A CERN állapotgép-alapú vezérlőprogramjainak automatikus formális ellenőrzése

Darvas Dániel (PAW380)

(Szoftver verifikáció és validáció tárgy, házi feladat)

1. Bevezető

Ez az összefoglaló a CERN CMS kísérlet véges állapotgépeken alapuló vezérlőjének formális analízisét tekinti át, főként [1] alapján. A kutatást a CERN, a TU Eindhoven és az ETH Zürich munkatársai végezték közösen, 2010–2013 közt.

A CERN, az Európai Részecskefizikai Kutatóintézet a kontinens legnagyobb részecskefizikai központja. A kutatások legfontosabb eszköze az LHC (Large Hadron Collider, Nagy hadronütköztető), egy 27 km kerületű gyűrű, amelyben egymással ellentétes irányban két protonnyaláb (vagy ionnyaláb) halad közel fénysebességgel. Az LHC kerületén négy ponton lehetséges a nyalábok ütköztetése. Ezeken a pontokon egy-egy detektor található, amelyek az ütközések eredményeit figyelik és rögzítik. A négy detektor egyike a CMS (Compact Muon Solenoid, Kompakt müon szolenoid).

A CMS detektor működtetését és felügyeletét a detektorvezérlő rendszer (Detector Control System, DCS) végzi. A rendszer működését egy hierarchikus állapotgép írja le. A teljes állapotgép működése igen komplex, így kézi helyességvizsgálata nem lehetséges. A feldolgozott cikkek a detektorvezérlő rendszer automatikus formális verifikációját tűzték ki célul.

Feldolgozott cikkek. Jelen összefoglaló elsősorban az [1] folyóiratcikkre épít, amely egy bővített, átdolgozott változata [4]-nek. Egy korábbi cikk, [5] részletesebben ír a modellezési elvekről. Ezek mellett felhasználtam a [6] diplomatervet, amely számos részlettel egészíti ki a többi cikket, különösképp a saját verifikációs és redukciós algoritmusokkal kapcsolatban.

Az összefoglaló felépítése. Az összefoglaló további felépítése a következő. A 2. fejezet bemutatja a detektorvezérlő rendszer felépítését és megvalósítását. A 3. fejezet áttekinti a cikkekben javasolt modellezési és verifikációs folyamatot. A 4. fejezet részletesebben ismerteti a modellezési megoldásokat, míg az 5. fejezet a verifikáció részleteiről szól. A dokumentum az eredmények ismertetésével és összefoglalással zárul.

2. A detektorvezérlő rendszer (DCS) felépítése

A CMS detektor működtetését és felügyeletét a detektorvezérlő rendszer (Detector Control System, DCS) végzi. Ennek segítségével a környezet állapota (pl. hőmérséklet, páratartalom, feszültség) megfigyelhető és befolyásolható. A számos megfigyelő- és beavatkozóeszköz (Device Unit, DU) egy hierarchikus (irányított aciklikus) struktúrába van szervezve, ahol minden levél egy eszközt, a többi csomópont pedig egy-egy alrendszert (Control Unit, CU) reprezentál. A CU és DU csomópontok működése, állapota egyaránt egy-egy véges állapotgéppel írható le [2, 3]. Ez azt jelenti, hogy például a legfelső szintű csomópont *OK* állapota a teljes rendszer helyesen működő állapotát reprezentálja. Ha bármelyik DU meghibásodik, akkor annak állapota pl. *ERROR*-ra változhat, aminek hatására várhatóan az összes, a hierarchiában felette lévő csomópont is *ERROR* állapotba fog kerülni, beleértve a legfelső szintű csomópontot. Ha a legfelső szintű csomópont egy *SWITCH_ON* eseményt kap, ennek hatására az összes gyerekcsomópontjának is bekapcsolásra felszólító jelzést küld, majd önmaga is bekapcsolt állapotba kerül.

