BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
TÁVKÖZLÉSI ÉS MÉDIAINFORMATIKAI TANSZÉK
INFORMATIKAI TUDOMÁNYOK DOKTORI ISKOLA

INKREMENTÁLIS TESZTGENERÁLÓ ALGORITMUSOK

Németh Gábor Árpád

Tézisfüzet

Konzulens:

Dr. Pap Zoltán
Dr. Csopaki Gyula

Budapest, Magyarország
2015.

1. Bevezetés

Mind a hardver-, mind a szoftverfejlesztés során a tesztelés fontos szereppel bír. A különböző hardver és szoftver termékek egyre összetettebbé válnak, miközben a termékfejlesztésre fordítható idő egyre rövidebb, ami a hibák kialakulásának esélyét növeli. Bár a minőségbiztosítás alapkövetelmény, a tesztelésre fordítható erőforrások sokszor korlátozottak az elvégzendő munka nagyságához képest. További probléma, hogy tesztelési folyamatokat rendszerint a tesztelendő termék belső architektúrájának részletes ismerete nélkül kell megvalósítani.

A fent vázolt problémákra egy ígéretes megoldás, ha a termék követelményeit egy formális nyelven írjuk le, és ezt a formális leírást használjuk később a tesztesetek készítésének kiindulópontjaként. A formális leírások a rendszer működésének megértésében, valamint annak szükség szerinti kibővítésében vagy újratervezésében segíthetnek már a kezdeti fejlesztési lépések során is. Telekommunikációs protokollok és szoftverek formális leírására leggyakrabban a *véges automatákat*, más néven *állapotgépeket* (Finite State Machines, FSM) használják.

Jóllehet a szakirodalom az utóbbi évtizedekben jól kidolgozta a véges automata alapú tesztgeneráló eljárások témakörét [15, 19, 6, 8], mégis kevés figyelem fordult a dinamikus esetek, így a változó rendszerspecifikációk vizsgálatára. Szinte az összes véges automata alapú tesztgeneráló eljárás merev, időben változatlan specifikációkat tételez fel. Ez annak fényében különösen meglepő, hogy a legújabb rendszerfejlesztési metodológiák inkrementális megközelítéseket javasolnak, mint például a fejlesztés alatt álló rendszer lépésről lépésre történő finomítását [26, 16, 9]. Egy ilyen módszertan például a szoftverfejlesztés világában elterjedt és népszerű Agile [21].

A tesztelés témakörében alkalmazva az iteratív megközelítések új lehetőségeket nyújtanak a jelenlegi eljárások továbbfejlesztésével. A fejlesztésben alkalmazott iterációs lépések során a változtatás hatásának felismerésével lehetővé válik a rendszer egy korábbi változatához készített minél több teszteset újrafelhasználása úgy, hogy új teszteseteket csak a változtatásban érintett részekhez kell készíteni. Ez a megközelítés folyamatosan fejlődő rendszerek esetében nagyságrendekkel hatékonyabb tesztgenerálást eredményezhet. A megközelítés egyszerre támogatja a rendszer egészén és a csupán a változtatásban érintett részeken végrehajtott tesztelést. A rendszer változatásban érintett és nem érintett részekre bontása egybecseng a nem modell alapú tesztelési technikákkal (például a funkcionális tesztelés a rendszer új funkcióira fókuszál, míg a regressziós tesztelés feladata a visszamenőleges kompatibilitás biztosítása). Ennek folyományaként a véges automata modell alapú tesztelésnél alkalmazott iteratív megközelítések könnyen meghonosíthatóak a jelenlegi szoftvertesztelési módszertanokban.

2. Kutatási célkitűzések

A disszertáció célja folyamatosan változó rendszerspecifikációkra olyan új inkrementális algoritmusokat javasolni, amelyek a meglévő eljárásoknál sokkal hatékonyabban képesek teszteseteket generálni. A figyelem középpontjában a véges automata modellezési megközelítések állnak, hiszen telekommunikációs szoftverekre és protokollokra ez a legelterjedtebb formális leírás.

Megközelítésem változó specifikációkat feltételez és egy segédinformáció – a rendszer előző változatára készített tesztkészlet – felhasználásával egy teljes tesztkészletet biztosít a továbbfejlesztett specifikációra. A kutatás a tesztesetekben a változtatások kiterjedésére összpontosít és arra, hogy hogyan lehet megfelelő frissítési függvényeket írni a különböző típusú fejlesztési lépésekre úgy, hogy hatékonyan legyünk képesek új tesztkészletet adni a megváltoztatott specifikációra.

Eredményeimet két téziscsoportra bontottam.

Az első téziscsoport egy ún. *ellenőrző szekvencia* karbantartására fókuszál, ami garantálja az adott véges automatában fellépő összes kimeneti és következő állapot hiba megtalálását. Ez a téziscsoport egy olyan strukturált tesztkészlettel foglalkozik, amelynek különböző részei akár önállóan is használhatóak különféle tesztelési célok elérésére (például a specifikáció véges automatájára vonatkozó feltételek ellenőrzésére). Ebben a téziscsoportban két algoritmust javaslok a HIS-method [28, 24] által megadott tesztszekvenciák karbantartására a rendszer specifikációjában végrehajtott változtatások során.

A második téziscsoport egy olyan tesztszekvencia karbantartására fókuszál, amelyik sokkal kevesebb megkötést kíván meg a specifikáció és implementáció állapotgépével, valamint a fejlesztési lépésekkel kapcsolatban. Ebben a téziscsoporban két algoritmust mutatok be, amelyek együtt az *állapotátmeneti séta* (Transition Tour, TT) [22] tesztszekvenciát tartják karban. Ez a tesztszekvencia a véges automata összes állapotát legalább egyszer végigjárja és – bár nem garantálja az állapotátmeneti hibák megtalálását – sokkal rövidebb, mint a HIS-method ellenőrző szekvenciája.

3. Módszertan

Disszertációmban a gráfelmélet, a véges automata elmélet és komplexitás elmélet eredményeit használtam fel.

A tézisek javasolt inkrementális algoritmusai a véges automata elmélet hagyományos algoritmusain alapulnak. A második tézis gráfelméleti eszközöket és eljárásokat is használ.

A bemutatott algoritmusokhoz mindig mellékeltem komplexitás számításokat. Az első tézis komplexitás számítása Ramalingam és Reps [25] megközelítésén alapul.

Az algoritmusokat Java-ban és C++-ban, a LEMON [2] könyvtárat felhasználva implementáltam. Az eljárásokat példákkal illusztráltam és kiterjedt szimulációkat végeztem, hogy megvizsgáljam hatékonyságukat a véges automata elmélet leginkább releváns, hagyományos eljárásaihoz képest.

4. Új eredmények

4.1. Korlátos Inkrementális Algoritmusok a HIS-method Teszteseteinek Karbantartásához a Változtatások Során

Adott egy teljesen specifikált, determinisztikus, minimál M véges automata n darab állapottal és p számú bemeneti szimbólummal. Az M állapotgép *ellenőrző szekvenciáján* azon x input szekvenciát értjük, amely képes M megkülönböztetésére minden n állapotú véges automatától. Azaz minden egyes $Impl$ implementációs állapotgép, amely nem felel meg M állapotgépnek, más kimenetet fog adni az x szekvenciára, mintha azt az M véges automatára alkalmaznánk.

Számos algoritmus született megbízható reset üzenettel rendelkező véges automaták ellenőrző szekvenciájának készítésére [19, 5]. Ilyen eljárás a W-method [7], a Wp-method [14] és a HIS-method [28, 24]. Ezen algoritmusok mindegyike hasonló, két fázisból álló struktúrát tartalmaz. Az első – állapotazonosító fázis – ellenőrzi minden egyes állapotra, hogy az az implementációban is szerepel-e. A második – állapotátmeneteket ellenőrző fázis – ellenőrzi az implementáció összes, előző fázis által nem vizsgált állapotátmetét úgy, hogy megfigyeli, hogy az állapotátmenet kimenete és következő állapota megfelel-e a specifikációban leírt viselkedésnek. A továbbiakban a HIS-methodra koncentrálunk, mert a három eljárás közül ez a leginkább általános megközelítés.

A HIS-method az alábbi két alapvető adatstruktúrát tartalmazza:

- $Q = \{q_1, \ldots, q_n\}$ *állapot lefedő halmaz* a véges automata összes $s_1 ... s_n$ állapotának eléréséhez.

- $Z = \{Z_1, \ldots, Z_n\}$ *elválasztó szekvenciák családja* (amire az angol szakirodalom időnként *family of Harmonized State Identifiers (HSI)*-ként is hivatkozik) az állapotátmenetek következő állapotának ellenőrzésére (ahol $Z_i = \{z_{ij}\}, j = = 1 \ldots n$ jelöli az elválasztó halmazt s_i állapot számára, azaz a Z_i halmaz elemei különböztetik meg az s_i állapotot az összes többi $s_j, j \neq i$ állapottól).

