Budapesti Műszaki és Gazdaságtudományi Egyetem

Távközlési és Médiainformatikai Tanszék

Test Data Generation

Pro Progressio alapítvány számára készítette:

Dr. Csopaki Gyula

tudományos főmunkatárs

Erős Levente

egyetemi tanársegéd

Budapest

2012. június

Összegzés

A modellalapú tesztelés az elmúlt években egyre népszerűbbé vált. Ennek során a teszt tervezője nem a konkrét teszteseteket, hanem a tesztelendő rendszer absztrakt modelljét készíti el, amelyből tesztgeneráló eljárásokkal, a tervező által megadott további tesztgenerálási paraméterek figyelembe vételével, automatikusan állnak elő a tesztesetek. Miután a korábbiakban kidolgoztuk a tesztelt rendszer, a teszttervezében jártas mérnökök számára jól ismert TTCN-3 tesztleíró nyelvvel történő leírásának módszerét – jelen félév célja az volt, hogy olyan eljárásokat dolgozzunk ki, amelyek segítségével a tesztelt rendszernek, annak egy adott végrehajtási állapotban megfelelő bemeneti tesztadatokat küldhetünk. Ezen feladat jelen félévre vonatkozó része felölelte egyrészt olyan eljárások elméleti alapjainak lefektetését tűzte ki célul, amelyek segítségével olyan tesztadatokat generálhatunk, amelyekkel a rendszer egy megadott állapotátmenete elérhető, tehát tesztelhető, másrészt olyan eljárások kidolgozását és implementálását, amelyek segítségével a tesztelt rendszer egy adott állapotátmenet-sorozata mentén lekövethetjük a rendszer belső változóinak változását. A rendszer reprezentálására a kiterjesztett véges állapotautomata modellt használtuk, amelyet hatékonysági megfontolásokból relációs adatbázissémára transzformálva tároltunk.

1. Bevezető

Feladatunk a félév során tesztadat-generáló eljárások kidolgozása volt kiterjesztett véges állapotautomataként (EFSM) tárolt tesztmodellek mentén történő modellalapú tesztelésre [EFSM, Utting, MBT]. Ez részfeladatként egyrészt magában foglalta olyan eljárások alapjainak kidolgozását, amelyek egy adott tranzíció a modell kezdőállapotától az adott tranzícióig olyan tesztadatokat generálnak, amelyek a tesztelt rendszernek való kiküldése után a kijelölt tranzíció bejárható lesz. Másrészt olyan eljárások kidolgozása volt a cél, amelyek a tesztmodellt relációs adatbázissémában tárolják, és adott tranzíciósorozat végrehajtása esetén lekövetik a rendszer belső változóinak transzformációit és állapotváltásait [Relációs]. A továbbiakban e két részfeladatot fejtjük ki bővebben.

2. Adott állapotátmenet elérése

Az itt bemutatott eljárás megállapítja, hogy egy adott tranzíció bejárásához a teszternek milyen konkrét paraméterekkel kell gerjesztenie a tesztelt rendszer mely tranzícióit. A feladatot egy leegyszerűsített változatban oldottuk meg, ahol a teszter nem paraméterezett TTCN-3 template-eket küld a tesztelt rendszernek, hanem egyszerű paramétereket, amelyeket belső változóknak adunk értékül [TTCN-3]. Ez könnyen kibővíthető arra a változatra, ahol a teszter paraméterezett template-eket küld a tesztelt rendszernek.

A módszer lényege az, hogy a modellben szereplő valamennyi értékadást és predikátumot egységesítve, logikai kifejezésekké alakítunk, és egy adott tranzíció bejárásához szükséges, paraméterértékeket egy, ezen logikai kifejezések ÉS kapcsolataként előálló kifejezést kielégítő értékekként kapjuk meg.

Az eljárás bemenete tehát egy t állapotátmenet, amelyet a tesztelt rendszer kezdőállapotából elindulva be kell járnunk. A fent említett logikai kifejezést a következőképpen írjuk fel:

Karban tartunk egy snapshot-okat és egy tranzíciókat (a teszt során potenciálisan megtörténő tranzíciókat) tároló táblát. Előbbinek Snapshots, utóbbinak Transitions a neve. A Snapshots tábla attribútumai (snapshot, state), tehát a snapshot azonosítóját és a snapshot-ban aktuális állapotot tárolja, míg a Transitions tábla attribútumai (snapshot1, snapshot2, transition, condition), ez tehát a snapshot-ok közötti állapotátmeneteket és az egyes állapotátmenetek bejárhatóságához szükséges logikai kifejezéseket tárolja. A továbbiakban állapotátmenet alatt két állapot, míg tranzíció alatt két snapshot közötti átmenetet értünk.

