FUNKCIONÁLIS PROGRAMOK HELYESSÉGE*

VERIFICATION OF FUNCTIONAL PROGRAMS

Horváth Zoltán, hz@inf.elte.hu
Pásztorné Varga Katalin, pkata@inf.elte.hu
Tejfel Máté, matej@inf.elte.hu
Eötvös Loránd Tudományegyetem
Általános Számítástudományi Tanszék

Abstract

We present the background, the structure and the main concepts of the one semester course called „Verification of Functional Programs" offered for informatics PhD students at the Eötvös Loránd University. The main concepts of general purpose verification systems are introduced via examples taken from PVS, Isabelle/HOL. Also dedicated verification tools are covered by the curricula. ErlVer was designed for verification of open distributed systems written in Erlang. Verification tools were developed for pure lazy functional languages too recently. The main topic is application of Sparkle, which is a dedicated proof tool for the programming language Clean. Evaluation of two years teaching experiences is presented in the last section of the paper.

Összefoglaló

A „Funkcionális programok helyessége" című, az ELTE Informatika Doktori Iskola hallgatói számára meghirdetett, egy féléves tantárgy tematikai felépítését mutatjuk be. Az általános célú helyességbizonyító rendszerek alapfogalmait ismertetjük PVS és Isabelle/HOL példákon keresztül. A tananyag részét képezi néhány olyan helyességbizonyító rendszer is, amely egy-egy adott funkcionális nyelven készített program helyességbizonyítását segíti. Erlangban írt nyitott, elosztott rendszerek helyességének bizonyítására szolgál az ErlVer. Az elmúlt években készítettek lusta kiértékelésű funkcionális nyelvhez is helyességbizonyító eszközt. Ezek közül a leginkább említésre méltó a Clean programozási nyelvhez készült Sparkle. Befejezésként értékeljük a tantárgy oktatása során szerzett tapasztalatokat.

* Készült az OTKA T037742 és az OMFB 01548 pályázat támogatásával.

FUNKCIONÁLIS PROGRAMOK HELYESSÉGE*

VERIFICATION OF FUNCTIONAL PROGRAMS

Horváth Zoltán, hz@inf.elte.hu
Pásztorné Varga Katalin, pkata@inf.elte.hu
Tejfel Máté, matej@inf.elte.hu
Eötvös Loránd Tudományegyetem
Általános Számítástudományi Tanszék

1. Bevezetés

A tárgy keretében megvizsgáljuk, hogy jelenleg milyen tételbizonyító eszközök léteznek, ezekben milyen bonyolultságú állítások fogalmazhatóak meg és mennyire használhatóak a mi konkrét céljainkra. Az ML (Meta Language) bizonyítási céllal létrejött funkcionális nyelv. Például az Isabelle, és a Coq olyan ML alapú eszközök, amelyek mohó kiértékelésű funkcionális nyelvű programok esetében függvények statikus tulajdonságainak igazolására, s így bizonyos mértékben a helyesség bizonyítására alkalmasak. Az elmúlt években készítettek lusta kiértékelésű funkcionális nyelvhez is helyességbizonyító eszközt, a Clean alapú Sparkle-t. Továbbra is nyitott kérdés tisztán funkcionális mobil kód helyességének, különösen az interaktív folyamatokat megvalósító funkcionális programkomponensek helyességének vizsgálata. Párhuzamos és elosztott programok esetén temporális logikai tulajdonságokkal írhatjuk le a programok viselkedését. A logika klasszikus tételbizonyító rendszerei közül a Gentzen szekvent kalkulus és a tabló kalkulus, amelyeknek temporális logikára való kiterjesztései megvalósíthatók. Magasabb rendű logikában megfogalmazott állítások bizonyításához ad segíséget például az Isabelle/HOL és a PVS. Egy viszonylag egyszerű lineáris idejű temporális logikai modell a Unity, amelyet HOL-ban is megfogalmaztak. Az imperatív nyelvekhez legközelebb álló Erlang esetében létezik egy verifikációs modell (ErlVer), amely egyes párhuzamos Erlang programok helyességének bizonyítására is alkalmas. Az ErlVer alapja a modális μ-kalkulus egy változata, amely temporális logikai következtetések levonására alkalmas. Haskell nyelven írt programok helyességének garantálását tűzte ki célul a DuckLogic. Részletesebben az Isabelle/HOL-t, a PVS-t, a Sparkle-t és az ErlVer-t vizsgáljuk meg.

