

**TESZTADAT ELŐÁLLÍTÁSA
KÉNYSZERKIELÉGÍTÉSI FELADAT
MEGOLDÁSÁNAK SEGÍTSÉGÉVEL**

Barta Ágnes

Budapesti Műszaki és Gazdaságtudományi Egyetem,
Budapest, Magyarország

Szatmári Zoltán

Budapesti Műszaki és Gazdaságtudományi Egyetem,
Budapest, Magyarország

TESZTADAT ELŐÁLLÍTÁSA KÉNYSZERKIELÉGÍTÉSI FELADAT MEGOLDÁSÁNAK SEGÍTSÉGÉVEL

Absztrakt

Egy alkalmazási terület fogalmainak, összefüggéseinek leírására széles körben használnak szakterület-specifikus nyelveket (Domain Specific Language, DSL). A célunk egy módszer bemutatása, amely támogatja az automatikus teszteset generálást absztrakt tesztadatok alapján. A módszer egy olyan leképezésen alapszik, amely egy metamodellt, jólformáltsági kényszereket és hiányos példánymodellt vár bemenetül. A keretrendszer a modellt kiegészíti új elemekkel, ami tesztesetként értelmezhető. A teszteset generálás komoly kihívást jelent, mert (i) az összetett DSL-eken történő logikai következtetés algoritmikusan eldönthetetlen probléma, (ii) elméleti nehézségei vannak a hozzáadott jólformáltsági kényszerek kezelésének, és (iii) olyan eszköz fejlesztésére van szükség, logikai ismeretek nélkül is használható. Az eszközt az R3COP ARTEMIS projektben alkalmaztuk.

Kulcsszavak: szakterület specifikus nyelv, jólformáltsági kényszer, elsőrendű logika, teszteset generálás, tesztelés

Bevezetés

Egy alkalmazási terület fogalmainak és összefüggéseinek leírására gyakran használnak szakterület specifikus nyelveket (DSL). A DSL-ek segítségével egyszerűen lehet az adott területhez kapcsolódó tesztadatokat definiálni, amely elkészítését grafikus tervezőeszközök segítik. A megtervezett metamodellekből könnyen származtathatóak és generálhatóak tesztesetek.

A következőkben egy módszert és egy arra épülő keretrendszert mutatunk be, amely képes metamodell és jólformáltsági kényszerek felhasználásával egy kezdeti kiindulási példánymodellt (Gogolla et al. 2005) kiegészíteni, úgy, hogy a definiált kényszereket és a metamodell megkötéseit kielégítse. Az így kapott érvényes modellt tesztesetként lehet felhasználni.

Az automatikus teszteset előállítás nehéz feladat, mivel (i) az összetett DSL-eken történő logikai következtetés algoritmikusan eldönthetetlen probléma, (ii) további elméleti nehézségei vannak a hozzáadott jólformáltsági kényszerek és származtatott értékek kezelésének, és (iii) olyan eszköz

fejlesztésére van szükség, ami a következtetési eljárás ismerete nélkül is használható.

A tanulmányban bemutatjuk az eszköz megvalósítását motiváló R3Cop ipari projektet, illetve példát mutatunk a sikeres tesztgenerálásra.

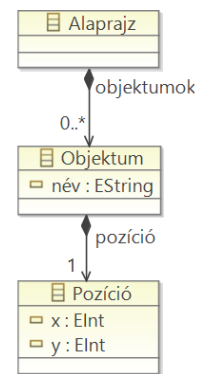
Motiváció

A módszer megalkotását az ARTEMIS R3COP ipari projekt motiválta, amelyben biztonságkritikus autonóm rendszerek (pl. ipari robotok) tesztelésének támogatása a cél. A konkrét feladat a tesztesetek előállítása volt OCL kényszerekkel meghatározott absztrakt tesztleírások alapján.