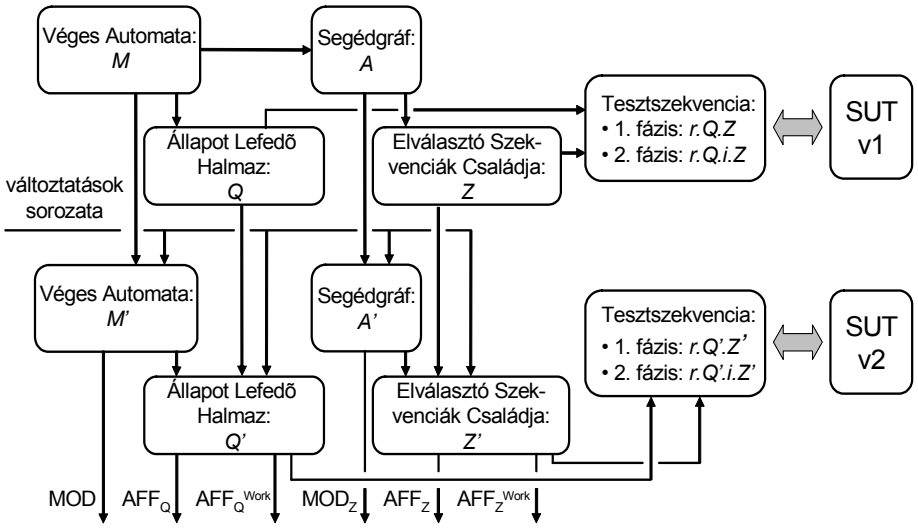
Az állapotok azonosítására szolgáló első-, és az állapotátmenetek ellenőrzésére szolgáló második rész teszteseteit a fent tárgyalt adatstruktúrák megfelelő elemeinek összefűzésével kapjuk.

A tesztelés témakörében az inkrementális megközelítést először El-Fakih és társai alkalmazták [12]. Eljárásuk azonban csak a rendszerspecifikáció változott részeire készített teszteseteket és nem volt képes a teljes tesztkészlet karbantartására a rendszerben végbemenő változtatások során. Mindemellett a bemutatott eljárás igazi, korlátos inkrementális algoritmusnak sem tekinthető, hiszen komplexitása a bemenetként kapott véges automata méretétől és nem a változtatás hatásának méretétől függ.

A kutatás fő célja egy olyan algoritmus megalkotása volt, amely hatékonyan generál ellenőrzőszekvenciát a változó specifikációkra. Legjobb tudomásom szerint a témában írott cikkünk [C1] előtt nem publikáltak olyan igazi, korlátos, inkrementális algoritmust, amelyik képes a teljes tesztkészlet karbantartására a rendszer specifikációjában végbemenő változtatások során.

1. Tézis. *[C1][C6] Két inkrementális algoritmust javasoltam az ellenőrző szekvencia karbantartására a specifikációban végrehajtott változtatások során, amennyiben a specifikáció megbízható reset üzenettel rendelkező, determinisztikus, teljesen specifikált, erősen összefüggő állapotgép formájában adott. Az algoritmusok felhasználják a specifikáció előző változatának teszthalmazát, hogy hatékonyan generáljanak új teszthalmazt a megváltozott specifikáció számára.*

Hasonlóan a HIS-methodhoz, a javasolt eljárás is két adatstruktúrát frissít a változtatások során. Az egyik algoritmus egy, az állapotok elérésére szolgáló prefixzárt állapot lefedő halmazt tart karban, rá a továbbiakban az SCSM (State Cover Set Maintenance) rövidítéssel hivatkozunk (ismertetése a 4.1.1. fejezetben). A másik algoritmus az állapotátmenetek következő állapotának azonosítására szolgáló elválasztó szekvenciákat tartja karban, rá a továbbiakban a SIM (State Identification Maintenance) rövidítéssel hivatkozunk (ismertetése a 4.1.2. fejezetben).



1. ábra. Blokkvázlat a HIS-method teszteseteinek karbantartásáról

A megközelítés vázlatos ábrázolása az 1. ábrán látható. Az ábra felső része a HIS tesztszekvenciájának hagyományos elkészítését mutatja: Az eljárás nulláról legenerálja a Q állapot lefedő halmazt és a Z elválasztó szekvenciák családját, majd az ellenőrző szekvenciát ezen halmazok megfelelő részszekvenciájának összefűzésével kapjuk meg. Ezt a tesztszekvenciát aztán a tesztelés alatt álló rendszerre (system under test, (SUT)) adjuk bemenetként. Az A gráf az M véges automata állapotpárjai fölött értelmezett segédgráf; a Z elválasztó szekvenciák családjának készítését segíti. Az ábra alsó része azt mutatja, hogy a változtatások sorozata hogyan érinti a HIS-method teszteseteinek az elkészítését. A változtatási operátorok sorozata az M véges automatát M'-be viszi. Ahelyett, hogy az ábra felső részén lévő folyamatot ismételnénk meg, az SCSM algoritmus (ismertetése a 4.1.1. fejezetben) felhasználja a rendszer előző verziójához készített Q állapot lefedő halmazt, hogy egy új, érvényes, Q' állapot lefedő halmazt adjon M'-re. A SIM algoritmus (ismertetése a 4.1.2. fejezetben) a már meglévő Z elválasztó szekvenciák családját tartja karban, hogy hatékonyan elkészítse a megváltoztatott automatára érvényes Z'-t. Ezen lépések után a megfelelő tesztszekvenciákat össze lehet fűzni, majd alkalmazni a megváltoztatott tesztelendő rendszerre (SUT v2).

Az algoritmusok az AFF_Q és AFF_Z halmazokba gyűjtik azon állapotokat és állapotpárokat, amelyeknek vonatkozó tesztesetei módosultak. Így a tesztelési céltól függően akár a teljes rendszer, akár annak csak változtatásban érintett részei megvizsgálhatóak. A M' állapotgép és A' segédgráf módosult elemeit az eljárások a MOD és MOD_Z, halmazokba gyűjtik.

Mivel az algoritmusok egy teljes ellenőrző szekvenciát, a specifikáció állapotgépet,

valamint ez utóbbi segédgráfját is karbantartják a változtatások során, ezért kimenetük a következő iteráció bemeneteléül is szolgálhatnak, azaz az algoritmusok támogatják a specifikáció lépésről-lépésre történő fejlesztését.

A [23] cikken alapulva az alábbi lehetséges változtatásokat vettük figyelembe az M véges automata módosításához:

– állapotátmenet következő állapotának megváltoztatása,

– állapotátmenet kimenetének megváltoztatása.

4.1.1. Inkrementális Algoritmus Prefix-zárt Állapot Lefedő Halmaz Karbantartására

1.1. Tézis. *[C1][C6] Készítettem egy algoritmust az s_0 kezdőállapotban gyökerező állapot lefedő halmaz karbantartására. A javasolt algoritmus a megadott állapot lefedő halmazt csak azon állapotokra frissíti, amelyeket a változtatások sorozata érintett. Az algoritmus képes észlelni, ha az erősen összefüggőségre vonatkozó kezdeti feltételezés sérül a változtatások sorozatának végrehajtása után.*

Adott egy M-el jelölt specifikáció állapotgép, M Q-val jelölt prefix-zárt állapot lefedő halmaza és a változtatások Ω sorozata, ami M állapotgépet egy M' állapotgépbe viszi. A cél egy új, érvényes, Q' prefix-zárt állapot lefedő halmaz készítése M'-re.

Mivel az M véges automata Q halmaza egy s_0 állapotban gyökerező ST feszítőfával reprezentálható, ezért a problémát vissza lehet vezetni egy feszítőfa karbantartására a változtatások során. A cél tehát egy olyan új, érvényes ST' feszítőfa karbantartása, ami az M' állapotgép Q' halmazának felel meg.

Az SCSM (State Cover Set Maintenance, Állapot Lefedő Halmazt Karbantartó) algoritmus leírásánál a következő jelöléseket használjuk. Egy állapotátmenetet ST állapotátmenetnek nevezünk akkor és csak akkor, ha az állapotátmenet része az ST feszítőfának. ST feszítőfa s_i állapotban gyökerező részfáját ST_{s_i}-vel jelöljük. A MOD halmaz tartalmazza a *módosított* állapotokat: $s_x \in MOD$ akkor és csak akkor, hogyha az M véges automata egy s_x állapotból kiinduló állapotátmenete módosult. Az s'_y állapotot *q-érintett* állapotnak nevezzük és AFF_Q halmazba helyezzük, akkor és csak akkor, ha M' állapotgép s'_y állapotát érintette a változtatás a Q halmazra nézve. Ha egy állapot q-érintett, akkor az állapot eléréséhez új input tesztszekvenciát kell rá készíteni. AFF_Q^{Work} jelöli azon érintett állapotok nyilvántartására szolgáló munkahalmazt, amelyekre még nem készült új tesztszekvencia. Egy adaptív paramétert szintén definiálunk a változtatás bemenetre és kimenetre gyakorolt hatásának nyomonkövetésére: $\Delta_Q = |MOD \cup AFF_Q|$. Az algoritmus komplexitását ezen Δ_Q paraméter segítségével fejtjük ki az 1.2 tézisben.

