



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

K-indukciós algoritmus fejlesztése a Theta verifikációs keretrendszerben

SZAKDOLGOZAT

Készítette
Jakab Richárd Benjámin

Konzulens
Dr. Vörös András

2020. december 3.

Tartalomjegyzék

Kivonat	i
Abstract	ii
1. Bevezetés	1
2. Háttérismeretek	3
2.1. Általános modellellenőrzés	3
2.2. Szoftververifikáció	3
2.3. K-indukció	3
2.4. A probléma formalizálása	4
3. K-indukciós algoritmus szoftverellenőrzésre	5
3.1. Control Flow Automata	5
3.2. A formalizált algoritmus	7
4. Implementáció	9
4.1. Theta keretrendszer	9
5. Kiértékelés	10
6. Összefoglaló	11
Köszönetnyilvánítás	12
Irodalomjegyzék	13

HALLGATÓI NYILATKOZAT

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

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

Budapest, 2020. december 3.

Jakab Richárd Benjámín
hallgató

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomatervnek. A sablon használata opcionális. Ez a sablon \LaTeX alapú, a *TeXLive* \TeX -implementációval és a PDF- \LaTeX fordítóval működőképes.

Abstract

This document is a L^AT_EX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* T_EX implementation, and it requires the PDF-L^AT_EX compiler.

1. fejezet

Bevezetés

A körülöttünk lévő világban számos helyen találunk olyan informatikai rendszereket, melyeknél a meghibásodás (hibás működés) következménye elfogadhatatlan. Hagyományosan ilyen területek az egészségügyi alkalmazások, légi közlekedés, atomenergia ipar, fegyverrendszerek stb., vagy például a szoftverrendszerek egy részcsoportha, így az autonóm járművezetés. Ezeket a rendszereket biztonságkritikus rendszereknek nevezzük, és létfontosságú a specifikációnak megfelelő működésük ellenőrzése.

A legtöbb biztonságkritikus rendszer rendelkezik komplex szoftverrendszerrel, melyek ugyanúgy biztonságkritikusak önmagukban is. Ezek ellenőrzésével a szoftververifikáció foglalkozik, mely azt vizsgálja, hogy egy szoftverrendszer megfelel-e a feléje támasztott követelményeknek. Ilyen követelmények lehetnek például a következők [4]:

- Rendelkezésre állás (*availability*) – Helyes szolgáltatás valószínűsége
- Megbízhatóság (*reliability*) – Folyamatos helyes szolgáltatás valószínűsége
- Biztonság (*safety*) – Elfogadhatatlan kockázattól való mentesség
- Integritás (*integrity*) – Hibás változás, változtatás elkerülésének lehetősége
- Karbantarthatóság (*maintainability*) – Javítás és fejlesztés lehetősége
- ...

Ennek ellenőrzésére különböző verifikációs technikák szolgálnak. Ezek egyike a modell-ellenőrzés, mely során a rendszer egy matematikai modelljét vizsgálva lehet különböző formalizált követelmények teljesülését ellenőrizni.

A munkám célja egy program leimplementálása mely a fentebb vázolt követelmények közül a biztonságosság követelmény teljesülését ellenőrzi. Ezt a programot a BME VIK Méréstechnika és Információs Rendszerek Tanszék¹ Hibatűrő Rendszerek Kutatócsoportja² által fejlesztett *Theta*³ verifikációs keretrendszerbe beillesztettem, majd azt széleskörűen teszteltem.

A munkámat három részre tagolhatjuk, melyet a szakdolgozatom felépítése is követ: először elmerültem a szoftververifikáció és modellezés tématerületében, kiemelten foglalkozva a *k*-indukció alapú szoftververifikációval, aztán a szakirodalom által bemutatott algoritmust leimplementáltam a *Theta* keretrendszerben, majd végezetül széleskörű

¹<https://www.mit.bme.hu/>

²<https://www.mit.bme.hu/research/ftsrg>

³<https://github.com/FTSRG/theta>

tesztelés alá vettem és így finomítottam az algoritmusomat.

