Szaturációalapú tesztbemenet-generálás színezett Petri-hálókkal

Darvas Dániel, Vörös András
Méréstechnika és Információs Rendszerek Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem
Budapest, Magyarország
darvas.daniel@inf.mit.bme.hu, vori@mit.bme.hu

***Kivonat* — Különböző informatikai rendszerek robusztusságtesztelése fontos feladat, amelyben fontos szerepe van a megfelelő tesztbemenetek előállításának. Ezen a területen is bizonyították hasznosságukat az ún. modellellenőrző eszközök, ugyanis megfelelő paraméterezéssel hatékonyan képesek a kívánt tesztbemeneteket előállítani. A cikkben bemutatjuk, hogy az aszinkron rendszerek ellenőrzésére hatékony szaturációs algoritmus hogyan használható akár komplex tesztkövetelményeknek is megfelelő tesztbemenetek generálására. Megközelítésünk szemléltetésére két alkalmazási példát is felvázolunk.**

Kulcsszavak — szaturáció, tesztgenerálás, nyomgenerálás, korlátos állapottér-felderítés, színezett Petri-háló

I. BEVEZETŐ

Hardver- és szoftverrendszerek vizsgálata, helyességének és teljességének vizsgálata régóta aktív kutatási terület. Ennek egyik lehetséges módja az, hogy elkészítjük a rendszert leíró modellt, majd azt szimulációval vagy matematikai módszerekkel elemezzük.

A modellek leírására számos formalizmus rendelkezésre áll. Jelen cikkben színezett Petri-háló formalizmussal megadott modelleket vizsgálunk. Számos alkalommal használták már sikeresen ezt a formalizmust különböző rendszerek vagy akár kommunikációs protokollok leírására és vizsgálatára is. Így elemezték például a MAC-réteget [1], mobil ad-hoc hálózati protokollokat [2], a TCP-protokollt [3] vagy például a BGP-protokollt [4]. A formális módszerek használatának köszönhetően több esetben is kimutattak hibákat és nem kellőképp specifikált elemeket, akár régóta használatban lévő protokollok esetében is.

A korábban publikált, színezett Petri-hálókat alkalmazó vizsgálatok megegyeztek abban, hogy a verifikációra explicit technikákat alkalmaztak, azaz minden elérhető állapotot különkülön reprezentáltak. Nagyméretű modellek esetén a modellellenőrzés világából ismert, hogy különböző szimbolikus technikák segítségével jóval nagyobb állapotterű modellek is vizsgálhatók [15].

Aszinkron rendszerek szimbolikus modellellenőrzésére egy hatékony módszer az ún. *szaturációalapú modellellenőrzés*. A szaturációs algoritmusok [9][10][11] előnyét két tulajdonságuk biztosítja: (1) döntési diagramokon alapuló adatreprezentációjuk főként aszinkron modellek esetén kompakt tárolást nyújt, (2) speciális iterációs stratégiájuk – megfelelő paraméterezés esetén – az aszinkronitás kihasználásával gyors állapottérfelderítést tesz lehetővé.

Jelen munkában a szaturációalapú algoritmusok alkalmazhatóságát vizsgáljuk nyomgenerálásra (állapottérben történő útgenerálásra). Ennek motivációja az, hogy a nyomgenerálás eredménye bizonyos esetekben közvetlenül vagy közvetve felhasználható tesztbemenetként. Különösen igaz ez a robusztusságtesztelés esetén, ahol szükség lehet arra, hogy a rendszert egy meghatározott szélsőséges érték felvételére kényszerítsük, ilyen állapotba tudjuk vinni.

Hasonló, modellellenőrzőn alapuló tesztgeneráló megoldások léteznek, lásd pl. [12][1]. A jelen cikkben ismertetett megoldás előnye, hogy képes több tesztkövetelményt egyszerre kielégítő tesztbemeneteket generálni. További előnye, hogy a szaturációs algoritmust használja, és amennyiben a verifikáció is ennek segítségével történik, a tesztbemenet generálása nem okoz jelentős többletköltséget (sem időben, sem memóriában).