Algoritmus 1: Prefix-zárt Állapot Lefedő Halmaz Karbantartása

input : $M = (I, O, S, T)$; $\Omega = \{\omega_1..\omega_K\}$; $ST = (S_{ST}, T_{ST}), T_{ST} \subset T$; $S_{ST} \subseteq S$;
output: ST'; MOD; AFF_Q; AFF_Q^{Work}; M';

0 data($k = 1..K$; $M' = (I, O, S', T')$; $ST' = (S'_{ST}, T'_{ST})$; $MOD \subseteq S$; $AFF_Q \subseteq S$; $AFF_Q^{Work} \subseteq S$; $ST_{temp} = (S_{temp}, T_{STtemp})$);

/* A $TreeWalk$ fv. az adott ST' feszítõfát járja be a megadott s'_b gyökércsomóponttól szélességi keresést használva. Kimenetként az s'_b csomópontból elérhetõ csomópontok ($.S$) és állapotátmenetek ($.T$) halmazát adja. A $SpanningTree$ fv. az AFF_Q^{Work} halmazban lévõ állapotokra ad egy feszítõfát szélességi kereséssel a megadott s'_j gyökércsomópontból. */

1 $M' := M$; $S'_{ST} := S_{ST}$; $T'_{ST} := T_{ST}$; $MOD := \emptyset$; $AFF_Q := \emptyset$; $AFF_Q^{Work} := \emptyset$;

/* 1. rész: Változtatások végrehajtása az M állapotgépen és a változtatásban érintett állapotok meghatározása */

2 **foreach** k *where* $\omega_k(s_a, i, o_x, s_b) = (s'_a, i, o_y, s'_c) \in \Omega, s_a, s_b \in S, s'_a, s'_c \in S', i \in I, o_x, o_y \in O$ **do**

3 $T' := T' \setminus (s'_a, i, o_x, s'_b) \cup (s'_a, i, o_y, s'_c)$; $MOD := MOD \cup \{s'_a\}$;

4 **if** $(s_b \neq s'_c) \wedge ((s'_a, i, o_x, s_b) \in T'_{ST})$ **then**

5 $AFF_Q := AFF_Q \cup \texttt{TreeWalk}(ST', s'_b).S$;

6 $AFF_Q^{Work} := AFF_Q^{Work} \cup \texttt{TreeWalk}(ST', s'_b).S$;

7 $S'_{ST} := S'_{ST} \setminus \texttt{TreeWalk}(ST', s'_b).S$; $T'_{ST} := T'_{ST} \setminus \texttt{TreeWalk}(ST', s'_b).T$;

/* 2. rész: Tesztesetek készítése az érintett állapotokra */

8 **foreach** $s'_j \in AFF_Q^{Work}$ **do**

9 **foreach** $t = (s'_i, i, o, s'_j) \in T$ **do**

10 **if** $s'_i \notin AFF_Q^{Work}$ **then**

11 $ST'_{temp}(S'_{STtemp}, T'_{STtemp}) := \texttt{SpanningTree}(M', s'_j, AFF_Q^{Work})$;
$T'_{ST} := T'_{ST} \cup \{(s'_i, s'_j)\} \cup T'_{STtemp}$; $S'_{ST} := S'_{ST} \cup \{s'_j\} \cup S'_{STtemp}$;

12 $AFF_Q^{Work} := AFF_Q^{Work} \setminus (\{s'_j\} \cup S'_{STtemp})$;

13 **return** $ST' \in (V', E')$, MOD, AFF_Q, AFF_Q^{Work}, M';

Miután AFF_Q^{Work} összes elemét megvizsgálta, az SCSM algoritmus futása befejeződik az alábbi kimenetekkel:

- ST' feszítőfa az M' állapotgép Q' frissített, prefix-zárt állapot lefedő halmazát tartalmazza.
- MOD halmaz elemei azon állapotokat jelölik, ahol az állapotgép módosult.
- AFF_Q halmaz elemei M' azon állapotait jelölik, amelyekre az állapot lefedő halmaz tesztszekvenciái megváltoztak

– AFF_Q^{Work} halmaz elemei M' azon elemeit reprezentálják, amelyek, az M-en alkalmazott Ω változtatások sorozata után elérhetetlenek lettek M' kezdőállapotából.

1.2. Tézis. *[C1] Bebizonyítottam, hogy az állapot lefedő halmaz karbantartásáért felelős inkrementális algoritmus komplexitása csupán a változtatás hatásának kiterjedésétől függ és $O(p \cdot \Delta_Q)$ komplexitású, ahol $1 \leq \Delta_Q \leq n$.*

Bizonyítás. Először is megjegyezzük, hogy elemi változtatás esetén $|MOD| = 1$, és összetett változtatás esetén $1 \leq |MOD| \leq n$. Az érintett állapotok száma – ahol a teszthalmaz módosítása szükséges – $0 \leq |AFF_Q| \leq n$.

Az algoritmus két részből áll. Az első rész az M' állapotgép q-érintett állapotait határozza meg és a hozzájuk vezető feszítőfa éleket törli. Ezután a 2. rész kibővíti a feszítőfát ezekre a q-érintett állapotokra egy érvényes Q' halmazt adva a módosított M' véges automatára.

Az SCSM algoritmus első része M állapotgépet frissíti M'-vé, meghatározza az érintett állapotokat és törli a hozzájuk vezető éleket. Az FSM frissítése Ω változtatások sorozatával $O(|MOD|)$ lépésszámot igényel. Az érintett állapotok meghatározása szélességi kereséssel, az állapotok AFF_Q és AFF_Q^{Work} halmazba tevése, valamint ezen állapotokba vezető feszítőfa élek törlése összesen $O(|AFF_Q|)$ komplexitású.

Az SCSM algoritmus második része állapotátmeneteket keres olyan állapotokból, amelyek nem tagjai az AFF_Q^{Work} halmaznak, olyan állapotokba, amelyek elemei az AFF_Q^{Work} halmaznak. Pontosan $p \cdot |AFF_Q^{Work}|$ állapotátmenet indul az AFF_Q^{Work} állapotokból, tehát legfeljebb $p \cdot |AFF_Q^{Work}| \leq p \cdot |AFF_Q|$ olyan lépés van, ami nem talál útvonalat egy nem-AFF_Q^{Work} állapotból egy AFF_Q^{Work} állapotba. Tehát legrosszabb esetben $(p+1) \cdot |AFF_Q|$ lépés szükséges. Amennyiben az algoritmus talált egy nem-AFF_Q^{Work} állapotból egy $s'_j \in AFF_Q^{Work}$ állapotba vezető állapotátmenetet, akkor az adott s'_j állapotból elérhető összes olyan állapotot, amely AFF_Q^{Work} halmazban található, törli és az ST' feszítőfát kibővíti. Mivel pontosan $p \cdot |AFF_Q^{Work}|$ állapotátmenet indul AFF_Q^{Work}-beli állapotokból, ez maximum $p \cdot |AFF_Q^{Work}| \leq$ $\leq p \cdot |AFF_Q|$ lépést igényel. Mivel minden egyes $p \cdot |AFF_Q|$ állapotátmenetet maximum kétszer vizsgált meg az SCSM algoritmus, $2 \cdot p \cdot |AFF_Q|$-nál nem több lépés, tehát $O(p \cdot |AFF_Q|)$ lépés szükséges az algoritmus második részében.

Mivel $\Delta_Q = |MOD \cup AFF_Q|$, az SCSM algoritmus teljes idő komplexitása $O(p \cdot \Delta_Q)$, ahol $1 \leq \Delta_Q \leq n$.

□

Az állapot lefedő halmaz karbantartásért felelős inkrementális algoritmus tárhely komplexitása $O(p \cdot n)$, ami megegyezik az eredeti HIS-method megfelelő részének tárhely komplexitásával.

