



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. november 30.

# Tartalomjegyzék

<b>Kivonat</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>1. Bevezetés</b>	<b>1</b>
<b>2. Háttérismeretek</b>	<b>3</b>
2.1. K-indukció . . . . .	3
2.2. A probléma formalizálása . . . . .	3
2.3. A formalizált algoritmus . . . . .	4
<b>3. Theta verifikációs keretrendszer</b>	<b>6</b>
<b>4. Implementálás</b>	<b>7</b>
4.1. Adatszerkezet . . . . .	7
<b>5. Esettanulmány</b>	<b>8</b>
5.1. Teszteredmények . . . . .	8
<b>6. Összefoglaló</b>	<b>9</b>
<b>Köszönetnyilvánítás</b>	<b>10</b>
<b>Irodalomjegyzék</b>	<b>11</b>

## 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. november 30.

---

*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  $\text{\LaTeX}$  alapú, a *TeXLive*  $\text{\TeX}$ -implementációval és a PDF- $\text{\LaTeX}$  fordítóval működőképes.

# Abstract

This document is a L<sup>A</sup>T<sub>E</sub>X-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<sub>E</sub>X implementation, and it requires the PDF-L<sup>A</sup>T<sub>E</sub>X 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 [2]:

- 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 Tanszék<sup>1</sup> Hibatűrő Rendszerek Kutatócsoportja<sup>2</sup> által fejlesztett *Theta*<sup>3</sup> 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

---

<sup>1</sup><https://www.mit.bme.hu/>

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

<sup>3</sup><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 [4] 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. 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 [4]  $k$ -indukciónak nevez, formálisan:<sup>1</sup>

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

### 2.2. A probléma formalizálása

Ahhoz, hogy a problémát precízebben megfogalmazhassuk, szükség van jelölések és fogalmak bevezetésére [3]. 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

---

<sup>1</sup>A  $k$ -indukció helyességének a bizonyítására a dolgozatomban nem térek ki.



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 összetehetőek 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.

### 2.3. A formalizált algoritmus

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

Például úgy, ha megnézzük, hogy a

$$\forall s_0 \dots s_i : \neg(I(s_0) \wedge \text{utvonal}(s_{[0..i]}) \wedge \neg P(s_i)) \quad (2.9)$$

teljesül  $i = 0, i = 1, i = 2, \dots$  -re. Ha a feltétel 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 hosszban 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  $\text{cmUtvonal}(s_{[0..i]}) \wedge \neg P(s_i)$  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)<sup>2</sup>, illetve akkor is biztonságos, ha a hátrafelé haladó keresés megakad.

A 2.1.-es alfejezetben 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 [1], illetve mellé az is, hogy ezzel az indukciós lépést bizonyítottuk [3]:

folyt. köv.

<sup>2</sup>Természetesen ha közben hibaállapotba jutna, akkor a teljes modellellenőrzés megakadna, s így nem tudná bejárni a teljes állapotteret.

## 3. fejezet

# Theta verifikációs keretrendszer

A *Theta*<sup>1</sup> 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.

---

<sup>1</sup><https://github.com/FTSRG/theta>

## 4. fejezet

# Implementálás

### 4.1. Adatszerkezet

## 5. fejezet

# Esettanulmány

### 5.1. Teszteredmények

6. fejezet

Összefoglaló

# Köszönetnyilvánítás

# Irodalomjegyzék

- [1] 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.
- [2] Dr. Majzik István: Rendszertervezés és -integráció. URL: [https://www.mit.bme.hu/system/files/oktatas/targyak/10019/VIMIMA11\\_RTI\\_08\\_Biztonsagi\\_alapfogalmak\\_1.pdf](https://www.mit.bme.hu/system/files/oktatas/targyak/10019/VIMIMA11_RTI_08_Biztonsagi_alapfogalmak_1.pdf), 2018. 12.
- [3] 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.
- [4] Thomas Wahl: The k-induction principle. URL <http://www.comlab.ox.ac.uk/people/Thomas.Wahl/Publications/k-induction.pdf>.