BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
TÁVKÖZLÉSI ÉS MÉDIAINFORMATIKAI TANSZÉK

HIBÁK FELDERÍTÉSE, DIAGNOSZTIKÁJA ÉS KORREKCIÓJA FORMÁLIS MÓDSZEREKEN ALAPULÓ TÁVKÖZLÉSI SZOFTVER FEJLESZTÉS SORÁN

Pap Zoltán

Tézisfüzet

Tudományos témavezetők

Dr. Csopaki Gyula, Dr. Dibuz Sarolta és Dr. Tarnay Katalin
Távközlési és Médiainformatikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

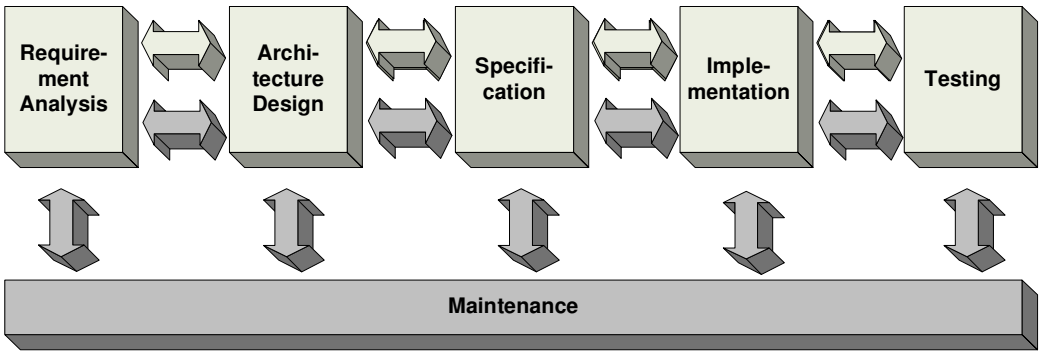
Budapest
2006

1. Bevezető

A távközlési szoftverek egyre fontosabb szerepet töltenek be mind a gazdasági életben, mind a társadalom számára, hiszen nélkülözhetetlen építőelemei napjaink kommunikációs infrastruktúrájának. Ahogy egyre jelentősebbé válnak, ellentmondó feltételeknek kell megfelelniük: Egyrészt egyre komplexebb szolgáltatásokat kell biztosítaniuk, másrészt megbízhatóan kell működniük, hiszen alkalmazások ezrei épülnek rájuk.

A komplexitás növekedésével egyre nagyobb lesz az esélye annak, hogy hibák kerülnek a rendszerbe a fejlesztési életciklus valamely fázisa során. A rendszerben található hibák téves működéshez vezetnek, és így a rendszer összeomlását okozhatják. A fejlesztőknek ezért mind komolyabb erőfeszítéseket kell tenniük ahhoz, hogy a távközlési rendszerek a lehető legstabilabbak legyenek, és a lehető legkevesebb hibát tartalmazzák. Ráadásul az alapos fejlesztési tevékenységeket egyre szűkülő időkorlátok betartásával kell végrehajtani, hiszen az élesedő piaci verseny megköveteli a termék életciklusok és fejlesztésre szánt idő rövidítését.

A legígéretesebb megközelítés e problémák megoldására a formális módszerek használata a szoftverfejlesztés során. A formális módszerek matematikailag jól megalapozott nyelvek és metodológiák, melyek támogatást nyújtanak a szoftver és egyéb rendszerek specifikációja és megvalósítása (implementációja) során. A két legszélesebb körben használt formális módszer – az Estelle [TC988] és az SDL (Specification and Description Language) [IT00] – matematikai alapját a véges automata modellezési technika alkotja.



1. ábra. Software fejlesztési életciklus

A legtöbb általános rendszerhez hasonlóan, a távközlési szoftverek fejlesztése is egy fejlesztési életciklus (software development lifecycle) szerint zajlik. E folyamat legfontosabb fázisai a követelmény-analízis, architektúra tervezés, specifikáció, megvalósítás, tesztelés és karbantartás (maintenance), ahogy azt az 1. ábrán bemutatom. A formális módszerek alapján történő szoftverfejlesztés alapvető gondolata az, hogy

egy precíz specifikáció az életciklus majdnem minden fázisának támogatására alkalmas, így a fejlesztési tevékenységeket hatékonyabbá és gyorsabbá teheti.

- A pontos specifikációk már a rendszerek szabványosítása során alkalmazhatóak, lehetővé téve azok egyértelmű leírását. A specifikációk tehát segítenek elkerülni a különböző rendszerek együttműködési (interoperabilitási) problémáit.
- A formális specifikációk elősegítik a rendszer megértését a fejlesztés korai fázisában, így a tervezési hibák korán felfedezhetővé válnak.
- A formális specifikációk a rendszerfejlesztés mozgatórugói lehetnek. A megvalósítás történhet a specifikáció részleteinek kidolgozásával, sőt, sok esetben a specifikáció alapján közvetlenül kód is előállítható.
- Formális specifikációk a tesztelési fázisban is hasznosak lehetnek. Lehetővé teszik tesztelő mérnökök számára a jó és rossz működés megkülönböztetését, valamint automatikus tesztgeneráló módszerekkel effektív tesztkészletek kialakítása is történhet a specifikáció alapján.

A disszertációm a távközlési szoftver fejlesztés tesztelés és karbantartás fázisaira koncentrál, valamint véges automata (FSM) és kiterjesztett véges automata (EFSM) alapú modellezési technikákkal foglalkozik.

A szoftvertesztelés a fejlesztési folyamat egyik legfontosabb, viszont igen időigényes és költséges fázisa. Annak ellenére, hogy a számítástudomány vívmányai számos területen megkönnyítették a szoftverfejlesztők dolgát és elősegítették a rendszerek pontos megvalósítását, a tesztelés még mindig kihagyhatatlan eleme a folyamatnak. Egyes tanulmányok szerint a fejlesztési életciklus időtartamának és költségeinek nagyságrendileg az 50%-áért felelősek a teszteléssel kapcsolatos tevékenységek. Éppen ezért elengedhetetlenül fontos feladat a tesztelési folyamat hatékonyabbá tétele, alapos és kevésbé időigényes módszerek kidolgozása. A legígéretesebb megközelítés a tesztelési tevékenységek automatizálása. Ezen belül is a tesztelési folyamat első lépése, a tesztkészletek létrehozása a legkritikusabb elem, mivel itt dől el, hogy mennyire hatékony teszteket fogunk a későbbiekben futtatni.

A karbantartási fázis alapvetően a szoftver kiadása utáni tevékenységeket foglalja össze. A fogalom alatt elsősorban a már megvalósított rendszerek utólagos változtatását értjük. A változtatás célja lehet olyan hibák kijavítása, melyek a kiadás után derültek ki plusz tesztelés, vagy a felhasználók visszajelzései alapján. Ezen kívül sok esetben szükség lehet új funkcionalitás bevezetésére is, mely szintén a karbantartás feladatai közé tartozik. Számos rendszer esetében a karbantartás egyetlen lehetséges megoldása a kiadott rendszer visszahívása. Szoftver-rendszerek kapcsán viszont a patch-ek (javító foltok) használata a legelterjedtebb megoldás. A patch-ek alkalmazásának számos előnye van, többek között gyorsabb és olcsóbb megközelítés, mint a visszahívás, ráadásul lehetővé teszi a felhasználói igények folytonos követését.

2. Kutatási célok

A disszertáció fő célkitűzése a szoftver-rendszerek hibáinak detektálásával, diagnosztikájával és javításával kapcsolatos problémák vizsgálata. A vizsgálatok fókuszában az FSM és EFSM formális technikát alkalmazó metodológiák állnak. A célkitűzéseim három fő aspektust fednek le, melyek a hiba fogalma köré vannak csoportosítva. A három aspektus a hibadetektálás, a hiba diagnosztika és a hibajavítás.