2. Általános célú helyességbizonyító rendszerek

Az Isabelle egy általános, interaktív, ML alapú rendszer logikai formalizmusok implementálására. Az Isabelle/HOL ennek egy speciális változata, amely kifejezetten a magasabbrendű logikai kifejezések kezelésére készült. Készítettek az eszközhöz egy kiegészítő könyvtárat is, melynek használatával Unity-ban leírt kifejezéseket közvetlenül is kezelhetünk. Ez azért lényeges, mert az általunk bizonyítani kívánt tulajdonságok pl. Unity-ban is megfogalmazhatóak. Az Isabelle, mint tételbizonyító használata gyakorlatilag azt jelenti, hogy elméleteket adunk meg az Isabelle metanyelvén és a nyelv egy kifejezéséről belátjuk, hogy az elméletnek tétele-e. Leegyszerűsítve, egy elmélet típusok, függvények és tételek gyűjteménye, majdnem olyan, mint egy modul egy programozási nyelvben, vagy mint

* Készült az OTKA T037742 és az OMFB 01548 pályázat támogatásával.

egy specifikáció egy specifikációs nyelvben. Isabelle/HOL-ban ezek közül az elmélet gyakorlatilag mindkettő lehet. A rendszer tartalmaz egy „Main" elméletet, amely az összes olyan alapvető előredefiniált elméletek uniója, mint például az aritmetikai műveletekre, listákra, halmazokra stb. vonatkozó elmélet. A beépített típusokon felül mi is definiálhatunk új típusokat és hozzájuk tartozó függvényeket (pl. konstruktor függvényeket). A bizonyítani kívánt állításokra interaktívan alkalmazhatunk bizonyítási lépéseket, amelyek lehetnek előre definiált szabályok, mint például az egyszerűsítés, átírás, esetszétválasztásos bizonyítás, rekurzió, indukció stb. vagy általunk definiált eljárások.

A PVS többek között tartalmaz egy specifikációs nyelvet és egy tételbizonyító rendszert. A specifikációs nyelv a klasszikus, típusos magasabbrendű logika nyelvével rokon. A nyelvben a beépített típusokon kívül (mint a logikai típus, az egész és valós számok), mi is definiálhatunk típusokat a típuskonstruktorok tartalmazhatnak függvényeket, halmazokat, párokat, felsorolásokat és rekurzívan definiált absztrakt adattípusokat is. A PVS bizonyító rendszere szekvent kalkuluson alapul. Interaktívan hajthatunk végre benne egyszerű következtetési szabályokat, például nulladrendű-, kvantorátírási szabályokat, indukciót, és lineáris aritmetikai döntési eljárásokat. Magasabbszintű bizonyítási stratégiák elérése érdekében a felhasználó által definiált eljárások összekombinálhatóak ezen egyszerű bizonyítási szabályokkal.

```
PVS példa:   eq: THEORY
             BEGIN

                 T : TYPE
                 a  : T
                 f  : [T → T]

         ex1: LEMMA f(f(f(a))) = f(a) IMPLIES f(f(f(f(a))))) = f (a)
    END eq
```

Az fenti példa bizonyítása elvégezhető egy szekvent kalkulus szerinti levezetés egy-egy lépésében kapott szekventre alkalmazva a diszjunktív (szétbontó) egyszerűsítést, és a helyettesítést. Ugyanez elérhető egy olyan összetett szabály alkalmazásával is, amely magában foglalja az összes nulladrendű szabályt.

