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

Hatékony szakterület-specifikus formális ellenőrzési technikák

Tézisfüzet

Hajdu Ákos

Témavezető:
Micskei Zoltán, PhD (BME)

Budapest, 2020.

1. Előzmények és célok

Életünk számos területén támaszkodunk egyre nagyobb mértékben a számítógépes rendszerek és programok megbízható és helyes működésére. Ilyen jellegű rendszerek gyakran fordulnak elő *kritikus* környezetekben is, ahol egy hiba súlyos károkat (pl. ipari vezérlők) vagy anyagi következményeket (pl. pénzügyi tranzakciók) okozhat. A hibák kiküszöbölésére verifikációs módszerek egy széles skálája áll rendelkezésre, az egyszerű fordítóprogram ellenőrzésektől a tesztelésen át a futásidejű monitorozásig. Ezen technikák között egyre nagyobb szerepet kap a *formális verifikáció* is, különösen a kritikus alkalmazási területeken. A formális verifikáción alapuló technikáknak precíz matematikai alapja van, és képesek bizonyos típusú hibák megléte mellett a hiányukat is bizonyítani. A számítógépes rendszerek és programok precíz vizsgálata McCarty [McC62], Floyd [Flo67], Hoare [Hoa69] és Dijkstra [Dij76] korszakalkotó munkáin keresztül több évtizedre nyúlik vissza. Bár a korai eredmények hamar megmutatták, hogy a legtöbb érdekes kérdés (pl. terminálódás) eldönthetetlen probléma [Tur36; Chu36; Ric53], mégis nagy érdeklődés támadt azon módszerek iránt, amelyek a gyakorlati esetekre jól működnek.

Az automatikus formális verifikáció akkor indult rohamos fejlődésnek, amikor a *modellellenőrzést* (model checking) [Cla+18] kidolgozták. Ez egy olyan technika, amely egy formálisan specifikált *tulajdonságot* (property) ellenőriz a rendszer egy formális *modelljén* (reprezentációján) úgy, hogy a rendszer összes lehetséges *állapotát* (state) és *átmenetét* (transition), azaz a teljes *állapotterét* (state space) szisztematikusan megvizsgálja. Jelen disszertációban *diszkrét* rendszerekre koncentrálunk, ahol a rendszer viselkedése diszkrét állapotokkal és átmenetekkel jellemezhető. A korai módszerek az állapotteret kimerítő módon járták be [CE82; QS82], amely ritkán skálázódott valós méretű rendszerekre és programokra. Ennek ellenére a formális helyességbizonyítás ígérete nagy érdeklődést keltett, és az ellenőrzési módszerek széles skálája lett kidolgozva, többek között a szimbolikus módszerek (symbolic methods) [Bur+90], a részleges rendezés-alapú redukciók (partial order reduction) [Val91; God91; Pel93], a korlátos modellellenőrzés (bounded model checking) [Bie+99], az absztrakció (abstraction) [CGL94; Cla+03] és a moduláris verifikáció (modular verification) [Mül02]. A folyamatos fejlődés ellenére azonban továbbra is vannak nyitott kérdések, amelyek nem lettek teljesen megoldva, emellett pedig minden új alkalmazási terület új kihívásokat tartogat. A jelen disszertáció ezeket a kihívásokat célozza meg, hogy a formális verifikációt *hatásosabbá* (more effective) tegye a gyakorlatban.

1.1. Tulajdonságok és kihívások

A formális verifikáció elméleti szempontból gyakran tanulmányozott tulajdonságai a *helyesség* és *teljesség* [JM09; Bey12; Mey19], amelyeket az 1. ábra illusztrál. Egy rendszer vagy program viselkedhet megfelelően (egy adott tulajdonság szempontjából) vagy lehetnek hibás viselkedései. A rendszeren alkalmazott formális verifikáció eredménye pedig lehet elfogadás (a tulajdonság teljesül) vagy elutasítás (a tulajdonság sérül).

Helyesség. Egy analízist *helyesnek* (sound) nevezünk, ha nem fogad el olyan rendszert vagy programot, amelyben hibás viselkedés van. Az elmulasztott hibák (hamis negatívok, false negative) kritikusak, hiszen hamis bizalmat adnak a rendszer helyes működését illetően.

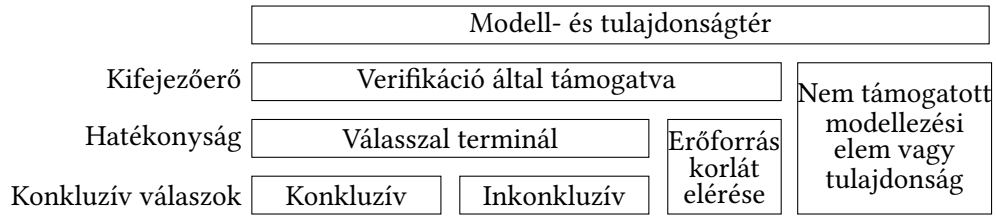
Teljesség. Egy analízist *teljesnek* (complete) nevezünk, ha csak valódi hibákat jelez, azaz csak valóban hibás viselkedéseket utasít el. A legtöbb esetben a hibajelzéshez egy példa lefutás (trace) is társul, amely megmutatja, hogy a hiba hogyan reprodukálható az eredeti rendszerben. A hamis riasztások (nem valódi hibák, false alarm) így egy szimuláció segítségével általában kiszűrhetőek, így kevésbé kritikusak mint az elmulasztott hibák. Ez a vizsgálat azonban extra munkát igényel, ezért túl sok hamis riasztás esetén is válhat egy módszer a gyakorlatban alkalmazhatatlanná.

		Formális verifikáció	
		Elfogadás	Elutasítás
Rendszer vagy program	Megfelelő viselkedés	Elfogadott helyes	Hamis riasztások (nem teljes)
	Hibás viselkedés	Elmulasztott hibák (nem helyes)	Elkapott hibák

1. ábra. A formális verifikáció lehetséges eredményei a valós viselkedéshez képest [Mey19]. Egy helyes analízis nem fogadhat el hibákat, míg egy teljes analízis nem adhat hamis riasztásokat.

A jelenlegi munkában a helyességet és teljességet nem vizsgáljuk közvetlenül. A bemutatásra kerülő algoritmusok és a hátterükben lévő logikák precíz matematikai alapokkal rendelkeznek és több kontextusban is használták már őket, egy részükhöz pedig formális helyességbizonyítás is tartozik. A helyesség és teljesség gyakran akkor sérül, amikor az eredeti, magasszintű modellt az algoritmusok által vizsgált alacsonyszintű matematikai formalizmusra képezzük le. A leképezések validálása (translation validation) viszont egy külön kutatási terület és túlmutat a jelenlegi munka keretein. Jelen disszertációban a módszerek helyességének és teljességének bizonyosságát úgy növeljük, hogy számos mesterséges és valós példán végzünk kiértékelést.

A helyesség és a teljesség fontos jellemzők, de csak akkor relevánsak, ha a verifikáció konkluzív eredménnyel terminál. Gyakorlati esetekben azonban általában a verifikáció jellemzőinek egy szélesebb skáláját kell vizsgálni (2. ábra), többek között a *kifejezőerőt*, a *hatékonyságot* és azon problémák körét, amelyre a módszer *konkluzív válasszal* terminál.



2. ábra. A verifikáció lehetséges kimenetei a gyakrolatban. Amikor a verifikáció nem tud konkluzív eredményt adni, terminálhat inkonkluzív módon, kifuthat az erőforrásokból, vagy nem támogatott tulajdonságba és formalizmusba ütközhet.

Kifejezőerő. A mérnökök és fejlesztők általában valamilyen magasszintű modellezési vagy programozási nyelven írják le a rendszereket és programokat. Ezen magasszintű modellek és tulajdonságok transzformációk sorozatán keresztül kerülnek leképezésre az algoritmusok által támogatott alacsonyszintű matematikai formalizmuskora (pl. automata) és tulajdonságokra (pl. temporális logika). Egy verifikációs módszer *kifejezőerejét* (expressive power) az általa támogatott modellezési formalizmusok és tulajdonságtípusok határozzák meg. A támogatott alacsonyszintű formalizmustól és tulajdonságtípusoktól függ általában az is, hogy milyen magasszintű modellezési elemeket és specifikációs megoldásokat lehet használni. Például korlátlan kapacitású kommunikációs csatornák ellenőrzéséhez az algoritmusnak potenciálisan végtelen állapotteret kell tudni kezelnie.