Az Elettric 80 cég lézervezérlésű autonóm villástargonca (Laser Guided Vehicle, LGV) gyártásával foglalkozik. A targoncákat raktár épületekben használják, ahol a sikeres működéshez be kell tartani különböző biztonsági követelményeket. A cél az volt, hogy ezen LGV-k tesztelését támogassam: olyan teszteseteket kellett létrehozni, amelyben az LGV környezete automatikusan áll elő a különböző hozzáadott kényszereket betartva, majd a robot tesztelése ebben a környezetben zajlik. A targonca környezetét egy DSL definiálásával foglalmaztuk meg.

Háttérismeretek

A DSL-eket metamodellek készítésével foglalmazzuk meg. A metamodel a modellezési nyelv modellje, tartalmazhat osztályokat, attribútumokat és referenciákat. A metamodelból példányosítással példánymodellek készíthetők el, amelyek a metamodelben megfogalmazott kényszereket, szabályokat betartják, illetve az objektumok attribútumai konkrét értéket kapnak. A következő metamodel az R3Cop projekt modelljének egy részlete, ami alaprajz definiálására alkalmas. Az *Alaprajz*on *Objektum*ok helyezkedhetnek el, amelyekhez tartozik egy-egy *Pozíció*. Az objektumnak van *neve*, a pozíciók pedig rendelkeznek *x* és *y* koordinátákkal. Az osztályokat az *objektumok* és a *pozíció* tartalmazási élek kötik össze, amelyekhez számossági kényszereket társítottunk. A metamodel példányosításával hozhatók létre példánymodellek, amire példa a 2. ábrán látható.



1. ábra
Metamodel

Az Object Constraint Language (OCL) (OMG 2006) segítségével különböző jólformáltsági kényszereket fogalmazhatunk meg a meta-illetve példánymodellel kapcsolatban. A kitűzött cél eléréséhez a kényszerek azon csoportját használtuk, amelyek kiértékelésük során vagy igazat, vagy hamisat

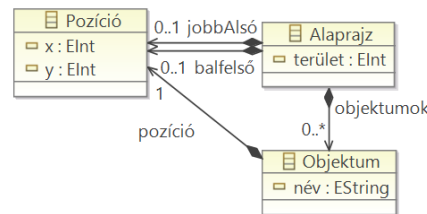
kapunk. Az OCL nyelv előnyét és használhatóságát egy példán keresztül mutatjuk be. Az előbbi metamodelt kiegészítettük egy jobbAlsó és balFelső referenciákkal, az új modell a 3. ábrán látható. Ezek a pozíciók megadják a tervrajz bal alsó és jobb felső sarkának koordinátáit. Az objektumoknak az alaprajz területén belülre kell esniük: $Alaprajz.allInstances() \rightarrow forall(a: Alaprajz | a.objektumok \rightarrow forall(o: Objektum | (a.jobbAlsó \neq null \text{ and } a.balFelső \neq null) \text{ implies } (o.pozíció.x > a.balFelső.x \text{ and } o.pozíció.x < a.jobbAlsó.x)))$. A kényszer az y koordinátákra is kimondható. Az OCL nyelv használatával egy olyan kényszer tudunk definiálni, amely a metamodellelemével nem írható le.

A származtatott értékek a modell azon elemei, amelyek a többi modellelemből egy meghatározott módon mindig kiszámíthatóak. Az előbbi metamodellel újonnan bevezetett terület attribútum származtatott értéként értelmezhető.

Az elsőrendű logika segítségével logikai állításokat fogalmazhatunk meg. A nyelv tartalmaz különböző kvantorokat, atomi mondatokat, termeket, konstansokat, változókat, logikai műveleteket, predikátumokat és függvényeket. A következő példa igazat ad eredményül, ha az *alap* nevű példány típusa *Alaprajz*: $isType!Alaprajz(alap)$. Az SMT (Satisfiability Modulo Theories) (SMT 2013) megoldók elsőrendű logika nyelvét használják a következtetési eljárások során, amelyet kiegészítettek különböző beépített axiómarendszerekkel, következtetési eljárásokkal. Az SMT megoldó egyik lehetséges felhasználása, a bementben adott lekötetlen változók kiegészítése úgy, hogy matematikailag helyes axiómarendszert kapjunk.