Kutatásom első területe a hibadiagnosztika problémájához kapcsolódik. Célkitűzésem volt a probléma megoldhatóságának vizsgálata véges automaták esetében. Bemutattam, hogy nem mindig lehetséges pontosan diagnosztizálni (azonosítani) egy egyedüli hibát egy FSM rendszerben, ezért további célomként egy olyan algoritmus kidolgozását tűztem ki, mely a létező megoldásokkal szemben soha nem ad rossz megoldást. Abban az esetben, ha pontos diagnosztika nem lehetséges, a módszerem a feltételezhető megoldások legszűkebb halmazát adja vissza.

Kutatásom második fő célkitűzése a konformancia tesztelés hatékonyságának növelése volt a tesztkészletek optimalizálásának segítségével. Célom volt egy algoritmus kidolgozása, mely SDL nyelven íródott specifikációk esetén lehetővé teszi a konformancia tesztek méretének csökkentését anélkül, hogy azok hibadetektálási képessége romlana. A megoldásom a mutáció-analízis eredményeire épít, és lehetővé teszi a tesztválogatás minden lépésének automatizálását.

A célkitűzéseim harmadik csoportja a patch-ek kidolgozásának problémájával kapcsolatos. A témakör a létező rendszerek megváltoztatásával foglalkozik, így többek között a későn felmerülő hibák kijavítására alkalmas. Fő célom a patch-ek modellezésének kidolgozása volt FSM leírások esetében, ezen felül definiáltam a patch-ek optimalizálásának elméleti alapjait.

3. Kutatási metodológia

A fenti célkitűzések elérése érdekében kutatási módszerek széles skáláját alkalmaztam. Ezek közé tartoznak matematikai eszközök a gráfelméletből, komplexitáselméletből és számításelméletből; valamint gyakorlati kísérletek és azok analízise. Az általam kidolgozott algoritmusokhoz – amennyiben az lehetséges volt – mindig egyértelmű komplexitás adatokat adtam, valamint azok működését példákon keresztül szemléletesen bemutattam. Heurisztikus algoritmusok esetén mind analitikai, mind gyakorlati vizsgálatokat végeztem a megközelítés értékelésére.

Mivel a legtöbb formális módszer (kiterjesztett) véges automaták segítségével modellezi a rendszerek működését, sok esetben a véges automata elmélet létező megoldásaira építettem kutatásom során. Tézis 1 -ben a létező hibadiagnosztikai algoritmusokat vizsgáltam és fejlesztettem tovább [LS93] [GvB92]. A tézis fő eredményeit

analitikai vizsgálatokkal hoztam létre. A kidolgozott algoritmushoz egzakt komplexitás értéket adtam és példán keresztül demonstráltam azt.

Tézis 2 a tesztválogatás problémakörével foglalkozik, mely alapjaiban nehéz probléma. Az általam javasolt megközelítés a mutáció-analízis [MLS78] megoldásaira épít és annak segítségével ad megoldást a feladatra. Az algoritmusomat mind analitikailag, mind kísérleteken keresztül megvizsgáltam.

Tézis 3-ban az automata elmélet eredményeire építettem. A patch-ek modellezésére vonatkozó megoldásom Chow korábbi hibamodellekkel kapcsolatos munkájára alapozott [Cho78]. Disszertációmban egy új problémát vetettem fel: célul tűztem ki a patch-ek optimalizálását. A problémát a gráfelmélet eszköztárával közelítettem meg. A komplexitáselmélet állításaira építkezve sikerült meghatároznom a probléma nehézségét. Mivel bebizonyítottam, hogy az NP-teljes, megmutattam, hogyan kell a problémát átalakítani úgy, hogy az létező heurisztikákkal megoldható legyen.

4. Új eredmények

Eredményeimet – a disszertációmnak megfelelően – három csoportba rendszereztem. Az itt leírt állítások és algoritmusok részletes kifejtése és bizonyítása a disszertáció megfelelő fejezetében található.

4.1. Véges automata alapú hibadiagnosztika problémájának megoldhatósága

A hibadiagnosztika problémaköre a következő feladattal foglalkozik: adott egy rendszer specifikációja és egy fekete doboz megvalósítás (implementáció). Cél a kettő közötti különbség(ek) lokalizálása. A probléma megoldása számos területen alkalmazható [LY96]. Ezek közül az egyik legfontosabb a megvalósítás hibáinak azonosítása és kijavítása.

Korábbi vizsgálatok jelentős eredményeket értek el a hibadiagnosztikával kapcsolatban. Különböző kutatások más-más feltételezéseket alkalmaztak mind a specifikációval, mind a megvalósítással kapcsolatban. Két hibadiagnosztikai cikk is létezik, mely a következő feltevésekkel él [GvB92] [LS93]:

- A specifikáció egy determinisztikus, teljesen specifikált, erősen összefüggő (strongly connected) minimál véges automata.

- A megvalósítás egyetlen hibát tartalmaz, mely a következő típusok valamelyike:

– Kimenet hiba – adott állapotban egy bemenet hatására a megvalósítás a specifikációtól eltérő kimenetet ad.
– Átmenet hiba – adott állapotban egy bemenet hatására az automata a specifikálttól eltérő állapotba jut.

A lehetséges különbségek behatárolásával mindkét cikk garantálja az egyetlen hiba pontos diagnosztikáját – egyértelmű azonosítását. Disszertációmban a problémát ugyanazon feltételezésekkel élve vizsgáltam, amelyeket e korábbi cikkek is alkalmaztak.

Tézis 1. *Megmutattam, hogy nem mindig lehetséges pontosan diagnosztizálni egyetlen kimenet vagy átmenet hibát egy véges automatában. Meghatároztam az elégséges feltételek olyan halmazát, melyek esetén a hiba pontos azonosítása garantálható. Az analitikus eredmények alapján létrehoztam egy algoritmust a hibadiagnosztika problémájára. Amennyiben lehetséges, a módszer pontosan azonosítja a különbséget a specifikáció és a megvalósítás között; amennyiben a pontos diagnózis elméletileg sem lehetséges, a feltételezhető hibák legszűkebb (tovább nem bontható) halmazát adja meg.*

4.1.1. A pontos hibadiagnosztika korlátai

Altézis 1.1. *[C1] Megmutattam, hogy nem mindig lehetséges pontosan diagnosztizálni egyetlen kimenet vagy átmenet hibát egy véges automatában, még akkor sem, ha a specifikáció egy determinisztikus, teljesen specifikált, erősen összefüggő minimál véges automata.*

Azonosítottam és demonstráltam a következő problémát: Adott egy specifikációs automata *Spec* és két megvalósítás: $Impl_1$, mely egyetlen hibát tartalmaz ($Fault_1$) és $Impl_2$, mely szintén egyetlen hibát tartalmaz ($Fault_2$). $Fault_1$ és $Fault_2$ különböző hibák. $Impl_1$ és $Impl_2$ nyilvánvalóan nem lehetnek a specifikációval ekvivalensek, hiszen a specifikáció egy determinisztikus, teljesen specifikált, erősen összefüggő minimál véges automata. $Impl_1$ és $Impl_2$ viszont egymással ekvivalensek lehetnek annak ellenére, hogy a hibák, melyeket tartalmaznak, különböznek. Ebben az esetben lehetetlen megkülönböztetni a két automatát, tehát lehetetlen egyértelműen diagnosztizálni az adott hibát.

