Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Monitorszintézis temporális követelmények alapján

Pallagi Péter (UCJQV8), IV. évfolyam (BSc) mérnök informatikus szakos hallgató
Konzulens: Majzik István egyetemi docens, MIT
Rendszertervezés szakirány
Önálló laboratórium összefoglaló
2013/14. I. félév

A temporális logikák lehetőséget nyújtanak arra, hogy kijelentések igazságának *logikai időbeliségi* (szekvenciális) változását vizsgálhassuk. A temporális logika rendszerében ún. *temporális operátorokat* alkalmazhatunk erre a célra.

Fontos megjegyezni, hogy az időbeliség vizsgálata itt nem a hagyományos értelemben vett időre, időpillanatokra, időintervallumokra, hanem tipikusan logikai időre, a bekövetkező események *sorrendiségére* vonatkozik. Az ilyen típusú logikákat folyamatosan működő rendszerek (operációs rendszerek, beágyazott rendszerek, protokollok) jellemzésére használhatjuk, melyeknél a működés helyessége nem fogalmazható meg a kezdeti- és végállapotra vonatkozó pre- és postkondíciók formájában. Segítségükkel azonban formálisan előírhatjuk a helyes működést az időbeliség aspektusában. A temporális operátorok lehetőséget adnak mind a rendszer *tulajdonságainak működés közbeni vizsgálatára*, mind pedig a rendszer működésével szembeni *követelmények felállítására*.

A tulajdonságok egy része *lokális* jellegű, tehát egy-egy időpillanathoz köthető, más részük *elérhetőségi*, azaz a működés során esetlegesen bekövetkező jövőbeli időpillanatokra vonatkozik. A elérhetőségi tulajdonságokat további két kategóriába sorolhatjuk, ezek az *biztonság* és az *élőség*. A biztonsági tulajdonságok lényegében bizonyos veszélyes, nem kívánatos helyzetek elkerülését fogalmazzák meg. Ebből adódóan *univerzális kvantort* alkalmaznak az időpillanatokra, hiszen minden jövőbeli pillanatban elkerülendőek ezek a helyzetek, így ezt egyértelműen univerzális követelményként támasztjuk a rendszerrel szemben. Az élő jellegű tulajdonságok tipikusan bizonyos kívánatos helyzetek elérését (pl. elküldött kérés kiszolgálásra kerül, nincs kiéheztetés, stb.) írják elő. Ezekhez a tulajdonságokhoz értelemszerűen *egzisztenciális kvantort* kell társítanunk.

Felmerül az igény, hogy a gyakorlatban is alkalmazható megoldást találjunk, mely a rendszer működését futásidőben figyeli, és felismeri az általunk előírt temporális kényszerek teljesítését illetve megsértését. Tehát egy monitorozó alkalmazásra van szükségünk. A kiértékelés algoritmusa rendelkezésünkre áll, működésének lényege, hogy a kiértékelés lényegi részét speciális kiértékelő blokkok végzik, melyeknek egyedi működési mechanizmusát a rendszer aktuális állapotában ellenőrizendő logikai kifejezések határozzák meg. Az algoritmus valójában csak menedzseli az ellenőrzési folyamatot, ha szükséges, példányosítja a megfelelő blokk-típust a *rendelkezésre álló készletből*, csatlakoztatja a kiértékelési lánchoz, átadja az aktuális állapot paramétereit, és elindítja a kiértékelést, majd ismétli a procedúrát, mindaddig, míg a követelmény egyértelműen nem sérül, vagy nincs több ellenőrizendő állapot. Azonban a követelmények dinamikusan változnak, és minden egyes követelményrendszer ellenőrzéséhez specifikus blokk-készletre van szükség. Ezek manuálisan történő implementációja nyilván nehéz és fáradságos feladat. Ezt a problémát úgy oldhatjuk fel, ha az eszközkészletet képesek vagyunk automatizáltan generálni, adott lineáris temporális logikai (LTL) követelmény alapján.

A félév során elkészített Java alkalmazás ezt kezeli hatékonyan. Bemenete egy LTL kifejezés melyet egy feldolgozó egység ekvivalens átalakítások sorozatával olyan formára hoz, hogy abból előállíthatóak a kiértékeléshez szükséges blokkok melyeknek belső működése, valamint interfészei alkalmassá teszik őket az elvárt működés ellenőrzésére, valamint az egymással való interakcióra. A feldolgozó egység addig transzformálja a kifejezést és konstruálja dinamikusan az újabb blokk típusokat, amíg a transzformáció eredményeként újabb, még nem létező blokk-típus áll elő. Ha a folyamat véget ér, a vezérlés a *kódgenerátor* egységhez kerül, mely az előállt adatszerkezetekből forráskódot generál (a generálás Java Emitter Templates segítségével történik, mely egy template alapú Eclipse plugin). A *kódgenerátor* egy teljesen új projektet hoz létre, melybe a blokkok dinamikusan előállt forráskódja mellett statikus (a bemeneti kifejezéstől független) osztályok forrását is elhelyezi. A létrejött, futtatható projekt központi része a már említett *végrehajtó egység*, mely új bemenet érkezésekor futtatja a kiértékelést, amennyiben szükséges, példányosítja, majd kaszkádosítja a következő – az aktuális állapot kiértékeléséhez szükséges – blokkot, ha pedig kiértékelhetővé vált a követelmény, visszaadja az eredményt, és terminál.