[MC]SQUARE

Wiandt Bernát (P0RKMZ)

2012. december 14.

1. Modellellenőrzés

A modellellenőrzés egy formális verifikációs eljárás, amely rendszerek automatikus analízisére használható. Ebben a leírásban a beágyazott rendszerek témakörén belüli alkalmazásokra koncentrálunk. Sok beágyazott rendszer kerül felhasználásra biztonságkritikus szituációkban, azonban a teljes rendszer széleskörű ellenőrzése általában nem megvalósítható, nem gazdaságos. Ezek a rendszerek hibás működés esetén jelentős károkat okozhatnak.

A modellellenőrzés alkalmazását a jól ismert állapotrobbanás problémáján kívül az egyes gyártók által használt különböző beviteli formátumok, főleg azok átjárhatatlansága nehezíti. Nagyon kevés eszköz kínál szabványos fejlesztői eszközökhöz vagy integrált fejlesztői környezetekhez interfészeket. Ha mégis, akkor általában csak a logika és a modell különálló, diszkrét elemeinek ellenőrzésére van lehetőség. A fejlesztők problémája továbbá, hogy egy projekten belül is számos fejlesztőeszközt alkalmaznak a cégeken belül, ami a használt modellellenőrzőtől megkövetelné az összes eszközhöz nyújtott interfészt vagy a különböző reprezentációk közötti átjárhatóságot.

A probléma megoldása lehet, ha a C forráskódú programok ellenőrzésére koncentrálunk, ugyanis a beágyazott szoftverfejlesztések valamely fázisában ez biztosan rendelkezésre áll: vagy eleve így írjuk a programot vagy egy magasabbszintű (pl.: MATLAB) leírásból generáljuk. A cikkben tekintett modellellenőrző szoftverek közül csak a CBMC[2] volt képes kezelni a mikrokontrollerre specializált C kódot, de ezt az eszközt sem sikerült a szerzőknek átalakítani úgy, hogy tetszőleges, a felhasználó által megadott feltételt képes legyen ellenőrizni. A már létező eszközök hiányosságai miatt inkább egy saját, a mikrokontrollerekre íródott, C forráskódú programok modellellenőrzésére alkalmas szoftvert készítettek.

2. [MC]SQUARE

A [MC]SQUARE[5] modellellenőrző alkalmazás az Atmel 16/32/28L mikrokontrollere íródott programokat képes ellenőrizni. A bemeneti formátum lehet *elf*, *od* (avr-objdump), *asm* (Atmel assembler syntax) vagy *s* (GNU assembler syntax). A [MC]SQUARE nem a C forráskódú programot, hanem az abból a GCC-vel fordított assembly kódot tudja ellenőrizni. A modellellenőrzéskor használt feltételeket Computation Tree Logic (CTL) szintaxissal kell megadni, továbbá a felhasználó modellezheti az egyes I/O regiszterek állapotait.

A [MC]SQUARE szoftver az állapottér generálásához az AVRORA szimulátor keretrendszer cycle-accurate részét használja fel. Az AVRORA rendszer képes futtatni az ATMEL ATmega16, 32, 128L mikrokontrollerekre írott programokat, mindezt úgy, hogy cikluspontosan hajtja végre az egyes utasításokat. Ennek a megoldásnak előnye, hogy például az időzítők hatása pontosan modellezhető.

2.1. Architektúra

A [MC]SQUARE szoftver négy fő alkotóeleme a *statespacebuilder, modelchecker, util* és *gui* csomagokban található. A *statespacebuilder* csomag az állapotterek generálásához használt funkcionalitást tartalmazza. Ez a csomag használja a fentebb már említett AVRORA cycle-accurate szimulációs rendszerét. A *sim* csomag a szimulációs funkciókat implementálja, tartalmazza például az *mcu* csomagot, ami a mikrokontroller-specifikus szimulációért felel. A *core* csomagban általános funkciók vannak, a *modelchecker* pedig a modellellenőrzést valósítja meg. A *gui* csomag a felhasználói felületet, a *util* pedig a többi csomag által használt, közös adatstruktúrákat tartalmazza.

2.2. Modellellenőrzés

A modellellenőrzés két különböző módon történhet a rendszerben. Az első lépés mindíg az ellenőrizni kívánt program futtatható állománnyá fordítása. Ez szokásos esetben a GCC fordítóval történik. A bináris mellé készíteni kell egy CTL nyelvű specifikációt is.

2.2.1. „A" módszer

A folyamat első lépéseként az állapottér felépítése történik, amihez az AVRORA programocsomag értelmezőjét (interpreter) használja fel a [MC]SQUARE. Az

értelmezőn változtatásokat kellett végrehajtani, mert alapesetben minden csatlakoztatott I/O eszköz pontos állapotát ismerte, azokat is szimulálta. A szerzők célja azonban egy olyan rendszer megalkotása volt, amelyben a lefordított binárison és a CTL leíráson kívül nem kell mást megadnia a fejlesztőnek. Ezért minden I/O regiszter értéke kiolvasáskor nemdeterminisztikus (olyan komponens által vezérelt, amely nincs modellezve). A további változások megszüntetik az AVRORA cycle-accurate tulajdonságát, mert ez csak az állapotrobbanás problémájának súlyosbodását idézte volna elő. Ez azt jelenti, hogy néhány megszokott AVRORA komponens (pl.: A/D konverter, ISP, SPI, TWI, USART és JTAG) jelenleg nem működik. A szerzők dolgoznak ezen komponensek funkcionalitásának helyreállításán, de a cikk írásának időpontjáig csak az időzítők működését tudták biztosítani.