4.1.2. A garantált hibadiagnosztika feltételei

Meghatároztam az elégséges feltételek egy olyan halmazát, melyek esetén a hiba pontos azonosítása garantáltan lehetséges. Másként fogalmazva analizáltam, hogy két vagy több megvalósítás, melyek egyaránt egyetlen hibát tartalmaznak, mikor nem lehetnek ekvivalensek.

Altézis 1.2. *[C1] Bebizonyítottam, hogy – ha a specifikáció egy determinisztikus, teljesen specifikált, erősen összefüggő minimál véges automata – két megvalósítás mely egy-egy különböző hibát tartalmaz, nem lehet ekvivalens, amennyiben a megvalósítások minimál automaták.*

Altézis 1.2 lényegében megmutatja, hogy amennyiben csak egyetlen különbség van egy megvalósítás és egy specifikáció között, és az implementáció minimál automata, akkor az egyedi, tehát nincs másik hiba, mely ugyanazt a változást okozná a működésben. Éppen ezért ebben az esetben garantáltan lehetséges a hiba pontos azonosítása.

Altézis 1.2-t a következő állítások igazolásával bizonyítottam be:

- Két – egy-egy különböző kimeneti hibát tartalmazó – automatát mindig meg lehet különböztetni.
- Két automatát mindig meg lehet különböztetni, ha az egyik kimenet-hibát, a másik átmenet-hibát tartalmaz, és mindkét automata minimál.
- Mindig megkülönböztethető két automata, ha egy-egy különböző átmenet-hibát tartalmaznak és minimál automaták.

Altézis 1.2 a megvalósított véges automata (implementáció) jellemzőivel foglalkozik. Mivel a megvalósított véges automata fekete doboz jellegű, a tézis eredményei közvetlenül nem alkalmazhatóak, viszont jól használhatóak a módosított hibadiagnosztikai algoritmus elemeként.

4.1.3. Egzakt hibadiagnosztikai algoritmus

Korábbi hibadiagnosztikai megoldások nem vették figyelembe az Altézis 1.1-ben bemutatott problémát. Egy véges automatában található egyetlen hiba diagnosztikáját mindig megoldhatónak tekintették. Éppen ezért a korábbi algoritmusok bizonyos esetekben rossz eredményeket szolgáltathatnak: Amennyiben a hiba pontos azonosítása nem lehetséges, akkor is egyetlen megoldást adnak, mely rossz is lehet (az adott hibától különböző, de ekvivalens automatát létrehozó hibát azonosítanak).

Altézis 1.3. *[C1] Egy algoritmust javasoltam a hibadiagnosztika probléma megoldására. Altézis 1.1 és Altézis 1.2 analitikai eredményei alapján az algoritmus csak akkor azonosítja pontosan a különbséget a specifikáció és megvalósítás között, ha az elméletileg lehetséges; amennyiben a pontos diagnosztika nem lehetséges, a feltételezhető hibák minimális halmazát adja meg. Létrehoztam az algoritmus két, kissé különböző változatát, melyekhez pontos komplexitás adatokat szolgáltattam, valamint egy példán keresztül demonstráltam a módszer működését.*

Algoritmusomba integráltam az Altézis 1.2 analitikai eredményeit. Segítségükkel gyorsan ellenőrizhető, hogy a probléma éppen vizsgált lehetséges megoldása az egyetlen-e. Amennyiben bebizonyosodik, hogy igen, a hiba egyértelműen azonosított, így az algoritmus véget ér, egyéb esetben az algoritmus folytatódik, és a lehetséges hibák minimális halmazát hozza létre. Az algoritmusom Lee módszerén alapul [LS93] és a következő alapvető lépésekből áll:

Lépés 1 – a hiba detektálása: Konformancia tesztelést végzünk a specifikációval szemben.

- Amennyiben nem találunk különbséget, a megvalósítás ekvivalens a specifikációval, tehát vagy nincs hiba az implementációban, vagy több, mint egy hiba van benne; az algoritmus vége.
- Amennyiben különbséget találunk, tovább megyünk a második lépésre.

Lépés 2 – A hiba diagnózisa: Az első lépés eredményei alapján meghatározzuk a lehetséges hibák halmazát. Ezekből minden hibára:

1. Az adott hibának megfelelő, *jelölt* automatát hozunk létre.
2. Konformancia tesztelést végzünk a megvalósításon a jelölt automatával szemben.
   - Amennyiben nem konform a két automata, az adott hiba kiesik; folytatás a 2.1-es lépéssel.
   - Amennyiben a két automata konform, azonosítottunk egy lehetséges hibát. Ezek után megvizsgáljuk, hogy biztosan az egyetlen lehetséges hibát találtuk-e meg: a jelölt automatát minimalizáljuk és összehasonlítjuk a specifikációval.
     - Amennyiben az állapotaik száma megegyezik, a talált hiba egyértelműen az egyetlen lehetséges – a hibát pontosan detektáltuk; az algoritmus vége.
     - Egyéb esetben az összes hátralévő lehetséges hibát ellenőriznünk kell; vissza a 2.1-es lépéshez.

Az algoritmus komplexitása – legrosszabb esetet figyelembe véve – $O(pn^5)$, ahol p a bemenetek száma $p = |I|$ és n az állapotok száma $n = |S|$ a specifikációban. Az algoritmus egy gyors verzióját is bemutattam, melynek komplexitása $O(pn^3 \log n)$. Az algoritmust disszertációmban demonstráltam egy részletes példa segítségével.

4.2. Automatikus hibaalapú tesztválogatás SDL rendszerek esetére

A legtöbb távközlési rendszer esetén az automatikus tesztgeneráló algoritmusok – sőt, több különböző szakértő együttes munkája – nagy számú tesztesetet hozhat létre. Gyakran tesztsorozatok ezreit tartalmazhatja egy tesztkészlet. Ilyenkor nem praktikus a tesztek mindegyikét lefuttatni, hiszen minden egyes futtatás jelentős időt vesz igénybe. Lehetséges megoldás a tesztkészlet csökkentése tesztválogatás segítségével.

A tesztválogatás problémát a következőképp lehet megfogalmazni [LH01]: Adott a tesztszekvenciák (nagy elemszámú) halmaza. Cél ennek minimális részhalmazát kiválasztani anélkül, hogy a hibadetektálási képességet rontanánk. Fő célom az volt, hogy a tesztválogatás minden lépését automatizáljam, és ez által egy praktikus megoldást hozzak létre konformancia tesztkészletek optimalizálására.

Tézis 2. *A mutáció-analízis alapjaira építve javasoltam egy konformancia teszt válogató algoritmust. Chow széles körben elfogadott FSM hibamodellje alapján egy mutáció operátor halmazt hoztam létre, mely segítségével válogatási kritériumként használt mutáns rendszereket generáltam. A megoldásomat alapos empirikus és analitikus vizsgálat segítségével igazoltam.*

4.2.1. SDL rendszerekhez javasolt mutációs operátorok

Korábbiakban számos kutatást végeztek már mutációs operátorokkal kapcsolatban, melyek során különböző formalizmusok esetére hoztak létre operátorokat. SDL rendszerekhez viszont eddig még nem készült operátor készlet. A leginkább kapcsolódó korábbi tanulmány Estelle specifikációkhoz definiált operátorokat [SFSM00]. Annak ellenére, hogy mind az Estelle, mind az SDL az EFSM modellen alapszik, úgy döntöttem, hogy Chow tradicionális FSM hibamodelljére [Cho78] alapozva hozom létre az operátor halmazomat. A döntés elsődleges oka az volt, hogy a létező operátorok – beleértve azokat, melyeket [SFSM00]-ben definiáltak – három fontos követelménynek nem feleltek meg:

Atomi operátorok: Mivel korábbi, hibamodellekkel kapcsolatos kutatások atomi operátorok használatát javasolták, én is a lehető legkisebb változtatásokat hozom létre a rendszerben. Ez a megközelítés összhangban van a mutáció analízis technika alapelveivel is [MLS78].