4.1.2. Inkrementális Algoritmus Elválasztó Szekvenciák Családjának Karbantartására

1.3. Tézis. *[C1][C6] Készítettem egy algoritmust, amely karbantartja az elválasztó szekvenciák családját a változtatások során. Az elválasztó szekvenciák családját egy feszítőerdőként ábrázoltam egy segédgráf fölött, amelynek minden egyes csomópontja a véges automata állapotgráfjában egy állapotpárnak felel meg. A javasolt algoritmus a feszítőerdőt csak azokra az állapotpárokra frissíti, amelyek a változtatásban érintettek. Az algoritmus képes annak észlelésére, ha a minimál állapotgépre vonatkozó eredeti feltételezések sérülnek a változtatások szekvenciájának végrehajtása során.*

A SIM (State Identification Maintenance, Állapot Azonosítást Karbantartó) algoritmusnak azonosítania kell M' azon állapotpárjait, amelyeket a továbbiakban nem lehet az M állapotgép Z halmazának megfelelő elválasztó szekvenciájával megkülönböztetni, és generálnia kell új elválasztó szekvenciákat ezen állapotpárokhoz. Definiáljunk egy A segédgráfot $n(n+1)/2$ darab csomóponttal úgy, hogy ebben a segédgráfban minden egyes csomópont megfelel M egy $< s_k, s_l >$ állapotpárjának, beleértve az $< s_k, s_k >$ önállapotpárt is. A-ban akkor és csak akkor vezet $< s_k, s_l >$-ből $< s_m, s_n >$-ba él i bemeneti szimbólummal jelölve, hogyha $\delta(s_k, i) = s_m$ és $\delta(s_l, i) = = s_n$ az M állapotgépben, ahol δ: $S \times I \to S$ jelöli M következő állapot függvényét.

Az A irányított segédgráfot használja a SIM algoritmus M állapotgép elválasztó szekvenciák családjának ábrázolásához és karbantartásához. Az algoritmus az A segédgráfot frissíti minden egyes inkrementális lépésben. Jelölje ezt a módosított segédgráfot A'!

Az algoritmus leírásához az alábbi definíciókat és jelöléseket használjuk:

Egy $< s_x, s_y >$ állapotpárt *elválasztó állapotpárnak* hívunk akkor és csak akkor, ha rendelkezik egy olyan i *elválasztó bemenettel*, ami különböző kimenetet ad a két állapoton alkalmazva, azaz $\lambda(s_x, i) \neq \lambda(s_y, i)$ adott $i \in I$ esetén (ahol λ: $S \times I \to O$ jelöli M kimeneti függvényét). M állapotgép minimál akkor és csak akkor, hogyha létezik egy útvonal minden egyes $< s_k, s_l >, k \neq l$ állapotpárból egy $< s_x, s_y >$ elválasztó állapotpárba. A bemeneti szimbólumokat az $< s_k, s_l >$ állapotpárból az $< s_x, s_y >$ állapotpárba haladva összefűzzük $< s_x, s_y >$ elválasztó bemenetével és így egy elválasztó szekvenciát kapunk az s_k és s_l állapotokra.

M állapotgép Z halmazának hatékony karbantartásához az alábbi megkötéseket tesszük Z elválasztó szekvenciáira vonatkozóan: (I) Minden egyes $< s_x, s_y >$ elválasztó állapotpárhoz egyetlen $i \mid \lambda(s_x, i) \neq \lambda(s_y, i)$ elválasztó bemenetet rendelünk. Amennyiben az adott elválasztó állapotpárnak több ilyen bemenete lenne, akkor azokból egyet véletlenszerűen választunk. (II) M állapotgép elválasztó szekvenciái prefix-zártak. Megjegyezzük, hogy a két megkötés nem korlátozza a SIM algoritmus alkalmazhatóságát, hiszen az minden egyes fejlesztési lépésben olyan Z' elválasztó szekvenciák családját készít, ami a fenti megkötéseknek megfelel.

Ha a fenti megkötések igazak, akkor az M elválasztó szekvenciáinak családja megadható az A állapotpárgráf nem önállapotpárjai fölött értelmezett SF erdőként úgy, hogy annak minden egyes fája egy elválasztó állapotpárban gyökerezik és a fák élei a gyökércsomópontok felé mutatnak. Azaz a Z' elválasztó szekvenciák családjának karbantartása a változtatott M' véges automatára visszavezethető az elválasztó állapotpárok, a hozzájuk rendelt elválasztó bemenet és A' nem önállapotpárjai fölött értelmezett SF' feszítőerdő karbantartására a változtatások során.

A egy élét SF-élnek hívjuk, akkor és csak akkor, ha SF-nek tagja. $SF < s_i, s_j >$ állapotpárban gyökerező részfáját $SF_{<s_i,s_j>}$-el jelöljük. A MOD_Z halmaz tartalmazza A' módosított segédgráf *módosított* állapotpárjait: $< s'_i, s'_j > \in MOD_Z$ akkor és csak akkor, ha az s_i vagy s_j kezdőállapotú állapotátmenet módosult M állapotgépben egy változtatás hatására. Ha egy változtatás módosított egy s_x kezdőállapotú állapotot, akkor az összes olyan állapotpár A'-ban mely s'_x-t tartalmazza, módosult. Az $< s'_x, s'_y >$ állapotpárt *z-érintettnek* hívjuk és AFF_Z halmazban gyűjtjük, ha az A' segédgráfban a $< s'_x, s'_y >$ állapotpárt érintette a változtatás a Z halmazra vonatkozóan, azaz új elválasztó szekvenciát kell készíteni az s'_x és s'_y állapotokra. Egy AFF_Z^{Work}-el jelölt munkahalmazt használunk azon érintett állapotpárok nyomonkövetésére, amelyekre új tesztszekvenciák még nem készültek. Definiálunk egy adaptív paramétert a változtatás bemenetre és kimenetre gyakorolt hatásának nyomonkövetésére is: $\Delta_Z = |MOD_Z \cup AFF_Z|$. Az algoritmus komplexitását ezen Δ_Z paraméter segítségével fejtjük ki az 1.4 tézisben.

Mint a korábban ismertetett SCSM algoritmus, a SIM algoritmus szintén két részből áll. Az első rész azonosítja a z-érintett állapotpárokat és törli az érvénytelen tesztkészleteiket, míg a második rész kibővíti az SF' feszítőfát úgy, hogy az egy új, érvényes Z' halmazt adjon az M' módosított állapotgépre.

Amikor az AFF_Z^{Work} halmaz összes elemét megvizsgálta, a SIM algoritmus futása lezárul:

- SF' reprezentálja a módosított M' állapotgép Z' elválasztó szekvenciáinak családját.
- MOD_Z jelöli A' azon állapotpárjait, amelyek módosultak.
- AFF_Z jelöli M' állapotgép azon állapotpárjait, amelyeknek elválasztó szekvenciája megváltozott.
- AFF_Z^{Work} jelöli az ekvivalens állapotpárokat; a SIM algoritmus tehát képes annak jelzésére is, hogy az Ω változtatások alkalmazása után is minimál marade az állapotgép.

Algoritmus 2: Elválasztó Szekvenciák Családjának Karbantartása

input : $M = (I, O, S, T)$, $A = (V, E)$, $V = S \times S$; $E \subseteq V \times V$;
$\Omega = \{\omega_1, .., \omega_K\}$; $SF = (V_{SF}, E_{SF}), V_{SF} \subset V, E_{SF} \subseteq E$;
output: : $SF' = (V'_{SF}, E'_{SF})$; MOD_Z; AFF_Z; AFF_Z^{Work}; A';

0 data($k = 1..K$; $A' = (V', E')$; $SF' = (V'_{SF}, E'_{SF})$; $MOD_Z \subseteq V$; $AFF_Z \subseteq V$;
$AFF_Z^{Work} \subseteq V$; $A'_{AFF} = (V'_{AFF}, E'_{AFF})$; $ST'_{temp} = (V'_{temp}, E'_{temp})$)

1 $A' := A$; $V'_{SF} := V_{SF}$; $E'_{SF} := E_{SF}$; $MOD_Z := \emptyset$; $AFF_Z := \emptyset$; $AFF_Z^{Work} := \emptyset$;

2 **foreach** k *where*
$\omega_k(s_a, i, o_x, s_b) = (s'_a, i, o_y, s'_c) \in \Omega, s_a, s_b \in S, s'_c \in S', i \in I, o_x, o_y \in O$ **do**

3 **foreach** $e = (v_m, i, o_x, v_n), v_m = (s_m, s_n), v_n \in V, s_m = s'_a \vee s_n = s'_a$ **do**

4 $v_n := (s'_c, s'_l)$; $e := (v_m, i, o_y, v_n)$; $MOD_Z := MOD_Z \cup \{(v_m)\}$

/* 1. rész: érintett állapotpárok meghatározása */

/* Kimeneti változtatások esete */

5 **if** $o_x \neq o_y$ **then**

6 **foreach** $s'_j \in S'$ **do**

7 **if** (i has been *sep. input* of $< s_a, s_j >$)and($\lambda'(s'_a, i) = \lambda'(s'_j, i)$) **then**