3. Konkrét funkcionális nyelvekhez készült helyességbizonyító rendszerek

Funkcionális nyelven írt programok helyessége könnyebben bizonyítható, mint a jelenleg általánosan elterjedt imperatív nyelveken írt programoké. Tisztán funkcionális nyelvekben érvényes az ún. hivatkozási átlátszóság, így számos programtulajdonság matematikai indukcióval bizonyítható. A modern funkcionális nyelvek (Haskell, Clean, stb.) a hivatkozási átlátszóság megsértése nélkül le tudják írni bonyolult elosztott rendszerek viselkedését, állapotváltozásait. Ezért ezen rendszerek biztonságossági tulajdonságainak, invariánsainak bizonyítása és ellenőrzése könnyebben elvégezhető.

Az Erlang nem tisztán funkcionális nyelv, az Ericsson cég alkalmazza távközlési rendszerek programozására. Ezen programok megbízhatósága alapvető követelmény, így a nyelvhez készítettek egy dedikált helyességbizonyító rendszert: az ErlVer-t, amely sok szempontból meghaladja más eszközök képességeit. Az ErlVer nyitott, elosztott rendszereket megvalósító Erlang nyelvű programok helyességének bizonyítását is támogatja. Gentzen

stílusú és tabló módszert alkalmazó, kompozícionális helyességbizonyító rendszer. Célvezérelt rendszer, a bizonyítandó állításból visszafelé indulva építi a bizonyítási fát. Az esetlegesen előálló ciklikus következtetési láncokat automatikusan felismeri és megszakítja. A kompozícionalitást az biztosítja, hogy szokásos levezetési szabályokat kiegészíti az összetett rendszerek vágási szabálya, amelynek hipotézise a részrendszerekre vonatkozó feltételeket tartalmaz, a következmény pedig az összetett rendszer tulajdonságait írja le. A bizonyítás során figyelembe veszi az Erlang forrásszöveg által meghatározott állapotátmeneteket, strukturális műveleti szemantikára épülő levezetési szabályokat is alkalmazva. Az ErlVer specifikációs nyelve a nyitott specifikációk megfogalmazására is alkalmas modális μ-kalkulus. A tulajdonságok leírásához használhatóak a legkisebb és legnagyobb fixpont operátorok (μ, ν), a lineáris temporális logika alapműveletei (□, ◊) és az Erlang nyelvre vonatkozó speciális atomi predikátumok (pl. a CSP stílusú aszinkron kommunikációra vonatkozó primitívek (?,!)). A helyességbizonyító nagymértékben automatizált. Egy vezérlő nyelv segítségével a bizonyítási lépések programozhatóak.

A Sparkle (a CPS továbbfejlesztése) Clean programok klasszikus logikában megfogalmazható tulajdonságainak bizonyítására szolgál. Bár jelenleg az eszköz csak elsőrendű logikai szabályok kezelésére képes és nem fogalmazhatunk meg benne olyan magasabb rendű logikai formulákat, mint az előző kettőben, de mivel folyamatos fejlesztés alatt áll remélhetőleg a közeljövőben képes lesz bonyolultabb formulák kezelésére is. Az eszköz nagy előnye, hogy képes tanácsot adni arra, hogy mi legyen a következő bizonyítási lépés és egyszerűbb esetekben (a lépéseket végre is hajtva) képes elvégezni a bizonyítást is.

Sparkle példa: $\forall$ n $\in$ Int, $\forall\alpha$, $\forall$ xs $\in[\alpha]$: n $\neq$ undef $\Rightarrow$ take n xs ++ drop n xs = xs

```
take :: Int ![a] -> [a]
take n [x:xs]
  | n > 0  = [x: take (n-1) xs]
  | otherwise = []
take n []  = []
```

```
drop n [x:xs]
        | n>0      = drop (n-1) xs
    | otherwise = [x:xs]
drop n []   = []
```

A bizonyításhoz a Sparkle 42 taktikája közül nyolcat kell felhasználnunk: az ‘xs’-re alkalmazott strukturális indukciót, a dedukció tétel alkalmazását, redukciót (a definíció helyettesítését), a reflexivitást, az esetszétválasztást, a meghatározottsági szabályt (a meghatározottsággal való ellentmondást), az átírást, és az ellentmondást.

