AUTOMATIKUS TESZTGENERÁLÁS MODELL ELLENŐRZŐVEL

Micskei Zoltán

Abstract. Testing is an essential, but time and resource consuming activity in the software development process. Generating a short, but effective test suite usually needs a lot of manual work and expert knowledge. In a model-based process, among other subtasks, test construction and test execution can also be partially automated. Based on the method suggested in [1], this paper presents a *test generator tool*, which can be used for test set generation in the development process of event-driven embedded systems. For a selected test coverage criterion and from the UML statechart model of the system the program generates test cases using the SPIN model checker. The test generator tool supports currently the test sequence construction on the basis of the "all states" and "all transitions" coverage criteria. The necessary model transformation [2] and requirement generation steps are performed automatically. The configuration of the model checker in the case of test generation, namely the settings required for constructing a short and minimal test suite, differs from the usual needs of classic model checking problems. The paper analyzes the possible settings of the model checker SPIN by measuring the efficiency of test construction in the case of different real-life statechart models, and introduces an optimized setting for test generation. The test generator is extended for real-time applications, in this case the model is available in the form of timed automata, and the model checker to be used is Uppaal [3].

A tesztelés a szoftverfejlesztés lényeges, ámde idő- és erőforrás-igényes része. Rövid, ugyanakkor hatékony tesztek generálása komoly tudást és rengeteg emberi munkát igényel. Modell-alapú fejlesztés esetén viszont, egyéb részfeladatok mellett, a tesztek előállítása és végrehajtása is részben automatizálható. A dolgozat bemutat egy *tesztgeneráló eszközt*, ami az [1] cikkben javasolt módszer alapján eseményvezérelt rendszerekhez készít tesztsorozatokat. A rendszer UML állapottérkép modelljéből kiindulva egy megadott fedési kritérium alapján a SPIN modell ellenőrző eszközt használja a teszt generáláshoz. Jelenleg a „minden állapot" és a „minden tranzíció" fedési kritériumok használhatóak, de az eszköz könnyedén bővíthető más, az irodalomban javasolt kritériummal is. A modell ellenőrző használatához szükséges modell transzformáció [2] és a követelmények temporális logikával való megfogalmazása automatikus. A modell ellenőrző azonban tesztgenerálás esetén más beállításokat igényel, mint a klasszikus verifikációs feladatoknál. Elég egy darab ellenpéldát találni, de az lehetőleg minél rövidebb legyen. A dolgozat megvizsgálja a SPIN modell ellenőrző különböző konfigurációs lehetőségeit, és valós életbeli állapottérképeken elvégzett mérések segítségével meghatároz egy optimalizált beállítást. Az eszköz kiterjeszthető valós-idejű alkalmazásokhoz is, ilyenkor az Uppaal [3] modell ellenőrzőt és időzített automatákat használ a tesztek generálásához.

1. Bevezető

A tesztelés során a tesztelendő rendszert bemeneti adatokkal látjuk el, megfigyeljük a kimeneteit, és ebből próbálunk arra következtetni, hogy a helyes, elvárt működést valósítja-e meg. A tesztsorozatok (bemenet – elvárt kimenet párok) előállításához segítséget nyújtanak a fedési kritériumok, melyek meghatároznak egy tesztelendő követelményhalmazt (pl. minden kódsort hajtsunk végre). Azonban még így is a klasszikus, implementáció alapú tesztelés rendkívül időigényes folyamat, sok benne a hibalehetőség. Erre a problémára nyújthatnak megoldást a *specifikáció alapú* tesztelési módszerek. Ilyenkor a rendszer valamilyen magas szintű, általában félformális modelljét hívjuk segítségül a tesztelés következő feladatainak automatizálásához:

- *Teszt orákulum előállítása*: a modellt arra használjuk fel, hogy egy bemeneti sorozathoz megmondja, megjósolja (innen a módszer neve) a rendszer kimenetét.
- *Fedés meghatározása:* modell alapú tesztelési módszerekkel megoldható, hogy meghatározzuk egy már meglévő tesztkészlet fedését egy adott fedési kritériumhoz [4].
- *Konformancia tesztelés*: Az implementáció és a modell szimultán futtatásával vizsgálható az, hogy a megvalósítás mennyire igazodik a specifikációhoz.
- *Tesztgenerálás*: A tesztelés talán legnehezebb részét, a tesztgenerálást is lehet automatizálni. A dolgozat további része ezzel a feladattal foglalkozik.

2. Tesztgenerálási módszerek és eszközök az irodalomban

A Pennsylvania Egyetem munkatársai által kidolgozott módszerben állapottérképes leírásokat transzformáltak egy modell ellenőrző bemeneti nyelvére, majd teszteket generáltak hozzá [1]. A javasolt megvalósítással kapcsolatban azonban több probléma is felmerül, például az állapottérkép szemantikájának formalizálása nem teljes, és nem készült eszköz sem a módszer demonstrálására.

