$$C(\{|\{b\}|\}) = \varphi^{\{b\}} = \boldsymbol{G}(a \vee hamis) = \boldsymbol{G}(a)$$

Amennyiben ezekkel a kifejezésekkel elvégezzük a modellellenőrzéseket, azt kapjuk, hogy mindkét formula kielégíthető a modellen, így a megtalált vacuity értékek V halmazához hozzáadjuk a két megvizsgált vacuity értéket.

Az algoritmus következő lépése ekkor az ezeknél eggyel nagyobb vacuity értékek ellenőrzése. Jelen esetben ez csak egyetlen vacuity értéket, az $\{|\{a\},\{b\}|\}$-t jelenti. Ehhez az alábbi módosított kifejezés tartozik:

$$C(\{|\{a\},\{b\}|\}) = \varphi^{\{a\}} \wedge \varphi^{\{b\}} = \boldsymbol{G}(b) \wedge \boldsymbol{G}(a)$$

Ez a kifejezés nem teljesíthető a modellen, így az $\{|\{a\},\{b\}|\}$ vacuity értéket nem vesszük fel a V halmazba. A megtalált vacuity értékek halmaza:

$$V = \{\{|\{a\}|\},\{|\{b\}|\}\}$$

A következő lépésben el kell távolítanunk a V halmazból azokat a v elemeket, amikhez nem létezik olyan π útvonal a modellben, hogy $vac(\pi,\varphi) = v$. Ehhez létre kell hoznunk minden vacuity értékhez a hozzá tartozó φ'_v kifejezést. Mivel a megtalált két vacuity értéknél eggyel nagyobb értékek már nem szerepelnek a V halmazban, ezért ezek a formulák azonosak lesznek a már ellenőrzött kifejezésekkel.

$$\varphi'_{\{|\{a\}|\}} = \boldsymbol{G}(b)$$

$$\varphi'_{\{|\{b\}|\}} = \boldsymbol{G}(a)$$

Mivel a korábbi modellellenőrzések alapján tudjuk, hogy ezek a kifejezések kielégíthetőek a modellen, nem kell újabb ellenőrzéseket futtatnunk, az új V' halmaz pedig megegyezik a V halmazzal.

Az algoritmus utolsó lépése, hogy ebből a V' halmazból elhagyja a nem minimális vacuity értékeket. Mivel V' nem tartalmaz egyik megtalált vacuity értéknél kisebb elemet sem, ezért

$$V_{min} = V' = \{\{|\{a\}|\},\{|\{b\}|\}\}$$

A V_{min} halmaz alapján ezek után létre lehet hozni a legerősebb teljesülő formulát:

$$\Phi_{min}(M,\varphi) = \bigvee_{v \in V_{min}} \bigwedge_{S \in v} (\varphi^S) = \boldsymbol{G}(a) \vee \boldsymbol{G}(b)$$

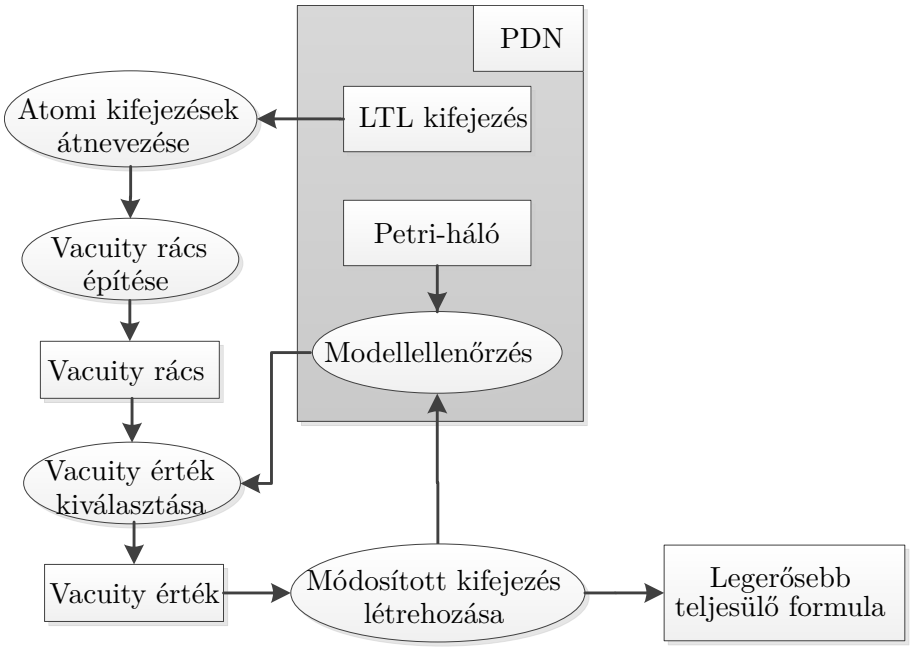
Megfigyelhetjük, hogy a $\Phi_{min}(M, \varphi)$ kifejezés az eredeti φ formulánál jóval pontosabban írja le a modell viselkedését, ezzel információt szolgáltatva a felhasználónak a formula teljesülésének okáról.

4. fejezet

Implementáció

Az algoritmust a PetriDotNet (PDN) [27] modellellenőrző keretrendszer részeként készítettem el. A PetriDotNet a BME Méréstechnikai és Információs Rendszerek Tanszékén fejlesztett, Petri-hálók [24] szerkesztésére és analízisére szolgáló eszköz, melynek része több, a tanszéken fejlesztett modellellenőrzési megoldás is [10, 11, 26]. Az algoritmusom ezek közül a korábbi TDK dolgozatunkban[23] bemutatott szaturáció alapú LTL-modellellenőrzést[7, 8, 9] használja fel. A keretrendszer .NET alapú, így elsősorban Windows alatt futtatható, bár létezik megoldás Linuxon történő használatra is (Mono framework). Ebben a fejezetben először ismertetem az implementált algoritmus és a PetriDotNet keretrendszer viszonyát, majd bemutatom az algoritmus működését biztosító programrész felépítését.

4.1. Az algoritmus és a PetriDotNet viszonya



4.1. ábra. Az algoritmus illeszkedése a PetriDotNethez.

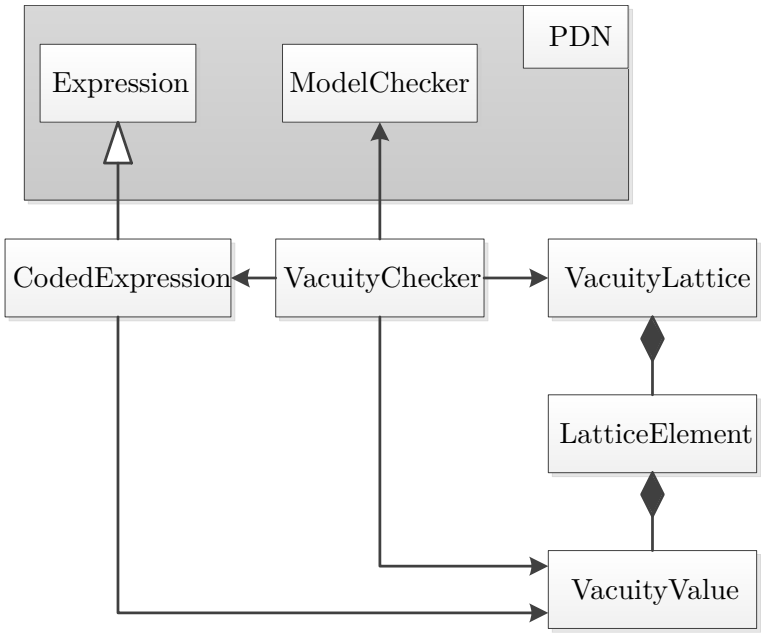
Ahogy az a 4.1. ábrán látható, az algoritmusnak szigorú értelemben véve egyetlen bemenete a felhasználó által megadott, az aktuális modellen teljesülő LTL-kifejezés, amiből egy erősebb teljesülő formulát hoz létre. Amennyiben a kifejezés nem teljesül a modellen, az algoritmus nem kerül meghívásra. A kifejezésen kívül lehetne még bemenetként tekinteni a Petri-hálóra, amin a modellellenőrzések történnek, azonban ezt az algoritmusom közvetlenül nem használja fel, csupán a PetriDotNet modellellenőrző funkcióján keresztül rendelkezik befolyással a végeredményre.