A dolgozat az alábbi részletesebb tartalmi felosztásban tárgyalja a fentebb felvázolt folyamatot:

- A második fejezetben a szoftververifikációt mutatom be általános megközelítésben
- A harmadik fejezetben kifejtem az algoritmusom alapját adó K-indukció módszer elméleti háttérét
- A negyedik fejezetben bemutatom az algoritmusom elkészítésének folyamatait és technikai felépítését
- Az ötödik fejezetben bemutatom az algoritmusom teszteredményeit és összehasonlítom más, verifikáló algoritmusokkal

TODO: fejezet lista frissítése

2. fejezet

Háttérismeretek

Ebben a fejezetben a dolgozat további részeinek megértéséhez szükséges elméleti előismereteket mutatom be. Először a k -indukció nevű matematikai módszert [6] ismertetem (Alfejezet 2.1.), majd formalizálom a problémát (Alfejezet 2.2.), végül pszeudokód szinten bemutatom az ezen a matematikai módszeren alapuló algoritmus működését és annak helyességét (Alfejezet 2.3.).

2.1. Általános modellellenőrzés

2.2. Szoftververifikáció

2.3. K-indukció

Tekintsük az alább látható teljes indukció tételét a természetes számok halmaza fölött (kiegészítve 0-val):

$$P(0) \wedge \forall n(P(n) \Rightarrow P(n+1)) \Rightarrow \forall nP(n). \quad (2.1)$$

Lényege, hogy megnézzük az első lépésre teljesül-e a feltétel (az angol szakirodalomban ez a *base-case*). Ha igen, akkor megnézzük ennek tudatában azt, hogy az $n+1$. lépés következik-e az n . lépésből (indukciós lépés – *induction case*). Ha sikerül ezt belátnunk, akkor készen vagyunk, bebizonyítottuk az összes lépésre a feltételt.

Ezt tovább gondolva megtehetjük azt, hogy az első két lépésre nézzük meg, hogy teljesítik-e a feltételt:

$$P(0) \wedge P(1) \wedge \forall n((P(n) \wedge P(n+1)) \Rightarrow P(n+2)) \Rightarrow \forall nP(n). \quad (2.2)$$

Ezt az elvet általánosíthatjuk k lépésre, $k \geq 1$, melyet a irodalom [6] k -indukciónak nevez, formálisan:¹

$$\left(\bigwedge_{i=0}^{k-1} P(i) \right) \wedge \forall n \left(\left(\bigwedge_{i=0}^{k-1} P(n+i) \right) \Rightarrow P(n+k) \right) \Rightarrow \forall nP(n). \quad (2.3)$$

¹A k -indukció helyességének a bizonyítására a dolgozatomban nem térek ki.

2.4. A probléma formalizálása

Ahhoz, hogy a problémát precízebben megfogalmazhassuk, szükség van jelölések és fogalmak bevezetésére [5]. Adott egy tranzakciós relációkból felépülő gráf, melyben $T(x, y)$ -al jelöljük azt, ha létezik egy, az x állapotból az y állapotba mutató tranzakciós reláció. Így már tudjuk definiálni az útvonal fogalmát, mely állapotok sorozatát jelenti T -n keresztül:

$$utvonal(s_{[0..n]}) \doteq \bigwedge_{0 \leq i < n} T(s_i, s_{i+1}) \quad (2.4)$$

Ahol $s_{[0..n]}$ rövidítés az (s_0, s_1, \dots, s_n) állapotsorozatot jelöli. Az *utvonal* n hosszúságú, ha n darab tranzakcióból áll. A nulla hosszúságú *utvonal* egy darab állapotot tartalmaz és nem értelmezzük rajta a tranzakció műveletét. Azt a megállapítást, hogy egy Q tulajdonság igaz egy útvonal összes állapotára, úgy fogjuk írni, hogy $\forall.Q(s_{[0..n]})$.

Definiáljuk emellett a ciklus mentes útvonalat is: olyan útvonal, melyben minden állapot maximum csak egyszer szerepelhet:

$$cmUtvonal(s_{[0..n]}) \doteq utvonal(s_{[0..n]}) \wedge \bigwedge_{0 \leq i < j \leq n} s_i \neq s_j \quad (2.5)$$

A továbbiakban lesz olyan, mikor egy útvonal alatt nem csak azt értjük, hogy az tranzakciók sorozata, hanem annak létezését is jelöli. Így, $utvonal_i(s_0, s_i)$ alatt azt jelöljük, hogy *létezik* egy útvonal s_0 -ból s_i -be, mely i darab T -ből áll.