Az AGEDIS [5] egy Európai Unió által támogatott kutatási projekt, melynek célja a szoftverek minőségének javítása a tesztelés automatizálásával. Módszerükben egy UML Profile-ban definiált jelölésekkel lehet annotálni a modell bizonyos elemeit, és ezekhez generál azután a rendszer teszteket, majd az absztrakt tesztesetek részeit az implementációnak megfeleltetve azok végrehajtását is elvégzi.

P. E. Black és munkatársai a mutációs analízis technikát kombinálják modell ellenőrző használatával [4]. Gargantini és társai cikkükben [6] bemutatnak egy módszert tesztek generálására ASM modellekből SPIN modell ellenőrző segítségével. Elkészítették a tesztgenerátor program prototípusát, és beszámolnak egy kis modellen végzett sikeres próbáról is. A fentebb bemutatott eszközökön kívül természetesen léteznek még tesztgeneráló programok. Ezeket tekinti át [7], részletezve az egyes szoftverek elérhetőségét, a használt formalizmusokat, a be- és kimeneti formátumokat.

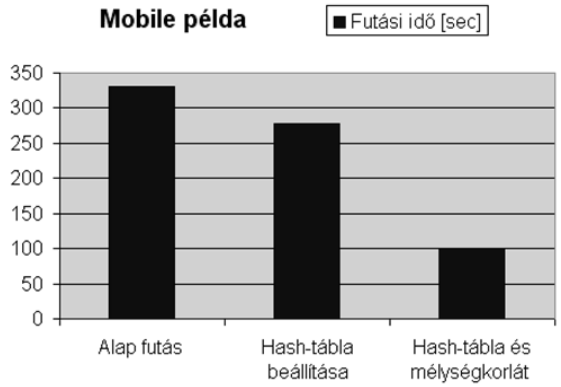
Láthatjuk tehát, hogy az automatikus tesztgenerálás területén aktív kutatás folyik, sok a témával kapcsolatos friss cikk, de a módszerek megvalósíthatóságával kapcsolatban még sok a nyitott kérdés.

3. Tesztgenerálási tapasztalatok

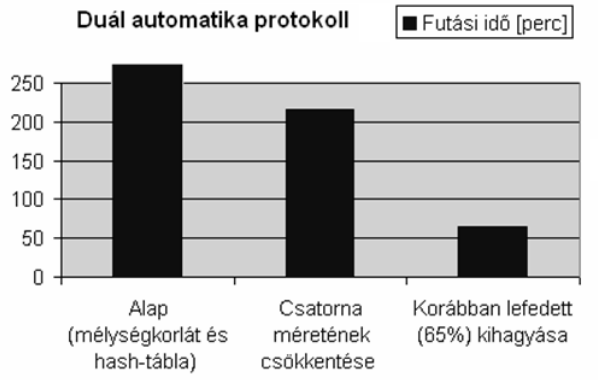
Az általam készített eszköz működésének főbb lépései a következők:

1. Az állapottérképet hierarchikus automatákká, majd a SPIN bemeneti nyelvére transzformálja.
2. A megadott fedési kritériumhoz (pl. minden állapothoz generáljunk tesztet) tartozó teszt követelményeket lineáris temporális logikai formulákká alakítja.
3. Mindegyik formulához egy olyan futást keres, ami teljesíti azt. Ezért a negáltját ellenőrizteti, és ha egy negált formula nem teljesül, a kiadódó ellenpélda éppen az eredeti formulát kielégítő teszt sorozat lesz.
4. A SPIN kimenetét feldolgozza, és bemeneti esemény – kimeneti akció párokból álló sorozatot állít elő belőle, ami már közvetlenül használható tesztként.

Az eszköz alkalmazhatóságát két példán keresztül vizsgáltam. Az első egy mobiltelefon állapottérképe (10 állapot, 21 átmenet). Jelen dolgozatban csak a főbb eredményeket ismertetem, a részletes mérési eredmények megtalálhatóak [8]-ban. A mérések során a szélességi bejárás, és az iteratív módon való rövidebb ellenpélda keresése módszerei nem bizonyultak alkalmasnak, túl nagy volt a memóriaigényük. Mint az az 1. ábrán is látszik, a mélységi bejárásnál használt megfelelő *mélységkorlát* megadása eredményezett hatékony futást.