Az algoritmus első lépésben az atomi propozíciók átnevezésével (és negált normált formára hozásával) megfelelő formára hozza a PetriDotNettől kapott kifejezést, majd annak atomi kijelentéseiből felépíti a vacuity rácsot. A rácsból kiválasztott vacuity érték alapján ezután létrehozza, majd a PetriDotNetnek modellellenőrzésre átadja a megfelelő módosított kifejezést, majd az eredmény ismeretében újabb vacuity értéket választ a vacuity rácsból. Végül a megfelelő eredmények ismeretében az összegyűjtött vacuity értékek alapján elkészíti a legerősebb teljesülő kifejezést.

4.2. A program felépítése

Ahogy az a 4.2. ábrán látszik, az algoritmus működéséhez viszonylag kevés új osztályra van szükség, melyek alapvetően három csoportba oszthatók:

- Kifejezéseket reprezentáló osztály
- Vacuity értékekkel kapcsolatos osztályok
- Az algoritmus működését megvalósító osztály



4.2. ábra. Az algoritmushoz kapcsolódó osztályok.

A kifejezéseket reprezentáló *CodedExpression* osztály a PetriDotNet korábban megvalósított Expression osztályának leszármazottja, és két funkcióval bővíti ki az eredeti osztályt:

- Atomi kifejezések átnevezése és visszaállítása
- Vacuity értékek alapján atomi kifejezések cseréje

Az algoritmus működéséhez egyedi nevű, egymástól megkülönböztethető atomi kijelentésekre van szükség, amit ez az osztály biztosít. Egy bemenetként kapott hagyományos Expression objektum alapján úgy épít fel egy kifejezést, hogy a benne szereplő atomi kijelentéseket új, egyedi névvel rendelkező kijelentésekre cseréli, illetve nyilvántartja azok eredeti megfelelőit. Ennek köszönhetően visszaállítható belőle a modellellenőrzéshez szükséges eredeti atomi kijelentéseket tartalmazó kifejezés.

Az osztály másik funkciója a kifejezés vacuity értékek alapján történő módosítása. Mivel az LTL-kifejezések a PetriDotNetben egy fa struktúraként vannak tárolva, ezért a feladat ennek bejárásával, az atomi kijelentések konstans logikai értékre cserélésével, illetve új részfák létrehozásával oldható meg. A 3.3.1. fejezetben leírtaknak megfelelően az egyes atomi kijelentéseket attól függően kell *hamis* vagy *igaz* értékre cserélni, hogy a negált normál formában az adott kijelentés ponáltan, vagy negáltan szerepel. Ahhoz, hogy az atomi kijelentések cseréjét el lehessen végezni az eredeti alakú kifejezésen is, az osztály nyilvántartja, hogy melyik atomi kijelentést milyen logikai értékre kellett lecserélni, illetve a vacuity értékeken túl ilyen információk alapján is lehetővé teszi az atomi kijelentések cseréjét.

Az osztályok második csoportja a vacuity értékekhez kötődik. Az *VacuityValue* osztály magukat a vacuity értékeket reprezentálja, a *VacuityLattice* pedig az ezekből építhető vacuity rácsokat. A *LatticeElement* osztály példányai vacuity egy-egy vacuity értéket tartalmaznak, és néhány a rács elemeihez kötődő funkcionalitással bővítik ki azokat.

A *VacuityValue* osztály a vacuity értékek fogalmának megfelelően atomi kijelentések halmazainak halmazát tartalmazza, illetve biztosítja a vacuity értékeken végezhető műveleteket. Ilyen műveletek például a vacuity értékek uniójának illetve metszetének képzése, vagy az értékek megszokott rendezés szerinti összehasonlítása.

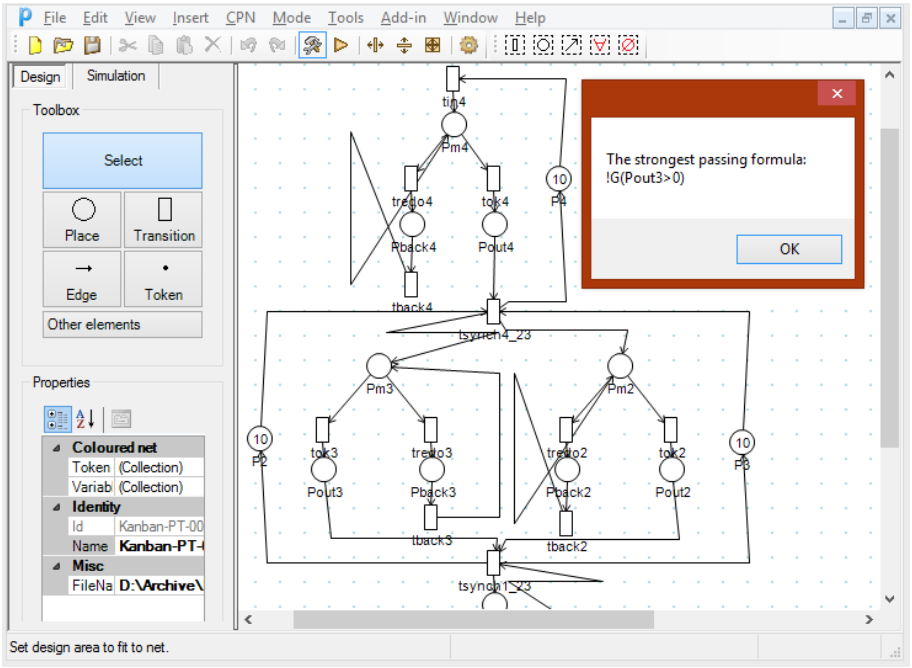
A *VacuityLattice* osztály felelős a vacuity rácsok kezeléséért, annak atomi kijelentésekből történő felépítéséért, és az elemeinek létrehozásáért. El lehet továbbá kérni tőle a legkisebb ($\bot$) és legnagyobb ($\top$) elemeket.

A *LatticeElement* osztály példányait tartalmazzák a vacuity rácsok. Ezeknek az objektumoknak tudomásuk van a rácsban náluk eggyel kisebb, illetve nagyobb elemekről, így megkönnyítik azok elérését.

Az utolsó osztály amiről nem esett szó, az algoritmus működését biztosító *VacuityChecker*. Ez az osztály kezeli a a PetriDotnettől megkapott kifejezést, a vacuity rácsot, illetve ezek segítségével megvalósítja az ismertetett algoritmust. Csak a VacuityCheckernek van kapcsolata a PetriDotNet korábban implementált modellellenőrző komponensével, így itt történik a módosított kifejezések átadása annak, illetve az eredmények feldolgozása is.

4.2.1. A funkció használata

A legerősebb teljesülő formula megkeresését a korábbi LTL-modellellenőrző funkcióval megegyezően lehet indítani. Egy Petri-háló megnyitása és az ellenőrizendő LTL-kifejezés megadása után először egy hagyományos modellellenőrzés hajtódik végre, majd teljesülő formula esetén lefut az implementált algoritmus. Egy ilyen futás eredménye a 4.3. ábrán látható.



4.3. ábra. Az algoritmus futtatásának eredménye.

5. fejezet

Mérési eredmények

Az elkészített implementáció teszteléséhez és teljesítményének méréséhez a *Model Checking Contest* Petri-háló modelljeit és az ezekhez a SPOT eszközzel generált véletlenszerű LTL formulákat használtam fel. A méréseket egy átlagos laptopon[1] végeztem el.

5.1. A megtalált formulák

Az alábbiakban különböző modelleken vizsgált kifejezéseket (φ_i), és az implementált algoritmus által megtalált erősebb teljesülő formulákat (Φ_i) mutatok be. Elsőként tekintsük a Kanban modellen [18] ellenőrzött φ_1 kifejezést, és a megtalált Φ_1 legerősebb teljesülő formulát.

$$\varphi_1 = \neg((Pback2 > 0) \wedge \mathbf{F}\neg(Pback2 > 0))$$
$$\Phi_1 = \neg((Pback2 > 0) \wedge \mathbf{F}igaz)$$

Ahogy az a módosított kifejezésből látható, az $\mathbf{F}\neg(Pback2 > 0))$ részkifejezésnek nincs hatása a modellellenőrzés eredményére, az eredeti formula érvényessége kizárólag annak első felétől függ, ami viszont csupán a modell kezdeti állapotára vonatkozik. Egy ehhez hasonló eredmény fontos visszajelzés lehet a felhasználó számára, hiszen az ellenőrzött kifejezés, vagy a modell alapvető hibájára hívhatja fel a figyelmet.