Hatékonyság. Gyakorlati alkalmazások esetén a formális verifikációt számos véges erőforrás korlátozza, például futásidő vagy memóriahasználat. A jelen disszertáció kontextusában akkor tekintünk egy módszert *hatékonynak* (efficient) ha skálázható verifikációt biztosít gyakorlati méretű és komplexitású rendszerekre. A „skálázhatóságot” nehéz pontosan definiálni, mert az alkalmazási területtől is

függ. Egy fejlesztőkörnyezetbe épített interaktív verifikációs algoritmus nem futhat pár másodpercnél tovább, míg folytonos integrációs környezetben tipikusan néhány perc áll rendelkezésre [Cal+15; Cho+20]. A verifikációs versenyek [Bey17; Cab+16; Amp+19] általában nagyobb futásidőket is megengednek (15–60 perc), és bizonyos területeken az is elfogadható lehet ha az analízisek éjszaka futnak több órán keresztül.[1]

Konkluzív válaszok. Az algoritmusok belefuthatnak el nem dönthető esetekbe vagy nem támogatott modell és tulajdonság alosztályokba. Ezekben az esetekben megállnak és *inkonkluzív választ* (inconclusive answer) adnak. Ez többek között akkor fordulhat elő, ha egy módszer alul- vagy felülbecslést (under/overapproximation) alkalmaz és a tulajdonságot vagy csak bizonyítani, vagy csak cáfolni tudja, de nem mindkettőt. Egy tipikus példa a korlátos modellellenőrzés [Bie+99], ami inkonkluzív eredményt ad ha eléri a korlátot anélkül, hogy hibát találna. Az inkonkluzív válasszal történő terminálás jobb, mint az erőforrások kimerítése vagy a helytelen válasz adása, de ideális esetben ezen esetek száma is minimalizálandó.

Kompromisszumok. A fent említett tulajdonságokat nehéz (vagy bizonyos esetekben elméletileg is lehetetlen) egyszerre teljes mértékben teljesíteni [JM09]. Például egy algoritmus kifejezőerejének növelése elméletileg is eldönthetetlenné teheti a problémát, így az algoritmus nem adhat konkluzív választ minden esetben. Emellett a hatékony verifikáció gyakran absztrakciókat igényel, ami hamis hibajelzésekkel a teljesség sérülését okozhatja.

Kihívások. Ebben a disszertációban a következő három kihívásra koncentrálunk.

1. *Kifejezőerő*: Hogyan lehet magasszintű modellezési formalizmusok és funkcionális tulajdonságok kifejezését és ellenőrzését támogatni?
2. *Hatékonyság*: Hogyan lehet egy módszer hatékonyságát növelni annak érdekében, hogy gyakorlati méretű rendszerek és programok szélesebb skálájára terminálódjon?
3. *Konkluzív válaszok*: Hogyan lehet azon problémák halmazát kiterjeszteni, amelyre a verifikáció konkluzív eredménnyel terminál?

> **Célok.** A disszertáció célja, hogy a különböző problématerületeken a gyakorlatban *hatásos* kompromisszumot (effective trade-off) érjen el az egyes kihívások közötti egyensúly megtalálásával.

1.2. Áttekintés

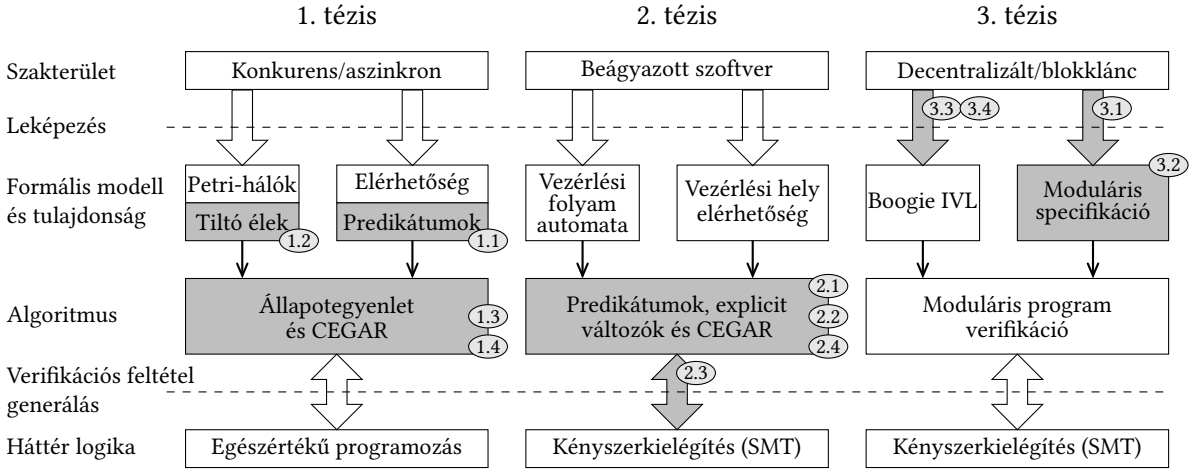
Jelen disszertációban három problématerület hatásos verifikációját célozzuk meg különböző modellezési formalizmusokkal és verifikációs módszerekkel:

1. konkurens és aszinkron rendszerek (1. tézis),
2. beágyazott szoftverkódok (2. tézis) és
3. blokklánc-alapú decentralizált rendszerek (3. tézis).

A kontribúciók áttekintése a 3. ábrán látható. A rendszerek és a programok minden területen (domain) általában valamilyen magasabb szintű, a mérnökök és fejlesztők számára kényelmes nyelven vannak megtervezve vagy leírva. Ezt a reprezentációt először egy formális modellre és tulajdonságra kell leképezni. A verifikációs algoritmus ezután a modell viselkedéseinek szisztematikus bejárásával ellenőrzi, hogy az teljesíti-e a tulajdonságot. Ennek során az algoritmus a tulajdonság teljesülését formulákká és egyenletekké, úgynevezett *verifikációs feltételekké* (verification condition) alakítja és valamilyen háttér logika segítségével oldja meg. A szürke háttér a saját kontribúciókat emeli ki, a megfelelő altéziseket

[1] Darvas Dániellel, egy CERN-ben használt PLC verifikációs eszköz [DFB15] fejlesztőjével történő személyes kommunikáció alapján.

számozott ellipszisekkel jelölve (és később a szövegben is hivatkozva). Ahogy korábban említésre került, minden terület különböző kihívásokra helyez nagyobb hangsúlyt. Ezen kihívások – a kifejezőerő, a hatékonyság, és a konkluzív válaszok – a 4. ábrán láthatóak.



3. ábra. A tézisekben bemutatott problématerületek és algoritmusok áttekintése. A saját kontribúciók szürke háttérrel vannak megjelölve.

	1. tézis	2. tézis	3. tézis
Kifejezőerő	1.1 1.2		3.1 3.2
Hatékonyság		2.1 2.2 2.3 2.4	3.3 3.4
Konkluzív válaszok	1.3 1.4		

4. ábra. A tézisekben megcélzott kihívások áttekintése.

1.3. Konkurens és aszinkron rendszerek

A konkurens rendszerek több komponensből állnak, amelyek egymással – gyakran aszinkron módon – együttműködve valamilyen közös célt oldanak meg. Ilyen rendszerek közé tartoznak a kölcsönös kizárási protokollok, az ütemezők vagy a gyártósorok. Ebben az esetben a verifikáció fő fókusza általában a komponensek közötti kommunikáció, az interakciók és a protokollok. Az egyes komponensek végrehajtásának nagyszámú lehetséges átlapolódása miatt azonban az ilyen rendszerek állapottere gyakran exponenciálisan (vagy nagyobb ütemben) növekszik a résztvevők számával. Általános esetben pedig például a korlátlan kapacitású kommunikációs csatornák akár végtelen állapotteret is eredményezhetnek.

A *Petri-háló* (Petri net) [Mur89] egy kompakt modellezési formalizmus, amely strukturális és dinamikus analíziseket is nyújt. A Petri-háló egy páros gráf *helyekkel* és *tranzíciókkal*. A helyek *tokenekkel* lehetnek *megjelölve*, amelyek a háló által modellezett rendszer aktuális állapotát reprezentálják. A tranzíciók a hozzájuk kapcsolódó helyek tokeneinek elvételével és létrehozásával a tokenek eloszlását (azaz a háló jelölését) változtatják. Számos érdekes tulajdonság visszavezethető az állapot *elérhetőség* (reachability) [Mur89] problémájára, amely azt vizsgálja hogy egy adott állapot (jelölés) elérhető-e a háló kiinduló állapotából. Az elérhetőségi probléma eldönthető [May81; Kos82], de legalább nemelemi komplexitású [Cze+19].

