Eötvös Loránd Tudományegyetem
Informatikai Kar
Programozási Nyelvek és
Fordítóprogramok Tanszék

Erlang modul refaktorálás - függvények áthelyezése modulok között

Horpácsi Dániel

Konzulens: Dr. Horváth Zoltán

Budapest, 2008

TARTALOMJEGYZÉK

1. BEVEZETÉS

1.1. Refaktorálás

A programok refaktorálása [1, 2] régi eredetű feladat, amely a forráskódok érthetőségének növelését hivatott elősegíteni a program jelentésének megváltoztatása nélkül. Egyes esetekben a program optimalizálása is elérhető ilyen jellegű módszerekkel. Jelentősége az, hogy a refaktorálással kapott kód által leírt program működésében ekvivalens az eredetivel, tehát a tartalmazott hibákat megtartja, újakat nem hoz létre, ám érthetőbbé, struktúráltabbá válhat a forrásszöveg, ami a továbbfejlesztést és csapatmunkát is nagyban megkönnyítheti.

A refaktorálás elterjedésével [12] manapság már szinte minden fejlesztői környezetben helyet kapnak refaktorálási funkciók, mivel vannak olyan műveletek, kódmódosítások, melyeket refaktorálással lehet a legkönnyebben és a leggyorsabban megvalósítani. Ilyen például, ha a fejlesztő egy változót vagy függvényt szeretne átnevezni anélkül, hogy meg kellene keresnie az összes olyan helyet, ahol a konzisztencia megőrzése miatt a módosítást tovább kellene görgetni. A refaktoráló eszközök összegyűjtik a szükséges információkat, amelyek alapján el tudják végezni az összes kompenzációt a programozó helyett, s ezzel időt és munkát spórolnak meg. (Mindemellett meg kell jegyeznünk, hogy egy tekintélyesebb méretű kompenzációs feladatot már nem is lenne könnyű kézzel elvégezni.)

A modern, divatos imperatív nyelvekhez készült fejlesztői környezetek természetesen tartalmaznak refaktoráló funkciókat, ám refaktorálásra régebbi, vagy esetlegesen kevésbé elterjedt nyelvekben is szükség lehet.

A funkcionális nyelvekhez még nem igazán indult meg az ilyen jellegű támogatások fejlesztése, ám vannak már kutatások etéren. A haskell nyelven írt programok refaktorálására készült a HaRe [13, 14], amely Emacs [15] és Vim [17] szerkesztőkből érhető el, továbbá a Clean nyelvhez is készült már hasonló célokat szolgáló szoftver [3, 4].

1.1.1. Az Erlang refactoring project

A dolgozatom témája egy refaktoráló eszköz, illetve annak egy viszonylag jól elkülöníthető része, s az ezen részfeladat megoldásához kapcsolódó elméleti és gyakorlati elemek. Az eszközt az ELTE hallgatói illetve munkatársai fejlesztik egyetemi project keretein belül.

A project célja egy Erlang nyelven [7, 8] írt és Erlang programokat refaktoráló szoftver készítése, amely ipari méretű és szerkezetű kódot is képes elemezni, manipulálni. Az elemzéshez kapcsolódó funkciók a program írása során rendelkezésemre álltak, nekem a szintaxisgráf módosítását kellett megvalósítanom a feladatomhoz kidolgozott korrekciós lépések által meghatározott módon. Az általunk fejlesztett eszköz Emacs-szerű szerkesztőkből [15, 16] az általunk megadott interfészen, menüből és gyorsbillentyűkkel vezérelhető, erre bővebben a felhasználói dokumentációban fogok kitérni.

1.1.2. A refaktorálás információbázisa

A refaktoráláshoz szükséges információkat valamilyen módon el kell tárolni ahhoz, hogy a kompenzációs lépéseket el tudjuk végezni. Ha egy változót átnevezünk, minden helyen, ahol a változó (fontos, hogy az adott, szóban forgó, nem pedig csupán azonos nevű változó) szerepel, el kell végezni a programszövegben ezt az átnevezést. Gondolnunk kell arra, hogy az átnevezés során nem csupán szöveges helyettesítésről van szó, hiszen a változók hatóköre befolyásolja, melyek azok a programrészek, amelyekben a módosítást eszközölnünk kell.

Látható, hogy a programnak csupán a forráskódjából nem könnyű az információk közvetlen kinyerése, érdemes valamilyen szintaxisgráfot építeni, majd abból gyűjteni a szükséges adatokat. Erre a feladatra használunk lexikális és szintaktikus elemzőket, amelyek felépítik a gráfot a forrásszöveg alapján.

1.1.3. A használt szintaxisgráf

A refaktoráló eszköz rendelkezik lexikális és szintaktikus elemzőkkel, amelyek felépítik a programszöveg szintaxisgráfját, s mindemellett még szemantikus információkkal is kiegészítik azt. A refaktorálási lépések információgyűjtésének megkönnyítésére szemantikus elemző modulok kiegészítik a gráfot szemantikus csúcsokkal és élekkel. Ilyenek például a modul és függvény objektumok, amelyek nélkül a függvényeket definíciójuk alapján kellene megkeresni a

szintaxisgráfban, ami nehezebb lenne és tovább tartana, ám így a moduloktól egyetlen éllel elérhetőek a függvények. Ezen kívül több hasznos tulajdonsággal is bírnak ezek a szemantikus információk, de ezeket később részletesebben is látni fogjuk.

A jelenlegi tárolási mechanizmus az Erlang adatbázis-szolgálatását használja, mnesia [9] táblákban tárolja a szintaxisgráfot és a hozzá tartozó kiegészítéseket. (Korábbi verziókban az információk tárolása MySQL adatbázisban történt [11], ám ez nem volt elég hatékony.)

1.2. A megvalósított transzformáció

Az átnevezéseknél előfordulhatnak sokkal bonyolultabb és komolyabb ellenőrzéseket igénylő refaktorálások, kódtranszformációk is, amelyeket szintén érdemes automatizálni, hisz a fejlesztő munkáját ezzel minimálisra tudjuk csökkenteni a transzformáció elvégzésében.

Szakdolgozatomban a tervezett témánál, a függvények és modulok átnevezésénél bonyolultabb és összetettebb részfeladatot, a függvények modulok közötti áthelyezését valósítottam meg. Ez a transzformáció sokkal több érdekes problémát vet fel, megoldásához a programnyelv mélyebb ismerete szükséges. A függvények modulok közötti áthelyezhetőségével nagyméretű rendszerek, sok blokkot, modult, kódsort tartalmazó programok refaktorálásának egyik legfontosabb transzformációja vált elérhetővé.

2. FELHASZNÁLÓI DOKUMENTÁCIÓ

2.1. Telepítés Windows rendszerre

A refaktoráló eszköz használatához több dologra is szükségünk lesz/lehet:

- Erlang környezet
- Emacs szerkesztő
- a refaktoráló eszköz forráskódja

Erlang

Az Erlang nyelvi környezet ingyenesen letölthető az Erlang honlapról. Telepítés után használható:

1. töltse le a telepítőt a következő címről:
   http://www.erlang.org/download.html
2. futtassa a telepítőt és válasszon vagy hozzon létre az Erlang telepítés számára egy könyvtárat, például: c:\refactorerl\erlang.

Emacs

Az Emacs egy ingyenes, nyílt forráskódú szerkesztő, illetve fejlesztői környezet.

1. Töltse le az Emacs programot a következő címről:
   http://www.gnu.org/software/emacs/
2. a letöltött tömörített állományt tömörítse ki egy alkalmas könyvtárba
3. kicsomagolás után futtassa a bin alkönyvtárban található addpm.exe futtatható állományt, amely bejegyzéseket ír a regisztrációs adatbázisba és létrehozza a parancsikonokat a start menüben

4. indítsa el az Emacs-et, majd válassza az „Options” menü „Save options” elemét, amely létrehozza a konfigurácis fájlokat

5. másolja be a .emacs fájljába a következő sorokat:

   (add-to-list 'load-path "c:/refactorerl/refactorerl/emacs")
   (require 'refactorerl)

6. ha a következő sort is bemásolja, a refactorerl szerver atomatikusan elindul majd, amikor erlang forrásfájlt nyit meg a szerkesztőben:

   (add-hook 'erlang-mode-hook 'refactorerl-mode)

7. másolja a .erlang.cookie fájlt a rendszer felhasználói könyvtárából (általában c:\Documents and Settings\Felhasználói Név) abba a könyvtárba, ahol a .emacs fájl található

8. a refactorerl/emacs könyvtárban lévő refactorerl.el fájlban állítsa be a forráskód elérési útját.

A refaktoráló eszköz forráskódja

Másolja a forrásfájlokat a betöltésre használt könyvtárba, például (c:\refactorerl \refactorerl).

2.2. Telepítés Linux rendszerre

Linux operációs rendszer alá hasonlóan lehet telepíteni az eszközt, mint Windows rendszerre.

Csomagból vagy forrásból fel kell telepíteni az Emacs szerkesztőt, majd a .emacs fájlt be kell állítani a home könyvtárban.

2.3. Minimális rendszerkövetelmények

A szoftver futtatásához szükséges hardveres és szoftveres követelmények:

Hardver

- körülbelül 250 MB lemezterület a szükséges szoftverekre
- kódmennyiségtől függő lemezterület az adatbázisnak

- 256 MB memória

Szoftver

A grafikus, egyszerű használathoz szükség van a fentebb ismertetett programokra és Windows XP/2000/2003/NT vagy Linux operációs rendszerre.

A Macintosh rendszereken is található Emacs-szerű szerkesztő (Aquamacs [16]) és Erlang környezet, így azokon is telepíthető az eszköz, habár részletesen nem ismertettük a Macintosh-on történő telepítés menetét (hasonló az előző rendszereken bemutatott telepítéshez).

A refaktoráló szoftver futtatásához elégséges az is, ha az adott operációs rendszeren támogatott az Erlang, nem feltétlenül szükséges a grafikus interfész. Ebben a megközelítésben használható a szoftver Solaris 2 és SunOS4 rendszereken is.

2.4. A refaktoráló eszköz futtatása

Az első lépés az Emacs szerkesztő elindítása, majd egy Erlang forrásfájl megnyitása. Amennyiben nem állította be a .emacs fájlban a refaktoráló eszköz automatikus betöltését, be kell állítani a refaktóráló módót (Alt+x refactorerl-mode), ezzel elindul a refactorerl szerver. Ezután parancsokkal lehet használni az eszközt: Alt+x refactorerl-'parancs'. A parancsok:

1. quit - leállítja a refactorerl szervert
2. restart - újraindítja a refactorerl szervert
3. add-file - hozzáadja az aktuális fájlt az adatbázishoz
4. drop-file - eldobja az aktuális fájlt az adatbázisból
5. debug-shell - megnyitja a debug shellt
6. move-function - megnyitja a függvényáthelyezés interfészét
7. draw-graph - bekér egy fájlnevet és kirajzolja a szintaxisgráfot a fájlba
8. update-status - frissíti a refactorerl szerver állapotát
9. clean - törli a visszaállítási mentéseket

10. undo - visszavon tranzakció(ka)t

Az Emacs szerkesztőben használható gyorsbillentyűk:

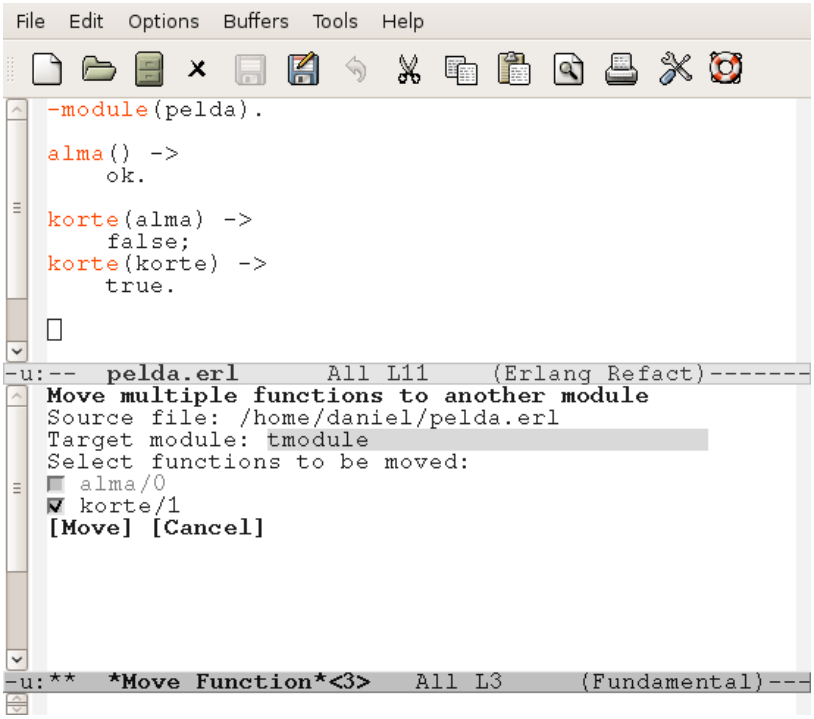
1. Ctrl+c Ctrl+r Q : quit
2. Ctrl+c Ctrl+r R : restart
3. Ctrl+c Ctrl+r a : add-file
4. Ctrl+c Ctrl+r d : drop-file
5. Ctrl+c Ctrl+r D : debug-shell
6. Ctrl+c Ctrl+r M : move-function
7. Ctrl+c Ctrl+r G : draw-graph
8. Ctrl+c Ctrl+r Ctrl+u : update-status
9. Ctrl+c Ctrl+r c : clean
10. Ctrl+c Ctrl+r U : undo

Mielőtt transzformációt végezne, hozzá kell adnia a fájlokat az adatbázishoz!

2.5. Függvényáthelyezés futtatása

A függvényáthelyezés legegyszerűbb módja, ha Emacs szerkesztőben megnyitjuk a forrásfájlt, majd a gyorsbillentyűvel függvényáthelyezést kezdeményezünk. Fontos, hogy a forrásfájlnak és a célfájlnak is az adatbázisban kell lennie, illetve minden olyan fájlnak, amit érint a transzformáció.

Miután megkaptuk a 2.1 ábrán látható beviteli ablakot, nincs más dolgunk, mint kitölteni a célmodul nevét, illetve bejelölni, hogy melyik függvényeket akarjuk áthelyezni. A „Move” gombra kattintva elindul az áthelyezés, majd értesítést kapunk arról, hogy sikerült-e a transzformáció, illetve ha nem sikerült, mi volt az akadálya.



2.1. ábra. Interfész a függvényáthelyezéshez

Amennyiben Erlang shellből szeretnénk futtatni a transzformációt, akkor a `refac_movefun:do/3` függvényt kell meghívnunk a megfelelő paraméterezéssel, amely a fejlesztői dokumentációban megtalálható (3.1.1).

3. FEJLESZTÉSI DOKUMENTÁCIÓ

A fejlesztett programrész több modulból áll össze, amelyek között megtalálhatóak az áthelyezéseket lebonyolító modulok és az ezeket kiszolgáló segédmodulok, melyek exportálva tartalmazzák a szükséges segédfüggvényeket.

A jól elkülöníthető vagy több helyen használt lekérdező függvények, illetve a szintaxisgráfot manipuláló függvények külön modulba kerültek az átláthatóság és továbbfejleszthetőség, újrafelhasználhatóság jegyében.

3.1. refac_movefun modul

A refac_movefun modul tervezése és megvalósítása során figyelembe kellett venni azt, hogy egy függvényáthelyezési művelet nem feltétlenül egyetlen lépésben történik meg, hiszen a művelet végzése közben is felmerülhetnek kérdések, amelyeket felhasználói beavatkozással érdemes megválaszolni. Egy ilyen lehetséges eset, ha az áthelyezés során egy, a mozgatott függvények mellett más függvény által is használt makrót kell kompenzálni. A lehetséges műveletek ekkor:

- a makrót másoljuk a célmodulba, s ezzel kódot duplikálunk
- a makrót kiemeljük egy fejlécfájlba és mindkét modulban include-oljuk (láthatóvá tesszük).

Látható, hogy az is kérdéses, melyik elvet kövessük (természetesen a második logikusabbnak tűnik), s a második esetben az is plusz információ, hogy milyen nevet adjunk az új fejlécfájlnak. Az ilyen esetek miatt a függvényáthelyezést interaktív műveletként terveztem meg, amely kérdezz-felelek stílusban valósul meg a felhasználó és a refaktoráló eszköz között.

Ezt a megoldást az Erlang nyelvben úgy implementáltam, hogy az Erlang OTP (Open Telecom Platform) által kínált generikus állapotátmenetautomatát használtam a kérdés-válasz fázisok elkülönítésére. Ekkor az automata állapotai és az épp aktuális kérdés vagy munka fázisok megfeleltethetőek.

Természetesen az áthelyezés a legtöbb esetben csak egyetlen kérdés-felelet fázisból áll, amelynek során a felhasználó megadja, mely függvényeket, melyik modulból melyik modulba kell áthelyezni. A modul jelenleg csak a fent említett, egy lépéses transzformációt képes végrehajtani, de a leírtak szerint a több fázis is támogatott, csupán implementálni kell a kérdések feltételét és a válaszok interpretálását, illetve felhasználói interfészt kell készíteni hozzá. Ez a jövőbeli fejlesztések egyik célja.

Az egyes állapotokban egy rekord tárolja a szükséges, illetve az összegyűjtött információkat:

- forrásmodul neve
- a célmodul neve
- a mozgatni kívánt függvények név/aritás párokként
- a forrásmodul gráfcsúcs
- a célmodul gráfcsúcs
- a forrásfájl gráfcsúcs
- a célfájl gráfcsúcs
- a mozgatni kívánt függvényobjektumok gráfcsúcsai
- a mozgatni kívánt függvény-formok csúcspontjai
- a makrókról összegyűjtött információk
- a rekordokról összegyűjtött információk
- az esetlegesen felmerült kérdések.

3.1.1. Exportált függvények

A következő függvények a gen_fsm működéséhez szükséges callback függvények, amelyeket definiálni és exportálni kell, jelenleg definíciójuk csak a szükséges minimális alapértelmezett elemeket tartalmazza:

- start_link/0
- init/1

- handle_event/3
- handle_sync_event/4
- handle_info/3
- terminate/3
- code_change/4

start/2. A kezdőállapotból történő állapotváltásra alkalmas függvény.

- Paraméterek:
  1. {`do`, From, FnList, Target} : a do atom, a forrásmodul, a függvénylista és a célmodul négyese
  2. #state{} : az az állapotleíró rekord, amely leírja az automata állapotát (nem használjuk)
- Implementáció:

  A megadott paraméterekkel állapotot vált a `prepare/3` függvény alapján (mivel csak do atomot fogadunk el az n-es első elemeként, emiatt a kezdőállapotból csak a függvényáthelyezés megkezdése lehetséges).

question/2. Az automata a kérdés felmerülése esetén question állapotba kerül, ekkor ezzel a függvénnyel lehet állapotot váltani.

- Paraméterek:
  1. {`answer`, Answers} | `cancel` : pár, amely egy answer atomból és egy válaszokból álló listából tevődik össze, vagy a cancel atom
  2. S = #state{} : az az állapotleíró rekord, amelyben leírja az automata állapotát
- Implementáció:
  - ha az első paraméter válaszokat tartalmaz, akkor állapotváltás következik az `answer/2` függvény alapján.
  - amennyiben az első paraméter cancel, akkor az áthelyezés abortál és start állapotba megyünk át.

do/3. Ez az interfész függvény szolgál a függvényáthelyezés meghívására.

- Paraméterek:
  1. From : a forrásmodul neve atomként vagy stringként
  2. FnList : {név, aritás} párok listája, az áthelyezni kívánt függvényeket adja meg
  3. Target : a célmodul neve atomként vagy stringként
- Implementáció:

  A megadott paramétereket közvetítve a szerver egy olyan eseményt küld el magának, amely elindítja az áthelyezést. Ezt a függvényt tekinthetjük a kívülről hívandó interfészfüggvénynek.

answer/1. Interfész függvény a válaszok elküldésére, amennyiben előzőleg kérdések merültek fel.

- Paraméterek:
  1. Answers : a válaszok listája
- Implementáció:

  A megadott válaszokkal a szerver egy olyan eseményt küld magának, amely elküldi a válaszokat.

cancel/0. Interfész függvény az áthelyezés megszakítására.

- Implementáció:

  A szerver eseményt küld magának a cancel paraméterrel.

test/0. Tesztelő függvény.

- Implementáció:

  Első lépésben kigyűjti azokat a fájlokat az adatbázisból, amelyek több, mint 3 definícióval rendelkező függvényt tartalmaznak. Véletlenszerűen kiválaszt közülük kettőt, s az elsőből kiválaszt szintén véletlenszerűen 4 függvényt, majd meghívja az áthelyezést ezekre a paraméterekre.

3.1.2. Lokális függvények

send_event/1. Egy esemény elküld az általunk indított szervernek.

- Paraméterek:
  1. Event : az esemény, amit el akarunk küldeni magunknak
- Implementáció:

  A `gen_fsm:send_event/2` függvény hívódik meg, s első paraméterként a `?MODULE` makró tartalma, tehát a modul neve adódik át, ezzel a saját szerverünknek küldjük el az eseményt.

next_state/1. A következő állapotba visz át a paraméterként kapott függvény eredményének hatására, hiba esetén abortál.

- Paraméterek:
  1. Fun : a kiértékelendő függvény
- Implementáció:
  - egy kivételkezelő blokkban kiértékeljük a függvényt, amely egy állapotleíró rekorddal tér vissza
  - amennyiben kérdés merült fel, azt megválaszoltatjuk, s question állapotba megyünk át
  - ha nincs kérdés, végrehajtja a transzformációt és kezdőállapotba megy át (start)
  - ha a kirtékelés során kivétel lép fel, akkor azt kiírjuk a felhasználói interfészre.

prepare/3. Feltételek ellenőrzését végző függvény, előkészül az áthelyezésre.

- Paraméterek:
  1. From : a forrásmodul neve atomként vagy stringként
  2. FnList : az áthelyezendő függvények
  3. Target : a célmodul neve atomként vagy stringként

- Implementáció:
  - az ellenőrző függvényeket egy listába gyűjtjük, majd mindegyiket lefuttatjuk az aktuális állapotleíró rekordon
  - a feltételek ellenőrzése során információgyűjtés is történik, amelyet szintén az állapotleíró rekordban tárolunk
  - amennyiben valamilyen feltétel nem teljesül, az adott feltételellenőrző függvény kivételt dob, ami innen továbbdobódik.

answers/2. A válaszok kezelésére szükséges lokális függvény.

- Paraméterek:
  1. St : aktuális állapotleíró rekord
  2. _Answers : a válaszok
- Implementáció:

  Ez a funkció jelenleg nincs implementálva, csupán az állapotleíró rekordot adja vissza.

transform/1. Ez a függvény írja le a transzformációt (a függvény tényleges áthelyezését és a szükséges kompenzációkat).

