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

A szaturáció algoritmus kiterjesztése és általánosítása modellellenőrzéshez

Tézisfüzet

Molnár Vince

Témavezető:
Majzik István, Ph.D. (BME)

Budapest, 2019.

1. Előzmények

1.1. A kutatás motivációi

1985 és 1987 között a Therac-25 sugárterápiás berendezésben található szoftverhiba következtében hat beteg halt meg vagy szenvedett súlyos sérüléseket kezelés közben [LT93]. A probléma két konkurens folyamat helytelen szinkronizációjából fakadt, amely egy versenyhelyzethez vezetett, ha az operátor bizonyos parancsokat túl gyorsan adott ki. A versenyhelyzetek detektálása tesztelés közben nagyon nehéz, mert csak nagyon ritkán jelennek meg. A versenyhelyzetek jó példái a nemdeterminisztikus viselkedésnek, amely egy matematikai absztrakció megjósolhatatlan, de lehetséges kimenetelek leírására.

A Therac-25-höz hasonló incidensek megmutatták, hogy a szoftverek fejlesztésében – főleg konkurens szoftverek esetén – nagyon könnyű hibát véteni, a hibákat pedig nagyon nehéz detektálni hagyományos tesztelési megközelítésekkel. Eddigre a formális módszerek a számítástechnikában már szilárd matematikai logikai alapokkal rendelkeztek, ami lehetővé tette, hogy a mérnökök a terveiket matematikailag precíz módszerekkel specifikálják, modellezzék és ellenőrizzék [BH14]. A formális verifikáció célja, hogy egy formális specifikáció és egy modell segítségével *bizonyítsa* a modellezett viselkedés helyességét a specifikáció tekintetében – szemben a teszteléssel, amely csak néhány kiválasztott végrehajtást vizsgál. A Therac incidensek idejében publikálta korszakalkotó munkáját a konkurenciával összefüggő problémák detektálásának témakörében Edmund M. Clarke és Ernest A. Emerson [EC80], illetve Joseph Sifakis [QS82]: megszületett a *modellellenőrzés* technikája.

Egy számítástechnikai rendszer egy adott (absztrakt) viselkedése reprezentálható egy *diszkrét állapotsorozattal*, ahol az állapotok a rendszert írják le adott időpillanatban. A rendszer egy adott ideig egy állapotban marad, majd (lényegében pillanatszerűen) új állapotot vesz fel (ez az állapotátmenet). Ilyen módon a rendszer teljes viselkedése leírható egy gráffal (ahol az állapotok mint csúcsok, az állapotátmenetek pedig mint élek szerepelnek), amelyben az egy állapotból kimenő állapotátmenetek nemdeterminisztikus döntéseket jelölnek (pl. a fent bemutatott konkurens folyamatok ütemezésénél). Minden út (állapotsorozat) ebben a gráfban egy lehetséges végrehajtása a rendszernek, amely bizonyos körülmények között megvalósítható. Emiatt bármely potenciális hiba valahol a gráfban is jelen van mint nem kívánt állapot vagy állapotsorozat. A modellellenőrzés célja ennek az állapotgráfnak (vagy más néven, állapottérnek) az előállítása, majd a rendszertulajdonságok logikai formulák által leírt specifikációja alapján a hibák feltárása.

Nyilvánvalóan minél összetettebb a rendszer, annál több állapota van, és sajnos ez a kapcsolat általában exponenciális – ez a jelenség *állapottér robbanás problémaként* ismert. A gyakorlatban egy realisztikus rendszermodellnek annyi állapota lenne, hogy képtelenség lenne mindet egy számítógép memóriájában tárolni. A *szimbolikus modellellenőrzés* [Bur+92] erre a problémára kínál megoldást.

A szimbolikus modellellenőrzés legfőbb motivációja, hogy az állapotok általában a rendszerváltozók által felvett értékekből képzett vektorok, és ezek közül az állapotvektorok közül az állapottérben sok a hasonló (kiváltképp konkurens rendszerekben). Például egy szál egy programban nagy valószínűséggel csak a lokális változóit és néhány megosztott globális változót módosít, de nincs ráhatással más szálak lokális változóira. A szimbolikus modellellenőrzés ezt használja ki, amikor állapotvektorok halmazait Boole-függvényként (karakterisztikus függvény) adja meg, amely a halmazbeli vektorok közös tulajdonságait és kapcsolatait írja le. Például az $f(x, y, z) = \neg x \vee y$ függvény az x, y és z változók felett 6 vektort ír le: igazat ad vissza azon hármasok esetén, amelyek kielégítik a $\neg x \vee y$ kifejezést. A modellellenőrzéshez szükséges alapvető halmazműveletek Boole-operátorokra képződnek le (az unió diszjunkcióra, a metszet konjunkcióra, a komplemensképzés pedig negálásra).

Randal Bryant a Boole-függvények kompakt reprezentálására javasolta a bináris *döntési diagramok* [Bry86] használatát. Ezek lényegében olyan döntési fák, amelyekben az azonos részfákat

összevonjuk. A döntési diagramok manipulációja logikai műveletek végrehajtására nagyon hatékonyan végrehajtható rekurzió és gyorsítótár segítségével, mivel az egyesített részfákat csak egyszer kell feldolgozni.

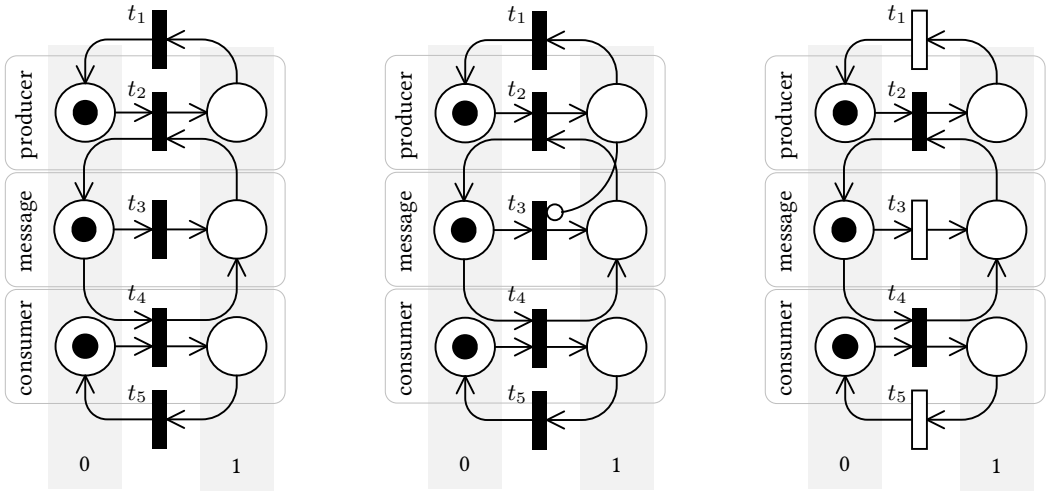
A döntési diagramok rekurzív természete inspirált egy új modellellenőrző algoritmust, amely szélességi bejárás (Breadth-First Search, BFS) vagy mélységi bejárás (Depth-First Search, DFS) használata helyett a döntési diagram struktúráját követi, és rekurzívan számítja ki az állapotgráfot absztrakt részmodellek egyre bővülő sorozatán keresztül: az algoritmus a részmodellek feldolgozásakor felhasználja a korábban feldolgozott (kisebb) részmodellek eredményeit. Az algoritmus nagyon jól illeszkedik a döntési diagramokhoz, és gyorsan és memóriahatékony módon képes felépíteni az elérhető rendszerállapotok halmazát reprezentáló döntési diagramot. Az algoritmus célja, hogy a döntési diagram aljától felfelé haladva csomópontonként bővítse az összes elérhető állapottal a rendszer kezdőállapotait elkódoló döntési diagramot. Az elnevezés ebből a megközelítésből fakad – *szaturáció*, vagyis telítés [CLS01; CMS06].

A szaturáció hatékonyságához szükség van egy kritikus tulajdonságra, amelyet lokalitásnak nevezünk. A lokalitás fogalma azon alapszik, hogy a rendszer állapotának (egy esemény hatására történő) változásakor csak néhány rendszerkomponens vesz fel új állapotot, a legtöbb komponens állapota nem változik. Ha az események lokálisak, az állapottér felderítése dekomponálható kisebb lépésekre, amelyek csak a komponensek részhalmazát leíró és az azokon lokális eseményeket tartalmazó részmodelleket derítenek fel. A szaturáció ezt a stratégiát egy egymást sorban tartalmazó részmodell-sorozat segítségével követi: a döntési diagram egy csomópontjának szaturálásakor csak azokat a változókat veszi figyelembe, amelyeknek az értékei szükségesek az adott részdiagram kiértékeléséhez (azokat pedig nem, amelyek értékei az adott csomóponthoz vezettek), és a csak ezeket figyelembe véve is elérhető állapotokat számítja csak ki.

A szaturáció az egyik leghatékonyabb döntési diagram-alapú szimbolikus modellellenőrző algoritmusnak bizonyult, különösen konkurens rendszerek esetében. Elsődlegesen Petri-háló modellekre [CLS01] tervezték, és eleinte kihasználta az állapotátmenetek úgynevezett Kroenecker tulajdonságát (amely egy nagyon egyszerű reprezentációt biztosít), de a későbbi fejlesztések megszüntették ezt a követelményt [CMS06]. Az algoritmus egy különálló variánsa a *vezérelt szaturáció* [ZC09], amely képes a felderítést egy előre definiált állapothalmazon (kényszerhalmaz) belül tartani a modell állapotátmeneteinek módosítása nélkül (ez a probléma például olyankor áll elő, amikor egy állapotból *visszafelé* kell keresni egy már felderített állapottérben). Azzal az egyszerű megoldással, hogy egyszerűen zárjuk ki a kényszerhalmazt elhagyó átmeneteket az állapotátmeneti relációkból, sajnos teljesen megszüntetnénk az események eredeti lokalitását, ugyanis általános esetben az összes állapotváltozót ismerni kell ahhoz, hogy a kényszerhalmazban maradást ellenőrizni tudjuk. Vezérelt szaturáció esetén az algoritmus a kényszerhalmazt külön kezeli, és ebben a speciális esetben képes továbbra is kiaknázni az eredeti állapotátmenetek lokalitását.

Az eddig megoldott problémákon felül sok nyitott kérdés van a szaturáció algoritmussal kapcsolatban. Habár a vezérelt szaturáció megoldja a kényszerhalmazokkal kapcsolatos problémákat, összességében a globális vagy egymástól függő állapotátmenetek kezelése a szaturáció számára nehéz feladat. Ilyen jellegű függőségeket kell kezelni például prioritásokkal rendelkező állapotátmenetek esetén. Ebben az esetben egy állapotátmenet végrehajtásakor azt is meg kell vizsgálni, hogy lehetséges lenne-e más, nagyobb prioritással rendelkező állapotátmenetet is végrehajtani. Ez globálissá teszi az állapotátmenetet olyan szempontból, hogy az összes állapotváltozót figyelembe kell venni, ami a többi állapotátmenet számára fontos. Prioritással rendelkező állapotátmenetek szerepelnek például az Általános Sztochasztikus Petri-Hálókban (Generalized Stochastic Petri Nets) [Chi+93], ami egy népszerű modellezési formalizmus sztochasztikus viselkedésű rendszerek modellezésére. Ebben a kontextusban a szimbolikus modellellenőrzésnek nemcsak hatékony állapottér felderítést kell biztosítania, de numerikus megoldók számára egy megfelelő reprezentáció előállítását is támogatnia kell.

Egy másik kihívás a szaturáció számára a lineáris temporális logikai (LTL) tulajdonságok



(a) Egyszerű verzió. (b) Tiltó éllel. (c) Időzített tranzíciókkal.