A táblák mellett karban tartunk a belső változókhoz és paraméterekhez egy-egy számlálót, amelyek a változók és paraméterek aktuális verziószámát követik. Kezdetben valamennyi változó és paraméter verziója 0.

Felveszünk első körben egy 0. snapshot-ot a Snapshots táblába, amely a bejárandó tranzíció célállapotában készül. Ezután felveszünk pontosan egy, a 0. snapshot-ból kiinduló (snapshot0) tranzíciót a Transitions táblába, amely transition attribútuma szigorúan a bejárandó állapotátmenet. Innentől kezdve adódik a cél-snapshot (snapshot1), a logikai feltételt (condition) pedig több logikai kifejezés konjunkciója lesz. Ezen logikai kifejezések a következőképpen alakulnak ki.

Amennyiben az állapotátmenethez tartozik predikátum, úgy a predikátumban szereplő belső változók aktuális verziói kerülnek a logikai feltételbe. Ha tehát a predikátum v=5, és a v belső változó jelenlegi verziója 3, akkor a logikai feltételbe v3=5 kerül.

Amennyiben az állapotátmenethez tartozik akció (értékadás), úgy a következőképpen járunk el: Az értékadásból egy egyenlőségi logikai feltételt készítünk úgy, hogy az értékadás bal oldalán álló változó jelenlegi verziójával fog szerepelni a logikai feltétel bal oldalán, ezután léptetjük az értékadás bal oldalán szereplő belső változó verziószámát, a jobb oldalon szereplő változók pedig szintén jelenlegi verziójukkal fognak szerepelni a logikai feltétel jobb oldalán. Ha tehát az akció v:=v+u, és a v változó jelenlegi verziója 3, az u változó jelenlegi verziója pedig 5, úgy a logikai kifejezés v5=v6+u3 lesz.

Amennyiben az állapotátmenethez olyan bemeneti esemény tartozik, amely során valamely belső változó egy paramétertől kap értéket, úgy úgy járunk el, mint egy sima értékadásnál, a paraméter és a változó verzióját megfelelően kezelve.

A fentebb írtaknak megfelelően abban az esetben, ha predikátum és értékadás is tartozik az állapotátmenethez, ezek konjunkciója lesz az állapotátmenethez tartozó logikai kifejezés.

Az első tranzíció felvétele a rendszert egy új snapshot-ba és bizonyos esetekben új állapotba viszi, amely a tranzícióhoz tartozó állapotátmenet forrásállapota. Ennek megfelelően felvesszük az új snapshot-ot a Snapshots táblába. Innentől kezdve az algoritmus lényegében a tesztmodell kezdőállapotig, az élek irányításával szemben történő szélességi bejárása, amelyet a Snapshots és a Transitions táblák karbantartásával végzünk el, a fent leírt módon. Tehát valamennyi új snapshot-ból a rendszer hozzá tartozó aktuális állapotának bemenő tranzícióin folytatjuk a bejárást, új állapotokat veszünk fel a Snapshots táblába, és ezt addig folytatjuk, amíg az elsőként felvett (bejárandó) tranzícióhoz tartozó állapotátmenet végállapotába (tehát a rendszer elsőként felvett snapshot-hoz tartozó állapotába) vagy a rendszer kezdőállapotába nem érünk.

Amennyiben az elsőként felvett snapshot-hoz tartozó állapotba érünk, onnan csak a bejárandó állapotátmenettől eltérő állapotátmeneteken haladunk tovább. Amennyiben a kezdőállapotba érünk, ott vesszük azon logikai kifejezéseket, amelyek a bejárás azon ágán szereplelnek, amelyen a bejárandó állapotátmenet végállapotából a kezdőállapotba értünk, majd ezen logikai kifejezéseket konjunkcióba fűzve megvizsgáljuk, hogy a kapott kifejezés kielégíthető-e. Amennyiben igen, úgy az adott ágon vett állapotátmenetek egy fordított sorrendje a kiértékeléssel kapott paraméterértékekkel kiegészítve a kijelölt állapotátmenet bejárásához fog vezetni. Amennyiben nem, a szélességi bejárást folytatjuk.