- Paraméterek:
  1. St : az aktuális állapotleíró rekord
- Implementáció:
  - az áthelyezés megvalósítása előtt és után kompenzációs lépéseket kell végezni
  - meg kell állapítanunk, milyen függvények hivatkoznak a mozgatott függvényekre
  - kompenzáljuk azokat a kifejezéseket, amik hivatkoznak a mozgatott függvények valamelyikére
  - ezek után a mozgatott függvények törzsében lévő függvényalkalmazásokat kompenzáljuk

- majd a makrók és a rekordok kerülnek kompenzálásra
- a korrekciókat végző függvények különböző információkat adnak eredményül listák formájában
- ezt követően a kompenzációs függvények eredményeit összevonjuk, majd csoportosítjuk:
  - fájlmódosítások: ezekre azért van szükség, hogy csak azokat a fájlokat mentsük majd el, amelyeket ténylegesen módosítottunk
  - függvényalkalmazás-módosítások: a transzformáció végén a függvényalkalmazás kifejezéseket töröljük, majd újra hozzáadjuk a gráfhoz
  - listamódosítások: összegyűjtjük, hogy milyen export/import listákat módosítottunk (mind a törléseket, mind a hozzáadásokat nyilván tartjuk)
- ekkor végrehajtjuk a függvény-formok áthelyezését, ami lényegében a szintaxisfa részgráfjainak áthelyezését jelenti
- miután megtörtént az áthelyezés, töröljük, majd újra beszúrjuk a módosított függvényalkalmazásokat (ezeket korábbi kompenzációs lépések alapján ismerjük), ezzel fenntartjuk a gráf szemantikus részének konzisztenciáját: biztosan arra a szemantikus függvényobjektumra fog mutatni a függvényreferencia, amit a kifejezés valóban alkalmaz, illetve modulminősítés esetén biztosan arra a modulobjektumra hivatkozunk, amivel minősítünk
- az export/import lista módosításokat hozzáadások és törlések részekre bontjuk, majd elvégezzük ezek kompenzációját a megfelelő függvényekkel (azért fontos, hogy először csak összegyűjtöttük a szükséges módosításokat, s csak most végezzük el egyben, mert így azokat a listákat, amelyek feleslegessé válnak, törölhetjük, illetve biztosan nem szúrunk be egy listaelemet többször)
- amennyiben minden kompenzációs feladatot végrehajtottunk, elmentjük azokat a fájlokat, amelyeket módosítottunk, majd erről értesítjük a felhasználót is.

check_target_module/1. Ellenőrzi, létezik-e a célmodulként megadott névvel modul az adatbázisban, és ha igen, eltárolja a modul és a modult definiáló fájl gráfbeli csúcspontját, egyébként kivételt dob.

- Paraméterek:
  1. St : az aktuális állapotleíró rekord
- Implementáció:
  - egy kivételkezelő blokkban megpróbáljuk atommá konvertálni az állapotleíró rekord `target` mezőjében lévő információt, ha az nem atom típusú (az adatbázisban atomként vannak tárolva a modulok nevei)
  - megvizsgáljuk, hogy van-e ilyen néven modul az adatbázisban
  - ha van, megkeressük az őt definiáló fájlt, majd a modul- és fájlinformációt módosítjuk az állapotleíró rekordban és visszaadjuk eredményül a frissített állapotleíró rekordot
  - ha nincs adott nevű modul, kivételt dobunk
  - ha nem létezik az adott atom és a konverzió meghiúsul, kivételt dobunk.

check_from_module/1. Ellenőrzi, hogy létezik-e az adatbázisban forrásmodulként megadott névvel modul, és ha igen, eltárolja a modul és a modult definiáló fájl gráfbeli csúcspontját, továbbá a függvény-formokat és függvényobjektumokat is megkeresi és tárolja a megadott név/aritás párok alapján. Ha nem létezik a modul, kivételt dob.

- Paraméterek:
  1. St : az aktuális állapotleíró rekord
- Implementáció:
  - egy kivételkezelő blokkban megpróbáljuk atommá konvertálni az állapotleíró rekord `from` mezőjében lévő információt, ha az nem atom típusú (az adatbázisban atomként vannak tárolva a modulok nevei)
  - megvizsgáljuk, hogy van-e ilyen néven modul az adatbázisban
  - ha van, megkeressük az őt definiáló fájlt, majd a modul- és fájlinformációt módosítjuk az állapotleíró rekordban

  - ezt követően az állapotleíró rekord `fnlist` mezője alapján (amely név/aritás párokat tartalmaz) megkeressük a függvényekhez tartozó formokat és szemantikus függvény objektumokat, majd zokat is eltároljuk a rekordban
  - ha nincs adott nevű modul, kivételt dobunk
  - ha nem létezik az adott atom és a konverzió meghiúsul, kivételt dobunk.

check_funnames/1. Ellenőrzi, léteznek-e a mozgatott függvényekkel azonos nevű és aritású függvények a célmodulban, tehát lesz-e ütközés a mozgatás miatt. Ha igen, kivételt dob, egyébként visszatér az állapotleíró rekorddal.

- Paraméterek:

  1. St : az aktuális állapotleíró rekord

- Implementáció:

  A mozgatott függvényeket megszűri az alapján, hogy okoznak-e ütközést a célmodulban a nevük/aritásuk alapján. Amennyiben akad ilyen ütköző függvény, kivételt dobunk, egyébként eredményül adjuk az állapotleíró rekordot.

check_macros/1. Ellenőrzi az esetleges makrónévütközéseket, illetve a célmodulban láthatóvá tétel feltételeit, s információt tárol az állapotleíró rekordba a mozgatott függvényekben használt makrókról.

- Paraméterek:

  1. St : az aktuális állapotleíró rekord

- Implementáció:

  Általános ellenőrző eljárást használ (`check_entites/4`), felparaméterezve az állapotleíró rekorddal, a `refac_query` modulban implementált makrólekérdező függvényekkel, illetve a `macro` atommal.

check_records/1. Ellenőrzi az esetleges rekordnévütközéseket, illetve a célmodulban láthatóvá tétel feltételeit, s információt tárol az állapotleíró rekordba a mozgatott függvényekben használt rekordokról.

- Paraméterek:
  1. St : az aktuális állapotleíró rekord
- Implementáció:

  Általános ellenőrző eljárást használ (`check_entites/4`), felparaméterezve az állapotleíró rekorddal, a `refac_query` modulban implementált rekordlekérdező függvényekkel, illetve a `record` atommal.

check_entities/4. Általános ellenőrző függvény, amely vizsgálja a makrók és a rekordok létezését, használatát, s amennyiben a feltételek teljesülnek az áthelyezéshez, akkor frissíti az állapotleíró rekordot a felderített információkkal, egyébként kivételt dob.

- Paraméterek:
  1. St : az aktuális állapotleíró rekord
  2. Existing : a létező entitásokat lekérdező függvény
  3. Used : a használt entitásokat lekérdező függvény
  4. Tag : `macro` vagy `record`
- Implementáció:
  - a paraméterként kapott függvényekkel
    - lekérdezzük a lokális és include-olt entitásokat a forrásból
    - lekérdezzük az összes a célmodulban létező entitást
    - összegyűjtjük a mozgatott függvényekben használt entitásokat (neveket és gráfpontokat)
  - vesszük a mozgatott függvényekben használt nevek és a célmodulban létező entitások metszetét
    - ha ez a metszet üres, megnézzük, hogy milyen entitásokat használunk fejlécfájlokból, és megvizsgáljuk, hogy azokat a fejlécfájlokat include-olni tudjuk-e a célmodulban

        - ha igen, frissítjük az állapotleíró rekordot, s ehhez az `update_state_info/5` függvényt használjuk
        - ha nem, kivételt dobunk azzal az információval, hogy melyik entitást melyik fejlécfájl miatt nem tudunk láthatóvá tenni
    * ha a metszet nem üres, megnézzük, hogy van-e a használt entitások között lokálisan definiált
        - ha van, akkor kivételt dobunk, mivel az ütközés biztosan akadálya az áthelyezésnek
        - ha nincs, akkor minden használt entitás fejlécfájlból származik, tehát lehetséges, hogy az ütközés oka az, hogy azonos fejlécfájlok vannak a forrásban és a célban indludeolva
        - vesszük a szükséges és a célmodulban include-olt fejlécfájlok különbségét, s ha ez üres, akkor frissítjük az állapotleíró rekordot az `update_state_info/5` függvénnyel, ellenkező esetben kivételt dobunk.

`correct_references/3`. Korrigálja a paraméterben adott mozgatott függvényre irányuló hivatkozásokat.

- Paraméterek:
    1. Fun : a függvényobjektum, amelynek referenciáit korrigáljuk
    2. Expr : a korrigálandó kifejezés
    3. St : az aktuális állapotleíró rekord
- Implementáció:

  Első lépésként megpróbálunk a gráfban felfelé eljutni két lépésben egy formhoz, hogy megtudjuk, export/import listaelemről van szó, vagy közönséges függvényalkalmazásról

    – amennyiben export listaelem hivatkozik a mozgatott függvényre, megkeressük a gráfban a formot és a listát, lekérjük a függvény nevét és aritását, s eredményül egy listát adunk, melyben elrendeljük a lista törlését forrásmodulból, és egy új export elem beszúrását a függvényre a célmodulban

  - ha import lista hivatkozást találtunk, akkor két esetre bontjuk a korrekciót:
    * ha modulimportról van szó, amennyiben a hivatkozó modul a célmodul, töröljük az importot, egyéb esetben átnevezzük az importot a célmodulra, hisz oda kerül a függvény
    * ha függvényimport hivatkozást korrigálunk, töröljük a régi importot, aztán pedig ha nem a célmodul a hivatkozó modul, lekérdezzük a függvény nevét és aritását, majd létrehozunk egy új függvényimport formot a célmodulra (a hivatkozott függvényre)
  - ha közönséges függvényalkalmazást kell korrigálni, meghívjuk a `correct_module_qulifier/2` függvényt, amely elvégzi a szükséges minősítőkorrekciót.

correct_body/3. Az adott függvény törzsében előforduló függvényalkalmazások korrekcióját végzi.

- Paraméterek:
  1. FunForm : a függvény-form gráfpontja
  2. ImportedFuns : a forrásban importált függvények
  3. St : az aktuális állapotleíró rekord
- Implementáció:

  Lekérdezzük a függvény összes függvényalkalmazó kifejezését, és alkalmazzuk az `analyse_application/3` függvényt a kifejezéssel, az importált függvényekkel és a rekorddal paraméterezve.

analye_application/3. Megvizsgálja, szükség van-e a függvényalkalmazás kompenzálására, s ha igen, elvégzi.

- Paraméterek:
  1. Application : az alkalmazás kifejezés
  2. ImportedFuns : az importált függvények
  3. St : az aktuális állapotleíró rekord

- Implementáció:

  Lekérdezzük, hogy milyen függvényre hivatkozik az alkalmazás, és ha ez a mozgatott függvények közül való, akkor nincs szükség kompenzációra, egyéb esetben viszont meghívjuk a `correct_application/4` függvényt a korrekció elvégzésére.

correct_application/4. Korrigálja az adott függényalkalmazást a mozgatott függvény törzsén belül.

- Paraméterek:

  1. Application : a függvényalkalmazó kifejezés
  2. ReferredFun : az alkalmazott függvény
  3. ImportedFuns : az importált függvények
  4. #state{} : az állapotleíró rekord

- Implementáció:

  Megvizsgáljuk, modulminősített alkalmazásról van-e szó:

  - amennyiben igen, s a célmodulra hivatkozunk, akkor töröljük a modulminősítést
  - ha nincs modulminősítés
    - importált függvény esetén megkeressük az importlistát, majd modul- és függvényimport esetén is importáljuk a hivatkozott függvényt a célmodulból, továbbá függvényimport esetén a régi importot töröljük is
    - ha nincs importálva, megnézzük, van-e definíciója (tehát beépített függvény-e, vagy nem), s ha van, beszúrunk egy modulminősítést a célmodulra, a célmodulban pedig exportáljuk a mozgatott függvényt
  - egyéb esetekben nincs teendő.

correct_module_qualifier/2. Korrigálja az adott függvényalkalmazás minősítését.