Automatikusan alkalmazható operátorok: Míg korábbi tanulmányok manuálisan alkalmazható operátorokat javasoltak, munkám egyik fő célkitűzése egy automatikus algoritmus és eszköz létrehozása volt. Éppen ezért elsődleges

szempont volt számomra olyan operátorok létrehozása, melyek automatikusan használhatóak bármely specifikáció esetén.

Operátorok SDL specifikus mechanizmusokhoz: SDL specifikus mechanizmusok – timerek, paraméterezett jelek, "save-mechanizmus", implicit jelfeldolgozás (mely gyakorlatilag az SDL "completeness assumption"-ja[1]) – speciális megoldások kidolgozását igényelték.

Az operátorok részletes definíciója disszertációm 5.1.1 fejezetében található.

Offut és társai eredményei alapján [OLR+96] olyan mutáns létrehozási stratégiát javasoltam, mely reprezentatív mutáns készletet, nem pedig kimerítő (teljes) mutáns készletet hoz létre. Ez azt jelenti, hogy az operátorokat szisztematikusan alkalmazom az SDL rendszer minden részén, de a specifikáció minden egyes elemére csak néhány mutánst hozok létre, nem pedig az összes lehetségest. Sőt, az operátorok maguk is e stratégia alapján lettek kidolgozva, mivel a definíciójuk önmagában lekorlátozza a segítségükkel létrehozható mutánsok számát.

Altézis 2.1. *[J0, J4, C4, C6] Egy tradicionális FSM hibamodellre építkezve mutáns operátor készletet hoztam létre SDL specifikációk esetére, ezen felül definiáltam az operátorok alkalmazási stratégiáját. Az operátorok automatikusan alkalmazhatók bármely SDL specifikációra és atomi változtatásokat okoznak a rendszerben. Az operátor készlet tartalmaz SDL specifikus megoldásokat. Mind az operátor készlet, mind a mutáns létrehozási stratégia reprezentatív mutáns készlet létrehozását célozza.*

4.2.2. Tesztválogató algoritmus

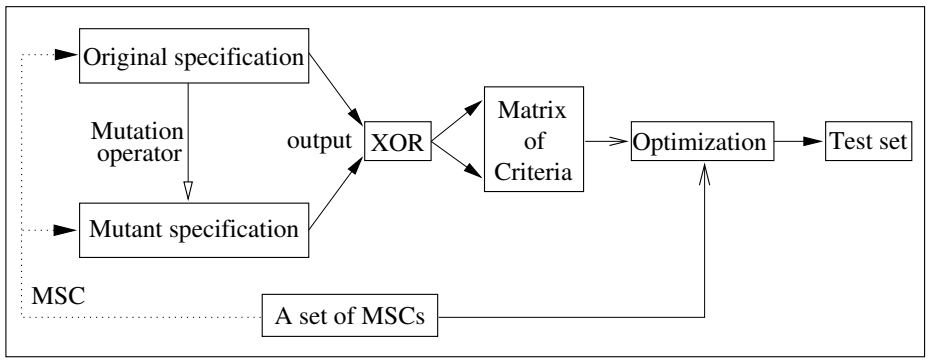
Altézis 2.2. *[J0, J4, C4, C6] Új algoritmust javasoltam konformancia tesztek válogatására. Az algoritmus inputjai a tesztelendő rendszer SDL specifikációja, valamint egy kiinduló teszt-szekvencia halmaz. Ezek alapján az algoritmus kiválasztja az eredeti teszthalmaz részhalmazát, melynek az eredetivel megegyezik a mutáns lefedettsége. Az algoritmus az Altézis 2.1-ben bemutatott operátorokat alkalmazza a válogatási kritériumként szolgáló mutánsok létrehozására. A megoldásomat alapos empirikus és analitikus vizsgálat segítségével igazoltam.*

A 2. ábra bemutatja algoritmusomat.

Az algoritmus lényege a következőképpen foglalható össze:

1. Mutáns specifikáció létrehozása.

[1]Ha egy olyan üzenet érkezik, melyre az adott állapotban nincs explicit átmenet definiálva, implicit jelfeldolgozás történik, tehát az üzenet kikerül a várakozási sorból és a rendszer az adott állapotban marad.



2. ábra. Mutáció analízis alapú tesztválogató algoritmus

2. A kiinduló tesztkészlet futtatása a mutáns specifikáción és inkonzisztencia keresése.

3. Azon tesztesetek megjelölése, melyek az adott mutánst felfedezik.

4. Az első három lépés megismétlése az összes mutáns esetére, melyeket a mutáns létrehozási stratégiám alapján az adott rendszerhez létrehozunk.

5. Kritérium-mátrix létrehozása, ahol a sorok reprezentálják a mutánsokat, az oszlopok az eredeti tesztkészlet szekvenciáit. A mátrix minden tesztesetre megjelöli azokat a mutánsokat, melyeket az adott teszteset megtalál.

6. A kritérium-mátrix alapján egy optimális tesztkészlet kiválasztása valamely optimalizáló módszer segítségével.

Az algoritmus utolsó lépése egy optimalizálási probléma: a kritérium-mátrix alapján meg kell találni a minimális számú tesztet a lehető legnagyobb mutáns lefedettséggel. Kiemelném, hogy egy mutánst akkor tekintünk lefedettnek, ha legalább egy olyan teszteset benne van a végső készletben, amelyik megtalálja azt. A problémára alkalmazható heurisztikus és egzakt optimalizáló algoritmusok részletes leírását ld. [Csö01].

Algoritmusom a következő előnyökkel rendelkezik a létező megoldásokkal szemben:[2]

Tesztválogató eljárások: Számos, a tesztválogatásról szóló cikk létezik az irodalomban, de ezek közül semelyik sem építkezik a mutáció analízis eszköztárára. A

[2]A bemutatott algoritmus bármilyen (pl. véletlenszerű) állapottér bejáró megoldással kiegészítve felfogható egy, két lépésből álló, tesztgeneráló algoritmusként. Éppen ezért – annak ellenére, hogy a kutatásom alapvetően a tesztválogatás problémájára koncentrál – az algoritmusomat érdemes összevetni létező EFSM tesztgeneráló algoritmusokkal is.

leginkább kapcsolódó kutatást Lee és társai végezték [LH01]-ban. A szerzők módszerükben a tesztkészleteket az elérési gráf (reachability graph) éleinek lefedettsége alapján optimalizálják. A cikkben prezentált eredmények tiszta és egyértelmű elméleti háttérrel rendelkeznek, de a javasolt megközelítés mégsem praktikus a legtöbb rendszer esetében, mivel még a legkisebb EFSM-ek esetén is elfogadhatatlanul nagy az elérési gráf mérete. Én ezzel szemben olyan algoritmust dolgoztam ki, mely realisztikus méretű rendszerek esetén is alkalmazható.

Hibalefedettség alapú tesztgeneráló algoritmusok: Algoritmusom közvetlenül összehasonlítható más mutáció tesztelésen alapuló megközelítésekkel. A korábbi kutatásokhoz – többek között [SFSM00]-höz – képest megoldásom számos újdonságot és fejlesztést tartalmaz. Ezek közül a legfontosabbak:

- Mutációs operátorok – algoritmusomban egy speciális mutációs operátor készletet használok, melyet Chow hibamodellje [Cho78] alapján dolgoztam ki az SDL nyelvhez. Az operátorok és kidolgozásuk hátterének részletes bemutatását ld. Altézis 2.1-ben.
- Tesztválogató algoritmus – munkám során egy olyan algoritmust javasoltam létező tesztkészletek optimalizálására, mely a mutáns detektáló képesség csökkentése nélkül válogatja ki az eredeti teszthalmaz egy részhalmazát. Korábbi tanulmányok ezzel szemben a mutációs operátorokat egyszerűen a manuális tesztgenerálási (tesztkészítési) eljárás támogatására használták.
- Eszköz és kísérletek – korábbi kutatások tesztelési stratégiája manuális alapokra építkezett, nem dolgoztak ki automatikus módszert és eszközt a megközelítés támogatására. Az én algoritmusom ellenben a kezdetektől fogva automatikus működésre lett tervezve és egy, a módszert megvalósító szoftver-eszköz is kifejlesztésre került.