8 **foreach** $i_t \in I$ **do**

9 **if** $\lambda'(s'_a, i_t) \neq \lambda'(s'_j, i_t)$ **then**

10 $< s'_a, s'_j >$ remains a *sep. state pair* with i_t *sep. input*

11 $AFF_Z := AFF_Z \cup$ TreeWalk$(SF', < s'_a, s'_j >).V$;

12 **else if** $\forall i_t \in I, \lambda'(s'_a, i_t) = \lambda'(s'_j, i_t)$ **then**

13 *separating state pair* mark removed from $< s'_a, s'_j >$

$AFF_Z := AFF_Z \cup$ TreeWalk$(SF', < s'_a, s'_j >).V$;

$AFF_Z^{Work} =: AFF_Z^{Work} \cup$ TreeWalk$(SF', < s'_a, s'_j >).V$;

14 $E'_{SF} := E'_{SF} \setminus$ TreeWalk$(SF', < s'_a, s'_j >).E$;

15 $V'_{SF} := V'_{SF} \setminus$ TreeWalk$(SF', < s'_a, s'_j >).V$;

16 **else if** $\neg(< s_a, s_j >$ *sep. s. p.*)and($\lambda'(s'_a, i) \neq \lambda'(s'_j, i)$) **then**

17 $E'_{SF} := E'_{SF} \setminus \{(< s'_a, s'_j >, < \delta(s'_a), \delta(s'_j) >)\}$;

18 $< s'_a, s'_j >$ marked as a *separating state pair* with i *sep. input*

19 $AFF_Z := AFF_Z \cup$ TreeWalk$(SF', < s'_a, s'_j >).V$;

Algoritmus 2: Elválasztó Szekvenciák Családjának Karbantartása

/* Következõ állapot változtatások esete */
else if $s_b \not\cong s'_c$ **then**
  foreach $s'_j \in S'$ **do**
    if $(< s'_a, s'_j >, i, o, < s'_c, \delta(s'_j) >) \in E'_{SF}$ **then**
      $AFF_Z := AFF_Z \cup \texttt{TreeWalk}(SF, < s'_a, s'_j >).V$;
      $AFF_Z^{Work} := AFF_Z^{Work} \cup \texttt{TreeWalk}(SF, < s'_a, s'_j >).V$;
      $E'_{SF} := E'_{SF} \setminus \texttt{TreeWalk}(SF, < s'_a, s'_j >).E$;
      $V'_{SF} := V'_{SF} \setminus \texttt{TreeWalk}(SF, < s'_a, s'_j >).V$;
/* 2. rész: Teszt esetek készítése a változtatásban érintett állapotpárokra */
/* A'_{AFF} algráf készítése az érintett állapotpárokra */
$V'_{AFF} := \emptyset$; $E'_{AFF} := \emptyset$;
foreach $v'_i \in AFF_Z^{Work}$ **do**
  $V'_{AFF} := V'_{AFF} \cup \{v'_i\}$;
  foreach $e = (v'_i, i, o, v'_j)$ **do**
    if $v'_j \in AFF_Z^{Work}$ **then** $E'_{AFF} := E'_{AFF} \cup (v'_i, v'_j)$
    else v'_i marked with v'_j
/* SF' feszítõerdõ kibõvítése az érintett állapotpárokra */
foreach v'_i *marked with* v'_j **do**
  $ST'_{temp}(V'_{temp}, E'_{temp}) := \texttt{SpanningTree}((A'_{AFF}), v'_i)$;
  $E'_{SF} := E'_{SF} \cup \{(v'_i, v'_j)\} \cup E'_{temp}$; $V'_{SF} := V'_{SF} \cup V'_{temp}$;
  $AFF_Z^{Work} := AFF_Z^{Work} \setminus V'_{temp}$;
return $SF' \in (V'_{SF}, E'_{SF})$, MOD_Z, AFF_Z, AFF_Z^{Work}, A';
/* A *TreeWalk* fv. az adott SF' feszítõerdõt járja be a megadott gyökércsomópontból. A *SpanningTree* fv. az AFF_Z^{Work} halmazban lévõ állapotpárokra ad egy feszítõfát a megadott gyökércsomópontból. */

1.4. Tézis. *[C1] Bebizonyítottam, hogy az elválasztó szekvenciák családjának karbantartásáért felelős inkrementális algoritmus komplexitása csupán a változtatás hatásának kiterjedésétől függ és $O(p \cdot \Delta_Z)$ komplexitású, ahol $n \leq \Delta_Z \leq n^2$.*

Bizonyítás. Először is megjegyezzük, hogy elemi változtatás esetén $|MOD_Z| = n$, és összetett változtatás esetén $n \leq |MOD_Z| \leq n \cdot (n+1)/2$. Az AFF_Z halmaz mérete $0 \leq |AFF_Z| \leq n \cdot (n-1)/2$.

Az első részben a SIM algoritmus az A segédgráfot A' segédgráfra frissíti, valamint meghatározza az érintett állapotpárokat. A segédgráf frissítése az Ω változtatások hatására $O(|MOD_Z|)$ lépést igényel. Az érintett állapotpárok meghatározása, AFF_Z

és AFF_Z^{Work} halmazokba tevése, valamint az érvénytelen feszítőerdő élek törlése összesen $O(|AFF_Z|)$ komplexitású.

A második részben a SIM algoritmus egy A'_{AFF} részgráfot hoz létre az AFF_Z^{Work} állapotpárokra és megjelöli azon állapotpárokat, amelyeknek van egy nem-AFF_Z^{Work} állapotpárba vezető tagja; ehhez $O(p \cdot |AFF_Z|)$ lépés szükséges. Az SF' feszítőerdő kibővítésére az AFF_Z^{Work} halmaz elemeivel $O(p \cdot |AFF_Z|)$ lépéssel jár.

Mivel $\Delta_Z = |MOD_Z \cup AFF_Z|$, a SIM algoritmus teljes komplexitása $O(p \cdot \Delta_Z)$, ahol $n \leq \Delta_Z \leq n^2$.

□

Az elválasztó szekvenciák családjának karbantartásáért felelős inkrementális algoritmus tárhely komplexitása $O(p \cdot n^2)$, ami megegyezik az eredeti HIS-method megfelelő részének tárhely komplexitásával.

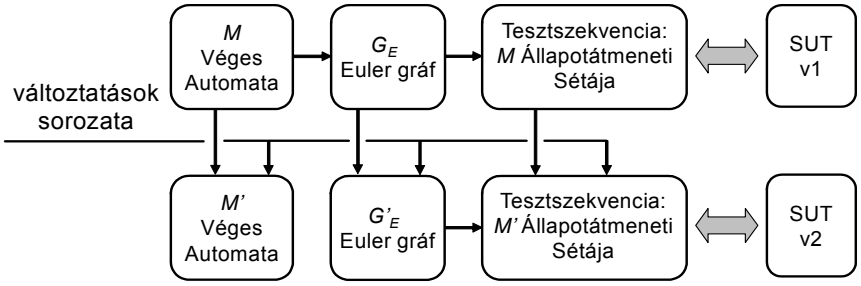
4.2. Inkrementális Algoritmusok az Állapotátmeneti Séta Karbantartására

A Kínai Postás Probléma feladata egy olyan minimális hosszúságú séta keresése, amely az adott gráf minden egyes élén legalább egyszer áthalad és visszatér a kiindulópontba. Az eredményül kapott szekvenciát az adott gráf postás útvonalának nevezik. Az eredeti problémát a [18] cikk ismerteti, a probléma különféle variációit és a rá adott megoldásokat pedig a [11] cikk foglalja össze. A Kínai Postás Probléma egy speciális esete az Irányított Kínai Postás Probléma, ahol a gráf minden egyes éle irányított. Véges automata alapú tesztelésre a szakirodalom [22] a TT-method-ot (Transition Tour, Állapotátmeneti Séta) javasolja. A TT-method egyetlen tesztszekvenciát ad, amely a determinisztikus, erősen összefüggő véges automataként adott specifikáció minden egyes állapotátmenetén áthalad. A TT-method megegyezik az Irányított Kínai Postás Probléma egy speciális esetével, ahol minden egyes élnek egységnyi súlya van.

A hagyományos TT-method változatlan specifikációkat tételez fel, így a tesztgenerálás nem túl hatékony iteratívan fejlődő rendszerek esetén. Célom egy olyan megoldás találása volt, amely képes a TT tesztszekvencia hatékony elkészítésére változó specifikációk esetében.