A következő, φ_2 formulát a *RES-Allocation-2* modellen ellenőriztem az alábbi eredménnyel.

$$\varphi_2 = \neg\mathbf{XX}((p_{0,0} > 0) \vee \mathbf{X}(r_{1,1} > 0)\mathbf{U}\neg(r_{1,1} > 0))$$
$$\Phi_2 = \neg\mathbf{XX}(igaz\mathbf{U}\neg(r_{1,1} > 0))$$

Ebben az esetben a kapott Φ_2 formulában az eredeti kifejezéshez képest több atomi kijelentés is lecserélésre került. Ha részletesebben megvizsgáljuk a kifejezést, láthatjuk, hogy csak az *until* operátor jobboldalán szereplő kijelentés maradt meg eredeti formájában. Bár az algoritmus ilyen típusú átalakításokat nem végez az eredményen, az $igaz\mathbf{U}\neg(r_{1,1} > 0)$ alakú részkifejezés ekvivalens az $\mathbf{F}\neg(r_{1,1} > 0)$ kifejezéssel. Ebből a felhasználó észreveheti,

[1] Intel i3-2310M CPU @ 2.10GHz, 8 GB rendszermemória, Samsung 840 EVO SSD, Windows 8.1 x64

hogy az eredeti formulában a szándékai szerint egy tulajdonság folyamatos teljesülését leíró részkifejezés valójában nem hordoz magában információt a teljes kifejezés érvényességének szempontjából.

A φ_3 kifejezést szintén a *RES-Allocation-2* modellen ellenőriztem.

$$\varphi_3 = \neg\mathbf{GF}((p_{0,1} > 0) \wedge \neg(r_{1,1} > 0))$$
$$\Phi_2 = \neg\mathbf{GF}\neg(r_{1,1} > 0) \vee \neg\mathbf{GF}(p_{0,1} > 0)$$

A φ_3 kifejezés ellenőrzésének eredményeként az előző esetekhez képest egy bonyolultabb formulát kaptunk olyan szempontból, hogy itt már nem olyan egyértelmű, hogy az eredeti formulának mely részei feleslegesek. A Φ_3 formula egy két tagú diszjunkció, amiből arra következtethetünk, hogy az általunk megfogalmazott komplex tulajdonság a modellben két egyszerűbb esetre osztható fel, amik közül bármelyik teljesülése esetén az eredeti kifejezés is teljesül a modellen. Egy ilyen visszajelzés alapján a felhasználó egyrészt következtethet valamilyen hibára, másrészt részletesebb képet alkothat a modell működéséről és arról, hogy milyen lefutások okozhatják a megadott követelmény teljesülését.

5.2. Az algoritmus teljesítménye

Az alábbiakban bemutatom, hogy milyen tényezőktől milyen mértékben függ a legerősebb formula meghatározásának sebessége, illetve mik jelenthetnek korlátot a módszer alkalmazásában.

Az implementált algoritmus két esetlegesen nagy számítási igényű részfeladatot végez el. Az egyik a modellellenőrzések futtatása, amely egyrészt önmagában sok időbe telhet nagy modellek, vagy sok temporális operátort tartalmazó kifejezések esetén, másrészt a vacuity ellenőrzés során akár az atomi kijelentések számában exponenciálisan sok ilyen ellenőrzés futtatására is szükség lehet. A másik nagy számítási igényű feladat a vacuity rács felépítése, aminek mérete szintén exponenciális a formulában szereplő atomi kijelentések számában. Az alábbi mérésekkel egyrészt az algoritmus teljes futási idejét, másrészt ezen részfeladatok végrehajtási idejének ehhez viszonyított arányát vizsgáltam különböző modelleken, különböző méretű kifejezésekkel.

A következő méréssorozatot, melynek eredménye az 5.1 táblázatban látható, a Kanban modellen végeztem, aminek egy változó paramétere a modellben szereplő tokenek száma. Ennek növekedésével jelentősen megnövekszik a modell állapottere, és ezzel a modellellenőrzés bonyolultsága is.

Kanban-n				
$\neg((Pback2 > 0) \wedge \mathbf{F}\neg(Pback2 > 0))$				
n	MC-k száma	Teljes idő (s)	Relatív időtartamok	
			Rács létrehozás	Modellellenőrzések
5	3	0,8012	0,01%	98,52%
10	3	0,8268	0,01%	98,53%
20	3	0,8649	0,01%	98,50%
50	3	1,3174	0,01%	99,13%
100	3	4,0456	0,00%	99,70%
200	3	27,1926	0,00%	99,96%
$\neg((\neg(Pback2 > 0) \vee \neg(Pm4 = 0)) \wedge \mathbf{G}(Pout3 > 0))$				
n	MC-k száma	Teljes idő (s)	Relatív időtartamok	
			Rács létrehozás	Modellellenőrzések
5	12	3,1611	0,02%	98,40%
10	12	3,2017	0,02%	98,19%
20	12	3,4436	0,01%	98,55%
50	12	5,6128	0,01%	99,15%
100	12	18,5222	0,00%	99,73%
200	12	132,9781	0,00%	99,95%
$\neg((Pm2 = 0) \wedge \neg(Pm3 > 0) \wedge (Pout3 = 0)\mathbf{R}\neg(Pout1 = 0))$				
n	MC-k száma	Teljes idő (s)	Relatív időtartamok	
			Rács létrehozás	Modellellenőrzések
5	35	36,9279	0,05%	99,17%
10	35	37,2151	0,06%	99,22%
20	35	38,5133	0,06%	99,25%
50	35	46,4225	0,04%	99,42%
100	35	96,8982	0,02%	99,67%
200	35	602,2061	0,00%	99,93%

5.1. táblázat. Mérések a Kanban modelleken

Az 5.1. táblázat alapján elmondható, hogy az algoritmus futási ideje a vártnak megfelelően alakul. A modell állapotterének növekedésével párhuzamosan jelentősen megnövekszik a modellellenőrzések futási ideje, míg a vacuity rács elkészítésének ideje a teljes futási időhöz képest elhanyagolható. A modellellenőrzések futási ideje és száma továbbá jelentős mértékben függ a megadott kifejezés méretétől (operátorok és atomi kijelentések száma) is.

A modellek állapotterének mérete mellett a futási időt leginkább befolyásoló tényező a specifikációs kifejezések mérete. A következő mérésekkel a futási időt ennek függvényében vizsgáltam. Az 5.2. táblázatban látható adatok a különböző méretű kifejezések átlagos futási adatait írják le.

AP-k száma	Teljes idő (s)	Relatív időtartamok		
		Rács létrehozás	Modellellenőrzések	Egyéb műveletek
2	2,4357	0,22%	97,24%	2,54%
3	55,6628	0,01%	98,98%	1,01%
4	114,2543	0,18%	98,92%	1,53%
5	>10800	-	-	-

5.2. táblázat. Futási idő a kifejezés méretének függvényében

A mérési eredmények azt mutatják, hogy a várakozásoknak megfelelően a legtöbb időt a modellellenőrzések jelentik. Öt atomi kijelentést tartalmazó kifejezések esetén olyan sok és nagy méretű formula kielégíthetőségét kell vizsgálni, hogy a legerősebb formulát több óra alatt sem sikerült meghatározni. Ennek megfelelően a módszer használhatóságának javításához célszerű lenne a jövőben megvizsgálni, hogy lehetséges-e az algoritmus futása közben előállított kifejezéseknek a számát és méretét csökkenteni.

Az összes mért adatot a függelékben megtalálható F.1.-F.9. táblázatok tartalmazzák.