1. ábra. Három Petri-háló modell, mindegyik egy termelő-fogyasztó (producer-consumer) rendszer variánst ír le egy (egyetlen üzenetet tároló) pufferrel. Mindegyik komponenst egy Boole-változó írja le, amely 0 értéket vesz fel, ha a baloldali hely van jelölve (tartalmaz tokent), illetve 1 értéket, ha a jobboldali.

modellellenőrzése. Egy LTL kifejezés a rendszer (végtelen) végrehajtásait írja le, amely minden esetben Büchi automatává fordítható. A Büchi automata a véges automaták egy végtelen szavakat elfogadó változata. Az LTL tulajdonságok modellellenőrzésekor komponáljuk a rendszermodellt az azt monitorozó Büchi automatával, ami a negált LTL kifejezésre illeszkedő viselkedést keres. Ha az automata elfogadja a rendszer egy végrehajtását, akkor bizonyítékunk van arra, hogy a tulajdonság megsérthető (hiszen az automata a negált tulajdonságot írta le). Mivel az automata általában a legtöbb állapotváltozót olvassa, a szinkronizált állapotátmenetek itt is elveszítik a lokalitásukat. Egy másik megoldandó kérdés, hogy miképpen lehet az explicit állapotgráf-alapú modellellenőrzőkhöz hasonlóan szimbolikus modellellenőrzésnél is felderítés közben keresni a sértő lefutást – ezt „menet közbeni” modellellenőrzésnek hívjuk, és lehetővé teszi, hogy az ellenpéldát adott esetben a rendszer teljes állapotterének felderítése nélkül találjuk meg.

Általánosságban elmondható, hogy a szaturáció nem túl erős olyan rendszerek esetében, amelyeknél az események nem elég lokálisak. Ezekben az esetekben a szaturáció könnyen BFS vagy DFS bejárássá degradálódik, aminek eredményeképp nagyobb méretű köztes döntési diagramok jöhetnek létre, ami nagyobb erőforrás felhasználáshoz és rosszabb hatékonysághoz, skálázódáshoz vezet. Az ebben a munkában bemutatott kutatás emiatt a szaturáció kiterjesztésére fókuszál, célul kitűzve, hogy az algoritmus ezeket az eseteket is hatékonyabban kezelje. A kutatás eredményeként kiderült, hogy a legtöbb ilyen jellegű kiterjesztés mögött ugyanaz az ötlet áll, és ez sok modellosztály esetén felhasználható magának a szaturációnak a továbbfejlesztésére, valamint általánosít sok, ezelőtt különböző algoritmusnak tekintett variációt.

1.2. Háttérismeretek

A bemutatott eredményeinek megértéséhez az olvasónak rendelkeznie kell a szaturáció algoritmus és az LTL modellellenőrzés működésével kapcsolatos alapvető ismeretekkel. Ez a fejezet egy vezérpéldán keresztül mutatja be a fő fogalmakat. Ugyanez a példa illusztrálja majd az új algoritmusokat is.

Petri-hálók. Az 1. ábra három Petri-háló modellt [Mur89] mutat be egy termelő-fogyasztó (producer-consumer) rendszerről, amelyben a komponensek üzenetet válthatnak egymással. A

baloldali modell (1a) egy egyszerű verzió, amelyben mind a termelő, mind a fogyasztó két-két *hellyel* rendelkezik (ezeket a körök jelölik), két lehetséges állapotot jelölve: *kész* és *elfoglalt* a termelő és fogyasztó esetén, illetve *feldolgozatlan* és *feldolgozott* az üzenet (message) esetén. Amikor egy helyet *token* jelöl (fekete belsejű kör), az azt jelenti, hogy a komponens az adott állapotban tartózkodik.

Az állapotok *tranzíciók* (téglalapok) végrehajtásával változhatnak. Az *élek* mutatják, hogy egy tranzíció *tüzeléskor* mely helyekről vesz le és mely helyekre állít elő tokeneket (alapértelmezett esetben egy tokent). Egy tranzíció tüzelése egy viselkedés végrehajtását jelenti, amely során a háló *jelölése* (a tokenek eloszlása a helyeken) az éleknek megfelelően változik. Egy tranzíció *engedélyezett*, ha a bejövő élek forráshelyein megfelelő számú token áll rendelkezésre. Egy engedélyezett tranzíció bármikor tüzelhet, így nemdeterminisztikus viselkedés is létrejöhet, ha egy időben több tranzíció is engedélyezett.

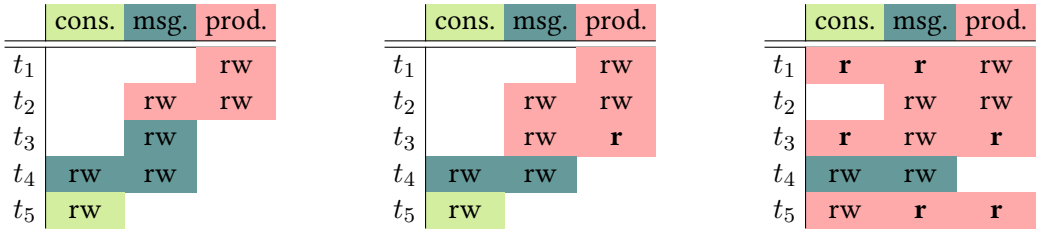
A fentieknek megfelelően a példamodellnek 5 lehetséges viselkedése van: t_1 és t_5 visszavisz egy *elfoglalt* termelőt vagy fogyasztót a *kész* állapotba; t_2 során egy *kész* termelő nyugtáz egy *feldolgozott* üzenetet és kiküld egy új, *feldolgozatlan* üzenetet, majd *elfoglalttá* válik; t_4 során hasonlóan a tesz a fogyasztó, miközben feldolgoz egy *feldolgozatlan* üzenetet; végül t_3 egy időtúllépést reprezentál, amikor egy *feldolgozatlan* üzenet spontán *feldolgozottá* válik. Ezen tranzíciók alapján látszik, hogy minden komponensnek a rendszer minden állapotában pontosan egy helye lesz jelölt, emiatt minden komponens egy Boole-változóként kezelhető: a változó 0, ha a baloldali hely jelölt, és 1, ha a jobboldali.

A középső modell (1b) kiterjeszti a rendszert egy *tiltó éllel* (olyan él, amely egy kis körben végződik, nem nyílban). Egy tiltó él *letilt* egy tranzíciót, ha a forráshely jelölt (alapértelmezett esetben egy tokennel). Emiatt ebben a variánsban időtúllépés csak akkor történhet, ha a termelő *kész* állapotban van, például azért, mert a termelő folyamat követi nyomon az időtúllépéseket, és csak akkor veszi észre őket, amikor *kész* állapotban van.

A harmadik modell (1c) az Általánosított Sztochasztikus Petri-Háló (Generalized Stochastic Petri Net, GSPN) [Chi+93] formalizmust használja, amely a Petri-háló formalizmus kiterjesztése probabilisztikus tüzeléssel és idővel. Amíg az egyszerű Petri-hálók csak a tranzíciók tüzelésének sorrendjét modellezik, a GSPN-ek folytonos időmodellel rendelkeznek, amelyben minden tranzíció tüzelése egy precízen definiált időpillanatban történik. Ebben a formalizmusban két tranzíciótípust különböztetünk meg: *azonnali* tranzíciókat, amelyek az előzőleg bemutatott tranzícióknak felelnek meg és rögtön tüzelnek, amint engedélyezetté válnak (nemdeterminisztikus sorrendben), illetve *időzített* tranzíciókat (üres téglalapok), amelyek egy *tüzelési ráta* segítségével sorsolják, hogy engedélyezetté válásuk után mennyi idővel tüzelnek. A tranzíció típusok megkülönböztetésének eredményeképp az azonnali tranzíciók *mindig* az engedélyezett időzített tranzíciók előtt tüzelnek, ezáltal ezekben a modellekben a tranzíciók *prioritással* rendelkeznek.

Ezt figyelembe véve, a harmadik modell időzítési információt hordoz a termelő és a fogyasztó tevékenységeiről: időt töltenek az üzenetek létrehozásával és feldolgozásával (*elfoglalt* állapot), és az időtúllépés is egy adott idő letelte után történik meg. Ehhez képest az új üzenetek kibocsátása (t_2) és feldolgozása (t_4) azonnal megtörténik, amikor csak lehet. Az implicit prioritások egy következménye, hogy például egy időtúllépés nem történhet meg, ha a fogyasztó kész feldolgozni egy beérkezett üzenetet.

Lokalitás. A példamodell alapjaiban 3 aszinkron konkurens komponensből áll, amelyek alkalmanként szinkronizálnak. Emiatt a legtöbb tranzíció csak néhány komponenst érint: ezt nevezzük *lokalitásnak*. A 2. ábra bemutatja a példarendszer 3 variánsának a függőségi mátrixát. A mátrix sorai a tranzíciók, az oszlopok a komponensek. Ha egy cella üres, akkor a tranzíció független a komponenstől, míg az *r* és *w* karakterek *olvasás* és *írás* függőségeket jelölnek (Petri-hálókban tiltó élek és teszt élek okoznak olvasási függőséget, minden más él egy olvasási-írási függőséget jelöl). A második és a harmadik variáns elsőhöz viszonyított további függőségei vastag betűvel



(a) Egyszerű verzió. (b) Tiltó éllel. (c) Időzített tranzíciókkal.

2. ábra. Függőségi mátrixok (dependency matrix – DM) a 3 példa variánshoz. Az *r* és *w* betűk az *olvasás* és/vagy *írás* függőségeket jelölik, az egyszerű modelltől való eltérések vastag betűvel vannak jelölve. A színek a legmagasabb komponenst jelölik, amely nem független a tranzícióktól (ezt az információt a szaturáció algoritmus használja).

vannak jelölve.

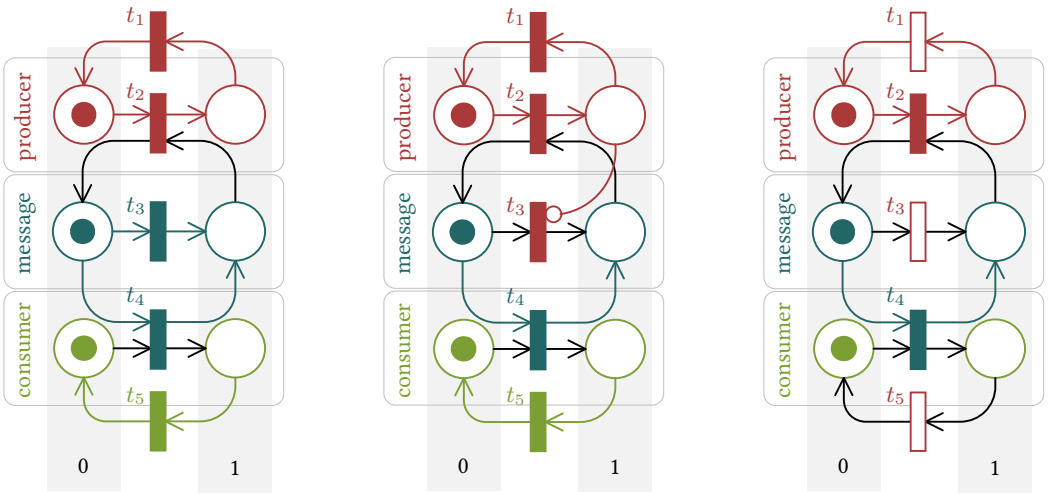
Példa a különbségre, hogy a második variánsban szereplő tiltó él egy (csak) olvasás függőséget vezet be a t_3 és a termelő komponens között, míg az azonnali és az időzített tranzíciók (t_1, t_3 és t_5) között fennálló implicit prioritások szintén (csak) olvasási függőségekhez vezetnek az időzített tranzíciók és minden komponens között, amelyet egy azonnali tranzíció olvas (hiszen ellenőrizni kell, hogy bármelyik azonnali állapotátmenet engedélyezett-e). Érdemes megfigyelni, hogy ezek a függőségek általánosságban a tranzíciók *lokalitásának csökkenéséhez* vezetnek.

Modellellenőrzés. A modellellenőrzés egy automatizált formális verifikáció módszer, amelynek célja bebizonyítani, hogy a részletes rendszermodell tényleg modellje-e a specifikációjának – más szavakkal, hogy a rendszermodell *helyes* adott követelmények tekintetében [CGP99]. A modellellenőrzés egyik kritikus lépése az állapottér-felderítés. A rendszereket általában nem az állapotgráfjukkal modellezzük, hanem egy magas szintű modellel, amely *elkódolja* az állapotteret (például Petri-hálóval). A modellellenőrzés az állapottéren dolgozik, ezért a legtöbb megközelítés magában foglal valamilyen állapottér-generálást. Egy hírhedt probléma ebben a megközelítésben az ún. *állapottér robbanás*: gyakori a kombinatorikus robbanás jelensége, amikor egy magas szintű modellről az állapottérére lépünk át.