- Paraméterek:

1. Expr : a függvényalkalmazás
2. #state{} : az állapotleíró rekord

- Implementáció:

  Megállapítjuk, melyik modulban található a kifejezés, melyik fájl definiálja ezt a modult, és hogy melyik függvényre hivatkozik az alkalmazás. Ha a modul, amelyben a kifejezés van

  – a célmodul, akkor megnézzük, van-e az alkalmazásnak modulminősítése, s ha van a forrásmodulra, akkor töröljük a minősítést

  – a forrásmodul, akkor megvizsgáljuk, hogy a hivatkozott függvény a mozgatottak között van-e, s ha nem, akkor beszúrunk az alkalmazás elé egy minősítőt a célmodulra

  – egyéb modulok esetén amennyiben van modulminősítés a forrásra, akkor frissítjük a modulminősítőt a célmodulra.

insert_module_qualifier/2. Modulminősítőt szúr be az alkalmazás elé, az adott modulra.

- Paraméterek:

  1. Expr : a függvényalkalmazó kifejezés
  2. Mod : a hivatkozott modul

- Implementáció:

  Megnézzük, hogy függvényalkalmazásról van szó, vagy implicit fun expression-ről, s ha a második eset fordul elő a modulminősítés beszúrása előtt átalakítjuk explicitté a fun expression-t. (Ez az Erlang kompatibilitás miatt szükséges, mert a régi verziókban implicit fun expression nem rendelkezhet modulminősítéssel.)

handle_list_addings/1. Beszúrja a paraméterben leírt export vagy import elemet az adott modulba.

- Paraméterek:

  1. #list_add{} : a típust, fájlt, modult, nevet és aritást írja le

- Implementáció:

  A típus (export/import) szerint különböző export/import létrehozó és beszúró függvényeket hív meg.

format_list_removings/1. Átformázza a listatörlésekre vonatkozó információkat úgy, hogy listák szerint csoportosítja.

- Paraméterek:

  1. ListRemoves : a törlendő listaelemek információi rekordokban

- Implementáció:

  Először összegyűjti az érintett listákat, aztán azokhoz rendeli a hozzájuk tartozó kifejezéseket. Ezzel segítjük annak eldöntését, hogy van-e még szükség az adott listára.

filter_removable_lists/3. Az import és export listaelemeket szűri aszerint, hogy törölhetőek-e.

- Paraméterek:

  1. Exprs : a listalemek {típus, kifejezés} alakban
  2. FModule : a forrásmodul
  3. AllApps : a mozgatott függvényekbeli összes alkalmazás

- Implementáció:

  Amennyiben export lista törlése a kérdés, az mindenképp törlendő, viszont ha import listaelemről van szó, csak akkor törölhető, ha nincs már rá hivatkozás, tehát a mozgatott függvényekben lévő függvényalkalmazások lefedik az összes referenciát az importált függvényre.

referers/2. Megadja, mely kifejezések hivatkoznak a forrásmodulban a paraméterként kapott import elem által importált függvényre.

- Paraméterek:

  1. ImportExpr : az import kifejezés, az import lista egy eleme
  2. FModul : a forrásmodul

- Implementáció:

  Megkeressük azokat a kifejezéseket, amelyek hivatkoznak az importált függvényre, majd megszűrjük őket. Csak azokat a kifejezéseket tartjuk meg, amelyek forrásmodulbeli közönséges függvényalkalmazások és nincs modulminősítésük, vagy forrásmodulbeli implicit fun expressionök.

handle_list_removings/4. Kezeli a listatörléseket. Megszűri a kifejezéseket azokra, amelyek törölhetőek, majd elvégzi a kifejezések törlését, sőt, amennyiben a lista megüresedik és nincs már rá szükség, a listát is.

- Paraméterek:

  1. List : az export vagy import lista
  2. Exprs : listaelemek {típus, kifejezés} alakban
  3. FModule : a forrásmodul
  4. FnForms : a mozgatott függvény-formok

- Implementáció:

  Először lekéri a mozgatott függvényekben lévő összes függvényalalmazást, majd megszűri a kifejezéseket törölhetőekre. Eztán

  – amennyiben a lista összes kifejezését lefedik a törölhető kifejezések, az egész listát töröljük
  – egyébként pedig export/import esetekre bontva elvégezzük a listaelemek törlését.

correct_rec_mac/3. Makró/rekord korrekció.

- Paraméterek:

  1. #entity_info{} : az entitás információs rekordja, amely tartalmazza a form gráfpontját, a szemantikus objektum gráfpontot, továbbá azt, hogy törölhető-e az entitás a transzformáció után, és hogy fejlécfájlból jön, vagy lokálisan definiált
  2. #state{} : az állapotleíró rekord
  3. Tag : `macro` vagy `record`

- Implementáció:
  - ha az entitás fejlécfájlban van definiálva, akkor megkeressük a fejlécfájlt és a hozzá tartozó forrásfájlbeli include formot, majd amennyiben az entitás törölhető, akkor áthelyezzük az include formot a célmodulba, ellenkező esetben pedig csak átmásoljuk
  - ha az entitás a forrásmodulban lett definiálva, akkor amennyiben törölhető, a formját áthelyezzük a célmodulba, ellenkező esetben csak másoljuk. A form áthelyezése/másolása a tranzakció befejezésével maga után vonja a szemantikus információk frissülését is.

update_state_info/5. Frissíti az állapotleíró rekordot az entitásokról összegyűjtött információkkal.

- Paraméterek:
  1. St : az állapotleíró rekord
  2. `macro` vagy `record`
  3. UsedNodes : a mozgatott függvényekben használt entitások szemantikus gráfpontjai
  4. FnForms : az áthelyezni kivánt függvények formjainak gráfpontjai
  5. UsedNames : a mozgatott függvényekben használt entitások nevei
- Implementáció:

  A paraméterként kapott információkból az `entity_info/4` segítségével #entity_info{} típusú rekordot hoz létre, majd frissíti vele az állapotleíró rekord megfelelő mezőjét (`macroinfo` vagy `recordinfo`).

include_conflicts/4. Azokat a fájl/entitás párokat adja meg, amelyek nem include-olhatóak a célfájlban.

- Paraméterek:
  1. FromFile : a forrásfájl gráfpontja
  2. UsedIncluded : a használt entitások, amelyek fejlécfájlból származnak

3. TargetFile : a célfájl gráfpontja
4. Tag : `macro` vagy `record`

- Implementáció:
  - készít egy listát (fájl, entitás) párokkal, ahol az adott entitás a vele párban lévő fájlban van definiálva
  - eztán ezt a listát megszűri aszerint, hogy melyek azok a fájlok, amelyeket nem lehet a célfájlban include-olni
  - végül az olvashatóság kedvéért a fájl gráfpontokat a fájlok elérési útjára cseréli a párokban
  - így olyan (elérési út, entitás) párokat ad eredményül, amelyek nem include-olhatóak a célfájlban.

entity_info/4. A paraméterek segítségével elkészíti az entitásra vonatkozó #entity_info{} rekordot, amely tartalmazza a kompenzációhoz szükséges információkat.

- Paraméterek:
  1. Entities : az entitások szamentikus gráfpontjai
  2. FnForms : a mozgatott függvények formjainak gráfpontjai
  3. UsedIncluded : a mozgatott függvényekben használt entitások (vagy makrók, vagy rekordok) nevei, amelyek fejlécfájlból származnak
  4. RefLink : `mref` vagy `recref`
- Implementáció:
  - minden entitást információs rekordra képezünk le
  - az első lépésben azt állítjuk be a rekordban, hogy törölhető lesz-e az entitás a transzformáció végén a forrásfájlból: ezt onnan tudjuk meghatározni, hogy lekérdezzük, milyen függvények hivatkoznak az entitásra, és kivonjuk belőlük a mozgatott függvények halmazát (amennyiben üreshalmazt kapunk, törölhető lesz az entitás)

  - a második lépésben azt állítjuk be, hogy az entitás lokálisan definiált a forrásfájlban, vagy fejlécfájlból származik, s a formot is eltároljuk, ami az entitást definiálja
  - a fenti két lépés kompozíciója leképezés az entitásokról a rájuk vonatkozó információs rekordra, s különböző műveleteket igényel makrók és rekordok esetében.

intersect/2. Két lista metszetét készíti el a kivonás műveletével.

- Paraméterek:
  1. L1 : az egyik lista
  2. L2 : a másik lista
- Implementáció:

  Kivonjuk az első listából a másodikat, majd az eredményt az első listából. Ezzel megkapjuk a metszetüket.

filterpairs/2. Párokat szűr az első elemeik alapján.

- Paraméterek:
  1. List : a párokat tartalmazó lista
  2. Atom : az atom, amit keresünk
- Implementáció:

  Megszűrjuk a listát aszerint, hogy a pár első eleme az adott atom-e. Az eredményben minden pár egyedi lesz.

3.2. refac_manip modul

Ez a modul az általános szintaxisgráf-manipuláló műveleteket megvalósító része a programnak, amely egyelőre csak a függvényáthelyezéshez megvalósított műveleteket tartalmazza, ám feltehetőleg később bővülni fog. Olyan függvényeket találunk benne, amelyek a több helyen használt, jól elkülöníthető, egyéni műveletként megvalósítható szintaxisgráfmódosító utasításcsoportokat és az ezekhez kapcsolódó lekérdezéseket és számításokat tartalmazzák. A függvények különböző részgráfokat törölnek vagy építenek, szigorúan megőrizve a gráf konzisztenciáját.

Mivel a szintaxisgráfot kezelő modul egyelőre csak olyan tranzakciók futtatására képes, amelyekben minden törlés megelőzi a lekérdezéseket és hozzáadásokat, emiatt azon függvényekben, amelyekben szintaktikus törlés található, a törlést mindig megelőzi egy tranzakciólezárási kérés. Így érjük el, hogy ne keletkezzen hibás logikájú tranzakció.

3.2.1. Exportált függvények

create_include_form/2. Include form beszúrását végzi el a megadott fájlba, a megadott elérési úttal. Amennyiben az include-olni kívánt fájl már include-olva volt, nem csinál semmit.

- Paraméterek:
  1. File : a fájl gráfpontja, ahova beszúrunk
  2. Path : a path, amit include-olni akarunk
- Implementáció:
  - amennyiben a lekérdező modul szerint az elérési út már include-olva van az adott fájlban, nem csinál semmit
  - ellenkező esetben létrehozza a megfelelő szintaktikus és lexikális elemeket, majd a hozzájuk tartozó éleket a gráfban, végül lezárja a tranzakciót.

create_import_form/2. Létrehoz és visszaad egy üres függvényimport listát az adott fájlban, a megadott modulra.

- Paraméterek:

1. File : a fájl gráfpontja, ahova beszúrunk
2. Mod : a modul gráfpontja, ahonnan importálni akarunk

- Implementáció:

  Létrehozza a megfelelő szantaktikus és lexikális elemeket, majd létrehozza közöttük az éleket. Eredményül a létrehozott üres import lista kifejezését adja.

add_import/3. Hozzáad egy elemet az import listához.

- Paraméterek:

  1. Import : az importlista, amibe be akarunk szúrni
  2. Name : a függvény neve
  3. Arity : a függvény aritása

- Implementáció:

  Meghívja az `add_expimp/3` függvényt az adott paraméterekkel.

remove_import/2. Kitörli a megadott kifejezést az adott importlistából.

- Paraméterek:

  1. ImportList : az importlista, amiből törlünk
  2. ImportExpr : a kifejezés, amit törlünk

- Implementáció:

  Meghívja az `remove_expimp/2` függvényt az adott paraméterekkel.

remove_import/1. Egy komplett import formot töröl.

- Paraméterek:

  1. ImportForm : az import form gráfpontja

- Implementáció:

  Az egész form részgráfot kitörli a gráfból.

rename_import/2. Átnevezi a függvényimport modul elemét.