6. fejezet

Összefoglalás

Dolgozatomban bemutattam egy algoritmumst, ami a modellellenőrzés során teljesülő kifejezések vizsgálatával hasznos visszajelzést képes adni a felhasználónak, ezzel elősegítve a modellben és a specifikációs kifejezésben előforduló hibák feltárását és javítását. Megalkottam és ismertettem az algoritmusnak egy egyszerűbb matematikai struktúrákat felhasználó, a PetriDotNet keretrendszerben használt algoritmusokhoz illeszkedő változatát, majd ennek implementációját is elkészítettem. Az implementált algoritmuson egy nagy elemszámú tesztkészlet segítségével különböző méretű és felépítésű modelleken és kifejezéseken méréseket végeztem, ezzel vizsgálva az algoritmus hatékonyságát és használhatóságát.

Az elvégzett munkám alapján kimondható, hogy a technológia segítségével lehetőség nyílik a felhasználó számára hasznos információk szolgáltatására teljesülő specifikációk esetén is, ezzel növelve a modellellenőrzés alkalmazhatóságát.

Mivel az implementált algoritmus jelenleg viszonylag kis méretű kifejezések esetén használható eredménnyel, így széleskörű alkalmazása előtt szükség van további fejlesztésekre. Ahhoz, hogy nagyobb LTL-kifejezések esetén is elfogadható időn belül eredményt adjon, két alapvető csoportra lehet bontani az ezzel kapcsolatos munkákat. Egyrészt a modellellenőrző algoritmusok fejlesztésével és hatékonyabbá válásával a bemutatott algoritmus időigénye is jelentősen lecsökkenhet, hiszen a mért adatok alapján a futás során ezzel telt el a legtöbb idő. Másrészt hasznos lenne annak vizsgálata, hogy milyen módon lehet az algoritmus futása során modellellenőrzésre átadott kifejezéseknek a számát, illetve elsősorban a méretét csökkenteni, ezzel gyorsítva a modellellenőrzések futását. További fejlesztési lehetőség a dolgozatomban bemutatott algoritmus eredeti változatának implementálása, és annak hatékonyságának összevetése az általam implementált változattal.

Ábrák jegyzéke

Irodalomjegyzék

[1] Derek L Beatty and Randal E Bryant. Formally verifying a microprocessor using a simulation methodology. In *Proceedings of the 31st annual Design Automation Conference*, pages 596–602. ACM, 1994.

[2] Ilan Beer, Shoham Ben-David, Cindy Eisner, and Yoav Rodeh. Efficient detection of vacuity in temporal model checking. *Formal Methods in System Design*, 18(2):141–163, 2001.

[3] Julius Richard Büchi. *On a decision method in restricted second order arithmetic*. na, 1960.

[4] Marsha Chechik, Benet Devereux, Steve Easterbrook, and Arie Gurfinkel. Multi-valued symbolic model-checking. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(4):371–408, 2003.

[5] Marsha Chechik, Benet Devereux, and Arie Gurfinkel. Model-checking infinite state-space systems with fine-grained abstractions using spin. In *Proceedings of the 8th international SPIN workshop on Model checking of software*, pages 16–36. Springer-Verlag New York, Inc., 2001.

[6] Hana Chockler, Arie Gurfinkel, and Ofer Strichman. Beyond vacuity: Towards the strongest passing formula. In Alessandro Cimatti and Robert B. Jones, editors, *FM-CAD*, pages 1–8. IEEE, 2008.

[7] Gianfranco Ciardo, Gerald Lüttgen, and Radu Siminiceanu. Saturation: an efficient iteration strategy for symbolic state space generation. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS), LNCS 2031*, pages 328–342. Springer-Verlag, 2001.

[8] Gianfranco Ciardo, Robert Marmorstein, and Radu Siminiceanu. Saturation unbound. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 379–393. Springer, 2003.

[9] Gianfranco Ciardo, Robert Marmorstein, and Radu Siminiceanu. The saturation algorithm for symbolic state-space exploration. *Int. J. Softw. Tools Technol. Transf.*, 8(1):4–25, 2006.

[10] Darvas Dániel. Szaturáció alapú automatikus modellellenőrző fejlesztése aszinkron rendszerekhez. *Tudományos Diákköri Konferencia, Budapesti Műszaki és Gazdaságtudományi Egyetem, Villamosmérnöki és Informatikai Kar*, 2010.

[11] Darvas Dániel, Jámbor Attila. Komplex rendszerek modellezése és verifikációja. *Tudományos Diákköri Konferencia, Budapesti Műszaki és Gazdaságtudományi Egyetem, Villamosmérnöki és Informatikai Kar*, 2011.

[12] Jr. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 2001.

[13] Antony Galton. *Temporal logics and their applications*, volume 10. Academic Press London, 1987.

[14] Rob Gerth, Doron Peled, Moshe Y. Vardi, R. Gerth, Den Dolech Eindhoven, D. Peled, M. Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *In Protocol Specification Testing and Verification*, pages 3–18. Chapman & Hall, 1995.

[15] Susanne Graf. On lamport's comparison between linear and branching time temporal logic. *ITA*, 18(4):345–353, 1984.

[16] Arie Gurfinkel and Marsha Chechik. Generating counterexamples for multi-valued model-checking. In *FME 2003: Formal Methods*, pages 503–521. Springer, 2003.

[17] Arie Gurfinkel and Marsha Chechik. How vacuous is vacuous? In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 451–466. Springer, 2004.

[18] Muris Lage Junior and Moacir Godinho Filho. Variations of the kanban system: Literature review and classification. *International Journal of Production Economics*, 125(1):13 – 21, 2010.

[19] Yonit Kesten, Zohar Manna, Hugh McGuire, and Amir Pnueli. A decision algorithm for full propositional temporal logic. In *Computer Aided Verification*, pages 97–109. Springer, 1993.

[20] Orna Kupferman and Yoad Lustig. Lattice automata. In Byron Cook and Andreas Podelski, editors, *VMCAI*, volume 4349 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2007.

[21] Orna Kupferman and Moshe Y Vardi. Vacuity detection in temporal model checking. *International Journal on Software Tools for Technology Transfer*, 4(2):224–233, 2003.

[22] Orna Lichtenstein and Amir Pnueli. Propositional temporal logics: Decidability and completeness. *Logic Journal of the IGPL*, 8(1):55–85, 2000.

[23] Molnár Vince, Segesdi Dániel. Múlt és jövő: Új algoritmusok lineáris temporális tulajdonságok szaturáció-alapú modellellenőrzésre. *Tudományos Diákköri Konferencia,*

Budapesti Műszaki és Gazdaságtudományi Egyetem, Villamosmornöki és Informatikai Kar, 2013.

[24] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[25] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, 1994.

[26] Vörös András, Darvas Dániel, Bartha Tamás. Bounded Saturation Based CTL Model Checking. In Jaan Penjam, editor, *Proceedings of the 12th Symposium on Programming Languages and Software Tools, SPLST'11*, pages 149–160, Tallinn, Estonia, 2011. Tallinn University of Technology, Institute of Cybernetics.

[27] A *PetriDotNet* keretrendszer honlapja. (elérve: 2014. december 20.) `http://petridotnet.inf.mit.bme.hu/`.