Követelmények tekintetében a lineáris temporális logika (LTL) egy népszerű formalizmus a kívánt viselkedés temporális (logikai időbeli) aspektusainak leírására [Pnu77]. Az LTL képes különböző követelmények, például „fairség” specifikálására: pl. a rendszerben minden folyamatot fair módon kell ütemezni, azaz egyik sem maradhat örökké ütemezésre kész állapotban. Az *ellenpéldákat* specifikáló negált LTL tulajdonságok a követelmény ellenőrzéséhez Büchi automatává alakítjuk, amely egy végtelen sorozatokat feldolgozó véges állapotú automata [Büc62]. Egy ilyen automata felhasználható annak a monitorozására, hogy a rendszer viselkedése hogyan befolyásolja a tulajdonság teljesülését: ha az automata végtelen sokszor veszi fel valamelyik *elfogadó állapotát*, akkor a sorozat egy ellenpélda.

Az LTL kifejezések vizsgálata véges állapotú rendszerekben elérhető körök keresésére redukálható: a rendszermodell és a Büchi automata kombinált állapotterében kell olyan elérhető kört keresni, amely tartalmaz legalább egy *elfogadó* állapotot [VW86]. Explicit, gráf alapú modellellenőrzők esetén a felderítés befejeződik amikor egy megfelelő kört találunk, így ilyen esetekben az ellenpélda detektálás gyorsan történik. Általánosságban a cél elérhető, *erősen összefüggő komponensek* (strongly connected component – SCC) keresése, mert egy SCC-ben minden állapot elérhető bármely másikból.

Döntési diagramok. Egy komponens alapú rendszer állapotát általában vektorként reprezentáljuk, ahol minden komponenshez (vagy állapotváltozóhoz) egy érték tartozik a vektorban. Ha a



(a) Egyszerű verzió. (b) Tiltó éllel. (c) Időzített tranzíciókkal.

3. ábra. A szaturációban használt, egyes komponenseknek megfeleltetett részmodellek. A színek a tranzíciók részmodellekhez rendelését jelölik. A hozzárendelésért felelős élek (amelyek a magasabban lévő komponenssel való függőségért felelnek) szintén színezettek. Érdemes megfigyelni, hogy a prioritások által behozott függőségek nem láthatók (ld. a 2. ábra a függőségi mátrixokkal).

tranzíciók lokálisak, akkor minden tüzelés csak néhány ilyen érték megváltozását eredményezi. Például az egyszerű variánsban az időtúllépés tranzíció (t_3) tüzelésekor a kezdeti $(0, 0, 0)$ állapotvektor a $(0, 1, 0)$ vektorra változik, ami csak az üzenet állapotában különbözik.

A *döntési fa* egy hatékony adatstruktúra azonos prefixekkel rendelkező vektorok halmazának reprezentálására, amelyben minden csomópont (a gyökértől kezdve) egy változóhoz van rendelve, és az (irányított) kimenő élek a változó értékeit jelölik. Ha egy vektor értékei alapján végigkövetünk egy utat a gyökértől kezdve gyerekcsomópontokon keresztül egy levélcsomópontig, akkor elkódolhatjuk a levélben, hogy a döntési fa által reprezentált halmaz tárolja-e a vektort, (**1**) vagy sem (**0**). Ily módon a tartalmazott vektorok közös prefixei csak egyszer szerepelnek az adatstruktúrában, ami a kompaktabb reprezentációhoz vezet.

Ha a közös szuffixeket is ki szeretnénk használni a kompaktáláshoz, akkor egyesíthetjük a döntési fa azonos részfáit is. Így egy pontosan két „levéllel” rendelkező döntési diagramot kapunk: a terminális egyes (**1**) és nullás (**0**) csomópontokat [Bry86; MD98]. Ebben a munkában feltételezzük, hogy a csomópontok a döntési diagramban minden úton ugyanazt a *változósorrendet* követik, és minden úton minden változó kiértékelésre kerül, ha az út nem a **0** csomópontba vezet. Következésképpen minden, a gyökérből a **1** csomópontba vezető út egy a reprezentált halmazban tárolt vektorral értékeivel van címkézve, a változók meghatározott sorrendjének megfelelően.

Szaturáció. Míg a döntési diagramok általában egy nagyméretű halmaz kompakt reprezentálására alkalmasak, a halmaz kiszámítása – az állapottér felderítése – továbbra is erőforrásigényes lehet. A modellellenőrzésben használt *szaturáció algoritmus* közvetlenül döntési diagramokon dolgozik és kiaknázza a tranzíciók lokalitását: a problémát rekurzívan kisebb problémákra osztja, és gyorsítótárazza a részszámítások eredményeit, hogy elkerülje a felesleges számításokat [CLS01; CMS06].

A részletek ismertetése nélkül, a szaturáció ötlete egyszerű és elegáns. A szaturáció önmagában egy függvény, amely bemenetként vár egy olyan döntési diagram csomópontot, ami elkódolja a kezdőállapotokat, illetve az állapotátmeneti reláció egy hasonlóan rekurzív reprezentációját (pl. egy másik döntési diagramot), és visszaad egy olyan döntési diagram csomópontot, ami elkódolja a kezdőállapotból egy (potenciálisan üres) tranzíció tüzelési sorozattal elérhető állapotok

halmazát. A függvény definíciója rekurzív, ami a következő megfigyelésen alapul.

A döntési diagram gyökér csomópontjának minden gyerek csomópontja elkódolja egy olyan másik modell kezdőállapotainak a halmazát, amit a gyökércsomóponthoz tartozó állapotváltozó (a *legfelső* változó) és az attól függő tranzíciók elhagyásával kaphatunk. Mivel ez a lépés egy *"muszáj" absztrakció*, minden, ami megtörténhet az absztrakt részmodellben, megtörténhet az eredetei modellben is. Emiatt, ha rekurzívan felderítjük ennek a modellnek az állapotterét minden kezdőállapot halmazzal, amelyet a gyökércsomópont gyerekei elkódolnak, akkor csak azok a tranzíciók nem kerülnek tüzelésre a felderítés során, amelyek a legfelső változótól függnek. A szaturáció függvénynek tehát ezeket a tranzíciókat kell kimerítően tüzelnie. Az utolsó feladat minden tranzíció tüzelése után annak az ellenőrzése, hogy keletkezett-e új kezdőállapot a részmodellben – ha igen, ezeket rekurzívan fel kell deríteni, ami további tüzeléseket engedélyezhet a szaturáció függvény aktuális futásában.

A szaturáció ereje abból a stratégiából fakad, hogy felosztja a modellt kisebb részmodellekre és az egyszerűbb problémák megoldását gyorsítótárazza is, hogy elkerülje a felesleges számításokat. Ennek a stratégiának a hatékonysága a tranzíciók lokalitásától, illetve a változók sorrendezésétől függ. Jelen munka az előbbi problémával foglalkozik, míg az utóbbi probléma aktív kutatás alatt áll [Amp+19]. A lokalitás hatását a részmodellekre a 3. ábrán lévő színek szemléltetik.

1.3. Kihívások összefoglalása

Challenge 1: A szaturáció kiterjesztése prioritásos Petri-hálók hatékony kezelésére. Az Általánosított Sztochasztikus Petri-Háló (Generalized Stochastic Petri net, GSPN) egy népszerű formalizmus sztochasztikus rendszerek modellezésére. A GSPNek állapotterének hatékony felderítése fontos részfeladata a sztochasztikus analízisnek, amely során a rendszer extrafunkcionális tulajdonságait ellenőrizzük (pl. teljesítmény vagy megbízhatóság). Vajon a szaturáció hatékonysága prioritásos Petri-hálók esetén is kiaknázható?

Challenge 2: A szaturáció kiterjesztése lineáris temporális logikai (LTL) tulajdonságok hatékony modellellenőrzésére. LTL tulajdonságok modellellenőrzésekor a rendszerviselkedést a tulajdonságot leíró Büchi automatával kell komponálni. A szaturáció hatékonyságához meg kell őrizni a lokalitást, ehhez pedig el kell kerülni a szinkronizált állapotátmeneti relációk direkt számítását. Vajon lehetséges úgy módosítani a szaturációt, hogy a komponált rendszer állapotterének számítása a rendszer állapotterének számításával egyszerre történjen, miközben a lokalitást is kihasználjuk?

Challenge 3: Állapottér felderítés közbeni elfogadó (azaz a tulajdonságot megsértő) végrehajtások keresése szaturációval. LTL tulajdonságok modellellenőrzésekor az automata által elfogadott végrehajtásokat (azaz ellenpéldákat) kell keresni a rendszermodell és a negált tulajdonsághoz tartozó Büchi automata szinkron szorzatában. Véges állapottérben egy elfogadó futás mindig egy „lasszó", azaz egy út, amely egy körhöz vezet. Emiatt ez a probléma az állapotérben elérhető erősen összefüggő komponensek (strongly connected components, SCC) keresésére redukálható. Az explicit állapotgráf-alapú modellellenőrzők erre az állapottér felderítése közben is képesek, és azonnal megállnak, amint egy elfogadott végrehajtást találnak. Vajon van rá mód, hogy létrehozzunk egy hatékony, állapottér bejárással egyszerre futó SCC detektáló algoritmust és integráljuk a szaturációalapú modellellenőrzésbe?

Challenge 4: A vezérelt szaturáció különböző variánsainak általánosítása. A vezérelt szaturáció egy hatékony módszer a lokalitás visszanyerésére speciális esetekben. Fő ötlete a sok speciális problémát megoldó variánsban megjelenik. Jelenleg ezeket a variánsokat különböző algoritmusokként tartják számon, ami meggátolja a modellellenőrzés különböző aspektusainak szabad kombinálását (pl. modellezési és specifiká-

ciós formalizmusokat). Vajon van rá mód, hogy általánosítsuk ezt az ötletet mint egy absztrakt algoritmust, amelyből a vezérelt szaturáció és más variánsok származtathatók?

Challenge 5: A szaturáció adaptálása globális hatással rendelkező eseményt tartalmazó modellek hatékonyabb kezelése érdekében. Sok modell főleg szinkron jellegű viselkedéssel rendelkezik, amely során minden tranzíciónak figyelembe kell vennie a legtöbb komponenst vagy változót. Ebben az esetben a szaturáció nem képes kiaknázni a lokalitást és kevésbé hatékony felderítési stratégiákká degradálódik. Vajon van rá mód, hogy felhasználjuk a szaturáció ötletét ezekben az esetekben is? Az előálló javulás konkurens aszinkron modellek esetében is megjelenik majd, amelyekben az állapotátmenetek lokálisak és a szaturáció eredetileg is hatékony volt?

A jelen munkában bemutatott kutatás a fenti kihívásokkal foglalkozik, és fő célja, hogy megszüntesse a szaturáció némely limitációját, még hatékonyabb algoritmusokat létrehozva. Ezzel nőne azon problémaosztályok köre, amelyekben a modellellenőrzés a gyakorlatban alkalmazható, így a kutatás elősegítheti az automatikus formális verifikációs eszközök terjedését. Ennek érdekében a disszertáció új tudományos eredményeket mutat be három új szaturáció alapú modellellenőrzési algoritmus formájában: szaturáció prioritásos modellekre kiterjesztve (1. tézis); egy teljes algoritmuscsalád LTL tulajdonságok inkrementális, állapottér felderítés közbeni szaturációalapú modellellenőrzésére (2. tézis); és végül maga a szaturáció algoritmus általánosítása és javítása, amely az eredeti algoritmusba integrálja a felfedezett ötleteket a további hatékonyságnövelés érdekében (3. tézis). Az 1. táblázat bemutatja, hogy az eredmények hogyan kapcsolódnak a leírt kihívásokhoz.

1. táblázat. A tézisek és a kihívások közötti kapcsolatok.