Az egyes csomópontok működését leíró állapotgépek igen egyszerűek, a CMS esetén átlagosan 8 állapotot tartalmaznak. A komplexitást az okozza, hogy a csomópontok egymástól nem függetlenek. A CMS esetén kb. 32000 csomópont működik együtt, amely kb. 10^{26000} különböző lehetséges globális

```
class: $FWPART_$TOP$RPC_Chamber_CLASS
    state: OFF
      when ( ( $ANY$FwCHILDREN in_state ERROR ) or
             ( $ANY$FwCHILDREN in_state TRIPPED ) ) move_to ERROR
      when ( $ANY$RPC_HV in_state {RAMPING_UP,RAMPING_DOWN} )
             move_to RAMPING
[...]
    action: STANDBY
      do STANDBY $ALL$RPC_HV
      do ON $ALL$RPC_LV
    action: OFF
      do OFF $ALL$FwCHILDREN
[...]
```

1. ábra. Példa SML-kód (forrás: [1])

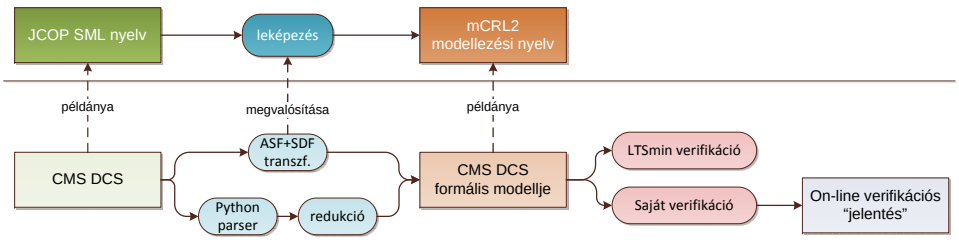
állapotot eredményez. Ilyen komplexitás mellett még alapvető helyességi kritériumok, mint például a deadlock- vagy livelock-mentesség kézi ellenőrzése nem megvalósítható.

Az állapotgépek leírására a JCOP (CERN Joint Controls Project) keretrendszerben definiált SML (State Machine Language) nyelvet használják. Ez egy szöveges formában definiálja minden egyes csomóponthoz annak lehetséges állapotait és akcióit (eseményeit), illetve azt, hogy adott aktuális állapot, akció és feltételek esetén mely állapotokba lehetséges az átmenet [2].

Az SML nyelvben hierarchikus állapotgép esetén lehetőség van kvantorok használatára is az őrfeltételekben. Így az őrfeltétel tartalmazhat `$ALL$<class> in_state S` feltételeket, amely pontosan akkor igaz, ha a csomópont összes `<class>` típusú gyereke `S` állapotban van, vagy `$ANY$<class> in_state S` feltételeket, amely pontosan akkor igaz, ha a csomópontnak van legalább egy `<class>` típusú gyereke `S` állapotban. Egy példa SML-kód látható az 1. ábrán, amely a *Chamber* alrendszer egy részletét mutatja be. Ez alapján ha a Chamber modul `OFF` állapotban van és bármely gyermeke `ERROR` vagy `TRIPPED` állapotú, úgy a Chamber modul is `ERROR` állapotba fog lépni. Ha a Chamber modul `STANDBY` üzenetet (eseményt) kap, akkor minden `RPC_HV` típusú gyerekének `STANDBY`, minden `RPC_LV` típusú gyerekének pedig `ON` üzenetet fog küldeni.

3. A javasolt ellenőrzési folyamat áttekintése

A feldolgozott cikkekben javasolt modellezési és verifikációs folyamatot tekinti át a 2. ábra. A CMS detektorvezérlő rendszeréhez is használt SML nyelvet leképezték az mCRL2 modellezési nyelvre[1]. A leképezést kétféle módszerrel valósították meg: elsőként az ASF+SDF metaprogramozási keretrendszerrel, majd később egy saját, Python-alapú implementációt is adtak [6]. Ez utóbbi modellredukciókkal is kiegészítésre került. Az előállított modellen egyrészt az LTSmin modellellenőrző eszközzel végeztek ellenőrzéseket, majd később saját verifikációs algoritmusokat is implementáltak. Végül ezek eredményét egy on-line verifikációs jelentés formájában tették a fejlesztők számára elérhetővé.