Legyen T egy tranzakciós reláció S állapothalmazán. Feltételezzük, hogy T a teljes állapottérre értelmezve van, tehát minden állapotnak (a kezdőállapotokat leszámítva) van egy szülőállapota T -n keresztül. Jelöljük I -vel a kezdőállapotokat, és azt vizsgáljuk, hogy az állapotok teljesítik-e a P tulajdonságot.

A problémát informálisan a következőképp foglalhatjuk össze: beszeretnénk azt látni, hogy ha egy kezdőállapotból elindulunk, akkor a tranzakciós relációt ismétlődően alkalmazva csak olyan állapotba fogunk eljutni, mely kielégíti P -t. Formálisan a következőt akarjuk belátni:

$$\forall i : \forall s_0 \dots s_i : (I(s_0) \wedge utvonal(s_{[0..i]}) \rightarrow P(s_i)) \quad (2.6)$$

Ahol $i \geq 0$. Később látni fogjuk, hogy az algoritmus felhasználja ennek a megfordítottját is: a „rossz” állapotokból (hibaállapotokból) elindulunk visszafelé, és azt vizsgáljuk, hogy elérjük-e valamelyik kezdőállapotot:

$$\forall i : \forall s_0 \dots s_i : (\neg I(s_0) \leftarrow utvonal(s_{[0..i]}) \wedge \neg P(s_i)) \quad (2.7)$$

Ahol $\neg I(s_0)$ azt jelenti, hogy s_0 nem kezdőállapot (nem teljesíti a „kezdőállapot tulajdonságot”), illetve $\neg P(s_i)$ azt, hogy s_i nem elégíti ki a P tulajdonságot. A két egyenlet ekvivalens és összehetők úgy, hogy azon a probléma szemléletesebb és szimmetrikusabb legyen:

$$\forall i : \forall s_0 \dots s_i : \neg(I(s_0) \wedge utvonal(s_{[0..i]}) \wedge \neg P(s_i)) \quad (2.8)$$

Azaz szavakkal elmondva – azt akarjuk megmutatni, hogy *nem létezik* olyan útvonal, mely kezdőállapotból indul és egy *nem-P* állapotba jut.

3. fejezet

K-indukciós algoritmus szoftverellenőrzésre

Ebben a fejezetben bemutatom azokat a technológiákat, melyek szükségesek a programom technikai oldalának megértéséhez. Először ismertetem a Control Flow Automata ábrázolás részleteit (Alfejezet 3.1)), aztán az előző fejezetben bemutatott jelölérendszerrel formalizálom az algoritmust (Alfejezet 3.2).

3.1. Control Flow Automata

A programkódokat sokféleképpen ábrázolhatjuk [3]. Legismertebb a programkód, melyet az ember könnyen, gyorsan tud olvasni illetve írni, ellenben a bájtkóddal, melyet a számítógép tud jóval hatékonyabban kezelni. A szoftveres modellellenőrzés elvégzéséhez a programkódot matematikailag pontos, formális ábrázolásban kell megadni, melyet a számítógép is jól tud kezelni. Egy széleskörűen ismert és használt ábrázolásmód a *Control Flow Automaton* (CFA), mely egy gráf alapú ábrázolást biztosít a programokhoz.