A fent leírt eljárás működésének bemutatására tekintsünk egy példát. A tesztmodellt leíró EFSM a következőképpen adható meg az állapotátmeneteivel (<állapotátmenet>: forrásállapot->predikátum, bemenet,akció,kimenet->célállapot formában):

```
1:0->-,t1,v,-:=0->1
2:1->-,t1,v:=v+1,->2
3:2->v=2,-,-,t2->0
4:2->v!=2,-,-,t3->1
```

A példában a cél a 3-as állapotátmenet bejárása. Ennek és a fentieknek megfelelően a Snapshots és Transitions táblák a következőképpen fognak kinézni:

Snapshots

snapshot	state
0	0
1	2
2	1
3	2
4	0
5	1
6	1

7	0

Transitions

snapshot1	snapshot2	transition	condition
0	1	3	v0=2
1	2	2	v0=v1+1
2	3	4	v1!=2
2	4	1	v1=0
3	5	2	v1=v2+1
5	6	4	v2!=2
5	7	1	v2=0

Megoldásként az 1-es tranzíció célállapotából az 1,2,4,2,3 állapotátmenet-sorozattal juthatunk el visszafelé haladva a kezdőállapotba úgy, hogy az ezen út mentén kapott v2=0 ÉS v1=v2+1 ÉS v1!=2 ÉS v0=v1+1 ÉS v0=2 logikai kifejezés kielégíthető legyen. Ezek szerint tehát az állapotátmenet bejárásához a 3,2,4,2,1 állapotátmenet-sorozat adódik.

Az eljárás hibája, hogy exponenciális lépésszámú. Konkrét vizsgálatoknak kell eldöntenie, hogy mekkora méretű tesztmodellnél mennyire hatékonyan alkalmazható. Nagyobb modellek esetén a keresési teret le kell szűkítenünk. Az irodalomban fellelhető módszerek esetén több helyen klasszifikálják az egyes tranzíciókat annak megfelelően, hogy egy adott változó a tranzíció akciójában szerepel bal oldalon vagy jobb oldalon, hogy predikátumában szerepel, vagy bementő paraméterében szerepel. Ezt a klasszifikációt felhasználhatjuk arra, hogy hosszabb tesztszekvenciát eredményező, de kevesebb lépést igénylő módszerekkel oldjuk meg a feladatot.

3. Tesztmodell és tesztadatok tárolása relációs adatbázisban

A tesztadatok típusait tároló táblák (relációs sémák) a következők:

- datatype(name, typetype)
- compositefield(parenttypename, fieldname, ttcntype, fieldnumber)

A datatype tábla az adattípusokat tárolja. A name attribútum az adattípus nevét a typetype pedig annak típusát tárolja (record, set, recordof, setof, simple).

A compositefield tábla az összetett típusok egyes mezőit tárolja, a parenttypename attribútum értékeként szerepel az összetett típus neve, a fieldname alatt az összetett típus egy mezőneve, a ttcntype alatt a parenttypename attribútumban megnevezett típus fieldname attribútumban

megnevezett mezőjének típusa, rekord típusok esetén pedig a fieldnumber attribútumban tároljuk az adott mező sorszámát az összetett típuson belül. Tömb jellegű típusok esetén (record of, field of), a fieldname attribútum üresen marad.

Ezeket a táblákat a tesztmodell belső reprezentációjának alapján töltjük fel.

A modell belső változóinak és az adatsablonok értékeinek vaamint típusainak tárolására használt táblák a következők:

- template (templatename, ttcntype)
- templatevalue (field, value)
- variable (variablename, ttcntype)
- variablevalue (field, value)

A továbbiakban a változók tárolására használt táblapárost mutatjuk be. Ezek lényegében ugyanúgy működnek, mint az adatsablonokat tároló táblák. Ahol az adatsablonokat tároló táblákat külön említjük, csak ott térnek el a változókat tároló tábláktól.

A template illetve variable táblákban tároljuk az egyes adatsablonok és változók neveit és típusait, rendre a templatename valamint ttcntype attribútumok alatt.

A templatevalue és az ezzel múködésében megegyező variablevalue táblában az egyes adatsablonok és változók illetve összetett típusú sablonok és változók esetén, azok mezőinek értékeit tároljuk a következő módon:

Amennyiben a változó egyszerű típusú, úgy a variablevalue tábla field attribútumában a változó neve szerepel, míg a value attribútum alatt annak értéke.

