Témavezető neve: **Dr. Horváth Zoltán**

A téma címe: **Elosztott funkcionális programok helyessége**

A kutatás időtartama: 2002-2005.

A pályázat célja egy olyan szoftvertechnológia megalkotása volt, amellyel programok biztonságosabbá tehetők. A programok közül mi a hálózati alkalmazásokra, elosztott programokra összpontosítottunk. Ilyen rendszerekben gyakran előfordul, hogy az egyik komponens kiszolgáltatott helyzetbe kerül, mert egy másik komponenstől hibás adatot vagy programot, mobil kódot kap. Például, ha egy web-böngésző program a hálózatról egy hibás kisegítő alkalmazást (plug-in-t) tölt le, lehetséges, hogy nem tud tovább működni. Biztonsági szempontból kritikus elosztott alkalmazások esetén különösen fontos szempont, hogy bizonyítottan helyes komponensekből épüljön fel a program és azokat helyesen alkalmazzuk. Ha a programkomponensek dinamikusan töltődnek le a világhálóról és futás közben szerkesztjük össze az alkalmazást, akkor a felhasznált kód előzetes ellenőrzése mindenképpen szükséges.

Számos új eszközt, modellt, helyességbizonyítási módszert és programozási nyelvi elemet hoztunk létre: megalkottuk funkcionális programok temporális tulajdonságai leírásának modelljét, kiterjesztettünk egy helyességbizonyító rendszert, altípusok invariánsainak kezelésére és bizonyítására új típuslevezetési rendszert terveztünk, új magasszintű nyelvi elemeket vezettünk be elosztott funkcionális programok leírásához. Megterveztük mobil kód verifikációjának, a bizonyítás hitelesítésének, ill. a kód tulajdonságainak ellenőrzése mellett történő felhasználásának modelljét és el is készült a rendszer prototípusa Clean nyelven írt mobil kód alkalmazásához.

A Clean funckionális nyelv lehetőséget biztosít arra, hogy egy programba futás közben file-ból vagy a hálózaton keresztül dinamikusan beszerkesszünk Clean-ben írt függvényket (konstansokat, ill. kódrészleteket). Ennek nyelvi eszköze a "dynamic". Idegen kód felhasználása azonban biztonsági kockázattal jár. Célunk az volt, hogy a kódrészleteket további, azok viselkedésére vonatkozó szemantikai információkkal is kiegészítsük, és ezen tulajdonságok futásidejű ellenőrzésének problémáját is megoldjuk.

Több helyességbizonyítót is megvizsgáltunk annak érdekében, hogy meghatározzuk azt, hogyan tegyük alkalmassá a Clean dedikált helyességbizonyító eszközét, a Sparkle-t Clean programok temporális tulajdonságainak bizonyítására. A HOL, a HOL/UNITY eszközkészletét Tejfel Máté elemezte. Az ErlVer-t a témavezető vizsgálta. Zsók Viktória a Theorema (Linz, RISC) projektbe kapcsolódott be egy CEEPUS ösztöndíj felhasználásával, a Theorema helyességbizonyítási módszereit tanulmányozta. Az automatikus helyességbizonyítás és a funkcionális programozás hátterében is meglévő átíró rendszerek kapcsolatát vizsgálta Pásztor Katalin. Tejfel Máté és Zsók Viktória doktori hallgatók részt vettek egy olyan nyári iskolán is, ahol modellellenőrzés alkalmazásával vizsgálták programok helyességét.

Ivicsics Mátyás és Várnagy Zoltán a Clean dinamikus típusrendszerének új modelljét dolgozta ki és a projekt számára feltétlenül szükséges szoftverelemeket készített el ERASMUS tanulmányút keretében. Az ő munkájuk eredménye a mobil kód átvitelét biztosító SendDynamic és ReadDynamic program. Vizsgálták a dinamikus szemétgyűjtés lehetőségét is annak érdekében, hogy a már nem használt, szükségtelen mobil kódrészleteket azonosítani és törölni lehessen.