Szintaxis. A CFA formálisan egy $CFA = (V, H, I_0, E)$ négyes [1], ahol

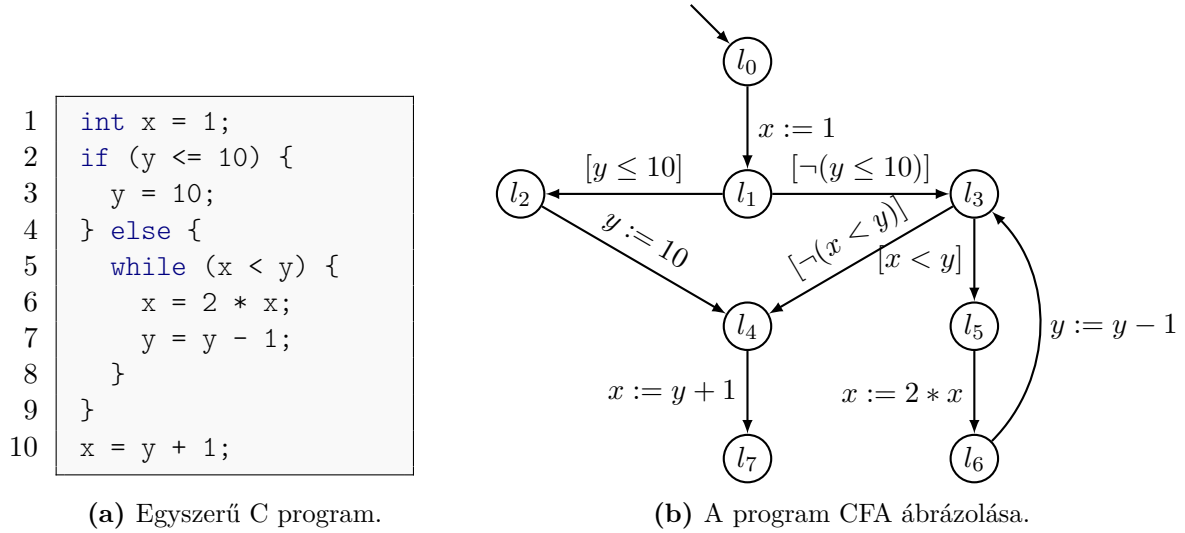
- $V = \{v_1, v_2, \dots\}$ a változók halmaza. Mindegyik $v_i \in V$ változó rendelkezik egy D_{v_i} doménnel, mely megszabja, hogy v_i milyen értékeket vehet fel,
- H a helyek halmaza,
- $I_0 \in H$ a kezdőállapota a gráfnak, a program belépőpontját mutatja,
- $E \subseteq H \times Op \times H$ az irányított élek halmaza, melyek helyeket kötnek össze és a változókra vonatkozó utasításokkal vannak felcímkézve

Utasítások. Háromféle utasítással foglalkozunk:

- A *hozzárendelés* utasítás a $v_i := expr$ összefüggéssel írható le. Azt jelöli, hogy a baloldali v_i változóhoz hozzárendeljük a jobb oldali kifejezést. Fontos, hogy az *expr* kifejezésnek is ugyanolyan doménnel kell rendelkeznie, mint a $v_i \in V$ változónak.
- A *feltevés* operátor a $[cond]$ formában írható le, ahol *cond* egy bináris (*Boolean*) kifejezés (feltétel). Ha egy él rendelkezik $[cond]$ feltétellel, akkor abban az esetben csakis akkor sült el (kerülünk át az egyik helyről a másikra), ha a feltétel teljesül. A feltétel változón nem változtat, arra nem hat ki.

- A *havoc* operátor a *havoc* v_i formában írható le, ahol $v_i \in V$ egy változó. A *havoc* hozzá rendel a v_i változóhoz egy nem-determinisztikus értéket, a többi változót érintetlenül hagyja. Például arra lehet használni, mikor szimulálni szeretnénk a felhasználói bemenetet.

Grafikai megjelenítés. A helyeket körök, az éleket nyilak jelölik. Az egyes élek felett illetve mellett láthatóak az utasítások, amely jelen esetben hozzárendelés vagy feltevés. A kezdőállapotot egy bejövő nyíllal jelöljük. [3].