Az elérhetőségi probléma hatékony megoldását számos munka célozta már meg [Amp+19]. Egy ígéretes algoritmus [WW11] a Petri-hálók *állapotegyenletét* (state equation) használja fel az elérhetőség felülbecslésére. Az állapotegyenlet egy strukturális analízis technika, amely egészértékű lineáris

programozáson [Sch86] alapul. Az állapotegyenlet egy fontos tulajdonsága, hogy strukturális technika lévén független az állapottér méretétől. Ebből adódóan képes nagy, vagy akár végtelen állapotterű modelleket is hatékonyan kezelni. Az állapotegyenlet megoldhatósága azonban csak szükséges, de nem elégséges feltétele az elérhetőségnek. Amennyiben az állapotegyenlet nem oldható meg, a célállapot biztosan nem érhető el. Ellenkező esetben azonban a kapott megoldást le kell ellenőrizni (szimulálni) a Petri-hálón. Ha a megoldás nem végrehajtható, az állapotegyenletet újabb megkötésekkel kell kiegészíteni hogy egy precízebb felülbecslést alkosson és új megoldást adjon. Ez a folyamat addig ismétlődik, amíg az állapotegyenlet megoldhatatlanná válik, vagy egy végrehajtható megoldást ad. Ez tulajdonképpen egy, a Petri-hálókra történő speciális alkalmazása az úgynevezett *ellenpélda-alapú absztrakciófinomításnak* (counterexample-guided abstracton refinement, CEGAR) [Cla+03].

> **1. tézis céljai.** Bár az algoritmus bizonyította hatékonyságát a Modellellenőrzési Versenyen (Model Checking Contest) [Kor+12], a kifejezőereje egyszerű Petri-hálókra korlátozódott, és nem volt megvizsgálva azon problémák köre, amelyre konkluzív válasszal terminál. Ebben a tézisben a fő kutatási cél azon problémák körének vizsgálata, amelyre az algoritmus *konkluzív választ* ad, továbbá a módszer *kifejezőerejének* növelése kiterjesztett Petri-hálókra és újfajta tulajdonságokra.

1.4. Beágyazott szoftverek

A biztonságkritikus szoftverek gyakran beágyazott rendszerekben és vezérlőkben működnek. Az ilyen programok általában C, vagy más alacsonyszintű nyelven íródnak, egy korlátozott elemkészlet felhasználásával. Ilyen szoftverek közé tartoznak például az ipari vezérlők kódjai és az eseményvezérelt rendszerek. A *vezérlési folyam automata* (control-flow automaton, CFA) [BHT07] egy széleskörűen használt formális reprezentációja a programoknak. A CFA egy gráf-alapú formalizmus, ahol a csúcsok a program vezérlési *helyeinek* felelnek meg, míg az *élek* a vezérlés lehetséges lefutásait reprezentálják a programváltozók feletti *műveletekkel*. Számos érdekes tulajdonság vezethető vissza egy kitüntetett *hibahely* (error location) elérhetőségének vizsgálatára, például assertion hibák, túlindexelés, nullával osztás, stb. [Bey15].

A szoftver-modellellenőrzés egyik fő kihívását az adja, hogy az adatváltozók gazdag értékkészlettel rendelkeznek (pl. egész számok, tömbök). Ezt a problémát gyakran *absztrakció* segítségével enyhítik. Az *ellenpélda-alapú absztrakciófinomítás* (counterexample-guided abstraction refinement, CEGAR) [Cla+03] egy automatikus verifikációs módszer, amely iteratívan épít és finomít absztrakciókat a program modelljéhez. Az elmúlt évtizedekben a CEGAR számos variánsát fejlesztették ki, mivel tipikusan más és más stratégiák bizonyultak megfelelőnek különböző rendszerek esetén. Egy általános CEGAR-alapú megközelítés általában két fő lépésből áll [j3]. Elsőként az *absztrakciós* fázis egy *absztrakt elérhetőségi gráfot* (abstract reachability graph, ARG) épít a kezdeti – általában durva – precizitással. Az ARG az absztrakt állapotteret reprezentálja valamilyen absztrakt domén, pl. *explicit értékek* (explicit values) [BL13] vagy *predikátumok* (predicate abstraction) [GS97] felett. Explicit értékek esetén a rendszer változóinak csak egy részhalmazát követi az analízis, predikátumabsztrakció használatakor pedig a konkrét értékek helyett logikai formulákkal leírt tényeket és kapcsolatokat tart nyilván. Az ARG egy felülbecslése az eredeti állapottérnek, így ha a hibahely nem érhető el, akkor az eredeti program is biztosan helyes. Ellenkező esetben egy absztrakt ellenpélda (egy hibahelyhez vezető útvonal) létezik. A második, *finomítási* fázis először megvizsgálja az ellenpélda végrehajthatóságát az eredeti modellen. Egy végrehajtható ellenpélda esetén az eredeti program hibás, egyébként pedig az absztrakció precizitása új változók vagy predikátumok követésével kerül növelésre [j3], és az absztrakt állapottér a hamis ellenpélda kiküszöbölésével lesz visszavágva. A következő iterációban az absztrakció a már finomított precizitással tud folytatódni, és ezek a lépések ismétlődnek iterációban addig, amíg a hibahely bizonyítottan nem érhető el, vagy egy valódi ellenpélda nem fordul elő. A CEGAR algoritmus a háttérben az ARG építése és a precizitás finomítása során *kényszerkielégítési* (satisfiability modulo theories, SMT) [BT18; BHM09] problémákra támaszkodik.

2. tézis céljai. Az absztrakció és CEGAR alkalmazása ellenére a skálázhatóság továbbra is egy fő korlátozó tényező a szoftver-modellellenőrzésben. A sikeres verifikáció általában különböző módszerek kombinációját [BLW15; BDW15; JD16] vagy eszközök egy portfólióját [Tul+14; Dem+17; Dar+18; GD19; RW19] igényli. Ebben a tézisben a fő kutatási cél a CEGAR-alapú szoftver-modellellenőrzés *hatékonyságának* növelése különböző új stratégiák és újszerű kiegészítések kidolgozásával, mind az absztrakció, mind a finomítás szempontjából.

1.5. Blokklánc-alapú decentralizált rendszerek

A blokklánc-alapú elosztott főkönyv (ledger) technológiák célja, hogy leváltsák az olyan centralizált megoldásokat, amelyeknél egy központi szereplőben (pl. bank) kell megbízni. A korai alkalmazások (mint például a Bitcoin [Nak08]) elsősorban digitális pénz, azaz kriptovaluták (cryptocurrencies) megvalósítására fókuszáltak. A sikerük hatalmas figyelmet generált és később általánosabb megoldások is megjelentek, mint például az Ethereum [Woo17]. Általános esetben a főkönyvre programok, úgynevezett *okosszerződések* (smart contracts [Sza94]) telepíthetőek, amelyek tetszőleges állapotot (adatot) tárolhatnak a blokkláncon és tranzakciók segítségével biztosíthatják az adatok manipulációját [AW18]. A népszerűség ellenére azonban számos hiba és sebezhetőség hívta fel a figyelmet arra, hogy az okosszerződések gyakran kritikus problémákat tartalmazhatnak [ABC17; DMH17; Dat18]. A szerződések kódja ugyan általában rövid, de gyakran minden egyes sor hatalmas értéket képvisel (pl. különböző javak vagy tokenek menedzselésével) [OHJ20].

Számos verifikációs módszer került kidolgozásra okosszerződésekhez, többek között statikus analízis [Tsa+18; Luu+16; Mue18; FGG19] és tételbizonyítók [Hil+18; Hir17] segítségével, ugyanakkor viszonylag kevés munkát fektettek a szerződések magasszintű, *funkcionális* tulajdonságainak automatizált ellenőrzésébe. A blokklánc tranzakcionális viselkedése miatt a *moduláris verifikáció* (modular verification) [Mül02] egy ígéretes módszer az okosszerződések ellenőrzésére. A *Boogie* [DL05] egy verifikáció-orientált köztes nyelv (intermediate verification language, IVL), amelyet több eszköz is támogat, például moduláris verifikációval [Bar+06]. A verifikáció egységei ebben az esetben a függvények, amelyeket különböző annotációkkal (pl. elő- és utófeltételek) lehet specifikálni. A moduláris program verifikáció ellenőrzésének célja, hogy az egyes modulok specifikációi teljesülnek-e, feltételezve a kapcsolódó modulok helyességét. Ezt többek között az egyes függvények SMT formulává alakításával (verifikációs feltétel) és ellenőrzésével lehet elérni.

3. tézis céljai. Ebben a tézisben a fő kutatási irány egy moduláris specifikáción és verifikáción alapuló, nagy *kifejezőerejű* és *hatékony* módszer kidolgozása okosszerződések magasszintű, funkcionális tulajdonságainak ellenőrzésére, a Boogie nyelv felhasználásával.

2. Új eredmények

Az új eredmények a három fő szakterület (konkurens és aszinkron rendszerek, beágyazott szoftverek, blokklánc-alapú decentralizált rendszerek) szerint vannak csoportosítva. A kutatások során általában más kutatókkal is együttműködtem, de mindenhol kiemelem a saját kontribúcióimat.