Megterveztük azt a szoftverarchitektúrát, amely alkalmas arra, hogy a mobil kód mellett annak tulajdonságait, valamint a tulajdonságok bizonyítását is eljuttassuk a felhasználóhoz. Kozsik Tamással és Tejfel Mátéval közösen a témavezető kidolgozta az objektum-absztrakció funkcionális programozási modelljét. Ez a modell teszi lehetővé funkcionális programok temporális tulajdonságainak formális megfogalmazását. Két esettanulmányban vizsgáltuk azt, hogy a Clean dedikált helyességbizonyító rendszere hogyan használható absztrakt funkcionális objektumok invariánsainak bizonyítására.

Kidolgoztuk a CPPCC (Certified Proven-Property-Carrying Code) módszert, amelynek az a lényege, hogy a kóddal együtt a felhasználó program a kód tulajdonságait és ezen tulajdonságok ellenőrzésének lehetőségét is megkapja. Az általunk kiterjesztett, temporális tulajdonságok bizonyítására is alkalmassá tett Sparkle segítségével elkészíthetjük a kód tulajdonságainak bizonyítását géppel ellenőrizhető formában, majd a programrészlet tulajdonságait és ezen tulajdonságok meglétére vonatkozó bizonyítás fő lépéseit a kódhoz hozzácsomagoljuk. A Clean kódot és a bizonyítást egy hitelesítő program automatikusan verifikálja, emberi közreműködés nélkül. A kódot felhasználó program már csak a hitelesnek elfogadott tulajdonságokat kapja meg a kóddal együtt. A fogadó oldal az így kapott adatokat felhasználva gyorsan és hatékonyan ellenőrizheti, hogy a kódrészlet valóban rendelkezik-e a megadott és elvárt tulajdonságokkal és ennek megfelelően a részfolyamatok, ill. a teljes rendszer biztonságossági tulajdonságai nem sérülnek-e meg. Daxkobler Károly elkészítette a Sparkle egy olyan módosított változatát, amelynek felhasználásával elkészült a CPPCC modell prototípusa. A prototípus szoftver igazolta, hogy az elképzelés helyes, a modell megvalósítható.

A Sparkle, a Clean helyességbizonyítója egy félautomatikus rendszer, egyes bizonyítási lépésekhez emberi közreműködésre van szükség. A félautomatikus helyességbizonyító rendszer számára a Clean forrásszöveget elemezni kell és a Clean Core változatának megfelelő szintaxisra átalakítani. Az állapotátmenetek és az objektumok meghatározása után a kapott adatokat át kell adni a helyességbizonyító rendszernek. Ezért bővítettük a helyességbizonyító nyelvi elemeit az általunk bevezetett annotációkkal és az invariánsok megadásához szükséges specifikációs elemekkel, illetve az invariánsok bizonyítását segítő levezetési szabállyal (Tejfel Máté. Kozsik Tamás, Horváth Zoltán). Ezen elemek jelentését, a kibővített nyelv szemantikáját is definiáltuk. Meghatároztuk az egyes biztonságossági tulajdonságok leírásának alkalmas programnyelvi eszközeit, a temporális tulajdonságok bizonyításának módszerét, a bizonyítási lépések kódolási technikáját és a futásidejű ellenőrzés módját. A temporális logikai specifikációk megfogalmazását, a specifikációk gépi elemzését, valamint az állapotátmenetek azonosítására alkalmas kiterjesztett Clean forráskód elemzését a Clean dedikált helyességbizonyító rendszere, a Sparkle kiterjesztésével oldottuk meg. A

Sparkle a specifikációk és a program forrásszövegének leírására a Core nyelvet használja. A Core nyelv három részből áll: a Clean magjából, egy egyszerű logikai formulák megfogalmazására alkalmas specifikációs nyelvből, és a bizonyítási lépések leírására alkalmas nyelvi elemekből. Kiterjesztettük a Core mind a három összetevőjét, megadtuk az új nyelvi elemek szintaxis és szemantikai definícióit és módosítottuk a helyességbizonyító szoftvert. A forrásszöveg belső ábrázolásában megjelennek az állapotátmenetek és az absztrakt objektumok, a specifikációs nyelvben lehetővé vált invariánsok és biztonságossági tulajdonságok megfogalmazása, a bizonyítások leírásakor megvalósítottuk a leggyengébb előfeltételek automatikus meghatározását, és az invariánsok bizonyításához szükséges levezetési szabályt is implementáltuk. A modell és az implementáció helyességét és használhatóságát példákon keresztül is bemutattuk. A kiterjesztett helyességbizonyító rendszert a Közép-Európai Funkcionális Programozás Nyári Iskola 60 résztvevője is kipróbálhatta, Clean kódrészletek helyességének bizonyításakor alkalmazta. Az eredményeket lektorált, referált folyóiratcikkben is bemutattuk.