Specifikáció-lefedettség alapú tesztgeneráló módszerek: A legtöbb EFSM tesztgeneráló megközelítés magát a specifikációt használja lefedettségi kritériumként. Ezek az eljárások azt az elvárást támasztják, hogy a specifikációs EFSM minden átmenetét legalább egyszer végrehajtsa a tesztkészlet. Mivel a módszeremben az operátorokat szisztematikusan alkalmazom a specifikáció minden részében, az optimalizált tesztkészlet specifikáció-lefedettsége jóformán minden gyakorlati protokoll esetén egyenlő lesz a kiinduló készletével. Másképpen fogalmazva az algoritmusom nem csökkenti az egy tesztkészlet által biztosított lefedettséget egy specifikáció alapú kritériumot figyelembe véve, viszont csökkenti a tesztkészlet méretét (amennyiben lehetséges).

Validáció Számos kísérletet hajtottam végre ismert protokollokon, mint például INRES-en [Hog91], a konferencia protokollon [BFV+99], vagy a WAP-WTP-n (Wireless Application Protocol – Wireless Transaction Protocol) a bemutatott algoritmus gyakorlati tanulmányozása céljából. A kísérletek két különböző sémát követtek: a kezdeti tesztkészleteket vagy automatikusan generáltam egy véletlenszerű állapottér bejáró algoritmus segítségével, vagy azok több szakértő manuálisan munkájával készültek el. Minden kísérlet a kezdeti tesztkészlet jelentős méretcsökkenését mutatta (74% és 93% közötti csökkenés), és nem rontották a lefedettséget a legelterjedtebb metrikát figyelembe véve – a strukturális lefedettség szempontjából. Szakértők manuális elemzés alá vették mind a kezdeti, mind az optimalizált tesztkészleteket, és a megközelítést megfelelőnek értékelték. Továbbá az algoritmus futási teljesítménye elfogadhatónak bizonyult, figyelembe véve Java nyelv sebességproblémáit és a használt hardvert.

4.3 Hibák korrekciója – a patch-elés elmélete

A patch-ek – vagy update-ek – használata a legszélesebb körben elterjedt megoldás létező szoftver-rendszerek működésének módosítására. Ezek – többek között – használhatóak a rendszer kiadása után talált hibák orvoslására. A patch-ek alkalmazásának legfontosabb előnye, hogy csak a különbséget – a rendszeren végrehajtandó változtatásokat – kell kiadni és elosztani a felhasználók között a teljes rendszer lecserélése helyett.

A patch-ek gyakorlati jelentősége és széles körű használata ellenére nem készültek korábban tanulmányok a patch-ek modellezése és analízise kapcsán. Kutatásom fő célja éppen ezért a patch-ek modellezésének kidolgozása volt FSM leírások esetére, valamint a patch-ek optimalizálásának elméleti alapjait szándékoztam lefektetni. Munkám során determinisztikus automatákra koncentráltam, és két FSM-et akkor tekintettem megegyezőnek, ha azok izomorfak – egy természetes feltételezés, hiszen a Mealy automaták alapvetően élcímkézett (edge-labeled) gráfokkal írhatók le, melyekben az állapotok címkézése irreleváns. Okfejtéseim során a következő véges automatákat használtam fel az állítások bemutatására: $M = (I, O, S, \delta, \lambda)$, $M' = (I, O, S', \delta', \lambda')$, és így tovább, ahol I, O és S a bemeneti jelek, a kimenti jelek valamint az állapotok véges halmazai; δ: $S \times I \to S$ az átmenti és λ: $S \times I \to O$ a kimeneti függvény.

A továbbiakban emellett egy általános teljességi hipotézist (completeness assumption) vettem figyelembe annak érdekében, hogy a nem teljesen specifikált automaták kezelhetővé váljanak. [3]

[3] A teljességi hipotézis szerint egy egyedi kimeneti jelet Υ_o adunk a kimenetek halmazához O, és egyedi állapotot Υ_s az állapotok halmazához S, melyek a null (hiba) jelet és állapotot jelölik.

Tézis 3. *Újszerű megközelítést mutattam be a patch-ek modellezésére. Bevezettem a patch-ek optimalizálásának problémáját, melynek alapos komplexitás-analízisét is végrehajtottam. Mivel az eredményeim azt mutatták, hogy polinomiális idejű megoldás létezése valószínűtlen, ezért meghatároztam, hogyan lehet az optimális patch problémát olyan állapottér keresési feladattá alakítani, melyre létező heurisztikus algoritmusokkal jó közelítő megoldást lehet adni.*

4.3.1. Patch-ek modellezése

A patch-ek modellezésére általam javasolt megközelítés létező hibamodelleken alapszik. Teljesen specifikált és determinisztikus véges automatákhoz a legszélesebb körben elfogadott hibamodellt Chow dolgozta ki [Cho78].

Altézis 3.1. *[C0] Megmutattam, hogy Chow hibatípusai, bizonyos megszorításokkal, szerkesztési operátorként (edit operator) foghatóak fel. Ezen felül rámutattam, hogy a szerkesztési operátorok szekvenciái alkalmasak patch-ek modellezésére FSM rendszerek esetén.*

A következő szerkesztési operátor típusokat definiáltam Chow hibamodellje alapján: Az átmenet-operátor $TRO : \delta(s,i) = s_1 \rightarrow s_2$,[4] ahol $s_1, s_2 \in S, S' = S$ (S-be beleértendő Υ_s is). A kimeneti operátor $OO : \lambda(s,i) = a \rightarrow b$, ahol $a, b \in O$ (O-ba beleértendő Υ_o is). A plusz állapot operátor $ESO : a,\ S' := \{S \cup a\}$; és a mínusz állapot operátor $MSO : a,\ S' := \{S \setminus a\}$.

A következő két megszorítást határoztam meg a szerkesztési operátorok kapcsán:

1. Egy plusz állapot operátor alkalmazása magába foglalja egy új állapot (s) létrehozását, melynek null átmeneti $\delta(s,i) = \Upsilon_s$ és null kimeneti $\lambda(s,i) = \Upsilon_o$ funkciói vannak minden $i \in I$-re; tehát ez egy üres állapot (blank state) hozzáadását jelenti.

2. Egy mínusz állapot operátor csak akkor használható s állapotra, ha s-nek nincsenek bemenő állapot-átmenetei, valamint minden átmeneti és kimeneti funkciója null ($\delta(s,i) = \Upsilon_s$ és $\lambda(s,i) = \Upsilon_o$); tehát csak olyan üres állapotok törölhetők, melyeknek nincs bejövő átmenete.

Szerkesztési operátorok szekvenciáját is definiáltam: vegyünk egy e szerkesztési operátort, melyet M_1 véges automatán alkalmazva M_2-t kapjuk; ezt $M_1 \Rightarrow M_2$ via e–vel jelöljük. Legyen E a szerkesztési operátorok szekvenciája $e_1, e_2, ..., e_k$. E FSM

[4] A jelölés átláthatósága végett $\delta(s,i) = s_1 \rightarrow s_2$ -et használom egy átmenet-operátor bemutatására $[\delta(s,i) = s_1] \rightarrow [\delta'(s',i) = s_2']$ helyett, ahol M az eredeti és M' a szerkesztett automata. Hasonló jelölést használok a kimeneti operátorok esetén is.