2. ábra
Példánymodell



3. ábra Kibővített metamodell

Tesztgenerálás módszere

A tesztgenerálás módszere egy olyan matematikai kényszerkielégítési problémán alapszik, aminek egy hiányos tesztadatot, és a tesztleírásra használt DSL különböző megkötéseit adjuk a bemenetére és logikai feladatmegoldó segítségével olyan tesztadatot állít elő, ami megfelel az összes definiált jólformáltsági követelménynek.

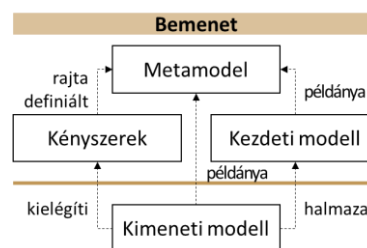
Az elkészült eszköznek bemenetként a következő paramétereket kell megkapnia: a DSL-t leíró metamodelt, egy hiányos kezdeti példánymodellt és OCL nyelven megfogalmazott jólformáltsági kényszereket. A hiányos

példánymodell egy speciális modell, amelyben nem követelmény még az összefüggőség, hiányozhatnak objektumok, kötelező attribútumok és referenciák, lehetnek plusz élek behúzva, illetve tartalmazhat absztrakt objektumokat.

A bemeneti paraméterek meghatározása után lehetőségünk van a példánymodell méretének meghatározására:

- a kezdeti mérettel azonos: a hiányos példánymodellhez nem adható új objektum
- korlátozott: a modell méretét fixálhatjuk, ebben az esetben limitált számú új objektum hozható létre
- nem meghatározott: a következtetési eljárás során a kényszereknek megfelelően alakul ki a modell mérete

A modellgenerálás során elvárás, hogy a kimenetként kapott modell tartalmazza teljes egészében a kezdeti modellt, a benne definiált kapcsolatok, az attribútumok megfeleljenek a metamodellben megfogalmazottaknak és kielégítsék a definiált jóformáltsági kényszereket. A felsorolt paraméterek és közöttük lévő viszonyok a 4. ábrán tanulmányozhatóak.



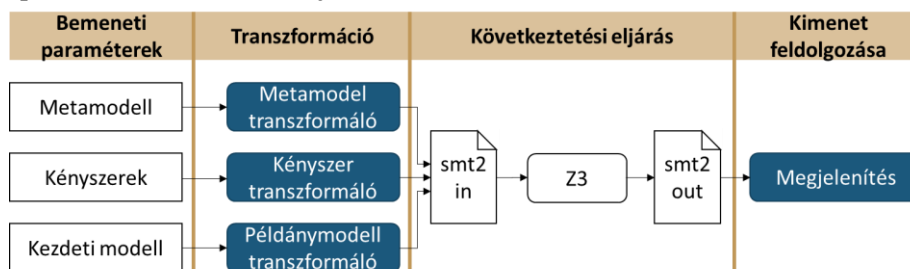
4. ábra A modellgenerálás paraméterei

A modellgenerálás során egy-egy transzformáció a metamodellt, a példánymodellt és a kényszereket automatizáltan feldolgozza, s lefordítja elsőrendű logikai axiómarendszerre, amelyet átadunk a Microsoft Z3 SMT megoldónak, ami a különböző következtetési eljárásainak köszönhetően hatékonyan meg tudja oldani a definiált problémát. Az SMT megoldó a következő kimeneteket adhatja az eljárás végeztével:

- sat: ebben az esetben a megadott probléma megoldható
- unsat: a megadott kényszerek vagy a DSL definíciója ellentmondást tartalmaz, azért a probléma megoldhatatlan
- timeout: a megoldó nem tud semmi biztosat mondani a probléma megoldhatóságáról