Általában is érdekes feladat funkcionális programok olyan transzformációja, amely a program tulajdonságait megőrzi, az ún. refaktorizáció. A refaktorizáció egy új modelljét fogalmazta meg Diviánszky Péter, Szabóné Nacsa Rozália és a témavezető. A programot egy relációs adatbázisban reprezentáljuk, az egyes transzformációkat a program alkalmas absztrakt nézetein hajtjuk végre, a transzformációk előfeltételeit ellenőrizzük. Az adatbázis alkalmas a program szemantikai tulajdonságainak tárolására is, sőt a helyességbizonyító által előállított bizonyítás tárolására is. Az adatbázisból előállítható a transzformált program forrásszövege és a forrásszöveg olyan változata is, amelyet a helyességbizonyító értelmezni tud. A refaktorizációt támogató felhasználói felület alkalmassá tehető az absztrakt funkcionális objektumok kijelölésének támogatására is.

A Clean fordító belső szerkezetét is elemeztük. Diviánszky Péter két különböző saját változatban is elkészítette a Clean fordítóprogram front-end-jét. Ezeket az eredményeket arra is felhasználtuk, hogy Haskell nyelven írt programokat is elemezzünk. Hegedűs Hajnalka, Simon János és Diviányszky Péter elkészítette a Clean fordító egy olyan változatának prototípusát, amelyik a Haskell modulok fordítására is képes, egyelőre még megszorításokkal. Ezen fordító alkalmas továbbfejlesztésével reményünk van a projekt eredményeinek szélesebb körű felhasználására is. A Clean fordító belső szerkezetének ismerete megkönnyíti azt, hogy a későbbi esetleg Haskell nyelven írt programok forrásszövegét a helyességbizonyítás céljainak megfelelő módon elemezzük.

Funkcionális nyelvek típusrendszerét - altípusok bevezetésével - ki lehet egészíteni típusinvariánsokkal is. A típuslevezetés és a programhelyességbizonyítás kapcsolatát vizsgálta Csörnyei Zoltán. A nyelv típuslevezetési rendszerének altípusokkal történő kiterjesztése egyszerűbb esetekben megoldja típusinvariánsok bizonyítását, ilyen esetekben a fordítóprogram át tudja venni a félautomatikus helyességbizonyító szerepét. Mindezen kutatásokat segítette, hogy meghívásunkra Sjaak Smetsers (Clean fejlesztői csoport, Nijmegeni Egyetem, Hollandia) a uniqueness típusrendszerről és a mobil funkcionális kód elkészítésének Clean nyelvi elemiről tartott a kutatócsoport és más érdeklődők számára előadássorozatot (útjának költségeit ERASMUS együttműködésből fedeztük).

Kozsik Tamás egy jelentésben foglalta össze annak az esettanulmánynak a tanulságait, amelyet Clean programok egy összetett típusinvariánsának mintegy 4000 lépéses bizonyítása során szerzett a Clean félig automatikus helyességbizonyító rendszere, a Sparkle – segítségével. Altípusok és invariánsok összekapcsolásának és altípus annotációk alkalmazásának modelljét Kozsik Tamás dolgozta ki. Az annotációk használhatók függvények elő- és utófeltételeinek, valamint tulajdonságátörökítő jellegének leírására. Elemezte, hogy a funkcionális kódban elhelyezett altípus információkat hogyan lehet egy tételbizonyító rendszer számára felhasználhatóvá tenni, belőlük bizonyítandó tételeket generálni. A bevezetett típusrendszer használatát egy erre a célra tervezett, mintaillesztést, ciklikus adatszerkezeteket, algebrai típusokat, magasabb rendű függvényeket, parametrikus polimorfizmust támogató tisztán funkcionális nyelv keretén belül mutatta be.