		Kihívás				
		1	2	3	4	5
1. tézis	*A disszertáció 3. fejezete*	•			•	
2. tézis	*A disszertáció 4–5. fejezete*		•	•	•	
3. tézis	*A disszertáció 6. fejezete*				•	•

2. Kutatási módszerek és új eredmények

Kutatási módszerek. A J. N. Amaral által javasolt besorolás szerint [Ama] jelen kutatás az *építő*, *formális* and *kísérleti* metodológiákat követi. Az építő módszerben új dolgot állítunk elő, „demonstrálandó hogy lehetséges” [Ama]. A formális módszerek „tényeket bizonyítanak be algoritmusokról és rendszerekről” [Ama], gyakran az építő módszerrel előállított dolgokról. Végül a kísérleti módszerben egy rendszert célzott kísérletekkel és mérésekkel értékelünk ki konkrét kérdések megválaszolása érdekében.

A disszertáció a matematikai logika, gráfelmélet és automataelmélet jól megalapozott területeire épít, valamint számos korábban bemutatott algoritmusra – ezeket háttérismeretként vagy kapcsolódó irodalomként mutatja be. A formális módszerek területén használt formalizmusok közé tartoznak a Kripke struktúrák, Petri-hálók, Büchi automaták és a lineáris temporális logika. A bemutatott kutatás a G. Ciardo által a szaturáció algoritmuson végzett munka folytatásaként is értelmezhető [CLS01].

Az eredményeket a Model Checking Contest (MCC) [Kor+18] nevű modellellenőrző verseny kiterjedt Petri-háló gyűjteményén értékeltem ki, melyben számos ipari példa is megtalálható a

mesterségesen előállított, a skálázódást vagy speciális esetek kezelésének képességét demonstráló modellek mellett. A legtöbb bemutatott algoritmust a kidolgozásukkor korszerű versenytársaihoz hasonlítottam – ide tartoznak a NuSMV2 [Cim+02], a nuXmv [Cav+14] és az ITS-tools [Dur+11] nevű eszközök.

2.1. Általánosított Sztochasztikus Petri-Hálók elemzése szimbolikus állapottérgenerálással

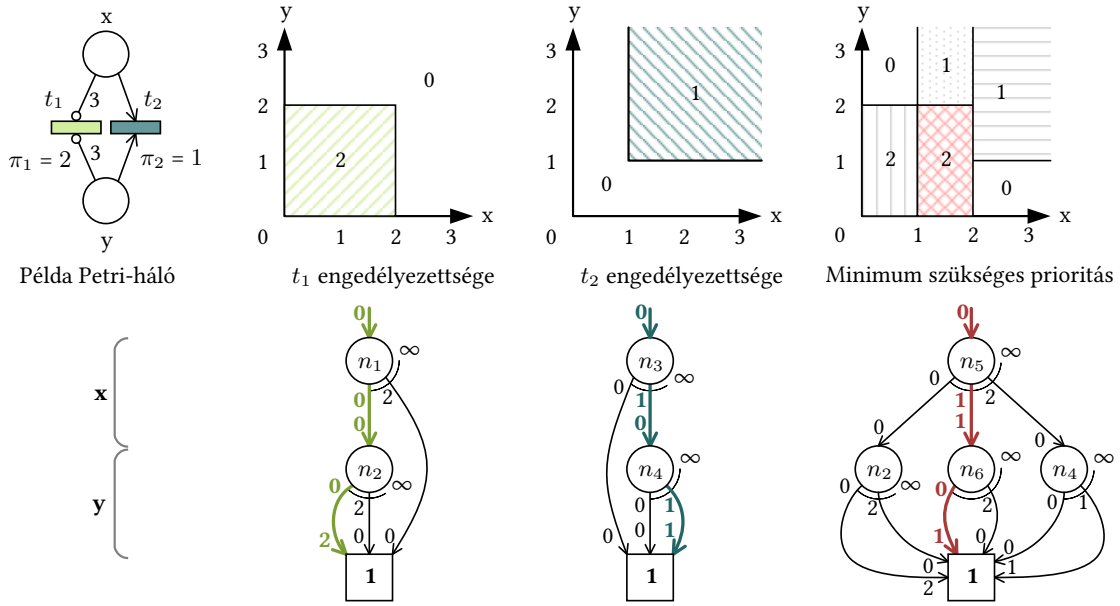
Ahogy az 1.2. fejezetben is megjegyeztük, a GSPN formalizmus bevezeti a tranzíciók közötti prioritás fogalmát a Petri-hálókba. Az időzített tranzícióknak mindig alacsonyabb prioritása van, mint az azonnaliaknak, és a formalizmus azt is lehetővé teszi, hogy a modellező az azonnali tranzíciók között is többszintű prioritásokat adhasson meg. Ennek megfelelően az állapottér felderítés során képesnek kell lennünk több prioritás-szint figyelembe vételére is, ami az 1c. ábrán bemutatott módon a lokalitás csökkenéséhez vezet.

Az állapotátmeneti relációk dekomponálása. Ezen tézispont kulcsgondolata a következő észrevételen alapul. Egy GSPN tranzíció tüzelhetőségét két aspektus határozza meg: az engedélyezettsége (mint a prioritás nélküli Petri-hálóknál) és a legmagasabb prioritás, ami bármelyik, az aktuális jelölés mellett engedélyezett tranzícióhoz tartozik. Prioritásos Petri-hálóknál a legmagasabb prioritás ami adott jelölés mellett engedélyezett tranzícióhoz tartozik statikusan, még az állapottér felderítés megkezdése előtt kiszámítható minden lehetséges jelöléshez. Ez az információ ráadásul független a tranzíció reprezentációjától, így attól külön tárolható: ennek megvalósítására bevezettem az élcímkézett intervallum döntési diagramokat (Edge-Valued Interval Decision Diagrams, EVIDD).

Az az állapottartomány, amiben egy tranzíció engedélyezett, egy (potenciálisan végtelen) hipertéglatest a Petri-háló jelölésinek n-dimenziós terében (n a helyek száma a modellben), és mindegyik ilyen tartomány címkézhető a tranzíció prioritásával. Az egyszerűség kedvéért a 4. ábra felső sora egy két hellyel rendelkező Petri-hálón keresztül illusztrálja a koncepciót: ennek megfelelően a tartományok téglalapok vagy síkrészek a 2-dimenziós térben. Ha adott egy vagy több, egymással esetlegesen átfedő tartományhalmaz, ami leírja a háló tranzícióinak engedélyezettségi tartományát (ebben az esetben t_1 és t_2), akkor az *Unió-Max* művelet segítségével kiszámítható egy új tartományhalmaz, ami pontosan ugyanazokat a pontokat fedi le, mint az eredeti operandusok (a 4. ábra jobb oldala), és az új tartományok címkéi az adott területet fedő eredeti régiók címkéinek maximuma. Azokhoz a tartományokhoz, ahol egyik tranzíció sem engedélyezett, a 0 címkét társítjuk. Az így kapott régióhalmaz megmutatja, hogy mi az a minimális prioritás, amivel adott pontban (állapotban) tüzelhető egy engedélyezett tranzíció.

Élcímkézett intervallum döntési diagramok. Egy EVIDD az élcímkézett döntési diagramok (Edge-valued Decision Diagrams, EDD) [RS10] és az intervallum döntési diagramok (Interval Decision Diagrams, IDD) [Tov08] közötti hibridnek tekinthető: a változók lehetséges értékeit nem felsoroljuk, hanem intervallumokra particionáljuk (mint az IDD-knél), és az élcímként keresztül minden döntés hozzájárul valamennyivel a vektorhoz kiszámolt végső függvényértékhez (mint az EDD-knél, ahol nem bináris értéket, hanem természetes számot rendelünk a vektorokhoz).

Egy EVIDD egy útja a gyökértől a terminális csomópontig képes reprezentálni egy tranzíció engedélyező tartományait az állapottérben, megcímkézve a tranzíció prioritásával: a 4. ábra alsó sorában láthatóak a felettük lévő régiókat reprezentáló EVIDD-k. Nyélnek nevezzük (a címkézett élek nyílhegyeként ábrázolt) a csomópont-*súly* párokat. A súly a régióhoz rendelt szükséges prioritás egy töredéke úgy, hogy az útvonal mentén összegzett súlyok kiadják a tüzeléshez szükséges minimális prioritást. Egy csomópont egy dimenziót reprezentál (ebben az esetben egy egy hely tokenszáma), és rendelkezik egy rendezett éllistával, amely partícionálja a dimenziót: minden él egy olyan intervallumnak felel meg, melynek zart alsó korlátját az él címkéje határozza



4. ábra. A felső sorban láthatóak egy példa prioritásos Petri-háló tranzíciói által meghatározott tartományok a jelölések terében, míg az alsó sorban a tartományoknak megfelelő EVIDD reprezentációk figyelhetők meg. A kiemelt régiókat a kiemelt útvonalak kódolják el a megfelelő EVIDD-ben.

meg, míg nyílt felső korlátja a következő él címkéje (ha van), vagy végtelen (amihez nem tartozik él). Az élek a csomópont gyerek nyeleire mutatnak, amik a következő dimenzió intervallumait írják le, vagy a terminális csomópontba. A súlyokat automatikusan osztjuk el redukciós szabályok segítségével: egy csomópont összes gyereke által birtokolt súlyt fel kell mozgatni a csomópont nyelébe (így mindig lesz egy 0 súlyú gyerek nyél). Az Unió-Max művelet rekurzívan definiálható a nyelek felett és kiszámolja azt az EVIDD-t, ami a fent leírt, tüzeléshez szükséges minimum prioritásokat adja meg diszjunkt régiók egy halmazához. Ezzel a módszerrel egyetlen EVIDD képes leírni a legnagyobb prioritású engedélyezett tranzíció prioritását bármelyik állapotban, ami tehát a legkisebb prioritás is, amivel tranzíció tüzelhető.

Egy ilyen EVIDD segítségével a tüzelhetőséget a szaturáción belül eldönthetjük a minimum szükséges prioritás és az éppen tüzelt tranzíció prioritása alapján. Ehhez párhuzamosan járjuk be az EVIDD-t az állapotteret leíró döntési diagrammal, és ha az útvonal mentén talált súlyok összege meghaladja a tüzelt tranzíció prioritását, akkor azt az állapotátmenetet figyelmen kívül hagyjuk.

Eredmények. Kevés korábbi kutatás érhető el prioritásos modellek szaturációalapú elemzéséről. A. S. Miner és munkatársainak cikke [Min01] egy olyan megoldást javasol, ami a prioritások hatását az állapotátmeneti relációba kódolja, amihez mátrix diagramokat használ (ez egy speciális döntési diagram állapotátmenetek reprezentálására). Mivel ezzel elveszíti a lokalitást, ezért bevezet egy hasító algoritmust, ami újrapartícionálja az állapotátmeneti relációt úgy, hogy az eredmény partíciók a lehető leglokálisabbak legyenek. Az én megközelítésem előnye, hogy megtartja a modell eredeti lokalitását, és a prioritásokkal kapcsolatos információkat a módosított szaturáció egy helyen kezeli, *minden tranzícióhoz közösen*, egy kompakt adatszerkezettel támogatva [c4]. Kísérleti eredményeim alapján az EVIDD-alapú megoldásom jobban skálázódik, mint a [Min01]-ben bemutatott módszer.

Thesis 1 Megterveztem egy algoritmust az Általánosított Sztochasztikus Petri-Hálók (Generalized Stochastic Petri Nets, GSPN) hatékony analíziséhez. Ez az eredeti szaturáció algoritmus kiterjesztése úgy, hogy hatékonyabb legyen az állapottér-generálás prioritásokkal rendelkező tranzíciók esetén.

1.1 Bevezettem egy újfajta döntési diagramot, melyet élcímkézett intervallum döntési diagramnak (Edge-valued Interval Decision Diagram) neveztem el és ami képes hatékonyan reprezentálni prioritizált tranzíciók engedélyezettségét.

1.2 Kiterjesztettem a szaturáció algoritmust úgy, hogy a prioritásos tranzíciókat hatékonyabban kezelje azáltal, hogy a prioritás hatását nem az állapotátmeneti relációban, hanem külön, egy EVIDD segítségével kódoljuk el.