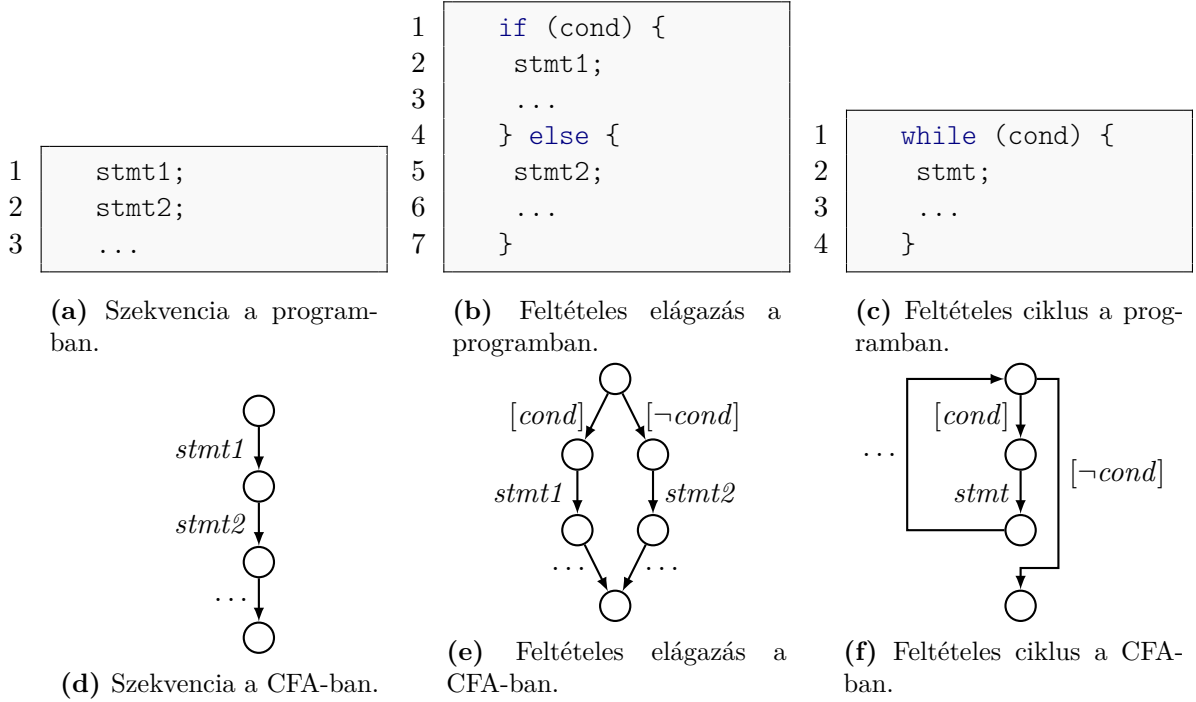


3.1. ábra. Egy C program és a hozzá tartozó Control Flow Automaton (CFA).

Példa 1. Egy C nyelvű program és egy hozzá tartozó CFA látható a 3.1 ábrán. A kezdőhely az l_0 , a termináló hely az l_7 , mely lehet végső- (final location) illetve hibahely (error location). Egy útvonal a kezdőhelytől az l_4 helyre leírható úgy, hogy $l_0 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4$. Az l_1 helyen egy elágazást figyelhetünk meg, ahol ha feltétel a $[y \leq 10]$ teljesül, akkor úgy a program az l_1 helyről továbbmegy az l_2 helyre, míg ha nem teljesül, akkor az l_3 helyre kerül a program. Az elágazásokban a kimenő élekre a feltételek úgy vannak megfogalmazva, hogy míg az egyikben az eredeti feltétel, addig a másikon annak a negáltja figyelhető meg. Ez azért van így, hogy szemléltesse az ábra, hogy ezt egy algoritmus fogja feldolgozni, amely számára ebben a formátumban könnyebb megadni az automatát.

Programábrázolás. A 3.2 ábra megmutatja, hogy az alap elemei a strukturált programozásnak miként képezhetők le CFA alakba [3].

- Szekvenciális állításokat (3.2a és 3.2d ábra) úttal reprezentáljuk, mely helyek és élek közt alternál.
- Feltételes elágazásokat (pl. ha-akkor állítások, 3.2b és 3.2e ábra) különvált utakkal tudunk reprezentálni őrfeltételekkel.
- Feltételes ciklusokat (3.2c és 3.2f ábra) a CFA-ban körökkel tudunk ábrázolni. Egy vezérlési hely felel a ciklusfejért, amelyből két kimenő él fut. Az egyik bemegy a ciklusba, a másik pedig kilép abból. A ciklusban további szekvenciák, elágazások vagy akár újabb ciklusok is történhetnek, azonban az út mindig visszatér a ciklusfejhez.



3.2. ábra. A strukturált programozás elemei (szekvencia, feltételes elágazás, feltételes ciklus) és a megvalósításuk CFA modelleken.