1. ábra mobil példa



2. ábra duál automatika példa

A mobiltelefon mintapéldánál szerzett tapasztalatokat felhasználva egy valódi ipari alkalmazáson is teszteltem az eszközt. Ez egy bit szinkronizációs (úgynevezett duál automatika) protokoll állapottérképe volt, mely 5 objektumból, a hozzájuk tartozó eseménysorokból, 31 állapotból és 174 átmenetből áll. A modell nagy mérete miatt állapottömörítési eljárásokat kellett alkalmazni. A SPIN közelítő eredményt adó *bitstate hashing* eljárása bizonyult a legalkalmasabbnak. Ennek során az állapotokat a memória egy-egy bitjén tároljuk, így azonban az állapot összevonás miatt az állapottér egy részét nem járjuk be. A tesztgenerálás során viszont elég, ha a tucatnyi lehetséges teszt közül akár egyet is megtalálunk, a bitstate hashingnek köszönhetően azonban ez a folyamat gyors lesz. Ha a modell méretét tovább szűkítjük, a szükséges futási idő még jobban csökkenthető, például az objektumokhoz tartozó eseménysorok méretének kisebbre állításával. A bejárandó állapottér így jóval kisebb lesz, a kapott teszt azonban az eredeti modellben is egy helyes tesztsorozat lesz. Végül tovább

csökkenthető az eszköz futási ideje azzal, hogy a program kihagyja azokat a tesztkövetelményeket, amiket egy korábban generált teszteset már lefed (2. ábra).

4. Bővítési lehetőségek, értékelés

Az eszköz hatékonyságát nagyban befolyásolja, hogy hogyan választja ki a kritériumokat lefedő tesztesetekből a végleges tesztkészletet. A minimális hosszúságú vagy minimális elemszámú tesztkészlet kiválasztása NP-teljes feladat [9], azonban további heurisztikák alkalmazása (pl. mélyen fekvő állapok előre vétele) segíthet a megközelítésében.

Beágyazott rendszerek egy fontos részhalmazát képezik a *valós-idejű rendszerek*. Az eszköz kiterjeszthető ezekre is, ilyenkor a SPIN helyett az Uppaal modell ellenőrzőt használjuk. A mobiltelefon mintapéldának elkészítettem az Uppaal-os modellét, és ehhez sikeresen generáltam minden állapotot lefedő tesztsorozatot. Az automatizálás akadálya, hogy az Uppaal, egyelőre még nem támogatja a hierarchikus automatákat [10], így a hierarchiát kisimító transzformációt el kell készíteni.

Dolgozatomban bemutattam a modell alapú tesztgenerálás lehetőségeit, és egy konkrét eszközt, mely modell ellenőrzőt használ a tesztesetek előállítására. A mérések során használt példákkal demonstráltam a módszer alkalmazhatóságát.

Irodalomjegyzék

1. Hyoung Seok Hong, Insup Lee, Oleg Sokolsky, Sung Deok Cha: Automatic Test Generation from Statecharts Using Model Checking, Technical Report MS-CIS-01-07, Feb 2001.
2. D. Latella et al: Automatic Verification of a Behavioural Subset of UML Statechart Diagrams Using the SPIN Model-checker. Formal Aspects of Computing, (11)6, pp 637-664, Springer Verlag, 1999.
3. T. Amnell et. al.: Uppaal - Now, Next, and Future. In Proc. Modelling and Verification of Parallel Processes (MOVEP'2k), LNCS 2067, pages 100-125, Springer Verlag, 2000.
4. Paul Ammann, Paul E. Black, and Wei Ding, Model Checkers in Software Testing, NIST-IR 6777, National Institute of Standards and Technology, 2002.
5. A. Hartman and K. Nagin: The AGEDIS Tools for Model Based Testing, Proceedings of ISSTA 2004, ACM, ISBN 1-58113-820-2, pages 129-132, 2004.
6. A.Gargantini, E.Riccobene, S.Rinzivillo: Using Spin to Generate Tests from ASM Specifications. Abstract State Machine 2003, LNCS 2589, Springer Verlag, page 263, 2003.
7. Alan Hartman: Model-based test generation tools, http://www.agedis.de/documents/ModelBasedTestGenerationTools_cs.pdf
8. Micskei Zoltán: Automatikus tesztgenerálás modell ellenőrzővel, TDK dolgozat, http://www.hszk.bme.hu/~mz271/tesztgeneralas_tdk.pdf, 2004.
9. Hyoung Seok Hong et al: Data flow testing as model checking, International Conference on Software Engineering, ISBN: 0270-5257, pages 232-242, 2003.
10. Alexandre David, M. Oliver Möller and Wang Yi: Verification of UML Statecharts with Real-Time Extensions, Technical report, Uppsala University 2003.

Micskei Zoltán, műszaki informatikus hallgató

Budapesti Műszaki és Gazdaságtudományi Egyetem, 1111 Budapest, Műegyetem rkp. 3-9.

Telefon: +36-1-789-6575, e-mail: mz271@hszk.bme.hu