2.1. Petri-hálók CEGAR-alapú vizsgálatának kiterjesztése

Az eredeti algoritmus szerzői [WW11] csak részleges bizonyítást adtak a módszerük helyességére és nem vizsgálták meg azon problémák halmazát, amelyre konkluzív válasszal terminál. Korábbi munkánk során bebizonyítottuk, hogy az algoritmus egyik heurisztikája helytelen, azaz egy elérhető állapotot nem elérhetőként jelölhet meg [c4]. Erre a problémára egy lehetséges javítást is adtunk [j1]. Emellett bemutattuk egy olyan alosztályát a Petri-hálóknak, amelyre az algoritmus nem tud konkluzív választ adni [c4].

Ebben a disszertációban definiáljuk a *távoli invariánsok* fogalmát és egy új *iterációs stratégiát* javasolunk (1.3), amely a vizsgálható elérhetőségi problémák körét szélesíti [c5]. A kiegészítés ellenére a továbbfejlesztett algoritmus is tud inkonkluzív választ adni, de vizsgálatot adunk a módszer elméleti korlátaira [c5].

Az eredeti algoritmus további hiányossága, hogy csak egyszerű, kiterjesztések nélküli Petri-hálókra működik. Egy kiemelten fontos kiterjesztés az úgynevezett tiltó él (inhibitor arc), amely lehetővé teszi a tokenek hiányának vizsgálatát és ezáltal a Petri-hálók kifejezőerejét Turing teljessé teszi [Pet81]. Az új megkötéseket generáló heurisztika kiterjesztésével lehetővé tesszük, hogy az algoritmus *tiltó éles* Petri-hálók elérhetőségét is tudja vizsgálni (1.2) [c4]. Bár tiltó élek esetén az elérhetőségi probléma általános esetben eldönthetetlen [Chr99], bemutatunk olyan példákat, ahol a kiegészítésünk jól működik.

Az algoritmus kifejezőerejének további növelése érdekében lehetővé tesszük az úgynevezett *predikátum elérhetőségi* (reachability of predicates) probléma vizsgálatát (1.1) [c4]. A predikátum elérhetőség egy olyan általánosítása az elérhetőségi problémának, ahol az elérendő állapotot tetszőleges lineáris feltételekkel (predikátumokkal) lehet definiálni. Ez növeli az algoritmus kifejezőerejét, mert így például az elérendő állapotot részlegesen is lehet definiálni (pl. egy komponens állapota egy nagyobb rendszerben).

Bár az algoritmus az állapotteret az állapotegyenlet segítségével becsli felül, a lehetséges megoldások terét továbbra is be kell járnia. Jelen munkában szélességi és mélységi bejárással kísérletezünk és kidolgozunk egy *hibrid keresési stratégiát* (1.4) (a megoldások felett definiált részleges rendezés segítségével), amely az előbbi módszerek előnyeit egyesíti.

Az eredeti algoritmust és az új kiterjesztéseket a PetriDotNet modellező és analízis eszköz [c7] kiegészítőjeként implementáltuk. A PetriDotNet eszköz ingyen hozzáférhető[2] és mind az oktatásban, mind kutatási projektekben felhasználásra került a Budapesti Műszaki és Gazdaságtudományi Egyetemen. Az új kontribúciókat körülbelül 40 bemeneti példa modellen értékeljük ki (a Modellellenőrzési Versenyről [Kor+12] és néhány saját példán). Az eredményeink azt mutatják, hogy az új algoritmusok számos példára jobban teljesítenek a meglévő eszközöknél és módszereknél a konkluzív válaszokat és a kifejezőerőt illetően [c5]. A saját kontribúcióim az alábbi módon foglalhatók össze.

1. tézis Különböző kiegészítéseket és továbbfejlesztéseket javasoltam a Petri-hálók CEGAR-alapú elérhetőségi analíziséhez, amelyek a módszer kifejezőerejét és a konkluzív eredményeinek számát növelik.

- 1.1 Általánosítottam az algoritmust, hogy képes legyen olyan állapotok elérhetőségét vizsgálni, amelyek lineáris megkötésekkel vannak definiálva.
- 1.2 Kiegészítettem az algoritmust, hogy képes legyen tiltó éles Petri-hálókat kezelni, ezzel növelve a kifejezőerejét.
- 1.3 Definiáltam a távoli invariánsok fogalmát és egy új iterációs stratégiát javasoltam, amely az algoritmus által megoldható problémák körét bővíti.
- 1.4 A részleges megoldások között egy sorrendezést és egy ehhez kapcsolódó hibrid keresési stratégiát definiáltam, amely az algoritmus konvergenciáját gyorsítja anélkül, hogy megoldások vesznének el.

Közös munka. Vörös András és Bartha Tamás B.Sc. konzulenseimként vettek részt ebben a kutatásban. Az inkonkluzív válaszokra vonatkozó bizonyítást Vörös András adta meg a disszertációjában [Vör18]. Mártonka Zoltán hallgatótársam kifejlesztett és megvalósított bizonyos optimalizációkat, illetve ő volt felelős a helyesség ellenpéldájáért.

Publikációk. A tiltó élek és a lineáris megkötések elérhetőségi analízise először az SPLST 2013 konferencián került bemutatásra [c4], majd később az Acta Cybernetica folyóiratban lett részleteseb-

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

ben kifejtve [j1]. A távoli invariánsok a szerző B.Sc. szakdolgozatában lettek definiálva [a20], majd a Petri Nets 2015 konferencián bemutatva a hibrid keresési stratégiával együtt [c5]. A PetriDotNet eszköz implementációja (beleértve a tézisben foglalt algoritmusokat megvalósító kiegészítőt) a Petri Nets 2016 konferencián lett bemutatva [c7] majd a Science of Computer Programming folyóiratban lett bővebben kifejtve alkalmazási példákkal [j2].

2.2. Hatékony stratégiák CEGAR-alapú szoftver modellellenőrzéshez

Korábbi munkánk során definiáltunk egy általános CEGAR keretrendszert tranzíciós rendszerekkel leírt programokhoz, amely lehetővé tette különböző módszerek kombinációját [c6]. A keretrendszer sikeresen támogatta a predikátumok és az explicit értékek használatát, emellett különböző interpolációs stratégiákat is magában foglalt. Később ezt a keretrendszert általánosítottuk, hogy vezérlési folyam automatákkal leírt programokat is támogasson [c9].

Ez a munka vezet el minket a jelenlegi tézishez, ahol számos továbbfejlesztést javasolunk a CEGAR módszer absztrakciós és finomítási fázisaihoz [j3]. Az absztrakcióhoz definiálunk egy kiterjesztett explicit-érték domént, amely képes a rákövetkező állapotokat *korlátosan felsorolni* (2.1) abban az esetben, ha egy kifejezést nem lehet egyértelműen kiértékelni (az absztrakció természete miatt). Ez ugyan minimális többletmunkával jár, de a befektetett munka később a megnövekedett precizitással megtérül. Emellett javaslunk egy olyan új *keresési stratégiát* (2.2) az absztrakt állapottérben, amely a program strukturális információját használja fel annak érdekében, hogy hatékonyabban irányítsa a keresést az ellenpéldák felé. Ez a módszer akkor is segíthet, ha a vizsgált program helyes, ugyanis a CEGAR algoritmus köztes iterációi során felléphetnek (absztrakt) ellenpéldák.

A finomításhoz kidolgozunk egy *hátrafelé keresésen alapuló interpolációs stratégiát* (2.3), amely az absztrakt ellenpélda végrehajthatatlanságának okát a program legkorábbi pontjára vezeti vissza. Emellett bevezetünk egy olyan módszert, amely *több ellenpéldát* (2.4) gyűjt össze az absztrakció során és azokat egyszerre finomítja, ezáltal lehetővé téve az ellenpéldák közötti információcserét. A finomításhoz javasolt mindkét új kontribúció célja a konvergencia felgyorsítása a megfelelő precizitás felé.

A CEGAR algoritmust és a továbbfejlesztéseit a nyílt-forráskódú[3] Theta keretrendszerben implementáltuk [c9]. Az új kontribúciókat 445 bemeneti modellen értékeljük ki a Szoftver Verifikációs Versenyről (Competition on Software Verification) [Bey15], és 90 példa PLC programon a CERN-ből [Fer+15]. Az eredményeink alapján számos kategóriát ki tudunk emelni a példákból, ahol az új kontribúciók szignifikánsan javították a módszer hatékonyságát. A saját kontribúcióim az alábbi módon foglalhatók össze.

2. tézis Különböző kiegészítéseket és stratégiákat javasoltam CEGAR-alapú szoftver modellellenőrzéshez, amelyek a módszer hatékonyságát növelik.