- Paraméterek:
  1. ImportForm : az import form gráfpontja
  2. Mod : a modul gráfpontja, amiből importálni akarunk
- Implementáció:

  Lekérdezi a megfelelő módosítandó gráfpontokat, megváltoztatja az őket leíró rekordok megfelelő mezőit, majd frissíti a gráfban is az információkat. Végezetül törli, s újra hozzáadja a formot a gráfhoz a szemantikus konzisztencia végett.

create_export_form/1. Létrehoz egy üres exportlistát az adott fájlban.

- Paraméterek:
  1. File : a fájl gráfpontja, ahol létrehozzuk az exportot
- Implementáció:

  Létrehozza a megfelelő szintaktikus és lexikális elemeket, majd a szükséges éleket is beköti, végezetül visszatér az üres lista kifejezés gráfpontjával.

add_export/3. Hozzáad egy elemet az export lisához.

- Paraméterek:
  1. Export : az export lista gráfpontja, ahova beszúrunk
  2. Name : a függvény neve
  3. Artiy : a függvény aritása
- Implementáció:

  Meghívja az `add_expimp/3` függvényt az adott paraméterekkel.

remove_export/3. Törli az adott név/aritású függvényt az export listából.

- Paraméterek:
  1. ExportList : az export lista gráfpontja, ahonnan törlünk
  2. Name : a függvény neve
  3. Artiy : a függvény aritása
- Implementáció:

  A `remove_export/2` függvényt hívja meg, csak előtte a név/aritás alapján meghatározza a kifejezést, amit törölni kell.

remove_export/2. Töröl egy kifejezést az export listából.

- Paraméterek:
  1. ExportList : az import lista gráfpontja, ahonnan törlünk
  2. ExportExpr : a törölt kifejezés
- Implementáció:

  Meghívja az `remove_expimp/2` függvényt az adott paraméterekkel.

remove_export/1. Töröl egy export elemet.

- Paraméterek:
  1. ExportExpr : a törölt kifejezés
- Implementáció:

  A `remove_export/2` függvényt használja, csak előtte megkeresi a kifejezés alapján a listát, amiből törölni kell.

insert_module_qualifier/2. Az adott modul nevét beszúrja modulminősítőként a kifejezés elé.

- Paraméterek:
  1. ModuleNode : a modul gráfpontja, amivel minősítünk
  2. FunRef : a kifejezés, ami elé beszúrunk
- Implementáció:

  Lezárja az előző tranzakciót, kitörli a szükségtelenné vált részgráfot, majd létrehozza a megfelelő szintaktikus és szemantikus elemeket, s beköti a szükséges éleket. Amennyiben nem képes lekérdezni valamilyen kifejezés első tokenjét, abortál és üzenetet ír a felhasználónak erről.

remove_module_qualifier/1. Törli egy kifejezés elől a modulminősítést.

- Paraméterek:
  1. FunRef : a kifejezés, amit megfosztunk a minősítőtől
- Implementáció:

  Lezárja az előző tranzakciót, törli a modulminősítőhöz tartozó gráfrészeket, majd felépíti a hagyományos függvényalkalmazás részgráfját. Amennyiben nem tudja lekérdezni valamelyik kifejezés első tokenjét, abortál és üzenetet ír erről a felhasználónak.

update_module_qualifier/2. Frissíti a kifejezés modulminősítőjét egy másik modulnévre.

- Paraméterek:
  1. FunRef : a kifejezés, amit módosítunk
  2. ModuleNode : a modul gráfpontja, amire frissítjük a minősítőt
- Implementáció:

  Először kitörli a minősítőt a `remove_module_qualifier/1` segítségével, majd beszúr egy újat az `insert_module_qualifier/2` függvénnyel.

extract_funexpr/1. Implicit fun expressiont alakít át explicitté.

- Paraméterek:
  1. FunExpr : az implicit fun expression kifejezésének gráfpontja
- Implementáció:
  - törli az implicit fun expression-höz tartozó részgráfot
  - létrehozza az explicit fun expression szignatúrája változóneveinek gráfpontjait (`get_var_nodes/3`)
  - létrehozza a paramétereket a `create_parameters/1` függvénnyel
  - létrehozza az alkalmazás név kifejezését (`create_name/1`)
  - a paraméterekből és a névből létrehozza a függvényalkalmazást a `create_application/2` segítségével
  - a már ismert adatokból létrehozza a klózt (`create_funclause/3`)
  - végül a `refac_token` modul függvényeivel helyreállítja a megfelelő lexikális szintű éleket.

move_form/3. Áthelyezi a formot egyik fájlból a másikba.

- Paraméterek:
  1. Form : a mozgatott form gráfpontja
  2. From : a forrásfájl
  3. To : a célfájl
- Implementáció:

  Lezárja az előző tranzakciót, majd törli a forrásfájlból a formot, s beszúrja a célfájlba.

move_include/3. Áthelyezi az include formot az egyik fájlból a másikba. Csak akkor szúr be, ha még nem volt include-olva.

- Paraméterek:
  1. Form : a mozgatott form gráfpontja

2. From : a forrásfájl
3. To : a célfájl

- Implementáció:

  Lezárja az előző tranzakciót, majd törli a forrásfájlból az include formot. Amennyiben még nincs include-olva a célfájlban, beszúrja az `insert_form/2` függvénnyel.

copy_include/2. Átmásolja az include formot az egyik fájlból a másikba. Csak akkor szúr be, ha még nem volt include-olva.

- Paraméterek:

  1. Form : a másolt form gráfpontja
  2. To : a célfájl

- Implementáció:

  Lezárja az előző tranzakciót, majd ellenőrzi, van-e include-olva a célfájlban a fájl. Ha nincs, beszúr egy include formot az `insert_form/2` függvénnyel.

create_and_add_export/3. Ha nincs exportálva az név/aritás pár, létrehoz egy üres export listát, majd hozzáadja a név/aritás párt a listához.

- Paraméterek:

  1. File : a célfájl gráfpontja
  2. Name : a függvény neve
  3. Artiy : a függvény aritása

- Implementáció:

  Megvizsgálja, exportálva van-e az adott név/aritású függvény a fájlban, s amennyiben nincs, akkor létrehoz egy új, üres export listát a `create_export_form/1` segítségével, s hozzáadja a név/aritás párt az `add_export/3` függvénnyel.

create_and_add_import/3. Ha nincs importálva a név/aritás pár, létrehoz egy üres import listát és hozzáadja a név/aritás párt a listához.

- Paraméterek:
  1. File : a célfájl gráfpontja
  2. Module : a modul gráfpontja, ahonnan importálunk
  3. Name : a függvény neve
  4. Artiy : a függvény aritása
- Implementáció:

  Megvizsgálja, importálva van-e az adott modulból az adott név/aritású függvény a fájlban, s amennyiben nincs, létrehoz egy üres import listát a `create_import_form/2` segítségével, s hozzáadja a név/aritás párt az `add_import/3` függvénnyel.

insert_form/2. A megadott fájlba beszúja az adott formot, méghozzá a meghatározott sorrend szerinti helyre.

- Paraméterek:
  1. File : a fájl gráfpont, ahova beszúrunk
  2. Form : a form gráfpontja
- Implementáció:

  Megállapítja, milyen indexre kell beszúrni, majd beszúrja a formot.

insert_application_again/1. Egy függvényalkalmazó kifejezést kitöröl, majd újra hozzáad a gráfhoz.

- Paraméterek:
  1. Funref : a kifejezés, amit újra hozzáadunk
- Implementáció:

  Lezárja az előző tranzakciót, majd a kifejezésnek megkeresi a szülőjét a gráfban. Törli, majd újra hozzáadja a kifejezést, s emiatt a szemantikus elemző modulok elemzik, ekkor helyreállítják a szemantikus részét a gráfnak.

3.2.2. Lokális függvények

antecedents/1. Megadja, milyen formtípusok előzik meg a paraméterként kapott formtípust.

- Paraméterek:
  1. export | import | macro | record | include
- Implementáció:

  Mintaillesztés segítségével különböző típusok esetén más-más lista az eredmény, amely azokat a típusokat tartalmazza atomokként, amelyek megelőzik az adott típust. A sorrend: export, import, macro, record, include.

insert_pos/2. Megadja, milyen pozícióra kell beszúrni az adott formtípust.

- Paraméterek:
  1. File : a fájl gráfpontja, ahol keressük a pozíciót
  2. Antecedents : a megelőző típusok
- Implementáció:

  Ha nincs megelőző típus, 2-vel tér vissza, ha van, akkor meghívja a `pos/3` függvényt.

pos/3. Megadja a beszúrandó elem indexét.

- Paraméterek:
  1. form akkumulátor
  2. Index : hányas indexnél tartunk
  3. Antecedents : megelőző típusok
- Implementáció:

  Adott a formok listája. A megelőző típusú formokon átugrunk és növeljük az indexet, ha pedig már nincs megelőző típusú form, visszadjuk az indexet. Rekurzív függvény, az alapesetben üres a formlista.

esg_close/0. Lezárja az aktuális szintaxisgráfmódosító tranzakciót.

- Implementáció:

  Amennyiben üres volt a tranzakció, elfedi a hibaüzenetet.

create_next_links/1. Létrehozza a tokenek között a next éleket.

- Paraméterek:

  1. TokenList : a tokenek gráfpontjainak listája

- Implementáció:

  Párosítja a tokeneket egymással, és létrehozza a rákövetkezést jelentő next éleket.

add_expimp/3. Export vagy import listához ad hozzá egy elemet.

- Paraméterek:

  1. List : az export/import lista gráfpontja
  2. Name : a függvény neve
  3. Arity : a függvény aritása

- Implementáció:

  Annak függvényében, hogy van-e már elem a listában, különböző gráfműveletek hajt végre, hogy beszúrható legyen az új elem, majd meghívja az `add_expimp_to_list/5` függvényt.

add_expimp_to_list/5. Hozzáadja az adott név/aritású elemet az export vagy import listához.

- Paraméterek:

  1. List : az export/import lista gráfpontja
  2. Name : a függvény neve
  3. Arity : a függvény aritása
  4. Prev : a megelőző token gráfpontja

5. Next : a rákövetkező token gráfpontja

- Implementáció:

  Létrehozza a megfelelő szintaktikus és lexikális elemeket, majd beköti a szükséges gráféleket is.

remove_expimp/2. Töröl egy elemet (kifejezést) az export/import listából.

- Paraméterek:

  1. List : az export/import lista gráfpontja
  2. Expr : a törlendő kifejezés

- Implementáció:

  Új tranzakciót nyit, lekérdezi a szükséges adatokat, majd törli a megfelelő éleket, illetve új élt is létrehoz.

insert_form_again/1. Törli, majd újra hozzáadja az adott formot a gráfhoz.

- Paraméterek:

  1. Form : a form, amit újra hozzáadunk

- Implementáció:

  Lezárja az előző tranzakciót, majd lekéri a form szülőjét a gráfban. Törli, majd újra hozzáadja a formot, s ezzel eléri, hogy a szemantikus elemző modulok létrehozzák a szemantikus éleket.

funref_parent_link/1. Megkeresi a kifejezés szülőjét a gráfban, illetve a közöttük lévő él indexét.

- Paraméterek:

  1. FunRef : a kifejezés

- Implementáció:

  A szintaxisséma éltípusai alapján megkeresi a szülőt, majd megkeresi az él indexét, s visszatér velük.

create_lexical_token/4. Létrehoz egy megfelelő típusú és tartalmú lexikális tokent.

- Paraméterek:
  1. Type : a token típusa
  2. Text : a szöveges reprezentáció
  3. Prews : a megelőző whitespace-ek
  4. Postws : a követő whitespace-ek
- Implementáció:

  A típustól függően különböző adattagot ír a tokenhez, létrehozza a gráfban a csúcsot és eredményül adja.

create_expr_and_token/2. Létrehoz megfelelő összetartozó kifejezés-token párt.