A kapott kimeneteknek megfelelően különböző a teendők. Timeout esetén a kényszerhalmaz redukálásával próbálkozhatunk, illetve magán a problémán egyszerűsíthetünk a sikeres tesztgenerálás érdekében. Unsat esetén az elsőrendű logika nyelvén megfogalmazott ellenpéldát a megalkotott keretrendszer értelmezi, s elkészíti a definiált érvénytelen példánymodellt, amelyet megvizsgálva megszüntethetjük az inkonzisztenciát a DSL-ben vagy a kényszerek között (Clavel et al. 2009). Sat esetén az elsőrendű logika nyelvén megfogalmazott példát az eszköz értelmezi, s előállítja az annak megfelelő példánymodellt, amelyet tesztetként használhatunk, különböző beépített

vizualizációs lehetőségekkel megjeleníthetjük vagy értelmezhetjük EMF példánymodellként, s esetlegesen további eszközök bemeneteként felhasználhatjuk. A tesztgeneráláshoz szükséges komponenseket és azok kapcsolatát az 5. ábra mutatja be.



5. ábra A modellgenerálás folyamata és fázisai

Leképezés

A metamodellt, a példánymodellt, s a kényszereket is le kell fordítani elsőrendű logikai kifejezéssé a sikeres modellgenerálás érdekében (Semeráth 2013) (Semeráth et al. 2013) (Beckert et al. 2002) (Cabot et al. 2007). A transzformáció célja, hogy készüljön egy olyan logikai axiómarendszer, amely konzisztens az eredeti DSL-lel.

A leképezés során a példánymodell és a kényszerek logikai megfelelőjének a metamodell leképezésére kell épülniük, hiszen a példánymodellek és a kényszerek hivatkoznak a metamodell elemekre.

A leképezés elméletének megfogalmazása volt a legbonyolultabb része a tesztgenerálási folyamat kialakításának, hiszen a leképezést egységesíteni kellett, úgy hogy a transzformáció automatizálható legyen. A leképezés során a másik probléma az volt, hogy a leképezendő nyelvek kifejezőereje nagyobb, mint az elsőrendű logikáé, így bizonyos nyelvi elemeket nem lehet egyértelműen megfeleltetni egy-egy logikai kifejezésnek. Ezekben az esetekben különböző approximációs technikákat alkalmaztam. Egy P^U kifejezés alulról közelíti, approximálja, a P^O pedig felülről a P kifejezést, ha teljesül, hogy: $P^U \rightarrow P$ illetve $P \rightarrow P^O$. Az approximációs technikák alkalmazása során a logikailag nem kifejezhető állításokat felülről közelítettük egy logikailag leírható kifejezéssel.

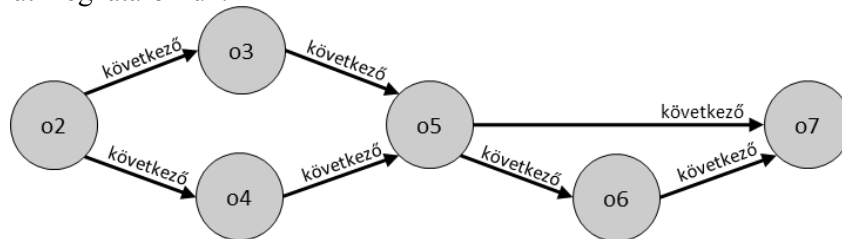
A leképezés során törekedtem arra, hogy a kapott kifejezések az elsőrendű logika egy hatékonyan eldönthető fregmensébe essenek (Effectively Propositional Logic, EPR). Az EPR az elsőrendű logika egy részhalmaza, amely egzisztenciális és univerzális kvantorokat ebben a kötött sorrendben tartalmazza. A leképezés során azért törekedtünk arra, hogy a kifejezések ide kerüljenek, mivel a tételbizonyító ezen a halmazon tud hatékonyan következtetéseket végezni.