Kettős szemantikát rendelt a bevezett altípusjelekhez, megteremtve ezáltal a kapcsolatot a típusellenőrzés és a helyességbizonyítás között. Az altípusjel-változókhoz olyan szemantikát rendelt, amely a Hindley-Milner-féle parametrikus polimorfizmussal rokon polimorfizmust indukál. Az altípusjelek axiomatikus szemantikája alapján (tehát a típusellenőrző rendszerrel) nem bizonyítható altípusjelek megjelölésére „hidd-el-jeleket” vezetett be, melyek segítségével pontosan lokalizálható a programokban, hogy hol van szükség tételbizonyító rendszerrel végzendő helyességbizonyításra. Az adatkonstruktorok dekompozíciós típusában az altípusjelölésekhez más jelentést társított, mint a funkcionális programokban előforduló más típusokban. Így bevezethette az illeszkedő alternatívák fogalmát, mellyel lehetővé vált a mintaillesztéssel definiált függvények pontosabb típusozása. Az altípusjel-változók által indukált polimorfizmus kifejezőerejét egyenlőtlenségek bevezetésével növelte. Lehetőséget biztosított arra, hogy ne csak algebrai típuskonstruktorokhoz, hanem a függvénytér típuskonstruktorhoz is lehessen altípusjeleket kapcsolni. Formalizálta a nyelvet és típusrendszerét. Programok típushelyességét oly módon definiálta, hogy a definíció ne csak az egyszerűsített nyelvhez, hanem más funkcionális nyelvekhez (pl. Clean) is alkalmazható legyen; ezáltal lehetőség nyílik a különböző polimorf altípusjeles nyelvek kifejezőerejének összehasonlítására. Megfogalmazott egy (különböző polimorf altípusjeles nyelvekhez használható) típusellenőrző algoritmust, és belátta, hogy helyes és teljes a típushelyesség definíciójára nézve. Kapcsolatot teremtett a típusrendszer és a tételbizonyító rendszerrel történő helyességbizonyítás között. Bevezetett egy logikai alapú kalkulust, amelyben az altípusjelekkel megfogalmazott programtulajdonságok teljesülése vizsgálható. A kalkulus az altípusjelek denotációs szemantikáján alapszik, és programok parciális helyességének vizsgálatát támogatja. Bebizonyította azon módszer helyességét, hogy a „hidd-el-jelekkel” megjelölt altípusjeleknek megfelelő programtulajdonságokat a kalkulusban, egy tételbizonyító rendszer felhasználásával bizonyítjuk, míg a többi altípusjel bizonyítását a típusellenőrző algoritmusra bízzuk.

Bizonyítottan helyes komponensek elosztott funkcionális programokban történő alkalmazásához olyan új nyelvi eszközöket is megalkottunk, amelyek lehetővé teszik elosztott funkcionális számítások megvalósítását. Zsók Viktória, Hernyák Zoltán és a témavezető tovább vizsgálta Clean programrészek CORBA-n keresztül történő összekapcsolásának lehetőségét. Elosztott generikus funkcionális számítási sémák (szkeletonok) készítésének támogatásához készítettünk új szoftver eszközöket. Megalkottuk a Clean nyelv egy olyan kiterjesztését, amelynek

segítségével funkcionális stílusban, magas absztrakciós szinten írhatunk elosztott funkcionális programot PC klaszterre. A megoldás során egy magas szintű (DClean) és egy alacsonyabb szintű nyelvet (DBox) definiáltunk. Az utóbbit a Petri dobozok modelljét elemezve terveztük meg. A Petri hálókról készült tankönyvfejezetet az oktatásban is használjuk. Mindkét nyelv szintaxisát és szemantikáját is megadtuk. Elkészült mindkét nyelv fordítóprogramja (a DClean programot DBox nyelvre, a DBox kódot Clean programokra fordítjuk) és az elosztott program végrehajtását támogató futtató rendszer. Az eredményeket lektorált, referált folyóiratcikkben is bemutattuk. A Közép-Európai Funkcionális Programozás Nyári Iskola résztvevői kisebb példaalkalmazásokat készítettek DClean, ill. DBox nyelven és ezeket sikeresen futatták is. Hegedűs Hajnalka azt vizsgálta a témavezető irányításával, hogy hogyan lehet ugyanezen cél elérésére a Clean új nyelvi elemét, a dynamic-ot alkalmazni, milyen tulajdonságú middleware szükséges az elosztott programozás támogatásához.