- Paraméterek:
  1. Type : a kifejezés/token típusa
  2. Name : a név/szöveg
- Implementáció:

  A `create_expr_and_token/3` függvényt hívja meg, ám előtte listává alakítja a nevet, s ezt adja át szöveges reprezentációként.

create_expr_and_token/3. Létrehoz megfelelő összetartozó kifejezés-token párt.

- Paraméterek:
  1. Type : a típus
  2. Text : a szöveges reprezentáció
  3. Value : az érték
- Implementáció:

  A típus függvényében létrehozza a megfelelő szintaktikus és lexikális elemet, majd beköti a lexikális éleket közöttük.

copy_prews/2. Átmásolja a megelőző whitespace-eket az egyik tokentől a másikhoz.

- Paraméterek:
  1. From: melyik tokentől
  2. To : melyik tokenhez
- Implementáció:

  A megfelelő adattagokat módosítja, majd frissíti a gráfban.

move_prews/2. Áthelyezi a megelőző whitespace-eket az egyik tokentől a másikhoz. Az elsőnek így nem lesz megelőző whitespace-e.

- Paraméterek:
  1. From: melyik tokentől
  2. To : melyik tokenhez
- Implementáció:

  A megfelelő adattagokat módosítja, majd frissíti a gráfban.

remove_short_funexpr/1. Törli az implicit fun expression részgráfját.

- Paraméterek:
  1. FunExpr : az implicit fun expression
- Implementáció:

  Lezárja az aktuális tranzakciót, összegyűjti a szükséges információkat, és törli a szükségtelenné vált részgráfo(ka)t, majd eredményül a használt név/aritás párt adja.

kill_token/1. Törli a tokenhez kapcsolódó éleket.

- Paraméterek:
  1. Token : a törölt token
- Implementáció:

  Törli a tokenből és tokenbe vezető lexikális éleket.

get_var_nodes/3. Létrehozza a szignatúrához szükséges változók gráfpontjait.

- Paraméterek:
  1. Prefix : az új változók neveinek prefixe
  2. Count : hány változóra van szükség
  3. List : a készen lévő pontok listája
- Implementáció:

  A `get_var_names/3` függvénnyel lekéri a változóneveket, majd létrehozza belőlük a gráfpontokat.

get_var_names/3. Adott számú, adott prefix-szel rendelkező változónevet ad vissza.

- Paraméterek:
  1. Prefix : az új változók nevének prefixe
  2. Count : hány van még, hányadiknál tartunk
  3. List : a készen lévő nevek listája
- Implementáció:

  Rekurzív függvény, a prefixhez konkatenálja 1-től n-ig a számokat.

create_funclause/3. Létrehoz egy függvényklózt.

- Paraméterek:
  1. Parent : a leendő szülő
  2. FunPatternNodes : a klóz szignatúrájának elemei
  3. ExprList : a klóz kifejezései
- Implementáció:

  Felépíti a megfelelő szintaktikus részgráfot, majd visszadja a klóz gráfpontját.

create_application/2. Létrehoz egy függvényalkalmazást.

- Paraméterek:
  1. PatternNodes : a szignatúra elemei
  2. NameNode : az alkalmazás névkifejezése
- Implementáció:

  Felépíti a megfelelő szintaktikus struktúrát, majd visszatér az alkalmazás gráfpontjával.

create_name/1. Létrehoz egy név kifejezést a gráfban.

- Paraméterek:
  1. Name : a név
- Implementáció:

  Létrehoz egy név kifejezést és eredményül adja.

create_parameters/1. A nevekből létrehozza a paraméterek kifejezéseit a gráfban.

- Paraméterek:
  1. Names : a paraméterek
- Implementáció:

  A nevek listáját leképezi a belőlük létrehozott paraméterek kifejezéseire.

3.3. refac_query modul

A refaktoráló transzformációknak a feltételek ellenőrzéséhez, illetve a kompenzációs lépések elvégzéséhez olyan információkra lehet szüksége, amelyet a szintaxisgráfból lehet kinyerni. Ha az információk lekérdezése több lépésből áll, illetve több helyen is használt művelet, akkor érdemes elkülöníteni a transzformáció kódjától és külön modulba helyezni. A `refac_query` modul tartalmazza azokat a lekérdező függvényeket, amelyeket a transzformációk a gráfból történő információgyűjtésre használnak.

3.3.1. Exportált függvények

is_exported/3. Megadja, hogy az adott entitásban (modulban vagy fájlban) exportálva van-e az adott névvel/aritással függvény.

- Paraméterek:
  1. Entity : a fájl vagy modul
  2. Name : a függvény neve
  3. Arity : a függvény aritása
- Implementáció:

  A `get_mod/1` függvénnyel az entitást biztosan modullá alakítjuk, megkeressük benne az adott névvel/aritással rendelkező függvényt, majd megnézzük, hogy ez a függvényobjektum szerepel-e a modul exportált függvényei között. Amennyiben nem szerepel, vagy nem is található meg a függvény a modulban (esetleg a modult sem találtuk), hamis a válasz, különben igen.

is_imported/4. Megadja, hogy az adott modulból adott névvel/aritással rendelkező függvény importálva van-e az entitásban (fájlban vagy modulban).

- Paraméterek:
  1. Entity : a fájl vagy modul
  2. ModuleName : a modul, ahonnan keressük az importot
  3. Name : a függvény neve
  4. Arity : a függvény aritása

- Implementáció:

  Az implementáció hasonló az `is_exported/3` függvényéhez, azzal a különbséggel, hogy a függvényobjektumot itt abban a modulban keressük meg, ahol az exportálva van, tehát a `ModuleName` nevű modulban.

is_included/2. Megadja, hogy a paraméterként kapott fájl vagy elérési út include-olva van-e az adott fájlban.

- Paraméterek:

  1. File : a fájl, ahol keresünk
  2. Data : az elérési út vagy fájl, amit keresünk

- Implementáció:

  Esetszétválasztással, amennyiben elérési utat keresünk, akkor `include` éleken keresünk olyan fájlokat, amiknek az adott elérési útja van, ha pedig fájlt keresünk, akkor ugyanezeken az éleken fájlobjektum egyezést keresünk.

is_header_file/1. Megadja egy fájlról, hogy fejlécfájl-e.

- Paraméterek:

  1. File : a fájl

- Implementáció:

  Megvizsgálja, hogy a fájl definiál-e modult. Amennyiben igen, akkor nem lehet fejlécfájl.

function_forms_and_nodes/2. Név/aritás párok listájából képez le függvény-formok és függvényobjektumok listájának párjára.

- Paraméterek:

  1. Module : a modul, amiben a függvények vannak
  2. FnList : a függvények listája {név, aritás} formátumban

- Implementáció:

  Első lépésben a szamentikus függvényobjektumokat keressük meg a gráfban, majd ezekből a `fundef` éleken megkeressük a definíciójuk formjait, s visszadjuk a listák párját.

function_exists/3. Megadja, létezik-e adott modulban adott név/aritás párral függvény.

- Paraméterek:

  1. ModuleNode : a modul gráfpontja
  2. Name : a függvény neve
  3. Arity : a függvény aritása

- Implementáció:

  Lekéri a gráfból a modulhoz tartozó, adott névvel/aritással rendelkező függvényeket. Amennyiben nincs ilyen, hamis válasszal tér vissza, egyébként igazzal.

applications/1. Megadja egy függvény-formlista függvényeinek összes alkalmazáskifejezését.

- Paraméterek:

  1. FnForms : a függvények formjai

- Implementáció:

  A gráfból lekérdezi az összes kifejezést, amelyet tartalmaznak a formok, majd megszűri őket, s csak azokat hagyja benne a listában, amelyek függvényalkalmazások vagy fun expression-ök.

referer_importlist/2. Megadja, hogy adott importált függvényt melyik importlista importál az adott fájlban.

- Paraméterek:

  1. File : a fájl, ahol importálunk
  2. FunNode : a függvényobjektum, amit importálunk

- Implementáció:
  - megkeresi a formokat, amelyek hivatkoznak a függvényre
  - ezek közül az import formokat adja vissza, s a függvény-, illetve modulimportok mellé még plusz információt is ad (az import form, esetleg az import lista, a forrásmodul). Ha nem talál import referenciát, üres litát ad vissza.

included_from_file/3. Megadja, melyik fájlból származik az adott entitás (makró vagy rekord).

- Paraméterek:
  1. File : a fájl, ahol használjuk (include-olva van)
  2. Name : az entitás neve
  3. Tag : `macro` vagy `record`
- Implementáció:

  Megkeresi az entitás szemantikus objektumát, majd abból jut el a definiáló fájlhoz.

included_from_path/3. Megadja, hogy milyen elérési úton található a fájl, amiből származik az adott entitás (makró vagy rekord).

- Paraméterek:
  1. File : a fájl, ahol használjuk (include-olva van)
  2. Name : az entitás neve
  3. Tag : `macro` vagy `record`
- Implementáció:

  Az `included_from_file/3` függvénytől lekéri a fájlt, majd a gráfból lekéri a fájl elérési útját.

has_module_qualifier/1. Megadja, hogy rendelkezik-e az adott kifejezés modulminősítéssel.

- Paraméterek:
  1. Expr : a kifejezés
- Implementáció:

  Amennyiben fun expressionről, vagy egyéb nem alkalmazás kifejezésről van szó, hamis a válasz. Egyébként megnézzük a gráfban, hogy `module_qualifier` típusú-e a kifejezés gyereke. Ha igen, akkor van modulminősítője.

existing_recordnames/1. Megadja az adott fájlban található rekordok neveit.

- Paraméterek:
  1. File : a fájl
- Implementáció:

  Az `existing_names/3` függvényt használja a nevek lekérdezésére.

existing_macronames/1. Megadja az adott fájlban található makrók neveit.

- Paraméterek:
  1. File : a fájl
- Implementáció:

  Az `existing_names/3` függvényt használja a nevek lekérdezésére, majd a `MODULE` makrót kiveszi az eredményből.

includable/2. Megadja, hogy az adott fájlba include-olható-e az adott fejlécfájl.

- Paraméterek:
  1. Target : a célfájl

2. Incl : a fejlécfájl

- Implementáció:

  Lekérdezi a forrásban és a fejlécfájlban lévő rekordokat és makrókat, azok neveit, forrásukkal együtt (ahol definiálták őket), s amennyiben nincs metszet, akkor nem lesz ütközés az include-olás esetén.

used_records/1. Megadja a függvény-formokban használt rekordokat és azok neveit.

- Paraméterek:

  1. Funs : a függvény-formok

- Implementáció:

  A gráfból lekérdezzük a megfelelő éleken keresztül a formokban használt rekord objektumokat, majd lekérdezzük azok neveit, s a gráfpontok és nevek listájának párja lesz a függvény eredménye.

used_macros/1. Megadja a függvény-formokban használt makrókat és azok neveit.

- Paraméterek:

  1. Funs : a függvény-formok

- Implementáció:

  A gráfból lekérdezzük a megfelelő éleken keresztül a formokban használt makró objektumokat, majd megkeressük azokat a makrókat is, amiket az adott makrók használnak, s így tovább rekurzívan. Kivonjuk belőlük a `MODULE` makrót, majd lekérdezzük a makrók neveit, s a gráfpontok és nevek listájának párja lesz a függvény eredménye.

macros_by_macro/1. Megadja az adott makró által használt makrókat.

- Paraméterek:

  1. Macro : a makró, ami által használt makrókat keresünk

- Implementáció:

  A gráfban `mref` éleken visszalépünk, s amennyiben találunk elemet, a `macros_by_macro_recur/2` függvénnyel megkeressük az összes makróhelyettesítést. Végül a makróhelyettesítések alapján megkeressük a hivatkozott makrókat (az eredényben minden makró különböző lesz).

records_by_macro/1. Makrók által használt rekordokat keres.