A cikk felépítése a következő: a II. fejezetben bemutatjuk a módszerekhez szükséges alapismereteket: a színezett Petri-hálókat és a modellellenőrzést. A III. fejezetben ismertetjük az egylépéses nyomgenerálást, amit közvetlenül tesztgenerálásra és összetettebb tesztgenerálás építőköveként használunk. A IV. fejezet a többlépéses nyomgenerálásról szól, amely segítségével szaturációs alapokon lehetséges összetett tesztbemeneteket generálni. Alkalmazási példákkal szemlélteti a módszer használhatóságát az V. fejezet.

II. HÁTTÉRISMERETEK

A. Színezett Petri-háló (CPN)

A színezett Petri-hálók [14] az egyszerű Petri-hálók kibővítései adatszerkezetekkel. Egy színezett Petri-háló formálisan egy $CPN = (P, T, E, \Sigma, C, G, A, M_0^c)$ struktúra, ahol P a helyek (véges) halmaza, T a tranzíciók (véges) halmaza, $E \subseteq (P \times T) \cup (T \times P)$ az élek halmaza, Σ az adattípusok (színosztályok) halmaza, $C: P \to \Sigma$ a színfüggvény, G az egyes tranzíciókhoz őrfeltételt rendelő függvény, A az egyes élekhez élkifejezést rendelő függvény, M_0^c pedig az egyes helyekhez kezdeti tokeneloszlást rendelő függvény [8]. Mivel az ismertetett algoritmusok nem közvetlenül színezett Petri-hálókon dolgoznak, ezek bővebb ismertetésétől itt eltekintünk. Azonban ezen formalizmus használata fontos tulajdonság, hiszen az

algoritmusaink eredménye a színezett Petri-hálóval leírt magas szintű modellre lesz visszavetítve.

B. *Modellellenőrzés*

A *modellellenőrzés* egy verifikációs módszer, melynek során egy temporális logikai kifejezés teljesülését vizsgáljuk egy formális modellen. Eredménye egy olyan állapothalmaz, amelyre igaz a megadott temporális logikai kritérium. Ha ebben szerepel a modell egy kezdőállapota, akkor a modellezett rendszerre is igaz lesz az állítás [6].

A modellellenőrzés tipikusan két fázisból áll: az állapottér felderítéséből és a megadott kritérium ellenőrzéséből. *Állapottér* alatt most egy olyan irányított gráfot értünk, amely csúcsai a modellben elérhető állapotokat, élei pedig állapotok közti lehetséges átmeneteket jelölik. Jelölje a továbbiakban $\mathcal{N}(s)$ az s állapotból egy átmenettel elérhető állapotokat, azaz az s-nek megfelelő csúcsból kimenő élek végén lévő csúcsok halmazát. A szaturációs algoritmusok a modellellenőrzés mindkét fázisának végrehajtására alkalmazhatók.

A szaturációs állapottér-felderítés eredménye két adatstruktúra: az elérhető állapotok S halmaza és a lehetséges állapotátmeneteket leíró reláció (azaz lényegében $\mathcal{N}$ függvény). Így a szaturációalapú állapottér-felderítés az állapotteret egy éllistás formátumban adja meg. Mindkét adatstruktúra kódolt formában, döntési diagramok segítségével kerül reprezentálásra.

A modellellenőrzés egy speciális módja a *korlátos modellellenőrzés*. Ilyenkor a kritériumellenőrzést megelőzően nem a teljes állapottér kerül felderítésre, csak ennek egy meghatározott B mélységű része (azaz az állapottérnek azon része, amely legalább egy kezdőállapottól legfeljebb B állapotátmenettel elérhető). Amennyiben a B mélységű állapottérrészből a kritérium teljesülése nem dönthető el, nagyobb B értékekkel iteratív módon folytatódik az állapottér-felderítés.

Korábban bemutatták, hogy szaturációs algoritmussal is lehetséges korlátos állapottér-felderítést végezni [5]. Ez alapján korábbi munkánkban bemutattunk egy módszert szaturációalapú korlátos modellellenőrzésre [8]. Ezen módszerek lényege az, hogy az állapotok halmazában eltároljuk azt is, melyik állapot milyen távolságra található a kezdőállapot(ok)tól. Ez megfelelő típusú döntési diagramot használva megtehető [5]. A jelen cikkben ismertetett algoritmusok építenek a [8]-ban leírt megoldásokra, de egy új felhasználási területet céloznak meg.