Esettanulmány

A következőkben a teszteset generálásra szeretnénk példát mutatni az R3Cop ipari projekt egyik speciális esetén keresztül. A példában események sorrendezésének teszteléséhez szeretnénk egy tesztesetet készíteni, ahol a referenciák jelölik a különböző események egymásutániságát, amely alapján az eseményekhez egy végrehajtási sorrendet szeretnénk készíteni.

Az esettanulmány metamodeljének részlete a 6. ábrán látható. A *Világ* osztály tartalmazza az *Esemény*eket. Az *Esemény*nek van egy *időBélyeg*: *Elnt*, amely az esemény végrehajtási sorszámát jelenti. Az *Esemény*ek az *előző* és a *következő* referenciákkal vannak összekötve. A *következő* referencia azt jelenti, hogy ha *x* és *y* esemény, akkor, ha az *x* *következő* *y* kapcsolat fenn áll, teljesül, hogy *x* időbélyege kisebb *y* időbélyegénél, illetve *x* *előző* *y* azt jelenti, hogy *x* időbélyege nagyobb, mint az *y* időbélyege, tehát $x \text{ következő } y \leftrightarrow y \text{ előző } x$.

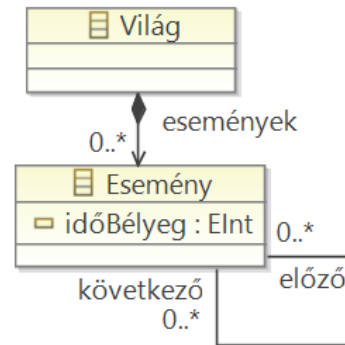
Adott egy kezdeti példánymodell, ami a 7. ábrán látható. Az körök szimbolizálják az eseményeket, a nyilak pedig a *következő* referenciákat. Az eseményekhez tartozó időbélyegek nincsenek kitöltve, tehát az a feladat, hogy azokat meghatározzuk.



7. ábra Kezdeti példánymodell

A probléma megoldásához szükséges különböző kényszerek definiálása, amelyek segítségével az időbélyeg értékek egyértelműen meghatározhatóak. A keretrendszerben ezeket a kényszereket OCL nyelven kell megadni, amelyeket jelen esetben természetes nyelven definiálunk:

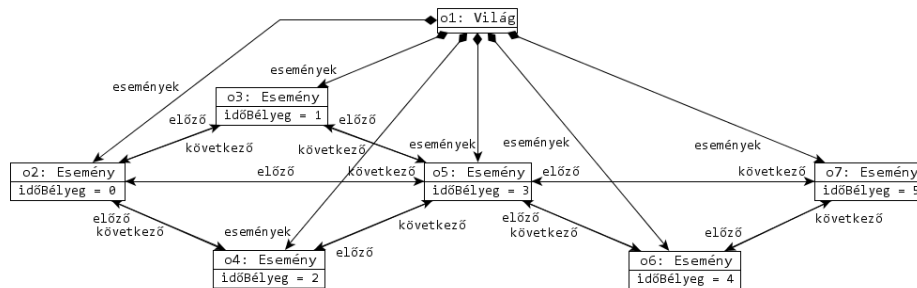
- Ha két esemény között teljesül, hogy *x* *következő* *y*, akkor *x* időbélyegének kisebbnek kell lennie, mint *y* időbélyege.
- Annak az eseménynek, amelynek nincs előzmény eseménye 0 időbélyeggel kell rendelkeznie.



6. ábra Esemény metamodel

- Minden időbélyegnek legalább 0 értékűnek kell lennie.
- A különböző eseményekhez különböző időbélyeg értékeknek kell tartoznia.