3.2. A formalizált algoritmus

A fenti ismeretek segítségével hogy tudnánk belátni, hogy a modell a P tulajdonságra nézve biztonságos?

Például úgy, ha megnézzük, hogy tetszőleges nemnegatív i egész esetén teljesül-e a

$$\forall s_0 \dots s_i : \neg(I(s_0) \wedge \text{utvonal}(s_{[0..i]}) \wedge \neg P(s_i)) \quad (3.1)$$

feltétel. Ha megsérül valamelyik elérhető állapotban, akkor ezzel a módszerrel meg fogjuk találni és az oda vezető útvonal ellenpélda lesz a modell P -biztonságosságára. Ez egy elvárható eredmény: az algoritmusnak két féle kimenetele kell, hogy legyen: vagy az, hogy a modell P -tulajdonságra nézve biztonságos (minden elérhető állapot teljesíti), vagy az, hogy a modell nem P -biztonságos, ekkor egy ellenpéldát is kell adnia, mely bizonyítja, hogy az adott kezdőállapotból elindulva, azon végighaladva valóban egy hibaállapotba kerülünk.

Ha a rendszer P -biztonságos, akkor (TODO) minden i -re igaz lesz, hiszen nem fogunk tudni találni olyan i értéket, melyre ne teljesülne. Felvetődhet a kérdés, hogy mikortól lehet azt mondani, hogy i további növelése céltalan, mert már teljes bizonyossággal kijelenthetjük, hogy a modell P -biztonságos? A $I(s_0) \wedge \text{utvonal}(s_{[0..i]})$ feltétel önmagában nem fog gyorsítást eredményezni: vagy végig megy az állapottéren amilyen hosszan csak lehetséges (ezt szeretnénk lerövidíteni), tekintve, hogy minden állapotnak van egy szülőállapota a T tranzakciós reláción keresztül, vagy végtelen ciklusba kerül.

Ennél jobb stratégia, ha akkor állunk meg, mikor $I(s_0) \wedge \text{cmUtvonal}(s_{[0..i]})$ ellentmondásos lesz. Ezt használva addig folytatjuk a keresést, míg az összes, ciklusmentes útvonalat be nem jártuk. Legrosszabb esetben ekkor is végigmegy a program a teljes állapottéren, viszont ha az állapottérben ciklikusság figyelhető meg, akkor azt a stratégia maximálisan kihasználja: nem fog végtelen ciklusba kerülni, illetve átlagosan rövidebb (de bizonyosan nem hosszabb) útvonalakat fog bejárni, mint az $I(s_0) \wedge \text{utvonal}(s_{[0..i]})$.

Ehhez hasonlóan tehetjük azt is, hogy addig ellenőrizzük, amíg a $cmUtvonal(s_{[0..i]}) \wedge \neg P(s_i)$ nem lesz ellentmondásos: egy, a P tulajdonságot sértő állapotból (hibaállapotból) kiindulva addig megyünk ciklusmentes útvonalakon visszafelé, míg be nem járjuk a teljes állapotteret (ez esetben kijelenthetjük, hogy a rendszer nem- P -biztonságos), ellenben ha nem járunk be minden *elérhető* állapotot, akkor a rendszer P -biztonságos (feltéve, hogy egyik hibaállapotból sem tudjuk bejárni a teljes rendszert). Ez azzal magyarázható, hogy ha a kezdőállapotok halmaza nem elérhető a hibaállapotokból (mert visszafele haladva útközben elakadunk), akkor kijelenthetjük, hogy a modell biztonságos.

A k -indukció alapú szoftververifikáció az előbb elmondottakra épül. A módszer lényege, hogy elindulunk mind a kezdőállapotokból, mind a hibaállapotokból: míg az előbbiekből előre felé, addig az utóbbiakból visszafelé. Kijelenthető, hogy a modell biztonságos, ha az előre felé haladó keresés bejárta a teljes elérhető állapotteret (nincs több ciklusmentes útvonal)¹, illetve akkor is biztonságos, ha a hátrafelé haladó keresés megakad.