III. Egylépéses nyomgenerálás

Az *egylépéses nyomgenerálás* feladata a következő: adott egy s_0 kezdőállapot és egy $S' \subseteq S$ állapothalmaz. Keressünk egy olyan $U = (s_0, s_1, \dots, s_m)$ állapotsorozatot az állapottérben, ahol az egyes állapotok szomszédosak ($\forall i \in \{1,2,\dots,m\}: s_i \in \mathcal{N}(s_{i-1})$) és $s_m \in S'$. Azaz a feladat s_0-ból egy $s_m \in S'$ állapotba vezető út keresése az állapottérben.

Annak érdekében, hogy az így generált utak (nyomok) könnyen használhatóak legyenek, célszerű azt is kikötni, hogy a generált $U = (s_0, s_1, \dots, s_m)$ állapotsorozat a lehető legrövidebb legyen, azaz s_0 kezdőállapotból legrövidebb utat keresünk az állapottér gráfjában S' egy tetszőleges elemébe.

Egyszerű Petri-hálókra már ismertettek korábban szaturációalapú nyomgeneráló algoritmust [5]. Ennek lényege az, hogy az s_0 kezdőállapotból felderítjük a teljes állapotteret úgy, hogy minden egyes $s \in S$ állapothoz eltároljuk az s_0 állapottól mért $\delta(s)$ távolságukat is. (Azaz az algoritmus egy távolságinformációkkal bővített teljes állapottér-felderítést végez.)

A nyomgenerálás módszere azon az észrevételen alapul, hogy a $(s_0, s_1, \dots, s_m)$ állapotsorozat minden s_i elemére $\delta(s_i) = i$, hiszen az egymás melletti állapotok távolsága egymástól eggyel térhet el a legrövidebb útban. Ha az állapotsorozat egy s_j elemére találnánk j-nél rövidebb utat, akkor az s_j-t megelőző $j-1$ darab állapot helyettesíthető lenne kevesebb állapottal, így rövidebb utat kapnánk, ami ellentmondás. Ha az állapotsorozat egy s_j elemébe j-nél hosszabb út vezetne, akkor valamely egymás melletti s_k, s_{k+1} állapotokra a távolságkülönbség egynél nagyobb lenne, ami viszont nem lehetséges, ha $s_{k+1} \in \mathcal{N}(s_k)$, azaz s_{k+1} egy s_k-t követő állapot.

Ez alapján az algoritmus úgy működik, hogy kiválasztja a megadott S' halmazból azt az s' állapotot, amelyre $\delta(s') = m$ a legkisebb. Ezután keres egy tranzíciót, amellyel ebbe az s' állapotba lehetett jutni, majd megkeresi azt az s_{m-1} állapotot, ahonnan az adott tranzíció tüzelése s'-be vezetett. Ilyen s_{m-1} állapot biztosan létezik, hiszen különben s' nem lenne m távolságra a kezdőállapottól. Ha ilyenből több is van, tetszőleges megválasztható s_{m-1}-nek, hiszen mindegyikhez vezet $m-1$ hosszú út a kezdőállapotból. Az algoritmus ezt folytatja rekurzívan a kezdőállapot eléréséig.

Gyakran az állapotok sorozatánál informatívabb az állapotátmenetek sorozata, amelyek a Petri-hálóbeli tranzíciók tüzeléseinek felelnek meg. Az algoritmus során elmenthetjük azt is, hogy egy s_k állapotból melyik t tranzíció (inverz) hatására sikerült az őt megelőző s_{k-1} állapotba jutni. Így a kimenet az $U = (s_0, s_1, \dots, s_m)$ állapotsorozat helyett ekvivalensen lehet egy $F = (t_1, \dots, t_m)$ tüzelési sorozat is (ahol $\forall i: t_i \in T$), így pedig a nyomgenerálás eredménye közvetlenül visszavetíthető a Petri-hálóval leírt modellre.