A problémát átadva a keretrendszernek a 8. ábrán látható megoldást kaptuk. A modell ki lett egészítve egy *Világ* típusú elemmel, illetve minden *Esemény* kapott egy kitöltött időbélyeget.



8. ábra Az érvényes példánymodell

A kapott példánymodell teljes, kielégíti a definiált kényszereket, illetve a metamodelben megfogalmazott megkötéseket. A modell tesztetésként felhasználható az LGV ütemezőjének tesztelésére.

A bemutatott példa egy nagyon gyakori sorrendezési problémára mutatott rá, amely más területeken is gyakran előkerül. A fent bemutatott példánál a kényszerekkel garantáljuk a körmentességet, amely nagyméretű, több ezer csúccsal rendelkező gráf esetén hasznosnak bizonyulhat.

Összefoglalás

Összességében elmondható, hogy elkészült a tesztgenerálási folyamat elmélete, amely során egy elkészített DSL-re alapozva jólformáltsági kényszerekkel és származtatott tulajdonságokkal definiálhatjuk egy-egy teszteteset absztrakt leírását, amelyből automatikusan előállíthatóak konkrét tesztetek. Mindennek a háttérében egy hatékonyan következtető SMT megoldó áll, amely nyelvére az absztrakt leírás automatikusan lefordítható, illetve a tételbizonyító által szolgáltatott tesztetes logikai leírásának modellé alakítása is automatikusan történik. A módszer előnye, hogy a definiált absztrakt leírást is ellenőrzi, ha logikai ellentmondást találhat benne, akkor az egy ellenpéldával megmutatható, ami alapján a tesztetes leírás könnyen javítható.

A hatékony tesztgenerálás érdekében az ismertett módszer alapján elkészült egy Eclipse rendszerbe integrált keretrendszer, ami rendelkezik egyszerűen használható grafikus felülettel, ezáltal megkönnyíti az absztrakt leírások elkészítését és a kezdeti modell definiálását. A keretrendszer további

előnye, hogy támogatja az iparban alkalmazott gyakori, szabványos formátumokat.

A keretrendszer működését az R3Cop ipari projekttel, mint esettanulmánnyal demonstráltuk. Az esettanulmány egy egyszerűbb példáján keresztül a tesztgenerálás gyakorlati haszna is bemutatásra került.

Hivatkozások

- The Satisfiability Modulo Theories Library (SMT)*, 2013. Július <http://www.smtlib.org/>
- The Object Management Group. *Object Constraint Language*, v2.0, 2006. Május <http://www.omg.org/spec/OCL/2.0/>
- Oszkár Semeráth. 2013. „Validation of Domain Specific Languages” Technical Report, <https://incquery.net/publications/dslvalid>
- Oszkár Semeráth, Ákos Horváth, and Dániel Varró. 2013. „Validation of derived features and well-formedness constraints in dsls.” In Ana Moreira, Bernhard Schätz, Jeff Gray, Antonio Vallecillo, and Peter Clarke, editors, *Model-Driven Engineering Languages and Systems*, volume 8107 of *Lecture Notes in Computer Science*, pages 538–554. Springer Berlin Heidelberg
- B. Beckert, U. Keller, and P. H. Schmitt. 2002. „Translating the Object Constraint Language into first-order predicate logic.” In *Proc of the VERIFY, Workshop at Federated Logic Conferences (FLoC), Copenhagen, Denmark*.
- Jordi Cabot, Robert Clarisó, and Daniel Riera. 2007. „UMLtoCSP: a tool for the formal verification of UML/OCL models using constraint programming.” In *Proc. of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07)*, pages 547–548, New York, NY, USA.
- Manuel Clavel, Marina Egea, and Miguel Angel García de Dios. 2009. „Checking unsatisfiability for OCL constraints.” *ECEASST*, 24.
- Martin Gogolla, Jörn Bohling, and Mark Richters. 2005. „Validating UML and OCL models in USE by automatic snapshot generation.” *Softw. Syst. Model.*, 4(4):386–398.