- Paraméterek:

  1. Macro : a makró, ami hivatkozhat rekordokra

- Implementáció:

  Megkeressük a makró által használt makrókat, majd az összes makró összes használati helyét. Ezek lexikális elemeiből keresünk rekordhivatkozásokat, s végül eredményül adjuk a hivatkozott rekordobjektumokat.

macro_users/1. Megadja, milyen makrók használják az adott makrót.

- Paraméterek:

  1. Macro : a makró, amit használhatnak

- Implementáció:

  A `macros_by_macro_recur/2` függvény segítségével keressük meg azokat a makrókat, amelyek használják az adott makrót. (Először itt is makróhelyettesítéseket keresünk, s csak aztán határozzuk meg belőlük a makróobjektumokat.)

first_token/1. Megadja egy kifejezés első tokenjét.

- Paraméterek:

  1. Expr : a kifejezés

- Implementáció:

  Meghívja a `first_token/2` függvényt.

3.3.2. Lokális függvények

get_mod/1. Modul lekérdezése fájl vagy modul esetén.

- Paraméterek:
  1. Entity : az entitás (modul vagy fájl)
- Implementáció:

  Ha az entitás modul, akkor visszaadjuk, ha fájl, lekérdezzük belőle a definiált modult, egyébként kivételt dobunk.

existing_names/3. Az adott fájlban található entitások neveit adja meg.

- Paraméterek:
  1. File : a fájl, ahol keresünk
  2. Type : a keresett entitás típusa
  3. Fun : függvény, amivel lekérhetjük a szemantikus objektumból az entitás nevét
- Implementáció:

  Lekérdezzük az összes, majd a lokálisan definiált entitásokat, s azokat leképezzük a neveikre. Eztán az összesből kivonjuk a lokálisalat, így megkapjuk a fejlécfájlból származókat. A lokális, include és összes entitást tartalmazó hármas a függvény eredménye.

existing_macros_with_source/1. Megadja az adott fájlban látható makrók neveit azzal a fájllal együtt, ahol definiálva lettek.

- Paraméterek:
  1. File : a fájl
- Implementáció:

  Az `existing_names_with_source/3` függvényt használja a lekérdezésre.

existing_records_with_source/1. Megadja az adott fájlban látható rekordok neveit azzal a fájllal együtt, ahol definiálva lettek.

- Paraméterek:
  1. File : a fájl
- Implementáció:

  Az `existing_names_with_source/3` függvényt használja a lekérdezésre.

existing_names_with_source/3. Megadja az adott fájlban látható entitások neveit azzal a fájllal együtt, ahol definiálva lettek.

- Paraméterek:
  1. File : a fájl
  2. Type : az entitás típusa
  3. Fun : a függvény, amivel lekérhetjük a szemantikus objektumból az entitás nevét
- Implementáció:

  Rekurzív függvény, addig megy, amíg talál include-ból származó entitást. Veszi a fájlban található entitásokat a fájllal együtt, majd meghívja magát a fájlból include-olt fájlokra, s ezek eredményét összegzi.

macros_by_macro_recur/2. Adott makróhelyettesítésen belül makróhelyettesítéseket keres.

- Paraméterek:
  1. Subst : makróhelyettesítés
  2. Link : az él, amelyen keresztül keresünk
- Implementáció:

  Adott gráfpontból adott élen megpróbálunk lépni, s ha `subst` típusú lexikális elemet találunk (makróhelyettesítés), akkor hozzávesszük az addigiakhoz, s azokból is megpróbálunk helyettesítést keresni rekurzívan.

first_token/2. Megadja egy kifejezés első tokenjét az adott élen haladva.

- Paraméterek:
  1. Entitiy : az entitás, amit épp vizsgálunk (kezdetben egy kifejezés)
  2. Link : az él, amin megyünk
- Implementáció:

  Lexikális elemet keresünk az entitásból az adott élen. Amennyiben tokent találunk, visszadjuk, egyébként rekurzívan tovább keresünk. Ha nem támogatott típusú lexikális elemet találunk, kivételt dobunk.

3.4. Tesztesetek

A transzformáció működésének tesztelésére megfelelő tartalmú és szerkezetű fájlcsoportokat használhatunk, amelyek alkalmasak arra, hogy kiderüljön, megfelelően működik-e az áthelyezés. Amennyiben a módosított fájlok által kapott program ugyanazt jelenti, mint a kiindulási program, akkor a transzformáció sikeres.

Az áthelyezés működésének bemutatására a következő teszteseteket dokumentáltam, melyekből egyfelől kiderül, milyen jellegű a transzformáció, másrészt rávilágítanak a refaktorálási művelet azon kompenzációs lépéseire, amelyekből érezhető az erőssége.

Természetesen a gyakorlatban sokkal nagyobb és bonyolultabb kódokon használatos ez a funkció, s úgy lett megtervezve, hogy ipari kódon is jól teljesítsen.

Megpróbáltam a fontosabb kompenzációs lépéseket és megoldásokat szemléltetni:

- a mozgatott függvényekre hivatkozik másik függvény
- a mozgatott függvény törzsében van kompenzálandó függvényalkalmazás
- implicit fun expression hivatkozik valamelyik mozgatott függvényre
- import listák, export listák hivatkoznak a mozgatott függvényekre
- export vagy import listák szükségtelenné válnak, törölhetőek
- makróhivatkozás van a mozgatott függvény törzsében
- makrók egymásra hivatkoznak
- makrók rekordokra hivatkoznak.

3.4.1. Egyszerű exportált függvény áthelyezése

Feladat. f/1, g/1 áthelyezése (e1.erl → e2.erl)
Az egyik függvény exportált. Az áthelyezés megvalósítható.

e1.erl

```
-module(e1).
-export([f/1]).

f(Var) -> Var.
g(Var) -> f(Var).
```

e2.erl

```
-module(e2).
```

→

e1.erl

```
-module(e1).
```

e2.erl

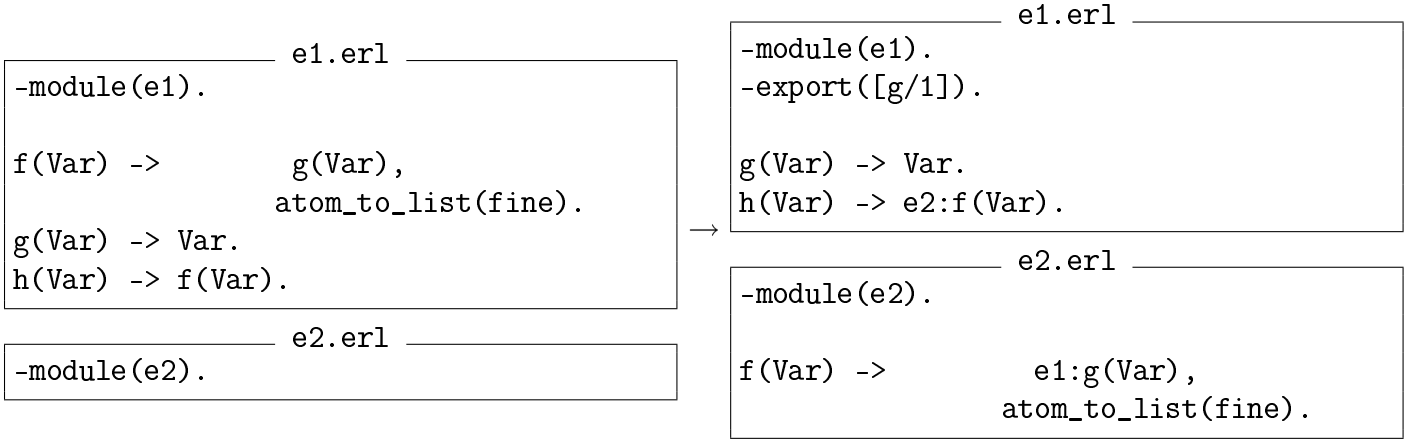
```
-module(e2).
-export([f/1]).

f(Var) -> Var.
g(Var) -> f(Var).
```

3.1. ábra. Egyszerű exportált függvény áthelyezése

Magyarázat. Az áthelyezés során fel kell ismerni, hogy az egyik függvény exportálva van a forrásmodulban, így a célmodulban is exportálni kell.

3.4.2. Több, egymásra hivatkozó függvény

Feladat. f/1 áthelyezése (e1.erl → e2.erl)
Az áthelyezés megvalósítható.



3.2. ábra. Több, egymásra hivatkozó függvény

Magyarázat. Az áthelyezett függvényre hivatkozunk a forrásmodulban, és maga is hivatkozik egy forrásmodulban maradó függvényre. Az áthelyezett függvény törzsében BIF (Built-In Function) is található, amelyet nem szabad megváltoztatni.

A forrásmodulban g/1-ben f-et modulminősíteni kell a célmodulra, az áthelyezett f/1-ben pedig g-t kell minősíteni a forrásmodulra. Észre kell vennünk, hogy ahhoz, hogy f használhassa g/1-et, exportálnunk kell g/1-et a forrásmodulban.

3.4.3. *Minősített, importált függvények*

Feladat. f/0, g/0 áthelyezése (e1.erl → e2.erl)
Többféle láthatósági és hivatkozási mód kompenzálásra szorul. Az áthelyezés megvalósítható.

e1.erl
```
-module(e1).
-export([f/0, g/0]).

f() -> ok.
g() -> f().
```

e2.erl
```
-module(e2).
-import(e1, [f/0]).

h() ->          f(),
                e1:g().
```

e3.erl
```
-module(e3).
-import(e1, [g/0]).

i() ->          e1:f(),
                g().
```

→

e1.erl
```
-module(e1).
```

e2.erl
```
-module(e2).
-export([f/0]).
-export([g/0]).

f() -> ok.
g() -> f().
h() ->          f(),
                  g().
```

e3.erl
```
-module(e3).
-import(e2, [g/0]).

i() ->          e2:f(),
                g().
```

3.3. ábra. Minősített, importált függvények

Magyarázat. Az áthelyezés után a mozgatott függvényekre vonatkozó hivatkozásokat kompenzálni kell. Ezek közé tartoznak a modulminősítéssel történő függvényalkalmazások és az import lista hivatkozások is.

- A célmodulban töröljük az importot, hisz az f/0 már lokális lesz.
- Szintén a célmodulban, h/0 törzsében töröljük a modulminősítést, hisz már nincs rá szükség hasonló okok miatt.
- Az f/0 és g/0 függvényeket exportáljuk a célmodulban, hisz a forrásmodulban is exportálva voltak.
- A harmadik modulban frissítjük a modulminősítést a forrásmodulról a célmodulra.
- Szintén a harmadik modulban az importált függvény modulnevét is frissíteni kell a forrásmodulról a célmodulra.

3.4.4. *Implicit fun expression, nem törölhető import*

Feladat. g/1 áthelyezése (e1.erl → e2.erl)
A g/1 függvényre implicit fun expression hivatkozás van, g/1 importált függvényt használ. Az áthelyezés megvalósítható.

e1.erl

```
-module(e1).
-export([f/0, g/1]).
-import(e3, [h/1]).

f() ->          F = fun g/1,
                 h(F).
g(Var) -> h(Var).
```

e2.erl

```
-module(e2).
```

e3.erl

```
-module(e3).
-export([h/1]).

h(Var) -> Var.
```

→

e1.erl

```
-module(e1).
-export([f/0]).
-import(e3, [h/1]).

f() ->          F = fun(V1) -> e2:g(V1),
                 h(F).
```

e2.erl

```
-module(e2).
-export([g/1]).
-import(e3, [h/1]).

g(Var) -> h(Var).
```

e3.erl

```
-module(e3).
-export([h/1]).

h(Var) -> Var.
```

3.4. ábra. Implicit fun expression, nem törölhető import

Magyarázat. A kompenzációt most nehezíti, hogy implicit fun expression-nel van hivatkozás a mozgatott függvényre, ami elé nem szúrhatunk be modulminősítést, előtte explicitté kell alakítani.