Függelék

Modell	Az ellenőrzött φ és a legerősebb teljesülő Φ kifejezés	MC[a]	Futási idő
FMS-002	$\varphi = \neg\mathbf{G}\neg(P3 > 0)$ $\Phi = \varphi$	0	0,07
FMS-002	$\varphi = \neg(\neg(P2 > 0)\mathbf{R}\neg(P3 > 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(P3 > 0))$	1	0,29
FMS-002	$\varphi = \neg((P2s > 0) \wedge \mathbf{F}\neg(P2s > 0))$ $\Phi = \neg((P2s > 0) \wedge \mathbf{F}igaz)$	3	0,92
FMS-002	$\varphi = \neg((P1 > 0) \wedge \neg(P1wP2 > 0) \wedge (P3s > 0)\mathbf{R}\neg(P12s = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(P12s = 0))$	35	39,47
FMS-005	$\varphi = \neg((P1 > 0) \wedge \neg(P1wP2 > 0) \wedge (P3s > 0)\mathbf{R}\neg(P12s = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(P12s = 0))$	35	41,67
FMS-010	$\varphi = \neg((P1 > 0) \wedge \neg(P1wP2 > 0) \wedge (P3s > 0)\mathbf{R}\neg(P12s = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(P12s = 0))$	35	39,54
FMS-020	$\varphi = \neg((P1 > 0) \wedge \neg(P1wP2 > 0) \wedge (P3s > 0)\mathbf{R}\neg(P12s = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(P12s = 0))$	35	45,83
FMS-050	$\varphi = \neg((P1 > 0) \wedge \neg(P1wP2 > 0) \wedge (P3s > 0)\mathbf{R}\neg(P12s = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(P12s = 0))$	35	59,53
FMS-100	$\varphi = \neg((P1 > 0) \wedge \neg(P1wP2 > 0) \wedge (P3s > 0)\mathbf{R}\neg(P12s = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(P12s = 0))$	35	148,18
FMS-200	$\varphi = \neg((P1 > 0) \wedge \neg(P1wP2 > 0) \wedge (P3s > 0)\mathbf{R}\neg(P12s = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(P12s = 0))$	35	800,31
Kanban-0005	$\varphi = \neg((Pback2 > 0) \wedge \mathbf{F}\neg(Pback2 > 0))$ $\Phi = \neg((Pback2 > 0) \wedge \mathbf{F}igaz)$	3	0,80
Kanban-0005	$\varphi = \neg((\neg(Pback2 > 0) \vee \neg(Pm4 = 0)) \wedge \mathbf{G}(Pout3 > 0))$ $\Phi = \neg\mathbf{G}(Pout3 > 0)$	12	3,16
Kanban-0005	$\varphi = \neg((Pm2 = 0) \wedge \neg(Pm3 > 0) \wedge (Pout3 = 0)\mathbf{R}\neg(Pout1 = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(Pout1 = 0))$	35	36,93
Kanban-0010	$\varphi = \neg((Pback2 > 0) \wedge \mathbf{F}\neg(Pback2 > 0))$ $\Phi = \neg((Pback2 > 0) \wedge \mathbf{F}igaz)$	3	0,83

F.1. táblázat. Mérési eredmények

[a]MC - Model-Checking, itt a modellellenőrzések számát jelöli

Modell	Az ellenőrzött φ és a legerősebb teljesülő Φ kifejezés	MC	Futási idő
Kanban-0010	$\varphi = \neg((\neg(Pback2 > 0) \vee \neg(Pm4 = 0)) \wedge \mathbf{G}(Pout3 > 0))$ $\Phi = \neg\mathbf{G}(Pout3 > 0)$	12	3,20
Kanban-0010	$\varphi = \neg((Pm2 = 0) \wedge \neg(Pm3 > 0) \wedge (Pout3 = 0)\mathbf{R}\neg(Pout1 = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(Pout1 = 0))$	35	37,22
Kanban-0020	$\varphi = \neg((Pback2 > 0) \wedge \mathbf{F}\neg(Pback2 > 0))$ $\Phi = \neg((Pback2 > 0) \wedge \mathbf{F}igaz)$	3	0,86
Kanban-0020	$\varphi = \neg((\neg(Pback2 > 0) \vee \neg(Pm4 = 0)) \wedge \mathbf{G}(Pout3 > 0))$ $\Phi = \neg\mathbf{G}(Pout3 > 0)$	12	3,44
Kanban-0020	$\varphi = \neg((Pm2 = 0) \wedge \neg(Pm3 > 0) \wedge (Pout3 = 0)\mathbf{R}\neg(Pout1 = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(Pout1 = 0))$	35	38,51
Kanban-0050	$\varphi = \neg((Pback2 > 0) \wedge \mathbf{F}\neg(Pback2 > 0))$ $\Phi = \neg((Pback2 > 0) \wedge \mathbf{F}igaz)$	3	1,32
Kanban-0050	$\varphi = \neg((\neg(Pback2 > 0) \vee \neg(Pm4 = 0)) \wedge \mathbf{G}(Pout3 > 0))$ $\Phi = \neg\mathbf{G}(Pout3 > 0)$	12	5,61
Kanban-0050	$\varphi = \neg((Pm2 = 0) \wedge \neg(Pm3 > 0) \wedge (Pout3 = 0)\mathbf{R}\neg(Pout1 = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(Pout1 = 0))$	35	46,42
Kanban-0100	$\varphi = \neg((Pback2 > 0) \wedge \mathbf{F}\neg(Pback2 > 0))$ $\Phi = \neg((Pback2 > 0) \wedge \mathbf{F}igaz)$	3	4,05
Kanban-0100	$\varphi = \neg((\neg(Pback2 > 0) \vee \neg(Pm4 = 0)) \wedge \mathbf{G}(Pout3 > 0))$ $\Phi = \neg\mathbf{G}(Pout3 > 0)$	12	18,52
Kanban-0100	$\varphi = \neg((Pm2 = 0) \wedge \neg(Pm3 > 0) \wedge (Pout3 = 0)\mathbf{R}\neg(Pout1 = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(Pout1 = 0))$	35	96,90
Kanban-0200	$\varphi = \neg((Pback2 > 0) \wedge \mathbf{F}\neg(Pback2 > 0))$ $\Phi = \neg((Pback2 > 0) \wedge \mathbf{F}igaz)$	3	27,19
Kanban-0200	$\varphi = \neg((\neg(Pback2 > 0) \vee \neg(Pm4 = 0)) \wedge \mathbf{G}(Pout3 > 0))$ $\Phi = \neg\mathbf{G}(Pout3 > 0)$	12	132,98
Kanban-0200	$\varphi = \neg((Pm2 = 0) \wedge \neg(Pm3 > 0) \wedge (Pout3 = 0)\mathbf{R}\neg(Pout1 = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(Pout1 = 0))$	35	602,21

F.2. táblázat. Mérési eredmények

Modell	Az ellenőrzött φ és a legerősebb teljesülő Φ kifejezés	MC	Futási idő
LamportFastMutEx-2	$\varphi = \neg(\neg(done_{1,1} > 0) \wedge \mathbf{G}(\neg(done_{2,1} = 0) \wedge (ify0_{4,2} = 0) \wedge (\neg(setbi_{5,0} = 0)\mathbf{R}(done_{2,2} > 0))))$ $\Phi = -$	-	-
LamportFastMutEx-3	$\varphi = \neg(\neg(done_{1,1} > 0) \wedge \mathbf{G}(\neg(done_{2,1} = 0) \wedge (ify0_{4,2} = 0) \wedge (\neg(setbi_{5,0} = 0)\mathbf{R}(done_{2,2} > 0))))$ $\Phi = -$	-	-
NeoElection-2	$\varphi = \neg(\mathbf{F}(pollNet_{1,0,AnsP,1} > 0) \wedge \mathbf{F}(net_{1,2,AskP,2} > 0))$ $\Phi = \varphi$	2	1,19
NeoElection-2	$\varphi = \neg(\mathbf{G}(electedSecondary_0 > 0) \vee \mathbf{G}\neg(pollNet_{1,1,AskP,1} = 0))$ $\Phi = \varphi$	0	0,01
NeoElection-2	$\varphi = \neg\mathbf{GF}((net_{1,0,AI,2} = 0) \wedge \neg(polling_1 = 0))$ $\Phi = \varphi$	2	1,23
NeoElection-2	$\varphi = \neg((startNegBroadcasting_{1,1} > 0) \wedge \mathbf{F}\neg(startNegBroadcasting_{1,1} > 0))$ $\Phi = \varphi$	2	0,95
NeoElection-2	$\varphi = \neg\mathbf{X}(XG\neg(net_{0,1,AskP,2} = 0)\mathbf{U}\neg(masterList_{0,1,2} > 0) \vee \mathbf{F}\neg(pollNet_{2,1,AskP,0} > 0))$ $\Phi = \varphi$	3	1,11
NeoElection-2	$\varphi = \neg((\neg(net_{1,0,AnsP,2} = 0) \vee \neg(stage_{2,SEC} = 0)) \wedge \mathbf{G}(net_{0,1,AnsP,1} > 0))$ $\Phi = \varphi$	3	1,94
NeoElection-2	$\varphi = \neg(\mathbf{F}(\neg(pollEnd_0 = 0) \wedge \mathbf{G}(pollNet_{0,1,AskP,2} > 0)) \vee (\neg(net_{0,1,AnnP,0} > 0)\mathbf{UX}\neg(net_{2,1,AI,2} = 0)))$ $\Phi = \varphi$	4	2,99
Peterson-2	$\varphi = \neg(\neg(TId_{0,1,2} = 0) \wedge \mathbf{G}((BL_{0,0,2} = 0) \wedge \neg(TId_{2,1,1} = 0) \wedge (\neg(BL_{1,0,0} > 0)\mathbf{R}(TId_{1,0,2} = 0))))$ $\Phi = -$	-	-
Peterson-3	$\varphi = \neg(\neg(TId_{0,1,2} = 0) \wedge \mathbf{G}((BL_{0,0,2} = 0) \wedge \neg(TId_{2,1,1} = 0) \wedge (\neg(BL_{1,0,0} > 0)\mathbf{R}(TId_{1,0,2} = 0))))$ $\Phi = -$	-	-
QuasiCertifProtocol-02	$\varphi = \neg((Astart = 0) \wedge \mathbf{F}\neg(Astart = 0))$ $\Phi = \varphi$	2	1,25
QuasiCertifProtocol-02	$\varphi = \neg(\neg(n8_{1,1} > 0)\mathbf{R}\neg(Cstart_1 > 0))$ $\Phi = \varphi$	1	1,03
QuasiCertifProtocol-02	$\varphi = \neg\mathbf{X}(\neg(Sstart_2 = 0)\mathbf{R}\neg(n7_{1,0} = 0) \wedge \mathbf{XG}(CstopOK_1 > 0))$ $\Phi = \varphi$	3	3,86