2. ábra. A folyamat áttekintése

[1] `http://www.mcrl2.org/`

4. Modellezés

A formális verifikációhoz először a vizsgálandó modellek formális szemantikáját kell definiálni. Ehhez elkészítették az SML nyelv leképezését az mCRL2 modellezési nyelvre. Az mCRL2 egy processzalgebra, amely szemantikája jól illeszthető az SML nyelv szemantikájához, az állapotgépeket és ezek hierarchikus kapcsolatát egyaránt jól le tudja képezni. Az mCRL2 előnye továbbá az is, hogy számos verifikációs eszköz kapcsolódik hozzá, amelyek így az mCRL2 nyelvre leképezett SML modelleken is használhatók. A leképezés formálisan nem kerül definiálásra a cikkekben, de az [1, 5] bemutatja a főbb leképezési szabályok egyszerűsített változatát.

Megvalósítás. A leképezést először az ASF+SDF metaprogramozási keretrendszer segítségével készítették el, amely egy gyors prototípust szolgáltatott. Később, mivel a keretrendszert már nem fejlesztik, továbbá hordozhatósági problémák is adódtak, a transzformációt újraimplementálták Python nyelven. Így a SML–mCRL2 átalakítást teljesen automatizálni tudták.

Helyesség. Tekintve, hogy az SML formális szemantikája nem definiált, a leképezés helyessége (azaz az, hogy a definiált formális szemantika illeszkedik-e a JCOP által implementált szemantikához) nehezen vizsgálható. Ehhez egyrészt az SML fejlesztői csapattal folytattak beszélgetéseket, a SML logikájának mélyebb megértése érdekében. Emellett az mCRL2 eszközkészletét felhasználva szimulációkat végeztek valós kódrészleteken, amelyeket összevetettek az SML szimulációjával [1].

5. Verifikáció

Az elkészített mCRL2 modelleken számos tulajdonságot vizsgáltak. Ezen vizsgálatok egy részét az LTSmin eszköz[2] segítségével, másokat pedig saját algoritmusokkal végezték. A verifikáció célja bizonyos előre definiált, tipikus hibák automatikus kiszűrése, így a fejlesztők által később specifikálandó követelmények vizsgálatára az áttekintett munkák nem térnek ki, erre nem nyújtanak támogatást.

5.1. Verifikáció az LTSmin eszközzel

Az LTSmin egy modellellenőrző keretrendszer, amely többek közt mCRL2 modellek vizsgálatára is képes. A fejlesztőkkel közösen több általánosan használható követelményt is megfogalmaztak modális μ-kalkulus segítségével, mint például holtpontmentesség vagy instabil állapotok nemléte.

A μ-kalkulus kifejezőereje nagy, így a fenti követelmények megfogalmazása nem okozott gondot. Ugyanakkor a kifejezések kiértékelése nagy számítási kapacitást igényel, így a gyakorlati használhatóság érdekében bizonyos problémákra saját, specifikus algoritmusokat fejlesztettek.

5.2. Saját verifikációs algoritmusok

Annak érdekében, hogy a fejlesztők gyors visszajelzést kaphassanak, a fontosabb kifejezések vizsgálatára specifikus algoritmusokat fejlesztettek. Itt két vizsgálatot tekintek át a [6] alapján: a livelock és a „páronkénti elérhetőség” problémáját.

Livelock vizsgálata. Livelock esetén a DCS nem válaszol az operátor kéréseire. Így nyilvánvaló, hogy fontos a livelock lehetséges jelenlétének ellenőrzése, azaz annak vizsgálata, hogy van-e olyan $a \to b \to c \to \ldots \to a$ kör valamelyik állapotgépben, amely végtelen sokszor tüzelhet. A problémának két alesetét különböztetik meg: amikor a kör egy adott csomópont állapotai közt van (local loop), illetve amikor több csomópont állapotai közt van kör (non-local loop). A teljesítmény javítása érdekében ennek egy felülbecslését vizsgálják: létezik-e olyan kör a hierarchikus állapotgépben, amelyben egyik tranzíció sincs eseményre triggerelve. Ez egy szükséges, de nem elégséges feltétele a fentiek szerint definiált livelocknak, ugyanakkor mindegyik ilyen kör tervezési hibát jelez. A körkeresési probléma megoldásához az állapotgépet először Kripke-struktúrává alakították, majd ezen a körkeresési problémát egy korlátos modellellenőrzési problémává alakították, amelyet egy SAT-megoldóval (Yices[3]) végeztek el.