Vegyük azonban észre, hogy az útgenerálás nem érint olyan $s \in S$ állapotokat, amelyekre $\delta(s) > m$. Így az útgeneráláshoz nem szükséges az állapottér ezen része. Emiatt a [5]-ban ismertetett távolságinformációval bővített állapottér-felderítés nem szükséges, a mi megvalósításunkban helyettesítettük egy korlátos állapottér-felderítéssel, amelynek leállási feltételként az S' halmaz tetszőleges elemének elérését adjuk.

További észrevétel, hogy bár a színezett Petri-hálók szaturációs analízise ugyanígy felderíti (és igény szerint távolságadatokkal ellátja) az állapotok halmazát, az útkereső algoritmus működésében lehet eltérés. Ezt az okozza, hogy míg egyszerű Petri-hálók esetén egy s állapotból egy t tranzíció tüzelése pontosan egy s' állapotba vihet át, addig színezett Petri-hálók esetén több ilyen s' állapot is létezhet (mivel a bemenő és kimenő élkifejezéseknek több különböző engedélyezett változóbehelyettesítése is lehet). Erre a [5]-ben ismertetett algoritmus nincs felkészítve. Emiatt a jelen algoritmusokhoz használt megvalósítást megfelelő visszalépéses kereséssel ki kellett egészítenünk: a lehetséges tranzíciókat és azok lehetséges változóbehelyettesítéseit sorra meg kell vizsgálnunk egészen addig, amíg nem találunk egy olyat, amellyel egy $\delta(s') - 1$ távolságú állapotból engedélyezett a tranzíció.

IV. TÖBBLÉPÉSES NYOMGENERÁLÁS

Jelen fejezetben az általunk elkészített többlépéses nyomgenerálást mutatjuk be. A *többlépéses nyomgenerálás* feladata a következő: adott követelmények egy $R = \{r_1, r_2, \dots, r_n\}$ halmaza. Minden egyes r_i követelményt az S állapothalmaz bizonyos állapotai elégítenek ki. Keressünk egy olyan $U = (s_0, s_1, \dots, s_m)$ állapotsorozatot az állapottérben, ahol az egyes állapotok szomszédosak ($\forall i \in \{1,2, \dots, m\}: s_i \in \mathcal{N}(s_{i-1})$) és az R követelményhalmaz minden r_k eleme valamely $s_k \in U$ állapotban ki van elégítve.

Az egyes $r \in R$ követelmények tetszőlegesek lehetnek, az egyetlen feltétel az, hogy kiértékelhető legyen egy állapothalmazra. Így, felhasználva a szaturációs modellellenőrzés lehetőségeit, ezek lehetnek atomi vagy összetett logikai kifejezések, de akár temporális logikai kifejezések is.

A cél az, hogy az egylépéses nyomgenerálás felhasználásával megoldást adjunk a többlépéses nyomgenerálás problémájára. Ha az egylépéses nyomgeneráláshoz hasonlóan itt is kikötjük, hogy a cél a legrövidebb út generálása, akkor a probléma speciális esete (amennyiben $n = |S|$ és minden $r_i \in R$ pontosan egy $s_i \in S$ állapotra teljesül) egy Hamilton-út keresése lenne az állapotteret leíró gráfban, ami NP-nehéz probléma. Emiatt optimális megoldást nem várunk el az algoritmustól. (Ezt a felhasználási cél – a tesztbemenet-generálás – sem indokolja.)

További nehézség, hogy az egylépéses nyomgenerálás során csak egy kezdő s_0 állapotból számoljuk ki a többi $s \in S$ állapot távolságát, nem ismert tetszőleges két állapot közt a távolság. Így mindössze azt állapíthatjuk meg, hogy melyik a legközelebbi állapot, amely egy $r \in R$ követelményt kielégít, tehát a rendelkezésre álló adatok alapján csak egy mohó útkereső algoritmust készíthetünk.

Könnyen belátható, hogy egy mohó útkereső algoritmus – mivel lokális optimumot keres – nem feltétlen a legrövidebb utat találja meg. Emellett az is elmondható, hogy mohó útkereséssel általános esetben a megfelelő út sem feltétlenül található meg. Előfordulhat ugyanis, hogy még azelőtt „zsákutcába”, kivezető élekkel nem rendelkező csúcsba jut az állapottérben, mielőtt a megtalált állapotsorozat kielégíti az összes R-beli követelményt.