A nemdeterminisztikus működést az AVRORA nem tudja kezelni, ezért egy új komponens (*splitter*) bevezetése vált szükségessé. Amikor az ellenőrzött kód egy nemdeterminisztikus regisztert próbál olvasni, akkor a *splitter* előállítja a kérdéses regiszter összes lehetséges értékét, majd ezek mindegyikére meghívja a szimulátort. Látható, hogy a modellellenőrző által vizsgált állapottérben így olyan állapotok is előfordulhatnak, amelyek a működés során nem jöhetnének létre (over approximation). Ha a szimulátor által előállított állapot még nem szerepel a már ismert állapotok között, akkor bekerül egy ú.n. *build stack*-be. A következő lépésben a *build stack* első elemétől indul az állapottér továbbépítése, amely folyamat akkor fejeződik be, ha a *build stack* üres. Az előállított állapotgráfot felépítő minden állapot komplett memóriaképet tartalmaz.

A nemdeterminizmus által előállított állapotok száma függ a nemdeterminisztikus bitek számától az egyes regiszterekben. Például, ha csak 2 bit állapota nem ismert előre, akkor 4 állapot jön létre.

A következő lépés a modellellenőrzés folyamatában az egyes állapotok által kielégített atomi állítások (*atomic propositions*) felderítése. A folyamat végiglépked az állapotokon és felcímkézi azokat a kielégített atomi állításokkal. A címkézés után az állapothoz tartozó memóriakép törölhető.

A modellellenőrzés utolsó lépéseként a CTL nyelven megfogalmazott állítások ellenőrzése történik. A CTL operátorok közül az *EX*, *EG* és *EU* kerültek megvalósításra [3, 1]. Minden más operátor kifejezhető ezen operátorok valamely kombinációjával.

2.2.2. „B" módszer

A második módszer az elsővel ellentétben nem őrzi meg minden állapot memóriaképét a teljes állapottér felépítése alatt. Az állapot felcímkézése után a teljes memóriakép törölhető, így jelentős erőforrások takaríthatók meg, ezért jóval nagyobb állapottér kezelésére alkalmas, mint az „A" módszer. Hátránya, hogy mivel nincsenek meg a memóriaképek a korábbi állapotokról, nem képes ellenőrizni, hogy egy állapotot látott-e már korábban vagy sem. Ennek enyhítésére a szerzők bevezettek két hash értéket az állapot neve mellé, amelyek a memóriakép tartalmán alapulnak. Az ütközések (fals pozitív egyenlőségek) elkerülése érdekében a két hash algoritmusa különbözik egymástól: az első egy bit megváltoztatásakor nagyon távoli értékeket, míg a második ugyanilyen változásra nagyon közeli értékeket ad vissza. Ezzel a módszerrel sikerült a hash ütközés valószínűségét $n * 2^{-64}$-re csökkenteni, ahol n az állapotok száma. Az állapotok egyenlőségvizsgálatban előforduló fals pozitív értékek miatt helyességbizonyításra nem használható a „B" módszer, viszont sokkal gyorsabb hibakeresésre (debugging) alkalmas, akár a fejlesztés folyamatába beépítve. Ha a második módszer talál egy hibát a kódban, akkor a felhasználó biztos lehet benne, hogy a hiba a programban valóban létezik.

3. Eredmények

A [MC]SQUARE teszteléséhez egy már korábbról ismert példát, a villanykapcsoló programot használták fel[4]. A lényeg, hogy van egy gomb és egy lámpa, amely világít. Ha a gombot megnyomják, a lámpa tompított módban üzemel tovább. Ha a gombot két másodpercen belül mégegyszer megnyomják, a lámpa teljes fényerejével világít tovább, azonban ha a második megnyomás két másodpercen túl történik, a lámpa kialszik.

A modellellenőrző tesztelésekor a szerzők inkrementális stratégiát alkalmaztak. Először csak egy adott tesztprogram állapotterét építtették fel, majd egyszerű feltételeken keresztül végül a program teljes működését ellenőrizték. A folyamat során mind a specifikációban (CTL leírás), mind a programban találtak számukra ismeretlen hibákat.

4. Összefoglalás

A [MC]SQUARE az első olyan modellellenőrző szoftver, amely képes úgy vizsgálni ATMEL ATmega 16/32/128L-re íródott programokat, hogy a felhasználó csak a C kódot és a CTL kifejezéseket készíti el. Semmilyen plussz információ nem szükséges a sikeres ellenőrzéshez. A fent bemutatott eszköz az első lépés lehet az integrált fejlesztőkörnyezetekbe épített modellellenőrző szoftverek irányában. A fejlesztett rendszeren semmilyen változtatás nem szükséges, mert a fordított kódot szimulálva történik az ellenőrzés. A bemutatott program ugyan csak egyes ATMEL eszközökön működik, azonban az általános elv más eszközökre is könnyedén alkalmazható.

Hivatkozások

[1] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. Systems and software verification: Model checking techniques and tools. *Springer, Berlin*, 2001.

[2] E. Clarke, D. Kroening, and F. Lerda. A tool for checking ansi-c programs. *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, pages 168–176, 2004.

[3] E.M. Clarke, O. Grumberg, and D.A. Peled. Model checking. *The MIT Press, Cambridge*, 1999.

[4] K.G. Larsen and P. Petterson. Timed and hybrid systems in uppaal2k. *Presentation at MOVEP 2000*.

[5] Bastian Schlich and Stefan Kowalewski. Model checking c source code for embedded systems. *International Journal on Software Tools for Technology Transfer*, 11:187–202, 2009.