Megvizsgáltuk azt is, hogy elosztott funkcionális programok hogyan valósíthatóak meg Grid környezetben, illetve ebben az esetben milyen szemantikai tulajdonságok (erőforrásigény) megfogalmazására és bizonyítására van szükség.

Diákköri dolgozat is készült a Clean nyelv szematikáját meghatározó gráfátíró rendszerről, illetve egy részletes elemzés a JoCaml nyelvről, különös tekintettel a nyelv mobil funkcionális kód készítését támogató nyelvi elemeire.

Az eredményeket hazai és nemzetközi konferenciákon bemutattuk, folyóirat és konferenciacikkek, dolgozatok (habilitációs tézisek, doktori dolgozat, lektorált technical report, független bíráló által elfogadott tudományos diákköri dolgozat, megvédett diplomamunka stb.), letölthető szoftverek formájában közzé tettük. A projekt honlapjáról (http:/aszt.inf.elte.hu/~fun ver) a dokumentumok és a szoftverek többsége szabadon letölthető. Az eredményeket nyugat-európai egyetemeken, ill. az ELTE „Funkcionális programozás” kutatószemináriuma heti rendszerességgel tartott előadásain is ismertettük.

A kutatási eredmények hatása

Eddig ismertté vált független hivatkozások száma: 5 dolgozatra, 1-1 esetben.

Az eredmények hatására a témavezetőt meghívták a szűkebb szakterület jelentős konferenciasorozatainak programbizottságaiba:

- Implementation and Application of Functional Languages: 2005, 2006;
- Trends in Functional Programming 2006;
- Joint Modular Languages (JMLC) 2006,
- Symposium on Programming Languages and Software Tools 2005.

A kutatócsoport EU pályázati támogatással megrendezte az első

- Central European Functional Programming School-t (CEFP’2005), illetve elnyerte az Implementation and Application of Functional Languages konferencia rendezési jogát 2006-ra.

A funkcionális programozás területén folyó kutatások eredményei részben az oktatásban is megjelennek, ezen oktatási tapasztalatokról számoltunk be az Informatika a Felsőoktatásban'2005 Konferencián.

Fontosabb publikációk száma

Lektorált (elfogadott) dolgozatok száma: 14 db
Referált, lektorált folyóiratcikk: 5 db
Referált kiadványban megjelent cikk: 6 db
Szakkönyvfejezet: 2 db
Sikeres habilitáció: 1 db (témavezető)
Doktori dolgozat: 1 db (benyújtva, szerző: Kozsik Tamás)
OTDK díjazott tudományos diákköri dolgozat: 2 db

Közlésre beküldött (még el nem fogadott) közlemények jegyzéke:

1. Kozsik T.: Altípusjeles típusok. Doktori értekezés. ELTE IK Informatikai Doktori Iskola, Doktori program: Az informatika alapjai és módszertana, 2005. pp. 1-188.
2. Lövei L. - Tejfel M. - Mészáros M. - Horváth Z. - Kozsik T.: Comparing Specification with Proved Properties of Clean Dynamics. CSCS - The Fifth Conference of PhD Students in Computer Science, Szeged, Hungary, June 27-30, 2006.

Elkészült szoftverek

- a CPPCC prototípusa,
- a Sparkle helyességbizonyító bővítése,
- a DClean és a DBox nyelv fordítóprogramja,
- a DBox nyelv programfejlesztői környezete,
- az új altípusrendszer típuslevezetési rendszerét megvalósító fordítóprogram,
- a Clean nyelv refactoring környezet prototípusa,
- a Clean nyelvet egyedi azonosítókkal kiterjesztő nyelvi elemeket implementáló fordító,
- kombinátoros szintaktikus elemző Clean nyelvhez, Clean-C fordító,
- Clean-Haskell fordító prototípusa,
- SendDynamic, ReadDynamic – a mobil kód továbbításához szükséges szoftverkomponensek.