- 2.1 Általánosítottam az explicit-érték analízist, hogy képes legyen egy előre definiált, de konfigurálható számú rákövetkező állapot felsorolására, ezáltal növelve az algoritmus precizitását állapottér robbanás nélkül.
- 2.2 A CEGAR kontextusába adaptáltam egy keresési stratégiát, amely a hibás állapot távolságát az absztrakt állapottérben a program struktúrája alapján becsli, ezáltal hatékonyan irányítva a felderítést az ellenpéldák felé.
- 2.3 Bevezettem egy hátrafelé keresésen alapuló interpolációs stratégiát, amely a végre nem hajtható ellenpélda okát a program lehető legkorábbi pontjára vezeti vissza, így eredményezve gyorsabb finomítási konvergenciát.
- 2.4 Definiáltam egy absztrakciófinomítási módszert, amely több ellenpéldát használ fel és képes jobb minőségű finomításokat adni az ellenpéldák közötti információk megosztásával.

Közös munka. Vörös András és Tóth Tamás a tranzíciós rendszerekre vonatkozó általános keretrendszer kifejlesztésében M.Sc. konzulenseimként vettek részt. Majzik István – Tóth Tamás Ph.D. té-

[3] https://github.com/FTSRG/theta

mavezetője – is segítette munkámat tanácsokkal és visszajelzésekkel. Az új stratégiák kidolgozásában Micskei Zoltán a Ph.D. témavezetőmként vett részt. A Theta implementációja Tóth Tamással közös munka. Ő elsősorban a keretrendszer közös funkcióiért és az időzített algoritmusokért felelt, míg én a CEGAR algoritmust fejlesztettem tranzíciós rendszerekhez és programokhoz. A C nyelvi frontend-et egy általam is konzultált M.Sc. hallgató, Sallai Gyula fejlesztette.

Publikációk. A tranzíciós rendszerekre vonatkozó általános keretrendszer a szerző M.Sc. diplomatervében lett definiálva [a21] majd a FORTE 2016 konferencián került publikálásra [c6]. Előzetes kísérletek és kiértékelések a BME Ph.D. Miniszimpóziumain kerültek bemutatásra [e12; e13]. Az absztrakciós és a finomítási algoritmus kiegészítései a Journal of Automated Reasoning folyóiratban lettek publikálva [j3]. A Theta eszköz implementációja az FMCAD 2017 konferencián került bemutatásra [c9], a C frontend pedig a VPT 2017 workshop-on [c8].

2.3. Okosszerződések moduláris specifikációja és verifikációja*

Ebben a tézisben egy moduláris specifikáción és verifikáción alapuló ellenőrzési módszert definiálunk Solidity nyelven [Eth18] írt okosszerződésekhez. Az okosszerződések kontextusába ültetünk át számos létező *specifikációs megoldást* (3.1) (pl. assert, elő- és utófeltétel, invariánsok) [c10]. Ezeket a tulajdonságokat a Solidity nyelv kiegészítésével a kódban lehet *annotációk* segítségével megadni. Emellett olyan *szakterület specifikus tulajdonságokat* (3.2) is ellenőrizhetővé teszünk (pl. egyenlegek összege), amelyek Solidity-ben és a verifikációs logikában közvetlenül nem fejezhetők ki [c10].

Definiálunk egy *leképezést* (3.3) az annotációkkal ellátott Solidity szerződésekről a Boogie köztes verifikációs nyelvre [c10]. Ez lehetővé teszi, hogy a verifikációs feltételeket automatikusan ellenőrizzük meglévő moduláris verifikációs eszközök és logikai megoldók segítségével. Bár a leképezés jelentős része hasonló a hagyományos programverifikációhoz, számos kihívást jelentő blokklánc-specifikus részletet kell kezelni, amely általános programozási nyelvekben nem gyakori. Az aritmetikai műveletekhez egy olyan *elkódolást* (3.4) dolgozunk ki maradékos osztás segítségével, amely megtartja a végrehajtás bitszintű precizitását, de közben gyakorlati méretű bit szélességekre (256 bit) skálázódik akár nemlineáris aritmetikai kifejezések esetén is [c10]. Ez lehetővé teszi hogy alul- és túlcsordulást ellenőrizzünk anélkül, hogy túl sok hamis riasztást adnánk.

A leképezést a nyílt-forráskódú[4] solc-verify eszközben [c10] implementáltuk, amely a Solidity fordítóra és a Boogie verifikációs eszközre épít. A módszerünket számos annotációkkal ellátott és anélküli valós példára értékeljük ki. Az eredmények alapján a módszerünk minimális felhasználói befektetéssel talál hibákat és bizonyítja a kijavított szerződések helyességét. A saját kontribúcióim az alábbi módon foglalhatók össze.

> **3. tézis** Definiáltam egy moduláris specifikációs és verifikációs módszert okosszerződésekhez, amely annotációkon és köztes verifikációs nyelvre való leképezésen alapul.
>
> 3.1 Az irodalomban létező moduláris specifikációs megoldásokat adaptáltam az okosszerződések kontextusába.
>
> 3.2 Szakterület-specifikus annotációkat javasoltam az okosszerződések moduláris specifikálására és verifikációjára.
>
> 3.3 Bevezettem egy leképezést a Solidity szerződés-orientált nyelvről a Boogie köztes verifikációs nyelvre.
>
> 3.4 Definiáltam egy maradékos osztáson alapuló leképezést aritmetikai műveletekhez, amely skálázható bitprecíz verifikációt tesz lehetővé.

Közös munka. Jovanović Dejan az SRI International szakmai gyakorlat során konzulensként vett részt a kutatásban. Emellett ő töltötte le a szerződéseket és futtatta rajtuk az eszközünket. Továbbá

*A jelen tézisben leírt munka során a szerző az SRI International (https://www.sri.com) alkalmazásában is állt.

[4] https://github.com/SRI-CSL/solidity

Michael Emmi és Ciocarlie Gabriela is segített tanácsokkal és visszajelzésekkel a közös beszélgetéseink során.

Publikációk. Az eredmények és az implementáció a VSTTE 2019 konferencián kerültek bemutatásra [c10]. Emellett egy fejlesztő-orientált előadást tartottam a Solidity Summit 2020 konferencián.[5] A referenciatípusok precíz modellezését leíró cikk az ESOP 2020 konferencián került publikálásra [c11] és az SMT 2020 workshop prezentációs szekciójába is elfogadásra került.[6] Továbbá 2018 decemberben egy szabadalmi kérés is be lett adva az Amerikai Egyesült Államokban, amely többek között az én kontribúcióimat is tartalmazza.

3. Az új eredmények gyakorlati felhasználása

3.1. Petri-hálók CEGAR-alapú vizsgálatának kiterjesztése

A CEGAR algoritmus és az új kontribúciók a PetriDotNet [c7] keretrendszerben kerültek implementálásra, amelyet a Budapesti Műszaki és Gazdaságtudományi Egyetemen az oktatásban és kutatási projektekben használnak. Az algoritmus az *evopro*[7] cégnél közösségi közlekedés modellezésére és analízisére lett felhasználva egy szakmai gyakorlat során [j2]. Emellett a PetriDotNet a BME Formális módszerek házi feladatainak egyik demonstrációs eszköze [c7; j2].

3.2. Hatékony stratégiák CEGAR-alapú szoftver modellellenőrzéshez

Az általános CEGAR keretrendszer és az algoritmikus kiegészítések a Theta nyílt-forráskódú keretrendszer részeként kerültek implementálásra [c9]. Egy CERN-nel közös projekt során a Theta a PLCverif[8] [DBM19] eszközhöz lett integrálva mint verifikációs szolgáltatás (backend). A PLCverif eszköz a PLC (programozható logikai vezérlő, programmable logic controller) programok forráskódját egy köztes (CFA-szerű) reprezentációra alakítja, amely különböző modellellenőrző eszközök bemenetére képezhető le [DFB15]. A Theta sikeresen integrációra került a PLCverif-hez, és egy széleskörű, 90 példa kódon végzett teljesítmény kiértékelés megmutatta, hogy a Theta két konfigurációja (köztük az új kontribúciók) képes az összes modellt verifikálni.

Számos hallgatói munka, szakdolgozat és diplomaterv épül a Theta eszközre [Czi16; Far16; FB18; Teg18], az általános CEGAR keretrendszerre [Sal16; ST17; Baj18; Dob19; MV20] és az új kontribúciókra [Sal19]. Emellett a Theta eszköz az oktatásban is felhasználásra került a Kritikus architektúrák laboratórium tárgyban, ahol a hallgatók korlátos modellellenőrző és absztrakció-alapú algoritmusokat implementálnak.

3.3. Okosszerződések moduláris specifikációja és verifikációja

A specifikációs és verifikációs módszer a nyílt-forráskódú solc-verify eszközben [c10] van implementálva. A solc-verify eszköz a Coimbrai Egyetemmel közös projektben (TÉT-16-PT) került felhasználásra. A projekt célja az okosszerződésekbe injektált hibák rendszerre mért hatásának vizsgálata volt. Az eredmények azt mutatták, hogy a solc-verify használatával szignifikánsan csökkenthető a felderítetlen hibák száma.