Vezérlők és protokollok és tervezése esetén azonban tipikus tervezési cél az, hogy ne legyen holtpont (azaz az állapottér gráfjában minden csúcsból vezessen ki él), továbbá a modell ciklikus (megfordítható) legyen, tehát minden állapotából vissza lehessen térni a kezdőállapotba (azaz az állapottér gráfjában minden csúcsból vezessen irányított út a kezdőállapotba), vagy legalább egy inicializálás utáni állapotba. Ilyen esetekben az algoritmus garantáltan talál megoldást.

A leírt algoritmusok használhatóságára adható egy elégséges feltétel. Jelölje S_R az összes olyan állapotot, amely kielégít legalább egy $r \in R$ feltételt egy adott R követelményhalmazból. Az adott R alapján a többlépéses nyomgenerálás elégséges feltétele, hogy minden $s \in S_R \cup \{s_0\}$ állapotból elérhető legyen minden $s' \in S_R$ állapot. Ez esetben a mohó algoritmus tetszőleges sorrendben elégítheti ki a követelményeket, minden esetben folytatható lesz a nyomgenerálás. Amennyiben a Petri-háló megfordítható, ez triviálisan teljesül.

Az eddigiekben leírtaknak megfelelő algoritmust mutat az 1. ábra. Megjegyzendő, hogy ez az algoritmus nem közelít multiplikatív hibával, az optimális és az adott megoldás közti különbségre nem adható felső korlát.

Adott: $R = \{r_1, r_2, \dots, r_n\}$ követelmények; s_0 kezdőállapot

$s \coloneqq s_0$; $U \coloneqq ()$
Ciklus amíg $R \neq \emptyset$
  Keressünk s állapotból korlátos állapottér-felderítéssel olyan s' állapotot, amelyre R valamelyik eleme igaz.
  $U' \coloneqq$ legrövidebb út (állapotsorozat) s-ből s'-be
  $s \coloneqq s'$
  $U \coloneqq U \circ U'$ // $\circ$: konkatenáció
  $R \coloneqq R \setminus \{r \in R \mid r$ igaz s állapotban$\}$
Ciklus vége

Eredmény: U állapotsorozat

1. ábra Többlépéses nyomgenerálás algoritmusa

Könnyen definiálható a többlépéses nyomgenerálásnak egy szigorúbb variánsa, a *sorrendezett többlépéses nyomgenerálás*. Ekkor a megtalált útra nem csak annak kell teljesülnie, hogy minden R-beli követelményt kielégít, hanem annak is, hogy ezeket egy meghatározott sorrendben elégíti ki. Pontosabban: $R = \{r_1, \dots, r_n\}$ követelmények és $U = (s_0, \dots, s_m)$ generált út esetén $\forall r_j \in R$-hez kell egy olyan s_i állapotnak léteznie U-ban, hogy (1) s_i állapot kielégíti r_j-t és (2) az $\{r_1, \dots, r_{j-1}\}$ követelmények mindegyikét kielégíti az $\{s_0, \dots, s_{i-1}\}$ állapotok valamelyike. Az előzőekben (1. ábra) ismertetett algoritmus az alább leírtak szerint alakítható át, hogy a sorrendezési követelménynek is eleget tegyen:

Adott: $R = \{r_1, r_2, \dots, r_n\}$ követelmények; s_0 kezdőállapot

$s \coloneqq s_0$; $U \coloneqq ()$
Ciklus $i = 1$-től n-ig
  Ha s állapotban nem igaz r_i:
    Keressünk s állapotból korlátos állapottér-felderítéssel olyan s' állapotot, amelyre r_i igaz.
    $U' \coloneqq$ legrövidebb út (állapotsorozat) s-ből s'-be
    $s \coloneqq s'$
    $U \coloneqq U \circ U'$ // $\circ$: konkatenáció
  Elágazás vége
Ciklus vége

Eredmény: U állapotsorozat

2. ábra Sorrendezett többlépéses nyomgenerálás algoritmusa

Sorrendezett esetben ahhoz, hogy garantáltan megoldást adjon az algoritmus, gyengébb feltétel is elégséges. Biztosan megoldást ad az algoritmus, ha minden $r_i \in R$-et kielégítő állapotból minden $r_{i+1} \in R$ követelményt kielégítő állapot elérhető, továbbá a kezdőállapotból elérhető minden $r_1 \in R$-et kielégítő állapot.