[2] `http://fmt.cs.utwente.nl/tools/ltsmin/`
[3] `http://yices.csl.sri.com/`

Páronkénti elérhetőség vizsgálata. Egy másik tipikus követelmény annak vizsgálata, hogy minden állapot minden más állapotból elérhető, azaz az állapotgép nem ragadhat az állapotok egy valódi részhalmazában. Ezt a szerzők „páronkénti kölcsönös elérhetőség" problémájának nevezik, amely lényegében annak vizsgálata, hogy az állapotgépek minden állapota visszatérő állapot-e. Ennek hatékony vizsgálatához egy alulbecslést alkalmaztak. A vizsgálat során előállítottak egy gráfot, amely csúcsai az egyes állapotok, élei pedig a lehetséges állapotátmenetek (függetlenül azok triggereitől és feltételeitől – emiatt az eredmény közelítő jellegű, nem tár fel feltétlenül minden hibát). Az így előállított gráfon erősen összekötött komponenseket kerestek. Amennyiben az egész gráf nem alkot egyetlen erősen összekötött komponenst (SCC), akkor lehetséges, hogy bizonyos állapothalmazból nem lehet kilépni a futás során. Amennyiben az összes csomópont egyetlen SCC-t alkot, garantáltan minden állapotpárra igaz a páronkénti elérhetőség.

5.3. Redukciók

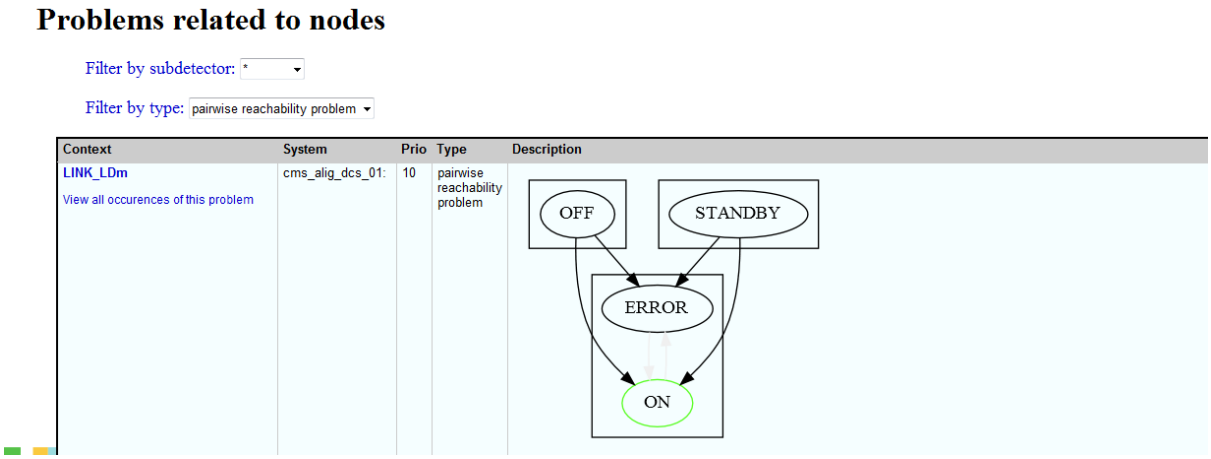
Annak érdekében, hogy a teljesítmény tovább javítható legyen, [6] bevezet néhány doménspecifikus redukciós szabályt. Ezek közül itt egy, az ún. *top bouncer reduction* kerül bemutatásra. A módszer alapötlete az, hogy ha helyi kör (local loop) nem található egyik csomópontban sem, akkor livelock csak akkor állhat elő, ha egy nem-helyi kör (non-local loop) található a csomópontok hierarchiájában. Ennek szükséges feltétele, hogy legyen olyan csomópont, ami a gyerekétől kapott információ alapján a gyerek állapotát befolyásoló akciót hajtson végre. Az ilyen csomópontot „top bouncer"-nek nevezik. Mivel a „top bouncer" csomópont létezése szükséges feltétele a nem-helyi körnek, ezért egy lehetséges redukció a körkeresés előtt az olyan csomópontok eltávolítása, amely nem lehet top bouncer, hiszen ezek nem lehetnek részei nem-helyi körnek sem. Az „elárvult" (mással nem összekötött) csomópontok is elhagyhatók, hiszen ezek sem vehetnek részt nem-helyi kör kialakításában. Így drasztikusan egyszerűsödhet az állapotgép-hierarchia, ami jelentősen gyorsíthatja a körkeresést. Természetesen az így redukált modelleket csak nem-helyi körök keresésére lehet felhasználni.