F.3. táblázat. Mérési eredmények

Modell	Az ellenőrzött φ és a legerősebb teljesülő Φ kifejezés	MC	Futási idő
QuasiCertifProtocol-02	$\varphi = \neg\mathbf{X}(XG\neg(s2_2 > 0)\mathbf{U}\neg(CstopAbort = 0) \vee \mathbf{F}\neg(n3_0 = 0))$ $\Phi = \varphi$	3	1,10
QuasiCertifProtocol-02	$\varphi = \neg((\neg(a2 > 0) \vee \neg(n6_1 > 0)) \wedge \mathbf{G}(s2_1 = 0))$ $\Phi = \varphi$	3	2,83
QuasiCertifProtocol-02	$\varphi = \neg(\neg(n8_{1,0} > 0)\mathbf{U}(c1_0 = 0)\mathbf{R}\mathbf{X}(n7_{2,0} = 0))$ $\Phi = \varphi$	3	1,08
QuasiCertifProtocol-02	$\varphi = \neg\mathbf{X}\mathbf{X}((s5_0 > 0) \vee \mathbf{X}((n7_{2,2} > 0)\mathbf{U}(n6_2 > 0))\mathbf{U}\neg(n6_2 = 0))$ $\Phi = \varphi$	4	1,99
Railroad-005	$\varphi = \neg((pl_{P9,1} > 0) \wedge \mathbf{F}\neg(pl_{P9,1} > 0))$ $\Phi = \neg((pl_{P9,1} > 0) \wedge \mathbf{F}igaz)$	3	1,12
Railroad-005	$\varphi = \neg(\neg(pl_{P27,1} = 0)\mathbf{R}\neg(pl_{P7,1} = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(pl_{P7,1} = 0))$	1	0,56
Railroad-005	$\varphi = \neg(\neg(pl_{P3,1} = 0) \wedge \mathbf{G}((pl_{P22,1} = 0) \wedge \neg(pl_{P40,1} > 0) \wedge (\neg(pl_{P1,1} = 0)\mathbf{R}(pl_{P41,1} > 0))))$ $\Phi = -$	-	-
Railroad-005	$\varphi = \neg(\mathbf{X}\neg(pl_{P32,1} > 0) \wedge \mathbf{G}(\neg(pl_{P37,1} > 0) \wedge \mathbf{X}(pl_{P17,1} > 0) \vee (pl_{P37,1} > 0) \wedge \mathbf{X}\neg(pl_{P17,1} > 0)))$ $\Phi = -$	-	-
Railroad-005	$\varphi = \neg(\neg(pl_{P3,1} = 0) \wedge \mathbf{G}((pl_{P22,1} = 0) \wedge \neg(pl_{P40,1} > 0) \wedge (\neg(pl_{P1,1} = 0)\mathbf{R}(pl_{P41,1} > 0))))$ $\Phi = -$	-	-
Railroad-005	$\varphi = \neg(\neg(pl_{P3,1} = 0) \wedge \mathbf{G}((pl_{P22,1} = 0) \wedge \neg(pl_{P40,1} > 0) \wedge (\neg(pl_{P1,1} = 0)\mathbf{R}(pl_{P41,1} > 0))))$ $\Phi = -$	-	-
Railroad-005	$\varphi = \neg(\mathbf{X}\neg(pl_{P32,1} > 0) \wedge \mathbf{G}(\neg(pl_{P37,1} > 0) \wedge \mathbf{X}(pl_{P17,1} > 0) \vee (pl_{P37,1} > 0) \wedge \mathbf{X}\neg(pl_{P17,1} > 0)))$ $\Phi = -$	-	-
Railroad-010	$\varphi = \neg\mathbf{G}\neg(pl_{P7,1} = 0)$ $\Phi = \varphi$	0	0,00
Railroad-010	$\varphi = \neg(\neg(pl_{P27,1} = 0)\mathbf{R}\neg(pl_{P7,1} = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(pl_{P7,1} = 0))$	1	1,86
Railroad-010	$\varphi = \neg((pl_{P9,1} > 0) \wedge \mathbf{F}\neg(pl_{P9,1} > 0))$ $\Phi = \neg((pl_{P9,1} > 0) \wedge \mathbf{F}igaz)$	3	2,42

F.4. táblázat. Mérési eredmények

Modell	Az ellenőrzött φ és a legerősebb teljesülő Φ kifejezés	MC	Futási idő
Railroad-010	$\varphi = \neg(\mathbf{G}(pl_P0_1 > 0) \vee \mathbf{G}\neg(pl_P38_1 = 0))$ $\Phi = \varphi$	0	0,01
Railroad-010	$\varphi = \neg((\neg(pl_{P9,1} = 0) \vee \neg(pl_P21_1 > 0)) \wedge \mathbf{G}(pl_{P19,1} = 0))$ $\Phi = \neg\mathbf{G}(pl_{P19,1} = 0)$	12	18,10
Railroad-010	$\varphi = \neg(\neg(pl_{P3,1} = 0) \wedge \mathbf{G}((pl_{P22,1} = 0) \wedge \neg(pl_{P40,1} > 0) \wedge (\neg(pl_{P1,1} = 0)\mathbf{R}(pl_{P41,1} > 0))))$ $\Phi = -$	-	-
Railroad-020	$\varphi = \neg((pl_{P9,1} > 0) \wedge \mathbf{F}\neg(pl_{P9,1} > 0))$ $\Phi = \neg((pl_{P9,1} > 0) \wedge \mathbf{F}igaz)$	3	28,40
ResAllocation-R002C002	$\varphi = \neg((r_{0,0} = 0) \wedge \mathbf{X}\neg(p_{1,1} > 0))$ $\Phi = \neg((r_{0,0} = 0) \wedge \mathbf{X}igaz)$	6	1,69
ResAllocation-R002C002	$\varphi = \neg\mathbf{GF}((p_{0,1} > 0) \wedge \neg(r_{1,1} > 0))$ $\Phi = \neg\mathbf{GF}\neg(r_{1,1} > 0) \vee \neg\mathbf{GF}(p_{0,1} > 0)$	5	1,52
ResAllocation-R002C002	$\varphi = \neg\mathbf{X}(\neg(r_{1,0} > 0)\mathbf{R}\neg(r_{0,1} > 0))$ $\Phi = \neg\mathbf{X}(igaz\mathbf{R}\neg(r_{0,1} > 0))$	3	0,82
ResAllocation-R002C002	$\varphi = \neg\mathbf{X}(\neg(p_{1,0} > 0) \wedge \mathbf{G}(r_{0,0} > 0))$ $\Phi = \neg\mathbf{XG}(r_{0,0} > 0)$	3	0,86
ResAllocation-R002C002	$\varphi = \neg(\neg(r_{0,1} > 0)\mathbf{R}\neg(r_{1,1} > 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(r_{1,1} > 0))$	1	0,47
ResAllocation-R002C002	$\varphi = \neg\mathbf{XX}((p_{0,0} > 0) \vee \mathbf{X}(r_{1,1} > 0)\mathbf{U}\neg(r_{1,1} > 0))$ $\Phi = \neg\mathbf{XX}(igaz\mathbf{U}\neg(r_{1,1} > 0))$	12	3,32
ResAllocation-R002C002	$\varphi = \neg(\mathbf{F}\neg(p_{0,1} > 0) \wedge \mathbf{G}\neg(r_{0,1} > 0) \vee \mathbf{F}(r_{0,1} > 0) \wedge \mathbf{G}(p_{0,1} > 0))$ $\Phi = \neg(\mathbf{F}igaz \wedge \mathbf{G}\neg(r_{0,1} > 0) \vee \mathbf{F}igaz \wedge \mathbf{G}(p_{0,1} > 0))$	17	4,64
ResAllocation-R003C002	$\varphi = \neg\mathbf{GF}((p_{0,1} > 0) \wedge \neg(r_{1,1} > 0))$ $\Phi = \neg\mathbf{GF}\neg(r_{1,1} > 0) \vee \neg\mathbf{GF}(p_{0,1} > 0)$	5	1,58
ResAllocation-R003C002	$\varphi = \neg\mathbf{X}(\neg(p_{1,0} > 0) \wedge \mathbf{G}(r_{0,0} > 0))$ $\Phi = \neg\mathbf{XG}(r_{0,0} > 0)$	3	0,81
ResAllocation-R003C002	$\varphi = \neg((r_{0,0} = 0) \wedge \mathbf{X}\neg(p_{1,1} > 0))$ $\Phi = \neg((r_{0,0} = 0) \wedge \mathbf{X}igaz)$	3	0,87