2. Tézis. *[J2][C7] Két együttműködő algoritmust javasoltam az Állapotátmeneti Séta tesztszekvencia karbantartására a változtatások során, amennyiben a rendszerspecifikáció determinisztikus, erősen összefüggő véges automata formájában adott. Az algoritmusok olyan létező információkat használnak fel, amelyek a specifikáció előző változatához készültek.*

Hasonlóan az Irányított Kínai Postás Problémát megoldó többi algoritmushoz [10, 3, 20], megoldásom – amelyik a (TT) [22] teszt szekvenciát tartja karban a változtatások során – két fő részből áll: Az első egy gráf kiegyenlítettségét tartja karban a változtatások során – erre a továbbiakban KGE-ként (Keep Graph Eulerian) hivatkozunk, ismertetése a 4.2.1. fejezetben – míg a második egy Euler-kört tart karban ezen a kiegyenlített gráfban – erre a továbbaikban KET-ként (Keep Euler Tour (KET) hivatkozunk, ismertetése a 4.2.2. fejezetben.



2. ábra. Blokkvázlat az Állapotátmeneti Séta tesztszekvencia karbantartásáról

A megközelítés magas szintű leírását a 2. ábra mutatja. Az ábra felső része a tesztszekvencia hagyományos készítési módját vázolja: az M véges automatát átalakítjuk egy G_E Euler gráffá[1], majd egy Euler-kört készítünk erre a G_E gráfra. G_E gráf Euler-köre az M állapotgép TT tesztszekvenciájának feleltethető meg, amit aztán a tesztelés alatt álló rendszerre (system under test, (SUT)) adunk bemenetként. Az ábra alsó fele azt mutatja, hogy a véges automatában végbemenő változtatások hogyan érintik a TT tesztszekvencia készítését. A változtatások az automatát M-ből M'-be viszik. Ahelyett, hogy megismételnénk M'-re a tesztgenerálás már ismertetett folyamatát, a KGE algoritmus (leírása a 4.2.1. fejezetben) a G_E gráf kiegyenlítettségét őrzi meg a változtatások során; eredményül a G'_E gráfot kapjuk. A KET algoritmus (ismertetése a 4.2.2 fejezetben) pedig egy Euler-kört tart karban az előző algoritmus által kimenetként adott G'_E gráfon, amely Euler-kör a módosított M' automata TT tesztszekvenciájának feleltethető meg. Ezután az eredményül kapott TT tesztszekvencia beadható a módosított tesztelendő rendszernek (SUT v2). Az eljárás kimenete a következő iteráció bemenetétül is szolgálhat, azaz az algoritmusok támogatják a specifikáció lépésről-lépésre történő fejlesztését.

[1] Az M véges automata átalakítható egy G irányított gráffá, melynek élei és csomópontjai M állapotátmeneteinek és állapotainak felelnek meg. Ezután G néhány élét úgy duplikáljuk, hogy egy Euler gráfot kapjunk; ezt a kibővített gráfot G_E-vel jelöljük.

A [4] cikkben ismertetett hiba modell alapján az algoritmusok az alábbi változtatási operátorokat veszik figyelembe a specifikáció állapotgépén:

– állapotátmenet hozzáadása,

– állapotátmenet elvétele,

– állapot hozzáadása egy belőle kifele mutató és egy rá mutató állapotátmenettel,

– állapot elvétele,

– állapotátmenet kimeneti címkéjének módosítása,

– állapotátmenet végállapotának módosítása.

A kimeneti állapot és következő állapot változtatások gyakran alkalmazott lépések változó rendszerspecifikációk leírásánál, mert megőrzik a véges automata determinisztikus tulajdonságát. Állapot hozzáadása egy-egy az állapotba irányuló és onnan induló állapotátmenettel szintén gyakran alkalmazott lépés, mert megóvja az újonnan hozzáadott állapotot attól, hogy el legyen zárva az automata többi részétől. Mindazonáltal a fent ismertetett változtatások némelyike lehet, hogy nem őrzi meg az állapotgép erősen összefüggőségét. Esetünkben azonban ez nem probléma, hiszen a 4.2.2 fejezetben ismertetett algoritmus képes annak észlelésére, hogyha az erősen összefüggőségre vonatkozó feltétel nem teljesül a változtatási lépések végrehajtása során.

A továbbiakban a véges automaták gráf reprezentációját használjuk az algoritmusok ismertetésénél, tehát a fent tárgyalt véges automata hibaoperátorok gráf reprezentációját fogjuk használni.

4.2.1. A gráf kiegyenlített állapotában tartására szolgáló algoritmus

2.1. Tézis. *[J2][C7] Készítettem egy algoritmust, amely egy adott Euler gráf minden egyes csomópontjára megőrzi a kiegyenlítettséget a változtatások során. A javasolt algoritmus biztosítja, hogy a kiegészítő éleket tartalmazó gráf körmentes legyen és ezáltal meggátolja annak túlságos növekedését a változtatások során. Az algoritmus képes annak észlelésére, hogyha a gráf erősen összefüggősége sérül a változtatások sorozatának alkalmazása után.*

Az eredeti G vagy G' gráf élét *fekete élnek* hívjuk. Egy élt *piros élnek* hívunk akkor, ha egy fekete él duplikálásával jött létre, hogy a kiegészített G_E vagy G'_E gráf Euler-gráf legyen.

A KGE algoritmus két fő logikai egységből áll, melyek két különböző alproblémára nyújtanak megoldást:

1. Egy függvény, amely kiegyenlít két paraméterül kapott csomópontot, amelyek korábban kiegyenlítetlenné váltak.

2. Egy fő egység, amely meghatározza, hogy a különböző típusú változtatások hogyan tették kiegyenlítetlenné a G_E gráf csomópontjait, és a fenti függvény segítségével kiegyenlíti őket.

Kiegyenlítetlen csomópontok kiegyenlítése

Röviden összefoglalva a KGE algoritmus új *zöld éleket* húz be azon csomópontok közé, amelyek a változtatások során kiegyenlítetlenné váltak. A KGE algoritmus ezután kifejti ezeket a zöld éleket piros élek sorozatává úgy, hogy duplikál néhány olyan élt, amelyek az eredeti gráfban szerepelnek. Először megtalálja az útvonalat a két kiegyenlítetlen csomópont között szélességi kereséssel; az útvonalon fekvő éleket hozzáadja G'_E gráfhoz piros élekként; ezeket *útvonal éleknek* nevezzük. Ezután az algoritmus ellenőrzi, hogy az új útvonal élek nem alkotnak-e köröket más piros élekkel. Ha igen, akkor az algoritmus törli a köröket, azaz törli a körhöz tartozó piros él példányokat. Azért törölhetőek ezek a piros élek, mert minden piros élhez tartozik egy fekete él, tehát az eredeti gráf minden egyes éle le lesz fedve a G'_E Euler-körével és a séta hossza csökkenthető azon körök törlésével, amelyek csak piros éleket tartalmaznak. A piros élekből álló körök meghatározásához és törléséhez a KGE algoritmus meghatározza a piros élek fölött értelmezett gráf erősen összefüggő komponenseit Tarjan algoritmusát [27] felhasználva.

Változtatások kezelése

Ez a rész meghatározza, hogy a különböző típusú változtatások hogyan alakították a G_E gráf csomópontjait kiegyenlítetlenné és az előbb ismertetett eljárást használják azok kiegyenlítésére. Hogyha a megváltozott élnek voltak piros él példányai is, akkor azokat szintén kezeli.

Az algoritmus futása véget ér, amikor a gráf összes csomópontját kiegyenlítette. Az eredményül kapott G'_E gráf piros élei fölött értelmezett részgráf továbbra is körmentes marad. Az algoritmus képes megmutatni, hogyha az erősen összefüggőségre vonatkozó feltételek már nem állnak fenn az Ω változatások alkalmazása után.

4.2.2. Az Euler-kör karbantartására szolgáló algoritmus

2.2. Tézis. *[J2][C7] Készítettem egy algoritmust, amely inkrementálisan karbantartja a következő-csomópont formában megadott Euler-kört. Az algoritmus az előző rendszerverzió számára készített Euler-kört használja fel és csak azokat a részeket módosítja, amelyeket a változtatások sorozata érintett. A javasolt algoritmus hatékonyabb, mint a hagyományos eljárások kis változtatások esetén. Az algoritmus képes annak észlelésére, ha az erősen összefüggőségre vonatkozó korábbi feltételezés nem áll fenn a változtatások sorozatának alkalmazása után.*

Az algoritmus bemenete (1) egy Euler-kör a G_E gráf fölött következő-csomópont formában [10] megadva, (2) a változtatások sorozata és (3) a 4.2.1. fejezetben bemutatott KGE algoritmus által készített G'_E Euler-gráf. Mivel az Euler-kör a következő-csomópont formában adott, ezért a karbantartása visszavezethető a TS feszítőfa karbantartására (amelynek élei a gyökércsomópont felé mutatnak). Az általánosság korlátozása nélkül feltesszük, hogy a TS feszítőfa kizárólag eredeti élekből épül fel.