M-et FSM M'-vé változtatja akkor, ha létezik az automaták olyan szekvenciája $M_0, M_1, ..., M_k$, hogy $M = M_0$, $M' = M_k$, és $M_{i-1} \Rightarrow M_i$ via e_i minden $1 < i < k$-re.

Nyilvánvalóan az adott bemeneti és kimeneti jelhalmazzal (I és O) rendelkező determinisztikus véges automaták halmaza zárt az előbbiekben meghatározott szerkesztési operátorok alatt, továbbá bármely két determinisztikus FSM esetén mindig létezik egy szerkesztési operátor szekvencia, mely az egyik automatát a másikkal izomorffá alakítja. Rámutattam, hogy mindezek miatt a szerkesztési operátorok szekvenciái patch-ek modelljeként használhatók, hiszen egyértelműen definiálják az FSM rendszeren végrehajtott változtatásokat.

4.3.2. Az optimális patch probléma

Egyértelmű, hogy adott két véges automata között végtelen sok lehetséges szerkesztési operátor szekvencia (patch) létezik. Célul tűztem ki, hogy ezek közül a legjobbat azonosítsam.

Altézis 3.2. *[C0] Továbbfejlesztettem a patch-ek modellezésének módszerét azzal, hogy költségeket rendeltem a szerkesztési operátorokhoz. Definiáltam és bemutattam egy új problémát, melyet az optimális patch – vagy optimális update – problémának nevezek. Egy adott változtatási követelményt figyelembe véve a probléma az optimális patch – minimális költségű szerkesztési operátor szekvencia – meghatározása úgy, hogy az a rendszert az elvárásoknak megfelelően módosítsa.*

A korábban bemutatott patch modellezési megközelítésemet ρ költségfüggvény bevezetésével finomítottam, mely egy nem negatív valós számot $\rho(e)$ rendel minden szerkesztési operátor típushoz. ρ-t távolság-mértékként definiáltam, tehát az teljesíti a pozitivitás, a szimmetria és a háromszög-egyenlőtlenség kritériumát. A ρ költségfüggvényt következőképpen terjesztettem ki szerkesztési operátorok szekvenciáira ($E = e_1, e_2, ..., e_k$-re):

$\rho(E) = \sum_{i=1}^{k} \rho(e_i)$.

Véges automaták fölött definiáltam az optimális patch problémát: FSM M modellezzen egy már megvalósított és kiadott – a felhasználókhoz eljuttatott – rendszert. Tegyük fel, hogy valamilyen okból szükségessé válik a rendszer (működésének) módosítása, ezért egy új – a követelményeknek megfelelő – designt dolgoznak ki, melyet M' modellez. A probléma a rendszert megváltoztató optimális patch meghatározása, vagyis azon szerkesztési operátorok azonosítása melyek a legkevesebb költséggel változtatják M rendszert M'-vé (M'-vel izomorffá) az adott költségfüggvényt figyelembe véve.

Mivel ρ-t távolság-metrikaként definiáltam, a problémát megfogalmazhatom úgy is, hogy annak célja a két automata – M és M' – (szerkesztési) távolságának meghatározása. Ekkor a két automata közti (szerkesztési) távolságot úgy definiálom,

hogy az a minimális költségű olyan szerkesztési operátor szekvencia, mely M rendszert M'-vé alakítja:

$$dist(M, M') = min\{\rho(E) \mid E\ az\ M\ rendszert\ M' - ve\ alakito\ szerkesztesi\ operator\ szekvencia\}$$

Disszertációmban hangsúlyoztam, hogy az optimális patch probléma megoldása (automaták távolságának meghatározása) felfogható az FSM rendszerek hasonlóságának mértékeként. Éppen ezért a patch-ek optimalizálásán kívül olyan gyakorlati hasznosítása is elképzelhető, mint a fejlesztők munkájának felbecslése és értékelése, valamint plagizálás detektálása.

4.3.3. Transzformációk és szerkesztési operátorok

A szerkesztési operátorok szekvenciáinak közvetlen kezelése nem megfelelő megközelítés, mivel két adott automata között végtelen sok lehetséges szekvencia létezik. Ezért bevezettem egy alternatív megoldást a változtatások leírására.

Altézis 3.3. *[C0] Bevezettem a transzformációk fogalmát, és felhasználtam őket az FSM rendszerek változtatásainak alternatív leírására. Létrehoztam egy algoritmust, mely minden transzformációhoz egy szerkesztési operátor szekvenciát rendel. Bebizonyítottam, hogy az algoritmus nem egy véletlen választás, hanem egy adott transzformációhoz a lehető legalacsonyabb költségű szerkesztési operátor szekvenciát rendeli. Ezzel bebizonyítottam, hogy az optimális patch probléma megoldható a minimális költségű transzformáció meghatározásával.*

Két FSM (M és M') esetén egy T transzformáció nem más, mint az állapotok párjainak halmaza (s, s'), ahol $s \in S_{sub} \subseteq S,\ s' \in S'_{sub} \subseteq S'$ valamint T-ben bármely (s_1, s'_1) és (s_2, s'_2) állapotpárokra: $s_1 = s_2$ akkor és csak akkor ha $s'_1 = s'_2$ (egy-az-egyes állapot összerendelés). Informálisan fogalmazva a transzformációk megadják, hogyan alakítsunk egy determinisztikus véges automatát egy másikká.

Létrehoztam egy algoritmust, mely meghatározza az FSM M-et M'-vé alakító T transzformációhoz tartozó szerkesztési operátor szekvenciát. Az algoritmus lényege a következőképp foglalható össze:

- Plusz állapot operátorok alkalmazásával M' minden olyan állapotára, mely nincs benne T-ben[5], létrehozunk egy üres állapotot M-ben.

[5]Az alatt, hogy egy állapot benne van (nincs benne) T-ben azt értjük, hogy az adott állapot (nem) szerepel T egyetlen állapotpárjában (sem).

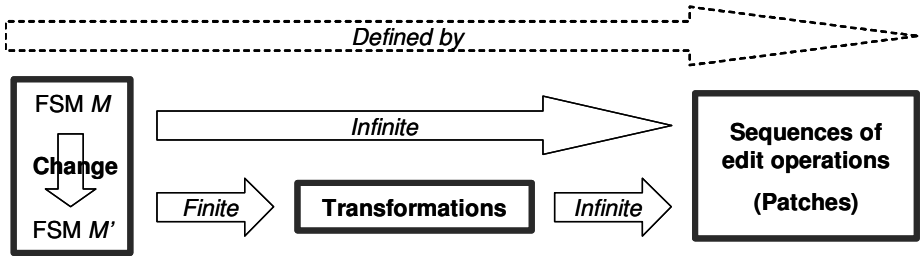
- Ezzel egy egyértelmű összerendelést (bijection) hoztunk létre S', valamint M-nek a T-ben lévő és új (üres) állapotai között.

- Kimeneti és átmenet operátorok alkalmazásával M-et az összerendelésnek megfelelően megváltoztatjuk.

- Mínusz állapot operátorok alkalmazásával eltávolítjuk M-nek azon állapotait, melyek nincsenek T-ben (ezek most már üres állapotok).

Az algoritmus polinom futásidejű $O(pn)$ komplexitással, ahol $n = |S|$ és $p = |I|$.

A transzformációkhoz költségeket is rendeltem: $\rho(T) = \sum_{i=1}^{k} \rho(e_i)$ ahol e_i-k a T transzformációhoz tartozó szerkesztési operátorok (a fent definiált algoritmus szerint).