F.5. táblázat. Mérési eredmények

Modell	Az ellenőrzött φ és a legerősebb teljesülő Φ kifejezés	MC	Futási idő
ResAllocation-R003C002	$\varphi = \neg(\neg(r_{0,1} > 0)\mathbf{R}\neg(r_{1,1} > 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(r_{1,1} > 0))$	1	0,29
ResAllocation-R003C002	$\varphi = \neg\mathbf{X}(\neg(r_{1,0} > 0)\mathbf{R}\neg(r_{0,1} > 0))$ $\Phi = \neg\mathbf{X}(igaz\mathbf{R}\neg(r_{0,1} > 0))$	3	0,82
ResAllocation-R003C002	$\varphi = \neg(\mathbf{F}\neg(p_{0,1} > 0) \wedge \mathbf{G}\neg(r_{0,1} > 0) \vee \mathbf{F}(r_{0,1} > 0) \wedge \mathbf{G}(p_{0,1} > 0))$ $\Phi = \neg(\mathbf{F}igaz \wedge \mathbf{G}\neg(r_{0,1} > 0) \vee \mathbf{F}igaz \wedge \mathbf{G}(p_{0,1} > 0))$	17	4,48
ResAllocation-R005C002	$\varphi = \neg\mathbf{X}\mathbf{G}(r_{0,1} > 0)$ $\Phi = \varphi$	1	0,28
ResAllocation-R005C002	$\varphi = \neg\mathbf{G}\mathbf{F}((p_{0,1} > 0) \wedge \neg(r_{1,1} > 0))$ $\Phi = \neg\mathbf{G}\mathbf{F}\neg(r_{1,1} > 0) \vee \neg\mathbf{G}\mathbf{F}(p_{0,1} > 0)$	5	1,76
ResAllocation-R005C002	$\varphi = \neg\mathbf{X}(\neg(p_{1,0} > 0) \wedge \mathbf{G}(r_{0,0} > 0))$ $\Phi = \neg\mathbf{X}\mathbf{G}(r_{0,0} > 0)$	3	0,85
ResAllocation-R005C002	$\varphi = \neg((r_{0,0} = 0) \wedge \mathbf{X}\neg(p_{1,1} > 0))$ $\Phi = \neg((r_{0,0} = 0) \wedge \mathbf{X}igaz)$	3	0,85
ResAllocation-R005C002	$\varphi = \neg\mathbf{X}(\neg(r_{1,0} > 0)\mathbf{R}\neg(r_{0,1} > 0))$ $\Phi = \neg\mathbf{X}(igaz\mathbf{R}\neg(r_{0,1} > 0))$	3	0,85
ResAllocation-R005C002	$\varphi = \neg(\neg(r_{0,1} > 0)\mathbf{R}\neg(r_{1,1} > 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(r_{1,1} > 0))$	1	0,29
ResAllocation-R005C002	$\varphi = \neg\mathbf{X}\mathbf{X}((p_{0,0} > 0) \vee \mathbf{X}(r_{1,1} > 0)\mathbf{U}\neg(r_{1,1} > 0))$ $\Phi = \neg\mathbf{X}\mathbf{X}(igaz\mathbf{U}\neg(r_{1,1} > 0))$	12	3,22
ResAllocation-R005C002	$\varphi = \neg(\mathbf{F}\neg(p_{0,1} > 0) \wedge \mathbf{G}\neg(r_{0,1} > 0) \vee \mathbf{F}(r_{0,1} > 0) \wedge \mathbf{G}(p_{0,1} > 0))$ $\Phi = \neg(\mathbf{F}igaz \wedge \mathbf{G}\neg(r_{0,1} > 0) \vee \mathbf{F}igaz \wedge \mathbf{G}(p_{0,1} > 0))$	17	4,52
ResAllocation-R020C002	$\varphi = \neg\mathbf{X}\neg(r_{1,0} > 0)$ $\Phi = \varphi$	1	0,30
ResAllocation-R020C002	$\varphi = \neg\mathbf{X}(\neg(r_{1,0} > 0)\mathbf{R}\neg(r_{0,1} > 0))$ $\Phi = \neg\mathbf{X}(igaz\mathbf{R}\neg(r_{0,1} > 0))$	3	0,90
ResAllocation-R020C002	$\varphi = \neg((r_{0,0} = 0) \wedge \mathbf{X}\neg(p_{1,1} > 0))$ $\Phi = \neg((r_{0,0} = 0) \wedge \mathbf{X}igaz)$	3	0,89