Az algoritmus három fő logikai egységből áll:

1. Eljárások, amelyek gyakran előforduló lépéseket írnak le, amiket az Euler-kör frissítése során fel lehet használni. Ilyen lépések például:
   - Nem-feszítőél hozzáadása csomóponthoz
   - Feszítőél hozzáadása csomóponthoz
   - Nem-feszítőél elvevése csomóponttól
   - Feszítőél elvevése csomóponttól
   - A KGE algoritmus által használt zöld él kifejtés kezelése
2. Egy főfüggvény, amely meghatározza, hogy a különböző típusú változtatások hogyan érintik a következő-csomópont formában adott Euler-kört és az előző pontban felsorolt függvényeket felhasználva karbantartják azt.
3. Egy függvény, amelyik egy algráfot vesz fel és feszítőéleket készít a következő csomópontok számara: (i) olyan csomópontok, amelyek irányított útvonala a feszítőfában elvételre vagy megváltoztatásra került a frissítési műveletek során, (ii) a gráfhoz újonnan hozzávett csomópontok.

5. Az eredmények alkalmazhatósága

A javasolt inkrementális algoritmusok nagyon hatékonyak nagy, fejlődő rendszerspecifikációk esetén. A különböző megkötések a specifikáció állapotgépére vonatkozóan és a tesztkészletekben lévő különbözőségek (a tesztkészletek eltérő összhosszúsága és a különböző hibafelderítési képességek) jó skálázhatóságot biztosítanak a tesztmérnöknek. Az algoritmusok használhatóak az Agile munkafolyamatában, hiszen támogatják a fejlesztés alatt lévő rendszer lépésről-lépésre történő finomítását.

5.1. Az első téziscsoport lehetséges alkalmazási területei

A javasolt algoritmusok képesek mind a tesztesetgenerálás, mind a tesztelés felgyorsítására a legtöbb iteratív fejlesztési esetben. A javasolt megoldás bármilyen modell-alapú tesztelési eljárás során használható, mint például kommunikációs protokollok és szoftverek tesztelése. Az eljárás alkalmazható például webportálok tesztelésére, ahol az egyes oldalak megfeleltethetőek a véges automata állapotainak, a linkek az állapotátmeneteknek, a HTTP kérések paraméterei a bemeneti szimbólumoknak és az oldalokon megjelenített üzenetek a kimeneti szimbólumoknak. A disszertációban megvizsgáltam a SIP [17] protokoll regisztrációt leíró részének tesztelésére szolgáló ellenőrző szekvencia karbantartását a specifikáció lépésről lépésre történő fejlesztése során.

Az a tény, hogy az algoritmusok a specifikáció változtatásban nem érintett részeire a teszteseteket változatlanul hagyják, szintén nagy segítség a tesztmérnök számára. Például az ipari modell-alapú tesztelési szoftverek egyik vezető alkalmazása a Conformiq Designer [1] eltárolja a rendszer előző változatához készített teszthalmazokat és újrahasznosítja azokat, amennyiben az lehetséges[2]. A Conformiq Designer nem tűnik úgy, hogy gyorsabban generálná le a teszteseteket változó specifikáció esetére[3], viszont a tesztelő mérnököt megóvja attól, hogy egy teljesen új teszthalmaz struktúrát kelljen megtanulnia az új rendszerverzióra akkor, amikor a rendszer előző verziójához készített tesztesetek nagy része újrahasznosítható.

5.2. A második téziscsoport lehetséges alkalmazási területei

A modell-alapú teszteléssel foglalkozó esettanulmányok tipikusan csak állapot- és állapotátmenet lefedettséggel foglalkoznak [13]. Tehát a TT-method, amely a véges

[2] Ez az új funkció a Conformiq Qtronic 1.3-ban jelent meg 2008 április 4.-én, majdnem egy évvel a [C1] cikkünk után.

[3] Mivel a Conformiq Designer nem egy nyílt forráskódú szoftver, ezért belső működése nem ismert. Mindazonáltal úgy tűnik, hogy a nyers erő módszerével ellenőrzi, hogy egy régi teszteset alkalmazható-e a rendszer új verziójára.

automata összes állapotát és állapotátmenetét lefedi, megfelelő a legtöbb esetben.

Egy szoftver tesztelésénél, ahol az új kódot lefordítás után csak néhány alkalommal tesztelik, a hagyományos TT-method tesztgenerálása sokkal több időt emészt fel, mint a tesztszekvencia végrehajtása. A javasolt megoldás képes a tesztgenerálás idejét akár több nagyságrenddel felgyorsítani változó specifikációk esetében, tehát algoritmusaim jelentős mértékű erőforrást képesek megtakarítani az automata tesztelési folyamatban.

Fontos megjegyezni, hogy a javasolt inkrementális algoritmusok szintén alkalmazhatok nem véges automata alapú protokoll- és szoftvertesztelésre.

A disszertációban megvizsgáltam, hogy az algoritmusok hogyan tartják karban a SIP [17] protokoll regisztrációt megvalósító részének tesztszekvenciáját folyamatosan bővülő specifikáció esetén.

6. Köszönetnyilvánítás

Először is meg szeretném köszönni témavezetőmnek, Pap Zoltánnak a szakmai útmutatását és támogatását. Ki szeretném fejezni hálámat Csopaki Gyulának folyamatos támogatásáért és Kovács Gábornak a segítő észrevételeiért.

Köszönetet szeretnék mondani a Nagy Sebességű Hálózatok Laboratóriumának (High Speed Networks Laboratory (HSNLab)) az általuk biztosított technikai és pénzügyi háttérért. Nagyon hálás vagyok Szabó Róbertnek és Győri Erzsébetnek az adminisztratív segítségükért.

Szintén szeretném megköszöni szobatársaimnak, Babarczi Péternek és Erős Leventének az értékes takácsaikat és az együtt töltött munkaidőt. Hálával tartozom Adamis Gusztávnak és Hunyadi Leventének a segítségükért, valamint Bőhm Tamásnak a SIP regisztrációs témakörrel kapcsolatos beszélgetéseinkért.

Köszönöm barátaimnak, hogy felvidítottak, amikor szükségem volt rá.

Végül és nem utolsó sorban, szeretném megköszönni feleségemnek, Zsófinak a folytonos és feltétel nélküli támogatást.

7. Függelék – A tézisfüzetben használt jelölések

A	állapotpár gráf (segédgráf)
AFF_Q	q-érintett állapotok halmaza
AFF_Q^{Work}	AFF_Q munkahalmaza
AFF_Z	z-érintett állapotpárok halmaza
AFF_Z^{Work}	AFF_Z munkahalmaza
G	irányított gráf
G_E	G irányított gráf, kibővített, Euler gráfja
i_x	a véges automata egy bemeneti szimbóluma
I	bemeneti szimbólumok halmaza
$Impl$	implementáció véges automatája
M	specifikáció véges automatája
MOD	módosított állapotok halmaza
MOD_Z	módosított állapotpárok halmaza
n	állapotok száma a véges automatában
o_x	a véges automata egy kimeneti szimbóluma
O	kimeneti szimbólumok halmaza
p	bemeneti szimbólumok száma
Q	állapot lefedő halmaz
r	reset szimbólum
S	állapotok halmaza
s_k	a véges automata egy állapota
$< s_i, s_j >$	a véges automatában egy állapotpár
SF	feszítőerdő a segéd állapotpárgráfban (az élek a gyökércsomópont irányába mutatnak)
ST	feszítőfa a véges automatában (az állapotátmenetek a gyökércsomópontból kifele mutatnak)

t	a véges automata egy állapotátmenete
T	állapotátmenetek halmaza
TS	gráf feszítőfája (az élek a gyökércsomópont felé mutatnak)
x	bemeneti szekvencia
Z	elválasztó szekvenciák családja
Z_i	s_i állapot elválasztó halmaza
λ	kimeneti függvény: λ: $S \times I \to O$
δ	következő állapot függvény: $S \times I \to S$
Δ	a bemenet módosított részének és a kimenet változásban érintett részének a mérete
Δ_Q	$\lvert MOD \cup AFF_Q \rvert$
Δ_Z	$\lvert MOD_Z \cup AFF_Z \rvert$
ω_k	elemi változtatási lépés a véges automatában
Ω	a véges automatán végrehajtott változtatások sorozata

Hivatkozások

[1] Conformiq Designer. Model-based testing tool. `http://www.conformiq.com/` Accessed: 2015-05-25.

[2] LEMON. A C++ library for efficient modeling and optimization in networks. `http://lemon.cs.elte.hu` Accessed: 2015-05-25.

[3] E. J. Beltrami and L. D. Bodin. Networks and vehicle routing for municipal waste collection. *Networks*, 4(1):65–94, 1974.

[4] Gregor von Bochmann, Anindya Das, Rachida Dssouli, Martin Dubuc, Abderrazak Ghedamsi, and Gang Luo. Fault Models in Testing. In *Proceedings of the IFIP TC6/WG6.1 Fourth International Workshop on Protocol Test Systems IV*, pages 17–30, Amsterdam, The Netherlands, 1992. North-Holland Publishing Co.