Emellett a solc-verify eszköz segítségével különböző interfész implementációk viselkedési ekvivalenciáját is vizsgálták [Bei+20].

[5] https://solidity-summit.ethereum.org/
[6] https://fscd-ijcar-2020.org/workshops#SMT
[7] http://www.evopro.hu
[8] http://cern.ch/plcverif/

4. Publikációs lista

Publikációk száma:	19
Angol nyelvű lektorált folyóiratcikkek száma:	3
WoS vagy Scopus által indexelt folyóiratcikkek száma:	3
Angol nyelvű publikációk a szerző legalább 50%-os hozzájárulásával:	8
Lektorált publikációk száma:	18
Független hivatkozások száma:	30

4.1. Tézisekhez kapcsolt publikációk

	Folyóirat-cikkek	Nemzetközi konferenciacikkek	Helyi cikkek
1. tézis	[j1] [j2]	[c4] [c5] [c7]	—
2. tézis	[j3]	[c6] [c8] [c9]	[e12] [e13]
3. tézis	—	[c10] [c11]	—

A cikkek besorolása a doktori iskola publikációs pontozásához használt besorolást követi.

Folyóiratcikkek

[j1] Ákos Hajdu, András Vörös, Tamás Bartha és Zoltán Mártonka. Extensions to the CEGAR approach on Petri nets. *Acta Cybernetica* 21(3), 2014, 401–417. old. DOI: 10.14232/actacyb.21.3.2014.8.

[j2] András Vörös, Dániel Darvas, Ákos Hajdu, Attila Klenik, Kristóf Marussy, Vince Molnár, Tamás Bartha és István Majzik. Industrial applications of the PetriDotNet modelling and analysis tool. *Science of Computer Programming* 157, 2018, 17–40. old. DOI: 10.1016/j.scico.2017.09.003.

[j3] Ákos Hajdu és Zoltán Micskei. Efficient strategies for CEGAR-based model checking. *Journal of Automated Reasoning* Online first, 2019. DOI: 10.1007/s10817-019-09535-x.

Nemzetközi konferencia és workshop cikkek

[c4] Ákos Hajdu, András Vörös, Tamás Bartha és Zoltán Mártonka. Extensions to the CEGAR approach on Petri nets. In: *Proceedings of the 13th Symposium on Programming Languages and Software Tools*, 274–288. old. University of Szeged, 2013.

[c5] Ákos Hajdu, András Vörös és Tamás Bartha. New search strategies for the Petri net CEGAR approach. In: *Application and Theory of Petri Nets and Concurrency*, Lecture Notes in Computer Science, 9115. köt., 309–328. old. Springer, 2015. DOI: 10.1007/978-3-319-19488-2_16.

[c6] Ákos Hajdu, Tamás Tóth, András Vörös és István Majzik. A configurable CEGAR framework with interpolation-based refinements. In: *Formal Techniques for Distributed Objects, Components and Systems*, Lecture Notes in Computer Science, 9688. köt., 158–174. old. Springer, 2016. DOI: 10.1007/978-3-319-39570-8_11.

[c7] András Vörös, Dániel Darvas, Vince Molnár, Attila Klenik, Ákos Hajdu, Attila Jámbor, Tamás Bartha és István Majzik. PetriDotNet 1.5: extensible Petri net editor and analyser for education and research. In: *Application and Theory of Petri Nets and Concurrency*, Lecture Notes in Computer Science, 9698. köt., 123–132. old. Springer, 2016. DOI: 10.1007/978-3-319-39086-4_9.

[c8] Gyula Sallai, Ákos Hajdu, Tamás Tóth és Zoltán Micskei. Towards evaluating size reduction techniques for software model checking. In: *Proceedings of the Fifth International Workshop on Verification and Program Transformation*, Electronic Proceedings in Theoretical Computer Science, 253. köt., 75–91. old. Open Publishing Association, 2017. DOI: 10.4204/EPTCS.253.7.

[c9] Tamás Tóth, Ákos Hajdu, András Vörös, Zoltán Micskei és István Majzik. Theta: a framework for abstraction refinement-based model checking. In: *Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design*, 176–179. old. 2017. DOI: 10.23919/FMCAD.2017.8102257.

[c10] Ákos Hajdu és Dejan Jovanović. Solc-verify: a modular verifier for Solidity smart contracts. In: *Verified Software. Theories, Tools, and Experiments*, Lecture Notes in Computer Science, 12301. köt., 161–179. old. Springer, 2020. DOI: 10.1007/978-3-030-41600-3_11.

[c11] Ákos Hajdu és Dejan Jovanović. SMT-friendly formalization of the Solidity memory model. In: *Programming Languages and Systems*, Lecture Notes in Computer Science, 12075. köt., 224–250. old. Springer, 2020. DOI: 10.1007/978-3-030-44914-8_9.

Helyi konferenciacikkek

[e12] Ákos Hajdu és Zoltán Micskei. Exploratory analysis of the performance of a configurable CEGAR framework. In: *Proceedings of the 24th PhD Mini-Symposium*, 34–37. old. Budapest University of Technology and Economics, Department of Measurement and Information Systems, 2017. DOI: 10.5281/zenodo.291895.

[e13] Ákos Hajdu és Zoltán Micskei. A preliminary analysis on the effect of randomness in a CEGAR framework. In: *Proceedings of the 25th PhD Mini-Symposium*, 32–35. old. Budapest University of Technology and Economics, Department of Measurement and Information Systems, 2018. DOI: 10.5281/zenodo.1219261.

4.2. Tézisekhez nem kapcsolt publikációk

Nemzetközi konferencia és workshop cikkek

[c14] Ákos Hajdu, Róbert Német, Szilvia Varró-Gyapay és András Vörös. Petri net based trajectory optimization. In: *ASCONIKK 2014: Extended Abstracts. Future Internet Services*, 11–19. old. University of Pannonia, 2014.

[c15] Bence Czipó, Ákos Hajdu, Tamás Tóth és István Majzik. Exploiting hierarchy in the abstraction-based verification of statecharts using SMT solvers. In: *Proceedings of the 14th International Workshop on Formal Engineering Approaches to Software Components and Architectures*, Electronic Proceedings in Theoretical Computer Science, 245. köt., 31–45. old. Open Publishing Association, 2017. DOI: 10.4204/EPTCS.245.3.

[c16] Rebeka Farkas, Tamás Tóth, Ákos Hajdu és András Vörös. Backward reachability analysis for timed automata with data variables. In: *Proceedings of the 18th International Workshop on Automated Verification of Critical Systems*, Electronic Communications of the EASST, 76. köt., 1–20. old. EASST, 2018. DOI: 10.14279/tuj.eceasst.76.1076.

Helyi konferenciacikkek

[e17] Rebeka Farkas és Ákos Hajdu. Activity-based abstraction refinement for timed systems. In: *Proceedings of the 24th PhD Mini-Symposium*, 18–21. old. Budapest University of Technology and Economics, Department of Measurement and Information Systems, 2017. DOI: 10.5281/zenodo.291891.

[e18] Viktória Dorina Bajkai és Ákos Hajdu. Software model checking with a combination of explicit values and predicates. In: *Proceedings of the 26th PhD Mini-Symposium*, 4–7. old. Budapest University of Technology and Economics, Department of Measurement and Information Systems, 2019. DOI: 10.5281/zenodo.2597969.

Kutatási jelentések

[r19] Ákos Hajdu. *Making the TTreeReader interface more accessible*. Techn. jel. CERN-STUDENTS-Note-2015-039. European Organization for Nuclear Research (CERN), 2015. aug.

4.3. További munkák

[a20] Ákos Hajdu. Extensions to the CEGAR Approach on Petri Nets. Bachelor's thesis. Budapest University of Technology és Economics, 2013.

[a21] Ákos Hajdu. A Survey on CEGAR-based Model Checking. Master's thesis. Budapest University of Technology és Economics, 2015.

[a22] Ákos Hajdu és Zoltán Micskei. *Supplementary Material for the paper "Efficient Strategies for CEGAR-based Model Checking"*. 2018. DOI: 10.5281/zenodo.1252784. (Dataset).

[a23] Ákos Hajdu, Dejan Jovanović és Gabriela Ciocarlie. Formal Specification and Verification of Solidity Contracts with Events. 2020. URL: https://arxiv.org/abs/2005.10382. (Preprint).

Hivatkozások

[ABC17] Nicola Atzei, Massimo Bartoletti és Tiziana Cimoli. A survey of attacks on Ethereum smart contracts. In: *Principles of Security and Trust*, Lecture Notes in Computer Science, 10204. köt., 164–186. old. Springer, 2017. DOI: 10.1007/978-3-662-54455-6_8.