4. Összegzés

Mindegyik vizsgált rendszer képes arra, hogy eltárolja a bizonyítási lépéseket és az eltárolt bizonyítást újra végrehajtsa. Eme lehetőség segítségével elegendő minden bizonyítást csak egyszer végrehajtanunk, a következő alkalommal pedig, amikor szükségünk van ugyanazon állítás bizonyítására elegendő csak az eltárolt lépések végrehajtásával ellenőrizni, hogy az előzőleg elvégzett bizonyítás valóban igazolja-e az állításunkat. Az Isabelle/HOL, a PVS és az ErlVer bonyolult állítások kezelésére is alkalmas, míg a Sparkle jelenleg csak viszonylag egyszerűbb leírások értelmezésére képes. Az ErlVer nagymértékben automatizált, az Isabelle/HOL-ban, és a PVS-ben pedig végrehajthatunk olyan bonyolultabb bizonyítási lépéseket, melyek több egyszerűbb lépést foglalnak magukba és így bizonyos szintű automatizálást érhetünk el (ami egyszerűbb esetekben azt is jelentheti, hogy egyetlen bonyolult bizonyítási lépéssel bebizonyíthatjuk az állítást). A Sparkle rendszer előnye ezzel

szemben egy olyan beépített mechanizmus, amely tanácsot ad a következő lépésre, és egyszerűbb esetekben elvégzi a teljes bizonyítást.

5. Oktatási tapasztalatok

Az ELTE-n funkcionális programok helyessége témában, a 2000/2001-es tanévben, az Informatika Doktori Iskola keretében indult egy olyan tantárgy, amelynek tematikája tartalmazza ezen területhez kapcsolódó alapismereteket. A tárgyat azon doktori iskolás hallgatók veszik fel, akiknek a kutatási témája kapcsolódik az adott tárgykörhöz. A hallgatók kiselőadás formájában adnak számot tudásukról. A megszerzett ismereteket egy kutatási projekt keretében is hasznosítjuk, amelyben elosztott Clean nyelvű alkalmazások helyességének igazolására, mobil Clean kód szemantikai tulajdonságainak meghatározására és futási időben történő ellenőrzésére alkalmas helyességbizonyító rendszer megvalósíthatóságát vizsgáljuk.

6. Irodalomjegyzék

[1] Horváth,Z.-Achten,P.-Kozsik,T.-Plasmeijer,R.: Verification of the Temporal Properties of Dynamic Clean Processes, *Proceedings of the 11th International workshop on the Implementation of Functional Languages, IFL'99, Lochem*, The Netherlands,1999, pp. 203-218.

[2] Mol, de M.-van Eekelen,M.: A Proof Tool Dedicated to Clean, *AGTIVE'99*, Kerkade.

[3] Pil, M.: Dynamic types and type dependent functions, LNCS 1467, pp. 169-185.

[4] Owre, S. - Shankar, N.- Rushby, J. M. - Stringer - Calvert, D.J.W.: PVS System Guide, http://pvs.csl.sri.com/doc/pvs-system-guide.pdf, 2001.

[5] Paulson, C.L.: The Isabelle Reference Manual, http://www.cl.cam.ac.uk/Research/HVG/Isabelle/docs.html, 2001

[6] Andersen, F.-Petersen, K.D.-Pettersson, J.S.: Program Verification using HOL-UNITY. *HUG'93*, LNCS 780. pp. 1-15, 1994.

[7] Andersen, F.: A Theory for UNITY in The Cambridge HOL System, *TFL LD* 1992-9, TFL - Telecommunications Research Laboratory, March 1992.

[8] Rushby, P.-Srivas, M.: Using PVS to Prove Some Theorems of David Parnas, LNCS 780, pp. 163-173, 1994.

[9] Thomas Arts, Mads Dam, Lars-åke Fredlund, Dilian Gurov: System Description: Verification of Distributed Erlang Programs, *Proceedings of 15th International Conference on Automated Deduction*, Lindau, Germany, July 5-10, 1998, LNCS 1421, pp. 38-41, Springer-Verlag, Berlin.