- A g/1 mozgatott függvény exportálását töröljük a forrásmodulban, a célmodulban pedig létrehozunk neki egy export lisát.
- Mivel a g/1 importált függvényt használ, a célmodulban importáljuk, a forrásmodulban viszont nem törölhetjük az importot, mivel f/0 is hivatkozik rá. Így az e3-ból származó import a forrás- és a célmodulban is megmarad.
- Az f/0 függvény a forrásmodulban implicit fun expression-nel hivatkozik g/1-re, tehát explicitté alakítjuk, majd a benne lévő függvényalkalmazást minősítjük a célmodulra.
- A forrásmodulban f/0 exportja megmarad, a harmadik modul változatlan marad.

3.4.5. Makróra, rekordra hivatkozó makrók

Feladat. f/0 áthelyezése (e1.erl → e2.erl)
A függvény törzsében olyan makróhivatkozás van, amely további makróra és rekordra hivatkozik. Az áthelyezés megvalósítható.

e1.erl

```
-module(e1).
-record(rekord, {}).
-record(rekord2, {}).
-define(MAKRO, #rekord{}).
-define(MAKRO2, ?MAKRO).
-define(MAKRO3, #rekord2{}).

f() -> ?MAKRO2, ?MAKRO3.
g() -> #rekord2{}.
```

e2.erl

```
-module(e2).
-record(MAKRO, {}).

h() -> #MAKRO{}.
```

→

e1.erl

```
-module(e1).
-record(rekord2, {}).

g() -> #rekord2{}.
```

e2.erl

```
-module(e2).
-record(rekord, {}).
-record(rekord2, {}).
-record(MAKRO, {}).
-define(MAKRO, #rekord{}).
-define(MAKRO2, ?MAKRO).
-define(MAKRO3, #rekord2{}).

h() -> #MAKRO{}.

f() -> ?MAKRO2, ?MAKRO3.
```

3.5. ábra. Makróra, rekordra hivatkozó makrók

Magyarázat. A kompenzáció magja most a makró- és a rekordhasználat vizsgálata.

- A mozgatott függvény két makrót, MAKRO2 és MAKRO3 makrókat használ, ám a helyzetet bonyolítja, hogy az egyik makró használ egy másik makrót, amely használ egy rekordot is.
- Észre kell vennünk a hivatkozásokat, és a célmodulban elérhetővé kell tennünk az entitásokat.
- Az összes használt entitást átvisszük, mivel szükségük van egymásra, ám a rekord2 nevű rekordot meg kell hagynunk a forrásfájlban is, mert g/0 hivatkozik rá, és az nem lett áthelyezve.
- A példa érdekessége, hogy rekordok és makrók között nem szabad ütközést detektálni, külön kell kezelni az entitásokat. Jelen esetben a MAKRO név makrók esetében nem foglalt, csak a rekordok között, így nem lesz ütközés és elvégezhető a transzformáció.

3.4.6. Makrók fejlécfájlokból

Feladat. f/0 áthelyezése (e1.erl → e2.erl)
A függvény makrókat használ fejlécfájlokból. Az áthelyezés megvalósítható.

e1.erl
```
-module(e1).
-include("h.hrl").
-include("h2.hrl").

f() -> ?MAKRO, ?MAKRO2.
g() -> ?MAKRO.
```

e2.erl
```
-module(e2).
```

h.hrl
```
-define(MAKRO, ok).
```

h2.hrl
```
-define(MAKRO2, ok).
```

→

e1.erl
```
-module(e1).
-include("h.hrl").

g() -> ?MAKRO.
```

e2.erl
```
-module(e2).
-include("h.hrl").
-include("h2.hrl").

f() -> ?MAKRO, ?MAKRO2.
```

3.6. ábra. Makrók fejlécfájlokból

Magyarázat. A kompenzáció során észre kell vennünk, hogy az entitások fejlécfájlból származnak, illetve azt, hogy az egyiket másik függvény, g/0 is használja, így annak include-ja nem törölhető, tehát h.hrl mindkét fájlban include-olva lesz.

3.4.7. Makróütközés az áthelyezés után

Feladat. f/0 áthelyezése (e1.erl → e2.erl)
Mivel a transzformáció után ütközés lépne fel, az áthelyezés nem valósítható meg.

e1.erl

```
-module(e1).
-include("h.hrl").
-define(MAKRO, #rekord{}).

f() -> ?MAKRO.
```

e2.erl

```
-module(e2).
-define(rekord, {mas}).
```

h.hrl

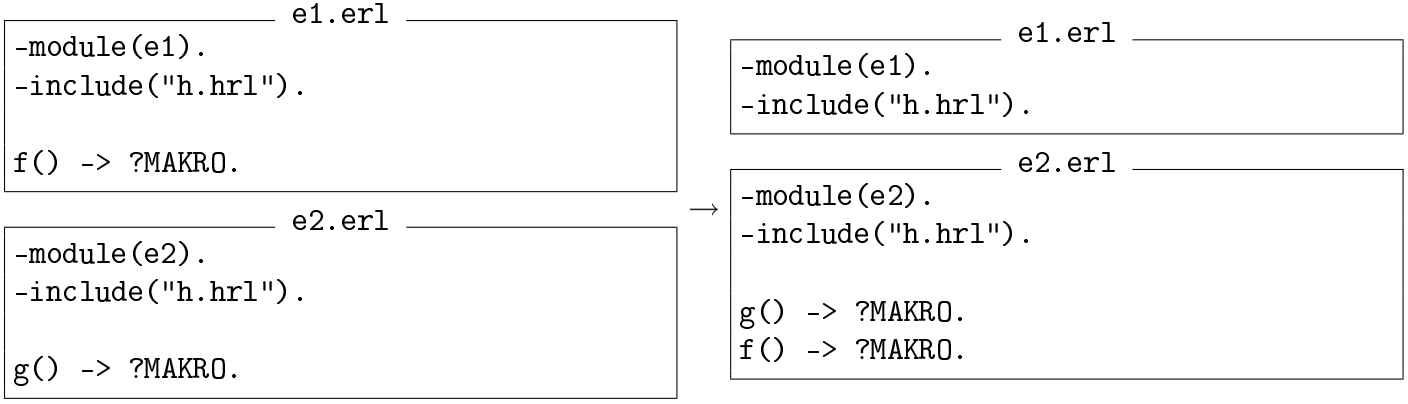
```
-define(rekord, {}).
```

3.7. ábra. Makróütközés az áthelyezés után

Magyarázat. Mivel a mozgatott függvény a fejlécfájlból használ makrót, a fejlécfájlt a célmodulban is include-olni kellene. Ám ha ezt megtesszük, névütközés lép fel.

3.4.8. Makrók azonos fejlécfájlból

Feladat. f/0 áthelyezése (e1.erl → e2.erl)
Ugyan látszólag ütközés lép fel, ám ez nem valódi probléma, hisz ugyanabból a fejlécfájlból származnak a makrók. Az áthelyezés megvalósítható.



3.8. ábra. Makrók azonos fejlécfájlból

Magyarázat. Az áthelyezésre kijelölt függvényben használt makró neve megegyezik egy célmodulban is létező makró nevével, s emiatt ütközésesnek tűnhetne a transzformáció, azonban mivel az azonos nevű makrók azonos fejlécfájlból származnak, nem lép fel semilyen névütközés.

4. BEFEJEZÉS

4.1. Áttekintés

A szakdolgozatomban megvalósítottam egy olyan refaktorálási transzformációt, amely képes modulok között függvényeket mozgatni, s az összes referenciakorrigálást és szükséges kompenzációkat elvégezni a felhasználó helyett. A megvalósított részfeladat egy nagyobb project részét képezi, melynek célja egy olyan Erlang refaktoráló eszköz készítése, amely nagy mennyiségű, nagy bonyolultságú ipari kódon is megfelelő pontossággal és sebességgel képes működni.

A részfeladatom implementálásra került, a forráskód angol nyelven lett dokumentálva. Ezen dolgozatban teljes függvényreferenciát írtam magyar nyelven, tesztesetekben bemutattam a transzformáció főbb képességeit, illetve a felhasználói dokumentációban leírtam a program telepítéséhez és használatához szükséges információkat, ábrával segítve a megértést.

A refaktorálási eljárások rendkívül jól használhatóak nagyobb programok írása során, s a függvények áthelyezése mellett még számos más transzformáció implementálható a jelenlegi keretbe, amelyek a jövőben valószínűleg meg is fognak valósulni.

4.2. Továbbfejlesztési lehetőségek

Az általam implementált program ipari méretű és bonyolultságú kódon lett tesztelve, s bár rendkívül jól teljesített, fény derült gyengeségekre is.

Úgy gondolom, hogy az ismert gyengeségek nagy része később teljes mértékben kiiktatható lesz, ha némileg módosításra kerül a szintaktikus gráf tranzakciókezelő modulja, illetve lehetővé válik a fejlécfájlok transzformáció közbeni megfelelő csatolása. Ha a transzformáció kevesebb tranzakcióból fog állni, hatékonyabb lesz és könnyebben vissza lehet görgetni egy esetleges hiba után.

ÁBRÁK JEGYZÉKE

IRODALOMJEGYZÉK

[1] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, 1999.

[2] Martin Fowler's refactoring site. http://www.refactoring.com/.

[3] R. Szabó-Nacsa, P. Diviánszky, and Z. Horváth. *Prototype environment for refactoring Clean programs.* In The Fourth Conference of PhD Students in Computer Science (CSCS 2004), Szeged, Hungary, July 1–4, 2004.

[4] P. Diviánszky, R. Szabó-Nacsa, and Z. Horváth. *Refactoring via database representation.* In L. Csőke, P. Olajos, P. Szigetváry, and T. Tómács, editors, The Sixth International Conference on Applied Informatics (ICAI 2004), Eger, Hungary, volume 1, page 129.

[5] J. Armstrong, R. Virding, M. Williams, and C. Wikstrom. *Concurrent Programming in Erlang.* Prentice Hall, 1996.

[6] J. Armstrong. *Making reliable distributed systems in the presence of software errors.* PhD thesis, The Royal Institute of Technology, Stockholm, Sweden, December 2003.

[7] J. Barklund and R. Virding. *Erlang Reference Manual*, 1999. Available from http://www.erlang.org/download/erl_spec47.ps.gz. 2007.06.01.

[8] Erlang homepage. http://www.erlang.org/.

[9] Mnesia Database Management System. http://www.erlang.org/documentation/doc-5.0.1/lib/mnesia-3.9.2/doc/index.html.

[10] Li, H., Thompson, S.J., Lövei, L., Horváth, Z., Kozsik, T., Víg, A., Nagy, T. *Refactoring Erlang Programs.* Accepted for 12th International Erlang/OTP User Conference,Stockholm, November 9-10, 2006.

[11] T. Nagy, A. Víg. *Storing Erlang source code in database*. Bachelor thesis, Faculty of Informatics, ELTE, Budapest, Hungary, February 2007.

[12] K. Beck. *Extreme Programming Explained*. Addison-Wesley, 2000. ISBN-0201-71091-9.

[13] H. Li. *Refactoring Haskell Programs*. PhD thesis, Computing Laboratory, University of Kent, Canterbury, United Kingdom, September 2006.

[14] H. Li, C. Reinke, and S. Thompson. *Tool support for refactoring functional programs*. Haskell Workshop: Proceedings of the ACM SIGPLAN workshop on Haskell, Uppsala, Sweden, 2003, pages 27–38.

[15] GNU Emacs homepage. http://www.gnu.org/software/emacs/.

[16] Aquamacs homepage. http://aquamacs.org/.

[17] VIM Editor homepage. http://www.vim.org/.