[Amp+19] Elvio Amparore és tsai. Presentation of the 9th edition of the Model Checking Contest. In: *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, 11429. köt., 50–68. old. Springer, 2019. DOI: 10.1007/978-3-030-17502-3_4.

[AW18] Andreas Antonopoulos és Gavin Wood. *Mastering Ethereum: Building Smart Contracts and Dapps*. O'Reilly Media, 2018.

[Baj18] Viktória Dorina Bajkai. Combining Abstract Domains for Software Model Checking. Bachelor's Thesis. Budapest University of Technology és Economics, 2018.

[Bar+06] Mike Barnett, Bor-Yuh Evan Chang, Robert DeLine, Bart Jacobs és K Rustan M Leino. Boogie: a modular reusable verifier for object-oriented programs. In: *Formal Methods for Components and Objects*, Lecture Notes in Computer Science, 4111. köt., 364–387. old. Springer, 2006. DOI: 10.1007/11804192_17.

[BDW15] Dirk Beyer, Matthias Dangl és Philipp Wendler. Boosting k-induction with continuously-refined invariants. In: *Computer Aided Verification*, Lecture Notes in Computer Science, 9206. köt., 622–640. old. Springer, 2015. DOI: 10.1007/978-3-319-21690-4_42.

[Bei+20] Sidi Mohamed Beillahi, Gabriela Ciocarlie, Michael Emmi és Constantin Enea. Behavioral simulation for smart contracts. In: *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 470–486. old. ACM, 2020. DOI: 10.1145/3385412.3386022.

[Bey12] Dirk Beyer. Competition on software verification. In: *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, 7214. köt., 504–524. old. Springer, 2012. DOI: 10.1007/978-3-642-28756-5_38.

[Bey15] Dirk Beyer. Software verification and verifiable witnesses. In: *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, 9035. köt., 401–416. old. Springer, 2015. DOI: 10.1007/978-3-662-46681-0_31.

[Bey17] Dirk Beyer. Software verification with validation of results. In: *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, 10206. köt., 331–349. old. Springer, 2017. DOI: 10.1007/978-3-662-54580-5_20.

[BHM09] Armin Biere, Marijn Heule és Hans van Maaren. *Handbook of satisfiability*. IOS press, 2009.

[BHT07] Dirk Beyer, Thomas A Henzinger és Grégory Théoduloz. Configurable software verification: concretizing the convergence of model checking and program analysis. In: *Computer Aided Verification*, Lecture Notes in Computer Science, 4590. köt., 504–518. old. Springer, 2007. DOI: 10.1007/978-3-540-73368-3_51.

[Bie+99] Armin Biere, Alessandro Cimatti, Edmund M Clarke és Yunshan Zhu. Symbolic model checking without BDDs. In: *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, 1579. köt., 193–207. old. Springer, 1999. DOI: 10.1007/3-540-49059-0_14.

[BL13] Dirk Beyer és Stefan Löwe. Explicit-state software model checking based on CEGAR and interpolation. In: *Fundamental Approaches to Software Engineering*, Lecture Notes in Computer Science, 7793. köt., 146–162. old. Springer, 2013. DOI: 10.1007/978-3-642-37057-1_11.

[BLW15] Dirk Beyer, Stefan Löwe és Philipp Wendler. Refinement selection. In: *Model Checking Software*, Lecture Notes in Computer Science, 9232. köt., 20–38. old. Springer, 2015. DOI: 10.1007/978-3-319-23404-5_3.

[BT18] Clark Barrett és Cesare Tinelli. Satisfiability modulo theories. In: *Handbook of Model Checking*, 305–343. old. Springer, 2018. DOI: 10.1007/978-3-319-10575-8_11.

[Bur+90] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill és Lain-Jinn Hwang. Symbolic model checking: 10^{20} states and beyond. In: *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science*, 428–439. old. 1990. DOI: 10.1109/LICS.1990.113767.

[Cab+16] Gianpiero Cabodi, Carmelo Loiacono, Marco Palena, Paolo Pasini, Denis Patti, Stefano Quer, Danilo Vendraminetto, Armin Biere, Keijo Heljanko és Jason Baumgartner. Hardware model checking competition 2014: an analysis and comparison of solvers and benchmarks. *Journal on Satisfiability, Boolean Modeling and Computation* 9, 2016, 135–172. old.

[Cal+15] Cristiano Calcagno, Dino Distefano, Jeremy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter O'Hearn, Irene Papakonstantinou, Jim Purbrick és Dulma Rodriguez. Moving fast with software verification. In: *NASA Formal Methods*, Lecture Notes in Computer Science, 9058. köt., 3–11. old. Springer, 2015. DOI: 10.1007/978-3-319-17524-9_1.

[CE82] Edmund M Clarke és E Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In: *Logics of Programs*, Lecture Notes in Computer Science, 131. köt., 52–71. old. Springer, 1982. DOI: 10.1007/BFb0025774.

[CGL94] Edmund M Clarke, Orna Grumberg és David E Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems* 16(5), 1994, 1512–1542. old. DOI: 10.1145/186025.186051.

[Cho+20] Nathan Chong, Byron Cook, Konstantinos Kallas, Kareem Khazem, Felipe R Monteiro, Daniel Schwartz-Narbonne, Serdar Tasiran, Michael Tautschnig és Mark R Tuttle. Code level model-checking in the software development workflow. In: *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2020. (In press).

[Chr99] Piotr Chrząstowski-Wachtel. Testing undecidability of the reachability in Petri nets with the help of 10th Hilbert problem. In: *Application and Theory of Petri Nets 1999*, Lecture Notes in Computer Science, 1639. köt., 268–281. old. Springer, 1999. DOI: 10.1007/3-540-48745-X_16.

[Chu36] Alonzo Church. A note on the Entscheidungsproblem. *The Journal of Symbolic Logic* 1(1), 1936, 40–41. old.

[Cla+03] Edmund M Clarke, Orna Grumberg, Somesh Jha, Yuan Lu és Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM* 50(5), 2003, 752–794. old. DOI: 10.1145/876638.876643.

[Cla+18] Edmund M Clarke, Thomas A Henzinger, Helmut Veith és Roderick P Bloem. *Handbook of model checking*. Springer, 2018. DOI: 10.1007/978-3-319-10575-8.

[Cze+19] Wojciech Czerwiundefinedski, Sławomir Lasota, Ranko Laziundefined, Jérôme Leroux és Filip Mazowiecki. The reachability problem for Petri nets is not elementary. In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, 24–33. old. ACM, 2019. DOI: 10.1145/3313276.3316369.

[Czi16] Bence Czipó. Hierarchical Abstraction for the Verification of State-based Systems. Bachelor's Thesis. Budapest University of Technology és Economics, 2016.

[Dar+18] Priyanka Darke, Sumanth Prabhu, Bharti Chimdyalwar, Avriti Chauhan, Shrawan Kumar, Animesh Basakchowdhury, R Venkatesh, Advaita Datar és Raveendra Kumar Medicherla. VeriAbs: verification by abstraction and test generation. In: *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, 10806. köt., 457–462. old. Springer, 2018. DOI: 10.1007/978-3-319-89963-3_32.

[Dat18] NIST National Vulnerability Database. *CVE-2018-10299: Beauty Ecosystem Coin (BEC) "batchOverflow" issue*. 2018. URL: https://nvd.nist.gov/vuln/detail/CVE-2018-10299.

[DBM19] Dániel Darvas, Enrique Blanco Viñuela és Vince Molnár. PLCverif re-engineered: An open platform for the formal analysis of PLC programs. In: *Proceedings of the 17th International Conference on Accelerator and Large Experimental Physics Control Systems*, JACoW, 2019.

[Dem+17] Yulia Demyanova, Thomas Pani, Helmut Veith és Florian Zuleger. Empirical software metrics for benchmarking of verification tools. *Formal Methods in System Design* 50(2), 2017, 289–316. old. DOI: 10.1007/s10703-016-0264-5.

[DFB15] Dániel Darvas, Borja Fernández Adiego és Enrique Blanco Viñuela. PLCverif: A tool to verify PLC programs based on model checking techniques. In: *Proceedings of the 15th International Conference on Accelerator and Large Experimental Physics Control Systems*, 911–914. old. JACoW, 2015. DOI: 10.18429/JACoW-ICALEPCS2015-WEPGF092.

[Dij76] Edsger Wybe Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[DL05] Robert DeLine és K Rustan M Leino. *BoogiePL: A typed procedural language for checking object-oriented programs*. Techn. jel. MSR-TR-2005-70. Microsoft Research, 2005.

[DMH17] Vikram Dhillon, David Metcalf és Max Hooper. The DAO hacked. In: *Blockchain Enabled Applications*, 67–78. old. Springer, 2017. DOI: 10.1007/978-1-4842-3081-7_6.