1.3 Kiértékeltem az algoritmus hatékonyságát és megmutattam, hogy a jobban skálázódik a korábban ismert megoldásoknál.

Ezen eredmények jelentőségét az adja, hogy a GSPN-ek hatékony állapottér-felderítése a sztochasztikus analízis egyik szük keresztmetszete lehet, tehát az ezen a területen elért eredmények az egész folyamat hatékonyságát növelhetik. A szaturációalapú állapottér-felderítés nagyobb állapotterek kezelésére képes, és az eredményként adódó döntési diagram arra is felhasználható, hogy a numerikus megoldók számára megfelelő dekompozíciót adjunk az állapottérhez [c5]. A megoldók emiatt szintén nagyobb modellek feldolgozására lesznek képesek [c7], ami összességében jobban skálázódó sztochasztikus analízis módszerekhez vezet, melyekre mindenképpen szüksége van a biztonságkritikus rendszerek extra-funkcionális tulajdonságainak kiértékeléséhez [j1; c6].

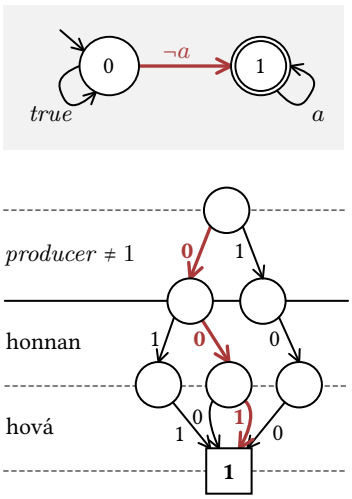
Ebben a kutatásban részt vett Majzik István mint Ph.D. témavezetőm. Az algoritmusok implementálását és a méréseket akkori hallgatóm, Marussy Kristóf végezte, és Vörös András is hozzájárul az eredményekhez hasznos meglátásaival. Az 1. tézis eredményeit a disszertáció 3. fejezete mutatja be. A kapcsolódó publikációk: [j1], [c4], [c5], [c6], [c7].

2.2. Szaturációalapú inkrementális modellellenőrzés lineáris idejű temporális tulajdonságokhoz

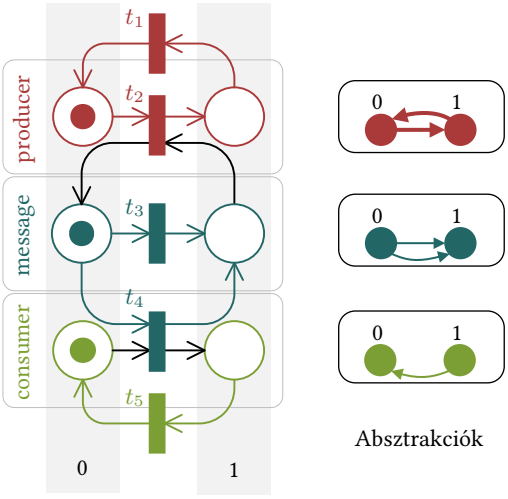
A 2. és 3. kihívások rámutattak, hogy az LTL tulajdonságok szimbolikus modellellenőrzésekor kétféle problémát is meg kell oldanunk: ki kell számítani a rendszermodell és a Büchi automata komponálásával előálló szorzat állapotteret, illetve az ellenpéldákat reprezentáló elfogadó köröket kell keresni ebben az állapottérben.

Komponálás a Büchi automatával. A szorzat állapottér kiszámítása nehéz feladat a szaturáció számára, mert a rendszer minden tranzícióját szinkronizálni kell az automata egy megfelelő állapotátmenetével úgy, hogy a választott átmenetre írt logikai kifejezést igazzá tegye a rendszermodell célállapota. Ez olvasási függőséget okoz az összes tranzíció és az LTL kifejezésben szereplő változók között, ami a lokalitás elvesztéséhez és a szaturáció hatékonyságának csökkenéséhez vezet.

Az erre a problémára adott megoldásom hasonló ötletre épül, mint az 1. tézisben látott módszerek. Mivel a Büchi automata állapottere és állapotátmenetei adottak, előre elkészíthetünk egy adatszerkezetet az automata-állapotátmenetek engedélyezettségének kiértékelésére. Feltételezve, hogy a tulajdonságban használt atomi kifejezések mind egyetlen állapotváltozót hasonlítanak egy konstanshoz (tehát soha nem hasonlítanak össze két változót), ez a struktúra egy véges méretű döntési diagram: a felső szintjei az atomi kifejezésekhez tartozik, melyeket az érintett változók sorrendje szerint sorrendezünk, és itt a 0 (hamis) és 1 (igaz) címkéjű élek a kifejezés igazságértékét jelentik; míg az alsó szintek a Büchi automata állapotátmeneti relációjának részhalmazaiból áll (az 5. ábra illusztrálja ezt a koncepciót). Mindegyik részhalmaz azokat az állapotátmenete-



5. ábra. Egyetlen elfogadó állapottal (dupla kör) rendelkező nemdeterminisztikus Büchi automata, amely a $GF(producer = 1)$ kifejezés negáltját reprezentálja, valamint az állapotátmeneti relációját leíró döntési diagram.



6. ábra. Az egyszerű Petri-háló példamodell a komponensekhez tartozó absztrakciókkal. Vegyük észre, hogy a termelő komponens nélkül nem találhatunk kört az állapottérben, tehát a fixpontszámító algoritmust csak a legfelső szinten kell futtatni.

ket tartalmazza, amelyeket az odavezető úton hamisra vagy igazra kiértékelt atomi kifejezések engedélyeznek.

A szaturáció algoritmust a következőképpen terjesztjük ki. Feltételezzük, hogy a Büchi automata állapotát elkódoló változó a legalsó szinten van. Az algoritmus folyamatosan ki fogja értékelni a szükséges atomi kifejezéseket az imént körülírt döntési diagramban amíg el nem éri ezt az alsó szintet. A modell állapotátmeneti relációja nem érinti az automatát – ehelyett az automatához épített döntési diagram alsó szintjeit használjuk majd az automata léptetésére, ezzel befejezve a komponált rendszer egy lépését. Ezzel a megoldással ismét külön tudjuk kezelni azt az információt, ami képes lenne elrontani a modell eredeti lokalitását, és ez ismét egy minden tranzícióhoz közösen tárolt, a módosított szaturáció által kezelt adatszerkezet lesz.

Ellenpéldák keresése. Erősen összefüggő komponensek (SCC) szaturációalapú keresésére léteznek ismert megoldások (pl. [ZC11]). Ezek általában feltételezik, hogy a felderített állapottér már rendelkezésre áll, és ezen végeznek fixpontszámításokat, ami során iteratívan eltávolítják a „zsákutca” állapotokat amíg vagy nem marad állapot (ekkor nem volt SCC), vagy az állapothalmaz már nem változik tovább (ekkor a maradék állapothalmaz tartalmaz egy SCC-t). Ugyan ez elég lenne az LTL modellellenőrzés kontextusában is, az LTL modellellenőrzők hagyományosan okosabbak ennél, és az állapottér felderítése közben képesek ellenpéldákat keresni, hogy azonnal megálljanak, ha találnak egyet. Ez gyakran sokkal gyorsabb, mint a teljes állapottér szimbolikus felderítése lenne.

A cél tehát az, hogy gyakrabban végezzük fixpontszámításokat az állapottér felderítése közben. Egy olyan megoldást javasoltam, amely jól illeszkedik a szaturáció rekurzív természetéhez: keressünk SCC-ket minden alkalommal, amikor egy döntési diagram csomópont szaturálttá válik, vagyis amikor egy részmodell állapotterét teljesen felderítettük. Ez a probléma hasonló feldarabolásához vezet, mint amit a szaturáció is tesz az állapottér felderítéssel. Ennek a megközelítésnek számos előnye van. Egyrészt észrevehetünk egy SCC-t egy absztrakt részmodellben anélkül, hogy a rendszer többi komponensét vizsgálnánk, ami korábbi ellenpélda-detektáláshoz vezet. Másrészről a számítás inkrementális lehet olyan értelemben, hogy az SCC-nek biztosan

tartalmaznia kell a legutóbbi részmodell legalább egy tranzícióját (hiszen másképp már egy korábbi, kisebb részmodellben is megtaláltuk volna), ami lehetőséget ad a keresési tér redukálására.

Ez az algoritmus azonban jelentős többletmunkával és redundáns számításokkal jár. Ennek kezelésére további két heurisztikát javasoltam, amelyek segítségével olcsón, szimbolikus fixpontszámítás nélkül bizonyítható az SCC-k hiánya.

A *visszatérő állapotok* olyan állapotok, amelyeket többször is elérünk a felderítés során. Ez lehet amiatt, hogy több párhuzamos út is vezetett hozzájuk, de azért is, mert önmagukból elérhetők, tehát egy körben vannak. Akárhogy is, ha nem találunk visszatérő állapotot a bejárás közben, akkor biztosan nem lesz SCC az állapottérben – ebből a megfigyelésből egy olcsó előszűrési lehetőség adódik. A visszatérő állapotokat hatékonyan gyűjthetjük a szaturáció apró módosításával, és felhasználásukkal még tovább szűkíthető a fixpontszámító algoritmus keresési tere is.

Egy erősebb heurisztika az állapottér absztrakcióin alapul. Maga a szaturáció is alkalmaz absztrakt részmodelleket, de ezek még mindig gyakran nagyon nagyok, különösen ahogy egyre több változót vesznek figyelembe. Ezek további absztrahálásával elhagyhatók a részmodell alsóbb változói, de ezúttal azt feltételezzük majd, hogy az alulról elhagyott változók értéke tetszőleges, tehát a részmodell ezekre hivatkozó tranzíciói mindig engedélyezettek lesznek, ha más, ismert változó le nem tiltja őket. Ez egy „lehet” absztrakció, mivel minden amit az absztrakt modell tud végrehajtható lehet az eredeti modellben is, de semmi több. A 6. ábra illusztrálja ezeket az absztrakciókat a példámodell mindhárom komponenséhez. Szűkítsük ezeket az absztrakciókat még tovább azokra az állapotokra, amiket az aktuálisan szaturált csomópont is elkódol (ez könnyen megtehető a csomópont kimenő élei alapján), ezzel megkapjuk az általam javasolt *csómópontszintű absztrakciót.*

Bebizonyítható, hogy ha egy ilyen absztrakcióban nincs SCC (amit már hatékonyan számíthatunk gráfalapú algoritmusokkal is), akkor a döntési diagram által kódolt állapottérben sincs olyan SCC, ami a jelenlegi részmodell tranzícióit használná. Ezek pedig éppen azok az SCC-k lennének, amiket az inkrementális fixpontszámító algoritmus keresne, tehát nincs értelme futtatni azt. Ha viszont van SCC az absztrakcióban, akkor a fixpontszámítás keresési terét még tovább limitálhatjuk az absztrakcióban talált SCC tranzíciói alapján.

Eredmények. Ez a négy komponens (a szorzat állapottér számítása, az inkrementális SCC keresés, a visszatérő állapotok keresése és az absztrakciók) együttesen egy hatékony, állapottér bejárás közbeni, inkrementális és szimbolikus LTL modellellenőrző algoritmust adnak, ami a legtöbb, akkoriban létező hasonló eszköznél jobb teljesítményt mutatott [j2; c8]. Kiterjedt méréseket végeztem a Model Checking Contest (MCC) nevű verseny modellkészletén, ami megmutatta, hogy a javasolt algoritmus gyakran nagyságrendekkel gyorsabb, mint versenytársai, ami jelentős lépés a jó skálázódó, valódi rendszereken is alkalmazható LTL modellellenőrzés felé.

Thesis 2 Megterveztem egy olyan, lineáris idejű temporális logikai kifejezések modellellenőrzésére alkalmas algoritmust, amely az ellenőrzést az állapottér felderítés közben, inkrementálisan végzi el, kiterjesztve a szaturációt a hatékony állapottér felderítéshez és felhasználva a szükséges fixpontszámításokban is.

- 2.1 Kiterjesztettem a szaturáció algoritmust, hogy közvetlenül a rendszermodell és a vizsgált LTL tulajdonságot leíró Büchi automata szinkron kompozíciójának állapotterét számítsa ki. A közvetlen generálás előnye, hogy elkerülhető a szorzat állapotátmeneti reláció explicit kiszámítása.
- 2.2 Terveztem egy algoritmust erősen összefüggő komponensek (SCC) inkrementális keresésére, ami az állapottér felderítést végző szaturációval együtt fut, s felhasználja a szaturációt a fixpontok kiszámításában is. Ez a megközelítés lehetővé teszi az állapottér felderítés közbeni modellellenőrzést, vagyis az algoritmus leállhat, amint talált egy ellenpéldát.