A sorrendezett algoritmusra szintén igaz, hogy nem multiplikatív hibával közelítő algoritmus.

Az itt leírt algoritmust (illetve az ehhez szükséges további algoritmusokat) implementáltuk a BME MIT Hibatűrő Rendszerek Kutatócsoport által fejlesztett PetriDotNet keretrendszerben [16].

V. Alkalmazási példák

A. Alkalmazás az R3-COP projektben

Jelen munka motivációját az R3-COP Artemis projekt adta. E projektben automatizált raktárak működését vizsgálják, amelyekben lézerrel tájékozódó járművek (Laser Guided Vehicles, LGV) közlekednek egy központi forgalomvezérlő számítógép által irányítva. Ez a központi számítógép biztosítja, hogy az LGV-k egymást elkerülő utakon közlekedjenek, az LGV-k felelőssége mindössze annyi, hogy a számukra allokált útról ne térjenek le. A projekt során a központi számítógép és az LGV-k közti kommunikációs protokollt vizsgáljuk: a bemutatott nyomgenerálási algoritmusok segítségével robusztusságtesztekhez generálunk tesztbemeneteket színezett Petri-háló modell alapján.

Mivel a projekt idevonatkozó részei nem publikáltak egyelőre, részletes elemzésre nincs módunk, viszont a teljesítményigény érzékeltethető. Kipróbáltuk a módszerünk működőképességét a vizsgált probléma egy összetett modelljére. Erre a modellre (amely állapottere $2{,}44 \cdot 10^9$ állapotot tartalmaz) 13,31 másodperc volt a kritériumellenőrzés és többlépéses nyomgenerálás futása együttesen egy 4 közbenső állapotot tartalmazó, összesen 63 hosszúságú tesztbemenetet adó esetben. A tesztbemenet előállításának többletköltsége a megadott kritérium ellenőrzését követően nem haladta meg az 1 másodpercet.

B. PRISE védelmi logika

A bemutatott módszert kipróbáltuk a korábbi munkánk során vizsgált vezérlőre is. A vizsgált, ún. PRISE védelmi logikát a Paksi Atomerőműben jelenleg is használják bizonyos üzemzavarok detektálására és a megfelelő vészhelyzetet elhárító lépések megindítására. A védelmi logika működéséről és vizsgálatáról bővebben értekeztünk [7]-ben.

Jelen cikkben ismertetett algoritmusok szempontjából a használt modell releváns tulajdonsága, hogy eddig csak szaturációs algoritmusokkal sikerült az állapotterét feltérképezni, más szimbolikus vagy explicit módszerekkel nem.

E védelmi logika vizsgálatánál fontos kérdés lehet, hogy milyen bemenetkombinációkkal érhető el a vészhelyzeti akció kiváltása, amelyre az egylépéses nyomgenerálás választ tud adni. Bonyolultabb elvárások is lehetnek a bemenetekkel kapcsolatban, például hogy úgy történjen vészhelyzeti akció, hogy előtte már a veszélyes szituáció egyes elemei már fennálltak, de később megszűntek. Vizsgálható az is, hogy egy vészhelyzeti akció után milyen bemenetek hatására lehet ezt az akciót megszüntetni. Ezen kérdésekre mind válasz adható a többlépéses nyomgenerálással, majd az így előállított tesztbemenetekkel a védelmi logika valódi implementációja is megvizsgálható.

Az ismertetett tesztbemenet-generálási feladatokat a rendszer egy kb. 10^{10} állapotot tartalmazó modelljén vizsgáltuk. Az leghosszabb bemenet generálása 71 másodpercet vett igénybe.