Amennyiben a változó összetett típusú, úgy record illetve set típus esetén a variablevalue tábla field attribútumában a változó azon mezőinek nevei szerepelnek, amelyek típusa már egyszerű. A value attribútumban pedig ezen egyszerű típusú mezők értékei szerepelnek. A mezők neveit teljes elérési útjukkal kell megadni, azaz <változónév>.<1. mezőnév>.[...].<n. mezőnév> formában, ahol az n. mezőnév már egyszerű típusú mezőre hivatkozik. Ha tehát az a nevű változó b nevű mezője még összetett, de a b nevű mező c nevű mezője már egyszerű, és értéke 0, akkor a táblában az (a.b.c, 0) bejegyzés fog állni.

Amennyiben a változó összetett tömb típusú, azaz record of vagy set of típusú, úgy a fieldname attirbútum alatt az egyes elemek neveit külön-külön hivatkozva tároljuk, a TTCN-3-ban megszokott szögletes zárójeles jelöléssel. Ha tehát az a nevű, record of típusú változó két elemet tartalmaz,

amelyek közül az első értéke "a", a második értéke pedig "b", akkor a táblában a következő bejegyzések fognak szerepelni: (a[1], "a"), (a[2], "b"). Amennyiben az adott változó üres, nem tároljuk a variablevalue vektorban.

A teszmodell belső változóinak nevét és típusát a tesztmodell belső, objektumorientált reprezentációjából vesszük. Kezdeti értéküknek egész típus esetén 0-t, lebegőpontos típus esetén 0.0-t, karakterfüzér típus esetén ""-t, logikai típus esetén false-ot adunk meg, a variablevalue tábla feltöltése tehát megfelel a változók deklarációjának, amely során a változók az adott típusra jellemző alapértelmezett értéket kapják.

Az adatsablonok értékeit tároló táblák feltöltése során jelenleg valamennyi típusra egy adatsablon jön létre, egy automatikusan generált névvel, amely mezőinek értékei dummy értékek, tehát egész típus esetén 0, lebegőpontos típus esetén 0.0, karakterfüzér típus esetén "", logikai típus esetén false.

4. A kiterjesztett állapotgép reprezentálása relációs adatbázisban

A tesztmodell, mint véges állapotgép adatbázisban való tárolásához a következő táblákat hoztuk létre:

- state(name, label)
- transition(name, fromstate, tostate, input, predicate)
- action(transition, serial, type, action)
- assignment(transition, variable)

Ezek leírása a következő:

A state táblában tároljuk az állapogép állapotait. A name attribútum az egyes állapotok azonosítója. A label attribútum az állapot nevét tárolja, amely a belső modellben tárolt, az állapotnak megfelelő Node típusú attribútum label attribútumában található.

A transition táblában az állapotátmneteket tároljuk. A name attribútum az egyes állapotátmenetek azonosítója,. A fromstate attribútum az állapotátmenet kezdőállapotának azonosítója, míg a tostate attribútum a végállapot azonosítója. Az input attribútumban tároljuk az állapotátmenetet kiváltó bemeneti eseményt szövegesen (amely lehet egy időzítő lejárta vagy egy üzenet fogadása valamely porton). A predicate attribútum az állapotátmenetet kiváltó predikátumot tárolja, szövegesen (ez tehát egy if-feltételnek felel meg). Amennyiben a tárolt állapotátmenetet fogadott üzenet vagy időzítő lejárta váltja ki, akkor a predicate attribútum üres lesz. Amennyiben a tárolt állapotátmenetet egy logikai feltétel teljesülése váltja ki, akkor az input attribútum üres lesz. Amennyiben a tárolt állapotátmenet egyik előző típusba sem esik, tehát automatikusan végrehajtódik (ilyen állapotátmenet az első

állapotot megelőző inicializáló állapotátmenet), akkor mind az input, mind a predicate attribútumok üresek lesznek.

Az action tábla az egyes állapotátmenetek során végrehajtott akciókat, tehát a belső változókon és időzítőkön végrehajtott transzformációkat valamint üzenetküldéseket tároljuk. A transition attribútum arra az állapotátmenetre hivatkozik, amely során az adott akciót végrehajtjuk. A serial attribútum azt tárolja, hogy az adott akció sorrendben a hanyadik a hozzá tartozó állapotátmeneten. A type attribútum azt adja meg, hogy az adott akció üzenetküldés, timer-művelet vagy változónak történő értékadás. Az action attribútum magát az akciót tárolja, szövegesen.

Az assignment táblában tároljuk azt, hogy az egyes bejövő üzeneteket a tranzíciók mely belső változókba tárolták el. A transition attribútum az állapotátmenetet azonosító sorszám, míg a variable attribútum arra a változóra hivatkozik, amelynek az állapotátmenet által fogadott üzenetet értékül adjuk.