2.3 Bevezettem kétféle heurisztikát amik kiegészíti az inkrementális SCC keresést. Absztrakcióalapú módszerekkel és a visszatérő állapotok alapján a heurisztikák képesek olcsón belátni, hogy a vizsgált részmodellben nincs SCC, így gyorsíthatják a keresést a szükségtelen fixpontszámítások elkerülésével.

2.4 A Model Checking Contest (MCC) verseny modelljein kiértékeltem az adódó LTL modellellenőrző algoritmust, összehasonlítva a futásidejét három, akkoriban koreszerűnek számító eszközzel – az algoritmus gyakran nagyságrendekkel gyorsabb volt a versenytársainál.

Ezen eredmények jelentőségét az adja, hogy a bemutatott LTL modellellenőrző algoritmus gyakran jobban skálázódik a korábban ismert megoldásoknál, míg főbb ötletei ortogonálisak sok más, az irodalomban található továbbfejlesztésre, ami még jobb eredményeket ígér, ha a megoldásokat kombináljuk. Az LTL népszerű formalizmus rendszerek temporális aspektusainak specifikálására, különösen a „fairséggel" kapcsolatos tulajdonságoknál, amiket nem tudunk a másik népszerű formalizmusokban, az elágazó-idejű temporális logika (computation-tree logic, CTL) nyelvén felírni. Emiatt az ebben az irányban történő előrelépés segíthet a formális verifikáció (konkurens) biztonságkritikus rendszereknél történő alkalmazásának szélesebb körű elterjedésében, ahol a fairség kritikus része vagy előfeltétele lehet a vizsgált rendszertulajdonságoknak. Az algoritmust a PetriDotNet keretrendszerben[1] implementáltam [j1; c6].

Ebben a kutatásban részt vett Vörös András és Bartha Tamás mint B.Sc. és M.Sc. konzulenseim. Darvas Dániel segített a szaturáció kezdeti megértésében és ő írta azt a kódot, ami az implementációm alapjául szolgált. A 2. tézisben bemutatott eredmények egy része a szakdolgozatomban és diplomamunkámban is megjelenik [a19; a18]. A 2. tézis eredményeit a disszertáció 4. és 5. fejezete tárgyalja. A kapcsolódó publikációk a következők: [j1], [j2], [c6], [c8].

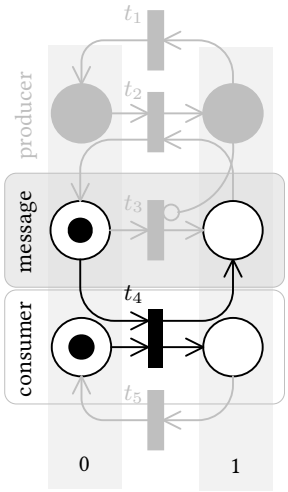
2.3. A szaturáció algoritmus továbbfejlesztése és általánosítása

Az előző két tézisben bemutatott kutatás felfedett néhány közös problémát, amik különböző kontextusban újra és újr megjelennek: *1)* a különböző állapotátmenet reprezentációk (többek között a különböző fajta döntési diagramok) általában egy speciális szaturáció-variánssal járnak együtt, emiatt egymással nem kompatibilisek; valamint *2)* a lokalitás elvesztése mindegyik variánsnál központi kérdés, iközben általában van valamilyen megoldás amivel a lokalitás egy része megőrizhető.

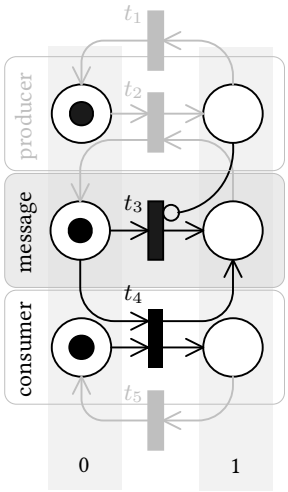
Feltételes lokalitás. A szaturációalapú modellellenőrzők implementálásában szerzett tapasztalatunk alapján szükség lenne egy, a szaturációval jól működő, általános állapotátmeneti reláció reprezentációra, hogy az algoritmikus és a reprezentációval kapcsolatos problémákat szétválaszthassuk. Ennek érdekében bevezettük az absztrakt állapotátmenet diagramokat (Abstract Next-State Descriptor, ANSD), amik általánosítják a *döntési diagram-szerű reprezentációkat* a közös tulajdonságaik kiemelésével, amivel egy absztrakt interfészt kapunk [c4]. Ugyan az 1. és 2. tézisben tárgyalt megközelítések egy része felfogható ennek az interfésznek a specializációjaként, az EVIDD-k és Büchi automata átmeneti relációk kezelésének szaturációba építésének más motivációja is volt.

Ennek a tézisnek a magját egy megfigyelés adja, ami a *feltételes lokalitás* fogalmának bevezetéséhez vezetett. A rekurzív szaturáció algoritmus kihasználja, hogy egy absztrakt részmodell tranzíciói függetlenek a részmodellben nem szereplő állapotváltozóktól, tehát csak a részmodellen is kiértékelhetők és tetszőlegesen sokszor végrehajthatók. Ez azért fontos, mert a részmodell állapotterének kiszámításánál a benne lévő tranzíciókat kimerítően tüzelni kell, hogy minden elérhető állapotot megkapjunk. De vajon más tranzíciókat is lehetne kimerítően tüzelni?

[1] http://petridotnet.inf.mit.bme.hu/en/



(a) Ahogy a szaturáció látja.



(b) Ahogy a GSA látja.

7. ábra. A tiltóéles példa (1b. ábra) üzenethez tartozó részmodellje ahogy az eredeti szaturáció, illetve ahogy a feltételes lokalitással általánosított szaturáció (GSA) látja. Vegyük észre, hogy a GSA képes megállapítani a t_3 tranzíció engedélyezettségét, de az nem fogja megváltoztatni a termelő állapotát.

A válasz igen: például azokat a tranzíciókat, amik csak olvassák a részmodellben nem szereplő változókat, elméletben akárhányszor tüzelhetnénk, mert nem változtatják meg az elhagyott változó értékét – *feltéve, hogy engedélyezettek*. Általánosságban is elmondható, hogy ha egy tranzíció nem változtatja meg az elhagyott változók értékét, akkor akárhányszr tüzelhető a részmodellben, még akkor is, ha az olvasott érték befolyásolja a tranzíció viselkedését.

A feltételes lokalitás éppen ezt a tulajdonságot formalizálja: egy tranzíció feltételesen lokális egy részleges állapotvektor mellett (de nem az összesben), ha bármelyik, ezt kiegészítő teljes állapotból tüzelve nem változtatja meg a részleges állapotvektorban is szereplő változók értékét [c3]. Ezt az elképzelést illusztrála a 7. ábra: az absztrakt részmodellekben most már minden változó megjelenik, de csak azok a tranzíciók, amik legfeljebb olvassák a felsőbb változókat (amiket eddig elhagytunk).

Az általánosított szaturáció algoritmus. A feltételes lokalitásra építve javasoltam a szaturáció egy általánosított változatát, amely dinamikusan számítja a fent leírt részmodelleket, automatikusan újraparticionálva az állapotátmeneti relációt annak érdekében, hogy mindent a lehető legalacsonyabb szinthez tartozó részmodellben dolgozhasson fel [c3]. Ez a megközelítés erősíti a „szaturáció effektust”, mivel a munka nagyobb részét végezzük a kisebb részmodelleken. Még ennél is hasznosabb azonban, hogy minél több állapotátmenetet tudunk végrehajtani az alacsonyabban lévő részmodellekben, annál valószínűbb, hogy az ott kapott részeredmény (döntési diagram csomópont) végleges lesz, ami egy gyorsabb és memóriatakarékosabb algoritmushoz vezet.

Az általánosított szaturáció algoritmus (generalized saturation algorithm, GSA) absztrakt állapotátmenet diagramokat használ az állapotátmeneti reláció leírására. Ennek automatikus particionálásával az 1. és 2. tézisben bevezetett reprezentációk (az EVIDD és a Büchi automata állapotátmeneti relációját megadó döntési diagram), valamint a korábban megjelent változatokban használt kényszerek (mint a vezérelt szaturáció [ZC09]) immár akár dinamikusan, akár statikusan összevonhatók a modell állapotátmeneti relációjával, és nem kell megváltoztatni a szaturáció algoritmust a speciális kezelésük és a lokalitás megtartása érdekében. Ráadásul a 2. tézisben bemutatott SCC kereső algoritmus is hatékonyabb lehet a tranzíciók alacsonyabb szinten történő

kiértékelésétől, mivel ez felgyorsíthatja a számításokat és korábbi (alacsonyabb szinten történt) SCC találatokhoz is vezethet.

Eredmények. Összehasonlítottam a GSA-t az eredeti szaturáció algoritmussal az MCC Petri-háló modelljein, és azt találtam, hogy gyakorlatilag nincs többletköltsége a javasolt továbbfejlesztéseknek sem abban az esetben, amikor a modell tartalmaz feltételesen lokális tranzíciókat, sem akkor amikor nem, miközben előbbi esetben gyakran nagyságrendekkel gyorsabb és kevesebb memóriát fogyaszt. A kísérletek alapján a GSA az eredeti szaturáció tisztán jótékony hatású ötleteit fejleszti tovább mindenféle járulékos veszteség vagy költség nélkül, miközben általánosítja is egy sor olyan probléma megoldását, ami az eredeti algoritmusnak nehézséget okozott. Ugyan ez az eredmény lefed néhányat a másik két tézis eredményei közül, ez a legfrissebb és legfontosabb kontribúcióm a területen, ami nem alakult volna ki a korábbi eredmények közös gondolatainak azonosítása és általánosítása nélkül.

Thesis 3 Kidolgoztam a szaturáció algoritmus feltételes lokalitáson alapuló továbbfejlesztését, valamint megmutattam, hogy a vezérelt szaturáció algoritmus, az 1. tézisben bemutatott kiterjesztés prioritásos Petri-hálókra, illetve a 2. tézisben bemutatott kiterjesztés a szorzat állapottér közvetlen számítására mind ennek az általánosított szaturáció algoritmusnak a példányai.

- 3.1 Definitáltam a feltételes lokalitás és a feltételesen szaturált csomópont fogalmát, ezzel relaxálva az eredeti algoritmusban használt lokalitásfogalmat.
- 3.2 A feltételes lokalitásra építve kidolgoztam az általánosított szaturáció algoritmust (generalized saturation algorithm, GSA), ami erősíti a "'szaturáció effektust" és ezáltal továbbfejleszti az eredeti algoritmust. Formálisan igazoltam az új algoritmus helyességét.
- 3.3 Megmutattam, hogy a GSA általánost egy sor, a vezérelt szaturáción alapuló szaturáció variánst. Megadtam az eredeti vezérelt szaturáció és az általam javasolt kapcsolódó kiterjesztések leírását a GSA kontextusában is.
- 3.4 Kiértékeltem a GSA hatékonyságát a Model Checking Contest (MCC) Petri-háló modelljein, ami megmutatta, hogy akár nagyságrendekkel is jobb teljesítményt tud nyújtani az eredeti szaturáció algoritmusnál, miközben gyakorlatilag semmilen esetben nem ront az eredeti algoritmus teljesítményén.

Ezen eredmények jelentőségét az adja, hogy a szatruáció algoritmus már eddig is az egyik legsikeresebb szimbolikus modellellenőrző algoritmus volt, amit még tovább sikerült fejleszteni a feltételes lokalitás bevezetésével. Az egyik legnagyobb probléma a modellellenőrzésben az állapottér robbanás, amit jobb skálázódással és absztrakciókkal tudunk kezelni – ebben az eredményben mindkét megközelítés kombinálódik. Ezen felül a bemutatott eredmény általánosít egy algoritmuscsaládot, elősegítve az algoritmusok helyességét, kompatibilitását és megértését, valamint a szaturációalapú modellellenőrző eszközök karbantartható implementációit, mindezt azáltal, hogy egy közös keretrendszert és általános helyességbizonyításokat ad ezeknek az algoritmusoknak. Végül, az állapotátmeneti relációk fölé az ANSD által bevezetett absztrakciós réteg és az állapotátmeneti relációk automatikus particionálása lehetővé teszi általánosabb modellezési formalizmusok közvetlen modellellenőrzését is (pl. hierarchikus állapotgépek), ami egyúttal elősegíti a modellellenőrző eszközök és a magas szintű modellező eszközök, mint a Gamma Statechart Composition Framework (amit Graics Bence korábbi hallgatómmal együtt fejlesztünk) integrálását is [c12; c13; c14].