A 2.3.-as fejezetben bemutatott, és így a módszer nevét adó *k-indukció* úgy kerül a képbe, hogy ha a modellt bejárjuk k mélységig, és belátjuk a fentebb említett metodika alapján, hogy a modell biztonságos, akkor kijelenthető, hogy $k+1$ mélységre is biztonságos lesz [2], illetve mellé az is, hogy ezzel az indukciós lépést bizonyítottuk [5]:

¹Természetesen ha közben hibaállapotba jutna, akkor a teljes modellellenőrzés megakadna, s így nem tudná bejárni a teljes állapotteret.

4. fejezet

Implementáció

4.1. Theta keretrendszer

A *Theta*¹ egy nyílt forráskódú, általános célú, moduláris és konfigurálható modellellenőrző keretrendszer, melyet absztrakciós finomításon alapuló algoritmusok tervezésének és értékelésének támogatására hoztak létre a különböző formalizmusok elérhetőségi elemzéséhez.

A keretrendszer a már évek óta tartó fejlesztéseknek köszönhetően számos eszközt tud nyújtani modellellenőrzéshez:²

- `theta-cfa-cli` – Control Flow Automata hibahelyeinek³ elérhetőségét vizsgálja CEGAR alapú algoritmusokkal
- `theta-sts-cli` – *Symbolic Transition Systems* biztonsági tulajdonságainak verifikációját végzi CEGAR alapú algoritmusokkal
- `theta-xta-cli` – Uppaal időzített automaták verifikációját lehet vele elvégezni
- `theta-xsts-cli` – *eXtended Symbolic Transition Systems* biztonsági tulajdonságainak verifikációját végzi CEGAR alapú algoritmusokkal

A Theta architektúrája négy rétegre osztható. Nevezetesen:

- **Formalizmusok** – A Theta legalapvetőbb elemei, melyek való-életbeli problémákat modelleznek le (pl. szoftvereket, hardvereket, protokollokat). A formalizmusok általában alacsony szintű, matematikai ábrázolások melyek elsőrendű logikai kifejezéseken és gráfszerű struktúrákon alapulnak. Ilyen például a *Symbolic Transition Systems*
- **Háttéranalízis** –
- **SMT megoldó** –
- **Eszközök** – Parancssori alkalmazások melyek futtatható jar fájlba fordíthatóak le. Jellemzően csak beolvassák az inputot és meghívják az alsóbb szinten lévő algoritmusokat.

¹<https://github.com/FTSRG/theta>

²2020 decemberének elején.

³Az angol irodalom a *location* kifejezést használja. Én a dolgozatomban a magyar megfelelőjét használtam.

5. fejezet

Kiértékelés

6. fejezet

Összefoglaló

Köszönetnyilvánítás

Irodalomjegyzék

- [1] Dirk Beyer – Stefan Löwe: Explicit-state software model checking based on CEGAR and interpolation. In *Fundamental Approaches to Software Engineering*. Lecture Notes in Computer Science sorozat, 7793. köt. 2013, Springer, 146–162. p.
- [2] Alastair F. Donaldson – Leopold Haller – Daniel Kroening – Philipp Rümmer: Software verification using k-induction. In Eran Yahav (szerk.): *Static Analysis* (konferenciaanyag). Berlin, Heidelberg, 2011, Springer Berlin Heidelberg, 351–368. p. ISBN 978-3-642-23702-7.
- [3] Dr. Hajdu Ákos: Formal software verification. URL: <https://ftsrg.mit.bme.hu/software-verification-notes/software-verification.pdf>, 2020. 11.
- [4] Dr. Majzik István: Rendszertervezés és -integráció. URL: https://www.mit.bme.hu/system/files/oktatas/targyak/10019/VIMIMA11_RTI_08_Biztonsagi_alapfogalmak_1.pdf, 2018. 12.
- [5] Mary Sheeran – Satnam Singh – Gunnar Stålmarck: Checking safety properties using induction and a sat-solver. In Warren A. Hunt – Steven D. Johnson (szerk.): *Formal Methods in Computer-Aided Design* (konferenciaanyag). Berlin, Heidelberg, 2000, Springer Berlin Heidelberg, 127–144. p. ISBN 978-3-540-40922-9.
- [6] Thomas Wahl: The k-induction principle. URL <http://www.comlab.ox.ac.uk/people/Thomas.Wahl/Publications/k-induction.pdf>.