3. ábra. A transzformációk és patch-ek számosságának összefüggése egy adott változtatás esetén

Láthatóan végtelen számú olyan szerkesztési operátor szekvencia létezik, mely kielégíti egy adott T transzformáció feltételeit – különböző költségekkel. A 3. ábra bemutatja a transzformációk és patch-ek számosságának összefüggését egy adott változtatás esetén.

Bebizonyítottam, hogy az a szerkesztési operátor szekvencia, amit a fenti algoritmussal T-hez rendeltem, nem egy véletlen választás, hanem a T-t kielégítő legjobb (legkisebb költségű) szerkesztési operátor szekvencia. Ezt a következő állítások bebizonyításával igazoltam:

- $\rho(T_1 \circ T_2) \leq \rho(T_1) + \rho(T_2)$, ahol $T_1 \circ T_2$ a transzformációk kompozíciója: $T_1 \circ T_2 = (s, s'')|\exists s' \in S'$ úgy, hogy $(s, s') \in T_1 \wedge (s', s'') \in T_2$.

- Bármely olyan szerkesztési operátor szekvenciára $E = e_1, e_2, ..., e_k$, mely FSM M-et M''-vé változtatja, létezik egy T transzformáció M és M'' között úgy, hogy $\rho(T) \leq \rho(E)$. Az állítás másik irányában pedig bármely T transzformációhoz létezik egy E szerkesztési operátor szekvencia, melyre $\rho(E) = \rho(T)$.

Az előző állítások bizonyításával tisztáztam a transzformációk és a szerkesztési távolság (optimális patch) kapcsolatát. Bebizonyítottam, hogy a legkisebb költségű transzformáció költsége egyenlő *dist*-el, tehát a minimális költségű transzformáció egy optimális patch-et határoz meg.

$$dist(M, M') = min\{\rho(T) \mid T\ transzformacio\ M\ es\ M'\ kozott\}.$$

A transzformációk éppen ezért lehetőséget adnak az optimális patch probléma konstruktív megközelítésére.

4.3.4 Komplexitás analízis és heurisztikák

Altézis 3.4. *[C0] Analizáltam az optimális patch probléma komplexitását és bebizonyítottam, hogy az NP-teljes. Meghatároztam, hogyan lehet az optimális patch problémát olyan állapottér keresési feladattá alakítani, melyre létező heurisztikus algoritmusokkal jó közelítő megoldást lehet adni.*

Bizonyításomban a 3 dimenziós házasítás (3-DH, angolul Tripartite Matching) problémát visszavezettem az optimális patch problémára, azaz $dist(M, M')$ meghatározására. Mivel a Tripartite Matching feladat azon problémák közé tartozik, melyekről legelőször be lett bizonyítva, hogy NP-teljesek [Kar72], a redukcióval bebizonyítottam, hogy az optimális patch probléma is NP-teljes. Ez egyértelművé teszi a heurisztikus módszerek szükségességét az eredmény hatékony becslésére.

Bemutattam, hogyan lehet az optimális patch problémát állapottér keresési feladattá alakítani, melyre létező heurisztikus algoritmusokkal jó közelítő megoldást lehet adni. Munkámban a probléma megfogalmazására összpontosítottam, nem pedig magukra a heurisztikákra. A különböző közelítő algoritmusok részletes tárgyalása, azok összehasonlító teljesítmény analízise, paraméter beállításai, etc., nem képezik disszertációm részét, e problémák elemzésének egy teljes disszertációt szentelhetnénk.

Megközelítésem lényege a következő:

Vegyünk két FSM-et, M és M'-t, és hozzunk létre egy állapotteret melyben minden M és M' közötti T transzformációhoz rendelünk egy állapotot. Az állapottérben σ_T reprezentálja a T transzformációhoz kapcsolódó állapotot, σ_T költségét pedig úgy definiáljuk, hogy az egyenlő a kapcsolódó T transzformáció költségével σ_T: $\rho(\sigma_T) = \rho(T)$.

Az állapottér két állapota σ_{T_1} és σ_{T_2} – melyek T_1 és T_2 transzformációhoz kapcsolódnak – akkor szomszédok, ha T_2-t elő lehet állítani T_1-ből úgy, hogy a T_1 által definiált állapot összerendeléshez egy állapotpárt hozzáadunk, vagy abból egyet elveszünk. Az állapottérben a szomszédos állapotok egy lépésben elérhetőek egymásból.

Az állapotok közötti lépések szekvenciáit lépéssorozatnak (walk) nevezzük. Az állapottér egyértelműen összefüggő, tehát bármely állapotpárhoz (σ_{T_i}, σ_{T_j}) létezik egy lépéssorozat σ_{T_i}-ből σ_{T_j}-be.

Ezzel a megközelítéssel a $dist(M, M')$ meghatározásának problémáját átalakítottam az állapottér legalacsonyabb költségű állapotának megkeresésévé. Példaként kifejtettem, hogy a *simulated annealing* (SA) módszer [KGV83] [Cer85], mely egy széles körben alkalmazott heurisztika, hogyan alkalmazható az eredmény közelítő meghatározására.

5. Az eredmények hasznosíthatósága

A disszertációmban bemutatott eredmények és algoritmusok elsősorban a formális metódusokat felhasználó szoftver fejlesztési életciklus tesztelési és karbantartási fázisában alkalmazhatók.

A hibadiagnosztika elméletének tisztázása mellett Tézis 1 megoldást ad egy hiba lokalizációjára fekete doboz FSM rendszerekben, rossz eredmények szolgáltatása nélkül. Az algoritmus közvetlenül alkalmazható véges automatákkal modellezett rendszerek esetén, vagy EFSM alapú formális leírások kontroll elemeire (control parts) hibadiagnózis céljából. A hatékony működés érdekében a módszert elég sűrűn érdemes alkalmazni egy megvalósítás monitorozására, hiszen ekkor kisebb a többszörös hibák előfordulásának valószínűsége.

Tézis 2 konformancia tesztkészletek optimalizálására ad hatékony módszert. A disszertációban bemutatott algoritmust bármely létező tesztkészlet redukálására alkalmazni lehet, ha a kapcsolódó rendszer SDL leírása létezik. Mivel a távközlési szoftverek fejlesztése esetén a formális módszerek alkalmazása egyre elterjedtebbé válik, az SDL leírás megkövetelése a jövőben várhatóan egyre kisebb problémát jelent majd.

Tézis 3 patch-ekkel foglalkozik, melyeket a gyakorlatban széles körben alkalmaznak már kiadott szoftverek működésének megváltoztatására. Tézis 3 eredményei megint csak közvetlenül alkalmazhatóak FSM-ekkel modellezett rendszerek esetén, vagy EFSM alapú formális leírások kontroll elemeire (control parts) a következő területeken:

- Patch-ek modellezésére, definiálására és megértésük elősegítésére.
- Patch-ek optimalizálására.
- Különböző rendszerek hasonlóságának mérésére, ami lehetőséget ad pl. fejlesztők tevékenységének értékelésére, vagy plágium detektálására.

Köszönetnyilvánítás

Az elmúlt néhány évben sokan hozzájárultak a disszertációmban bemutatott ötletek és módszerek kidolgozásához. Elsősorban köszönetet szeretnék mondani a témavezetőimnek: Csopaki Gyulának, Dibuz Saroltának és Tarnay Katalinnak. Kitartó segítségük és iránymutatásuk nélkül ez a disszertáció nem jöhetett volna létre. Szeretném megköszönni a Budapesti Műszaki és Gazdaságtudományi Egyetem Távközlési és Médiainformatikai Tanszékének a kutatáshoz szükséges háttér biztosítását, melyben lehetőségem nyílt független ötletek és gondolatok alapos elemzésére. Hálával tartozom a Távközlési Protokoll és Tesztelés témakörön dolgozó minden munkatársamnak a kellemes és ösztönző munkakörnyezet megteremtéséért. Végül, de nem utolsósorban szeretném megköszönni kollégáimnak és szerzőtársaimnak építő javaslataikat, kritikájukat és támogatásukat, melyet a közös munkán során adtak.