Ebben a kutatásban részt vett Majzik István mint Ph.D. témavezetőm. A 3. tézis eredményeit a 6. fejezet mutatja be. A kapcsolódó publikációk a következők: [c3], [c4].

3. Az új eredmények alkalmazása

3.1. Általánosított Sztochasztikus Petri-Hálók sztochasztikus analízise

A prioritásos Petri-hálókra kiterjesztett szaturáció felhasználásra került GPSN-ek stochasztikus analízisére egy olyan eszközben, amelyben az állapottér felderítést a szaturáció végzi el [c4], az állapottér-dekompozíció döntési diagramokon alapul [c5], és a numerikus analízist egy konfigurálható megoldó keretrendszer végzi el [c7]. A sztochasztikus analízis a PetriDotNet keretrendszerben[2] [j1; c6] is implementálásra került, amelyet felhasználtak autóipari beágyazott rendszerek megbízhatósági analízisére.

3.2. LTL modellellenőrzés

A szaturációalapú inkrementális LTL modellellenőrzés szintén implementálásra került az online szabadon hozzáférhető PetriDotNet keretrendszerben. Ezzel kiegészülve a PetriDotNet egy széleskörű analízis algoritmushalmazzal rendelkezik, mindezt egy grafikus felületbe építve, lehetővé téve a felhasználóknak Petri-hálók és sztochasztikus Petri-hálók modellezését, szimulációját és analízisét invariáns, CTL vagy LTL formulákkal, illetve extrafunkcionális metrikákkal kifejezett tulajdonságok alapján. Amint az bemutatásra került [j1]-ben, az eszközt több független projektben is felhasználták. Az ebben a tézisben bemutatott LTL modellellenőrző algoritmus volt az első algoritmus, amely sikeresen igazolta LTL tulajdonságok teljesülését az ún. PRISE modellben. A PRISE modell egy biztonsági eljárást ír le a magyarországi Paksi atomerőműben, amelynek célja az elsődleges hűtőkör másodlagos hűtőkörbe történő szivárgásának detektálása [Ném+09].

3.3. Az általánosított szaturáció algoritmus

Habár az általánosított szaturáció algoritmus viszonylag újnak tekinthető, dolgozunk rajta, hogy kiaknázzuk a tulajdonságait a kutatócsoportunkban fejlesztett Theta Keretrendszerben (Theta Configurable Abstraction Refinement-based Verification Framework) [Tót+17]. A keretrendszer absztrakcióalapú szoftver-modellellenőrzési lehetőségeket kínál ellenpélda-alapú absztrakció finomítással (counterexample-guided abstraction refinement, CEGAR), illetve több bemeneti nyelvet támogat, hogy elősegítse az integrációt más modellezési eszközökkel. A szaturáció és a CEGAR kombinációja egy ígéretes kutatási iránynak tűnik. Ezzel párhuzamosan dolgozunk a keretrendszer Gamma Keretrendszerhez (Gamma Statechart Composition Framework) történő integrálásán [c12; c13; c14], ahol a komponens-alapú rendszerek verifikációját a szaturáció algoritmus nagyban tudja segíteni. A Gamma keretrendszer felhasználásra került ipari projektekben beágyazott rendszerek modellezésére és analízisére, valamint kód- és tesztgenerálásra.

3.4. A modellellenőrzés használati esetei

Kutatásom egy része, amely nem szigorúan a disszertáció témájához kapcsolódik, a modellellenőrzés különböző felhasználási lehetőségeit vizsgálta, ami a munkámat a gyakorlati alkalmazhatóság felé terelte. A CECRIS projektben[3] kifejlesztettem egy modellellenőrzés-alapú metodológiát szoftverek automatikus hibamod- és hatás analízisére (Failure Mode and Effect Analysis, FMEA) [j10; c16]. A megközelítés nemdeterminisztikusan aktiválódó hibák injektálásán alapszik, melynek lehetséges hatásait több célból modellellenőrzés segítségével ellenőrizzük. FMEA esetén alapvetően az a kérdés, hogy az aktivált hiba elvezet-e rendszerszintű problémáig, amit külön specifikálunk. A megközelítés arra is alkalmas, hogy hibadetektorokat és hibakezelő módszereket értékeljünk ki vele a hibamentes, valamint a hibainjektált modell és a vizsgált módszer

[2] http://petridotnet.inf.mit.bme.hu/en/
[3] FP7–Marie Curie (IAPP) number 324334

együttesének konformancia-alapú összehasonlításával. A projekt eredményeit, beleértve az én munkámat is, egy könyvben publikálták [b17].

Bajczi Levente hallgatómmal kooperációban feltártunk és megvizsgáltunk egy új problématerületet is, ami párhuzamos programok többmagos rendszereken történő futásával kapcsolatos [c11]. Ezekben az esetekben a memóriavezérlőnek kritikus szerepe van a teljes rendszer helyességében. A beágyazott rendszerekben széleskörűen használt alkalmazásspecifikus processzorok korában a hibás memóriavezérlők egyre kevésbé ritkák, részben azért, mert nagyon bonyolultak, és azért is, mert a hardver verifikációs technikák egyre több olyen hibát képesek kimutatni akár a processzor megjelenése után is. Szerencsére ezek az egy célra tervezett chipek gyakran egyetlen programot futtatnak egész élettartamuk alatt, a hibák aktiválódása pedig gyakran speciális helyzetekhez köthető. Ha a programunk soha nem idéz elő ilyen helyzetet, akkor a hiba sosem aktiválódik és a probléma szunnyadó állapotban maradhat. Munkánkban megmutattuk, hogy napjaink modellellenőrző algoritmusai nincsenek felkészítve ehhez hasonló problémák megoldására, és a probléma formális definíciójával új kutatási területet és kihívásokat nyitottunk a modellellenőrzés területén.

4. Publikációs lista

Publikációk száma:	17
Angol nyelvű lektorált folyóiratcikkek száma:	4
WoS vagy Scopus által indexelt folyóiratcikkek száma:	4
Angol nyelvű publikációk a szerző legalább 50%-os hozzájárulásával:	7
Lektorált publikációk száma:	17
Független hivatkozások száma:	18

4.1. Tézisekhez kapcsolt publikációk

	Folyóiratcikkek	Nemzetközi konferenciacikkek
1. tézis	[j1]*	[c4]*, [c5], [c6]*, [c7]
2. tézis	[j1]*, [j2]	[c6]*, [c8]
3. tézis	—	[c3] [c4]*

* A megjelölt publikációk több tézishez is kapcsolódnak.
A cikkek besorolása a doktori iskola publikációs pontozásához használt besorolást követi.

Folyóiratcikkek

[j1] András Vörös, Dániel Darvas, Ákos Hajdu, Attila Klenik, Kristóf Marussy, Vince Molnár, Tamás Bartha, and István Majzik. Industrial applications of the PetriDotNet modelling and analysis tool. *Science of Compututer Programming* 157, 2018, pp. 17–40. DOI: 10.1016/j.scico.2017.09.003. URL: https://doi.org/10.1016/j.scico.2017.09.003.
▷ *A PetriDotNet keretrendszer implementációja a szerzők közös munkája, melyben Szilvási Bertalan eredeti munkájára alapoztunk. A szaturációalapú LTL modellellenőrző algoritmus a saját kontribúcióm, melyet Vörös András és Bartha Tamás témavezetése alatt készítettem.*

[j2] Vince Molnár, András Vörös, Dániel Darvas, Tamás Bartha, and István Majzik. Component-wise incremental LTL model checking. *Formal Aspects of Computing* 28(3), 2016, pp. 345–379. DOI: 10.1007/s00165-015-0347-x. URL: https://doi.org/10.1007/s00165-015-0347-x.
▷ *Saját kontribúcióm, közös cikk a B.Sc., M.Sc. és Ph.D. témavezetőimmel.*

Nemzetközi konferencia- és workshop cikkek

[c3] Vince Molnár and István Majzik. Saturation enhanced with conditional locality: Application to Petri nets. In: Susanna Donatelli and Stefan Haar (eds.), *Application and Theory of Petri Nets and Concurrency - 40th International Conference, PETRI NETS 2019, Aachen, Germany, June 23-28, 2019, Proceedings*, Lecture Notes in Computer Science, vol. 11522, pp. 342–361. Springer, 2019. DOI: 10.1007/978-3-030-21571-2_19. URL: https://doi.org/10.1007/978-3-030-21571-2%5C_19.
▷ *Saját kontribúcióm, közös cikk a Ph.D. témavezetőmmel.*

[c4] Kristóf Marussy, Vince Molnár, András Vörös, and István Majzik. Getting the priorities right: Saturation for prioritised Petri nets. In: Wil M. P. van der Aalst and Eike Best (eds.), *Application and Theory of Petri Nets and Concurrency - 38th International Conference, PETRI NETS 2017, Zaragoza, Spain, June 25-30, 2017, Proceedings*, Lecture Notes in Computer Science, vol. 10258, pp. 223–242. Springer, 2017. DOI: 10.1007/978-3-319-57861-3_14. URL: https://doi.org/10.1007/978-3-319-57861-3%5C_14.
▷ *Saját kontribúcióm, közös cikk az M.Sc. és Ph.D. témavezetőimmel és a diákommal, Marussy Kristóffal.*

[c5] Kristóf Marussy, Attila Klenik, Vince Molnár, András Vörös, István Majzik, and Miklós Telek. Efficient decomposition algorithm for stationary analysis of complex stochastic Petri net models. In: Fabrice Kordon and Daniel Moldt (eds.), *Application and Theory of Petri Nets and Concurrency - 37th International Conference, PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings*, Lecture Notes in Computer Science, vol. 9698, pp. 281–300. Springer, 2016. DOI: 10.1007/978-3-319-39086-4_17. URL: https://doi.org/10.1007/978-3-319-39086-4%5C_17.
▷ *A sztochasztikus analízis algoritmus Marussy Kristóf kontribúciója. A makró-állapot dekompozíciós algoritmus saját kontribúcióm.*

[c6] András Vörös, Dániel Darvas, Vince Molnár, Attila Klenik, Ákos Hajdu, Attila Jámbor, Tamás Bartha, and István Majzik. PetriDotNet 1.5: Extensible Petri net editor and analyser for education and research. In: Fabrice Kordon and Daniel Moldt (eds.), *Application and Theory of Petri Nets and Concurrency - 37th International Conference, PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings*, Lecture Notes in Computer Science, vol. 9698, pp. 123–132. Springer, 2016. DOI: 10.1007/978-3-319-39086-4_9. URL: https://doi.org/10.1007/978-3-319-39086-4%5C_9.
▷ *A PetriDotNet keretrendszer implementációja a szerzők közös munkája, melyben Szilvási Bertalan eredeti munkájára alapoztunk. A szaturációalapú LTL modellellenőrző algoritmus a saját kontribúcióm, melyet Vörös András és Bartha Tamás témavezetése alatt készítettem.*

[c7] Kristóf Marussy, Attila Klenik, Vince Molnár, András Vörös, Miklós Telek, and István Majzik. Configurable numerical analysis for stochastic systems. In: *2016 International Workshop on Symbolic and Numerical Methods for Reachability Analysis (SNR)*, pp. 1–10. Apr. 2016. DOI: 10.1109/SNR.2016.7479383.
▷ *Marussy Kristóf és Klenik Attila hallgatóim kontribúciója, melyet Vörös András, Telek Miklós és Majzik István mellett én is konzultáltam.*

[c8] Vince Molnár, Dániel Darvas, András Vörös, and Tamás Bartha. Saturation-based incremental LTL model checking with inductive proofs. In: Christel Baier and Cesare Tinelli (eds.), *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, Lecture Notes in Computer Science, vol. 9035, pp. 643–657. Springer, 2015. DOI: 10.1007/978-3-662-46681-0_58. URL: https://doi.org/10.1007/978-3-662-46681-0%5C_58.