[Dob19] Mihály Dobos-Kovács. Combining testing and formal verification in automotive software development. Bachelor's thesis. Budapest University of Technology és Economics, 2019.

[Eth18] Ethereum. *Solidity Documentation*. 2018. URL: https://solidity.readthedocs.io/en/v0.4.25/.

[Far16] Rebeka Farkas. Verification of Timed Automata by CEGAR-Based Algorithms. Master's thesis. Budapest University of Technology és Economics, 2016.

[FB18] Rebeka Farkas és Gábor Bergmann. Towards reliable benchmarks of timed automata. In: *Proceedings of the 25th PhD Mini-Symposium*, 20–23. old. Budapest University of Technology and Economics, Department of Measurement and Information Systems, 2018.

[Fer+15] Borja Fernández Adiego, Dániel Darvas, Enrique Blanco Viñuela, Jean-Charles Tournier, Simon Bliudze, Jan Olaf Blech és Víctor M González Suárez. Applying model checking to industrial-sized PLC programs. *IEEE Transactions on Industrial Informatics* 11(6), 2015, 1400–1410. old. DOI: 10.1109/TII.2015.2489184.

[FGG19] Josselin Feist, Gustavo Greico és Alex Groce. Slither: a static analysis framework for smart contracts. In: *Proceedings of the 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain*, 8–15. old. IEEE, 2019. DOI: 10.1109/WETSEB.2019.00008.

[Flo67] Robert W Floyd. Assigning meanings to programs. In: *Proceedings of Symposia in Applied Mathematics Vol. 19*, 19–32. old. 1967.

[GD19] Mitchell J Gerrard és Matthew B Dwyer. ALPACA: a large portfolio-based alternating conditional analysis. In: *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings*, 35–38. old. IEEE, 2019. DOI: 10.1109/ICSE-Companion.2019.00032.

[God91] Patrice Godefroid. Using partial orders to improve automatic verification methods. In: *Computer-Aided Verification*, Lecture Notes in Computer Science, 531. köt., 176–185. old. Springer, 1991. DOI: 10.1007/BFb0023731.

[GS97] Susanne Graf és Hassen Saidi. Construction of abstract state graphs with PVS. In: *Computer Aided Verification*, Lecture Notes in Computer Science, 1254. köt., 72–83. old. Springer, 1997. DOI: 10.1007/3-540-63166-6_10.

[Hil+18] Everett Hildenbrandt és tsai. KEVM: a complete formal semantics of the Ethereum virtual machine. In: *Proceedings of the IEEE 31st Computer Security Foundations Symposium*, 204–217. old. IEEE, 2018. DOI: 10.1109/CSF.2018.00022.

[Hir17] Yoichi Hirai. Defining the Ethereum virtual machine for interactive theorem provers. In: *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, 10323. köt., 520–535. old. Springer, 2017. DOI: 10.1007/978-3-319-70278-0_33.

[Hoa69] Charles A R Hoare. An axiomatic basis for computer programming. *Communications of the ACM* 12(10), 1969, 576–580. old. DOI: 10.1145/363235.363259.

[JD16] Dejan Jovanović és Bruno Dutertre. Property-directed k-induction. In: *Proceedings of the 2016 Conference on Formal Methods in Computer-Aided Design*, 85–92. old. IEEE, 2016. DOI: 10.1109/FMCAD.2016.7886665.

[JM09] Ranjit Jhala és Rupak Majumdar. Software model checking. *ACM Computing Surveys* 41(4), 2009, 1–54. old. DOI: 10.1145/1592434.1592438.

[Kor+12] Fabrice Kordon, Alban Linard, Didier Buchs, Maximilien Colange, Sami Evangelista, Kai Lampka, Niels Lohmann, Emmanuel Paviot-Adet, Yann Thierry-Mieg és Harro Wimmel. Report on the model checking contest at Petri nets 2011. In: *Transactions on Petri Nets and Other Models of Concurrency VI*, Lecture Notes in Computer Science, 7400. köt., 169–196. old. Springer, 2012. DOI: 10.1007/978-3-642-35179-2_8.

[Kos82] S Rao Kosaraju. Decidability of reachability in vector addition systems. In: *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, 267–281. old. ACM, 1982. DOI: 10.1145/800070.802201.

[Luu+16] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena és Aquinas Hobor. Making smart contracts smarter. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 254–269. old. ACM, 2016. DOI: 10.1145/2976749.2978309.

[May81] Ernst W Mayr. An algorithm for the general Petri net reachability problem. In: *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, 238–246. old. ACM, 1981. DOI: 10.1145/800076.802477.

[McC62] John McCarthy. Towards a mathematical science of computation. In: *IFIP Congress*, 21–28. old. 1962.

[Mey19] Bertrand Meyer. *Soundness and completeness: with precision*. 2019. URL: https://bertrandmeyer.com/2019/04/21/soundness-completeness-precision/.

[Mue18] Bernhard Mueller. Smashing Ethereum smart contracts for fun and real profit. In: *Proceedings of the 9th Annual HITB Security Conference*, 2018.

[Mül02] Peter Müller. *Modular specification and verification of object-oriented programs*. Springer, 2002. DOI: 10.1007/3-540-45651-1.

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

[MV20] Milán Mondok és András Vörös. Abstraction-based model checking of linear temporal properties. In: *Proceedings of the 27th PhD Mini-Symposium*, 29–32. old. Budapest University of Technology and Economics, Department of Measurement and Information Systems, 2020.

[Nak08] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008. URL: http://www.bitcoin.org/bitcoin.pdf.

[OHJ20] Gustavo A Oliva, Ahmed E Hassan és Zhen Ming (Jack) Jiang. An exploratory study of smart contracts in the Ethereum blockchain platform. *Empirical Software Engineering*, 2020. DOI: 10.1007/s10664-019-09796-5.

[Pel93] Doron Peled. All from one, one for all: on model checking using representatives. In: *Computer Aided Verification*, Lecture Notes in Computer Science, 697. köt., 409–423. old. Springer, 1993. DOI: 10.1007/3-540-56922-7_34.

[Pet81] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall, 1981.

[QS82] Jean-Pierre Queille és Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In: *International Symposium on Programming*, Lecture Notes in Computer Science, 137. köt., 337–351. old. Springer, 1982. DOI: 10.1007/3-540-11494-7_22.

[Ric53] Henry Gordon Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society* 74(2), 1953, 358–366. old.

[RW19] Cedric Richter és Heike Wehrheim. PeSCo: predicting sequential combinations of verifiers. In: *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, 11429. köt., 229–233. old. Springer, 2019. DOI: 10.1007/978-3-030-17502-3_19.

[Sal16] Gyula Sallai. Development of a Verification Compiler for C Programs. Bachelor's Thesis. Budapest University of Technology és Economics, 2016.

[Sal19] Gyula Sallai. LLVM IR-based Transformations for Software Model Checking. Master's thesis. Budapest University of Technology és Economics, 2019.

[Sch86] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., 1986.

[ST17] Gyula Sallai és Tamás Tóth. Boosting software verification with compiler optimizations. In: *Proceedings of the 24th PhD Mini-Symposium*, 66–69. old. Budapest University of Technology and Economics, Department of Measurement and Information Systems, 2017. DOI: 10.5281/zenodo.291903.

[Sza94] Nick Szabo. *Smart contracts*. 1994.

[Teg18] Tamás Tegzes. Applying Incremental, Inductive Model Checking to Software. Bachelor's thesis. Budapest University of Technology és Economics, 2018.

[Tsa+18] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Bünzli és Martin Vechev. Securify: practical security analysis of smart contracts. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 67–82. old. ACM, 2018. DOI: 10.1145/3243734.3243780.

[Tul+14] Varun Tulsian, Aditya Kanade, Rahul Kumar, Akash Lal és Aditya V Nori. MUX: algorithm selection for software model checkers. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*, 132–141. old. ACM, 2014. DOI: 10.1145/2597073.2597080.

[Tur36] Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. *Journal of Math* 58, 1936, 345–363. old.

[Val91] Antti Valmari. Stubborn sets for reduced state space generation. In: *Advances in Petri Nets 1990*, Lecture Notes in Computer Science, 483. köt., 491–515. old. Springer, 1991. DOI: 10.1007/3-540-53863-1_36.

[Vör18] András Vörös. Symbolic Verification of Petri Net Based Models. Dissz. Budapest University of Technology és Economics, 2018.

[Woo17] Gavin Wood. *Ethereum: A secure decentralised generalised transaction ledger*. 2017. URL: https://ethereum.github.io/yellowpaper/paper.pdf.

[WW11] Harro Wimmel és Karsten Wolf. Applying CEGAR to the Petri net state equation. In: *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, 6605. köt., 224–238. old. Springer, 2011. DOI: 10.1007/978-3-642-19835-9_19.