[5] Gregor von Bochmann and Alexandre Petrenko. Protocol testing: review of methods and relevance for software testing. In *Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, ISSTA '94, pages 109–124, New York, NY, USA, 1994. ACM Press.

[6] Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner (Eds.). *Model-Based Testing of Reactive Systems*. Springer, 2005.

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

[8] Rita Dorofeeva, Khaled El-Fakih, Stephane Maag, Ana R. Cavalli, and Nina Yevtushenko. FSM-based conformance testing methods: A survey annotated with experimental evaluation. *Information and Software Technology*, 52(12):1286–1297.

[9] R. Dove. *Response Ability – The Language, Structure and Culture of the Agile Enterprise*. John Wiley and Sons, 2001.

[10] Jack Edmonds and Ellis L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5(1):88–124, 1973.

[11] H. A. Eiselt, M. Gendreau, and G. Laporte. Arc Routing Problems, Part I: The Chinese Postman Problem. *Operations Research*, 43(2):231–242, 1995.

[12] Khaled El-Fakih, Nina Yevtushenko, and Gregor von Bochmann. FSM-Based Incremental Conformance Testing Methods. *IEEE Transactions on Software Engineering*, 30(7):425–436, 2004.

[13] ETSI. Methods for Testing and Specification (MTS); Model-Based Testing (MBT); Application of MBT in ETSI case studies. Technical Report DTR/MTS 001411. http://docbox.etsi.org/MTS/MTS/07−Drafts/00141_MBT_CaseStudies/ Accessed: 2015-05-25.

[14] S. Fujiwara, G. v. Bochmann, F. Khendec, M. Amalou, and A. Ghedamsi. Test selection based on finite state model. *IEEE Transactions on Software Engineering*, 17(6):591–603, 1991.

[15] A. Gill. *Introduction to the theory of finite-state machines*. McGraw-Hill electronic sciences series. McGraw-Hill, 1962.

[16] James A. Highsmith. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Dorset House Publishing Co., Inc., New York, NY, USA, 2000.

[17] IETF. RFC 3261: SIP: Session Initiation Protocol, 2002. https://tools.ietf.org/html/rfc3261 Accessed: 2015-05-25.

[18] M. K. Kwan. Graphic programming using odd or even points. *Chinese Math*, (1):273–277, 1962.

[19] David Lee and Mihalis Yannakakis. Principles and Methods of Testing Finite State Machines – A Survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[20] Y. Lin and Y. C. Zhao. A new algorithm for the directed chinese postman problem. *Computers and Operations Research*, 15(6):577–584, 1988.

[21] Robert Cecil Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.

[22] S. Naito and M. Tsunoyama. Fault detection for sequential machines by transition-tours. In *Proceedings of the 11th IEEE Fault-Tolerant Computing Conference (FTCS 1981)*, pages 238–243. IEEE Computer Society Press, 1981.

[23] Zoltán Pap, Gyula Csopaki, and Sarolta Dibuz. On the Theory of Patching. In *Proceedings of the 3rd IEEE International Conference on Software Engineering and Formal Methods, SEFM*, pages 263–271, 2005.

[24] Alexandre Petrenko, Nina Yevtushenko, Alexandre Lebedev, and Anindya Das. Nondeterministic State Machines in Protocol Conformance Testing. In *Proceedings of the IFIP TC6/WG6.1 Sixth International Workshop on Protocol Test systems VI*, pages 363–378, 1994.

[25] G. Ramalingam and Thomas Reps. On the computational complexity of dynamic graph problems. *Theoretical Computer Science*, 158(1–2):233–277, 1996.

[26] P. G. Smith and D. G. Reinertsen. *Developing Products in Half the Time: New Rules, New Tools*. John Wiley and Sons, 1998.

[27] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2):146–160, June 1972.

[28] Mihalis Yannakakis and David Lee. Testing finite state machines: fault detection. In *Selected papers of the 23rd annual ACM symposium on Theory of computing*, pages 209–227, 1995.

Publikációk

Külföldön megjelent idegen nyelvű folyóiratcikkek

[J1] G. Kovács, **G. Á. Németh**, Z. Pap, and M. Subramaniam: Deriving compact test suites for telecommunication software using distance metrics. In *Journal of Communications Software and Systems*, Volume: 5 Number: 2. 2009. pp. 57–61.

[J2] **G. Á. Németh** and Z. Pap: Incremental Maintenance of Transition Tour. In *Fundamenta Informaticae*, Volume: 129 Number: 3. 2014. pp. 279–300.

Idegen nyelvű lektorált könyvfejezetek

[B1] **G. Á. Németh**: Protocol Operation. In *Advanced Communication Protocol Technologies: Solutions, Methods and Applications*. Hershey; New York: IGI Global, Information Science Reference, 2011. pp. 20–37.

[B2] G. Kovács, **G. Á. Németh** and Z. Pap: Convergence of fix and mobile networks. In *Advanced Communication Protocol Technologies: Solutions, Methods and Applications*. Hershey; New York: IGI Global, Information Science Reference, 2011. pp. 226–246.

Lecture Notes in Computer Science konferencia cikkek

[C1] Z. Pap, M. Subramaniam, G. Kovács and **G. Á. Németh**: A Bounded Incremental Test Generation Algorithm for Finite State Machines. In *Proc., Testcom/Fates 2007*, Tallin, Estonia, June 2007. pp. 244–259.

[C2] G. Kovács, **G. Á. Németh**, Mahadevan Subramaniam and Zoltán Pap: Optimal String Edit Distance Based Test Suite Reduction for SDL Specifications. In *Proc., SDL 2009*, Bochum, Germany, September 2009. pp. 82–97.

[C3] G. Adamis, A. Wu-Hen-Chang, **G. Á. Németh**, L. Erős, G. Kovács: Data Flow Testing in TTCN-3 with a Relational Database Schema. In *Proc., SDL 2013*, Montreal, Canada, June 2013. pp. 1–18.

Egyéb nemzetközi konferencia-kiadványban megjelent cikkek

[C4] G. Kovács, **G. Á. Németh**, Z. Pap, and M. Subramaniam: Deriving compact test suites for telecommunication software using distance metrics. In *Proc., SoftCOM 2008*, Split-Dubrovnik, Croatia, September 2008. pp. 394–398.

[C5] G. Németh and **G. Á. Németh**: A Generic Model for Advanced Networks Handling Imprecise Information. In *Proc., SoftCOM 2009*, Hvar, Croatia, September 2009. pp. 116–120.

[C6] **G. Á. Németh**, Z. Pap and G. Kovács: The Incremental Maintenance of a Checking Sequence. In *Proc., AACS'13*, Budapest, Hungary, June 2013. pp. 58–69.

Magyar nyelvű konferencia publikáció

[C7] **G. Á. Németh** and Z. Pap: Inkrementális tesztgenerálás. In *Tavaszi Szél 2012*, Győr, Hungary, 2012. pp. 442–448.

Független hivatkozások

[C1-1] A. Simão and A. Petrenko: Fault coverage-driven incremental test generation. In *Computer Journal* Volume 53, Issue 9, November 2010, pp. 1508–1522

[C1-2] A. Simão and A. Petrenko: Incremental Test Generation Guided by Fault Coverage: Technical Report. 2009. *http://www.crim.ca/Publications/2009/documents/plein_texte/ASD_SimA_al_0903--01.pdf*. Accessed: 2015-05-25. ISBN-13: 978-2-89522-116-6. ISBN-10: 2-89522-116-2.

[C1-3] E. Uzuncaova, S. Khurshid, and D. Batory: Incremental Test Generation for Software Product Lines. In *IEEE Transactions on Software Engineering*, Volume 36, Number 3, May/June 2010, pp. 309–322

[C1-4] E. Uzuncaova: Efficient Specification-based Testing Using Incremental Techniques. Dissertation. 2008. *http://www.library.utexas.edu/etd/d/2008/uzuncaovae51392/uzuncaovae51392.pdf*. Accessed: 2015-05-25.

[C1-5] H. Luo: On Distributed Multi-Point Concurrent Test System and Its Implementation. In *Social Informatics and Telecommunications Engineering* 4: 125–129 (2009)

[C1-6] L. Erős: Performance Testing and Performance Improvement Methods for Communicating Systems. Dissertation. 2013. *http://www.omikk.bme.hu/collections/phd/Villamosmernoki_es_Informatikai_Kar/2013/Eros_Levente/ertekezes.pdf*. Accessed: 2015-05-25.

[C2-1] Y. Ledru, A. Petrenko, S. Boroday and N. Mandran: Prioritizing test cases with string distances In *Automated Software Engineering*, Volume 19, Issue 1, March 2012, pp. 65–95