▷ *Saját kontribúcióm, közös cikk a B.Sc., M.Sc. és Ph.D. témavezetőimmel.*

Helyi rendezvények

[l9] Vince Molnár and András Vörös. Synchronous product automaton generation for controller optimization. In: *ASCONIKK 2014: Extended Abstracts. I. Information Technologies for Logistic Systems*, pp. 22–29. Veszprém, Hungary: University of Pannonia, Dec. 2014.
▷ *A tabló-automataalapú szorzat számító algoritmus Vörös András kontribúciója, melyben a jelen munka alapjául is szolgáló közös ötletek is megjelennek.*

4.2. További publikációk (a tézisek használati eseteihez és alkalmazásaihoz kapcsolódva)

Folyóiratcikkek

[j10] Vince Molnár and István Majzik. Model checking-based software-FMEA: Assessment of fault tolerance and error detection mechanisms. *Periodica Polytechnica Electrical Engineering and Computer Science* 61(2), 2017, pp. 132–150. DOI: https://doi.org/10.3311/PPee.9755. URL: https://pp.bme.hu/eecs/article/view/9755.

Nemzetközi konferencia- és workshop cikkek

[c11] Levente Bajczi, András Vörös, and Vince Molnár. Will my program break on this faulty processor? - Formal analysis of hardware fault activations in concurrent embedded software. *Transactions on Embedded Computing Systems* 18(5s), 2019, pp. 1–21. DOI: https://doi.org/10.1145/3358238.

[c12] Vince Molnár, Bence Graics, András Vörös, István Majzik, and Dániel Varró. The Gamma statechart composition framework: Design, verification and code generation for component-based reactive systems. In: Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (eds.), *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pp. 113–116. ACM, 2018. DOI: 10.1145/3183440.3183489. URL: https://doi.org/10.1145/3183440.3183489.

[c13] Bence Graics and Vince Molnár. Mix-and-match composition in the Gamma framework. In: Béla Pataki (ed.), *Proceedings of the 25th PhD Mini-Symposium*, pp. 24–27. Budapest, Hungary, 2018.

[c14] Bence Graics and Vince Molnár. Formal compositional semantics for Yakindu statecharts. In: Béla Pataki (ed.), *Proceedings of the 24th PhD Mini-Symposium*, pp. 22–25. Budapest, Hungary, 2017.

[c15] Vince Molnár and István Majzik. Constraint programming with multi-valued decision diagrams: A saturation approach. In: Béla Pataki (ed.), *Proceedings of the 24th PhD Mini-Symposium*, pp. 54–57. Budapest, Hungary, 2017.

[c16] Vince Molnár and István Majzik. Evaluation of fault tolerance mechanisms with model checking. In: Béla Pataki (ed.), *Proceedings of the 23rd PhD Mini-Symposium*, pp. 30–33. Budapest, Hungary, 2016.

Könyvfejezetek

[b17] Valentina Bonfiglio, Francesco Brancati, Francesco Rossi, Andrea Bondavalli, Leonardo Montecchi, András Pataricza, Imre Kocsis, and Vince Molnár. Composable framework support for software-FMEA through model execution. In: Bondavalli Andrea and Brancati

Francesco (eds.), *Certifications of Critical Systems – The CECRIS Experience*, pp. 183–200. Delft, Netherlands: River Publishers, 2017.

4.3. További munkák

[a18] Vince Molnár. Advanced Saturation-based Model Checking. Master's Thesis. Budapest University of Technology and Economics, 2014. URL: https://diplomaterv.vik.bme.hu/en/Theses/Szaturacio-alapu-modellellenorzes.

[a19] Vince Molnár. Szaturáció alapú modellellenőrzés lineáris idejű tulajdonságokhoz [in Hungarian; Saturation-based model checking for linear temporal properties]. Bachelor's Thesis. Budapest University of Technology and Economics, 2013. URL: http://petridotnet.inf.mit.bme.hu/publications/BSThesis2013_Molnar.pdf.

[a20] Vince Molnár and Dániel Segesdi. Múlt és jövő: Új algoritmusok lineáris temporális tulajdonságok szaturáció-alapú modellellenőrzésére [in Hungarian; Future and past: New algorithms for the saturation-based model checking of linear temporal properties]. Scientific Students' Association Report. 1st prize, national 2nd prize (OTDK 2015). 2013. URL: http://petridotnet.inf.mit.bme.hu/publications/TDK2013_MolnarSegesdi.pdf.
▷ *A Past-LTL-ből Büchi automatára fodító és automata egyszerűsítő algoritmusok Segesdi Dániel kontribúciói. A szaturációalapú automataelméleti modellellenőrző algoritmus a saját kontribúcióm.*

[a21] Vince Molnár. Szaturáció alapú modellellenőrzés lineáris idejű tulajdonságokhoz [in Hungarian; Saturation-based model checking for linear temporal properties]. Scientific Students' Association Report. 2nd prize. 2012. URL: http://petridotnet.inf.mit.bme.hu/publications/TDK2012_Molnar.pdf.

Hivatkozások

[Ama] José Nelson Amaral. About Computing Science Research Methodology. URL: https://webdocs.cs.ualberta.ca/~c603/readings/research-methods.pdf.

[Amp+19] Elvio Gilberto Amparore, Gianfranco Ciardo, Susanna Donatelli, and Andrew S. Miner. i_{Rank}: A variable order metric for DEDS subject to linear invariants. In: Tomás Vojnar and Lijun Zhang (eds.), *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II*, Lecture Notes in Computer Science, vol. 11428, pp. 285–302. Springer, 2019. DOI: 10.1007/978-3-030-17465-1_16.

[BH14] Dines Bjørner and Klaus Havelund. 40 years of formal methods - some obstacles and some possibilities? In: Cliff B. Jones, Pekka Pihlajasaari, and Jun Sun (eds.), *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, Lecture Notes in Computer Science, vol. 8442, pp. 42–61. Springer, 2014. DOI: 10.1007/978-3-319-06410-9_4.

[Bry86] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* C-35(8), 1986, pp. 677–691. DOI: 10.1109/TC.1986.1676819.

[Büc62] J. Richard Büchi. On a decision method in restricted second order arithmetic. In: Ernest Nagel, Patrick Suppes, and Alfred Tarski (eds.), *Proc. of the 1960 International Congress on Logic, Methodology and Philosophy of Science*, pp. 1–11. Stanford Univ. Press, 1962.

[Bur+92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation* 98(2), 1992, pp. 142–170. DOI: 10.1016/0890-5401(92)90017-A. (Visited on 09/08/2014).

[Cav+14] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Alberto Griggio, Marco Roveri, and Stefano Tonetta. *The nuXmv Symbolic Model Checker*. Tech. rep. Fondazione Bruno Kessler, 2014.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[Chi+93] Giovanni Chiola, Marsan Marco Ajmone, Gianfranco Balbo, and Gianni Conte. Generalized stochastic Petri nets: A definition at the net level and its implications. *IEEE Trans. Software Eng.* 19(2), 1993, pp. 89–107.

[Cim+02] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: an opensource tool for symbolic model checking. In: Ed Brinksma and Kim G. Larsen (eds.), *Computer Aided Verification*, Lecture Notes in Computer Science, vol. 2404, pp. 359–364. Springer, 2002. URL: http://dx.doi.org/10.1007/3-540-45657-0_29.

[CLS01] Gianfranco Ciardo, Gerald Lüttgen, and Radu Siminiceanu. Saturation: an efficient iteration strategy for symbolic state-space generation. In: *Proc. of the 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 328–342. 2001.

[CMS06] Gianfranco Ciardo, Robert Marmorstein, and Radu Siminiceanu. The saturation algorithm for symbolic state-space exploration. *International Journal on Software Tools for Technology Transfer* 8(1), 2006, pp. 4–25. DOI: 10.1007/s10009-005-0188-7. (Visited on 08/24/2014).

[Dur+11] Alexandre Duret-Lutz, Kaïs Klai, Denis Poitrenaud, and Yann Thierry-Mieg. Self-loop aggregation product – A new hybrid approach to on-the-fly LTL model checking. In: Tevfik Bultan and Pao-Ann Hsiung (eds.), *Automated Technology for Verification and Analysis*, Lecture Notes in Computer Science, vol. 6996, pp. 336–350. Springer, 2011. DOI: 10.1007/978-3-642-24372-1_24.

[EC80] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In: J. W. de Bakker and Jan van Leeuwen (eds.), *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, Lecture Notes in Computer Science, vol. 85, pp. 169–181. Springer, 1980. DOI: 10.1007/3-540-10003-2_69.

[Kor+18] F. Kordon et al. Complete Results for the 2018 Edition of the Model Checking Contest. http://mcc.lip6.fr/2018/results.php. 2018. (Visited on 2018).

[LT93] N. G. Leveson and C. S. Turner. An investigation of the Therac-25 accidents. *Computer* 26(7), 1993, pp. 18–41.

[Mar+16b] Kristóf Marussy, Attila Klenik, Vince Molnár, András Vörös, Miklós Telek, and István Majzik. Configurable numerical analysis for stochastic systems. In: *2016 International Workshop on Symbolic and Numerical Methods for Reachability Analysis (SNR)*, pp. 1–10. 2016. DOI: 10.1109/SNR.2016.7479383.

[MD98] D. Michael Miller and Rolf Drechsler. Implementing a multiple-valued decision diagram package. In: *Proc. of the 28th IEEE International Symposium on Multiple-Valued Logic*, pp. 52–57. 1998. DOI: 10.1109/ISMVL.1998.679287.

[Min01] Andrew S Miner. Efficient solution of GSPNs using canonical matrix diagrams. In: *Petri Nets and Performance Models, 2001. Proceedings. 9th International Workshop on*, pp. 101–110. 2001.

[Mur89] Tadao Murata. Petri nets: properties, analysis and applications. *Proceedings of the IEEE* 77(4), 1989, pp. 541–580. DOI: 10.1109/5.24143.

[Ném+09] Erzsébet Németh, Tamás Bartha, Csaba Fazekas, and Katalin M. Hangos. Verification of a primary-to-secondary leaking safety procedure in a nuclear power plant using coloured Petri nets. *Reliability Engineering & System Safety* 94(5), 2009, pp. 942–953. DOI: 10.1016/j.ress.2008.10.012.

[Pnu77] Amir Pnueli. The temporal logic of programs. In: *Proc. of the 18th Annual Symposium on Foundations of Computer Science*, pp. 46–57. IEEE Computer Society, 1977. DOI: 10.1109/SFCS.1977.32.

[QS82] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In: Mariangiola Dezani-Ciancaglini and Ugo Montanari (eds.), *International Symposium on Programming*, pp. 337–351. Springer Berlin Heidelberg, 1982.

[RS10] Pierre Roux and Radu Siminiceanu. Model checking with edge-valued decision diagrams. In: *Proc. of the 2nd NASA Formal Methods Symposium*, pp. 222–226. 2010.

[Tót+17] Tamás Tóth, Ákos Hajdu, András Vörös, Zoltán Micskei, and István Majzik. Theta: A framework for abstraction refinement-based model checking. In: Daryl Stewart and Georg Weissenbacher (eds.), *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017*, pp. 176–179. IEEE, 2017. DOI: 10.23919/FMCAD.2017.8102257.

[Tov08] Alexey A. Tovchigrechko. Efficient symbolic analysis of bounded Petri nets using interval decision diagrams. PhD thesis. Brandenburg University of Technology, Cottbus-Senftenberg, Germany, 2008.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In: *Proc. of the Symposium on Logic in Computer Science*, pp. 332–344. IEEE Computer Society, 1986.

[ZC09] Yang Zhao and Gianfranco Ciardo. Symbolic CTL model checking of asynchronous systems using constrained saturation. In: *Proc. of the 7th Int. Conf. Automated Technology for Verification and Analysis*, pp. 368–381. 2009.

[ZC11] Yang Zhao and Gianfranco Ciardo. Symbolic computation of strongly connected components and fair cycles using saturation. en. *Innovations in Systems and Software Engineering* 7(2), 2011, pp. 141–150. DOI: 10.1007/s11334-011-0146-3. (Visited on 08/24/2014).