VI. Összefoglalás

A cikkben egy módszert ismertettünk, amely segítségével a szaturációalapú korlátos modellellenőrzést felhasználva az állapottérben utak generálhatók. A generált utak közvetlenül is felhasználhatók tesztbemenetekként, például robusztusságtesztelés esetén. Emellett bemutattunk egy módszert arra, hogy az egylépéses nyomgenerálás alapján többlépéses nyomgenerálás is végezhető (a közbenső lépések sorrendjének megadásával és sorrendezés nélkül egyaránt), amellyel komplex tesztbemenetek is generálható a formális modellek alapján. A módszer előnye, hogy formális verifikációval együtt alkalmazva alacsony költséggel elvégezhető. A bemutatott algoritmusokat implementáltuk a PetriDotNet keretrendszer részeként és felhasználtuk az R3-COP Artemis projekt keretében.

VII. Hivatkozások

[1] P.H. Morera, T.M.P. Gonzalez. “A CPN model of the MAC layer”, in Proc. of the 2nd Workshop on Practical Use of Coloured Petri Nets and Design/CPN, 1999, pp. 153–172.

[2] L.M. Kristensen, K. Jensen. “Specification and validation of an Edge Router Discovery Protocol for mobile ad-hoc networks”, in Integration of Software Specification Techniques for Application in Engineering, Springer-Verlag, 2004, pp. 248–269.

[3] J.C.A. de Figueiredo, L.M. Kristensen. “Using coloured Petri nets to investigate behavioural and performance issues of TCP protocols”, in Proc. of the 2nd Workshop on Practical Use of Coloured Petri Nets and Design/CPN, 1999, pp. 21–40.

[4] D.A. Zaitsev. “Verification of protocol BGP via decomposition of Petri net model into functional subnets”, in Proc. of the Design, Analysis, and Simulation of Distributed Systems Symposium, 2005, pp. 72–78.

[5] G. Ciardo and R. Siminiceanu. “Using edge-valued decision diagrams for symbolic generation of shortest paths”, in Proc. Fourth International Conference on Formal Methods in Computer-Aided Design (FMCAD), LNCS 2517, Springer-Verlag, Berlin, 2002, pp. 256–273.

[6] E.M. Clarke. “The birth of model checking”, *25 years of model checking*, Springer-Verlag, Berlin, 2008, pp. 1–26.

[7] T. Bartha, A. Vörös, A. Jámbor, D. Darvas. “Verification of an industrial safety function using coloured Petri nets and model checking”, in 14th International Conference on Modern Information Technology in the Innovation Processes of the Industrial Entreprises, 2012, pp. 472-485.

[8] A. Vörös, D. Darvas, T. Bartha. “Bounded saturation-based CTL model checking”, *Proceedings of the Estonian Academy of Sciences*, vol. 62 (issue 1), 2013, pp. 59–70.

[9] G. Ciardo, G. Lüttgen, R. Siminiceanu. “Saturation: An efficient iteration strategy for symbolic state-space generation”, in Proc. of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2001, pp. 328–342.

[10] G. Ciardo, R. Marmorstein, R. Siminiceanu. “The saturation algorithm for symbolic state-space exploration”, *International Journal on Software Tools for Technology Transfer*, vol. 8 (issue 1), 2006, pp. 4–25.

[11] G. Ciardo, Y. Zhao, X. Jin. “Ten years of saturation: A Petri net perspective”, in Transactions on Petri Nets and Other Models of Concurrency, LNCS vol. 6900, Berlin: Springer, 2012, pp. 51–95.

[12] G. Fraser, F. Wotawa, P.E. Ammann. “Testing with model checkers: A survey”, *Software Testing, Verification & Reliability*, vol. 19 (issue 3), 2009, pp. 215–261.

[13] S. Rayadurgam. “Coverage based test-case generation using model checkers”, in Proc. of the 8th Annual IEEE Int. Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2001), 2001, pp. 83–91.

[14] K. Jensen, L.M. Kristensen. Coloured Petri Nets, Berlin: Springer, 2009.

[15] E.M. Clarke, O. Grumberg, D.A. Peled. Model checking, Cambridge: MIT Press, 1999.

[16] A PetriDotNet keretrendszer honlapja. (elérve: 2013. március 15.) http://petridotnet.inf.mit.bme.hu/