5. A belső változók változásainak lekövetése

A változók lekövetése során megoldandó feladat az, hogy a végrehajtandó állapotátmenet azonosítójának mint bemeneti paraméternek alapján a relációs adatmodellben érvényesítsük az állapotátmenet végrehajtásának következményeit, azaz frissítsük az állapotátmenet által módosított változók értékeit, és a modellt abba az állapotba vigyük, amelybe a végrehajtandó állapotátmeneten fogadott bemeneti esemény/a végrehajtandó állapotátmenet időzítőjének lejárta vinné. Ennek során először a paraméterként átadott állapotátmenetet kell végrehajtani, majd amennyiben az egy predikátum-csomópontra mutat, amelyből tehát predikátumélek indulnak ki, ezek predikátumainak vizsgálatával a rendszert olyan állapotba kell továbbvinni, amelyet a predikátumélek meghatároznak. (Természetesen predikátum-csomópontot is követhet predikátum-csomópont, így a predikátumélek részfájának bejárását egészen addig kell folytatni, amíg a rendszer egy olyan csomópontba nem ér, amelyből kiinduló állapotátmeneteket már bemeneti esemény vagy időzítő vált ki.) A változókövetést a következőképpen valósítottuk meg:

A modell mindenkori belső állapotát egy globális változó, az egész típusú state tárolja.

A változók, adatsablonok és magának a kiterjesztett véges állapotgépnek relációs adatbázissá való transzformációját követően elsőként fel kell dolgozni azokat a tranzíciókat, amelyeknek se predikátumuk, sem kiváltó bemeneti eseményük nincs. Ilyen állapotátmenetek azok, amelyek a belső változók inicializálását végzik például a végrehajtás elején. A bemeneti esemény és predikátum nélküli tranzíciók feldolgozása után a rendszer első stabil állapotába jutottunk (ahova a tesztelt rendszer is eljut), így véget ér a tesztadat-generátor inicializálása.

A változókat visszaadó metódus leírása után nézzük meg, hogyan történik egy adott tranzíció akcióinak érvényesítése az adatbázison. Első lépésben lekérdezzük az adatbázisból azon akciók halmazát, amelyek egyrészt változókat transzformálnak, másrészt pedig a bemeneti paraméterként kapott állapotátmenethez tartoznak.

A lekérdezés eredményeként előálló akciókat ezek után egyenként dolgozzuk fel. Az adott akció végrehajtásához érvényesítjük az akció hatását az adatbázison, majd megváltoztatjuk az aktuális állapotot. Az állapotváltást követően végül feldolgozzuk az új állapotból, mint gyökérből kiinduló, predikátumélekből álló esetleges részfát (azaz a rendszert az új állapot elején álló if-then-else szerkezetsorozat által meghatározott állapoba helyezzük).

6. Jövőbeni tervek

A közeljövőre két megvalósítandó célt tűztünk ki.

Egyrészt olyan eljárást dolgozunk ki, amely a teszt végrehajtásához egy kezdeti bemeneti adatot generál, majd az itt ismertetett változókövetési eljárás segítségével eldönti, hogy a generált adatot bemenetként kapva a tesztelt rendszer mely állapotba fog jutni. Innentől kezdve pedig állapotról állapotra lépve a fent leírtak szerint működik. Ez tehát egy mohó tesztadat-generáló eljárás.

Másik célunk, hogy a 2. fejezetben leírt eljárást finomítva implementáljunk egy olyan eljárást, amely adott tranzíció elérését valósítja meg megfelelő tesztadatok generálásával.

Irodalom

[EFSM] Lee, D. and Yannakakis, M. Principles and Methods of Testing Finite State Machines – A Survey *In Proceedings of the IEEE vol. 84 issue 8, pp 1090–1123*, 1996.

[MBT] Broy, M.; Jonsson, B.; Katoen, J.-P.; Leucker, M.; Pretschner, A.: *Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science)*, Springer-Verlag New York, Inc., 2005.

[Relációs] Ullman, Jeffrey D. and Garcia-Molina, Hector and Widom, Jennifer: *Database Systems: The Complete Book*, Prentice Hall PTR, 2001.

[TTCN-3] ETSI ES 201 873-1 ver. 4.2.1: *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language*, 2010.

[Utting] Utting, Mark and Legeard, Bruno: *Practical Model-Based Testing: A Tools Approach*, Morgan Kaufmann Publishers Inc., 2007.