F.6. táblázat. Mérési eredmények

Modell	Az ellenőrzött φ és a legerősebb teljesülő Φ kifejezés	MC	Futási idő
ResAllocation-R020C002	$\varphi = \neg(\neg(r_{0,1} > 0)\mathbf{R}\neg(r_{1,1} > 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(r_{1,1} > 0))$	1	0,31
ResAllocation-R020C002	$\varphi = \neg(\mathbf{F}\neg(p_{0,1} > 0) \wedge \mathbf{G}\neg(r_{0,1} > 0) \vee \mathbf{F}(r_{0,1} > 0) \wedge \mathbf{G}(p_{0,1} > 0))$ $\Phi = \neg(\mathbf{F}igaz \wedge \mathbf{G}\neg(r_{0,1} > 0) \vee \mathbf{F}igaz \wedge \mathbf{G}(p_{0,1} > 0))$	17	4,61
ResAllocation-R100C002	$\varphi = \neg\mathbf{X}\neg(r_{1,0} > 0)$ $\Phi = \varphi$	1	0,34
ResAllocation-R100C002	$\varphi = \neg\mathbf{X}(\neg(r_{1,0} > 0)\mathbf{R}\neg(r_{0,1} > 0))$ $\Phi = \neg\mathbf{X}(igaz\mathbf{R}\neg(r_{0,1} > 0))$	3	1,02
ResAllocation-R100C002	$\varphi = \neg(\neg(r_{0,1} > 0)\mathbf{R}\neg(r_{1,1} > 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(r_{1,1} > 0))$	1	0,41
ResAllocation-R100C002	$\varphi = \neg((r_{0,0} = 0) \wedge \mathbf{X}\neg(p_{1,1} > 0))$ $\Phi = \neg((r_{0,0} = 0) \wedge \mathbf{X}igaz)$	3	1,29
ResAllocation-R100C002	$\varphi = \neg(\mathbf{F}\neg(p_{0,1} > 0) \wedge \mathbf{G}\neg(r_{0,1} > 0) \vee \mathbf{F}(r_{0,1} > 0) \wedge \mathbf{G}(p_{0,1} > 0))$ $\Phi = \neg(\mathbf{F}igaz \wedge \mathbf{G}\neg(r_{0,1} > 0) \vee \mathbf{F}igaz \wedge \mathbf{G}(p_{0,1} > 0))$	17	5,61
Ring-none	$\varphi = \neg\mathbf{GF}((P15 = 0) \wedge \neg(P92 > 0))$ $\Phi = \neg\mathbf{GF}\neg(P92 > 0)$	5	10,85
Ring-none	$\varphi = \neg(\neg(P60 = 0)\mathbf{R}\neg(P46 = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(P46 = 0))$	1	0,53
Ring-none	$\varphi = \neg(\neg(P7 > 0) \wedge \mathbf{G}((P15 > 0) \wedge \neg(P74 > 0) \wedge (\neg(P131 > 0)\mathbf{R}(P77 > 0))))$ $\Phi = -$	-	-
Ring-none	$\varphi = \neg(\mathbf{X}\neg(P48 = 0) \wedge \mathbf{G}(\neg(P62 > 0) \wedge \mathbf{X}(P125 = 0) \vee (P62 > 0) \wedge \mathbf{X}\neg(P125 = 0)))$ $\Phi = -$	-	-
SimpleLoadBal-02	$\varphi = \neg(\neg(serverNotification_1 = 0)\mathbf{R}\neg(lbLoad_{2,1} = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(lbLoad_{2,1} = 0))$	1	0,31
SimpleLoadBal-02	$\varphi = \neg((lbLoad_{2,2} > 0) \wedge \mathbf{F}\neg(lbLoad_{2,2} > 0))$ $\Phi = \neg((lbLoad_{2,2} > 0) \wedge \mathbf{F}igaz)$	3	0,82
SimpleLoadBal-02	$\varphi = \neg((\neg(lbLoad_{2,2} > 0) \vee \neg(serverWaiting_1 = 0)) \wedge \mathbf{G}(clientAck_2 > 0))$ $\Phi = \neg\mathbf{G}(clientAck_2 > 0)$	12	3,44

F.7. táblázat. Mérési eredmények

Modell	Az ellenőrzött φ és a legerősebb teljesülő Φ kifejezés	MC	Futási idő
SimpleLoadBal-02	$\varphi = \neg(\mathbf{F}\neg(clientWaiting_1 > 0) \wedge (\neg(serverRequest_{2,2} = 0)\mathbf{U}\neg(serverNotification_2 = 0)))$ $\Phi = \neg(\mathbf{F}igaz \wedge (\neg(serverRequest_{2,2} = 0)\mathbf{U}\neg(serverNotification_2 = 0)))$	12	3,59
SimpleLoadBal-05	$\varphi = \neg(\neg(serverNotification_1 = 0)\mathbf{R}\neg(lbLoad_{2,1} = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(lbLoad_{2,1} = 0))$	1	0,32
SimpleLoadBal-05	$\varphi = \neg((lbLoad_{2,2} > 0) \wedge \mathbf{F}\neg(lbLoad_{2,2} > 0))$ $\Phi = \neg((lbLoad_{2,2} > 0) \wedge \mathbf{F}igaz)$	3	0,85
SimpleLoadBal-05	$\varphi = \neg((\neg(lbLoad_{2,2} > 0) \vee \neg(serverWaiting_1 = 0)) \wedge \mathbf{G}(clientAck_2 > 0))$ $\Phi = \neg\mathbf{G}(clientAck_2 > 0)$	12	3,55
SimpleLoadBal-05	$\varphi = \neg(\mathbf{F}\neg(clientWaiting_1 > 0) \wedge (\neg(serverRequest_{2,2} = 0)\mathbf{U}\neg(serverNotification_2 = 0)))$ $\Phi = \neg(\mathbf{F}igaz \wedge (\neg(serverRequest_{2,2} = 0)\mathbf{U}\neg(serverNotification_2 = 0)))$	12	3,63
SimpleLoadBal-10	$\varphi = \neg(\neg(serverNotification_1 = 0)\mathbf{R}\neg(lbLoad_{2,1} = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(lbLoad_{2,1} = 0))$	1	0,37
SimpleLoadBal-10	$\varphi = \neg((lbLoad_{2,2} > 0) \wedge \mathbf{F}\neg(lbLoad_{2,2} > 0))$ $\Phi = \neg((lbLoad_{2,2} > 0) \wedge \mathbf{F}igaz)$	3	0,97
SimpleLoadBal-10	$\varphi = \neg(\mathbf{F}\neg(clientWaiting_1 > 0) \wedge (\neg(serverRequest_{2,2} = 0)\mathbf{U}\neg(serverNotification_2 = 0)))$ $\Phi = \neg(\mathbf{F}igaz \wedge (\neg(serverRequest_{2,2} = 0)\mathbf{U}\neg(serverNotification_2 = 0)))$	12	4,47
SimpleLoadBal-10	$\varphi = \neg((\neg(lbLoad_{2,2} > 0) \vee \neg(serverWaiting_1 = 0)) \wedge \mathbf{G}(clientAck_2 > 0))$ $\Phi = \neg\mathbf{G}(clientAck_2 > 0)$	12	4,05
SimpleLoadBal-20	$\varphi = \neg(\neg(serverNotification_1 = 0)\mathbf{R}\neg(lbLoad_{2,1} = 0))$ $\Phi = \neg(igaz\mathbf{R}\neg(lbLoad_{2,1} = 0))$	1	0,68
SimpleLoadBal-20	$\varphi = \neg((lbLoad_{2,2} > 0) \wedge \mathbf{F}\neg(lbLoad_{2,2} > 0))$ $\Phi = \neg((lbLoad_{2,2} > 0) \wedge \mathbf{F}igaz)$	3	1,70
SimpleLoadBal-20	$\varphi = \neg(\mathbf{F}\neg(clientWaiting_1 > 0) \wedge (\neg(serverRequest_{2,2} = 0)\mathbf{U}\neg(serverNotification_2 = 0)))$ $\Phi = \neg(\mathbf{F}igaz \wedge (\neg(serverRequest_{2,2} = 0)\mathbf{U}\neg(serverNotification_2 = 0)))$	12	7,48
SimpleLoadBal-20	$\varphi = \neg((\neg(lbLoad_{2,2} > 0) \vee \neg(serverWaiting_1 = 0)) \wedge \mathbf{G}(clientAck_2 > 0))$ $\Phi = \neg\mathbf{G}(clientAck_2 > 0)$	12	6,90
TokenRing-005	$\varphi = \neg(\mathbf{G}(State_{3,1} > 0) \vee \mathbf{G}\neg(State_{2,4} = 0))$ $\Phi = \varphi$	0	0,11

F.8. táblázat. Mérési eredmények

Modell	Az ellenőrzött φ és a legerősebb teljesülő Φ kifejezés	MC	Futási idő
TokenRing-005	$\varphi = \neg((\neg(State_{0,0} = 0) \vee \neg(State_{1,4} > 0)) \wedge \mathbf{G}(State_{4,2} > 0))$ $\Phi = \neg\mathbf{G}(State_{4,2} > 0)$	12	6,76
TokenRing-010	$\varphi = \neg((\neg(State_{0,0} = 0) \vee \neg(State_{1,4} > 0)) \wedge \mathbf{G}(State_{4,2} > 0))$ $\Phi = \neg\mathbf{G}(State_{4,2} > 0)$	12	98,07
TokenRing-015	$\varphi = \neg((\neg(State_{0,0} = 0) \vee \neg(State_{1,4} > 0)) \wedge \mathbf{G}(State_{4,2} > 0))$ $\Phi = \neg\mathbf{G}(State_{4,2} > 0)$	12	779,76

F.9. táblázat. Mérési eredmények