Hivatkozások

[BFV+99] A. Belinfante, J. Feenstra, R.G. de Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, and L. Heerink. Formal test automation: A simple experiment. In G. Csopaki, S. Dibuz, and K. Tarnay, editors, *Proceedings of the* 12^{th} *International Workshop on Testing of Communicating Systems*, pages 179–196. Kluwer Academic Publishers, 1999.

[Cer85] V. Cerny. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, pages 41–51, 1985.

[Cho78] T. Chow. Testing software design modelled by finite-state machines. *IEEE Transactions on Software Engineering*, 4(3):178–187, May 1978.

[Csö01] T. Csöndes. *Conformance Test Suite Optimization*. PhD thesis, Budapest University of Technology and Economics, 2001.

[GvB92] A. Ghedamsi and G. v. Bochmann. Test result analysis and diagnostics for finite state machines. In *Proceedings of the 12th International Conference on Distributed Systems*, 1992.

[Hog91] D. Hogrefe. Osi formal specification case study: The inres protocol and service (revised). Technical Report IAM-91-012, Universitat Bern, Institut fur Informatik, 1991.

[IT00] ITU-T. Recommendation z.100: Specification and description language, 2000.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[LH01] D. Lee and R. Hao. Test sequence selection. In *Formal Techniques for Networked and Distributed Systems, FORTE 2001, IFIP TC6/WG6.1 -* 21^{st} *International Conference on Formal Techniques for Networked and Distributed Systems, Cheju Island, Korea*, pages 269–284, 2001.

[LS93] D. Lee and K. Sabnani. Reverse-engineering of communication protocols. In *Proceedings of the IEEE International Conference on Network Protocols, California*, pages 208–216, 1993.

[LY96] D. Lee and M. Yiannakakis. Principles and methods of testing finite state machines – a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[MLS78] R. A. De Millo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11(4):34–41, April 1978.

[OLR+96] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf. An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(2):99–118, 1996.

[SFSM00] S. R. S. Souza, S. C. P. F. Fabbri, W. L. Souza, and J. C. Maldonado. Mutation testing applied to estelle specifications. In *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8*, Washington, DC, USA, 2000. IEEE Computer Society.

[TC988] ISO TC97/SC21. Estelle – a formal description technique based on an extended state transition model. international standard 9074, 1988.

Publikációk

Folyóirat Cikkek

[J0] Gábor Kovács, **Zoltán Pap**, Gyula Csopaki and Katalin Tarnay. Iterative automatic test generation method for telecommunication protocols. *Computer Standards & Interfaces*, Accepted paper (Corrected proof in Press), 2005.

[J1] Gábor Vincze, **Zoltán Pap** and Róbert Horváth. Peer-to-Peer Based Distributed File Systems, *International Journal of Internet Protocol Technology (IJIPT)*, Accepted paper, 2005.

[J2] Gábor Vincze, **Zoltán Pap**, Róbert Horváth. Peer-to-peer alapú elosztott fájlrendszerek. *Hiradástechnika*, Volume LX, Pages 21-26, 2005. (in Hungarian).

[J3] Gusztáv Adamis, Róbert Horváth, **Zoltán Pap** and Katalin Tarnay. Standardized languages for telecommunication systems. *Computer Standards & Interfaces*, Volume 27, Number 3, Pages 191-205, March 2005.

[J4] Gábor Kovács, **Zoltán Pap**, Gyula Csopaki. Automatic Test Selection Based on CEFSM Specifications. *Acta Cybernetica*, Volume 15, Number 4, pages 583-599, 2002.

[J5] **Zoltán Pap**, Zoltán Rétháti, Róbert Horváth, Gusztáv Adamis. Standardized Event Pair Based Test Generation Method Using TSS&TP. *Acta Cybernetica*, Volume 15, Number 4, Pages 653-667, 2002.

[J6] Róbert Horváth, **Zoltán Pap**, Zoltán Rétháti. Development of Telecommunications Software Using Formal Languages. *Magyar távközlés selected papers 2000*, Pages 48-50, 2000.

[J7] Róbert Horváth, **Zoltán Pap**, Zoltán Rétháti. Távközlési szoftver fejlesztés formális nyelvek felhasználásával. *Magyar távközlés*, Volume X, Number 7, Pages 3-6, July 1999. (in Hungarian).

[J8] **Zoltán Pap**. IP alapú távközlés. *Magyar távközlés*, Volume X, Number 6, Pages 19-22 June 1999. (in Hungarian).

Konferencia Cikkek

[C0] **Zoltán Pap**, Gyula Csopaki, Sarolta Dibuz. On the Theory of Patching, in *Proceedings of the 3rd IEEE International Conference on Software Engineering and Formal Methods, SEFM 2005,* Pages 263-271, Koblenz, Germany, September 5-9, 2005.

[C1] **Zoltán Pap**, Gyula Csopaki, Sarolta Dibuz. On FSM-based Fault Diagnosis, in *Proceedings of the Testing of Communicating Systems, 17th IFIP TC6/WG 6.1 International Conference, TestCom 2005,* Pages 159-174, Montreal, Canada, May 31 - June 2, 2005.

[C2] Gusztáv Adamis, **Zoltán Pap**, Róbert Horváth. Introduction of Aspect-Oriented Methodology to Formal Description Techniques, in *Proceedings of the IFIP WG6.3 Workshop on Next Generation Networks & EUNICE'2003,* Balatonfüred, Hungary, September 8-10, 2003.

[C3] Gusztáv Adamis, **Zoltán Pap**, Róbert Horváth. Self-adaptive Testing of Communication Protocols, in *Proceedings of IWSAS 2003, International Workshop on Self-Adaptive Software,* Arlington, VA, USA, June 9-11, 2003.

[C4] Gábor Kovács, **Zoltán Pap**, Dung Le Viet, Antal Wu-Hen-Chang, Gyula Csopaki. Applying Mutation Analysis to SDL Specifications, in *Proceedings of the Eleventh SDL Forum "System Design",* Pages 269-284, Stuttgart, Germany, 1-4 July, 2003.

[C5] **Zoltán Pap**, Zoltán Rétháti, Gusztáv Adamis. Standardized Beta Test Subsequence Based Test Generation Method Using Formal Documents, in *Proceedings of the 4th Symposium on Wireless Personal Multimedia Communications - WPMC'01,* Aalborg, Denmark, September 9-12 2001.

[C6] Gábor Kovács, **Zoltán Pap**, Gyula Csopaki. Automatic Test Selection Method Applied to WAP, in *Proceedings of IFIP Workshop on IP and ATM Traffic Management WATM'2001 & EUNICE'2001,* Pages 115-121, Párizs, France, September 3-5, 2001.

[C7] **Zoltán Pap**, Zoltán Rétháti, Gusztáv Adamis. Alternative Test Generation Method Based on Atomic Test Cases, in *Proceedings of IFIP Workshop on IP and ATM Traffic Management WATM'2001 & EUNICE'2001,* Pages 107-114, Párizs, France, September 3-5, 2001.

[C8] Róbert Horváth, **Zoltán Pap**, Zoltán Rétháti. Protokoll fejlesztés és tesztelés formális nyelveken, in *Proceedings of Students Scientific Conference,* Budapest University of Technology and Economics, Budapest, Hungary, November, 1999.