A CMS esetén ezzel a redukcióval kb. 32000 csomópontról 15000-re sikerült csökkenteni a modell méretét, amely a 10^{26402} méretű állapotteret 10^{1189}-re csökkentette.

6. Értékelés és összefoglalás

A feldolgozott cikkekben a CERN hierarchikus állapotgépeken alapuló detektorvezérlő rendszereinek vizsgálatára mutattak be egy módszert. A módszer a vezérlőrendszer automatikus formális modellre leképezéséből és ezen történő automatikus vizsgálatokból áll. Az elkészített eszközök segítségével a detektorvezérlő rendszer fejlesztői önállóan is vizsgálhatják a rendszer működését. A vizsgálatok eredményét egy intuitív, könnyen érthető formátumban tárják a felhasználók elé (lásd a 3. ábrát példaként).

A javasolt módszert integrálták a CERN fejlesztési folyamatába és számos hibát sikerült a formális verifikáció segítségével találni. Ezek közül többről megmutatták korábbi naplófájlok alapján, hogy éles üzemben is előfordulhatnak. A javasolt módszerek segítségével sikerült a detektorvezérlő rendszerek minőségét javítani úgy, hogy a fejlesztési költséget nem növelték.



3. ábra. A CMS DCS vizsgálatának eredményei a CMS Online rendszerben (forrás: [6, Fig. 5.4])

Hivatkozások

[1] Yi Ling Hwong, Jeroen J.A. Keiren, Vincent J.J. Kusters, Sander Leemans, and Tim A.C. Willemse. Formalising and analysing the control software of the Compact Muon Solenoid experiment at the Large Hadron Collider. *Science of Computer Programming*, 78(12):2435–2452, 2013. Special Section on International Software Product Line Conference 2010 and Fundamentals of Software Engineering (selected papers of FSEN 2011). `http://vincentkusters.org/pdf/AnalysingCMS_SCICO2013.pdf`.

[2] Sascha Marc Schmeling. Joint PVSS & JCOP framework course. Course Manuscript CERN-JCOP-2004-016, 2004. `http://www-cdf.fnal.gov/upgrades/controls/Meetings/PVSS_Cource.pdf`.

[3] Clara Gaspar. SMI++ – the finite state machine toolkit of the JCOP framework (presentation), 2004. `http://indico.cern.ch/event/a041114/material/0/1?contribId=s1t4`.

[4] Yi-Ling Hwong, Vincent J.J. Kusters, and Tim A.C. Willemse. Analysing the control software of the Compact Muon Solenoid experiment at the Large Hadron Collider. In Farhad Arbab and Marjan Sirjani, editors, *Fundamentals of Software Engineering*, volume 7141 of *Lecture Notes in Computer Science*, pages 174–189. Springer, 2012. `http://arxiv.org/pdf/1101.5324v1.pdf`.

[5] Yi Ling Hwong, Tim Willemse, Vincent Kusters, et al. An analysis of the control hierarchy modelling of the CMS detector control system. *Journal of Physics: Conference Series*, 331(2):022010, 2011. `http://iopscience.iop.org/1742-6596/331/2/022010/pdf/1742-6596_331_2_022010.pdf`.

[6] Sander J. J. Leemans. Validation of CERN's finite state machines. Master's thesis, Eindhoven University of Technology, 2012. `http://www.win.tue.nl/~timw/downloads/Leemans_thesis.pdf`.